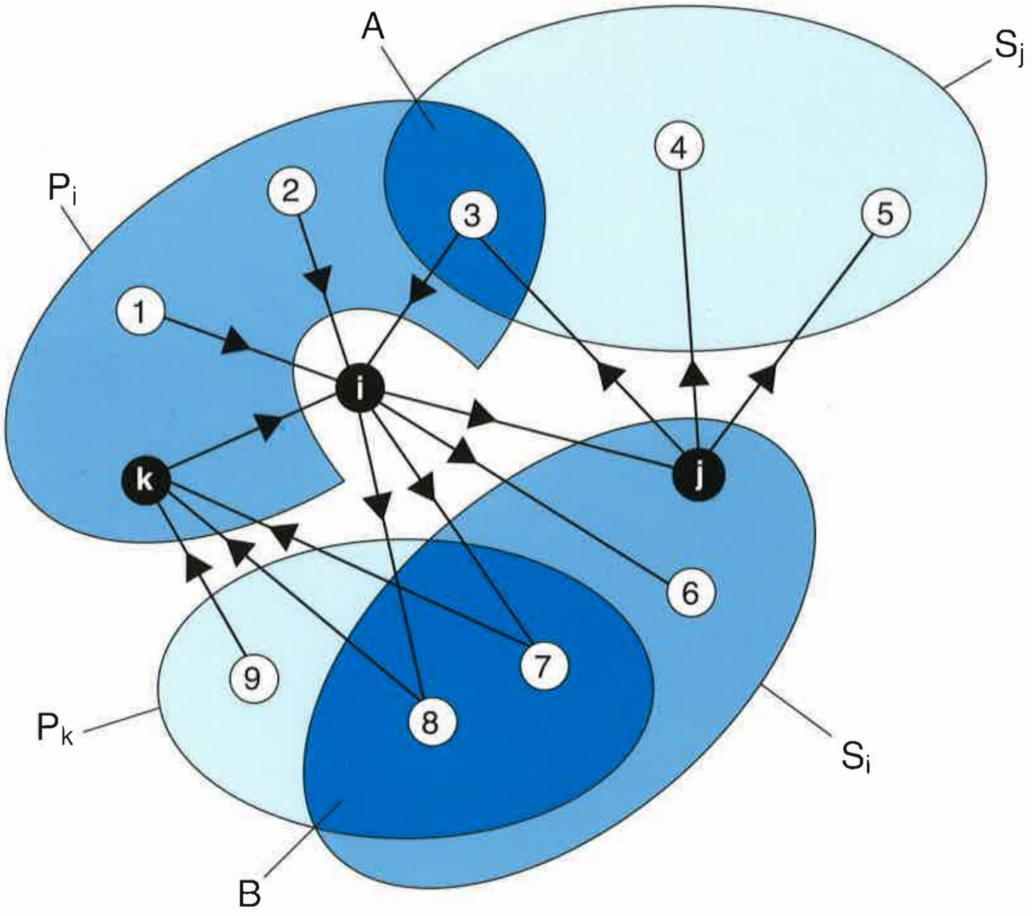
Mémoires de Géologie (Lausanne) No. 34, 1999

# Discrete Biochronological Scales and Unitary Associations : Description of the BioGraph Computer Program

Jean Savary and Jean Guex



IMPRIMERIE CHABLOZ S.A., LAUSANNE

ISSN 1015-3578

# Mémoires de Géologie (Lausanne)

*Section des Sciences de la Terre*
*Université de Lausanne*
BFSH-2, 1015 Lausanne, Suisse

34

# Mémoires de Géologie (Lausanne)

Mémoires de Géologie (Lausanne)
Section des Sciences de la Terre
Institut de Géologie et Paléontologie
Université de Lausanne
BFSH-2, CH-1015 Lausanne

SAVARY Jean and GUEX Jean
Titre : Discrete biochronological scales and Unitary Associations: Description of the BioGraph computer program
Mém. Géol. (Lausanne), n° 34, 1999, 282 pp., 21 text-figs.
ISSN: 1015-3578

Imprimeur : Chabloz S.A., Lausanne & Tolochenaz

Publié avec l'appui du Fonds national suisse de la recherche scientifique

# Discrete Biochronological Scales and Unitary Associations: Description of the BioGraph Computer Program

Jean Savary & Jean Guex

Mémoires de Géologie (Lausanne), No 34, 1999

## Abstract

The goal of the present paper is threefold. In the first part we summarize the algorithm used to construct Unitary Associations (UA's) originally described by Guex 1988, 1991 and Savary and Guex 1991.

This summary is followed by a short user's guide of the BioGraph program (Savary and Guex 1991). The listing of the program is provided in Section 5.

For several years, the users of the main program have been provided with unpublished tools designed to help the interpretation of the outputs of BioGraph: these tools will also be described in the present Memoir.

That material can be obtained for free from the authors.

## Résumé

Le présent mémoire a trois buts principaux. Dans la première partie nous résumons la démarche algorithmique suivie pour construire les Associations Unitaires, originellement décrite par Guex 1988, 1991 et Savary et Guex 1991.

Cette partie introductive est suivie par un bref Guide de l'Utilisateur du programme BioGraph (Savary et Guex 1991) et le listing complet de ce programme est publié dans le chapitre 5.

Depuis plusieurs années les utilisateurs de ce programme ont également reçu divers outils accessoires non publiés destinés à faciliter l'interprétation des outputs du programme BioGraph : ces outils sont aussi décrits formellement dans ce travail.

Ce matériel est disponible gratuitement auprès des auteurs.

## Authorship

Chapters 1 to 4 have been written by Jean Guex and Chapter 5 was written by Jean Savary.

## Table of Contents

# 1 Introduction

The Unitary Associations (UAs) method is a deterministic mathematical model designed to construct concurrent range zones. The basic idea of the method is to construct a discrete sequence of coexistence intervals of species. Each interval consists of a maximal set of intersecting ranges (= intervals of minimal duration or UAs). Each of these units is characterized by a set of species or species pairs allowing us to identify it in the stratigraphic sections.

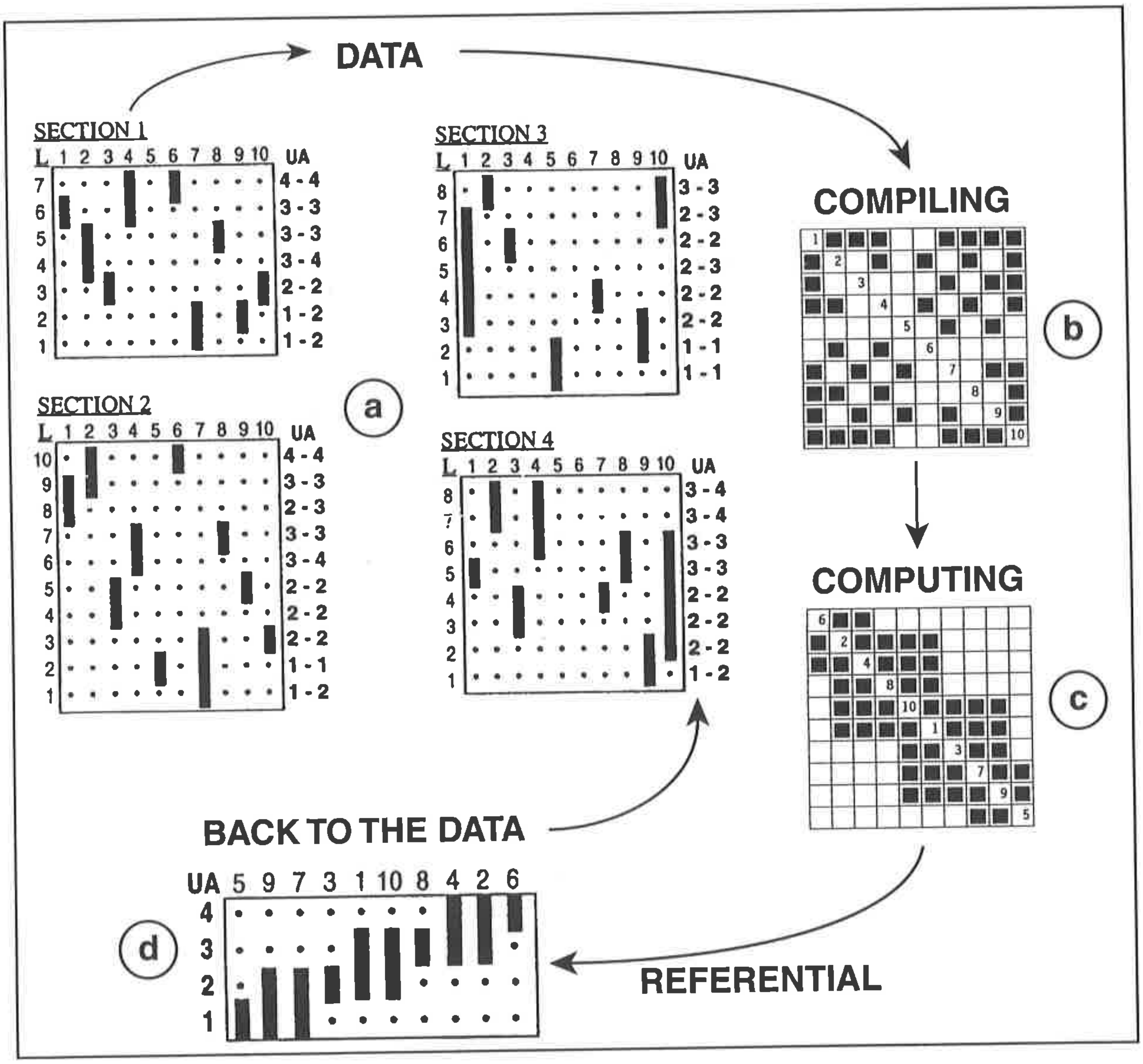


Fig.1 Simplified presentation of the UA method (from Guex 1991; see text for details)

The basic steps of the method are summarized in Fig.1. In this figure we represent 4 stratigraphic sections with the local distribution of 10 species (1 to 10) which may be present or absent in the sections (Fig.1a). The observed inter-species coexistences are compiled in a species-species matrix (Fig.1b). This matrix can be organized by a permutation of its rows and columns to allow the appearance of sets of mutually coexisting species (Fig.1c). From this reorganized matrix we can extract maximal sets of intersecting species' ranges (Fig.1d) and represent them in a table called a UA range chart. This chart is used to go back to the data and assign relative ages to the fossiliferous beds of the differents sections (Fig.1).

Biostratigraphic data are usually complicated by the fact that species' ranges are highly conflicting from place to place. As an example, consider two pairs of coexisting species (*ad*) and (*bc*). We say that their ranges are conflicting if species "*a*" occurs below species "*c*" and if species "*b*" occurs below species "*d*" in some localities. Such stratigraphic relationships mean that either the range of "*a*" virtually overlap that of "*c*" or that the range of "*b*" virtually overlap that of "*d*" (see Fig.2)

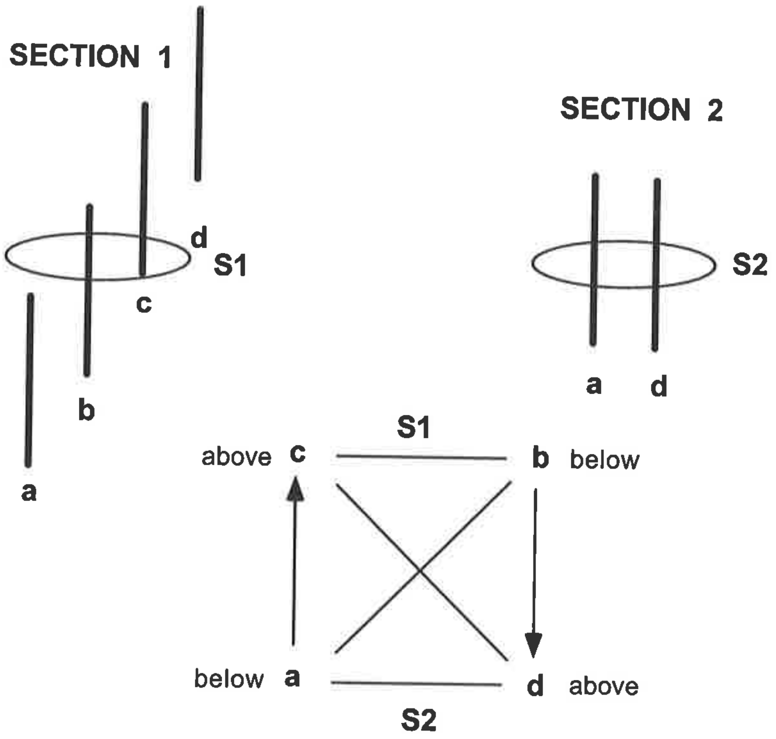


Fig.2 Sample S2 containing species *a* and *d* in section 2 is simultaneously "above" and "below" sample S1 containing *b* and *c* in section 1 because of the contradictive superpositions *a*→*c* and *b*→*d*. Note that this contradiction implies that either *a* coexisted chronologically with *c* or *b* coexisted chronologically with *d*: such coexistences are called virtual

The computer program BioGraph described below optimizes the constructions of such virtual coexistences and produces range charts where the conflicting stratigraphic relationships are expressed as virtual co-occurrences.

# 2 Summary of the Graph Theoretical Method

## 2.1 Definitions

The graph theoretical notations and definitions of symbols used herein are described in Guex 1991 and Savary and Guex 1991 and only a few basic terms will be repeated here.

### 2.1.1 The biostratigraphic graph $G^*$

In the following pages, $X = \{x_1,...,x_n\}$ denotes a set of fossil species. The stratigraphic relationships observed among the species of $X$ can be expressed using a matrix $A = (a_{ij})$ as follows:

$$a_{ij} = \begin{matrix} 1 \text{ if } x_i \text{ is in the same level or in a lower} \\ \text{level than } x_j \\ \\ 0 \text{ otherwise.} \end{matrix}$$

This matrix $A$ is the *adjacency matrix* of the biostratigraphic graph $G^*$. The edges of $G^*$ represent pairs of compatible (= chronologically coexisting) species. Edges that represent truly associated pairs of species (i.e observed coexistences) are called *real* edges; those that represent virtually associated pairs of species are called *virtual* edges.

### 2.1.2 Maximal cliques and UAs

Two vertices connected by an edge are *neighbors* in $G^*$. The set of neighbors of a vertex $x$ is denoted $\Gamma(x)$: this set corresponds to the *neighborhood* of the species $x$. A set of mutually neighbouring vertices is called a clique. A maximal clique (i.e. not contained in a larger clique) with real and virtual edges corresponds to a Unitary Association.

### 2.1.3 Intersection graph and interval graph

Let $S$ be a set, and $A = \{A_1,...,A_n\}$ be a family of subsets of $S$. The intersection graph of $(A,S)$ has one vertex $x_i$ for each subset $A_i$, and an edge joining $x_i$ to $x_j$ if and only if $A_i$ and $A_j$ intersect.

The intersection graph of a family of intervals on a line is called an interval graph. This means that if $J = \{J_1,...,J_n\}$ is a family of intervals, then the corresponding intersection graph has a set $X = \{x_1,...,x_n\}$ of vertices, with an edge connecting $x_i$ and $x_j$ if and only if $J_i$ and $J_j$ intersect. Interval graphs are characterized by several major theorems in graph theory (Fulkerson and Gross 1965, Lekkerkerker and Boland 1962, Gilmore and Hoffman 1964). Our problem is to transform the non oriented part of a biostratigraphic graph $G^*$ into an interval graph.

## 2.2 Conventions

If a fossil species has a discontinuous vertical distribution in a given stratigraphic section, it is considered as virtually present in all the beds that are flanked by its first local appearance and disappearance.

In a given stratigraphic section, we consider only the levels that record the maximal intersections of local existence intervals of the species. These levels are called *maximal horizons*. Maximal horizons that are strictly distinct from each other (with regard to the inclusion relationship) are called *residual maximal horizons*.

## 2.3 Method

### 2.3.1 Constructing maximal cliques of $G^*$

To construct the maximal cliques of the biostratigraphic graph $G^*$ using residual maximal horizons, the following procedure is used.

Let $k_m$ be the residual horizon with the greatest cardinality, and let $\Gamma(x_i)$ be the set of neighboring vertices of $x_i$ in G*.

- If $k_m - \Gamma(x_i) = \varnothing$ , we add $x_i$ to $k_m$ ($x_i$ is compatible with all the elements of $k_m$). The result is then denoted $k_m^* = \{k_m \cup x_i\}$.
- To $k_m^*$ we then add the next element $x_j$, for which $k_m^* - \Gamma(x_j) = \varnothing$ ($k_m^*$ thus becomes equal to $\{k_m \cup x_i \cup x_j\}$). We thus compare the enlarged $k_m^*$ successively to all the vertices of $G^*$.
- At the end of this operation, we eliminate all the residual horizons that are contained in $k_m^*$, then we follow the same procedure for the next $k_m$, etc... The resulting $k_m^*$ are now distinct cliques of $G^*$.

### 2.3.2 Stratigraphic relationships among the maximal cliques

The stratigraphic relationship between two maximal cliques, $k_i$ and $k_j$, can be deduced from the stratigraphic relationships observed among the species belonging respectively to $k_i$ and to $k_J$ Three kinds of relationships exist.

- The clique $k_i$ is above (or below) $k_j$ if there exists at least one species of $k_i$ that is above (or below) a species of $k_j$ (univoque superposition).
- The relationship between $k_i$ and $k_j$ is undetermined if the species of $k_i$ are not compatible with those of $k_j$ and if their superpositional relationships are undetermined.
- The relationship between $k_i$ and $k_j$ is conflicting if some species of $k_i$ are above some species of $k_j$, while other species of $k_i$ are below some species of $k_J$

### 2.3.3 Searching for and eliminating contradictions

To transform a biostratigraphic graph into an interval graph by ordering its maximal cliques according to raw biostratigraphic observations, it is first essential to eliminate any contradictory superpositions of species. This can be done in two ways. The first consists of detecting all the forbidden configurations of G* and replacing certain superpositions by deduced virtual coexistences. The second consists of ignoring certain contradictory and poorly reproducible superpositions between two maximal cliques.

The BioGraph program uses the latter approach. It is based on the following principle: faced with two conflicting superpositions, we are forced to admit (in the absence of non-biochronologic arguments) that a superposition that is reproducible (i.e. observed over a vast geographic area) has more chronologic "value" than one that is not. In other words, if two distinct cliques of *G** contain species that situate them both above and below each other, we will in each case consider as "true" the superposition that is more reproducible and as "false" (i.e.: generated by insufficient documentation, by reworking or bad taxonomy) the superposition that is less reproducible.

To do this, we proceed as follows. For each pair $k_i$,$k_j$ of maximal cliques of *G** showing a contradictory stratigraphic relationship, we define two sets of arcs, *A* and *B*, where *A* is the set of arcs that links the elements of $k_i$ to those of $k_j$ (in the direction $k_i \rightarrow k_j$) and where *B* is the set of arcs (of the opposite orientation) that links the elements of $k_j$ to those of $k_i$. To each set *A* and *B* we attribute a value *V(A)* (resp. *V(B)*) equal to the sum of the individual reproducibilities of the arcs belonging to *A* (resp.*B*) added to the number of arcs of *A* (resp.*B*).

If $V(A) > V(B)$ we say that the clique $k_j$ is "located above" $k_i$ (keeping in mind that this is an abuse of language and that the superposition is conflicting).

If $V(A) = V(B)$ we say that the stratigraphic relationship between $k_i$ and $k_j$ is undetermined.

This precedure is equivalent to making a global (and no longer individual) search for the forbidden configurations of *G** (Fig. 3).

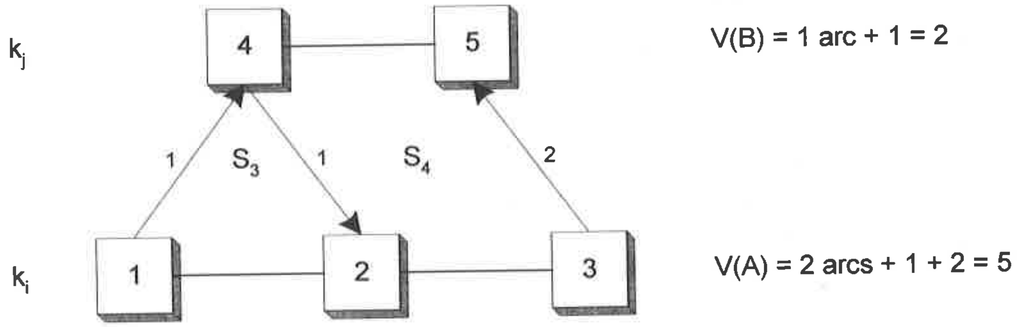


Fig.3 Example of conflicting relationship between two maximal cliques

### 2.3.4 Matrix *M* and graph $G_k$

Once the stratigraphic relationships of the cliques have been established, they are entered in a "maximal clique-maximal clique" matrix *M*.

The relationships represented by *M* are denoted as follows:

- $k_i << k_j$ = $k_i$ above $k_j$ without contradiction
- $k_i >> k_j$ = $k_i$ below $k_j$ without contradiction
- $k_i \leftarrow k_j$ = $k_i$ above $k_j$ with contradiction
- $k_i \rightarrow k_j$ = $k_i$ below $k_j$ with contradiction
- $k_i$ ?? $k_j$ = undetermined relationship

Using this matrix, we next contruct the oriented graph $G_k$ that is associated to it.

### 2.3.5 Strongly connected components of $G_k$

2.3.5.1 Detection and destruction

Poorly documented biostratigraphic data (polluted by undetected reworking, by false taxonomic identifications or by highly discontinuous record of the species) often are responsible for the presence of strongly connected components (cycles) in $G_k$. These components must of course be eliminated.

The algorithm used to detect them is similar to the one described by Carre (1978, p.47): we will not discuss the latter further here.

In our program, the connected components are destroyed according to the following rule:

Consider, in a given component of $G_k$, the set of pairs $(k_i,k_j)$ of cliques whose deduced stratigraphic relationship is conflicting (i.e.: $\rightarrow$ or $\leftarrow$ in the matrix $M$). We denote $A$ the set of arcs that link $k_i$ to $k_j$ and $B$ the set of arcs of opposite orientation. To each set $A$ and $B$ we attribute a value $V(A)$ (resp. $V(B)$) equal to the sum of the individual reproducibilities of the arcs belonging to $A$ (resp.$B$) added to the number of arcs of $A$ (resp.$B$) (see above). To each arc $k_i \rightarrow k_j$ we now attribute a value

$$C_{ij} = \min(V(A);V(B)) \,/\, \max(V(A);V(B))$$

The arc $k_i \rightarrow k_j$ for which the value $C_{ij}$ is the greatest is destroyed (the relationship $k_i,k_j$ is considered to be undetermined). If this undetermination is not sufficient to destroy the connected component, we proceed in the same way for the next couple $(k_i,k_j)$, etc....

2.3.5.2 Technical remark

It is essential to stress the fact that in the case of particularly poor data, the destruction of the connected components takes place in a quasi-arbitrary fashion, and depends on the order in which $G_k$ is examined. Such situations can be detected by permuting the order of the stratigraphic sections in the input (which has the effect of modifying the indexing of the maximal horizons and consequently that of the vertices of $G_k$). If the solutions thus obtained differ from one output to the next, it is then necessary to generate virtual coexistences artificially by grouping the unitary associations whose characteristic elements were permuted: the different positions that were assigned to them in the different solutions tested result from the fact that their real chronologic relationships are impossible to determine.

The treatment of strongly connected components is also discussed with more details in Section 4.2.

### 2.3.6 Maximal paths of $G_k$

When the graph obtained at the end of these operations consists of several maximal paths, we reduce it to a unique path using a technique called "best fit". The BioGraph program adds an additional constraint to this technique: a vertex that does not belong to the longest maximal path of $G_k$ ($L$) can be united only with a successor of its immediate predecessor in *L* (resp. predecessor of its immediate successor in *L*) (when such elements exist).

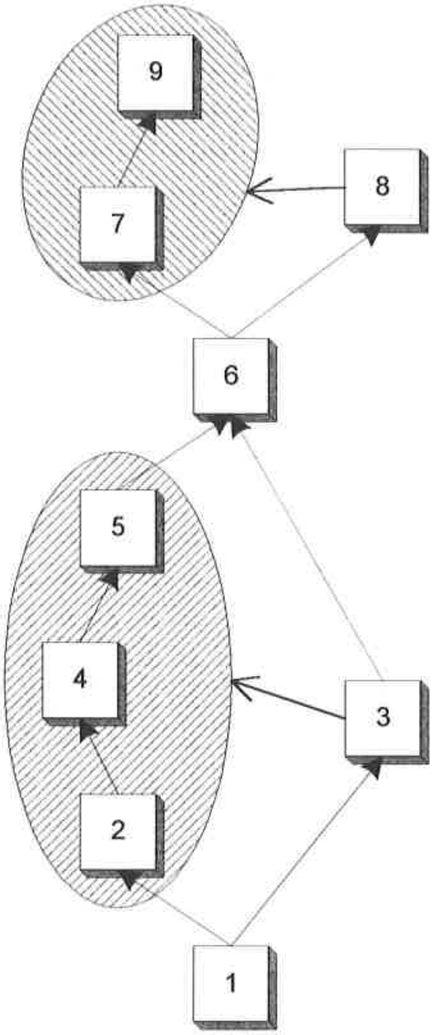


Fig. 4 Reductions of $G_k$ to a unique path. Bold arrows indicate where the unions will occur.

### 2.3.7 Matrix TGH and unitary associations

The last step of the method consists of transcribing the species content of the cliques of the reduced graph L in the maximal cliques - species incidence matrix associated to it.

This matrix, denoted TGH in the output of BioGraph, records the discontinuities in the stratigraphic distribution of the species. To give it the consecutive 1's property, it is sufficient to replace with 1's the 0's that are flanked by 1's in the columns.

In the end only the rows of the resulting matrix that correspond to maximal cliques are conserved (the others are eliminated): each row of this final compacted matrix corresponds to a unitary association and the graph associated with it is an interval graph.

### 2.3.8 Residual virtual edges

The procedure described above does not take into account the virtual edges in the initial step of the calculations. This is because (by definition) the pairs of species that correspond to these edges are not observed in any maximal horizon. In general these edges are generated artificially during the transformation of $G^*$ into an interval graph. When this is not the case, we use a detecting procedure that enables us to recover them during the last step of the calculations.

A residual virtual edge is detected if there is an original arc $x_j \rightarrow x_i$ when $x_j$ belongs to an UA ($k_{k+n}$) located above the UA $k_k$ containing $x_i$ (Fig. 5a).

To recover the virtual edge we must optimize the preservation of the arcs connecting $x_i$ (resp. $x_j$) and its successors (resp. predecessors) present within the interval $k_k$... $k_{k+n}$.

Let $A = (k_{k+1} \cup \ ... \cup \ k_{k+n}) - k_k$ and $B = (k_k \cup \ ... \cup \ k_{k+n+1}) - k_{k+n}$

The sum of the reproducibility of the arcs $x_i \rightarrow A$ and $B \rightarrow x_j$ provides 2 values, $V(A)$ and $V(B)$, that can be used to make a choice between three possible solutions for recovering the virtual edge ($x_i$, $x_j$):

- If $V(A) > V(B)$, then $x_j$ is added to the interval $k_{k+n-1}$ ...$k_n$ (Fig. 5b)
- If $V(A) < V(B)$, then $x_i$ is added to the interval $k_{k+1}$ ... $k_{k+n}$ (Fig. 5c)
- If $V(A) = V(B)$, then $x_i$ is added to $k_{k+1}$ ...$k_{k+n/2}$ and $x_j$ is added to $k_{k+n-1}$ ... $k_{k+n/2}$ (Fig. 5d).

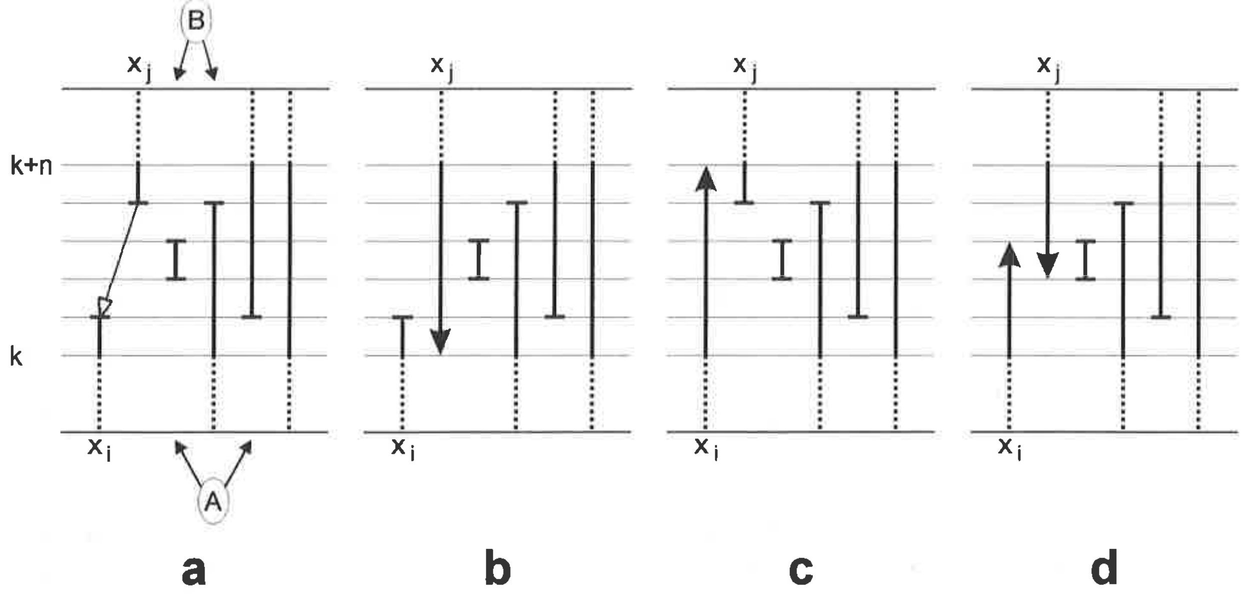


Fig. 5 Inserting residual virtual edges (see text)

### 2.3.9 Residual arcs

Certain conflicting stratigraphic superpositions destroyed during the transformation of $G^*$ can be restored during the last step of the procedure. This is done as follows (Fig. 6).

In each unitary association $k_i$ of the *first* "range chart" calculated by the program (i.e. constructed immediately after compacting the matrix associated with $L$), we define two sets of species, $A$ and $D$, where $A$ is the set of species that appear in $k_i$ and where $D$ is the set of species that disappear in $k_i$ (Fig. 6a).

If there exists an arc $x \rightarrow y$ between an element $x$ of $D$ and an element $y$ of $A$, we turn $k_i$ into two associations, $k_i$ and $k_j$ (initially $k_i = k_j$). Next we remove species $x$ from $k_j$ and species $y$ from $k_i$ (Fig. 6b).

If there exist several distinct arcs of the type $x \rightarrow y$, we repeat the operation for each arc, verifying at each step that the removal of $x$ from $k_j$ (resp.$y$ from $k_i$) does not lead to the destruction of a real edge (e.g. $(x, z)$ or $(y, z)$). If it does, the species is not removed (Fig. 6c).

When this operation has been completed for all the unitary associations, the test of maximality is applied once more and sets that are not maximal are eliminated.

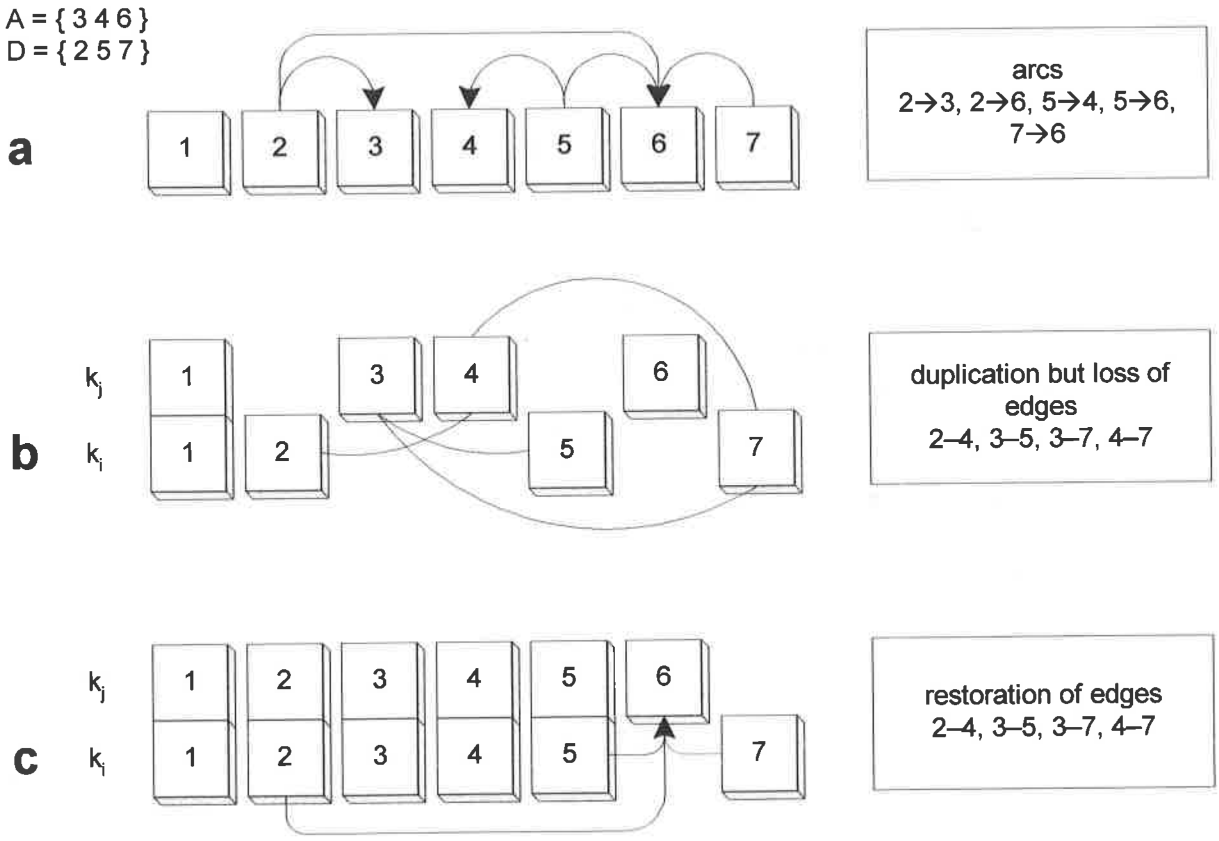


Fig. 6 Recovering residual arcs (see text)

### 2.3.10 Identification of unitary associations and correlations

2.3.10.1. Method

The last operation performed by our program consists of constructing correlation tables based on the unitary associations. This is done as follows.

We denote $X_{ij}$ the specific content of the $i^{th}$ level of the $j^{th}$ section.

We denote $K = \{k_m, m=1 \text{ to } p\}$ the set of unitary associations obtained.

For each level of each section we calculate

$X_{ij} - k_m = I$

If $|I| = 0$, the level is assigned to the corresponding unitary association (or unitary associations).

2.3.10.2 Using previously published UA-scales

Each new stratigraphic section introduced into a database modifies the previously calculated biochronological scale and its calibration to chronostratigraphy. For this reason the user should compute the position of each new sample separately when applying an already published complex UA-zonation to his new data. In other words, in order to avoid a complete reinterpretation and recalibration of the new scale, one should not calculate the full new section against the old zonation.

# 3 Short Userguide of BioGraph

## 3.1 Install

When installing the progam, the user is merely asked to declare the name of the word processor. When working in Windows, you can type *c:\windows\command\edit.com* or chose the path leading to another word processor.

All the other operations are done automatically.

## 3.2 Input of the data

### 3.2.1 Dictionary

When one studies a biostratigraphic data set to correlate one's sections, it is reasonable to start by coding the fossil species under the form of a dictionary. Such a dictionary will be named "Filename.DCT" (i.e. the unformatted file or ASCII file is given an extension DCT). It contains the code numbers of your taxa (one to five alphanumerical characters) followed by their full name.

The standard alveolinid problem (Drobne, 1977) is taken here as an example:

Filename: ALVEOLINID.DCT

| | |
|---|---|
| 01mou | Alveolina moussoulensis |
| 02ara | Alveolina aramea |
| 03sol | Alveolina solida |
| 04glo | Alveolina globosa |
| 05ave | Alveolina avellana |
| 06pis | Alveolina pisiformis |
| 07pas | Alveolina pasticillata |
| 08leu | Alveolina leupoldi |
| 09mon | Alveolina montanarii |
| 10arg | Alveolina aragonensis |
| 11ded | Alveolina dedolia |
| 12spy | Alveolina subpyreneica |
| 13lax | Alveolina laxa |
| 14gui | Alveolina guidonis |
| 15dec | Alveolina decipiens |

This file is ready to be read by one of the tools (BG_T11 described below) converting the species code numbers into full names in the computed UA-range chart (output *.TGJ of BioGraph).

### 3.2.2 Database: format and conventions

The database file is also an unformatted (ASCII) file denoted "Filename.DAT" (i.e. with extension DAT).

BioGraph accepts two kinds of data inputs: the list of the species with their total range in each section (first and last occurrence = DATUM format) or the specific content of each sample in the sections (SAMPLES format). The database format is first announced in the data file. Following this, the user declares the title of the file (TITLE " ..."). The section's name is then followed by the number of levels. If a given section *X* has 12 levels, you declare SECTION X, BOTTOM 1 TOP 12. Then you type the list of the species present in the section (denoted by their respective code numbers) and indicate their ranges (first and last occurrence if you use the DATUM option). If you use the SAMPLES option, you type the taxonomic content of each level and the level number is preceded by the sign <.

Comments can be inserted anywhere in the database. They just need to be framed by { } to be ignored during the computation of the data. Particular data that are supposed to be ignored during the computation can also be framed by the { } parentheses.

We finally note that the user should leave a void space between the last data entry and the end of file's mark.

3.2.2.1 Example of the alveolinid database with the DATUM option

```
DATUM
TITLE "Short example Alveolinid"
SECTION Fatji_hrib  BOTTOM 1 - TOP 4
02ara 1-1  03sol 4-4  04glo 3-3  06pis 2-2  07pas 3-3
08leu 4-4  12spy 3-3
```

```
SECTION Dane_Divaca BOTTOM 1 - TOP 4
01mou 3-3  03sol 1-1  05ave 1-1  06pis 2-3  07pas 1-3
09mon 3-4  11ded 3-3  12spy 3-3  13lax 2-2  14gui 4-4
SECTION Veliko BOTTOM 1 - TOP 7
02ara 1-1  03sol 2-2  06pis 2-3  07pas 2-3  09mon 6-7
10arg 7-7  11ded 5-5  12spy 4-4  13lax 3-3  14gui 6-7
15dec 4-6
SECTION Ritomece BOTTOM 1 - TOP 4
01mou 1-2  04glo 3-3  07pas 1-1  08leu 2-2  09mon 1-4
10arg 3-4  11ded 1-2  12spy 2-2  14gui 3-4  15dec 4-4
SECTION Podgorje BOTTOM 1 - TOP 2
04glo 2-2  05ave 1-1
SECTION Podgrad_Hrusica BOTTOM 1 - TOP 4
01mou 2-2  04glo 3-3  07pas 1-2  09mon 3-3  10arg 4-4
13lax 1-1  14gui 4-4
SECTION Kozina_Socerb BOTTOM 1 - TOP 4
02ara 1-1  03sol 3-3  05ave 2-2  07pas 3-4  13lax 4-4
15dec 4-4
SECTION Golez BOTTOM 1 - TOP 6
02ara 1-1  05ave 1-1  06pis 3-3  07pas 2-2  08leu 5-5
10arg 6-6  13lax 4-4  14gui 5-6
SECTION Zbernica BOTTOM 1 - TOP 2
06pis 2-2  10arg 1-2  12spy 2-2  15dec 1-1
```

#### 3.2.2.2 Example of the alveolinid database with the SAMPLES option

```
SAMPLES
TITLE "Short example Alveolinid"
SECTION FATJI_HRIB bottom 1 - top 4
< 4 03SOL 08LEU
< 3 04GLO 07PAS 12SPY
< 2 06PIS
< 1 02ARA
SECTION DANE_DIVACA bottom 1 - top 4
< 4 09MON 14GUI
< 3 01MOU 06PIS 07PAS 09MON 11DED 12SPY
< 2 06PIS 07PAS 13LAX
< 1 03SOL 05AVE 07PAS
SECTION VELIKO bottom 1 - top 7
< 7 09MON 10ARG 14GUI
< 6 09MON 14GUI 15DEC
< 5 11DED 15DEC
< 4 12SPY 15DEC
< 3 06PIS 07PAS 13LAX
< 2 03SOL 06PIS 07PAS
< 1 02ARA
SECTION RITOMECE bottom 1 - top 4
< 4 09MON 10ARG 14GUI 15DEC
```

< 3 04GLO 09MON 10ARG 14GUI
< 2 01MOU 08LEU 09MON 11DED 12SPY
< 1 01MOU 07PAS 09MON 11DED etc...

3.2.2.3 Conversion of format

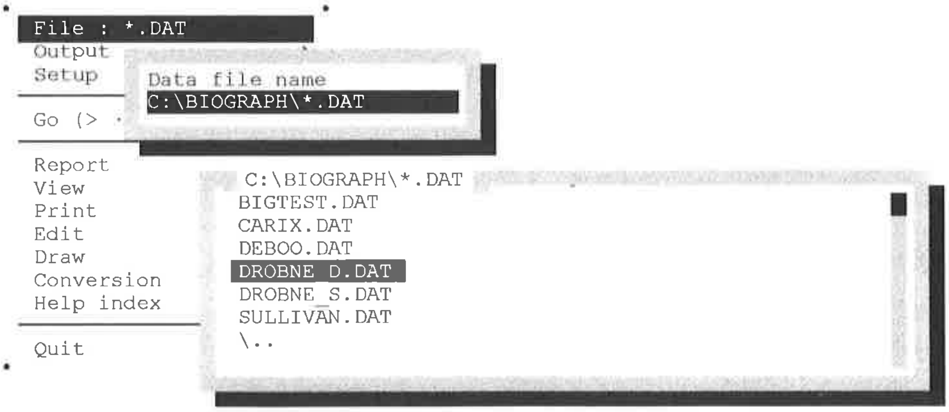
Files in the Samples format can be converted automatically into the Datum format and vice and versa (see below).

### 3.2.3 Size limit of data and memory overflows

The upper limit of the database size accepted by the program is a balance between 500 taxa and 1000 sections, depending on the complexity of the probem. Memory overflows occur whenever the size of the database is too large. In such cases the program usually stops automatically. However, when calculating exceptionally large databases by means of BioGraph 2.01, we noticed recently that some memory overflows passed the security checks of the program and produced inaccurate strongly connected components in the $G_k$ graph due to an inversion of the sign of the reproducibility value of some arcs. These rare cases are easily detected by running the tool BG_CON described in section 3.4.18 which displays the detail of the weights of all arcs in $G_k$ : abnormal negative values of the reproducibility are immediately apparent. If this happens the user should reduce the size of his database and reprocess the smaller database.

## 3.3 The main menu

The main menu contains three first instructions: FILE, OUTPUT and SETUP (Figs. 7 - 9).



Data file name
C:\BIOGRAPH\*.DAT

C:\BIOGRAPH\*.DAT
BIGTEST.DAT
CARIX.DAT
DEBOO.DAT
DROBNE_D.DAT
DROBNE_S.DAT
SULLIVAN.DAT
\..

Fig. 7 Choice of the data file

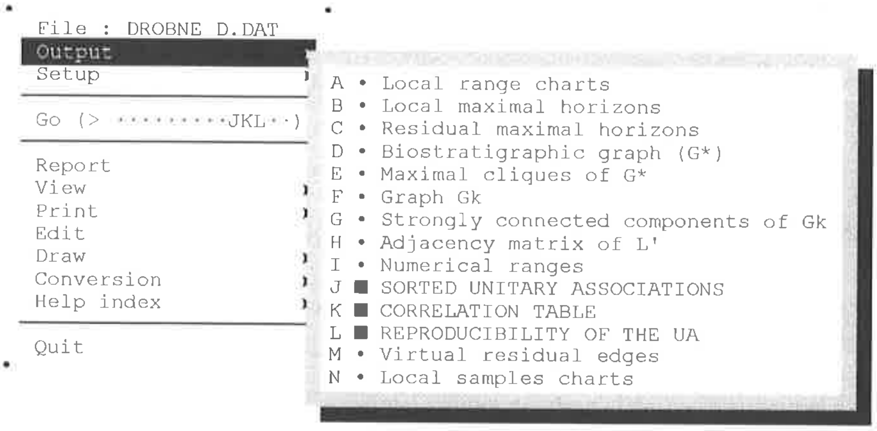
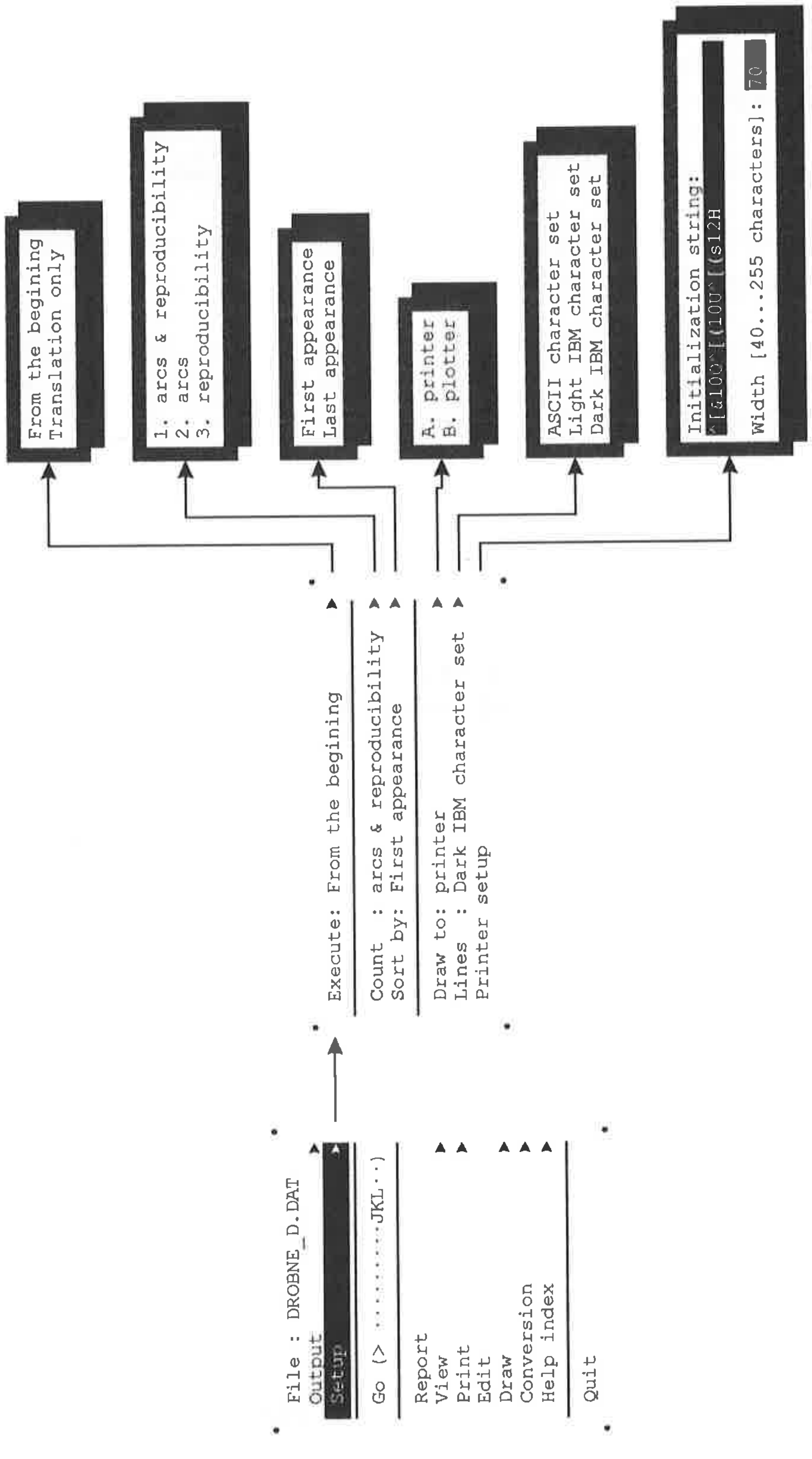

File : DROBNE D.DAT
Output
Setup
Go (> ·········JKL··)
Report
View
Print
Edit
Draw
Conversion
Help index
Quit
A • Local range charts
B • Local maximal horizons
C • Residual maximal horizons
D • Biostratigraphic graph (G*)
E • Maximal cliques of G*
F • Graph Gk
G • Strongly connected components of Gk
H • Adjacency matrix of L'
I • Numerical ranges
J ■ SORTED UNITARY ASSOCIATIONS
K ■ CORRELATION TABLE
L ■ REPRODUCIBILITY OF THE UA
M • Virtual residual edges
N • Local samples charts

Fig. 8 Choice of the outputs

- FILE opens a window giving the list of the data files (Filename.DAT) recorded in BioGraph's directory (Fig.7).
- OUTPUT is a list of text-files produced by BioGraph. These files are denoted TGA to TGN. In Fig. 8 we have illustrated the choice of TGJ, TGK and TGL which correspond respectively to the "Unitary Association Chart", the correlation table and the reproducibility table. That choice appears automatically in the list of the GO instruction (= start of the data processing).
- SETUP (Fig. 9) offers several choices. EXECUTE allows you to compute the data from the beginning or to limit the computation to the translation of the binary files that were already computed by the program (see "Output"). COUNT offers three choices: you can use the number of arcs and their respective reproducibility as inter-UA superpositional constraint, or you can limit that constraint by using only the number of arcs or the reproducibility of these arcs. The first constraint is the strongest and is the most frequently used. SORT allows you to produce the UA-range chart (TGJ) with the species organized by first appearance or by disappearance. DRAW allows you to use a printer or a plotter. LINES provides you with a choice of different characters. PRINTER offers you the possibility to change the code of the printer and to increase the number of characters per lines in the outputs: when you have very large range charts, it is recommended to use the 255 characters option.



```
File : DROBNE_D.DAT
Output
Setup
Go (> ·········JKL··)
Report
View
Print
Edit
Draw
Conversion
Help index
Quit
```

```
Execute: From the begining
Count  : arcs & reproducibility
Sort by: First appearance
Draw to: printer
Lines  : Dark IBM character set
Printer setup
```

```
From the begining
Translation only
```

```
1. arcs & reproducibility
2. arcs
3. reproducibility
```

```
First appearance
Last appearance
```

```
A. printer
B. plotter
```

```
ASCII character set
Light IBM character set
Dark IBM character set
```

```
Initialization string:
^[&l0O^[(10U^[(s12H

Width [40...255 characters]: 70
```

Fig. 9 Setup of the program

The following options are also available directly from the main menu.

- REPORT contains the list of the sections with their respective numerical code used in some outputs of the program and the list of the species with their respective numerical code.

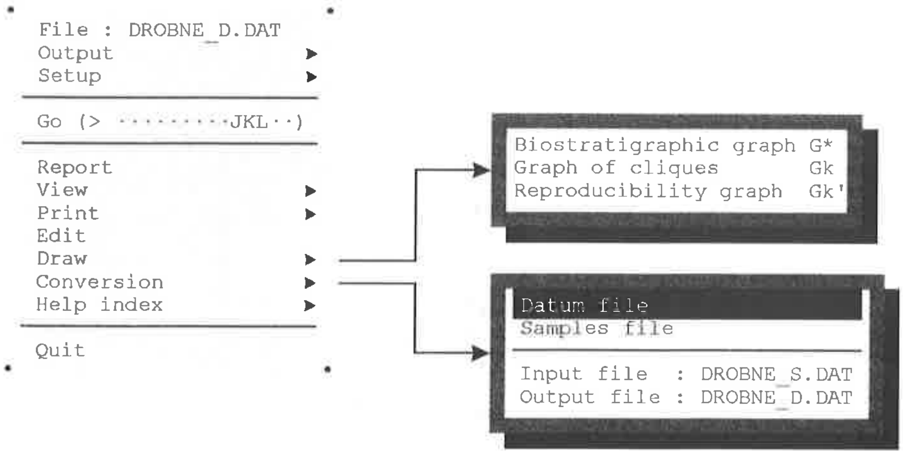
- VIEW allows you to visualize directly any output produced by the program.

- PRINT allows you to print the outputs. It is however strongly recommended to edit the output you want to print and to format it with your word processor.

- DRAW allows you to edit the graphs *G**, $G_k$ and $G_k'$. You can play with those graphs by moving their vertices and arranging them in a didactic way prior to print them by means of a dot matrix printer (Fig. 10).

- CONVERSION allows you to convert a datafile written in one format into the other format (i.e. DATUM into SAMPLES and vice versa: see Fig. 10).

This last option is very important when the user wants to combine 2 or more range charts by superposing them. First he should convert the numercial range chart (*.TGI) which is produced in DATUM format into a SAMPLES numerical range chart. Then he can merge the 2 (or more) files and renumber his UAs in the resulting data file.



```
File : DROBNE_D.DAT
```

```
Input file  : DROBNE_S.DAT
Output file : DROBNE_D.DAT
```

Fig. 10 Options Draw and Conversion

## 3.4 BG-TOOLS

Sixteen different tools are provided with the main program but they are not directly integrated with it. To use them, you must work in the directory C:\BioGraph. The syntax used in the tools is described below. The standard "Alveolinid problem" (see Guex 1991, p.52) is used as database to illustrate the applications of the tools.

### 3.4.1 BG_T01: Species' geofrequency

Syntax: C:\BIOGRAPH> BG_T01 Filename.
Output: Filename.T01

This routine provides the species list with the number of localities where they occur (geofrequency). The tool is designed to locate the species present in only one locality and that are not useful in correlating sections (even if they can be interesting as paleoecological markers).

If the graph contains a single local sequence of such restricted species, it can theoretically generate a non significant longest path in Gk and have a bad effect in the treatment of the data.

Example: Alveolinid

| | | | |
|---|---|---|---|
| 01MOU | 4 | 09MON | 5 |
| 02ARA | 4 | 10ARG | 7 |
| 03SOL | 4 | 11DED | 4 |
| 04GLO | 4 | 12SPY | 6 |
| 05AVE | 4 | 13LAX | 5 |
| 06PIS | 6 | 14GUI | 6 |
| 07PAS | 7 | 15DEC | 4 |
| 08LEU | 4 | | |

### 3.4.2 BG_T02: Frequency of observed co-occurrences

Syntax: C:\BIOGRAPH> BG_T02 Filename.
Output: Filename.T02

This tool enumerates the edges of G* (pairs of species) and their frequency of occurrence.

Example: Alveolinid

| | | | | | |
|---|---|---|---|---|---|
| 01MOU | 06PIS | 2 | 03SOL | 07PAS | 3 |
| 01MOU | 07PAS | 3 | 03SOL | 08LEU | 1 |
| 01MOU | 08LEU | 1 | 04GLO | 07PAS | 1 |
| 01MOU | 09MON | 2 | 04GLO | 09MON | 2 |
| 01MOU | 11DED | 3 | 04GLO | 10ARG | 1 |
| 01MOU | 12SPY | 3 | 04GLO | 12SPY | 1 |

```
02ARA  05AVE      1        04GLO  14GUI     1
03SOL  05AVE      1        05AVE  07PAS     1
03SOL  06PIS      1        06PIS  07PAS     2
                                                        etc
```

### 3.4.3 BG_T03 : Virtual edges

Syntax: C:\BIOGRAPH> BG_T03 Filename.
Output: Filename.T03

Routine providing the list of initial virtual edges.

Example: Alveolinid: 3 "virtual edges"

```
"03SOL" "12SPY"
"04GLO" "08LEU"
"08LEU" "10ARG"
```

### 3.4.4 BG_T04 : Observed arcs

Syntax: C:\BIOGRAPH> BG_T04 Filename.
Output: Filename.T04

Routine providing the list of the initial inter-species superpositions (arcs of G*) and their respective frequency.

Example: Alveolinid

```
"01MOU" "←" "03SOL"  1        "03SOL" "→" "15DEC"     2
"01MOU" "→" "04GLO"  2        "04GLO" "←" "05AVE"     1
"01MOU" "←" "05AVE"  1        "04GLO" "←" "06PIS"     1
"01MOU" "→" "10ARG"  3        "04GLO" "←" "11DED"     1
"01MOU" "←" "13LAX"  2        "04GLO" "←" "13LAX"     1   etc...
```

### 3.4.5 BG_T05 : Statistics on UAs

Syntax: C:\BIOGRAPH> BG_T05 Filename.
Output: Filename.T05

Routine providing a variety of statistics on the UAs. The output consists of 12 columns denoted A to L:

A. UA #
B. number of species present in the UA (taxonomic diversity)
C. number of species present uniquely in the considered UA
D. LADs without column C
E. FADs without column C
F. species in common with the previous and the next UA
G. LADs + column C

H. FADs + column C
I. species common with the previous UA
J species common with the next UA
K. sum of LADs from the first UA (cumulated LADs)
L. sum of FADs from the first UA (cumulated FADs)

Example 1: Alveolinid database (data taken from the range chart given in Fig. 13A)

| A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 1 | 5 | 0 | 0 | 6 | 1 | 5 | 0 | 15 | 15 |
| 6 | 7 | 0 | 2 | 1 | 4 | 2 | 1 | 6 | 5 | 9 | 14 |
| 5 | 9 | 0 | 3 | 1 | 5 | 3 | 1 | 8 | 6 | 7 | 13 |
| 4 | 9 | 0 | 1 | 5 | 3 | 1 | 5 | 4 | 8 | 4 | 12 |
| 3 | 5 | 1 | 0 | 2 | 2 | 1 | 3 | 2 | 4 | 3 | 7 |
| 2 | 3 | 0 | 1 | 2 | 0 | 1 | 2 | 1 | 2 | 2 | 4 |
| 1 | 2 | 1 | 0 | 1 | 0 | 1 | 2 | 0 | 1 | 1 | 2 |

The most important information of the present output appears in columns K and L. The relationship between the number of extinctions versus number of appearances (or originations) per UA can be plotted onto a bivariate graph providing an indirect measurement of the faunal turnover rates (see Guex 1991, p. 166 for details). The slope of the curve is obtained directly by the ratio # column G/ # column H. We note in passing that the beginning and the end of such turnover curves are biased by the fact that the base and the top of the original range chart are respectively recording only truncated originations and disappearances. These parts of the curve must obviously be ignored. We also note that large gaps in the fossil record generate turnover curves which show high extinction peaks immediately followed by high diversification peaks.

Column B shows the variation of the faunal diversity. A study of the variations of these parameters in correlation with other biotic and abiotic events during the Cretaceous has been done by O'Dogherty & Guex (in press)

Example 2: Tethyan Jurassic to Lower Cretaceous Radiolarians Zonations

Some interesting comparisons between old and recent databases can be established by calculating faunal turnover rates using BG_T05.

As an example we illustrate here (Figs. 11A and B) the tethyan Jurassic to Lower Cretaceous radiolarian turnover rates deduced from Baumgartner's 1984 database (after Guex 1991) and compare it with the faunal turnover rates deduced from Baumgartner et al. 1995 new database. We note that the general shape of the two graphs is the same, both differing mainly by the number of dots which is 10 times larger in the most recent one. The significance of this is that Baumgartner's 1984 UA zonation did not change drastically over one decade.

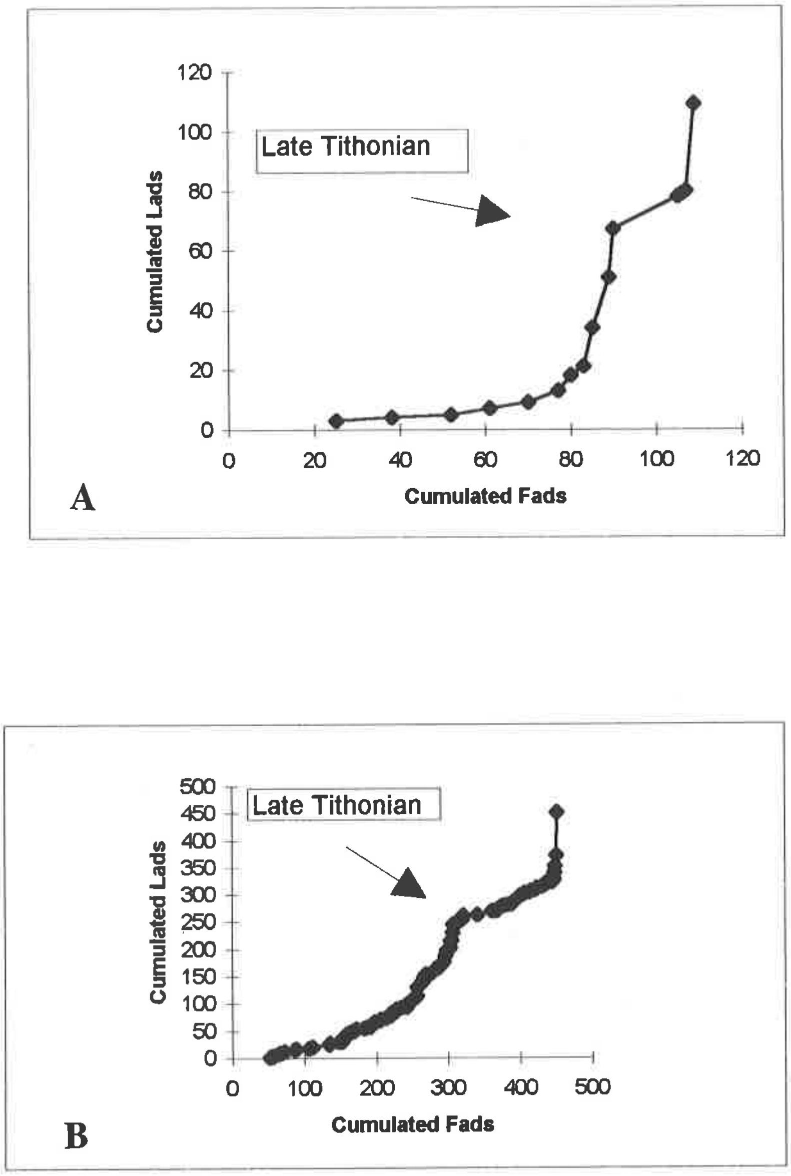


Fig.11 A: Faunal turnover rate of the Bathonian to Lower Cretaceous radiolarians established from Baumgartner's 1984 data (after Guex 1991). B: The same calculated after the data compliled in Baumgartner et al. (1995). Note the similitude of the two curves

This conclusion is complemented by an unpublished test done by Luis O'Dogherty who noticed that the taxa used by Baumgartner (1984) to construct his first major radiolarian UA zonation are used about hundred times more often than the other ones in the recent literature.

Example 3: Correlation between radiolarians extinction rates and oceanic anoxic events

Using the kind of diagram discussed here, Luis O'Dogherty (1994) was the first to demonstrate a strong correlation between radiolarians high extinction rates and oceanic anoxic events during the Cretaceous. His result is illustrated below (Fig.12).

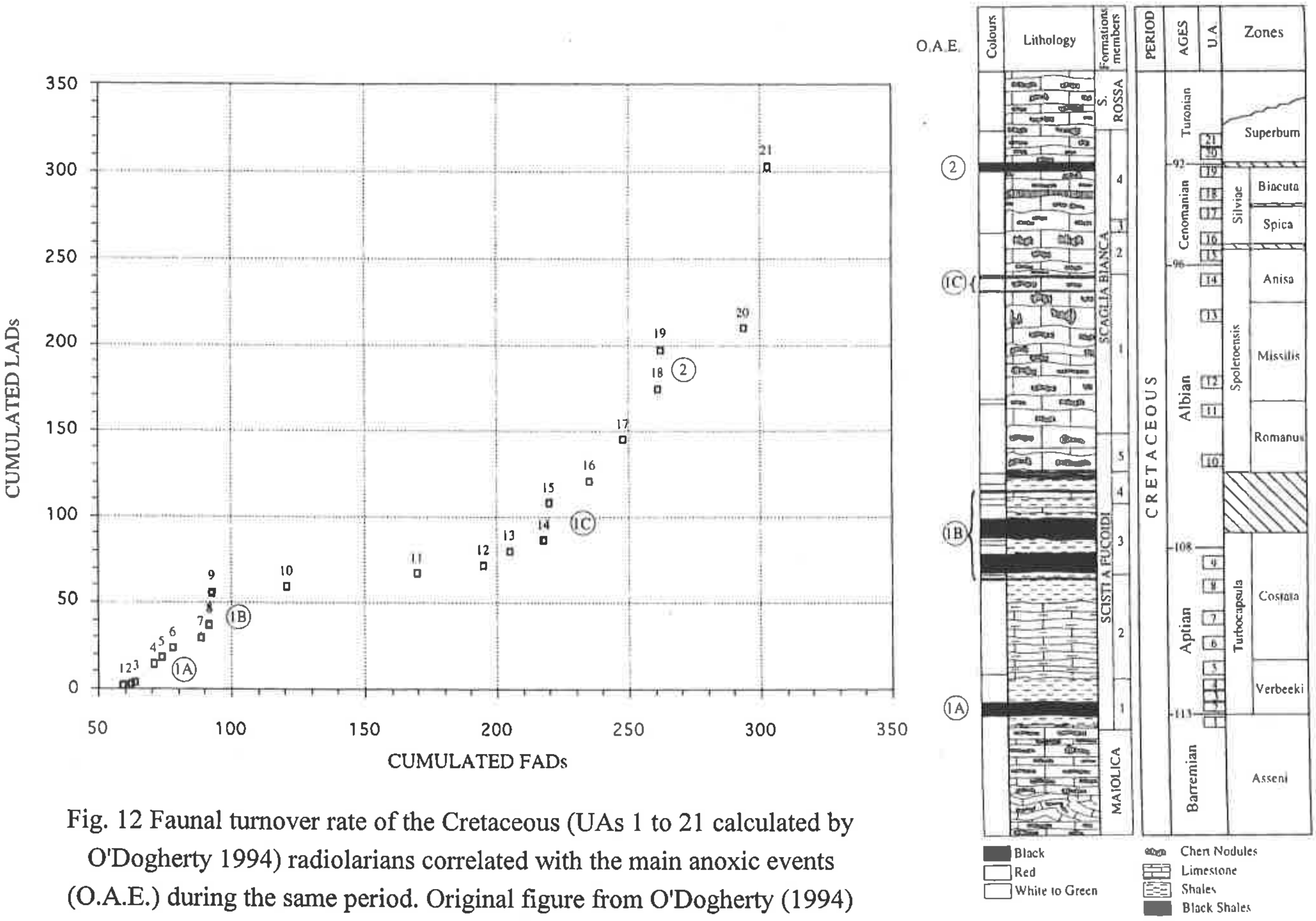


Fig. 12 Faunal turnover rate of the Cretaceous (UAs 1 to 21 calculated by O'Dogherty 1994) radiolarians correlated with the main anoxic events (O.A.E.) during the same period. Original figure from O'Dogherty (1994)

The precise origin of the Cretaceous oceanic anoxic events is unknown but it is probably different from that occurring in the Lower Toarcian (Morettini, Baumgartner, Guex, Hunziker in press).

The great Upper Domerian regression is followed by a major gap in the topmost Pliensbachian and Lowermost Toarcian in NW-Europe. That gap corresponds to the Polymorphum/Mirabile Beds found in Tethys. It is probably concomitant with the major development of forests on the newly emerged lands. At the onset of the transgression occurring during the deposition of the Paltum/Tenuicostatum Beds (i.e. younger than the tethyan Polymorphum Beds) a large amount of organic matter has been washed into the sea and its partial oxidation induced the major anoxic event and a high rate of organic deposition.

### 3.4.6 BG_T06 : Compare UA with *G**

Syntax: C:\BIOGRAPH> BG_T06 Filename-1.
Output: Filename.T06

This tool is designed to compare a biochronological synthesis constructed by means of BioGraph with a synthesis (range chart) published in the literature and constructed by means of any other method (e.g. probabilistic, multivariate or empirical). To be meaningful, the comparison should run on the very same original database.

Let "Filename-1" be the original database processed by BioGraph and let "Filename-2" be a formerly published range chart constructed from the same biostratigraphic data: both must have exactly the same set of taxa. We consider this range chart as a single section and process it with BioGraph. Then we proceed as follows:

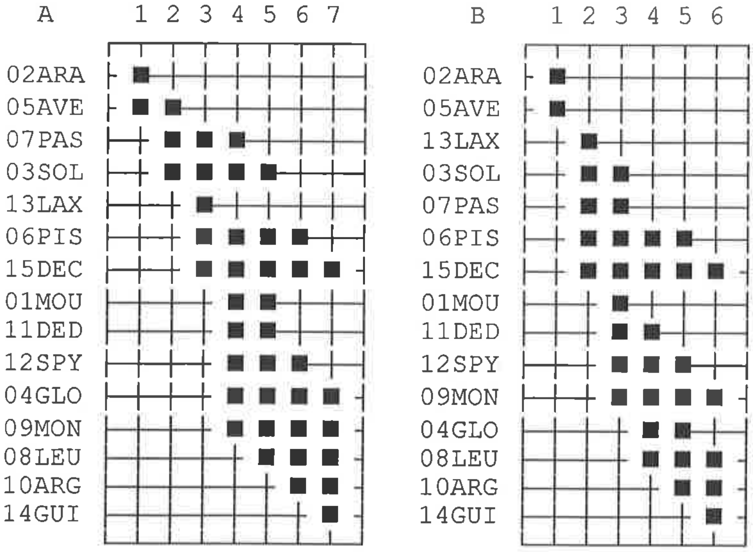
- erase the initial binary file called "Filename-1.BGI" created in BioGraph's directory
- rename as "Filename-1.BGI" the resulting "Filename-2.BGI" generated in BioGraph's directory
- run the tool

The output provides the following lists:

- destroyed edges = co-occurrences that were observed in the original database but were omitted in the published synthesis ("Filename-2")
- reversed arcs = superpositions that were observed in one direction in the original database but were reversed in the published synthesis ("Filename-2")

Example 1: Alveolinid

Comparing the false range chart given in Fig. 13B with the original data of our standard alveolinid example produces the result given in Fig. 13C.



C

DESTROYED EDGES
01MOU 08LEU
03SOL 05AVE
03SOL 08LEU
04GLO 07PAS
04GLO 14GUI
05AVE 07PAS
6 destroyed edges

REVERSE ARCS
03SOL 04GLO
1 reverse arc

Fig. 13 A: Correct range chart calculated after the Alveolinid database. B: False range chart to be compared with the correct one given in A. C: List of the mistakes found by BG_T06 in range chart B.

### 3.4.7 BG_T07 : Inter-UAs distances

Syntax: C:\BIOGRAPH> BG_T07 Filename.
Output: Filename.T07

The details of the formula are given in Guex 1991 (p. 166-167).

Example: Alveolinid

```
 1 \  2   1.166667
 2 \  3   0.933333
 3 \  4   0.755556
 4 \  5   0.222222
 5 \  6   0.476190
 6 \  7   0.452381
```

### 3.4.8 BG_T08 : Real arcs joining the UAs

Syntax: C:\BIOGRAPH> BG_T08 Filename.
Output: Filename.T08

Several inter-UA superpositions (arcs) represented in the range chart produced by BioGraph (Filename.TGJ) are constructed by transitivity. The present tool provides the number of real inter-species superpositions joining two adjacent UAs in the range chart.

Example: Alveolinid UAs

```
 1 >  2     2
 2 >  3     3
 3 >  4     5
 4 >  5     1
 5 >  6     3
 6 >  7     2
```

### 3.4.9 BG_T09 : Inter-cliques contradictions.

Syntax: C:\BIOGRAPH> BG_T09 Filename.
Output: Filename.T09

This tool produces an output consisting of 5 columns. Columns 1 and 2 give the list of the maximal cliques $k_i$ and $k_j$ being compared. Column 3 gives the number of arcs going from $k_i$ to $k_j$ ("*A*>" = arcs $i \rightarrow j$) ) and column 4 gives the number of arcs going from $k_j$ to $k_i$ ("*A*<" = arcs $i \leftarrow j$). Column 5 gives the ratio between *A*> and *A*<. This routine must be used to analyse the problems in which $G_k$ contains strongly connected components. It is designed to locate the pairs of maximal horizons that are generating the highest rates of inter-cliques contradictions. The individual pairs of highly contradictive horizons can be studied by means of the tool BG_T14 (see below).

Example of output: Alveolinid

```
"ki  kj"  "A>" "A<" "f(A)"          "ki  kj"  "A>" "A<" "f(A)"
 2   4     2    2   1.0000           4   7     2    7   0.2857
* 7  11    4    3   0.7500           1   7     1    4   0.2500
 1   11    2    3   0.6667           3   11    1    4   0.2500
 2   9     2    1   0.5000           4   5     4    1   0.2500
 2   11    2    4   0.5000           5   11    1    4   0.2500
 3   5     2    4   0.5000           7   9     4    1   0.2500
 4   9     2    1   0.5000           9   10    1    4   0.2500
 5   9     1    2   0.5000           2   3     10   2   0.2000
 6   7     2    1   0.5000           3   7     2    11  0.1818
 6   9     2    1   0.5000           3   6     1    10  0.1000
 9   11    2    1   0.5000           3   10    1    11  0.0909
* see below
```

NB: The cliques' code numbers are those used by BioGraph in the computation. The relationship beween these numbers and the residual maximal horizons' original numbers which are "sources" of the cliques is given in the output "Filename.TGE" In the above example we note that the pair of cliques #7 and #11 have a very high level of contradictions (4 arcs in one direction and 3 in the opposite direction). The sources of these cliques are respectively maximal horizons 2.2 and 1.4 (see below: example in TOOL BG_T14).

Example taken from the same database (Alveolinid.TGE):

```
clique          km    card species code numbers
1          2.3    [6]: 01MOU 06PIS 07PAS 09MON 11DED 12SPY
2          3.5*   [6]: 06PIS 07PAS 09MON 11DED 12SPY 15DEC
3          8.5*   [5]: 04GLO 08LEU 09MON 10ARG 14GUI
4          4.2    [5]: 01MOU 08LEU 09MON 11DED 12SPY
5          9.2*   [5]: 06PIS 09MON 10ARG 12SPY 15DEC
6          3.2*   [4]: 03SOL 06PIS 07PAS 12SPY
7          2.2*   [4]: 06PIS 07PAS 13LAX 15DEC
8          4.4    [4]: 09MON 10ARG 14GUI 15DEC
9          1.3*   [4]: 04GLO 07PAS 09MON 12SPY
10         2.1    [3]: 03SOL 05AVE 07PAS
11         1.4*   [3]: 03SOL 08LEU 12SPY
12         8.1    [2]: 02ARA 05AVE
```

### 3.4.10 BG_T10 : Strongly connected components of *G**

Syntax: C:\BIOGRAPH> BG_T10 Filename.
Output: Filename.T10

This tool enumerates the list of vertices (taxa) that are present in strongly connected components of *G**. It provides the frequency and reproducibility of the arcs incident to each of them.

Note: The outputs of the tools BG_CS3 and BG_CSZ4 described below provide a complete list of the semi-oriented and oriented cycles of length 3 and 4 present in *G**. These lists are more informative about the complexity of the studied problems than the present routine.

### 3.4.11 BG_T11: Replacing code numbers by names in the range chart *.TGJ

Syntax: C:\BIOGRAPH> BG_T11 Filename1. Filename2.

NB: Filename1 is the database and Filename2 is the dictionary (*.DCT) (see sect. 3.2.1 above). The two names (1 and 2) can be identical.

Output: Filename.TGJ with the full taxonomic names.

The range chart automatically generated by BioGraph (output *.TGJ) contains only the code numbers of the taxa. These codes will be replaced by the complete names of the taxa by running the current tool.

Example: Alveolinid

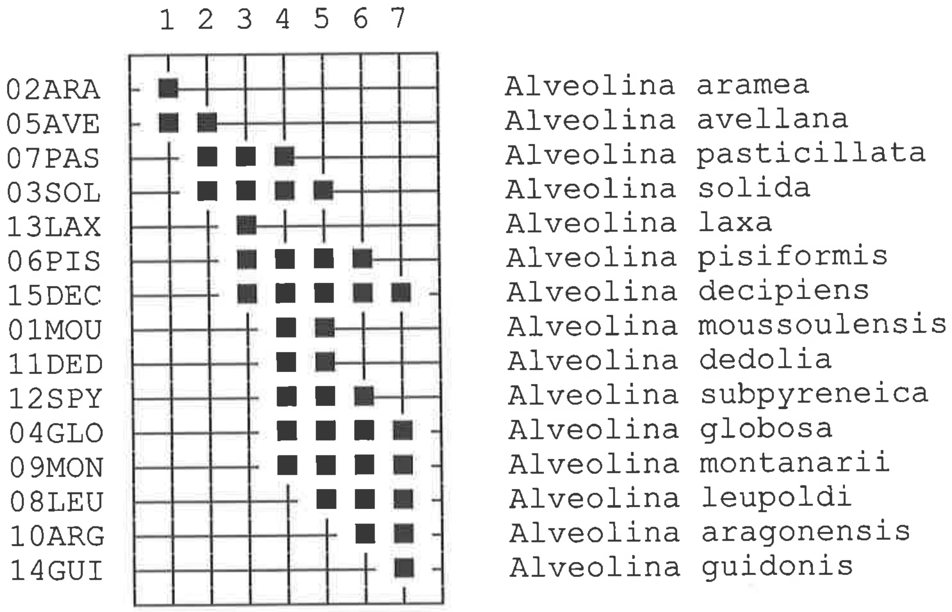


Fig. 14 Range chart with the species names introduced by BG_T11

### 3.4.12 BG_T12: Contradictions between a biochronological scale and its application to correlation

Syntax: C:\BIOGRAPH> BG_T12 Filename1. Filename2.

Output: The list of stratigraphic sections with their respective levels and the correlation of these beds by means of the formerly published range-chart currently analysed. The message "unknown correlation" appears everywhere a particular bed contains a fossil assemblage omitted in that range chart.

The philosophy of this tool is close to that of BG_T06. It is designed to compare biochronological correlations constructed by means of BioGraph with correlations based on a range chart published in the literature and constructed by means of another method. To be meaningful, the comparison must concern the very same original database.

Let "Filename1" be the original database processed by BioGraph and let "Filename2" be a formerly published range chart constructed after the same biostratigraphic data: both have exactly the same set of taxa. We consider the range chart called "Filename2" as a single section and process it with BioGraph to construct the binary files which are necessary for the comparison. Then we just run the tool.

Example: Alveolinid

If we apply the false range chart of Fig. 13B to the original alveolinid database we obtain the following result.

```
Section 1
 4: Unknown correlation ! (because association between species 03 and 08 is not
represented in the false range chart etc)
 3: Unknown correlation !
 2:  2 ..  5
 1:  1
Section 2
 4:  6
 3:  3
 2:  2
 1: Unknown correlation !     etc....
```

### 3.4.13 BG_T13: Reproducibility table with groupings of UAs

Syntax: C:\BIOGRAPH> BG_T13 Filename.
Output: Filename.T13

The reproducibility table produced automatically in the output *.TGL of BioGraph's main menu shows only the UAs which are strictly identified in the sections. It does not include the groupings of UAs that appear in the correlation table (output *.TGK) (see Guex 1991, p.28 for details). The present tool produced a completed reproducibility table where these unions of UAs are shown. This makes easier the zonal interpretation of the data (e.g. finding significant zonal boundaries etc).

Example: the completed reproducibility table of the alveolinid problem is illustrated in fig.15.

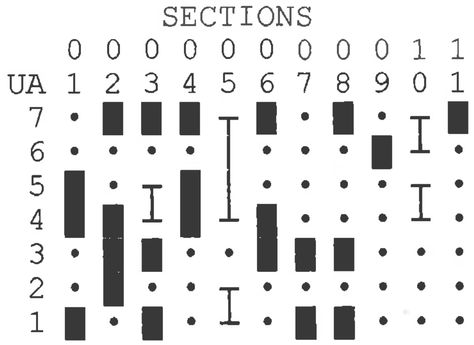


Fig. 15 Completed reproducibility table indicating the unions of UAs which are identified in the sections (eg: union of UAs 4,5 and 6,7 in section 10 etc). Alveolinid data

### 3.4.14 BG_T14: Detailed analysis of the inter-horizon contradictions

Syntax: C:\BIOGRAPH> BG_T14 Filename. *i.j* *k.l*
Note: *i.j* means horizon *j* of section *i* and *k.l* means horizon *l* of section *k*
Output: Filename.T14

During the analysis of highly contradictory data (e.g. problems with strongly connected components in $G_k$), it is very useful to locate the pairs of maximal horizons which are connected by the greatest number of opposed arcs (see Tool BG_T09 above). The output of the present tool shows the list of arcs connecting horizon *j* of section *i* to horizon *l* of section *k* and the list of arcs connecting both horizons in opposite directions.

```
Example Alveolinid. 1.4 2.2
Horizon 1.4 checked with horizon 2.2
List of arcs: 1.4→2.2
03SOL→13LAX
List of arcs: 1.4←2.2
08LEU←06PIS   08LEU←07PAS   08LEU←13LAX
```

### 3.4.15 BG_CS3: List of $C_3$ and $S_3$ present in $G^*$

Syntax: C:\BIOGRAPH>BG_CS3 Filename > Filename.CS3
Output: List of $C_3$ and $S_3$ present in $G^*$

This tool is designed to compute the list of semi-oriented circuits $S_3$ and cycles $C_3$ present in the biostratigraphic graph. The taxonomic code numbers used in that list are those appearing in the repertoir (*.REP) created by BioGraph.

Example: Alveolinid

```
List of cycles of length 3
C3:  1>   4>   3>          S3:  3>   14-  4>
S3:  1>   15-  13>         S3:  3>   13>  8-
S3:  3-   5>   4>          S3:  3>   13>  12-
S3:  3-   6>   4>          S3:  4>   15-  6>
S3:  3>   9-   4>          S3:  4>   15-  11>
S3:  3>   10-  4>          S3:  4>   15-  13>
C3:  3>   11>  4>          S3:  6>   8>   15-
C3:  3>   13>  4>          S3:  7>   8>   15-
S3:  8>   15-  13>

Number of cycles in which the species 1 to 15 are implicated:
        1:   2               9:  1
        2:   0              10:  1
        3:   10             11:  2
        4:   11             12:  1
        5:   1              13:  6
        6:   3              14:  1
        7:   1              15:  7
        8:   4
```

### 3.4.16 BG_CSZ4: List of $S_4$, $S'_4$, $Z_4$, $Z'_4$, $Z''_4$ and $C_4$ of $G^*$

Syntax: C:\BIOGRAPH>BG_CSZ4 Filename > Filename.CS4

Output: List of $S_4$, $S'_4$, $Z''_4$ and $C_4$ present in $G^*$

This tool is designed to compute the list of semi-oriented circuits $S_4$, $S'_4$, $Z''_4$ and the cycles $C_4$ present in the biostratigraphic graph. The taxonomic code numbers used in that list are those appearing in the repertoir (*.REP) created by BioGraph.

Note that the routine is slow when used to treat problems with a great number of taxa: the computation time grows as $N_4$ (where $N$ = number of taxa). It is not recommended to use it for more than 200 species because it is too time consuming and a normal hard disk does not accept the output which can be too large.

Example: Alveolinid

```
List of cycles of length 4:
Z"4: 6-   3-   8-   1-;  6>   8;   3>   1;
Z"4: 7-   3-   8-   1-;  7>   8;   3>   1;
Z"4: 1-   7-   4-   8-;  1>   4;   7>   8;
Z"4: 1-   6-   10-  8-;  1>   10;  6>   8;
Z"4: 7-   4-   8-   3-;  7>   8;   4>   3;
Z"4: 3-   6-   9-   8-;  3>   9;   6>   8;
Z"4: 3-   6-   10-  8-;  3>   10;  6>   8;
Z"4: 3-   6-   11-  8-;  3>   11;  6>   8;
Z"4: 3-   7-   9-   8-;  3>   9;   7>   8;
Z"4: 3-   7-   11-  8-;  3>   11;  7>   8;
Z"4: 7-   6-   10-  4-;  7>   10;  6>   4;
```

```
Z"4: 7-   11-  8-   4-;  7>   8;   11>  4;
Z"4: 4-   7-   15-  10-; 4>   15;  7>   10;
Z"4: 4-   7-   15-  14-; 4>   15;  7>   14;
Z"4: 4-   12-  15-  14-; 4>   15;  12>  14;
Z"4: 6-   11-  8-   10-; 6>   8;   11>  10;
Z"4: 8-   11-  15-  10-; 8>   15;  11>  10;
Z"4: 8-   11-  15-  14-; 8>   15;  11>  14;
Z"4: 8-   12-  15-  14-; 8>   15;  12>  14;
```

Implication coefficient of species 1 to 15:

| | | | |
|---|---|---|---|
| 1: | 4 | 9: | 2 |
| 2: | 0 | 10: | 6 |
| 3: | 8 | 11: | 6 |
| 4: | 7 | 12: | 2 |
| 5: | 0 | 13: | 0 |
| 6: | 7 | 14: | 4 |
| 7: | 9 | 15: | 6 |
| 8: | 15 | | |

Correspondance of taxonomic codes and codes used in BG_CSZ4 is given in the output Repertoir (Filename. REP) of the main program.

| | | | |
|---|---|---|---|
| 1 | 01MOU | 9 | 09MON |
| 2 | 02ARA | 10 | 10ARG |
| 3 | 03SOL | 11 | 11DED |
| 4 | 04GLO | 12 | 12SPY |
| 5 | 05AVE | 13 | 13LAX |
| 6 | 06PIS | 14 | 14GUI |
| 7 | 07PAS | 15 | 15DEC |
| 8 | 08LEU | | |

### 3.4.17 BG_DIA: Diachronism of datums

Syntax: C:\BIOGRAPH>BG_DIA Filename > Filename.DIA
Output: Diachronism of the datums.

This tool is designed to help the analysis of datum's diachronism. Its output is divided into two parts. The first is a list of the species in each section. Two double columns of numbers face the code number of the species. The first column represents the position of the species' FAD expressed in terms of UAs. If the two numbers are identical the position is assigned to a single UA. If they are different, that position is assigned to an uncertainty interval corresponding to more than one UA. The second column represents the LAD of the species expressed in the same manner. At the end of the file there is a table taking the census of the biochronologic dispersion of the different datums.

Technical remarks

- To express the diachronisms in terms of zones and not in terms of UAs the user should proceed as follows. First: Edit the numerical range chart (*.TGI) produced by BioGraph and renumber the UAs of that table by their corresponding zonal number. For example if UAs 7, 8 and 9 of a given output correspond to UA-Zone 4 of the zonal interpretation, we replace 7, 8 and 9 by 4. Then we edit the data file of the problem and we add the renumberd *.TGI to it. Finally we run this modified datafile and thus obtain the correlation expressed in terms of zones.

- If the lower (or upper) limit of a UA-Zone is outside the stratigraphic interval recorded in a given section, we must ignore the first local appearances (or disappearances) that occur at the base (or top) of this section when establishing the values of the biochronologic dispersions (see Guex 1991 p.105 for details). These corrections are not done automatically by GB_DIA and they must be done manually.

- HELLY's property and datum's diachroneity

It is important to keep in mind that the uniqueness of a coexistence interval of $n$ species (in other words, a maximal clique with $n$ vertices) can be established based on fragmentary biostratigraphic data coming from a great many localities (at most $(n^2\text{-}n)/2$ localities in which only a pair of species is found each time).

This fundamental property of cliques whose vertices represent intervals is called HELLY's property. BERGE (1973, p.352) states it as follows:"If a family of intervals does not contain two disjoint intervals, then a point exists that is common to all of them."

We can prove this proposition thus:

Let

$J = \{J_1, J_2, \ldots, J_n\}$ be a family of intervals;

$x$ be the set of points of the interval $J_i$

$m_i$ and $M_i$ be the minimum and maximum of $J_i$ so that $J_i = [m_i, M_i] = \{x \mid M_i \geq x \geq m_i$ for all $i\}$;

$J_k$ be the interval with the smallest $M_k$;

$J_j$ be the interval with the largest $m_j$

By hypothesis, $J_j \cap J_k \neq \varnothing \implies M_k \geq m_i$ for all $i$. In other words we have:

$M_k \geq m_i$

Consequently: $[m_j, M_k] = \cap\ [m_i, M_j] \neq \varnothing$.

It follows from this assertion that, in an interval graph, a maximal clique characterizes a unique interval. This interval of minimal duration is the intersection of all the intervals making up the maximal clique.

A datum (FAD or LAD) is demonstrated to be diachronous if it occurs stratigraphically below the interval $[m_j, M_k]$ in some locality and above $[m_j, M_k]$ in some other locality.

Example: Alveolinid database

Local diachronism

Section 1

| Taxa | FAD in UA(s) | LAD in UA(s) |
|---|---|---|
| 2: | 1- 1 | 1- 1 |
| 3: | 5- 5 | 5- 5 |
| 4: | 4- 4 | 4- 4 |
| 6: | 3- 6 | 3- 6 |
| 7: | 4- 4 | 4- 4 |
| 8: | 5- 5 | 5- 5 |
| 12: | 4- 4 | 4- 4 |

Section 2

| Taxa | FAD in UA(s) | LAD in UA(s) |
|---|---|---|
| 1: | 4- 4 | 4- 4 |
| 3: | 2- 2 | 2- 2 |
| 5: | 2- 2 | 2- 2 |
| 6: | 3- 3 | 4- 4 |
| 7: | 2- 2 | 4- 4 |
| 9: | 4- 4 | 7- 7 |
| 11: | 4- 4 | 4- 4 |
| 12: | 4- 4 | 4- 4 |
| 13: | 3- 3 | 3- 3 |
| 14: | 7- 7 | 7- 7 |

etc...

Total diachronism:

```
 x|FADmin [s ] FADmax [s ] = Da |LADmin [s ] LADmax [s ] = Dd |Dt
--+-----------------------------+------------------------------+
 1| 4-  4 [ 2]  4-  5 [10] =  0 | 4-  4 [ 2]  5-  5 [ 4] =  1 | 1
 2| 1-  1 [ 1]  1-  1 [ 1] =  0 | 1-  1 [ 1]  1-  1 [ 1] =  0 | 0
 3| 2-  2 [ 2]  5-  5 [ 1] =  3 | 2-  2 [ 2]  5-  5 [ 1] =  3 | 6
 4| 4-  4 [ 1]  7-  7 [ 4] =  3 | 4-  4 [ 1]  7-  7 [ 4] =  3 | 6
 5| 1-  1 [ 8]  2-  2 [ 2] =  1 | 1-  1 [ 8]  2-  2 [ 2] =  1 | 2
 6| 3-  3 [ 2]  6-  6 [ 9] =  3 | 3-  3 [ 3]  6-  6 [ 9] =  3 | 6
 7| 2-  2 [ 2]  4-  4 [ 1] =  2 | 2-  4 [ 8]  4-  4 [ 1] =  0 | 2
 8| 5-  5 [ 1]  7-  7 [ 8] =  2 | 5-  5 [ 1]  7-  7 [ 8] =  2 | 4
 9| 4-  4 [ 2]  7-  7 [ 3] =  3 | 4-  7 [ 6]  7-  7 [ 2] =  0 | 3
10| 6-  7 [ 9]  7-  7 [ 3] =  0 | 6-  6 [ 9]  7-  7 [ 3] =  1 | 1
11| 4-  4 [ 2]  4-  5 [ 3] =  0 | 4-  4 [ 2]  5-  5 [ 4] =  1 | 1
12| 4-  4 [ 1]  6-  6 [ 9] =  2 | 4-  4 [ 1]  6-  6 [ 9] =  2 | 4
13| 3-  3 [ 2]  3-  3 [ 2] =  0 | 3-  3 [ 2]  3-  3 [ 2] =  0 | 0
14| 7-  7 [ 2]  7-  7 [ 2] =  0 | 7-  7 [ 2]  7-  7 [ 2] =  0 | 0
15| 3-  3 [ 7]  7-  7 [ 4] =  4 | 3-  3 [ 7]  7-  7 [ 3] =  4 | 8
```

### 3.4.18 BG_CON: Weights of the arcs in $G_k$

Syntax: C:\BIOGRAPH\BG_CON Filename > Filename.CON
Output: Weights of the contradictive arcs in $G_k$.

This tool is similar to BG_T09 described above but it expresses the inter-cliques contradictive relationships by means of the value of coefficient $C_{ij}$ defined in Section 2.4.5.1 in Guex (1991, p. 83 ; see also Savary and Guex 1991, p. 322). That coefficient is here multiplied by 1000. The output is divided into seven columns. $k_i$ is the $i^{th}$ clique and $k_j$ is the $j^{th}$ clique. The sign > symbolizes arc → going from $i$ to $j$ (remember that $i \to j$ means $i$ below $j$). Column A: Number of arcs going from $i$ to $J$ Column B: Total reproducibility of the arcs of column A. Column C: Number of arcs going from $j$ to $i$. Column D: Total reproducibility of the arcs of column C. The cliques are numbered ordinally and the maximal horizons that are sources of the maximal cliques are listed in the file *.TGE (output of BioGraph).

Example: Alveolinid

```
ki  kj      A   B    C   D    Cij
1 → 3:     10>+ 29 &  2< + 2  =  102
```

means that clique $k_1$ is considered to be older than clique $k_3$. There are 10 arcs going from $k_1$ to $k_3$ and the total reproducibility of these arcs is 29. There are 2 conflicting arcs going from $k_3$ to $k_1$ and the reproducibility of these arcs is equal to 2. The value of $C_{ij}$ is 1000 x (2+2) / (10+39) = 102. Whenever two inter-cliques conflicting arcs have the same weight, the relationship between $i$ and $j$ is undefined (?) and $C_{ij}$=1000.

Remark: The beginning of the output of tool BG_CON provides the number and reproducibility of the non conflicting arcs connecting two maximal cliques. This is not commented here.

### 3.4.19 BG_UNI: Unions of cliques

Syntax: C:\BIOGRAPH\BG_UNI Filename > Filename.UNI
Output: Unions of cliques

This tool provides the list of the cliques that are included in larger cliques during the computation of the maximal cliques.

Example: Alveolinid

```
 3.4* +  3.5*
 4.3* +  8.5*
 7.4* +  2.2*
```

# 4 Miscellaneous Comments and Recommendations

## 4.1 Basic relationships between the UA method and graph theory

Biostratigraphers are familiar with all sorts of range charts representing biochronological syntheses. The inter-taxa relationships (coexistences and superpositions) given in such range charts can be represented by graphs (i.e. in the graph theoretical sense). Such graphs are called interval graphs and they are characterized by the fact that they admit no cycles of any kind ($S_3$, $Z_4$, $C_n$ etc) and a fortiori no cycles in their maximal clique graph ($G_k$). The goal of the UA method it to analyze the cycles present in a dataset and to optimize their destruction by adding virtual coexistences into the initial graph. Such adjunctions are obviously justified by the fact that each individual cyclic stratigraphic relationship implies that one chronological coexistence between two species has not been recorded in the sediments (see Fig. 2). Note also that one single conflicting superposition can generate several cycles.

When constructing biochronological scales based on several tens or hunderds of species it is tempting to reduce the complexity of the stratigraphic data by chosing a few species within particular evolutionary lineages and use them to recognize some evolutionary events defining zonal boundaries. Doing so is often equivalent to ignoring the majority of the species making up the stratigraphic data which are potentially useful to construct biochronological scales. It is also equivalent to ignoring the fact that cycles and strongly connected components in $G_k$ are mathematical concepts with a very real and concrete significance in biostratigraphic data: they are generated by the multitude of conflicting ranges generated by discontinuous fossil record (see Fig.2) and by taxonomic mistakes. On the other hand we recall that maximal cliques of a graph representing biostratigraphic data correspond to inter-species coexistence intervals of minimal duration. Datums (Fads/Lads) which occur stratigraphically below such intervals in some locality and above them in some other locality are demonstrated to be diachronous and cannot be used to define zonal boundaries. Ignoring these facts can lead to biochronological syntheses where the relative positions of the datums which are supposed to be chronologically significant are in total contradiction with the rest of the data.

## 4.2 Strongly connected components in the graph $G_k$

The BioGraph computer program is designed to produce an output for any kind of data, even the worst. Taxonomic errors, reworking, poor sampling etc...can generate strongly connected components in the graph $G_k$. Such data should be treated with caution and the user should discover himself why such structures are generated. To do this, the user is provided with several tools and outputs:

- The text-file *.TGG provides a list of vertices (representing maximal cliques) of $G_k$ which belong to the strongly connected components.
- The tool BG_T09 is designed to locate the pairs of maximal cliques with the greatest score of contradictive superpositions. To locate the maximal horizons that are sources of these cliques, the user should print the output *.TGE of BioGraph and then compare the pairs of chosen horizons by means of the tool BG_T14 (see example above).

- Contradictive stratigraphic relationships can be eliminated by going back to the original samples. Suppose that two samples *A* (containing species 1 and 2) and *B* (containing species 3 and 4) are connected by two contradictive arcs (1→3) and (4→2) (go back to Fig.2). A revision of the samples can show that species 3 (or 4) was present in sample *A* but remained unnoticed, or that species 1 (or 2) was present in sample *B* but remained unnoticed. The stratigraphic contradiction is solved if the revision displays such a situation.

The strongly connected components of $G_k$ are sometimes generated by local sequences wich are partially reversed when compared to the other sequences. Once a local sequence is suspected to contain inaccurate biostratigraphic data, it is recommended to disconnect its local horizons and treat each sample as a single sections before reprocessing the data: this can immediately reveal which samples have an abnormal fossil content.

As an example we can apply this procedure to the Alveolinid data. In this database, section 1 (Fatji Hrib) is known to contain a reworked sample (Level 4). By disconnecting the samples we transform the original section into 4 new sections, each of which corresponding to a single bed of the original section. Then we reprocess the data and get an output with 8 UAs. The resulting correlation chart (*.TGK) for the disconnected samples is as follows:

```
Section FATJI_HRIB4
Level 4:  UA 3 -   3
Section FATJI_HRIB3
Level  3:  UA 6 -   6
Section FATJI_HRIB2
Level  2:  UA 3 -   7
Section FATJI_HRIB1
Level  1:  UA 1 -   1
```

We note immediately that level 4 of Fatji Hrib, which contains a reworked fauna, is assigned to an UA (UA 3) which is older than that of level 3 (assigned to UA 6): this is typically one kind of test which allows us to solve the difficult problem of interpreting the strongly connected components in a given problem.

### 4.3 Composite sections and common sense

Suppose you create a database with the following data: Section 1 contains Ammonite, Section 2 contains Trilobite and Section 3 contains Nummulite. Even the most sophisticated program will be unable to organize these 3 sections in the correct order, for it will be unable to guess that Nummulite is younger than Ammonite which is itself younger than Trilobite. BioGraph will consider the three fossil groups as identically uninformative from a biochronologic viewpoint. This example illustrates one of the reasons why composite stratigraphic sections should be constructed whenever possible (other details in Guex 1991, p.18 and p.42). In reality, most of the contradictive stratigraphic relationships between taxa are generated by discontinuities in the fossil record. Such discontinuities can be largely reduced by revisions and reexamination of the original data and by the use of composite sections.

Another occasion where the use of composite sections is necessary is when the stratigraphic thickness of a section recording a given time interval is shorter than the life span of the species used to construct the zonation: such sections have no reliable information about chronological sequences of species and should be considered as isolated samples.

Long ranging and discontinuous species can give rise to difficult problems. Suppose for example that a fauna *A* in locality 1 contains a particular species *x*. By means of non-biochronological correlations, fauna *A* is known to be older than a fauna *B* occurring in a locality 2. Suppose that species *x* is observed above fauna *B* in that locality 2. Running the data without making a composite section superposing locality 1 and locality 2 will produce an output where fauna *A* appears to be younger than *B* because *A* contains *x*. This would clearly be false, but the program has no way of guessing that *x* is a bad chronological marker. The quality of the outputs depends mostly on the quality of the inputs.

In general BioGaph requires common sense before calculating correlations. It goes without saying that it would be nonsense to try to correlate a Triassic section containing bivalve sp., brachiopod sp, and ammonite sp. with a Cretaceous section containing a taxonomically identical fauna.

## 4.4 Precision of data and precision of results

Excellent correlations can be obtained thanks to fossil groups with bad reputation such as benthic forams (see below) and bad correlations can be generated by ideal fossils like ammonites.

In fact each individual correlation problem has it's own complexity, depending on the longevity of the taxa under study (i.e. short ranging vs long ranging) and on the nature of the stratigraphical record. The most difficult problems are generated by vertical discontinuities in the record of long ranging organisms. Such discontinuities can generate data containing several hundreds of thousends of individual cycles (S4 and Z4 type: see Fig.2). Remember that each individual cycle corresponds to one missing coexistence, justifying the adjunction of virtual edges into the biostratigraphic graph. Remember also that each conflicting superposition can generate a great number of such cycles. It is not surprising that highly discontinuous data do not generate exceedingly precise zonations.

However some very complex data can be unbelievably good. As an example we will go back to Deboo's thesis (1965) which is well known from the theoretical stratigraphers because it was used as an example in several publications on quantitative stratigraphy (details in Guex 1991).

In the early 1960s Deboo began a comprehensive biostratigraphic study of the Paleogene west of Alabama and east of Mississippi. His goal was two-fold: first, to solve some problems posed by correlating the lithologic units classically used in this region (Fig. 16) and second, to use new biochronologic arguments to determine more precisely the boundary between the Jacksonian and Vicksburgian stages. His investigations concerned mainly the distribution of foraminifera and ostracods in five sections, which he sampled in great detail.

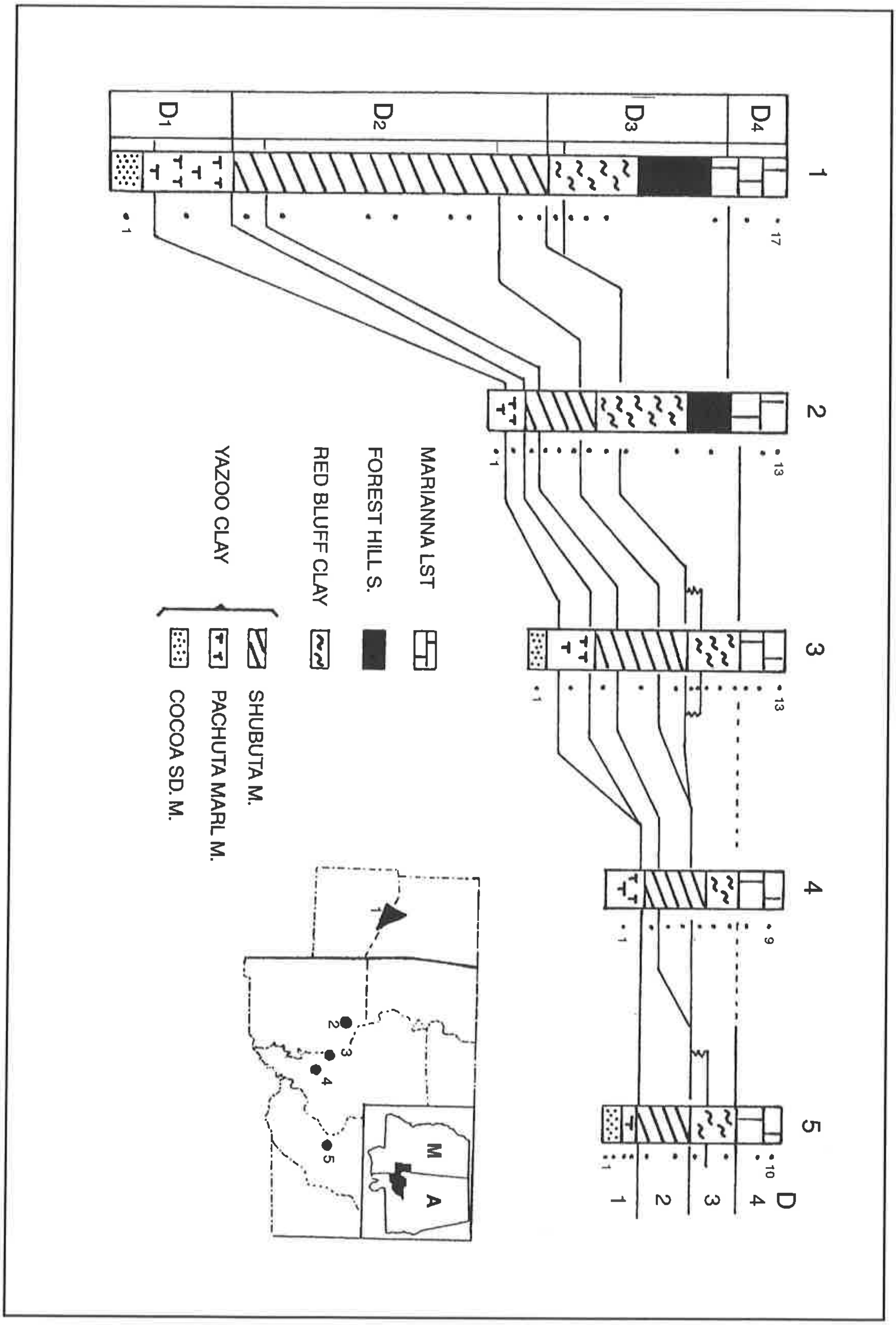


Fig. 16 Correlation of Deboo's (1963) sections by means of BioGraph. The program reveals the existence of two major trangressive-regressive cycles which are hidden when the data are analyzed empirically.

Thanks to his original observations, Deboo was able to recognize four successive zones (D1 to D4 in Fig. 16) distributed over his study area. Our analysis of the same dataset by means of the BioGraph program produces a sequence of 31 UAs demonstrating a superposition of transgressive-regressive cycles delimitated by important gaps.

Deboo's database is published in Guex (1991) and the illustration of the correlations established by means of our program is given in Fig. 16.

## 4.5 UA-Zones, Oppel Zones and Standard Zones

In our concept, UA-Zones and Oppel Zones are very similar in the sense that they are both defined by mutually exclusive associations of taxa. Classical Oppel-Zones used in the Mesozoic biochronological scale based on ammonites are sometimes called Standard Zones. We will briefly go back to some ideas expressed earlier (Guex 1979, 1991) about this concept.

The term Standard Zones is frequently used to describe the subdivisions of a biochronologic scale of supposed general value. Two drastically different views of the properties of standard zonations now exist.

The first considers that the subdivisions of a standard biochronologic scale are discrete when based on discontinuous biostratigraphic data.

According to the second point of view, a standard biochronologic scale must have the following properties:

- It must be continuous.
- Its subdivisions (zones) must be contiguous (no gap, no overlap).
- The lower limit of each standard zone must be defined by a Golden Spike in a stratotype.
- The *Golden Spike* automatically defines the top of the subjacent zone, even when faunal criteria are lacking.

We disagree with this last concept of standard scales for three reasons:

- It fails to take into account both the nature of the biostratigraphic data on which the scale is based, and the analytical methods used to establish the correlations (no distinction between continuous vs. discrete scales).
- It ignores the fact that discontinuous paleontological data (like radiolarians, nannoplankton and even ammonites etc..) necessarily generate discrete biochronologic scales: imposing a *continuum* on such scales is equivalent to overlooking that fundamental property.
- It neglects the essential requirement that a stratotype must locally record the superpositional control between two consecutive discrete zones (i.e. the Golden Spike methodology doesn't require any superpositional control between two zones).

To this we add that the famous rule according to which "*the base of a zone fixed by a Golden Spike in a stratotype automatically defines the top of the preceding zone*" can be applied only if the oldest part of the higher zone is recorded in that stratotype.

The rule obviously makes no sense if that condition is not satisfied. In other words, that convention can generate confusion and has no place in a codification of biochronologic procedures. The above ideas have been first defended by Guex in 1979 and it is interesting to note that the most recent Guidelines for the establishment of global chronostratigraphic standard (Remane et al. 1996) rightly emphasizes that correlation must precede the definition of a boundary. Schindewolf would be pleased.

In summary, we can describe a biochronologic scale as "standard" only if it is widely applicable and results from a correct synthesis. The quality of such a scale depends entirely on the method used to establish it. Each type of zonation (continuous vs. discrete) requires its own procedure for zonal definition. But whatever the procedure, the definition must not contradict observed facts.

Several tools described in the present paper have been constructed to allow the user to avoid such contradictions.

## 4.6 Interval Zones vs UA-Zones

### 4.6.1 Mixing ammonite and radiolarian data

Some fossil groups such as the radiolarians can take a long time to recover a full diversity after extinctions generated by major crises like the Triassic-Jurassic boundary. In such cases there are long periods of time during which we observe only appearances of taxa and no extinctions generating chronologically significant superpositions of taxa. Such periods void of short ranging radiolarians must be subdivided by means of Interval Zones (i.e. they cannot be subdivided by means of UA-Zones). For example, in a recent study of the Hettangian to Pliensbachian interval (Carter et. al. 1998) we were forced to use ammonites to calibrate the successive appearances of radiolarian taxa and define Interval Zones for the Lower and Middle Hettangian. True UA-Zone were defined only in the Upper Hettangian to Pliensbachian part of the sequence.

### 4.6.2 Jurassic radiolarians of Japan: Interval Zones vs UAs

In 1995 Matsuoka published an important synthesis on the Jurassic to Lower Cretaceous radiolarians from Japan. From these data he proposed to subdivide this period into nine Interval Zones. As his outstanding and highly internally consistent database is available for further investigation in Baumgartner's et al. book (1995), we think that it is useful to illustrate the 37 Unitary Associations (Fig. 17) which can be obtained thanks to these data by means of the BioGraph program. The correlation between these UAs and the Interval Zones is given in Fig. 18.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Higumastra sp. A |
|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Tripocyclia sp. |
|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Elodium cameroni |
|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Paronaella sp. aff. P. corpulenta |
|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Tympaneides charlottensis |
|■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Tetratrabs izeensis |
|■|■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Turanta morinae gr. |
|■|■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Xiphostylus spp. |
|■|■|■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Trillus spp. |
| |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Andromeda praecrassa |
| |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Eucyrtidiellum (?) quinatum |
| |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Sethocapsa cometa |
| |■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Andromeda sognoensis |
| |■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Angulobracchia sicula |
| |■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Linaresia beniderkoulensis |
| |■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Parahsuum (?) natorensis |
| |■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Bernoullius rectispinus |
| |■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Archaeohagiastrum longipes |
| |■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Parahsuum (?) magnum |
| |■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Laxtorum (?) jurassicum |
| |■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Napora sp. A |
| |■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Parahsuum (?) olorizi |
| |■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Linaresia chrafatensis |
| |■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Hexasaturnalis tetraspinus |
| |■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Mirifusus proavus |
| |■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Transhsuum hisuikyoense |
| |■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Bernoullius rectispinus s.l. |
| |■|■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Ares cylindricus cylindricus |
| |■|■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Hsuum matsuokai |
| |■|■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Ares cylindricus |
| | |■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Andromeda praepodbielensis |
| | |■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■| | | | | | | | | | | | | | Mirifusus fragilis |
| | | |■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Tetraditryma sp. cf. T. praeplena |
| | | |■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Hexasaturnalis hexagonus |
| | | | |■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Bernoullius furcospinus |
| | | | | |■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Acaeniotyle (?) variata |
| | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Quinquecapsularia megasphaerica |
| | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Tetraditryma praeplena |
| | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Archaeohagiastrum munitum |
| | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Pantanellium berriasianum |
| | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Napora nipponica |
| | | | | | |■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Zartus imlayi gr. |
| | | | | | |■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Zartus dickinsoni gr. |
| | | | | | |■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Cyrtocapsa (?) kisoensis |
| | | | | | |■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Gorgansium spp. |
| | | | | | |■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Unuma echinatus |
| | | | | | |■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Archicapsa (?) pachyderma |
| | | | | | |■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Hexastylus (?) tetradactylus |
| | | | | | |■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Acaeniotyle (?) variata ssp. B |
| | | | | | |■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Yamatoum spinosum |
| | | | | | |■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■| | | | | | | | Napora pyramidalis |
| | | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Triactoma jonesi |
| | | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Tetraditryma corralitosensis |
| | | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Emiluvia premyogii |
| | | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Emiluvia lombardensis |
| | | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Transhsuum medium |
| | | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Tritrabs simplex |
| | | | | | | |■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Acanthocircus suboblongus subobl. |
| | | | | | | |■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■| | | | | | | | | | | | | | Acanthocircus suboblongus |
| | | | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Eucyrtidiellum unumaense unumaense |
| | | | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Saitoum pagei |
| | | | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Saitoum sp. aff. S. levium |
| | | | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Napora latissima |
| | | | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Orbiculiforma (?) heliotropica |
| | | | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Leugeo hexacubicus |
| | | | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Unuma typicus |
| | | | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | | Parvicingula dhimenaensis ssp. A. |
| | | | | | | | |■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | | Tricolocapsa plicarum plicarum |
| | | | | | | | |■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | | Tricolocapsa (?) fusiformis |
| | | | | | | | |■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | | Parvicingula dhimenaensis |
| | | | | | | | |■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | Stichocapsa japonica |
| | | | | | | | |■|■|■|■|■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | Tricolocapsa plicarum s. l. |
| | | | | | | | |■|■|■|■|■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | Eucyrtidiellum unumaense |
| | | | | | | | | |■| | | | | | | | | | | | | | | | | | | | | | | | | | | | Cyrtocapsa mastoidea |
| | | | | | | | | |■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | Stichocapsa convexa |
| | | | | | | | | |■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | Tricolocapsa sp. S |
| | | | | | | | | |■|■|■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | Unuma sp. A |
| | | | | | | | | |■|■|■|■|■|■|■|■|■|■|■| | | | | | | | | | | | | | | | | | Tricolocapsa (?) aff. T. fusiformis |
| | | | | | | | | | |■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | Stichocapsa sp. E |
| | | | | | | | | | |■|■|■| | | | | | | | | | | | | | | | | | | | | | | | | Stichocapsa sp. F |
| | | | | | | | | | |■|■|■|■| | | | | | | | | | | | | | | | | | | | | | | | Eucyrtidiellum semifactum |
| | | | | | | | | | |■|■|■|■|■|■| | | | | | | | | | | | | | | | | | | | | | Tricolocapsa plicarum ssp. A |
| | | | | | | | | | |■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■|■| | | | | | | | | | Williriedellum sp. A |

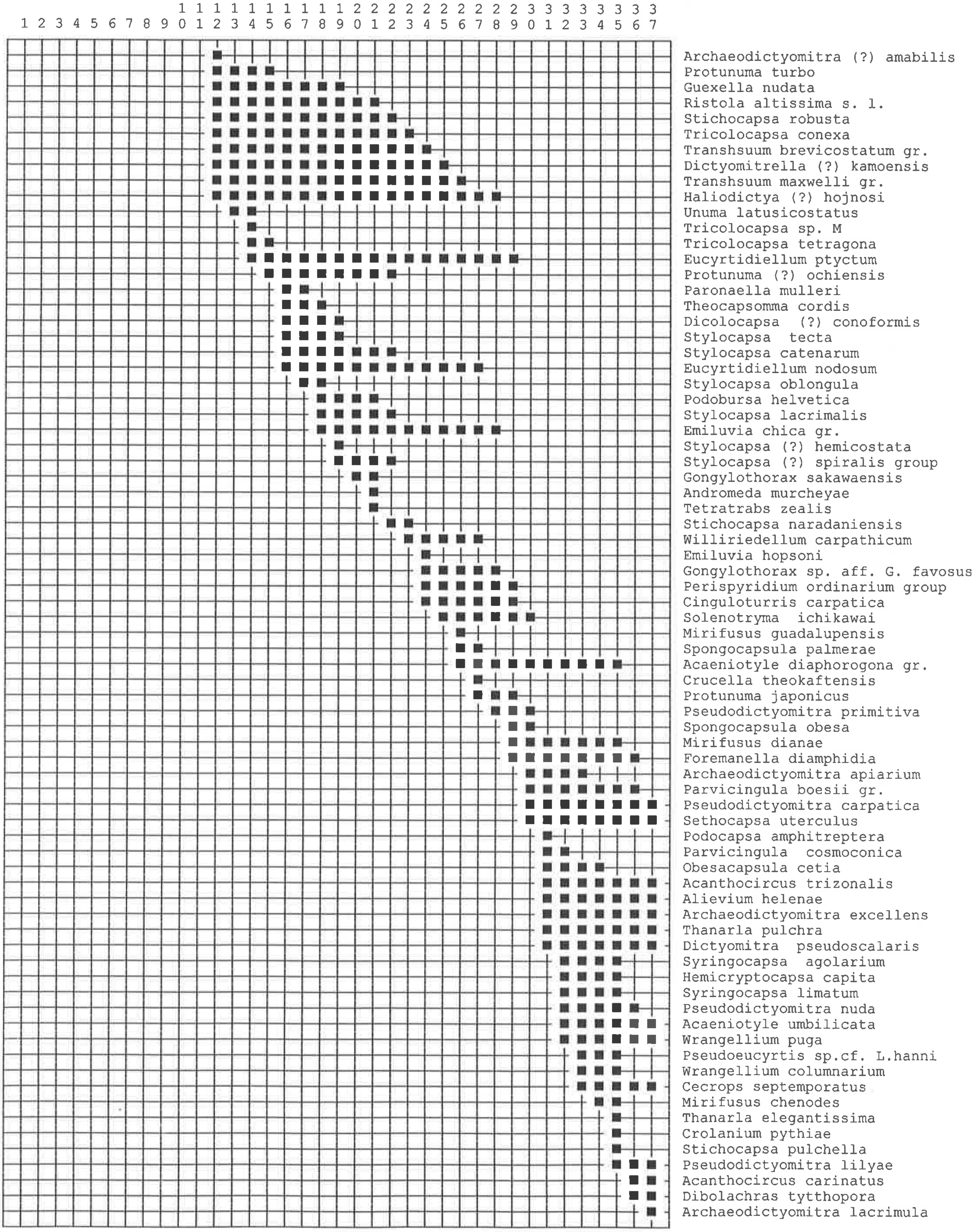
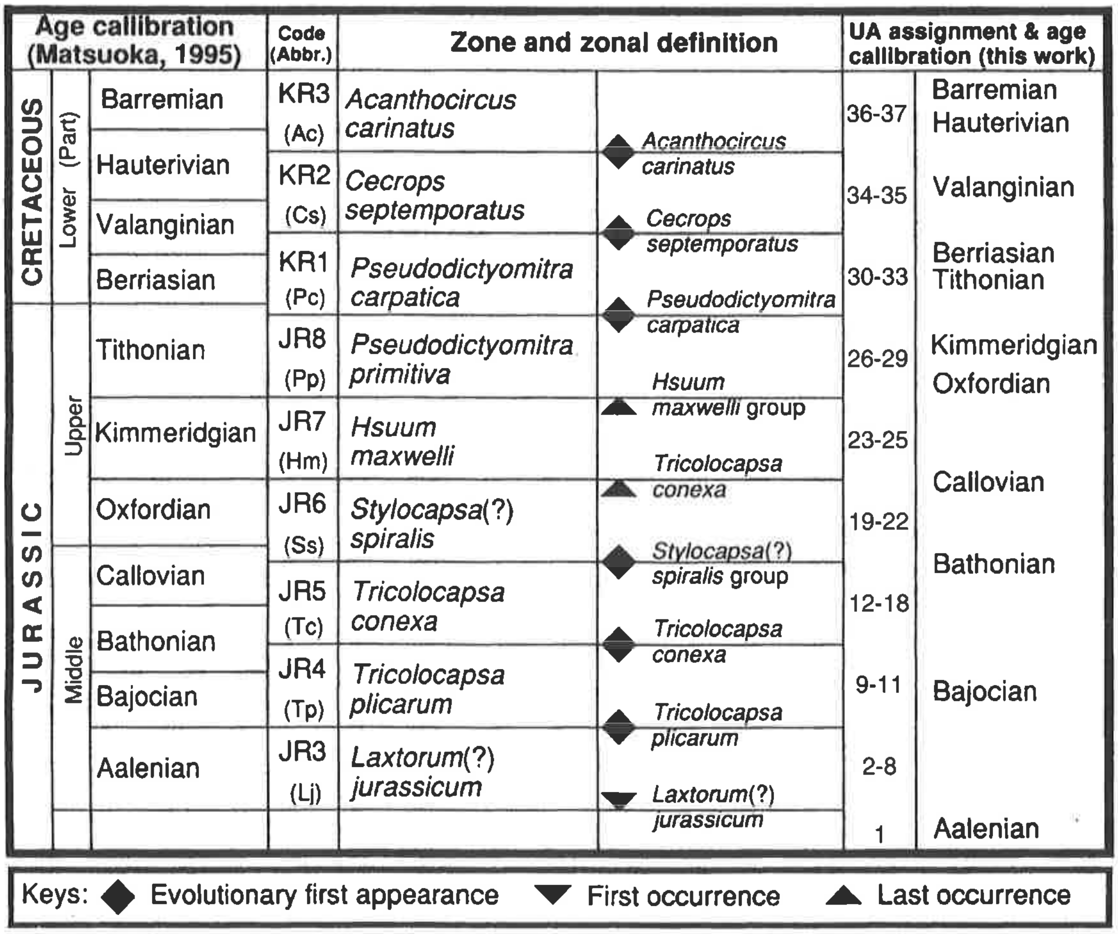


Fig. 17 Range chart showing the distribution of the Jurassic to Lower Cretaceous radiolarians in Japan. Constructed after the data published by Matusoka (1995).



| Age callibration (Matsuoka, 1995) | | | Code (Abbr.) | Zone and zonal definition | |
|---|---|---|---|---|---|
| CRETACEOUS | Lower (Part) | Barremian | KR3 (Ac) | *Acanthocircus carinatus* | ◆ *Acanthocircus carinatus* |
| | | Hauterivian | KR2 (Cs) | *Cecrops septemporatus* | ◆ *Cecrops septemporatus* |
| | | Valanginian | | | |
| | | Berriasian | KR1 (Pc) | *Pseudodictyomitra carpatica* | ◆ *Pseudodictyomitra carpatica* |
| JURASSIC | Upper | Tithonian | JR8 (Pp) | *Pseudodictyomitra primitiva* | ▲ *Hsuum maxwelli* group |
| | | Kimmeridgian | JR7 (Hm) | *Hsuum maxwelli* | ▲ *Tricolocapsa conexa* |
| | | Oxfordian | JR6 (Ss) | *Stylocapsa*(?) *spiralis* | ◆ *Stylocapsa*(?) *spiralis* group |
| | Middle | Callovian | JR5 (Tc) | *Tricolocapsa conexa* | ◆ *Tricolocapsa conexa* |
| | | Bathonian | | | |
| | | Bajocian | JR4 (Tp) | *Tricolocapsa plicarum* | ◆ *Tricolocapsa plicarum* |
| | | Aalenian | JR3 (Lj) | *Laxtorum*(?) *jurassicum* | ▼ *Laxtorum*(?) *jurassicum* |

| UA assignment & age callibration (this work) | |
|---|---|
| 36-37 | Barremian<br>Hauterivian |
| 34-35 | Valanginian |
| 30-33 | Berriasian<br>Tithonian |
| 26-29 | Kimmeridgian<br>Oxfordian |
| 23-25 | Callovian |
| 19-22 | Bathonian |
| 12-18 | |
| 9-11 | Bajocian |
| 2-8 | |
| 1 | Aalenian |

Keys: ◆ Evolutionary first appearance ▼ First occurrence ▲ Last occurrence

Fig. 18 Correlation between the 37 UAs given in Fig.17 and Matsuoka's (1995) Interval Zones

## 4.7 Phylogenetic seriations and phylozones

Lacking precise information on the superpositional control between different fauna, some paleontologists who work on sparse and stratigraphically disconnected populations are forced to infer sequences of species (or assemblages of species) from an estimate of the relative degree of evolution of the individuals belonging to the species (see Thaler 1972 and Godinot 1981 for the theoretical grounds of this methodology).

The unitary association method can be quite useful in establishing phylogenetic seriations of such disconnected fossil assemblages. We simply replace the notion of *biostratigraphic graph* with that of *phylogenetic graph* in which the arcs represent the (hypothetical) relationships *primitive→ advanced* (instead of representing true superpositions of species) and we proceed as follows.

First we start by coding each lineage as 1, 2, 3 etc. Then we code each species in one single lineage as *x*1, *x*2, *x*3 etc. The final taxonomic coding consists of lists of taxa 1*x*1, 1*x*2, 1*x*3,...2*x*1, 2*x*2, 2*x*3 etc resulting from the coupling of the lineage codes and of the species codes within each lineage.

Each evolutionary sequence is recorded in a single stratigraphic section coded under the "samples" format (see Sect. 3.2.2.): level 1 contains 1*x*1, level 2 contains 1*x*2, level 3 contains 1*x*3 etc. Coding all the lineages this way is equivalent to constructing an oriented graph representing the phylogenetic sequences.

Then each assemblage is considered as a single section composed of a single level. For example level 1 of section n contains species 1*x*3, 2*x*3, 5*x*4 etc (note that these levels will never contain the co-occurrence of two distinct evolutionary stages belonging to the same lineage; e.g. 1*x*1 and 1*x*2).

The resulting phylogenetic graph has the same properties as a biostratigraphic graph (see Sect. 2.2.1) and can be treated by our program.

Detecting and interpreting the contradictions that can result from applying such phylogenetic seriations can also be done by applying the tools described above. Our experience (Guex unpublished results) shows that there are always intermediate populations which are virtually connecting two distinct evolutionary stages within one lineage. In other words some more advanced forms can well coexist chronologically with more primitive forms within a single lineage.

We finally note that the present method can also be applied to archaeological seriations of artefacts when true stratigraphic information is missing (but see also Blackham 1998).

## 4.8 Taxonomy and paleobiology in ammonoids biochronology: sexual dimorphism, covariation and septal spacing

Since the accuracy of biochronological correlations is mostly subordinate to the quality of taxonomy, as mentioned in an earlier work "*False identification equals false measurement*" (Guex 1977), we discuss briefly a few problems of ammonoid paleobiology which we consider to be important in their taxonomic consequences.

Constructing phylogenies is a basic step to establishing the taxonomy of most fossil groups but it is particularly important in ammonoids. To establish phylogenies in this group requires the recognition of two fundamental phenomena: sexual dimorphism and covariation.

Firstly, it is well known that microconch ammonoids have a truncated ontogeny (*progenetic*) in comparison with their macroconch counterparts. From this we conclude that microconchs had a precocious sexual maturity and were thus the male of the ammonites, by analogy with the extant Dibranchiates (Guex 1970). Introducing an ancestral microconch form into a peramorphic macroconch lineage (e.g. the very common evolute → involute peramorphic transformation) leads naturally to interpreting the process as paedomorphosis, which is absurd. Here we will also note that the evolute → involute transformation is often global because it affects the whole ontogeny of the descendant. When this is the case it is also false to invoke paedomorphosis to describe that mode of transformation.

The second important factor in understanding phylogenies is the particular case of variability called *covariation.* First osberved by Buckman (1892) in *Sonninia* and *Amaltheus* and rediscussed later by Westermann (1966), covariation was originally described as follows: "Roughly speaking, inclusion and compression of the whorls correlate with the amount of ornament - the most ornate species being the more evolute and having almost circular whorls..." It is now known (Guex, unpublished results) that covariation depends of the internal shell geometry, namely the lateral and ventral curvature of the shell which controls the amount of morphogens present in the mantle (see below). The most salient ornamentation is developed where the worls are the most curved, angular shells being often spinose or carinate and flat ones being almost smooth. As a general rule, juvenile ammonites belonging to peramorphic lineages are more evolute (with a greater lateral curvature of the whorl) than the adult ones: this is why we observe so often ancestral spinose or coarsely ornate forms giving rise to involute "smooth" or weakly ornamented descendants.

Guex's empirical conclusions have recently been tested by Andre Koch within the conceptual framework of Meinhardt's reaction - diffusion models (1995). Koch simulated the distribution of morphogens in a quadrangular body chamber and demonstrated that morphogens maxima are located in the parts of the mantle situated in the angular parts of the shell. These results will be detailed in a forthcoming paper by Koch, Guex, O'Dogherty and Bucher. It is worth mentioning that the phenomenon of *ornamental compensation* (= post-traumatic disappearance of an ornamentation and its replacement by the adjacent ornamentation) observed in pathological ammonites (Guex 1967) is a strong argument in favour of the application of Meinhardt's models to the morphogenesis of our favourite fossil group.

Another interesting problem indirectly related to the taxonomy of ammonites concerns the significance of septal spacing. It is now known (Guex, unpublished results) that septal spacing is basically a function of the sutural complexity and of its lateral amplitude on one side and of the increasing weight of the animal on the other (obvious necessity to compensate the loss of buoyancy). Consequently we consider that septal spacing is not a reliable indicator of the growth rate of the organism; this is contrary to a common held and false belief which assumes that closely spaced septa mean slow growth and widely spaced septa mean rapid growth. This point is particularly important to the paleobiological and evolutionary interpretation of the Liparoceratids which are ancestral to the Amaltheids mentioned above and which are also the most important groups for the biochronology of the middle Lias. These results will be detailed in a forthcoming paper by Guex, Bucher, Carruzzo and Chirat.

## 4.9 Calibrating UAs with the numerical time scale

Calibrating UAs with the numerical time scale is often very difficult. This is mainly due to the scarcity of reliable tie-points between biochronological units and radiometric measurements or paleomagnetic "datings".

We can illustrate this point by showing that many species' sequences are erroneously calibrated. This is best demonstrated by examining classical "age-depth" bivariate graphs because most such diagrams show conflicting chronological relationships of the type given in Fig. 19, thus throwing light on two main kinds of contradictions.

- Top-Bottom relationships (Fig. 19A)

The worst case of contradictory relationship is when the "top" (=last occurrence "datum" or LAD etc) of a taxon $X$ ($T_X$) is numerically older but stratigraphically higher than the "bottom" of a taxon $Y$ ($B_Y$) (i.e. $X$ and $Y$ co-occur stratigraphically whereas they ought to be in sequence according to their respective numerical age). This kind of situation appears to be very common when numerical ages of successive last and first occurrences of species are tested against good biostratigraphic evidence by means of "age-depth" graphs. In such cases it is clearly the calibrations of the datums with the numerical ages which are false and not the stratigraphic observations which are biased. All pairs of numerical ages demonstrated to be false by good stratigraphic data should be eliminated from the lists of numerical ages compiled in the literature. To our knowledge this has never been done. We finally note that this very frequent kind of basic contradiction has never been discussed in the literature or has been merely attributed to the general diachroneity of "datums" without drawing the necessary conclusions concerning those erroneous numerical ages.

- Bottom-bottom and top-top relationships (Fig. 19B, C)

The most common contradictory relationship is when the bottom (resp.top) of a given taxon is numerically older but stratigraphically higher than the bottom (resp. top) of another taxon. This kind of situation is generated by the general and common diachroneity of the so-called "datums" (first and last occurrences of taxa). The majority of these diachronisms are now known to be generated by discontinuous vertical record (mostly fossilization and less often by biotic factors).

An actual example (DSDP leg 90, site 586B) is given in Fig. 20. Most of the DSDP/IPOD sites with an accurate fossil record show the same kind of contradictions when the datums relationships (top-bottom, bottom-bottom and top-top) are plotted against their supposed numerical age.

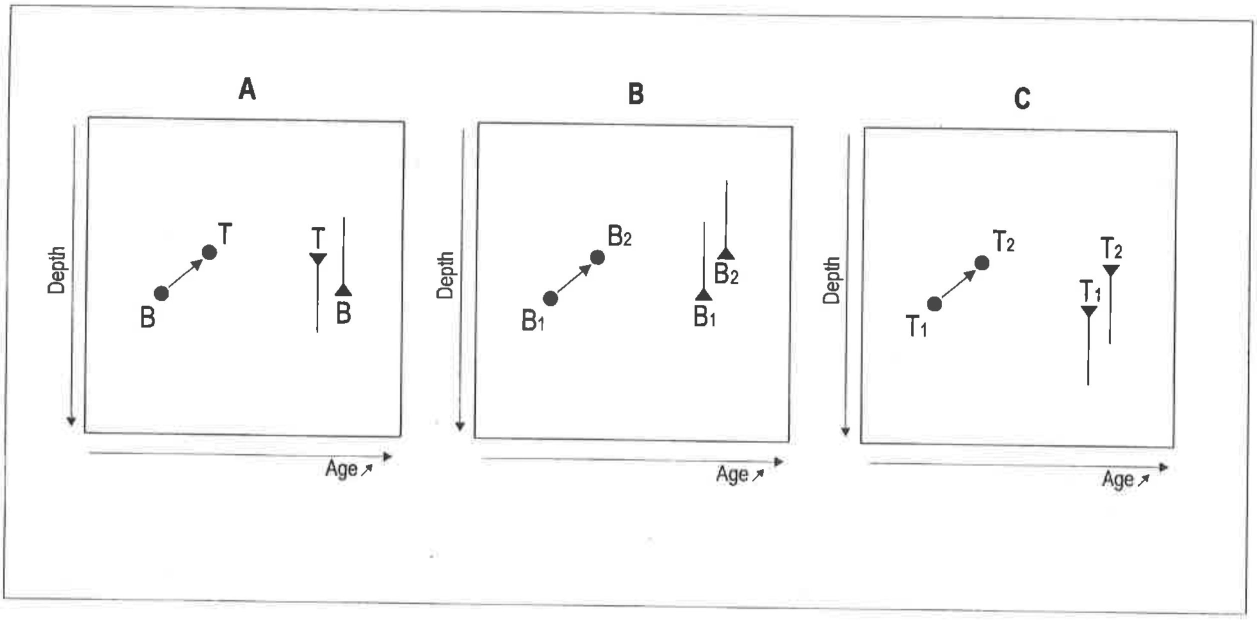


Fig.19 Different kind of contradictions between numerical ages assigned to "datums" and the stratigraphic relationships which are observed in the sedimentary sequences

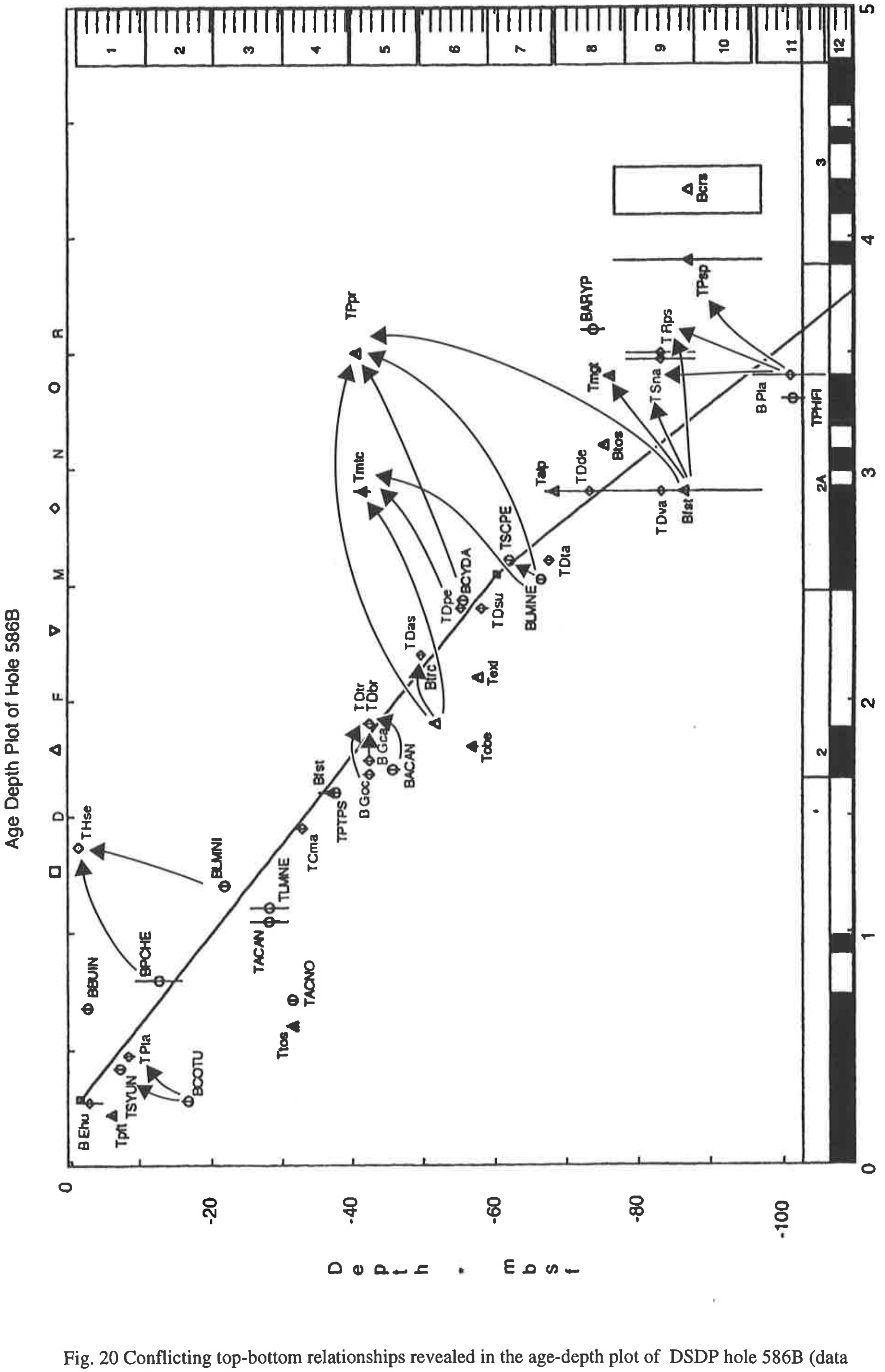


Fig. 20 Conflicting top-bottom relationships revealed in the age-depth plot of DSDP hole 586B (data from Lazarus 1995)

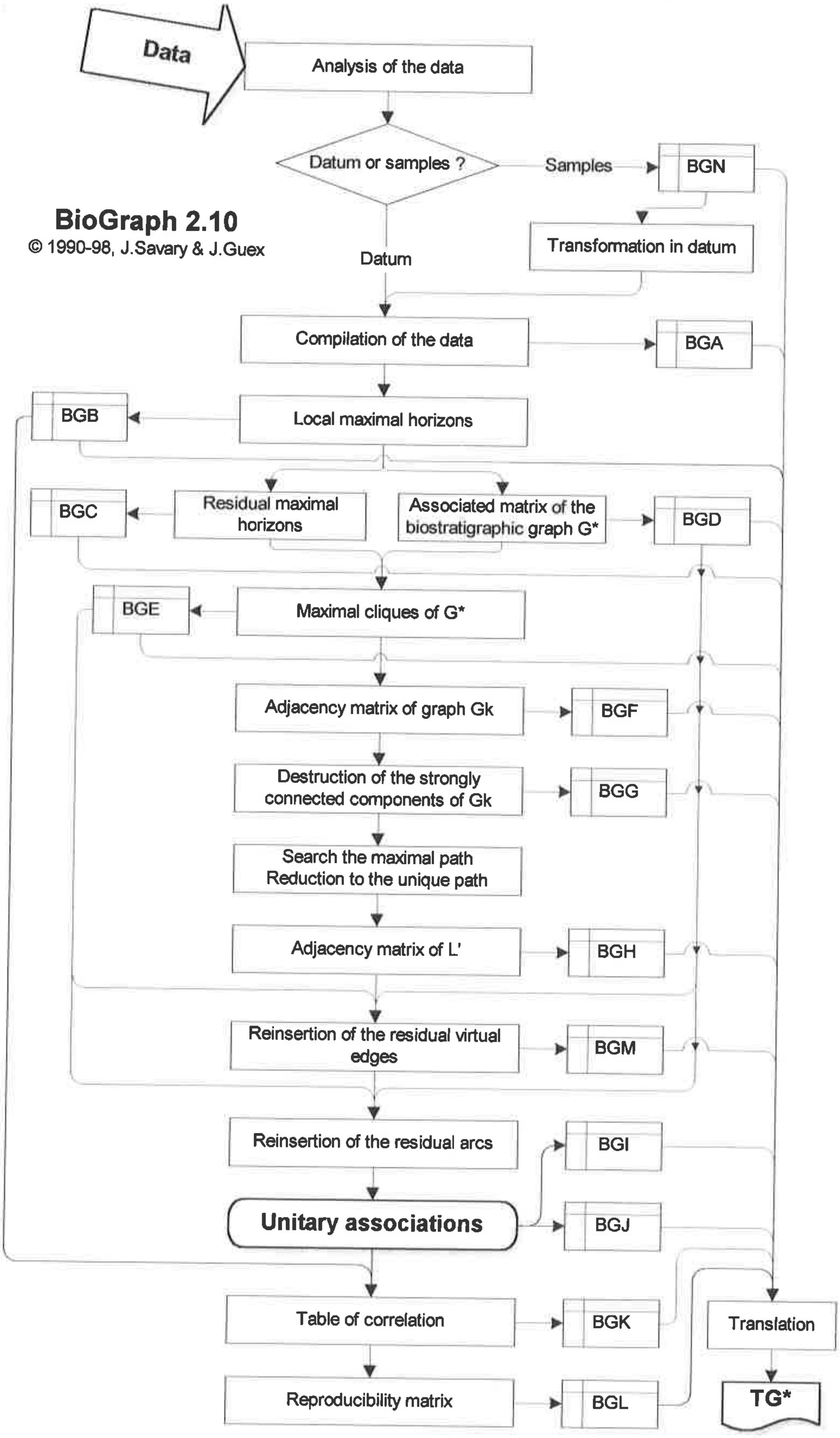


Fig.21 Flow chart of the BioGraph program

# 5 The BioGraph Program

## 5.1 Architecture of the program (Fig. 21)

The program was developed on IBM-PC and written in two different languages: Modula-2 (compiler Top Speed Modula-2 by Jensen and Partners) and Turbo Pascal (Borland International). To combine the two languages and to minimize the size of the code, BioGraph is segmented into several programs and files articulated around BG_EXE which is written in Turbo Pascal. That program verifies the presence of necessary files and it manages the calls and answers of the different elements. The communication is established by means of a memory sector that is reserved and accessible to all the other programs.

The following programs are invoked:

- BG_MENU.EXE (Turbo Pascal) is used as an interface with the user. It is the interactive menu described in section 3. The selections chosen by the user are recorded in BG_CFG.
- BG_DATA.EXE (Turbo Pascal) analyses the syntax and the accuracy of the datafile and translates it into a binary file.
- BG_GRAP.EXE (Modula-2) analyses the graph and records all the intermediate and final results in binary files.
- BG_CONV.EXE (Turbo Pascal) translates the chosen binary files generated by BG_GRAP into text files.
- BG_DRAW.EXE (Turbo Pascal) represents the graphs $G^*$, $G_k$ and $G_k'$ and is designed to make a didactic handling of these graphs which can then be printed on a graphic printer or plotter.

## 5.2 Main program (Modula 2)

### *Program BG_GRAP*

Function : analysis and interpretation of the configuration of the data

```
MODULE BG_GRAP;

IMPORT IO;

FROM Lib     IMPORT IncAddr, DecAddr, AddAddr, HSort, WordMove, WordFill;
FROM BG_ERR  IMPORT Error;
FROM BG_MEM  IMPORT HEAD_REC, HEAD_PTR, CORR_REC, CORR_PTR, WORD_PTR, INT_PTR,
                    SET_PTR, WORD_LIST, SetRange, GetRange, New, Release,
                    SetSegment, GetSegment, PointSet, SwapSet, ClearMemory,
                    Union, Difference, Cardinality, DisposeHeap,
                    SizeOfDiagMat, DiagIndex, Intersection, Min, Max, Arc,
                    Foreward, Backward, BioStrat, SetArc, SetMat, ArcCounting,
                    DepthFirstSearch, PutInSet, BestFit, Subtraction,
                    AU_Sorting, WithSetDo, MemAvail, EDGE;
FROM BG_CRT  IMPORT Gauge, ClearGauge, ElapsedTime, Message, Nb_Message,
                    St_Message, Static_Message;
FROM BG_FIL  IMPORT EXTENSION, Load_Levels, Save_Levels, Save_Clique,
                     Save_Matrix, Load_Matrix, Create_Connex, Save_Connex,
                     Save_DestroyedArc, Close_Connex, Save_AU, Save_Corr,
                     Save_Report, _REPORT;

VAR TAXA, NSEC, NLEV : CARDINAL;
    MAT_A, MAT_H     : INT_PTR;
```

## Grouping of non maximal cliques

```
PROCEDURE SortByCard(a, b : CARDINAL) : BOOLEAN;
VAR ptr1, ptr2 : HEAD_PTR;
BEGIN
   ptr1:=PointSet(a);
   ptr2:=PointSet(b);
   RETURN ptr1^.Cardinal>ptr2^.Cardinal
END SortByCard;

PROCEDURE Compaction(Taxa, NbLevel : CARDINAL) : CARDINAL;
VAR L, W, Levels, OldLevels, Range : CARDINAL;
    Segment                        : ADDRESS;
    Ptr1, Ptr2                     : SET_PTR;
BEGIN
  IF 1<NbLevel THEN
      ClearGauge;
      Levels:=NbLevel;
      OldLevels:=Levels-1;
      Segment:=GetSegment();
      Range:=((GetRange()-SIZE(HEAD_REC)) >> 1) - 1;
      HSort(Levels, SortByCard, SwapSet);
      REPEAT
         FOR L:=2 TO Levels DO
            W:=0;
            Ptr1:=PointSet(1);
            Ptr2:=PointSet(L);
            WHILE (W<=Range) AND ((Ptr2^.Bit[W]-Ptr1^.Bit[W])={}) DO INC(W) END;
            IF Range<W THEN
               Ptr2^.H.Cardinal:=0;
               DEC(Levels);
               DEC(NbLevel)
            END
         END;
         DEC(Levels);
         SetSegment(PointSet(2));
         HSort(OldLevels, SortByCard, SwapSet);
         OldLevels:=Levels-1;
         Gauge(FLOAT(NbLevel-Levels)/FLOAT(NbLevel))
      UNTIL Levels<2;
      Gauge(1.0);
      SetSegment(Segment)
   END;
   RETURN NbLevel
END Compaction;
```

## Search of local maximal horizons

```
PROCEDURE Superposition(a, b : CARDINAL) : BOOLEAN;
VAR ptr1, ptr2 : HEAD_PTR;
BEGIN
   ptr1:=PointSet(a);
   ptr2:=PointSet(b);
   IF ptr1^.Cardinal=0 THEN
      RETURN FALSE
   ELSIF ptr2^.Cardinal=0 THEN
      RETURN TRUE
   ELSIF ptr1^.Section<ptr2^.Section THEN
      RETURN TRUE
   ELSIF ptr1^.Section>ptr2^.Section THEN
      RETURN FALSE
   ELSE
      RETURN ptr1^.Level>ptr2^.Level
   END
END Superposition;

PROCEDURE AllReduction(Taxa, NSec, NLev : CARDINAL) : CARDINAL;
VAR Section, Nb_Level, NewLevels, NewNLev : CARDINAL;
    Segment                               : ADDRESS;
    ptr, l1, l2                           : HEAD_PTR;
BEGIN
   Message(1);
   NewNLev:=0;
   Segment:=GetSegment();
   ptr:=Segment;
   FOR Section:=1 TO NSec DO
      Nb_Level:=0;
      l1:=ptr;
      LOOP
         l2:=AddAddr(l1, GetRange());
         INC(Nb_Level);
         IF l1^.Section<>l2^.Section THEN EXIT END;
         l1:=l2
      END;
      Nb_Message(2, Section);
      Nb_Message(3, Nb_Level);
      NewLevels:=Compaction(Taxa, Nb_Level);
      Nb_Message(4, NewLevels);
      Gauge(FLOAT(Section)/FLOAT(NSec));
      INC(NewNLev, NewLevels);
      IncAddr(ptr, Nb_Level*GetRange());
      SetSegment(ptr)
   END;
   Nb_Message(5, NewNLev);
   SetSegment(Segment);
   HSort(NLev, Superposition, SwapSet);
   RETURN NewNLev
END AllReduction;
```

## Constructing the adjacency matrix of the biostratigraphic graph

```
PROCEDURE MakeMatrix_A(FLoad : EXTENSION;
                       Taxa  : CARDINAL;
                   VAR Mat   : INT_PTR);
VAR mat                          : INT_PTR;
    Level, Arc, Edge             : SET_PTR;
    NSec, S, NLev, L, Lmax, T, t : CARDINAL;
BEGIN
   New(Mat, SizeOfDiagMat(Taxa));
   ClearMemory(Mat, SizeOfDiagMat(Taxa));
   New(Edge, LONGCARD(GetRange()));
   New(Arc,  LONGCARD(GetRange()));
   Load_Levels(FLoad, Taxa, NSec, NLev);
   Message(8);
   Message(9);
   ClearGauge;
   Level:=PointSet(NLev+1);
   Level^.H.Section:=0;
   Lmax:=0;
   REPEAT
      Level:=PointSet(Lmax+1);
      S:=Level^.H.Section;
      SetSegment(Level);
```

search of number of maximal horizons contained in the section

```
      Lmax:=0;
      REPEAT
         INC(Lmax);
         Level:=PointSet(Lmax+1)
      UNTIL Level^.H.Section<>S;

      FOR T:=1 TO Taxa DO
```

Search  of the presence of the species

```
         Level:=PointSet(Lmax);
         L:=Lmax;
         WHILE (0<L) AND NOT (T-1 IN Level^.Set) DO
            DEC(L);
            Level:=PointSet(L)
         END;
```

If species is present

```
         IF 0<L THEN
            ClearMemory(Edge, LONGCARD(GetRange()));
            ClearMemory(Arc , LONGCARD(GetRange()));
```

Union of the  horizons containing the species

```
            REPEAT
               Edge^.H.Cardinal:=WithSetDo(Union, Edge, Level, Edge);
               DEC(L);
               Level:=PointSet(L)
            UNTIL (L=0) OR NOT (T-1 IN Level^.Set);
```

Union of the upper horizons

```
            WHILE 0<L DO
               Arc^.H.Cardinal:=WithSetDo(Union, Arc, Level, Arc);
               DEC(L);
               Level:=PointSet(L)
            END;
```

Filling the matrix

```
            FOR t:=1 TO Taxa DO
               IF T<>t THEN
                  IF t-1 IN Edge^.Set THEN
                     mat:=DiagIndex(Mat, T, t, Taxa);
                     mat^:=EDGE
                  ELSIF t-1 IN Arc^.Set THEN
                     mat:=DiagIndex(Mat, T, t, Taxa);
                     IF mat^<>EDGE THEN
                        IF T<t THEN
                           IF 0<mat^ THEN
                              mat^:=EDGE
                           ELSE
                              DEC(mat^)
                           END
                        ELSIF mat^<0 THEN
                           mat^:=EDGE
                        ELSE
                           INC(mat^)
                        END
                     END
                  END
               END
            END
         END;
         Gauge(FLOAT((S-1)*Taxa+T)/FLOAT(NSec*Taxa))
      END
   UNTIL S=NSec;
```

Conservation of the matrix A in the memory

```
   Release(Edge)
END MakeMatrix_A;
```

## Union of the compatible cliques

Construction of the neighborhood of each species, including the species itself

```
PROCEDURE Neighbours(VAR FirstN : ADDRESS;
                         Mat_A  : INT_PTR;
                         Taxa   : CARDINAL);
VAR t1, t2 : CARDINAL;
    Near   : SET_PTR;
    Mat    : INT_PTR;
BEGIN
   New(FirstN, LONGCARD(Taxa*GetRange()));
   ClearMemory(FirstN, LONGCARD(Taxa*GetRange()));
   FOR t1:=1 TO Taxa-1 DO
      FOR t2:=t1+1 TO Taxa DO
         Mat:=DiagIndex(Mat_A, t1, t2, Taxa);
         IF Mat^=EDGE THEN
            Near:=PointSet(t1);
            PutInSet(t2, Near);
            Near:=PointSet(t2);
            PutInSet(t1, Near)
         END
      END
   END;
   FOR t1:=1 TO Taxa DO
      Near:=PointSet(t1);
      PutInSet(t1, Near)
   END
END Neighbours;
```

### Union by means of the neighborhoods

```
PROCEDURE Unification(FLoad, FSave, FClique : EXTENSION;
                      Taxa, NLev            : CARDINAL;
                      Mat_A                 : INT_PTR) : CARDINAL;
VAR NSec, Level, t : CARDINAL;
    Clique, Scrap  : SET_PTR;
    Levels, FirstN : ADDRESS;
BEGIN
   ClearGauge;
   Message(10);
   New(Scrap, LONGCARD(GetRange()));
   Neighbours(FirstN, Mat_A, Taxa);
   Load_Levels(FLoad, Taxa, NSec, NLev);
   Levels:=GetSegment();
   Level:=1;
   REPEAT
      Gauge(FLOAT(Level)/FLOAT(NLev));
      SetSegment(Levels);
      Clique:=PointSet(Level);
      SetSegment(FirstN);
      FOR t:=1 TO Taxa DO
         IF NOT (t-1 IN Clique^.Set) THEN
            IF WithSetDo(Subtraction, Clique, PointSet(t), Scrap)=0 THEN
               PutInSet(t, Clique);
               INCL(BITSET(Clique^.H.Section), 15)
            END
         END
      END;
      INC(Level)
   UNTIL NLev<Level;
   Gauge(1.0);
   SetSegment(Levels);
   NLev:=Compaction(Taxa, NLev);
   Nb_Message(11, NLev);
   Save_Levels(FSave, Taxa, NSec, NLev);
   Save_Clique(FClique, Taxa, NLev);
   Release(Scrap);
   RETURN NLev
END Unification;
```

### Construction of the graph of the MRH (not of the cliques)

```
PROCEDURE HMRonly(FLoad, FSave, FClique : EXTENSION;
                  Taxa, NLev            : CARDINAL) : CARDINAL;
VAR NSec : CARDINAL;
BEGIN
   Load_Levels(FLoad, Taxa, NSec, NLev);
   Save_Levels(FSave, Taxa, NSec, NLev);
   Save_Clique(FClique, Taxa, NLev);
   Release(GetSegment());
   RETURN NLev
END HMRonly;
```

### Construction of the adjacency matrix of the cliques graph

```
PROCEDURE MakeMatrix_H(FLoad      : EXTENSION;
                       Taxa, NLev : CARDINAL;
                       Mat_A      : INT_PTR;
                   VAR Mat_H      : INT_PTR);
VAR matA, matH                   : INT_PTR;
    NSec, L1, L2, N1, n1, N2, n2 : CARDINAL;
    Level, Arc                   : ADDRESS;
    arc, Above, Below            : INTEGER;
    Arc1, Arc2                   : WORD_LIST;
BEGIN
   New(Mat_H, SizeOfDiagMat(NLev));
   New(Arc1 , LONGCARD(Taxa*SIZE(CARDINAL)));
   New(Arc2 , LONGCARD(Taxa*SIZE(CARDINAL)));
   Load_Levels(FLoad, Taxa, NSec, NLev);
   Message(8);
   Message(12);
   Message(13);
   ClearGauge;
   _REPORT.Contras:=0;
   FOR L1:=1 TO NLev-1 DO
      Level:=PointSet(L1);
      FOR L2:=L1+1 TO NLev DO
         Difference(Level, PointSet(L2), Arc1, Arc2, N1, N2);
         Below:=0;
         Above:=0;
         FOR n1:=0 TO N1-1 DO
            FOR n2:=0 TO N2-1 DO
               matA:=DiagIndex(Mat_A, Arc1^[n1], Arc2^[n2], Taxa);
               IF (matA^<>0) AND (matA^<>EDGE) THEN
                  IF Arc1^[n1]<Arc2^[n2] THEN
                     arc:=matA^
                  ELSE
                     arc:=-matA^
                  END;
                  ArcCounting(arc, Below, Above)
               END
            END
         END;
```

computing the uncertainty coefficient:
- bit 15 :
  $c_i < 0$ : below⇒ above
  $0 < c_i$ : below ⇐ above
- bit 14 : no link
- bit 13 : virtual arc in the transitive closure
- bit 12 : certitude, ration replace by
  Max(Below, Above)
- after exclusion of bit 15 :
  0 : no contradiction
  $0 < |c_i| \leq 1000$ : contradiction

```
         matH:=DiagIndex(Mat_H, L1, L2, NLev);
         Above:=ABS(Above);
         Below:=ABS(Below);
         IF (Above=0) AND (Below=0) THEN
            matH^:=1000
         ELSE
            matH^:=TRUNC(1000.0*FLOAT(Min(Above, Below))
                               /FLOAT(Max(Above, Below)))
         END;
         IF matH^=0 THEN
            matH^:=Min(Max(Above, Below), 1000);
            INCL(BITSET(matH^), 12)
         ELSE
            INC(_REPORT.Contras)
         END;
         IF matH^=1000 THEN
            matH^:=4000H
         ELSIF Above<Below THEN
            INCL(BITSET(matH^), 15)
         END;
         Static_Message(_REPORT.Contras)
      END;
      Gauge(FLOAT(L1)/FLOAT(NLev-1))
   END
END MakeMatrix_H;
```

## Destruction of the strongly connected components

Method not used for the small components

```
PROCEDURE Connexe(FSave      : EXTENSION;
                  Mat        : ADDRESS;
                  Nb_Vertex  : CARDINAL);
VAR Vertex, Destroyed, v1, v2, vr1, vr2, vf1, vf2 : CARDINAL;
    Factor, Reprod, Arc                           : INTEGER;
    Conn, Succ, Pred, Memo                        : SET_PTR;
    mat                                           : INT_PTR;
    Count                                         : BOOLEAN;
BEGIN
   ClearGauge;
   Message(14);
   SetRange(Nb_Vertex);
   New(Succ, LONGCARD(GetRange()));
   New(Pred, LONGCARD(GetRange()));
   New(Memo, LONGCARD(GetRange()));
   New(Conn, LONGCARD(GetRange()));
   ClearMemory(Memo, LONGCARD(GetRange()));
   SetMat(Mat, Nb_Vertex);
   Create_Connex(FSave);
   Count:=TRUE;
   _REPORT.Connex :=0;
   _REPORT.Destroy:=0;
   Vertex:=1;
   REPEAT
      DepthFirstSearch(Foreward, Vertex, Succ);
      DepthFirstSearch(Backward, Vertex, Pred);
      Conn^.H.Cardinal:=WithSetDo(Intersection, Succ, Pred, Conn);
      IF Conn^.H.Cardinal<3 THEN
         INC(Vertex);
         Count:=TRUE;
         Gauge(FLOAT(Vertex)/FLOAT(Nb_Vertex-1))
      ELSE
         IF Count THEN
            INC(_REPORT.Connex);
            Memo^.H.Cardinal:=WithSetDo(Union, Conn, Memo, Memo);
            Destroyed:=0;
            Count:=FALSE;
            Nb_Message(15, _REPORT.Connex);
            Nb_Message(16, Conn^.H.Cardinal);
            Message(17);
            Conn^.H.Section:=0;
            Save_Connex(Conn)
         END;
         Factor:=0;
         Reprod:=MAX(INTEGER);
         FOR v1:=Vertex TO Nb_Vertex-1 DO
            IF v1-1 IN Conn^.Set THEN
               FOR v2:=v1+1 TO Nb_Vertex DO
                  IF v2-1 IN Conn^.Set THEN
                     mat:=DiagIndex(Mat, v1, v2, Nb_Vertex);
                     Arc:=mat^;
                     EXCL(BITSET(Arc), 15);
                     IF NOT (14 IN BITSET(Arc)) THEN
```

Arc with the lowest reproducibility

```
                        IF 12 IN BITSET(Arc) THEN
                           EXCL(BITSET(Arc), 12);
                           IF Arc<Reprod THEN
                              Reprod:=Arc;
                              vr1:=v1;
                              vr2:=v2
                          END
```

The most conflictual arc

```
                        ELSIF Factor<Arc THEN
                           Factor:=Arc;
                           vf1:=v1;
                           vf2:=v2
                        END
                     END
                  END
               END
            END
         END;
         IF 0<Factor THEN
            v1:=vf1;
            v2:=vf2
         ELSE
            v1:=vr1;
            v2:=vr2
         END;
         mat:=DiagIndex(Mat, v1, v2, Nb_Vertex);
         INCL(BITSET(mat^), 14);
         INC(_REPORT.Destroy);
         INC(Destroyed);
         Static_Message(Destroyed);
         Save_DestroyedArc(v1, v2)
      END
   UNTIL Nb_Vertex-1<=Vertex;
   IF 0<_REPORT.Connex THEN
      Nb_Message(18, _REPORT.Destroy)
   ELSE
      Message(19)
   END;
   _REPORT.Vertex:=Memo^.H.Cardinal;
   Close_Connex(Nb_Vertex, _REPORT.Connex, _REPORT.Destroy);
   Release(Succ)
END Connexe;
```

Method used for the large components

```
PROCEDURE Connexe(FSave     : EXTENSION;
                  Mat       : ADDRESS;
                  Nb_Vertex : CARDINAL);
VAR Vertex, Destroyed, v1, v2 : CARDINAL;
    Arc, a, b                 : INTEGER;
    Conn, Succ, Pred, Memo    : SET_PTR;
    mat                       : INT_PTR;
    Count, ok                 : BOOLEAN;
    Reprod, Contra            : ARRAY[0..1000] OF CARDINAL;
BEGIN
   ClearGauge;
   Message(14);
   SetRange(Nb_Vertex);
   New(Succ, LONGCARD(GetRange()));
   New(Pred, LONGCARD(GetRange()));
   New(Memo, LONGCARD(GetRange()));
   New(Conn, LONGCARD(GetRange()));
   ClearMemory(Memo, LONGCARD(GetRange()));
   ClearMemory(Conn, LONGCARD(GetRange()));
   SetMat(Mat, Nb_Vertex);
```

Histogram of the values of the contradictions and of the reproducibility

```
   WordFill(ADR(Reprod), 1001, 0);
   WordFill(ADR(Contra), 1001, 0);
   FOR v1:=1 TO Nb_Vertex-1 DO
      FOR v2:=v1+1 TO Nb_Vertex DO
         mat:=DiagIndex(Mat, v1, v2, Nb_Vertex);
         IF NOT (14 IN BITSET(mat^)) THEN
            Arc:=mat^;
            EXCL(BITSET(Arc), 15);
            IF 12 IN BITSET(Arc) THEN
               EXCL(BITSET(Arc), 12);
               INC(Reprod[Arc])
            ELSE
               INC(Contra[Arc])
            END
         END
      END
   END;

   Create_Connex(FSave);
   Count:=TRUE;
   _REPORT.Connex :=0;
   _REPORT.Destroy:=0;
   Vertex:=1;
   REPEAT
      DepthFirstSearch(Foreward, Vertex, Succ);
      DepthFirstSearch(Backward, Vertex, Pred);
      Conn^.H.Cardinal:=WithSetDo(Intersection, Succ, Pred, Conn);
      IF Conn^.H.Cardinal<3 THEN
         INC(Vertex);
         Count:=TRUE;
         Gauge(FLOAT(Vertex)/FLOAT(Nb_Vertex-1))
      ELSE
         IF Count THEN
            INC(_REPORT.Connex);
            Memo^.H.Cardinal:=WithSetDo(Union, Conn, Memo, Memo);
            Destroyed:=0;
            a:=1001;
            b:=-1;
            Count:=FALSE;
            Nb_Message(15, _REPORT.Connex);
            Nb_Message(16, Conn^.H.Cardinal);
            Message(17);
            Conn^.H.Section:=0;
            Save_Connex(Conn)
         END;
         ok:=TRUE;
         WHILE ok AND (0<a) DO
            DEC(a);
            IF 0<Contra[a] THEN
               FOR v1:=Vertex TO Nb_Vertex-1 DO
                  IF v1-1 IN Conn^.Set THEN
                     FOR v2:=v1+1 TO Nb_Vertex DO
                        IF v2-1 IN Conn^.Set THEN
                           mat:=DiagIndex(Mat, v1, v2, Nb_Vertex);
                           IF BITSET(mat^)*BITSET{12,14}=BITSET{} THEN
```

### The most conflicting arc

```
                              Arc:=mat^;
                              EXCL(BITSET(Arc), 15);
                              IF Arc=a THEN
                                 INCL(BITSET(mat^), 14);
                                 INC(_REPORT.Destroy);
                                 INC(Destroyed);
                                 DEC(Contra[a]);
                                 ok:=FALSE;
                                 Static_Message(Destroyed);
                                 Save_DestroyedArc(v1, v2)
                              END
                           END
                        END
                     END
                  END
               END
            END
         END;
         IF ok THEN
            WHILE (b<1000) AND ok DO
               INC(b);
               IF 0<Reprod[b] THEN
                  FOR v1:=Vertex TO Nb_Vertex-1 DO
                     IF v1-1 IN Conn^.Set THEN
                        FOR v2:=v1+1 TO Nb_Vertex DO
                           IF v2-1 IN Conn^.Set THEN
                              mat:=DiagIndex(Mat, v1, v2, Nb_Vertex);
                              IF BITSET(mat^)*BITSET{12,14}=BITSET{12} THEN
```

### The least reproducible arc

```
                                 Arc:=mat^;
                                 EXCL(BITSET(Arc), 12);
                                 EXCL(BITSET(Arc), 15);
                                 IF Arc=b THEN
                                    INCL(BITSET(mat^), 14);
                                    INC(_REPORT.Destroy);
                                    INC(Destroyed);
                                    DEC(Reprod[b]);
                                    ok:=FALSE;
                                    Static_Message(Destroyed);
                                    Save_DestroyedArc(v1, v2)
                                 END
                              END
                           END
                        END
                     END
                  END
               END
            END
         END
      END
   UNTIL Nb_Vertex-1<=Vertex;
   IF 0<_REPORT.Connex THEN
      Nb_Message(18, _REPORT.Destroy)
   ELSE
      Message(19)
   END;
   _REPORT.Vertex:=Memo^.H.Cardinal;
   Close_Connex(Nb_Vertex, _REPORT.Connex, _REPORT.Destroy);
   Release(Succ)
END Connexe;
```

## Search of the maximal path and destruction of the parallel paths

### Search of the successors and predecessors of each vertex
### Search of one maximal path
### Search of the extremities of the maximal path

```
PROCEDURE SearchPathSet(PathSet          : SET_PTR;
                        Depth            : ADDRESS;
                        Number_Of_Vertex : CARDINAL;
                    VAR Head, Tail       : CARDINAL);
VAR MaxS, MaxP, Vertex : CARDINAL;
    Succ, Pred         : SET_PTR;
BEGIN
```

```
   ClearGauge;
   SetSegment(Depth);
   MaxS:=0;
   MaxP:=0;
   FOR Vertex:=1 TO Number_Of_Vertex DO
      Succ:=PointSet(2*Vertex-1);
      Pred:=PointSet(2*Vertex);
      DepthFirstSearch(Foreward, Vertex, Succ);
      DepthFirstSearch(Backward, Vertex, Pred);
      IF (Pred^.H.Cardinal=1) AND (MaxS<Succ^.H.Cardinal) THEN
         MaxS:=Succ^.H.Cardinal;
         Head:=Vertex
      ELSIF (Succ^.H.Cardinal=1) AND (MaxP<Pred^.H.Cardinal) THEN
         MaxP:=Pred^.H.Cardinal;
         Tail:=Vertex
      END;
      Gauge(FLOAT(Vertex)/FLOAT(Number_Of_Vertex))
   END;
   Succ:=PointSet(2*Head-1);
   Pred:=PointSet(2*Tail);
```

Verification of the unicity of the path

```
   IF NOT (Tail-1 IN Succ^.Set) THEN
      Message(21);
      Tail:=0;
      REPEAT
         INC(Tail);
         Pred:=PointSet(2*Tail)
      UNTIL (Pred^.H.Cardinal=MaxP) AND (Head-1 IN Pred^.Set)
   END;
   PathSet^.H.Cardinal:=WithSetDo(Intersection, Succ, Pred, PathSet)
END SearchPathSet;
```

Search of  maximal path

```
PROCEDURE SearchPathMax(PathSet                      : SET_PTR;
                        Depth                        : ADDRESS;
                        Number_Of_Vertex, Head, Tail : CARDINAL;
                        PathMax                      : WORD_LIST);
VAR SuccH, SuccT, Clique : SET_PTR;
    MaxS, Len, v         : CARDINAL;
BEGIN
   New(SuccH, LONGCARD(GetRange()));
   New(SuccT, LONGCARD(GetRange()));
   SetSegment(Depth);
   Len:=0;
   PathMax^[Len]:=Head;
   REPEAT
      Clique:=PointSet(2*Head-1);
      SuccH^.H.Cardinal:=WithSetDo(Intersection, Clique, PathSet, SuccH);
      EXCL(PathSet^.Set, Head-1);
      MaxS:=0;
      FOR v:=1 TO Number_Of_Vertex DO
         IF v-1 IN PathSet^.Set THEN
            Clique:=PointSet(2*v-1);
            SuccT^.H.Cardinal:=WithSetDo(Intersection, Clique, PathSet, SuccT);
            IF (SuccT^.H.Cardinal<SuccH^.H.Cardinal) AND
               (MaxS<SuccT^.H.Cardinal) THEN
               MaxS:=SuccT^.H.Cardinal;
               Head:=v
            END
         END
      END;
      INC(Len);
      PathMax^[Len]:=Head
   UNTIL Head=Tail;
   Release(SuccH);
```

Construction of the set restricted to the maximal path

```
   Nb_Message(22, Len+1);
   ClearMemory(PathSet, LONGCARD(GetRange()));
   FOR v:=0 TO Len DO PutInSet(PathMax^[v], PathSet) END
END SearchPathMax;

PROCEDURE MaximumPath(FLoad                   : EXTENSION;
                      Mat                     : ADDRESS;
                  VAR Taxa, Number_Of_Vertex : CARDINAL);
VAR PathMax          : WORD_LIST;
```

```
    PathSet, Cliques : SET_PTR;
    Depth            : ADDRESS;
    Head, Tail       : CARDINAL;

   PROCEDURE MakeBestFit;
   VAR Clique, Succ, Pred, Authorized : SET_PTR;
       List                           : ADDRESS;
       v                              : CARDINAL;
   BEGIN
      New(Authorized, LONGCARD(GetRange()));
      New(List, LONGCARD(Number_Of_Vertex)*SIZE(CARDINAL));
```

Ordering of the cliques of the path

```
      SetSegment(Cliques);
      SetRange(Taxa);
      FOR v:=1 TO PathSet^.H.Cardinal DO
         Clique:=PointSet(PathMax^[v-1]);
         Clique^.H.Section:=0;
         Clique^.H.Level:=v
      END;
```

Best fit

```
      ClearGauge;
      FOR v:=1 TO Number_Of_Vertex DO
         IF NOT (v-1 IN PathSet^.Set) THEN
```

Clique to be united by best-fit: determining the set of known relationships for which the fit is forbidden

```
            SetSegment(Depth);
            SetRange(Number_Of_Vertex);
            Succ:=PointSet(2*v-1);
            Pred:=PointSet(2*v);
            Authorized^.H.Cardinal:=WithSetDo(Union, Succ, Pred, Authorized);
            Authorized^.H.Cardinal:=WithSetDo(Subtraction, PathSet, Authorized,
            Authorized);
```

Marking the clique to be fitted (united) into another clique

```
            SetSegment(Cliques);
            SetRange(Taxa);
            Clique:=PointSet(v);
            Clique^.H.Cardinal:=0;
            BestFit(Clique, Authorized, Number_Of_Vertex, List)
         END;
         Gauge(FLOAT(v)/FLOAT(Number_Of_Vertex))
      END
   END MakeBestFit;

BEGIN
   Load_Levels(FLoad, Taxa, Head, Tail);
   Cliques:=GetSegment();
   IF Number_Of_Vertex<2 THEN
      Cliques^.H.Section:=0;
      Cliques^.H.Level:=1;
      RETURN
   END;
   Message(20);
   SetRange(Number_Of_Vertex);
```

Set containing the maximal path

```
   New(PathSet, LONGCARD(GetRange()));
```

Maximal path

```
   New(PathMax, LONGCARD(Number_Of_Vertex)*SIZE(CARDINAL));
```

Set of the successors and predecessors of each vertex

```
   New(Depth, 2*LONGCARD(Number_Of_Vertex)*LONGCARD(GetRange()));
   SetMat(Mat, Number_Of_Vertex);
   SearchPathSet(PathSet, Depth, Number_Of_Vertex, Head, Tail);
   SearchPathMax(PathSet, Depth, Number_Of_Vertex, Head, Tail, PathMax);
   Message(23);
   MakeBestFit;
   HSort(Number_Of_Vertex, Superposition, SwapSet);
   Number_Of_Vertex:=PathSet^.H.Cardinal;
   Release(PathSet)
END MaximumPath;
```

## Consecutive ones

```
PROCEDURE Continuity(Taxa, Number : CARDINAL);
VAR t, Below, Above, c : CARDINAL;
    Clique             : SET_PTR;
BEGIN
   ClearGauge;
   Message(24);
   FOR t:=0 TO Taxa-1 DO
      Below:=0;
      REPEAT
         INC(Below);
         Clique:=PointSet(Below);
      UNTIL (t IN Clique^.Set) OR (Below=Number);
      IF (t IN Clique^.Set) AND (Below<Number) THEN
         Above:=Number+1;
         REPEAT
            DEC(Above);
            Clique:=PointSet(Above)
         UNTIL (t IN Clique^.Set) OR (Above=Below);
         FOR c:=Below+1 TO Above DO
            Clique:=PointSet(c);
            IF NOT (t IN Clique^.Set) THEN PutInSet(t+1, Clique) END
         END
      END;
      Gauge(FLOAT(t)/FLOAT(Taxa-1))
   END
END Continuity;
```

## Insertion of the residual virtual edges

```
PROCEDURE InsertEdge(FLoad1, FLoad2, FSave : EXTENSION;
                 VAR Taxa, Nb_Of_AU        : CARDINAL;
                 VAR Mat_A                 : INT_PTR);
VAR AU, Corr   : ADDRESS;
    Set1, Set2 : SET_PTR;
    Mat_B      : INT_PTR;
    k          : CARDINAL;

   PROCEDURE Detect_ReverseArc(Edge, Arc : SET_PTR) : BOOLEAN;
   VAR t1, t2, n  : CARDINAL;
       arcA, arcB : INTEGER;
       Clique     : SET_PTR;
       mat        : INT_PTR;
   BEGIN
      n:=0;
      Nb_Message(26, n);
      ClearGauge;
      FOR t1:=1 TO Taxa DO
```

### Search of the FAD of $t_1$

```
         k:=Nb_Of_AU+1;
         REPEAT
            DEC(k);
            Clique:=PointSet(k)
         UNTIL (k=0) OR (t1-1 IN Clique^.Set);
         IF 0<k THEN
            ClearMemory(Edge, LONGCARD(GetRange()));
            ClearMemory(Arc , LONGCARD(GetRange()));
```

### Search of the existence interval of $t_1$

```
            REPEAT
               Edge^.H.Cardinal:=WithSetDo(Union, Edge, Clique, Edge);
               DEC(k);
               Clique:=PointSet(k)
            UNTIL (k=0) OR NOT (t1-1 IN Clique^.Set);
            IF 0<k THEN
```

### Arcs of $t_1$

```
               REPEAT
                  Arc^.H.Cardinal:=WithSetDo(Union, Arc, Clique, Arc);
               DEC(k);
               Clique:=PointSet(k)
            UNTIL k=0;
            Arc^.H.Cardinal:=WithSetDo(Subtraction, Arc, Edge, Arc);

            FOR t2:=1 TO Taxa DO
               IF t2-1 IN Arc^.Set THEN
```

Direction of arc $t_1 \Rightarrow t_2$

```
                  mat:=DiagIndex(Mat_A, t1, t2, Taxa);
                  arcA:=mat^;
                  IF arcA=EDGE THEN
                     IF t1<t2 THEN
                        arcB:=-4000H
                     ELSE
                        arcB:= 4000H
                     END
                  ELSIF t1<t2 THEN
                     arcB:=-ABS(mat^)
                  ELSE
                     arcB:=ABS(mat^)
                  END;
```

Storage and elimination of the inverted arc

```
                  IF arcA<>arcB THEN
                     mat^:=0;
                     mat:=DiagIndex(Mat_B, t1, t2, Taxa);
                     mat^:=arcB;
                     INC(n);
                     Static_Message(n)
                  END
               END
            END
         END
      END;
      Gauge(FLOAT(t1)/FLOAT(Taxa))
   END;
   _REPORT.Edge:=n;
   RETURN 0<n
END Detect_ReverseArc;

PROCEDURE Correction(T, FAD, LAD : CARDINAL);
VAR corr : HEAD_PTR;
BEGIN
   corr:=AddAddr(Corr, (T-1)*SIZE(HEAD_REC));
   IF corr^.Section<FAD THEN corr^.Section:=FAD END;
   IF (corr^.Level>LAD) OR (corr^.Level=0) THEN corr^.Level:=LAD END
END Correction;

PROCEDURE Insert_VirtualEdge(postX, anteY : SET_PTR);
VAR t1, t2, X, Y, kX, kY, vX, vY, T1, T2 : CARDINAL;
    mat                                  : INT_PTR;
    CliqueX, CliqueY, auX, auY           : SET_PTR;
    corr                                 : HEAD_PTR;
BEGIN
   Message(27);
   ClearGauge;
   New(auX, LONGCARD(GetRange()));
   New(auY, LONGCARD(GetRange()));
   SetSegment(AU);
   FOR t1:=1 TO Taxa-1 DO
      FOR t2:=t1+1 TO Taxa DO
         mat:=DiagIndex(Mat_B, t1, t2, Taxa);
         IF mat^<>0 THEN
```

Direction of arc X ⇒ Y

```
            IF mat^<0 THEN
               X:=t1;
               Y:=t2
            ELSE
               X:=t2;
               Y:=t1
            END;
```

Search  of the LAD of X

```
            kX:=0;
```

```
      REPEAT
        INC(kX);
        CliqueX:=PointSet(kX)
      UNTIL X-1 IN CliqueX^.Set;
```

### Search of FAD of Y

```
      kY:=Nb_Of_AU+1;
      REPEAT
        DEC(kY);
        CliqueY:=PointSet(kY)
      UNTIL Y-1 IN CliqueY^.Set;
```

### Interval between $k_X$ and $k_Y$

```
      ClearMemory(postX, LONGCARD(GetRange()));
      FOR k:=kY+1 TO kX-1 DO
        postX^.H.Cardinal:=WithSetDo(Union, postX, PointSet(k), postX)
      END;
      WordMove(postX, anteY, GetRange() >> 1);
      WordMove(CliqueX, auX, GetRange() >> 1);
      WordMove(CliqueY, auY, GetRange() >> 1);
```

### Successors of X with $k_Y$ and predecessors of Y with $k_X$

```
      postX^.H.Cardinal:=WithSetDo(Union, postX, CliqueY, postX);
      anteY^.H.Cardinal:=WithSetDo(Union, anteY, CliqueX, anteY);
```

### Set restricted to the non common parts

```
      auX^.H.Cardinal:=WithSetDo(Subtraction, auX, postX, auX);
      auY^.H.Cardinal:=WithSetDo(Subtraction, auY, anteY, auY);
      postX^.H.Cardinal:=WithSetDo(Subtraction, postX, CliqueX, postX);
      anteY^.H.Cardinal:=WithSetDo(Subtraction, anteY, CliqueY, anteY);
```

### Injection criteria

```
      vX:=0;
      vY:=0;
      FOR T1:=1 TO Taxa DO
        IF (T1-1 IN auX^.Set) OR (T1-1 IN auY^.Set) THEN
          FOR T2:=1 TO Taxa DO
            mat:=DiagIndex(Mat_A, T1, T2, Taxa);
            IF (T1-1 IN auX^.Set) AND (T2-1 IN postX^.Set) THEN
              IF mat^<>0 THEN INC(vX, ABS(mat^)+1) END
            END;
            IF (T1-1 IN auY^.Set) AND (T2-1 IN anteY^.Set) THEN
              IF mat^<>0 THEN INC(vY, ABS(mat^)+1) END
            END
          END
        END
      END;
```

### Record of the insertions

```
      IF vX=vY THEN
        k:=(kX+kY) DIV 2;
        Correction(X, kX-1, k);
        Correction(Y, k, kY+1)
      ELSIF vX<vY THEN
        Correction(X, kX-1, kY)
      ELSE
        Correction(Y, kX, kY+1)
      END
    END
  END;
  Gauge(FLOAT(t1)/FLOAT(Taxa))
END;
Release(auX);
```

### Insertion of the virtual edges

```
corr:=Corr;
FOR t1:=1 TO Taxa DO
  IF 0<corr^.Section THEN
    FOR k:=corr^.Level TO corr^.Section DO
      CliqueX:=PointSet(k);
      PutInSet(t1, CliqueX)
```

```
            END
         END;
         IncAddr(corr, SIZE(HEAD_REC))
      END
   END Insert_VirtualEdge;

BEGIN
   Load_Levels(FLoad1, Taxa, k, Nb_Of_AU);
   AU:=GetSegment();
   Load_Matrix(FLoad2, Mat_A, Taxa);
   New(Mat_B, SizeOfDiagMat(Taxa));
   ClearMemory(Mat_B, SizeOfDiagMat(Taxa));
   New(Corr, LONGCARD(Taxa)*SIZE(HEAD_REC));
   ClearMemory(Corr, LONGCARD(Taxa)*SIZE(HEAD_REC));
   New(Set1, LONGCARD(GetRange()));
   New(Set2, LONGCARD(GetRange()));
   SetSegment(AU);
   Message(25);
   IF Detect_ReverseArc(Set1, Set2) THEN
      Insert_VirtualEdge(Set1, Set2)
   END;
   Save_Matrix(FSave, Mat_B, Taxa);
```

Reloading of Mat_A for Split

```
   Release(Mat_A);
   Load_Matrix(FLoad2, Mat_A, Taxa);
   SetSegment(AU)
END InsertEdge;
```

## Indexing the UAs and diverse manipulations

```
PROCEDURE Make_AU(AU : CARDINAL);
VAR k      : CARDINAL;
    Clique : HEAD_PTR;
BEGIN
   HSort(AU, Superposition, SwapSet);
   Clique:=GetSegment();
   FOR k:=1 TO AU DO
      Clique:=PointSet(k);
      Clique^.Level:=AU-k+1
   END
END Make_AU;

PROCEDURE SplitTrick(Taxa         : CARDINAL;
                 VAR Number_Of_AU : CARDINAL;
                     Mat_A        : INT_PTR);
VAR Clique1, Clique2           : ADDRESS;
    Kp, K1, K2, Ks, Kl, Kf, Ko : SET_PTR;
    k1, k2, tl, tf             : CARDINAL;
    Split, dont                : BOOLEAN;
    Mat                        : INT_PTR;
BEGIN
   Message(28);
   Clique1:=GetSegment();
```

Duplicating the cliques

```
   New(Clique2, LONGCARD((2*Number_Of_AU+2)*GetRange()));
   FOR k1:=1 TO Number_Of_AU DO
      SetSegment(Clique1);
      Kp:=PointSet(k1);
      SetSegment(Clique2);
      k2:=2*k1+1;
      K1:=PointSet(k2-1);
      K2:=PointSet(k2);
      WordMove(Kp, K1, GetRange() >> 1);
      WordMove(Kp, K2, GetRange() >> 1);
      K1^.H.Level:=2*(Number_Of_AU-k1+1)+1;
      K2^.H.Level:=2*(Number_Of_AU-k1+1)
   END;
   K1:=PointSet(1);
   ClearMemory(K1, LONGCARD(GetRange()));
   K1^.H.Level:=2*(Number_Of_AU+1);
   K1:=PointSet(2*(Number_Of_AU+1));
   ClearMemory(K1, LONGCARD(GetRange()));
   K1^.H.Level:=1;

   New(Kl, LONGCARD(GetRange()));
   New(Kf, LONGCARD(GetRange()));
```

### Memorising the modified taxons

```
   New(Ko, LONGCARD(GetRange()));
   SetSegment(Clique2);
   SetMat(Mat_A, Taxa);
   SetArc(BioStrat);
   FOR k1:=1 TO Number_Of_AU DO
      k2:=2*k1+1;
      Ks:=PointSet(k2-2);
      K2:=PointSet(k2-1);
      Kl:=PointSet(k2  );
      Kp:=PointSet(k2+1);
      Kl^.H.Cardinal:=WithSetDo(Subtraction, K2, Ks, Kl);
      Kf^.H.Cardinal:=WithSetDo(Subtraction, Kl, Kp, Kf);
      ClearMemory(Ko, LONGCARD(GetRange()));
      Split:=FALSE;
      FOR tl:=1 TO Taxa DO
         IF tl-1 IN Kl^.Set THEN
            FOR tf:=1 TO Taxa DO
               IF (tf-1 IN Kf^.Set) AND (tl<>tf) THEN
                  IF Arc(tl, tf) THEN
                     Split:=TRUE;
                     IF NOT (tf-1 IN Ko^.Set) THEN
                        INCL(Ko^.Set, tf-1);
                        EXCL(Kl^.Set, tf-1);
                        DEC(Kl^.H.Cardinal)
                     END;
                     IF NOT (tl-1 IN Ko^.Set) THEN
                        INCL(Ko^.Set, tl-1);
                        EXCL(K2^.Set, tl-1);
                        DEC(K2^.H.Cardinal)
                     END
                  END
               END
            END
         END
      END;
```

### Verifying the perenity of the edges

```
      IF Split THEN
         Kl^.H.Cardinal:=WithSetDo(Subtraction, Kl, K2, Kl);
         Kf^.H.Cardinal:=WithSetDo(Subtraction, K2, Kl, Kf);
         FOR tl:=1 TO Taxa DO
            IF tl-1 IN Kl^.Set THEN
               FOR tf:=1 TO Taxa DO
                  IF tf-1 IN Kf^.Set THEN
                     Mat:=DiagIndex(Mat_A, tl, tf, Taxa);
                     IF Mat^=EDGE THEN
                        dont:=TRUE
                     ELSIF tl<tf THEN
                        dont:=Mat^>0
                     ELSE
                        dont:=Mat^<0
                     END;
                     IF dont THEN
                        IF NOT (tf-1 IN Kl^.Set) THEN
                           INCL(Kl^.Set, tf-1);
                           INC(Kl^.H.Cardinal)
                        END;
                        IF NOT (tl-1 IN K2^.Set) THEN
                           INCL(K2^.Set, tl-1);
                           INC(K2^.H.Cardinal)
                        END
                     END
                  END
               END
            END
         END
      END
   END;
   Release(Kl);
   Number_Of_AU:=2*Number_Of_AU+1
END SplitTrick;

PROCEDURE SwapAU(T1, T2 : CARDINAL);
VAR Taxon1, Taxon2 : HEAD_PTR;
    Buffer         : HEAD_REC;
BEGIN
   Taxon1:=AddAddr(GetSegment(), (T1-1)*SIZE(HEAD_REC));
   Taxon2:=AddAddr(GetSegment(), (T2-1)*SIZE(HEAD_REC));
   Buffer:=Taxon1^;
   Taxon1^:=Taxon2^;
   Taxon2^:=Buffer
END SwapAU;

PROCEDURE Sort_AU(Taxa, AU : CARDINAL);
VAR k, t   : CARDINAL;
    Clique : SET_PTR;
    Taxon  : HEAD_PTR;
    Base   : ADDRESS;
BEGIN
   Clique:=GetSegment();
   New(Base, LONGCARD(Taxa*SIZE(HEAD_REC)));
   Taxon:=Base;
   SetSegment(Clique);
   FOR t:=0 TO Taxa-1 DO
      Taxon^.Section:=t+1;
      k:=0;
      REPEAT
         INC(k);
         Clique:=PointSet(k)
      UNTIL (t IN Clique^.Set) OR (k=AU);
      IF t IN Clique^.Set THEN
         Taxon^.Cardinal:=Clique^.H.Level
      ELSE
         Taxon^.Cardinal:=AU+1
      END;
      k:=AU+1;
      REPEAT
         DEC(k);
         Clique:=PointSet(k)
      UNTIL (t IN Clique^.Set) OR (k=1);
      IF t IN Clique^.Set THEN
         Taxon^.Level:=Clique^.H.Level
      ELSE
         Taxon^.Level:=AU+1
      END;
      IncAddr(Taxon, SIZE(HEAD_REC))
   END;
   SetSegment(Base);
   HSort(Taxa, AU_Sorting, SwapAU)
END Sort_AU;
```

## Creating the correlation and reproducibility tables

```
PROCEDURE Correlation(FLoad, AU_File, CORR_File : EXTENSION;
                  VAR Taxa, NSec, NLev          : CARDINAL);
VAR Nb_Of_AU, l, k           : CARDINAL;
    AU, Levels, Reprod       : ADDRESS;
    Clique, Level, Test, rep : SET_PTR;
    Corr, corr               : CORR_PTR;
BEGIN
   Load_Levels(AU_File, Taxa, k, Nb_Of_AU);
   AU:=GetSegment();
   Load_Levels(FLoad, Taxa, NSec, NLev);
   Levels:=GetSegment();
   New(Corr, LONGCARD(NLev)*SIZE(CORR_REC));
   corr:=Corr;
   SetRange(NSec);
   New(Reprod, LONGCARD(GetRange())*LONGCARD(Nb_Of_AU));
   ClearMemory(Reprod, LONGCARD(GetRange())*LONGCARD(Nb_Of_AU));
   SetRange(Taxa);
   New(Test, LONGCARD(GetRange()));
   Message(29);
   ClearGauge;
   FOR l:=1 TO NLev DO
      SetSegment(Levels);
      Level:=PointSet(l);
      corr^.Section:=Level^.H.Section;
      corr^.Level:=Level^.H.Level;
      SetSegment(AU);
      k:=Nb_Of_AU+1;
      REPEAT
         DEC(k);
         Clique:=PointSet(k);
         Test^.H.Cardinal:=WithSetDo(Subtraction, Level, Clique, Test)
      UNTIL (Test^.H.Cardinal=0) OR (k=0);
```

This is just a useful protection because k=0 is impossible

```
      IF k=0 THEN
         corr^.Fau:=0;
         corr^.Lau:=0
      ELSE
         corr^.Fau:=Nb_Of_AU-k+1;
         REPEAT
            DEC(k);
            Clique:=PointSet(k);
            Test^.H.Cardinal:=WithSetDo(Subtraction, Level, Clique, Test)
         UNTIL (0<Test^.H.Cardinal) OR (k=0);
         corr^.Lau:=Nb_Of_AU-k;
         IF corr^.Fau=corr^.Lau THEN
            SetRange(NSec);
            SetSegment(Reprod);
            PutInSet(Level^.H.Section, PointSet(Nb_Of_AU-corr^.Lau+1));
            SetRange(Taxa)
         END
      END;
      IncAddr(corr, SIZE(CORR_REC));
      Gauge(FLOAT(1)/FLOAT(NLev))
   END;
   SetSegment(Corr);
   Save_Corr(CORR_File, NLev);
   SetRange(NSec);
   SetSegment(Reprod);
   FOR k:=1 TO Nb_Of_AU DO
      rep:=PointSet(k);
      rep^.H.Level:=Nb_Of_AU-k+1;
      rep^.H.Cardinal:=Cardinality(rep)
   END;
   Taxa:=NSec;
   NLev:=Nb_Of_AU
END Correlation;

BEGIN
   Load_Levels('BGA', TAXA, NSEC, NLEV);
   NLEV:=AllReduction(TAXA, NSEC, NLEV);
   Save_Levels('BGB', TAXA, NSEC, NLEV);
   Message(6);
   NLEV:=Compaction(TAXA, NLEV);
   Nb_Message(7, NLEV);
   _REPORT.HMR:=NLEV;
   Save_Levels('BGC', TAXA, NSEC, NLEV);
   DisposeHeap;

   MakeMatrix_A('BGB', TAXA, MAT_A);
   Save_Matrix('BGD', MAT_A, TAXA);
   NLEV:=Unification('BGC', 'BGE', 'BG0', TAXA, NLEV, MAT_A);
   _REPORT.Cliques:=NLEV;
   MakeMatrix_H('BGE', TAXA, NLEV, MAT_A, MAT_H);
   Save_Matrix('BGF', MAT_H, NLEV);
   Release(MAT_A);

   Load_Matrix('BGF', MAT_H, NLEV);
   Connexe('BGG', MAT_H, NLEV);
   MaximumPath('BGE', MAT_H, TAXA, NLEV);
   Save_Levels('BGH', TAXA, 0, NLEV);

   Continuity(TAXA, NLEV);
   NLEV:=Compaction(TAXA, NLEV);
   Make_AU(NLEV);
   Save_Levels('BG1', TAXA, 0, NLEV);
   DisposeHeap;

   InsertEdge('BG1', 'BGD', 'BGM', TAXA, NLEV, MAT_A);
   NLEV:=Compaction(TAXA, NLEV);
   Make_AU(NLEV);
   SplitTrick(TAXA, NLEV, MAT_A);
   NLEV:=Compaction(TAXA, NLEV);
   Make_AU(NLEV);
   Save_Levels('BGI', TAXA, 0, NLEV);
   Sort_AU(TAXA, NLEV);
   Save_AU('BGJ', TAXA, NLEV);
   _REPORT.AU:=NLEV;
   DisposeHeap;
   Correlation('BGA', 'BGI', 'BGK', TAXA, NSEC, NLEV);
   Save_Levels('BGL', TAXA, 0, NLEV);
   DisposeHeap;
   Message(30);
   ElapsedTime;
   Save_Report('REP');
   Nb_Message( 4, _REPORT.HMR    );
   Nb_Message(32, _REPORT.Cliques);
   Nb_Message(33, _REPORT.AU     );
   Message(35)
END BG_GRAP.
```

## Library BG_MEM

Function : management of the memory of BG_GRAP and of the set theoretical and matricial functions

```
DEFINITION MODULE BG_MEM;

FROM Lib IMPORT CompareProc;

   TYPE HEAD_REC   = RECORD
                        Section, Level, Cardinal : CARDINAL
                     END;
        HEAD_PTR   = POINTER TO HEAD_REC;
        SET_REC    = RECORD
                        H : HEAD_REC;
                        CASE : BOOLEAN OF
                        | TRUE  : Set : SET OF CARDINAL
                        | FALSE : Bit : ARRAY CARDINAL OF BITSET
                        END
                     END;
        SET_PTR    = POINTER TO SET_REC;
        CORR_REC   = RECORD
                        Section, Level, Fau, Lau : CARDINAL
                     END;
        CORR_PTR   = POINTER TO CORR_REC;
        WORD_PTR   = POINTER TO CARDINAL;
        WORD_LIST  = POINTER TO ARRAY CARDINAL OF CARDINAL;
        INT_PTR    = POINTER TO INTEGER;
        ARC_TYPE   = (Foreward, Backward, BioStrat);
        SET_PROC   = PROCEDURE(BITSET, BITSET) : BITSET;
        COUNT_PROC = PROCEDURE(INTEGER, VAR INTEGER, VAR INTEGER);

   CONST EDGE = MIN(INTEGER);

   VAR ArcCounting : COUNT_PROC;
       AU_Sorting  : CompareProc;

   PROCEDURE Min(min, max : INTEGER) : INTEGER;
   PROCEDURE Max(min, max : INTEGER) : INTEGER;
   PROCEDURE SetRange(Bits : CARDINAL);
   PROCEDURE GetRange() : CARDINAL;
   PROCEDURE SizeOfDiagMat(Size : CARDINAL) : LONGCARD;

   PROCEDURE SetSegment(HeapPointer : ADDRESS);
   PROCEDURE GetSegment() : ADDRESS;

   PROCEDURE DiagIndex(Mat        : ADDRESS;
                       i, j, Size : CARDINAL) : INT_PTR;

   PROCEDURE PointSet(Set : CARDINAL) : ADDRESS;
   PROCEDURE SwapSet(Set1, Set2 : CARDINAL);
   PROCEDURE PutInSet(Element : CARDINAL;
                      Set     : SET_PTR);
   PROCEDURE Union(Set1, Set2 : BITSET) : BITSET;
   PROCEDURE Intersection(Set1, Set2 : BITSET) : BITSET;
   PROCEDURE Subtraction(Set1, Set2 : BITSET) : BITSET;
   PROCEDURE WithSetDo(Do                  : SET_PROC;
                       Set_C, Set_B, Set_A : SET_PTR) : CARDINAL;
   PROCEDURE Difference(Set1, Set2   : SET_PTR;
                        Diff1, Diff2 : WORD_LIST;
                        VAR n1, n2   : CARDINAL);
   PROCEDURE Cardinality(Set : SET_PTR) : CARDINAL;

   PROCEDURE SetArcCounting(Method : CARDINAL);
   PROCEDURE Arc(i, j : CARDINAL) : BOOLEAN;
   PROCEDURE SetArc(Direction : ARC_TYPE);
   PROCEDURE DepthFirstSearch(Direction : ARC_TYPE;
                              Vertex    : CARDINAL;
                              Visited   : SET_PTR);
```

```
    PROCEDURE BestFit(Set1, Forbidden   : SET_PTR;
                      Number_Of_Vertex : CARDINAL;
                      List             : WORD_LIST);
    PROCEDURE SetMat(Mat  : ADDRESS;
                     Size : CARDINAL);
    PROCEDURE SetAUSorting(Method : CARDINAL);

    PROCEDURE New(VAR Ptr   : ADDRESS;
                      Count : LONGCARD);
    PROCEDURE Release(Ptr : ADDRESS);
    PROCEDURE ClearMemory(Ptr   : ADDRESS;
                          Count : LONGCARD);
    PROCEDURE DisposeHeap;
    PROCEDURE MemAvail() : LONGCARD;

END BG_MEM.

IMPLEMENTATION MODULE BG_MEM;

FROM Lib     IMPORT AddAddr, IncAddr, DecAddr, WordFill;
FROM Storage IMPORT HeapAllocate, HeapDeallocate, HeapTotalAvail, MainHeap;
FROM BG_ERR  IMPORT Error;

TYPE ARC_PROC = PROCEDURE(INTEGER) : BOOLEAN;

CONST HEAP_SIZE = 20;
```

SEGMENT : pointer on the list of the levels in current use

```
VAR _HEAP, _SET_RANGE    : CARDINAL;
    _HEAP_LIST           : ARRAY[0..HEAP_SIZE] OF RECORD
                               Ptr  : ADDRESS;
                               Size : CARDINAL
                           END;
    _MAT_ADDR, _SEGMENT : ADDRESS;
    _VERTEX             : CARDINAL;
    _ARC                : ARC_PROC;
```

## Utilities

```
PROCEDURE SwapCard(VAR A, B : CARDINAL);
VAR Buffer : CARDINAL;
BEGIN
   Buffer:=A;
   A:=B;
   B:=Buffer
END SwapCard;

PROCEDURE Min(min, max : INTEGER) : INTEGER;
BEGIN
   IF min<max THEN
      RETURN min
   ELSE
      RETURN max
   END
END Min;

PROCEDURE Max(min, max : INTEGER) : INTEGER;
BEGIN
   IF min<max THEN
      RETURN max
   ELSE
      RETURN min
   END
END Max;
```

## Occupation of memory

A set always consists of a heading followed by its binary elements.
Bits converted into bytes rounded to WORD

```
PROCEDURE SetRange(Bits : CARDINAL);
BEGIN
   _SET_RANGE:=2*(((Bits-1) >> 4)+1) + SIZE(HEAD_REC)
END SetRange;

PROCEDURE GetRange() : CARDINAL;
BEGIN
```

```
   RETURN _SET_RANGE
END GetRange;
```

That procedure expresses the size of the matrix in bytes et not in number of entries: this one is half of the number

```
PROCEDURE SizeOfDiagMat(Size : CARDINAL) : LONGCARD;
BEGIN
   RETURN LONGCARD(Size)*LONGCARD(Size-1)
END SizeOfDiagMat;
```

## Positionning the pointers

```
PROCEDURE SetSegment(Segment : ADDRESS);
BEGIN
   _SEGMENT:=Segment
END SetSegment;

PROCEDURE GetSegment() : ADDRESS;
BEGIN
   RETURN _SEGMENT
END GetSegment;

PROCEDURE PointSet(Set : CARDINAL) : ADDRESS;
VAR l : LONGCARD;
    p : ADDRESS;
BEGIN
   l:=LONGCARD(Set-1)*LONGCARD(_SET_RANGE);
   p:=_SEGMENT;
   WHILE MAX(CARDINAL)<l DO
      IncAddr(p, MAX(CARDINAL));
      DEC(l, MAX(CARDINAL))
   END;
   RETURN AddAddr(p, CARDINAL(l))
END PointSet;

PROCEDURE DiagIndex(Mat        : ADDRESS;
                    i, j, Size : CARDINAL) : INT_PTR;
VAR l : LONGCARD;
BEGIN
   IF j<i THEN SwapCard(i, j) END;
   l:=(LONGCARD(i-1)*LONGCARD(Size) + LONGCARD(j-1)
       - ((LONGCARD(i)*LONGCARD(i+1)) >> 1)) << 1;
   WHILE MAX(CARDINAL)<l DO
      IncAddr(Mat, MAX(CARDINAL));
      DEC(l, MAX(CARDINAL))
   END;
   RETURN AddAddr(Mat, CARDINAL(l))
END DiagIndex;
```

## Set theoretical operations

```
PROCEDURE SwapSet(Set1, Set2 : CARDINAL);
VAR Ptr1, Ptr2 : SET_PTR;
    Buffer     : BITSET;
    W          : CARDINAL;
    Header     : HEAD_REC;
BEGIN
   Ptr1 := PointSet(Set1);
   Ptr2 := PointSet(Set2);
   Header  := Ptr1^.H;
   Ptr1^.H := Ptr2^.H;
   Ptr2^.H := Header;
   FOR W:=0 TO ((_SET_RANGE-SIZE(HEAD_REC)) >> 1)-1 DO
      Buffer       := Ptr1^.Bit[W];
      Ptr1^.Bit[W] := Ptr2^.Bit[W];
      Ptr2^.Bit[W] := Buffer
   END
END SwapSet;

PROCEDURE PutInSet(Element : CARDINAL;
                   Set     : SET_PTR);
BEGIN
   INC(Set^.H.Cardinal);
   INCL(Set^.Set, Element-1)
END PutInSet;

PROCEDURE Union(Set1, Set2 : BITSET) : BITSET;
BEGIN
   RETURN Set1+Set2
```

```
END Union;

PROCEDURE Intersection(Set1, Set2 : BITSET) : BITSET;
BEGIN
   RETURN Set1*Set2
END Intersection;

PROCEDURE Subtraction(Set1, Set2 : BITSET) : BITSET;
BEGIN
   RETURN Set1-Set2
END Subtraction;

PROCEDURE WithSetDo(Do              : SET_PROC;
                    Set1, Set2, Set : SET_PTR) : CARDINAL;
VAR W, bit, Count    : CARDINAL;
BEGIN
   Count:=0;
   FOR W:=0 TO ((_SET_RANGE-SIZE(HEAD_REC)) >> 1)-1 DO
      Set^.Bit[W]:=Do(Set1^.Bit[W], Set2^.Bit[W]);
      FOR bit:=0 TO 15 DO INC(Count, ORD(bit IN Set^.Bit[W])) END
   END;
   RETURN Count
END WithSetDo;

PROCEDURE Difference(Set1, Set2    : SET_PTR;
                     Diff1, Diff2 : WORD_LIST;
                     VAR n1, n2   : CARDINAL);
VAR W, Bit : CARDINAL;
    A, B   : BITSET;
BEGIN
   n1:=0;
   n2:=0;
   FOR W:=0 TO ((_SET_RANGE-SIZE(HEAD_REC)) >> 1)-1 DO
      A:=Set1^.Bit[W]-Set2^.Bit[W];
      B:=Set2^.Bit[W]-Set1^.Bit[W];
      FOR Bit:=0 TO 15 DO
         IF Bit IN A THEN
            Diff1^[n1]:=(W << 4)+Bit+1;
            INC(n1)
         END;
         IF Bit IN B THEN
            Diff2^[n2]:=(W << 4)+Bit+1;
            INC(n2)
         END
      END
   END
END Difference;

PROCEDURE Exclusion(Set1, Set2 : SET_PTR) : CARDINAL;
VAR W, Count, Bit : CARDINAL;
    Set           : BITSET;
BEGIN
   Count:=0;
   FOR W:=0 TO ((_SET_RANGE-SIZE(HEAD_REC)) >> 1)-1 DO
      Set:=Set1^.Bit[W]/Set2^.Bit[W];
      FOR Bit:=0 TO 15 DO INC(Count, ORD(Bit IN Set)) END
   END;
   RETURN Count
END Exclusion;

PROCEDURE Cardinality(Set : SET_PTR) : CARDINAL;
VAR W, Count, Bit : CARDINAL;
BEGIN
   Count:=0;
   FOR W:=0 TO ((_SET_RANGE-SIZE(HEAD_REC)) >> 1)-1 DO
      FOR Bit:=0 TO 15 DO INC(Count, ORD(Bit IN Set^.Bit[W])) END
   END;
   RETURN Count
END Cardinality;
```

## Operations on graphs

```
PROCEDURE Count_Arc_Freq(arc               : INTEGER;
                         VAR Below, Above : INTEGER);
BEGIN
   IF arc<0 THEN
      INC(Below, arc-1)
   ELSIF 0<arc THEN
      INC(Above, arc+1)
   END
END Count_Arc_Freq;
```

```
PROCEDURE Count_Arc(arc          : INTEGER;
                VAR Below, Above : INTEGER);
BEGIN
   IF arc<0 THEN
      DEC(Below)
   ELSIF 0<arc THEN
      INC(Above)
   END
END Count_Arc;

PROCEDURE Count_Freq(arc          : INTEGER;
                 VAR Below, Above : INTEGER);
BEGIN
   IF arc<0 THEN
      INC(Below, arc)
   ELSIF 0<arc THEN
      INC(Above, arc)
   END
END Count_Freq;

PROCEDURE SetArcCounting(Method : CARDINAL);
BEGIN
   CASE Method OF
   |  0 : ArcCounting:=Count_Arc_Freq
   |  1 : ArcCounting:=Count_Arc
   |  2 : ArcCounting:=Count_Freq
   END
END SetArcCounting;

PROCEDURE Back_Arc(arc : INTEGER) : BOOLEAN;
BEGIN
   RETURN (NOT (15 IN BITSET(arc))) AND (NOT (14 IN BITSET(arc)))
END Back_Arc;

PROCEDURE Fore_Arc(arc : INTEGER) : BOOLEAN;
BEGIN
   RETURN (15 IN BITSET(arc)) AND (NOT (14 IN BITSET(arc)))
          AND (arc<>EDGE)
END Fore_Arc;

PROCEDURE Bio_Arc(arc : INTEGER) : BOOLEAN;
BEGIN
   RETURN (EDGE<arc) AND (arc<0)
END Bio_Arc;
```

Before using  Arc and DepthFirstSearch we must initialize with SetArc and SetMat

```
PROCEDURE Arc(i, j : CARDINAL) : BOOLEAN;
VAR Index : INT_PTR;
    arc   : INTEGER;
BEGIN
   Index:=DiagIndex(_MAT_ADDR, i, j, _VERTEX);
   arc:=Index^;
   IF j<i THEN
      IF 15 IN BITSET(arc) THEN
         EXCL(BITSET(arc), 15)
      ELSE
         INCL(BITSET(arc), 15)
      END
   END;
   RETURN _ARC(arc)
END Arc;

PROCEDURE SetArc(Direction : ARC_TYPE);
BEGIN
   CASE Direction OF
   |  Foreward : _ARC:=Fore_Arc;
   |  Backward : _ARC:=Back_Arc
   |  BioStrat : _ARC:=Bio_Arc
   END
END SetArc;
```

The set of successors or predecessors of Vertex is visited and contains Vertex itself

```
PROCEDURE DepthFirstSearch(Direction : ARC_TYPE;
                           Vertex    : CARDINAL;
                           Visited   : SET_PTR);
```

```
   PROCEDURE DepthSearch(Vertex : CARDINAL);
   VAR v : CARDINAL;
   BEGIN
      IF NOT (Vertex-1 IN Visited^.Set) THEN
         PutInSet(Vertex, Visited);
         FOR v:=1 TO _VERTEX DO
            IF (Vertex<>v) AND Arc(Vertex, v) THEN DepthSearch(v) END
         END
      END
   END DepthSearch;

BEGIN
   ClearMemory(Visited, LONGCARD(_SET_RANGE));
   SetArc(Direction);
   DepthSearch(Vertex)
END DepthFirstSearch;

PROCEDURE SetMat(Mat    : ADDRESS;
                 Vertex : CARDINAL);
BEGIN
   _MAT_ADDR:=Mat;
   _VERTEX:=Vertex
END SetMat;

PROCEDURE BestFit(Set1, Authorized : SET_PTR;
                  Number_Of_Vertex : CARDINAL;
                  List             : WORD_LIST);
VAR Min, v : CARDINAL;
    Set2   : SET_PTR;
    L      : CARDINAL;
BEGIN

     List of the cardinalities of each exclusion

   L:=0;

     List of vertices to be compared

   Set2:=_SEGMENT;

     Initialisation of the test value

   Min:=MAX(CARDINAL);
   FOR v:=1 TO Number_Of_Vertex DO

     If vertex allowed

      IF v-1 IN Authorized^.Set THEN

     Cardinality of exclusion

         List^[L]:=Exclusion(Set1, Set2);

     Test

         IF List^[L]<Min THEN Min:=List^[L] END
      ELSE
         List^[L]:=MAX(CARDINAL)
      END;
      IncAddr(Set2, _SET_RANGE);
      INC(L)
   END;
   L:=0;

     Union the the vertex/vertices with positive test

   FOR v:=1 TO Number_Of_Vertex DO
      IF List^[L]=Min THEN
         Set2:=PointSet(v);
         Set2^.H.Cardinal:=WithSetDo(Union, Set1, Set2, Set2)
      END;
      INC(L)
   END
END BestFit;

PROCEDURE ByFAD(T1, T2 : CARDINAL) : BOOLEAN;
VAR Taxon1, Taxon2 : HEAD_PTR;
BEGIN
```

```
   Taxon1:=AddAddr(_SEGMENT, (T1-1)*SIZE(HEAD_REC));
   Taxon2:=AddAddr(_SEGMENT, (T2-1)*SIZE(HEAD_REC));
   IF Taxon1^.Level<Taxon2^.Level THEN
      RETURN TRUE
   ELSIF Taxon1^.Level=Taxon2^.Level THEN
      IF Taxon1^.Cardinal=Taxon2^.Cardinal THEN
         RETURN Taxon1^.Section<Taxon2^.Section
      ELSE
         RETURN Taxon1^.Cardinal<Taxon2^.Cardinal
      END
   ELSE
      RETURN FALSE
   END
END ByFAD;

PROCEDURE ByLAD(T1, T2 : CARDINAL) : BOOLEAN;
VAR Taxon1, Taxon2 : HEAD_PTR;
BEGIN
   Taxon1:=AddAddr(_SEGMENT, (T1-1)*SIZE(HEAD_REC));
   Taxon2:=AddAddr(_SEGMENT, (T2-1)*SIZE(HEAD_REC));
   IF Taxon1^.Cardinal<Taxon2^.Cardinal THEN
      RETURN TRUE
   ELSIF Taxon1^.Cardinal=Taxon2^.Cardinal THEN
      IF Taxon1^.Level=Taxon2^.Level THEN
         RETURN Taxon1^.Section<Taxon2^.Section
      ELSE
         RETURN Taxon1^.Level<Taxon2^.Level
      END
   ELSE
      RETURN FALSE
   END
END ByLAD;

PROCEDURE SetAUSorting(Method : CARDINAL);
BEGIN
   CASE Method OF
   |  0 : AU_Sorting:=ByFAD
   |  1 : AU_Sorting:=ByLAD
   END
END SetAUSorting;
```

## Allocations of memories

This is not a dynamic managment of the memory. The pile is done by the logical succession of appearance of the variableds within the main module. The allocation puts the pointer on the new allocated segment. The available memory is rounded at the paragraph

```
PROCEDURE New(VAR Ptr   : ADDRESS;
                  Count : LONGCARD);
BEGIN
   IF _HEAP=HEAP_SIZE THEN Error(203) END;
   Count:=((Count-1) >> 4)+1;
   IF HeapTotalAvail(MainHeap)<CARDINAL(Count) THEN Error(254) END;
   HeapAllocate(MainHeap, Ptr, CARDINAL(Count));
   _SEGMENT:=Ptr;
   INC(_HEAP);
   _HEAP_LIST[_HEAP].Ptr:=Ptr;
   _HEAP_LIST[_HEAP].Size:=CARDINAL(Count)
END New;
```

Liberation of all the pile to the mentioned pointer but without modification of the value of the current pointer.

```
PROCEDURE Release(Ptr : ADDRESS);
BEGIN
   WHILE _HEAP_LIST[_HEAP].Ptr<>Ptr DO
      HeapDeallocate(MainHeap, _HEAP_LIST[_HEAP].Ptr, _HEAP_LIST[_HEAP].Size);
      DEC(_HEAP)
   END;
   HeapDeallocate(MainHeap, _HEAP_LIST[_HEAP].Ptr, _HEAP_LIST[_HEAP].Size);
   DEC(_HEAP)
END Release;
```

Purge of a memory sector the size of which is defined by an even  number of bytes

```
PROCEDURE ClearMemory(Ptr   : ADDRESS;
                      Count : LONGCARD);
BEGIN
   Count:=Count >> 1;
   WHILE 4000H<Count DO
      WordFill(Ptr, 4000H, 0);
      IncAddr(Ptr, 8000H);
      DEC(Count, 4000H)
   END;
   WordFill(Ptr, CARDINAL(Count), 0)
END ClearMemory;

PROCEDURE DisposeHeap;
BEGIN
   Release(_HEAP_LIST[1].Ptr)
END DisposeHeap;

PROCEDURE MemAvail() : LONGCARD;
BEGIN
   RETURN LONGCARD(HeapTotalAvail(MainHeap)) << 4
END MemAvail;

BEGIN
   _HEAP:=0;
   _HEAP_LIST[0].Ptr:=NIL
END BG_MEM.
```

## *Library BG_CRT*

Function : management of the displays of BG_GRAP

```
DEFINITION MODULE BG_CRT;

   PROCEDURE Gauge(g : REAL);
   PROCEDURE ClearGauge;

   PROCEDURE Message(Msg : CARDINAL);
   PROCEDURE Nb_Message(Msg, Nb : CARDINAL);
   PROCEDURE St_Message(Msg : CARDINAL;
                        St  : ARRAY OF CHAR);
   PROCEDURE Static_Message(Nb : CARDINAL);
   PROCEDURE ElapsedTime;

END BG_CRT.

IMPLEMENTATION MODULE BG_CRT;

FROM Window IMPORT RelCoord, WinType, WinDef, CenterUpperTitle,
                   Black, Blue, Magenta,
                   LightGray, LightGreen, LightCyan, LightRed, Yellow, White,
                   SingleFrame, FullScreen,
                   Clear, ClrEol, WhereX, WhereY, GotoXY, CursorOff,
                   TextColor, TextBackground,
                   Open, SetTitle, Use;
FROM Lib    IMPORT Dos, SetReturnCode, ParamCount, ParamStr,
                   Sound, NoSound, Delay,
                   WordFill;
FROM Str    IMPORT Append, StrToCard;
FROM SYSTEM IMPORT Registers;
FROM BG_ERR IMPORT Error;
FROM BG_MEM IMPORT MemAvail, SetArcCounting, SetAUSorting;
FROM BG_FIL IMPORT _REPORT;

IMPORT IO, FIO;

VAR _C         : BOOLEAN;
    _W         : WinType;
    _X         : RelCoord;
    _H, _M, _S : CARDINAL;

PROCEDURE GetTime(VAR hour, min, sec : CARDINAL);
VAR r : Registers;
BEGIN
   r.AH := 2CH;
   Dos(r);
   hour := CARDINAL(r.CH);
   min  := CARDINAL(r.CL);
   sec  := CARDINAL(r.DH)
END GetTime;

PROCEDURE GetDate(VAR day, month, year : CARDINAL);
VAR r : Registers;
BEGIN
   r.AH := 2AH;
   Dos(r);
   day  := CARDINAL(r.DL);
   month:= CARDINAL(r.DH);
   year := CARDINAL(r.CX)
END GetDate;

PROCEDURE CommandLine;
VAR cmd : ARRAY[0..1] OF CHAR;
BEGIN
   WordFill(ADR(_REPORT), SIZE(_REPORT) >> 1, 0);

   IF ParamCount()<>4 THEN Error(111) END;
   ParamStr(cmd, 2);
```

Type of screen

```
   _C:=cmd[0]='C';
   ParamStr(cmd, 3);
```

Method of counting

```
   CASE cmd[0] OF
   |  'S' : _REPORT.Met:=0
   |  'A' : _REPORT.Met:=1
   |  'R' : _REPORT.Met:=2
   END;
   ParamStr(cmd, 4);
```

Key of sorting

```
   _REPORT.Sor:=ORD(cmd[0]='L');
   SetArcCounting(_REPORT.Met);
   SetAUSorting(_REPORT.Sor)
END CommandLine;

PROCEDURE Beep;
VAR i, j : CARDINAL;
BEGIN
   FOR i:=600 TO 200 BY -200 DO
      j:=1;
      REPEAT
         Sound(i*j);
         Delay(100);
         j:=j*2
      UNTIL 4<j
   END;
   NoSound
END Beep;

PROCEDURE Gauge(g : REAL);
VAR p : CARDINAL;
BEGIN
   Use(FullScreen);
   IF 1.0<g THEN
      GotoXY(20,8);
      IO.WrCharRep('_', 24);
      IO.WrStr(' Overflow !!! __')
   ELSE
      p:=TRUNC(80.0*g);
      GotoXY(20,8); IO.WrCharRep('_', p >> 1);
      IF ODD(p) THEN IO.WrChar('_') END
   END
END Gauge;

PROCEDURE ClearGauge;
BEGIN
   Use(FullScreen);
   GotoXY(20,8);
   IO.WrCharRep('·', 40)
END ClearGauge;

PROCEDURE Message(Msg : CARDINAL);
VAR m : ARRAY[0..70] OF CHAR;
```

```
BEGIN
   Use(_W);
   IO.WrLn;
   CASE Msg OF
      Message of the  module BG_FIL
   |  0 : m:="  Writing"
      Message of the procedure AllReduction
   |  1 : m:="> Searching the local maximal horizons"
      Message of the procedure AllReduction
   |  2 : m:="Section"
      Message of the procedures AllReduction and Main
   |  3 : m:="Levels"
      Message of the procedures AllReduction and Main
   |  4 : m:="Residual horizons"
      Message of the procedure AllReduction
   |  5 : m:="Number OF local maximal horizons"
      Message of the procedure Main
   |  6 : m:="> Searching the residual maximal horizons"
      Message of the procedure Main
   |  7 : m:="RESIDUAL MAXIMAL HORIZONS"
      Message of the procedures MakeMatrix_A and MakeMatrix_H
   |  8 : IO.WrLngCard(MemAvail(), 0); m:=" bytes OF available memory"
      Message of the procedure MakeMatrix_A
   |  9 : m:="> Making the adjacency matrix OF the biostratigraphic graph G*"
      Message of the procedure Unification
   | 10 : m:="> Maximal cliques OF G*"
      Message of the procedure Unification
   | 11 : m:="CLIQUES"
      Message of the procedure MakeMatrix_H
   | 12 : m:="> Making the adjacency matrix OF the graph Gk"
      Message of the procedure MakeMatrix_H
   | 13 : m:="Counting the number OF contradictions"
      Message of the procedure Connexe
   | 14 : m:="> Searching the strongly connected components"
      Message of the procedure Connexe
   | 15 : m:="  Strongly connected component"
      Message of the procedure Connexe
   | 16 : m:="Connected vertices"
      Message of the procedure Connexe
   | 17 : m:="Undetermined arcs"
      Message of the procedure Connexe
   | 18 : m:="Total undetermined arcs"
      Message of the procedure Connexe
   | 19 : m:="No strongly connected component"
      Message of the procedure MaximumPath
   | 20 : m:="> Searching the longest path OF Gk (L)"
      Message of the procedure SearchPathSet in MaximumPath
   | 21 : m:="¡ Bad news: disconnected parallel longest paths => despotic choice !"
      Message of the procedure SearchPathMax in MaximumPath
   | 22 : m:="Path length"
      Message of the procedure MaximumPath
   | 23 : m:="> Reduction OF Gk TO a single path (L')"
      Message of the procedure Continuity
   | 24 : m:="> Making the adjacency matrix OF L'"
      Message of the procedure InsertEdge
   | 25 : m:="> Searching the residual virtual edges"
      Message of the procedure Detect_ReverseArc in InsertEdge
   | 26 : m:="Residual virtual edges"
      Message of the procedure Insert_VirtualEdge in InsertEdge
   | 27 : m:="Insertion OF the virtual edges"
      Message of the procedure SplitTrick
   | 28 : m:="> Residual arcs"
      Message of the procedure Correlation
   | 29 : m:="> Correlations and reproducibility table"
      Message of the procedure Main
   | 30 : m:="> REPORT:"
```

```
     Message of the procedure Main
   | 32 : m:="Cliques"
     Message of the procedure Main
   | 33 : m:="UNITARY ASSOCIATIONS"
     Message of the procedure ElapsedTime in module BG_CRT
   | 34 : m:="Elapsed time:"
   | 35 : m:="> Press a key TO continue..."
   | 100 : m:="¡ FATAL ERROR 100: bad disk read"
   | 111 : m:="¡ FATAL ERROR 111: invalid command line"
   | 203 : m:="¡ FATAL ERROR 203: heap overflow"
   | 254 : m:="¡ FATAL ERROR 254: not enough memory"
   END;
   Append(m, ' ');
   CASE m[0] OF
   |  '>' : IF _C THEN
               TextColor(White)
            ELSE
               TextColor(Black)
            END;
            TextBackground(LightGray)
   |  '¡' : IF _C THEN
               TextColor(LightRed);
               TextBackground(Magenta)
            ELSE
               TextColor(Black);
               TextBackground(LightGray)
            END
   END;
   IO.WrStr(m);
   _X:=WhereX();
   IF (m[0]='>') OR (m[0]='¡') THEN
      IO.WrCharRep(' ', 78-_X);
      IF _C THEN TextColor(LightGreen)
            ELSE TextColor(LightGray) END;
      TextBackground(Black)
   END
END Message;

PROCEDURE Nb_Message(Msg, Nb : CARDINAL);
BEGIN
   Use(_W);
   Message(Msg);
   IO.WrCard(Nb, 0)
END Nb_Message;

PROCEDURE St_Message(Msg : CARDINAL;
                     St  : ARRAY OF CHAR);
BEGIN
   Use(_W);
   Message(Msg);
   IO.WrStr(St)
END St_Message;

PROCEDURE Static_Message(Nb : CARDINAL);
BEGIN
   Use(_W);
   GotoXY(_X, WhereY());
   IO.WrCard(Nb, 0);
   ClrEol
END Static_Message;

PROCEDURE ElapsedTime;
VAR h, m, s : CARDINAL;
BEGIN
   GetTime(h, m, s);
   IF s<_S THEN
      INC(s, 60);
      INC(_M)
   END;
   _S:=s-_S;
   IF m<_M THEN
      INC(m, 60);
      INC(_H)
   END;
   _M:=m-_M;
   IF h<_H THEN INC(h, 24) END;
   _M:=60*(h-_H)+_M;
   Message(34);
   IO.WrCard(_M, 0); IO.WrChar("'");
   IO.WrCard(_S, 0); IO.WrChar('"');
   IF 1<_M THEN Beep END;
   _REPORT.Min:=_M;
   _REPORT.Sec:=_S
END ElapsedTime;

BEGIN
   CommandLine;
   CursorOff;
   IF _C THEN TextColor(LightCyan)
         ELSE TextColor(LightGray) END;
   TextBackground(Black);
   Clear;
   GotoXY(1,1);
   IO.WrStr('+----------------------------------------------------------------------+');
   IO.WrStr('_ B I O G R A P H - BG_GRAP v2.01    Copyright (c) 1990 by J. Savary & J.
            Guex _');
   IO.WrStr('_ Institute OF Geology, University OF Lausanne, UNIL - BFSH2, CH-1015
            Lausanne _');
   IO.WrStr('+---------------------------------------------------------------------+');
   IF _C THEN TextColor(LightGreen) END;
   GotoXY(22,5); IO.WrStr('Analysis OF the biostratigraphic graph');
   GotoXY(18,6); IO.WrStr('+-----------------------------------------------------+');
   GotoXY(18,7); IO.WrStr('_ 0     20      40      60      80     100 % _');
   GotoXY(18,8); IO.WrStr('_                                           _');
   GotoXY(18,9); IO.WrStr('+-----------------------------------------------------+');
   IF _C THEN
      TextColor(Yellow);
      TextBackground(Blue);
      _W:=Open(WinDef(0, 9, 79, 24, LightGreen, Black,
                      FALSE, TRUE, FALSE, TRUE, SingleFrame, White, Black))
   ELSE
      _W:=Open(WinDef(0, 9, 79, 24, LightGray, Black,
                      FALSE, TRUE, FALSE, TRUE, SingleFrame, LightGray, Black))
   END;
   SetTitle(_W, ' Messages ', CenterUpperTitle);
   ClearGauge;
   GetTime(_H, _M, _S)
END BG_CRT.
```

## *Library BG_ERR*

Function : module de gestion des erreurs de BG_GRAP

```
DEFINITION MODULE BG_ERR;

   PROCEDURE Error(Err : SHORTCARD);

END BG_ERR.

IMPLEMENTATION MODULE BG_ERR;

IMPORT SYSTEM;

FROM AsmLib IMPORT SetInProgramFlag;
FROM Lib    IMPORT SetReturnCode;
FROM BG_CRT IMPORT Message;

PROCEDURE Error(Err : SHORTCARD);
BEGIN
   Message(CARDINAL(Err));
   SetReturnCode(Err);
   HALT
END Error;

(*$C 00,N*)
PROCEDURE (*$F*) ForceSave(p : CARDINAL) : CARDINAL; FORWARD;
(*$C F0,F*)

(*$C FF,J+*)
PROCEDURE Int24Handler(Dummy : CARDINAL);

TYPE IntReg1 = RECORD
                 DI,SI,ES,DS : CARDINAL;
                 CASE : BOOLEAN OF
                 |  TRUE  : BX,DX,CX,AX             : CARDINAL;
                 |  FALSE : BL,BH,DL,DH,CL,CH,AL,AH : SHORTCARD
```

```
                  END
               END;

VAR RegP1 : POINTER TO IntReg1;

BEGIN
     RegP1:=[SYSTEM.Seg(Dummy):ForceSave(0)+2];
     SetInProgramFlag(TRUE);
     Error(SHORTCARD(RegP1^.DI+150));
     HALT
END Int24Handler;
(*$C F0,J-*)

PROCEDURE ForceSave(p : CARDINAL): CARDINAL;
BEGIN
   RETURN SYSTEM.Ofs(p)
END ForceSave;

VAR Int24Vec[0:24H*4] : PROCEDURE ( CARDINAL );

BEGIN
   SYSTEM.DI;
   Int24Vec:=Int24Handler;
   SYSTEM.EI
END BG_ERR.
```

## *Library BG_FIL*

Function : management of the input/outputs of BG_GRAP

```
DEFINITION MODULE BG_FIL;

IMPORT FIO;

   TYPE EXTENSION  = ARRAY[0..3] OF CHAR;
        REPORT_REC = RECORD
                        HMR, Cliques, AU,
                        Min, Sec, Contras, Edge,
                        Connex, Vertex, Destroy, Met, Sor : CARDINAL
                     END;

   VAR _REPORT : REPORT_REC;

   PROCEDURE Load_Levels(Ext              : EXTENSION;
                     VAR Taxa, NSec, NLev : CARDINAL);
   PROCEDURE Load_Matrix(Ext  : EXTENSION;
                     VAR Mat  : ADDRESS;
                     VAR NLev : CARDINAL);
   PROCEDURE Save_Levels(Ext              : EXTENSION;
                         Taxa, NSec, NLev : CARDINAL);
   PROCEDURE Save_Clique(Ext        : EXTENSION;
                         Taxa, NLev : CARDINAL);
   PROCEDURE Save_Matrix(Ext  : EXTENSION;
                         Mat  : ADDRESS;
                         Taxa : CARDINAL);
   PROCEDURE Create_Connex(Ext : EXTENSION);
   PROCEDURE Save_Connex(Conn : ADDRESS);
   PROCEDURE Save_DestroyedArc(V1, V2 : CARDINAL);
   PROCEDURE Close_Connex(Nb_Vertex, Connex, Destroyed : CARDINAL);
   PROCEDURE Save_AU(Ext        : EXTENSION;
                     Taxa, NLev : CARDINAL);
   PROCEDURE Save_Corr(Ext : EXTENSION;
                       N   : CARDINAL);
   PROCEDURE Save_Report(Ext : EXTENSION);

END BG_FIL.

IMPLEMENTATION MODULE BG_FIL;

IMPORT FIO;

FROM Str    IMPORT Append;
FROM Lib    IMPORT IncAddr, ParamStr;
FROM BG_ERR IMPORT Error;
FROM BG_MEM IMPORT HEAD_REC, CORR_REC, GetSegment, SetRange,
                   GetRange, SizeOfDiagMat, New, ClearMemory;
```

```
FROM BG_CRT IMPORT St_Message;

CONST BUFFER = 0FFF8H;

VAR _F        : FIO.File;
    _FILENAME : FIO.PathStr;

PROCEDURE CreateFileName(Name : FIO.PathStr;
                         Ext  : EXTENSION) : FIO.PathStr;
BEGIN
   Append(Name, Ext);
   RETURN Name
END CreateFileName;

PROCEDURE LoadFile(ptr  : ADDRESS;
                   size : LONGCARD);
VAR read : CARDINAL;
BEGIN
   WHILE BUFFER<size DO
      read:=FIO.RdBin(_F, ptr^, BUFFER);
      IncAddr(ptr, read);
      DEC(size, LONGCARD(read))
   END;
   read:=FIO.RdBin(_F, ptr^, CARDINAL(size));
   IF read<CARDINAL(size) THEN Error(100) END;
   FIO.Close(_F)
END LoadFile;

PROCEDURE Load_Levels(Ext              : EXTENSION;
                  VAR Taxa, NSec, NLev : CARDINAL);
VAR Level : ADDRESS;
BEGIN
   _F:=FIO.Open(CreateFileName(_FILENAME, Ext));
   IF FIO.RdBin(_F, Taxa, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   IF FIO.RdBin(_F, NSec, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   IF FIO.RdBin(_F, NLev, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   SetRange(Taxa);
   New(Level, LONGCARD(NLev)*LONGCARD(GetRange())+SIZE(CARDINAL));
   ClearMemory(Level, LONGCARD(NLev)*LONGCARD(GetRange())+SIZE(CARDINAL));
   LoadFile(Level, LONGCARD(NLev)*LONGCARD(GetRange()))
END Load_Levels;

PROCEDURE Load_Matrix(Ext  : EXTENSION;
                  VAR Mat  : ADDRESS;
                  VAR Size : CARDINAL);
BEGIN
   _F:=FIO.Open(CreateFileName(_FILENAME, Ext));
   IF FIO.RdBin(_F, Size, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   New(Mat, SizeOfDiagMat(Size));
   LoadFile(Mat, SizeOfDiagMat(Size))
END Load_Matrix;

PROCEDURE SaveFile(ptr  : ADDRESS;
                   size : LONGCARD);
BEGIN
   WHILE BUFFER<size DO
      FIO.WrBin(_F, ptr^, BUFFER);
      IncAddr(ptr, BUFFER);
      DEC(size, BUFFER)
   END;
   FIO.WrBin(_F, ptr^, CARDINAL(size));
   FIO.Close(_F)
END SaveFile;

PROCEDURE Save_Levels(Ext              : EXTENSION;
                      Taxa, NSec, NLev : CARDINAL);
BEGIN
   _F:=FIO.Create(CreateFileName(_FILENAME, Ext));
   St_Message(0, CreateFileName(_FILENAME, Ext));
   FIO.WrBin(_F, Taxa, SIZE(CARDINAL));
   FIO.WrBin(_F, NSec, SIZE(CARDINAL));
   FIO.WrBin(_F, NLev, SIZE(CARDINAL));
   SaveFile(GetSegment(), LONGCARD(NLev)*LONGCARD(GetRange()))
END Save_Levels;

PROCEDURE Save_Clique(Ext        : EXTENSION;
                      Taxa, NLev : CARDINAL);
VAR ptr : ADDRESS;
    L   : CARDINAL;
BEGIN
   _F:=FIO.Create(CreateFileName(_FILENAME, Ext));
   St_Message(0, CreateFileName(_FILENAME, Ext));
   ptr:=GetSegment();
   FOR L:=1 TO NLev DO
      FIO.WrBin(_F, ptr^, 2*SIZE(CARDINAL));
      IncAddr(ptr, GetRange())
   END;
   FIO.Close(_F)
END Save_Clique;

PROCEDURE Save_Matrix(Ext  : EXTENSION;
                      Mat  : ADDRESS;
                      Size : CARDINAL);
BEGIN
   _F:=FIO.Create(CreateFileName(_FILENAME, Ext));
   St_Message(0, CreateFileName(_FILENAME, Ext));
   FIO.WrBin(_F, Size, SIZE(CARDINAL));
   SaveFile(Mat, SizeOfDiagMat(Size))
END Save_Matrix;

PROCEDURE Create_Connex(Ext : EXTENSION);
VAR Head : HEAD_REC;
BEGIN
   Head.Section :=0;
   Head.Level   :=0;
   Head.Cardinal:=0;
   _F:=FIO.Create(CreateFileName(_FILENAME, Ext));
   FIO.WrBin(_F, Head, SIZE(Head))
END Create_Connex;

PROCEDURE Save_Connex(Conn : ADDRESS);
BEGIN
   FIO.WrBin(_F, Conn^, GetRange())
END Save_Connex;

PROCEDURE Save_DestroyedArc(V1, V2 : CARDINAL);
BEGIN
   FIO.WrBin(_F, V1, SIZE(V1));
   FIO.WrBin(_F, V2, SIZE(V2))
END Save_DestroyedArc;

PROCEDURE Close_Connex(Nb_Vertex, Connex, Destroyed : CARDINAL);
VAR stop : CARDINAL;
BEGIN
   stop:=0;
   FIO.WrBin(_F, stop     , SIZE(stop     ));
   FIO.Seek(_F, 0);
   FIO.WrBin(_F, Nb_Vertex, SIZE(Nb_Vertex));
   FIO.WrBin(_F, Connex   , SIZE(Connex   ));
   FIO.WrBin(_F, Destroyed, SIZE(Destroyed));
   FIO.Close(_F)
END Close_Connex;

PROCEDURE Save_AU(Ext        : EXTENSION;
                  Taxa, NLev : CARDINAL);
BEGIN
   St_Message(0, CreateFileName(_FILENAME, Ext));
   _F:=FIO.Create(CreateFileName(_FILENAME, Ext));
   FIO.WrBin(_F, Taxa, SIZE(Taxa));
   FIO.WrBin(_F, NLev, SIZE(NLev));
   SaveFile(GetSegment(), LONGCARD(Taxa)*SIZE(HEAD_REC))
END Save_AU;

PROCEDURE Save_Corr(Ext : EXTENSION;
                    N   : CARDINAL);
BEGIN
   St_Message(0, CreateFileName(_FILENAME, Ext));
   _F:=FIO.Create(CreateFileName(_FILENAME, Ext));
   FIO.WrBin(_F, N, SIZE(N));
   SaveFile(GetSegment(), LONGCARD(N)*SIZE(CORR_REC))
END Save_Corr;

PROCEDURE Save_Report(Ext : EXTENSION);
BEGIN
   _F:=FIO.Append(CreateFileName(_FILENAME, Ext));
   FIO.WrLn(_F);
   FIO.WrStr(_F, '   Elapsed time: ');
   FIO.WrCard(_F, _REPORT.Min,0);
   FIO.WrChar(_F, "'");
   FIO.WrCard(_F, _REPORT.Sec,0);
   FIO.WrChar(_F, '"'); FIO.WrLn(_F);
   FIO.WrStr(_F, 'Counting method: ');
   CASE _REPORT.Met OF
   |   0 : FIO.WrStr(_F, 'Arcs & reproducibility')
   |   1 : FIO.WrStr(_F, 'Arcs')
   |   2 : FIO.WrStr(_F, 'Reproducibility')
   END;
   FIO.WrLn(_F);
   FIO.WrStr(_F, ' Sorting method: ');
   CASE _REPORT.Sor OF
   |   0 : FIO.WrStr(_F, 'By first appearance')
   |   1 : FIO.WrStr(_F, 'By last appearance')
   END;
   FIO.WrLn(_F);
   FIO.WrLn(_F);
   FIO.WrCard(_F, _REPORT.Contras, 5);
   FIO.WrStr(_F, ' Contradictions');
   FIO.WrLn(_F);
   FIO.WrCard(_F, _REPORT.Edge    , 5);
   FIO.WrStr(_F, ' Residual virtual edges');
   FIO.WrLn(_F);
   FIO.WrCard(_F, _REPORT.Connex , 5);
   FIO.WrStr(_F, ' Strongly connected components');
   FIO.WrLn(_F);
   FIO.WrCard(_F, _REPORT.Vertex , 5);
   FIO.WrStr(_F, ' Vertices in strong components');
   FIO.WrLn(_F);
   FIO.WrCard(_F, _REPORT.Destroy, 5);
   FIO.WrStr(_F, ' Undetermined arcs in strong components');
   FIO.WrLn(_F);
   FIO.WrCard(_F, _REPORT.HMR     , 5);
   FIO.WrStr(_F, ' Residual maximal horizons');
   FIO.WrLn(_F);
   FIO.WrCard(_F, _REPORT.Cliques, 5);
   FIO.WrStr(_F, ' Maximal cliques');
   FIO.WrLn(_F);
   FIO.WrCard(_F, _REPORT.AU      , 5);
   FIO.WrStr(_F, ' Unitary associations');
   FIO.WrLn(_F);
   FIO.Close(_F)
END Save_Report;

BEGIN
   ParamStr(_FILENAME, 1)
END BG_FIL.
```

## 5.3 Managers (Pascal)

### *Program BG*

Function : pilote of the interface and of the diverse modules

```
PROGRAM BG;

{$A+,B-,D-,E+,F-,G+,L-,N-,O-,I+,R-,S+,V-,X+}

{$M 4000, 0, 0}

USES Crt, Dos, BG_Def;

TYPE STR12 = STRING[12];

CONST Nfiles = 7;
      FILES  : ARRAY[1..Nfiles] OF STR12 = ('BG.CFG',      { 1 }
                                            'BG.HLP',      { 2 }
                                            'BG_MENU.EXE', { 3 }
                                            'BG_DATA.EXE', { 4 }
                                            'BG_GRAP.EXE', { 5 }
                                            'BG_CONV.EXE', { 6 }
                                            'BG_DRAW.EXE');{ 7 }
      BIOGRAPH  : PathStr = 'C:\USER\';
      EDITOR    : PathStr = 'C:\BRIEF\B.EXE';

VAR _Cfg      : CFG;
    _ExitSave : POINTER;

{$F+} PROCEDURE FatalError; {$F-}
```

```
BEGIN
   ErrorAddr:=NIL;
   ExitProc:=_ExitSave
END; { FatalError }

PROCEDURE Verify;
VAR I : BYTE;
BEGIN
   FOR I:=1 TO Nfiles DO
      IF FSearch(FILES[I], BIOGRAPH)=''
         THEN BEGIN
            Writeln(FILES[I], ' not found: run INSTALL');
            HALT
         END
END; { Verify }

PROCEDURE ReadCFG;
VAR f : FILE OF CFG;
BEGIN
   Assign(f, BIOGRAPH+FILES[1]);
   Reset(f);
   Read(f, _Cfg);
   Close(f)
END; { ReadCFG }

PROCEDURE Pause;
BEGIN
   REPEAT UNTIL readkey<>#0
END; { Pause }

FUNCTION Abort(Exe     : BYTE;
               Command : STRING) : BOOLEAN;
BEGIN
   SwapVectors;
   IF Exe=8 THEN
      Exec(EDITOR, Command)
   ELSE
      Exec(BIOGRAPH+FILES[Exe], Command);
   SwapVectors;
   IF (DosError<>0) OR ((Exe=3) AND (DosExitCode<>0)) THEN BEGIN
      Writeln('Fatal error ', DosError, ': retry BG or run INSTALL');
      HALT
   END;
   Abort:=DosExitCode<>0
END; { Abort }

PROCEDURE DoAll;
VAR color, method, sort : char;

BEGIN
   IF Abort(4, AddrToString(_Cfg)) THEN EXIT;
   IF _Cfg.Color THEN            color :='C'
   ELSE                          color :='m';
   IF ArRe in _Cfg.Setup THEN    method:='S'
   ELSE IF Ar in _Cfg.Setup THEN method:='A'
   ELSE                          method:='R';
   IF First in _Cfg.Setup THEN   sort  :='F'
   ELSE                          sort  :='L';
   IF Abort(5, _Cfg.Dpath+_Cfg.Dname+'. '+color+' '+method+' '+sort) THEN BEGIN
      textcolor(lightgray);
      textbackground(black);
      clreol;
      GotoXY(1, wherey+1); clreol;
      gotoxy(1, wherey+1); Write('Press a key TO continue...'); clreol;
      GotoXY(1, wherey+1); clreol;
      Pause;
      exit
   END;
   Pause;
   IF 0<_Cfg.Output THEN
       IF Abort(6, AddrToString(_Cfg)) THEN
END; { DoAll }

BEGIN
   _ExitSave:=ExitProc;
   ExitProc:=@FatalError;
   Verify;
   CheckBreak:=FALSE;
   ReadCfg;
   IF Abort(3, AddrToString(_Cfg)+' S') THEN;
   WHILE _Cfg.Run<>Quit DO BEGIN
      CASE _Cfg.Run OF
```

```
         GoAll    : DoAll;
         GoTrans : IF Abort(6, AddrToString(_Cfg)) THEN;
         Edit     : IF Abort(8, _Cfg.EPath+_Cfg.EName+_Cfg.EExt) THEN;
         Grb,
         Grk,
         Grr      : IF Abort(7, AddrToString(_Cfg)) THEN;
         ConvDS,
         ConvSD   : IF Abort(4, AddrToString(_Cfg)) THEN
      END;
      IF Abort(3, AddrToString(_Cfg)) THEN
   END
END.
```

## *Program BG_MENU*

Function : interface with the user

```
PROGRAM BG_MENU;

{$A+,B-,D-,E+,F-,G+,I-,L-,N-,O-,R-,S+,V-,X+}
{$M 24576, 0, 655360 }

USES Video, Crt, Dos, Printer, TP_Win_E, Tools, Error, BG_Def, BG_Scr;

CONST
   wfCopy : ScreenColor = (LightGray, White     );
   wbCopy : ScreenColor = (Black    , Magenta   );
   ffCopy : ScreenColor = (LightGray, LightRed );

   IX1 = 49; IY1 =  7; IX2 = 78; IY2 = 19;
   Mx1 = 39; My1 = 12; Mx2 = 76; My2 = 22;

   _Path    : PathStr = 'C:\USER\';
   _Editor : PathStr = 'C:\BRIEF\B.EXE';

VAR
   _DataFile : PathStr;
   _Out      : ARRAY[81..93] OF StrPtr;
   _Cfg      : CFGPTR;

   _Run, _Exec, _Count, _Sort, _DrawTo, _Sym, _Draw, _Conv : BYTE;

   _Set1   , _Set4   ,
   _Set21  , _Get31  , _Get32  ,
   _Set22  , _Get41  , _Get42  , _Get43,
   _Set23  , _Get51  , _Get52  ,
   _Set24  , _Get61  , _Get62  ,
   _Set25  , _Get71  , _Get72  , _Get73,
   _Set113, _Set114                     : StrPtr;

{$F+} PROCEDURE FatalError(error : INTEGER); {$F-}
BEGIN
   IF error=399 THEN BEGIN
      DisplayMsg('Bad environment: run INSTALL');
      Pause
   END
   ELSE IF error<>0 THEN BEGIN
      DisplayMsg(ErrorStr(error));
      Pause
   END;
   CloseJob
END; { FatalError }

{$F+} FUNCTION Warnings(error : WORD) : Str74; {$F-}
BEGIN
   CASE error OF
      400 : Warnings:='Data file not found';
      401 : Warnings:='Data file not selected';
      402 : Warnings:='Input and output files are not defined';
      403 : Warnings:='Input file not defined';
      404 : Warnings:='Output file not defined';
      405 : Warnings:='Input and output files are identical';
      406 : Warnings:='Not a BGD file';
      407 : Warnings:='Not a BGF file';
      408 : Warnings:='Not a BGL file';
      409 : Warnings:='Mouse not installed';
      410 : Warnings:='Editor not installed (run INSTALL)';
      411 : Warnings:='VGA/EGA graphic adaptator not detected'
   END
END; { Warnings }

FUNCTION SetupToByte(setup : OPTIONS) : BYTE;
VAR s : OPTIONSET;
    w : WORD ABSOLUTE s;
BEGIN
   CASE setup OF
      All   : s:=_Cfg^.Setup*[All, Trans];
      ArRe  : s:=_Cfg^.Setup*[ArRe, Ar, Re];
      First : s:=_Cfg^.Setup*[First, Last];
      Print : s:=_Cfg^.Setup*[Print, Plot];
      ASCII : s:=_Cfg^.Setup*[ASCII, Light, Dark];
      Gb    : s:=_Cfg^.Setup*[Gb, Gk, Gr]
   END;
   SetupToByte:=w SHR (Ord(setup)+1)
END; { SetupToByte }

PROCEDURE ByteToSetup(option : WORD;
                      setup  : OPTIONS);
VAR s : OPTIONSET;
    w : WORD ABSOLUTE s;
BEGIN
   CASE option OF
      0 : w:=1;
      1 : w:=2;
      2 : w:=4
   END;
   w:=w SHL Ord(setup);
   _Cfg^.Setup:=_Cfg^.Setup+s
END; { ByteToSetup }

PROCEDURE LoadConfig;
BEGIN
   _Cfg:=StringToAddr;
   _Exec  :=SetupToByte(All   );
   _Count :=SetupToByte(ArRe  );
   _Sort  :=SetupToByte(First );
   _DrawTo:=SetupToByte(Print );
   _Sym   :=SetupToByte(ASCII );
   _Draw  :=SetupToByte(Gb    );
   _C     :=_Cfg^.Color;
   _Conv  :=3;
   CASE _Cfg^.Run OF
      Quit           : _Run:= 0;
      GoAll, GoTrans : _Run:= 4;
      Edit           : _Run:= 9;
      Grb, Grk, Grr  : _Run:=10;
      ConvSD         : BEGIN
                          _Run:=11;
                          _Conv:=0
                       END;
      ConvDS         : BEGIN
                          _Run:=11;
                          _Conv:=1
                       END
   END;
   _DataFile:=Concat(_Cfg^.DPath, _Cfg^.DName, _Cfg^.DExt)
END; { LoadConfig }

PROCEDURE SplitPath(path : PathStr;
                VAR p    : PathStr;
                VAR n    : NameStr;
                VAR e    : ExtStr);
VAR dir : DirStr;
BEGIN
   FSplit(path, dir, n, e);
   p:=dir
END; { SplitPath }

PROCEDURE SaveConfig;
VAR i : BYTE;
BEGIN
   _Cfg^.Output:=0;
   FOR i:=0 TO 12 DO
      IF _Out[i+81]^[3]=#240
         THEN _Cfg^.Output:=_Cfg^.Output  OR (1 SHL i);
   _Cfg^.Setup:=[];
   ByteToSetup(_Exec  , All  );
   ByteToSetup(_Count , ArRe );
   ByteToSetup(_Sort  , First);
   ByteToSetup(_DrawTo, Print);
   ByteToSetup(_Sym   , ASCII);
   ByteToSetup(_Draw  , Gb    )
END; { SaveConfig }

FUNCTION DoMask(DataFile : PathStr;
                cast     : BYTE)    : PathStr;
VAR dir  : DirStr;
    Name : NameStr;
    ext  : ExtStr;
BEGIN
   FSplit(DataFile, dir, Name, ext);
   CASE cast OF
        1, 113, 114 : DoMask:=Concat(dir,      '*.DAT');
        5, 162      : DoMask:=Concat(dir, Name, '.REP');
        7           : DoMask:=Concat(dir, Name, '.TG?');
      101           : DoMask:=Concat(dir, Name, '.BGD');
      102           : DoMask:=Concat(dir, Name, '.BGF');
      103           : DoMask:=Concat(dir, Name, '.BGL');
      141, 161      : DoMask:=Concat(dir, Name, '.DAT');
      142..154      : DoMask:=Concat(dir, Name, '.TG', chr(cast-77));
      163..175      : DoMask:=Concat(dir, Name, '.TG', chr(cast-98))
   END
END; { DoMask }

PROCEDURE OutDoor;
VAR f : FILE OF CFG;
BEGIN
   _DataFile:=DoMask(_DataFile, 1);
   SplitPath(_DataFile, _Cfg^.DPath, _Cfg^.DName, _Cfg^.DExt);
   _Cfg^.Run:=Quit;
   _Cfg^.Ipath:=''; _Cfg^.IName:=''; _Cfg^.IExt:='';
   _Cfg^.Opath:=''; _Cfg^.OName:=''; _Cfg^.OExt:='';
   SaveConfig;
   Assign(f, Concat(_Path, 'BG.CFG'));
   Rewrite(f);
   Write(f, _Cfg^);
   Close(f)
END; { OutDoor }

PROCEDURE MarkOutput;
VAR i : BYTE;
BEGIN
   FOR i:=0 TO 12 DO
      IF Odd(_Cfg^.Output SHR i)
         THEN _Out[i+81]^[3]:=#240
END; { MarkOutput }

FUNCTION TestFile(mask : PathStr) : BOOLEAN;
BEGIN
   IF (mask='') OR (0<Pos('*', mask)) OR (0<Pos('?', mask)) THEN BEGIN
      ErrorMsg(401);
      TestFile:=FALSE
   END
   ELSE IF FileExist(mask) THEN
      TestFile:=TRUE
   ELSE BEGIN
      ErrorMsg(400);
      TestFile:=FALSE
   END
END; { TestFile }

PROCEDURE Stop(path : PathStr;
               run  : RUNTYPE);
BEGIN
   IF TestFile(path)
      THEN BEGIN
         _Cfg^.Run:=run;
         SaveConfig;
         Window(1, 1, 80, 25);
         TextColor(LightGray);
         TextBackGround(Black);
         IF _Cfg^.Run=Edit THEN ClrScr;
         HALT
      END
END; { Stop }

PROCEDURE DisplayMemo(l : BYTE);
VAR win : ScreenStatus;
    i   : BYTE;
BEGIN
```

```
   GetScreenAspect(win);
   SetColor(_Dwfg, _Dwbg);
   gotoxy(Mx1+13, My1+1);
   CASE l OF
      1 : BEGIN
             write(_Cfg^.DName+_Cfg^.DExt);
             FOR i:=length(_Cfg^.DName+_Cfg^.DExt)+1 TO 12 DO write(' ')
          END;
      2 : IF _Exec=0
             THEN write('from the begining')
             ELSE write('translation only ');
      3 : CASE _Count OF
             0 : write('arcs & reproducibility');
             1 : write('arcs                  ');
             2 : write('reproducibility       ')
          END;
      4 : IF _Sort=0
             THEN write('first appearance')
             ELSE write('last appearance ');
      5 : IF _DrawTo=0
             THEN write('printer')
             ELSE write('plotter');
      6 : CASE _Sym OF
             0 : write('ASCII character set     ');
             1 : write('light IBM character set ');
             2 : write('dark IBM character set  ')
          END;
      7 : BEGIN
             write(copy(_Cfg^.InitStr, 1, Mx2-Mx1-14));
             FOR i:=length(_Cfg^.InitStr)+1 TO Mx2-Mx1-14 DO write(' ');
             gotoxy(Mx1+13, My1+l+1);
             write(_Cfg^.width);
             IF _cfg^.Width<100 THEN write(' ')
          END;
      9 : Write(copy(_Set4^, 7, 13))
   END;
   SetScreenAspect(win)
END; { DisplayMemo }

PROCEDURE PrinterSetup;
VAR m : MaskDef;
    w : LONGINT;
BEGIN
   OpenWin(42, 6, 77, 11, _Qwfg, _Qwbg, _Qrfg, _Qrbg, _Qffg, _Qfbg, _QFrame);
   NewMask(m);
   SetSMask(m, @_Cfg^.InitStr, SizeOf(_Cfg^.InitStr)-1,  2, 2, 32, _AllChar, FALSE ,
            26);
   w:=_Cfg^.Width;
   SetIMask(m, @w, 40, 255, 3, 30, 4, 27);
   GotoXY(2, 1); Write('Initialization STRING:');
   GotoXY(2, 4); Write('Width [40..255 characters]:');
   UseMask(m);
   DisposeMask(m);
   CloseWin;
   _Cfg^.Width:=w;
   DisplayMemo(7)
END; { PrinterSetup }

FUNCTION InitString : STRING;
VAR s    : STRING;
    i    : BYTE;
BEGIN
   s:='';
   i:=1;
   WHILE i<=Length(_Cfg^.InitStr) DO BEGIN
      IF _Cfg^.InitStr[i]='^' THEN BEGIN
         Inc(i);
         IF i<=Length(_Cfg^.InitStr) THEN
            s:=s+Chr(Ord(_Cfg^.InitStr[i])-64)
      END
      ELSE
         s:=s+_Cfg^.InitStr[i];
      Inc(i)
   END;
   InitString:=s
END; { InitString }

PROCEDURE PrintFile(path : PathStr);
CONST color : ScreenColor = (LightGray+Blink, Yellow+Blink);
VAR f     : TEXT;
    Line  : STRING;
    error : BYTE;
```

```
BEGIN
   IF TestFile(path) THEN IF PrinterOk
      THEN BEGIN
         DisplayMsg('Printing...');
         HelpLine(0);
         error:=0;
         Assign(f, path);
         Reset(f);
         {$I-} Write(Lst, InitString); {$I+}
         error:=IOResult;
         IF 0<error THEN ErrorMsg(253);
         WHILE (error=0) AND NOT Eof(f) DO
            BEGIN
               Readln(f, Line);
               {$I-} Writeln(Lst, Line); {$I+}
               error:=IOResult;
               IF 0<error THEN ErrorMsg(253);
               IF KeyPressed AND (Inkey=ESC) THEN Error:=255
            END;
         Close(f);
         IF Error=0 THEN {$I-} Write(Lst, ^L); {$I+}
         CloseWin
      END
      ELSE ErrorMsg(253)
END; { PrintFile }

PROCEDURE ModifyGo;
VAR status : ScreenStatus;
BEGIN
   GetScreenAspect(status);
   GotoXY(6, 9);
   Write(_Set4^);
   SetScreenAspect(status)
END; { ModifyGo }

PROCEDURE SetGo;
VAR i : BYTE;
BEGIN
   IF _Exec=0 THEN _Set4^[5]:=#16;
   FOR i:=0 TO 12 DO
      IF (_Cfg^.Output shr i) and 1=1 THEN
         _Set4^[7+i]:=chr(i+65)
END; { SetGo }

{$F+} PROCEDURE DoOption(Action : BYTE); {$F-}
VAR mask : PathStr;
BEGIN
   CASE Action OF
     File
       1: BEGIN
            mask:=DoMask(_DataFile, Action);
            IF GetFileName(mask, 'Data File Name', Action) THEN BEGIN
               _DataFile:=mask;
               SplitPath(_DataFile, _Cfg^.DPath, _Cfg^.DName, _Cfg^.DExt);
               _Set1^:=Copy(_Set1^, 1, 8)+_Cfg^.DName+_Cfg^.DExt;
               DisplayMemo(1)
            END
         END;
     Go
       4: IF _Exec=0 THEN
            Stop(_DataFile, GoAll)
         ELSE
            Stop(_DataFile, GoTrans);
     Report
       5: BEGIN
            mask:=DoMask(_DataFile, Action);
            IF GetFileName(mask, 'Report File Name', Action)
               THEN BEGIN
                  _Sxmin:=17; _Symin:= 2;
                  _Sxmax:=65; _Symax:=23;
                  DisplayText(mask)
               END
         END;
     Edit
       8: IF FileExist(_Editor) THEN BEGIN
            mask:=_DataFile;
            IF GetFileName(mask, 'Edit File Name', Action) THEN BEGIN
               SplitPath(mask, _Cfg^.EPath, _Cfg^.EName, _Cfg^.EExt);
               Stop(mask, Edit)
            END
   END
   ELSE ErrorMsg(410);
Quit
12: OutDoor;
Printer setup
26: PrinterSetup;
From the begining
31: BEGIN
          _Exec:=0;
          _Set4^[5]:=#16;
          _Set21^:=Copy(_Set21^, 1, 9)+_Get31^;
          ModifyGo;
          DisplayMemo(2)
     END;
Translation only
32: BEGIN
          _Exec:=1;
          _Set4^[5]:=' ';
          _Set21^:=Copy(_Set21^, 1, 9)+_Get32^;
          ModifyGo;
          DisplayMemo(2)
     END;
1. Arcs & reproducibility
41: BEGIN
          _Count:=0;
          _Set22^:=Copy(_Set22^, 1, 9)+Copy(_Get41^, 4, 22);
          DisplayMemo(3)
     END;
2. Arcs
42: BEGIN
          _Count:=1;
          _Set22^:=Copy(_Set22^, 1, 9)+Copy(_Get42^, 4, 22);
          DisplayMemo(3)
     END;
3. Reproducibility
43: BEGIN
          _Count:=2;
          _Set22^:=Copy(_Set22^, 1, 9)+Copy(_Get43^, 4, 22);
          DisplayMemo(3)
     END;
First appearance
51: BEGIN
          _Sort:=0;
          _Set23^:=Copy(_Set23^, 1, 9)+_Get51^;
          DisplayMemo(4)
     END;
Last appearance
52: BEGIN
          _Sort:=1;
          _Set23^:=Copy(_Set23^, 1, 9)+_Get52^;
          DisplayMemo(4)
     END;
Draw TO printer
61: BEGIN
          _DrawTo:=0;
          _Set24^:=Copy(_Set24^, 1, 9)+Copy(_Get61^, 4, 7);
          DisplayMemo(5)
     END;
Draw to plotter
62: BEGIN
          _DrawTo:=1;
          _Set24^:=Copy(_Set24^, 1, 9)+Copy(_Get62^, 4, 7);
          DisplayMemo(5)
     END;
ASCII character set
71: BEGIN
          _Sym:=0;
          _Set25^:=Copy(_Set25^, 1, 9)+_Get71^;
          DisplayMemo(6)
     END;
Light IBM character set
72: BEGIN
          _Sym:=1;
          _Set25^:=Copy(_Set25^, 1, 9)+_Get72^;
          DisplayMemo(6)
```

```
    END;
```

Dark IBM character set

```
73: BEGIN
      _Sym:=2;
      _Set25^:=Copy(_Set25^, 1, 9)+_Get73^;
      DisplayMemo(6)
   END;
```

Output

```
81..93: BEGIN
      IF _Out[Action]^[3]=#240
         THEN BEGIN
            _Out[Action]^[3]:=#32;
            _Set4^[Action-74]:=#249
         END
         ELSE BEGIN
            _Out[Action]^[3]:=#240;
            _Set4^[Action-74]:=chr(Action-16)
         END;
      ModifyGo;
      DisplayMemo(9)
   END;
```

Draw G*

```
101: IF _Mouse THEN
      If IsHercules THEN
         ErrorMsg(411)
      ELSE BEGIN
         _Draw:=0;
         mask:=DoMask(_DataFile, Action);
         IF GetFileName(mask, 'G* File Name', Action) THEN
            IF pos('.BGD', mask)=length(mask)-3 THEN BEGIN
               SplitPath(mask, _Cfg^.EPath, _Cfg^.EName, _Cfg^.EExt);
               Stop(_Cfg^.EPath+_Cfg^.Ename+'.BGD', Grb)
            END
            ELSE ErrorMsg(406)
      END
   ELSE ErrorMsg(409);
```

Draw Gk

```
102: IF _Mouse THEN
      If IsHercules THEN
         ErrorMsg(411)
      ELSE BEGIN
         _Draw:=1;
         mask:=DoMask(_DataFile, Action);
         IF GetFileName(mask, 'Gk File Name', Action) THEN
            IF pos('.BGF', mask)=length(mask)-3 THEN BEGIN
               SplitPath(mask, _Cfg^.EPath, _Cfg^.EName, _Cfg^.EExt);
               Stop(_Cfg^.EPath+_Cfg^.Ename+'.BGF', Grk)
            END
            ELSE ErrorMsg(407)
      END
   ELSE ErrorMsg(409);
```

Draw Gk'

```
103: IF _Mouse THEN
      If IsHercules THEN
         ErrorMsg(411)
      ELSE BEGIN
         _Draw:=2;
         mask:=DoMask(_DataFile, Action);
         IF GetFileName(mask, 'Gk'#39'File Name', Action) THEN
            IF pos('.BGL', mask)=length(mask)-3 THEN BEGIN
               SplitPath(mask, _Cfg^.EPath, _Cfg^.EName, _Cfg^.EExt);
               Stop(_Cfg^.EPath+_Cfg^.Ename+'.BGL', Grr)
            END
            ELSE ErrorMsg(408)
      END
   ELSE ErrorMsg(409);
```

Datum file, Samples file

```
111, 112: IF (_Set113^[16]=' ') and (_Set114^[16]=' ') THEN ErrorMsg(402)
   ELSE IF _Set113^[16]=' ' THEN ErrorMsg(403)
   ELSE IF _Set114^[16]=' ' THEN ErrorMsg(404)
   ELSE IF (_Cfg^.Ipath=_Cfg^.Opath) and
           (_Cfg^.Iname=_Cfg^.Oname) and
           (_Cfg^.Iext =_Cfg^.Oext ) THEN ErrorMsg(405)
   ELSE BEGIN
      IF FileExist(_Cfg^.Opath+_Cfg^.Oname+_Cfg^.Oext) and
         not Question('Output file already exists, continue') THEN exit;
      IF Action=111 THEN
         Stop(_Cfg^.Ipath+_Cfg^.Iname+_Cfg^.Iext, ConvSD)
      ELSE
```

```
              Stop(_Cfg^.Ipath+_Cfg^.Iname+_Cfg^.Iext, ConvDS)
        END;
```

Input file

```
      113: BEGIN
            mask:=DoMask(_Cfg^.IPath+_Cfg^.IName+_Cfg^.IExt, Action);
            IF GetFileName(mask, 'Input File Name', Action) THEN BEGIN
               SplitPath(mask, _Cfg^.IPath, _Cfg^.IName, _Cfg^.IExt);
               _Set113^:=Copy(_Set113^, 1, 15)+_Cfg^.IName+_Cfg^.IExt
            END
         END;
```

Output file

```
      114: BEGIN
            mask:=DoMask(_Cfg^.OPath+_Cfg^.OName+_Cfg^.OExt, Action);
            IF GetFileName(mask, 'Output File Name', Action) THEN BEGIN
               SplitPath(mask, _Cfg^.OPath, _Cfg^.OName, _Cfg^.OExt);
               _Set114^:=Copy(_Set114^, 1, 15)+_Cfg^.OName+_Cfg^.OExt
            END
         END;
```

Help index

```
      121..132: DisplayHelp(Action);
```

View

```
      141..154: BEGIN
            mask:=DoMask(_DataFile, Action);
            IF GetFileName(mask, 'View File Name', Action)
               THEN BEGIN
                  _Sxmin:= 3; _Symin:= 2;
                  _Sxmax:=76; _Symax:=22;
                  DisplayText(mask)
               END
         END;
```

Print

```
       161..175: BEGIN
            mask:=DoMask(_DataFile, Action);
            IF GetFileName(mask, 'Print File Name', Action)
               THEN PrintFile(mask)
         END;
   END
END; { DoOption }

BEGIN
   InitFatalError(FatalError);
   LoadConfig;
   Frontispice(_C);
   SetMenu(0, 3, 4, _Run);
      SetOption('File  :              ', 0,  1, Root);
      _Set1:=GetOption;
      _Set1^:=Copy(_Set1^, 1, 8)+_Cfg^.DName+_Cfg^.DExt;
      SetOption('Setup'                ,  1,  2, Root);
      SetOption('Output'               ,  7,  3, Root);
      SetOption(''                     ,  0,  0, Root);
      SetOption('Go (  •••••••••••••)', 0,  4, Root);
      _Set4:=GetOption;
      SetGo;
      SetOption(''                     ,  0,  0, Root);
      SetOption('Report'               ,  0,  5, Root);
      SetOption('View'                 , 11,  6, Root);
      SetOption('Print'                , 12,  7, Root);
      SetOption('Edit'                 ,  0,  8, Root);
      SetOption('Draw'                 ,  8,  9, Root);
      SetOption('Conversion'           ,  9, 10, Root);
      SetOption('Help index'           , 10, 11, Root);
      SetOption(''                     ,  0,  0, Root);
      SetOption('Quit'                 ,  0, 12, GoBack);
   SetMenu(1, 26, 3, 0);
      IF _Exec=0
         THEN SetOption('Execute: From the begining', 2, 21, Stay)
         ELSE SetOption('Execute: Translation only ', 2, 21, Stay);
      _Set21:=GetOption;
      SetOption('', 0, 0, Stay);
      CASE _Count OF
         0 : SetOption('Count  : arcs & reproducibility' , 3, 22, Stay);
         1 : SetOption('Count  : arcs                    ' , 3, 22, Stay);
         2 : SetOption('Count  : reproducibility         ' , 3, 22, Stay)
      END;
      _Set22:=GetOption;
      IF _Sort=0
         THEN SetOption('Sort by: First appearance', 4, 23, Stay)
         ELSE SetOption('Sort by: Last appearance ', 4, 23, Stay);
      _Set23:=GetOption;
   SetOption('', 0, 0, Stay);
   IF _DrawTo=0
      THEN SetOption('Draw TO: printer', 5, 24, Stay)
      ELSE SetOption('Draw TO: plotter', 5, 24, Stay);
   _Set24:=GetOption;
   CASE _Sym OF
      0 : SetOption('Lines  : ASCII character set     ', 6, 25, Stay);
      1 : SetOption('Lines  : Light IBM character set ', 6, 25, Stay);
      2 : SetOption('Lines  : Dark IBM character set  ', 6, 25, Stay)
   END;
   _Set25:=GetOption;
   SetOption('Printer setup', 0, 26, Stay);
SetMenu(2, 35, 5, _Exec);
   SetOption('From the begining', 0, 31, GoBack);
   _Get31:=GetOption;
   SetOption('Translation only ', 0, 32, GoBack);
   _Get32:=GetOption;
SetMenu(3, 35, 7, _Count);
   SetOption('1. arcs & reproducibility', 0, 41, GoBack);
   _Get41:=GetOption;
   SetOption('2. arcs'                  , 0, 42, GoBack);
   _Get42:=GetOption;
   SetOption('3. reproducibility'       , 0, 43, GoBack);
   _Get43:=GetOption;
SetMenu(4, 35, 8, _Sort);
   SetOption('First appearance', 0, 51, GoBack);
   _Get51:=GetOption;
   SetOption('Last appearance ', 0, 52, GoBack);
   _Get52:=GetOption;
SetMenu(5, 32, 10, _DrawTo);
   SetOption('A. printer', 0, 61, GoBack);
   _Get61:=GetOption;
   SetOption('B. plotter', 0, 62, GoBack);
   _Get62:=GetOption;
SetMenu(6, 35, 11, _Sym);
   SetOption('ASCII character set'    , 0, 71, GoBack);
   _Get71:=GetOption;
   SetOption('Light IBM character set', 0, 72, GoBack);
   _Get72:=GetOption;
   SetOption('Dark IBM character set' , 0, 73, GoBack);
   _Get73:=GetOption;
SetMenu(7, 32, 4, 0);
   SetOption('A   Local range charts'                 , 0, 81, Stay);
   _Out[81]:=GetOption;
   SetOption('B   Local maximal horizons'             , 0, 82, Stay);
   _Out[82]:=GetOption;
   SetOption('C   Residual maximal horizons'          , 0, 83, Stay);
   _Out[83]:=GetOption;
   SetOption('D   Biostratigraphic graph (G*)'        , 0, 84, Stay);
   _Out[84]:=GetOption;
   SetOption('E   Maximal cliques OF G*'              , 0, 85, Stay);
   _Out[85]:=GetOption;
   SetOption('F   Graph Gk'                           , 0, 86, Stay);
   _Out[86]:=GetOption;
   SetOption('G   Strongly connected components OF Gk', 0, 87, Stay);
   _Out[87]:=GetOption;
   SetOption('H   Adjacency matrix OF L'#39           , 0, 88, Stay);
   _Out[88]:=GetOption;
   SetOption('I   Numerical ranges'                   , 0, 89, Stay);
   _Out[89]:=GetOption;
   SetOption('J   SORTED UNITARY ASSOCIATIONS'        , 0, 90, Stay);
   _Out[90]:=GetOption;
   SetOption('K   CORRELATION TABLE'                  , 0, 91, Stay);
   _Out[91]:=GetOption;
   SetOption('L   REPRODUCIBILITY OF THE UA'          , 0, 92, Stay);
   _Out[92]:=GetOption;
   SetOption('M   Virtual residual edges'             , 0, 93, Stay);
   _Out[93]:=GetOption;
SetMenu(8, 26, 15, _Draw);
   SetOption('Biostratigraphic graph G*'   , 0, 101, Stay);
   SetOption('Graph OF cliques       Gk'   , 0, 102, Stay);
   SetOption('Reproducibility graph  Gk'#39, 0, 103, Stay);
SetMenu(9, 26, 16, _Conv);
   SetOption('Datum file'                 , 0, 111, Stay);
   SetOption('Samples file'               , 0, 112, Stay);
   SetOption(''                           , 0,   0, Stay);
   SetOption('Input file  :              ', 0, 113, Stay);
   _Set113:=GetOption;
   _Set113^:=Copy(_Set113^, 1, 15)+_Cfg^.IName+_Cfg^.IExt;
   SetOption('Output file :              ', 0, 114, Stay);
   _Set114:=GetOption;
   _Set114^:=Copy(_Set114^, 1, 15)+_Cfg^.OName+_Cfg^.OExt;
```

```
SetMenu(10, 32, 9, 0);
   SetOption('A. Typing your data file'           , 0, 121, Stay);
   SetOption('B. Saving your data file'           , 0, 122, Stay);
   SetOption('C. Technical terminology'           , 0, 123, Stay);
   SetOption('D. Coding the taxa'                 , 0, 124, Stay);
   SetOption('E. Coding the stratigraphic ranges', 0, 125, Stay);
   SetOption('F. Coding the sections'             , 0, 126, Stay);
   SetOption('G. Comments and titles'             , 0, 127, Stay);
   SetOption('H. Syntax and rules'                , 0, 128, Stay);
   SetOption('I. Options and examples'            , 0, 129, Stay);
   SetOption('J. Common errors'                   , 0, 130, Stay);
   SetOption('K. Recommendations'                 , 0, 131, Stay);
   SetOption('L. Bibliography'                    , 0, 132, Stay);
SetMenu(11, 32, 4, 0);
   SetOption('1. Data file'                          , 0, 141, Stay);
   SetOption(''                                      , 0,   0, Stay);
   SetOption('A. Local range charts'                 , 0, 142, Stay);
   SetOption('B. Local maximal horizons'             , 0, 143, Stay);
   SetOption('C. Residual maximal horizons'          , 0, 144, Stay);
   SetOption('D. Biostratigraphic graph (G*)'        , 0, 145, Stay);
   SetOption('E. Maximal cliques OF G*'              , 0, 146, Stay);
   SetOption('F. Graph Gk'                           , 0, 147, Stay);
   SetOption('G. Strongly connected components OF Gk', 0, 148, Stay);
   SetOption('H. Adjacency matrix OF L'#39           , 0, 149, Stay);
   SetOption('I. Numerical ranges'                   , 0, 150, Stay);
   SetOption('J. SORTED UNITARY ASSOCIATIONS'        , 0, 151, Stay);
   SetOption('K. CORRELATION TABLE'                  , 0, 152, Stay);
   SetOption('L. REPRODUCIBILITY OF THE UA'          , 0, 153, Stay);
   SetOption('M. Virtual residual edges'             , 0, 154, Stay);
SetMenu(12, 32, 4, 0);
   SetOption('1. Data file'                          , 0, 161, Stay);
   SetOption('2. Report file'                        , 0, 162, Stay);
   SetOption(''                                      , 0,   0, Stay);
   SetOption('A. Local range charts'                 , 0, 163, Stay);
   SetOption('B. Local maximal horizons'             , 0, 164, Stay);
   SetOption('C. Residual maximal horizons'          , 0, 165, Stay);
   SetOption('D. Biostratigraphic graph (G*)'        , 0, 166, Stay);
   SetOption('E. Maximal cliques OF G*'              , 0, 167, Stay);
   SetOption('F. Graph Gk'                           , 0, 168, Stay);
   SetOption('G. Strongly connected components OF Gk', 0, 169, Stay);
   SetOption('H. Adjacency matrix OF L'#39           , 0, 170, Stay);
   SetOption('I. Numerical ranges'                   , 0, 171, Stay);
   SetOption('J. SORTED UNITARY ASSOCIATIONS'        , 0, 172, Stay);
   SetOption('K. CORRELATION TABLE'                  , 0, 173, Stay);
   SetOption('L. REPRODUCIBILITY OF THE UA'          , 0, 174, Stay);
   SetOption('M. Virtual residual edges'             , 0, 175, Stay);
MarkOutput;
_UserError:=Warnings;
_Action:=DoOption;
_HelpPath:=Concat(_Path, 'BG.HLP');
_Hxmin:=23;
_Hymin:=12;
_Hxmax:=76;
_Hymax:=22;
_HelpMsg[0]:='ESC: abort';

SetBackScreen(0,0,'');
SetColor(ffCopy, wbCopy);
WrRepChar(47, 2, #223, 30);
WrRepChar(47, 7, #220, 30);
TextColor(wfCopy[_C]);
GotoXY(47,3); Write('   BIOGRAPH :  BG_MENU 2.02   ');
GotoXY(47,4); Write(' (c) 1990,  J.Savary & J.Guex ');
GotoXY(47,5); Write('     Institute OF Geology     ');
GotoXY(47,6); Write('    University OF Lausanne    ');

FillBox(Mx1, My1, Mx2, My2, (_Dwfg[_C] shl 8)+(_Dwbg[_C] shl 12)+$20);
ColorBox(Mx1+2, My2+1, Mx2+2, My2+1, $07);
ColorBox(Mx2+1, My1+1, Mx2+2, My2  , $07);
SetColor(_Dwfg, _Dwbg);
gotoxy(Mx1+2, My1+1); write('Data file:');
gotoxy(Mx1+2, My1+2); write('Execute   :');
gotoxy(Mx1+2, My1+3); write('Count     :');
gotoxy(Mx1+2, My1+4); write('Sort by   :');
gotoxy(Mx1+2, My1+5); write('Draw TO   :');
gotoxy(Mx1+2, My1+6); write('Lines     :');
gotoxy(Mx1+2, My1+7); write('Init      :');
gotoxy(Mx1+2, My1+8); write('Width     :');
gotoxy(Mx1+2, My1+9); write('Output    :');
DisplayMemo(1);
DisplayMemo(2);
DisplayMemo(3);
```

```
   DisplayMemo(4);
   DisplayMemo(5);
   DisplayMemo(6);
   DisplayMemo(7);
   DisplayMemo(9);
   MenuWin(0)
END.
```

## *PROGRAM BG_DATA;*

Function :   syntax analysis and compilation of data

```
{$A+,B-,D-,E+,F-,G+,I-,L-,N-,O-,R-,S+,V-,X+}
{$M 65520,0,655360} {v2.20}
USES BG_Def, Crt, Dos, Error, GaugeWin, Math, {Screen,} Tools, TP_Win_E;
type
   SectionType  = array[1..1000] of DataSet;
   CardinalList = array[1..1000] of word;
   ActionType   = (binary, datum, samples);

const
   TOKEN          : AnyChar
                  = [#0..#32, '!', '&', #39, '(', ')', '*', '+',
                     ',', '-', '.', '/', ':', ';', '=', '>', '?',
                     '[', '\', ']', '^', '`', '|', '}', '~', #127];
   BEGINCOMMENT = '{';
   ENDCOMMENT   = '}';
   STRINGTOKEN  = '"';

   _NSections : word    = 0;
   _NLevels   : word    = 0;
   _NTaxa     : word    = 0;
   _Pos       : Longint = 0;

VAR
   _F        : FileOfChar;
   _Line     : Word;
   _Datum    : boolean;
   _Sections : SectionList;
   _Taxa     : TaxaList;
   _Title    : string;
   _Section  : ^SectionType;
   _Cardinal : CardinalList;

{$F+} function ErrorMsg(error : word) : Str74; {$F-}
var msg : Str64;
begin
   case error of                                          {|}
       50 : msg:='invalid command line';
      301 : msg:='DATUM/SAMPLES missing, or not an ASCII file';
      302 : msg:='too long title (>253 characters)';
      303 : msg:='SECTION missing';
      304 : msg:='section'#39's identifier missing';
      305 : msg:='duplicate section'#39's identifer';
      306 : msg:='too many sections (>1000)';
      307 : msg:='invalid numeric format or number missing';
      308 : msg:='invalid level'#39's number (1<=n<=1000)';
      309 : msg:='BOTTOM missing';
      310 : msg:='TOP missing';
      311 : msg:='bottom'#39's number greater than top'#39's number';
      312 : msg:='taxa'#39's identifier missing';
      313 : msg:='too many taxa (>500)';
      314 : msg:='duplicate taxa'#39's identifier in the section';
      315 : msg:='Fad out of range';
      316 : msg:='Lad out of range';
      317 : msg:='Fad greater than Lad';
      318 : msg:='empty section';
      319 : msg:='< missing';
      320 : msg:='duplicate horizon';
      321 : msg:='horizon out of range';
      322 : msg:='duplicate taxa'#39's identifier in the horizon';
      323 : msg:='empty horizon';
   else ErrorMsg:=''
   end;
   if msg='' then
      ErrorMsg:=''
   else
      ErrorMsg:='line '+IStr(_Line, 0)+', '+msg
```

```
end; { ErrorMsg }

{$F+} procedure FatalError(error : integer); {$F-}
begin
   DisplayMsg(ErrorStr(error));
   Tools.Pause;
   ClearMsg
end; { FatalError }

procedure Initialization;
var cmd : ComStr;
begin
   cmd:=paramstr(1);
   if cmd[1]='-' then begin
      new(_Cfg);
      case cmd[2] of
         'd','D' : _Cfg^.Run:=ConvSD;
         's','S' : _Cfg^.Run:=ConvDS
      else _Cfg^.Run:=GoAll
      end;
      if _Cfg^.Run=GoAll then begin
         _Cfg^.Dpath:='';
         _Cfg^.Dname:=paramstr(2);
         _Cfg^.Dext :=paramstr(3)
      end
      else begin
         _Cfg^.Ipath:='';
         _Cfg^.Iname:=paramstr(2);
         _Cfg^.Iext :=paramstr(3);
         _Cfg^.Opath:='';
         _Cfg^.Oname:=paramstr(4);
         _Cfg^.Oext :=paramstr(5)
      end
   end
   else begin
      _Cfg:=StringToAddr;
      _C:=_Cfg^.color
   end;
   SetBackScreen(0, 0, '');
   if _C then begin
      textcolor(white);
      textbackground(magenta)
   end
   else begin
      textcolor(black);
      textbackground(white)
   end;
   gotoxy(9, 2);
   Write(' B I O G R A P H - BG_DATA v2.20 (c) 90-96 by J.Savary &
J.Guex ');
   gotoxy(9, 3);
   Write(' Institute of Geology,   University of Lausanne,
Switzerland ');
   OpenGauge(15, 6);
   if _Cfg^.Run=GoAll then
      InitGauge(2*TextSize(_Cfg^.Dpath+_Cfg^.Dname+_Cfg^.Dext))
   else
      InitGauge(2*TextSize(_Cfg^.Ipath+_Cfg^.Iname+_Cfg^.Iext));
   OpenWin(17, 10, 64, 23, _Dwfg, _Dwbg, _Drfg, _Drbg, _Dffg, _Dfbg,
_MFrame);
   gotoxy(2, 1);
   if _Cfg^.Run=GoAll then
      write(_Cfg^.Dname, _Cfg^.Dext, ' => ', _Cfg^.Dname, '.BGA')
   else
      write(_Cfg^.Iname, _Cfg^.Iext, ' => ', _Cfg^.Oname, _Cfg^.Oext);
   gotoxy(2,  5); write('Line');
   gotoxy(2,  7); write('Section       :');
   gotoxy(2,  8); write('Top horizon   :');
   gotoxy(2,  9); write('Bottom horizon:');
   gotoxy(2, 10); write('Taxa          :')
end; { Initialization }

procedure DisplayStr(l : byte;
                     s : string);
begin
   case l of
      1 : gotoxy(18,  7);
      2 : gotoxy(18, 10)
   end;
   write(copy(s, 1, 25));
   clreol;
end; { DisplayStr }
```

```
procedure DisplayNum(l : byte;
                     n : word);
begin
   case l of
      1 : gotoxy(18,  8);
      2 : gotoxy(18,  9);
      3 : gotoxy(18, 11);
      4 : gotoxy(18, 12)
   end;
   write(n);
   clreol;
end; { DisplayNum }

procedure RdF(var item : string);
begin
   item:=RdItem(_F, TOKEN, BEGINCOMMENT, ENDCOMMENT, STRINGTOKEN, _Line,
_Pos);
   Gauge(_Pos);
   if item[1]<>'"' then begin
      gotoxy(6, 5);
      if item='' then
         write(_Line:10, ': end of file')
      else
         write(_Line:10, ': ', copy(item, 1, 25));
      clreol;
      item:=UpString(item);
   end
end; { RdF }

procedure RdFNumber(var n : word);
var item  : string;
    error : integer;
begin
   RdF(item);
   Gauge(_Pos);
   Val(item, n, error);
   if error<>0 then halt(307);
   if (n<1) or (1000<n) then halt(308)
end; { RdFNumber }

function Keyword(item : string) : boolean;
const KEYWORDS : array[0..6] of string[7]
               = ('DATUM', 'SAMPLES', 'TITLE', 'SECTION', 'BOTTOM',
'TOP', '<');
var i : byte;
begin
   i:=0;
   while (i<=6) and (item<>KEYWORDS[i]) do inc(i);
   Keyword:=i<=6
end; { Keyword }

function OldSection(item : string) : boolean;
var i : word;
begin
   i:=1;
   while (i<=_NSections) and (item<>_Sections[i]) do inc(i);
   OldSection:=i<=_NSections
end; { OldSection }

function AddTaxa(item : string;
             var taxa : TaxaList;
             var n    : word) : boolean;
var i : word;
begin
   i:=1;
   while (i<=n) and (item<>taxa[i]) do inc(i);
   if i<=n then
      AddTaxa:=false
   else begin
      AddTaxa:=true;
      if n=500 then halt(313);
      inc(n);
      taxa[i]:=item
   end
end; { AddTaxa }

{$F+} function Less(a, b : integer) : boolean; {F-}
begin
   Less:=_Taxa[a]<_Taxa[b]
end; { Less }
```

```
{$F+} procedure Swap(a, b : integer); {F-}
var buffer : { Str5} Str25;
begin
   buffer:=_Taxa[a];
   _Taxa[a]:=_Taxa[b];
   _Taxa[b]:=buff§er
end; { Swap }

procedure Analyze;
var item                          : string;
    bottom, top, ntaxa, fad, lad,
    level, nlevels, min, max      : word;
    endsection, endlevel          : boolean;
    taxa                          : TaxaList;
    levels                        : array[1..1000] of boolean;
begin
   gotoxy(2, 2); write('analyze...');
   if _Cfg^.Run=GoAll then
      assign(_F, _Cfg^.Dpath+_Cfg^.Dname+_cfg^.Dext)
   else
      assign(_F, _Cfg^.Ipath+_Cfg^.Iname+_cfg^.Iext);
   reset(_F);
   _Line:=1;

   { type of data}

   RdF(item);
   if item='DATUM' then begin
       gotoxy(2,  3); write('DATUM file');
       gotoxy(2, 11); write('FAD level     :');
       gotoxy(2, 12); write('LAD level     :');
      _Datum:=true
   end
   else if item='SAMPLES' then begin
       gotoxy(2,  3); write('SAMPLES file');
       gotoxy(2, 11); write('Horizon       :');
      _Datum:=false
   end
   else halt(301);

   { titre }

   RdF(item);
   if item='TITLE' then begin
      RdF(item);
      if item[1]='"' then begin
         _Title:=copy(item, 2, 255);
         if _Title[length(_Title)]='"' then
            _Title[0]:=chr(length(_Title)-1)
         else
            halt(302);
         RdF(item)
      end
   end
   else _Title:='';

   { sections }

   repeat

      {heading of section }

      if item<>'SECTION' then halt(303);
      RdF(item);
      if Keyword(item) then halt(304);
      if OldSection(item) then halt(305);
      if _NSections=1000 then halt(306);
      inc(_NSections);
      _Sections[_NSections]:=item;
      DisplayStr(1, item);

      RdF(item);
      if item<>'BOTTOM' then halt(309);
      RdFNumber(bottom);
      DisplayNum(2, bottom);
      RdF(item);
      if item<>'TOP' then halt(310);
      RdFNumber(top);
      DisplayNum(1, top);
      if top<bottom then halt(311);

      { content of section under DATUM format}
      fillchar(levels, sizeof(levels), 0);
      if _Datum then begin
         ntaxa:=0;
         min:=1001;
         max:=0;
         repeat
            RdF(item);
            endsection:=((item='') or (item='SECTION'));
            if not endsection then begin
               if Keyword(item) then halt(312);
               DisplayStr(2, item);
               if AddTaxa(item, _Taxa, _NTaxa) then;
               if not AddTaxa(item, taxa, ntaxa) then halt(314);
               RdFNumber(fad);
               DisplayNum(3, fad);
               RdFNumber(lad);
               DisplayNum(4, lad);
               if (fad<bottom) or (top<fad) then halt(315);
               if (lad<bottom) or (top<lad) then halt(316);
               if lad<fad then halt(317);
               if fad<min then min:=fad;
               if max<lad then max:=lad;
               for level:=fad to lad do levels[level]:=true
            end
         until endsection;
         if ntaxa=0 then halt(318);
         inc(_NLevels, max-min+1);
         for level:=min+1 to max-1 do
            if not levels[level] then dec(_NLevels, 1)
      end

      { content of section under SAMPLES  format}

      else begin
         nlevels:=0;
         RdF(item);
         repeat
            endsection:=((item='') or (item='SECTION'));
            if not endsection then begin
               inc(nlevels);
               if item<>'<' then halt(319);
               RdFNumber(level);
               DisplayNum(3, level);
               if levels[level] then
                  halt(320)
               else
                  levels[level]:=true;
               if (level<bottom) or (top<level) then halt(321);
               ntaxa:=0;
               repeat
                  RdF(item);
                  endlevel:=((item='<') or (item='') or
(item='SECTION'));
                  if not endlevel then begin
                     DisplayStr(2, item);
                     if Keyword(item) then halt(312);
                     if AddTaxa(item, _Taxa, _NTaxa) then;
                     if not AddTaxa(item, taxa, ntaxa) then halt(322)
                  end
               until endlevel;
               if ntaxa=0 then halt(323)
            end
         until endsection;
         if nlevels=0 then halt(318);
         inc(_NLevels, nlevels)
      end

   until item='';
   close(_F);
   HSort(Less, Swap, _NTaxa)
end; { Analyze }

procedure DisplayResult;
var i : byte;
begin
   gotoxy(2, 7); write(_NSections:5, ' sections'); clreol;
   gotoxy(2, 8); write(_NLevels:5, ' horizons'); clreol;
   gotoxy(2, 9); write(_NTaxa:5, ' taxa'); clreol;
   for i:=10 to 12 do begin
      gotoxy(2, i);
      clreol
   end;
   gotoxy(1,11);
   for i:=1 to IMin(length(_Title), 46) do
      if _Title[i] in [#0..#31] then write(' ') else write(_Title[i])
end; { DisplayResult }

procedure Lexicon;
var f : file;
begin
   assign(f, _Cfg^.Dpath+_Cfg^.Dname+'.LEX');
   rewrite(f, 1);
   blockwrite(f, _Title, sizeof(_Title));
   blockwrite(f, _NSections, sizeof(_NSections));
   blockwrite(f, _NTaxa, sizeof(_NTaxa));
   blockwrite(f, _Sections, sizeof(Str25)*_NSections);
   blockwrite(f, _Taxa, sizeof({Str5}Str25)*_NTaxa);
   close(f)
end; { Lexicon }

procedure Report;
var f  : text;
    t1 : longint;
    t2 : DateTime;
    i  : word;

   function N(v : word) : Str3;
   var s : Str3;
   begin
      str(v:2, s);
      if s[1]=' ' then s[1]:='0';
      N:=s
   end; { N }

begin
   gotoxy(9, 2); write(', report...');
   assign(f, _Cfg^.Dpath+_Cfg^.Dname+_Cfg^.Dext);
   reset(f);
   getftime(f, t1);
   close(f);
   unpacktime(t1, t2);
   assign(f, _Cfg^.Dpath+_Cfg^.Dname+'.REP');
   rewrite(f);
   writeln(f, 'BIOGRAPH v2.20 (c) 1990-96, J.Savary & J.Guex');
   writeln(f, '');
   if _Datum then write(f, 'Datum file: ':14) else write(f, 'Samples
file: ':14);
   writeln(f, _Cfg^.Dpath, _Cfg^.Dname, _Cfg^.Dext);
   writeln(f, 'created: ':14, N(t2.day), '/', N(t2.month), '/',
                           N(t2.year mod 100), '  ', N(t2.hour),
                           ':', N(t2.min));
   getdate(t2.year, t2.month, t2.day, t2.sec);
   gettime(t2.hour, t2.min, t2.sec, t2.sec);
   writeln(f, 'calculated: ':14, N(t2.day), '/', N(t2.month), '/',
                              N(t2.year mod 100), '  ', N(t2.hour),
                              ':', N(t2.min));
   getdate(t2.year, t2.month, t2.day, t2.sec);
   gettime(t2.hour, t2.min, t2.sec, t2.sec);
   writeln(f, '');
   writeln(f, _Title);
   writeln(f, '');
   writeln(f, _NSections:4, ' sections:', 1:4, ' ', _Sections[1]);
   for i:=2 to _NSections do writeln(f, i:18, ' ', _Sections[i]);
   writeln(f, '');
   writeln(f, _NTaxa:4, ' taxa:', 1:4, ' ', _Taxa[1]);
   for i:=2 to _NTaxa do writeln(f, i:14, ' ', _Taxa[i]);
   writeln(f, '');
   writeln(f, _NLevels, ' horizons');
   close(f)
end; { Report }

procedure PutInSet(data, level : word);
begin
   dec(data);
   _Section^[level, data div 256]:=_Section^[level, data div 256]+[data
mod 256];
   inc(_Cardinal[level])
end; { PutInSet }

function FindTaxa(taxa : {Str5}Str25) : word;
var i : word;
begin
   i:=0;
   repeat inc(i) until _Taxa[i]=taxa;
   FindTaxa:=i
end; { FindTaxa }

procedure WriteSectionB(var f                          : file;
                            size, section, bottom, top : word);
var level : word;
begin
   for level:=top downto bottom do
      if _Cardinal[level]<>0 then begin
         blockwrite(f, section, sizeof(section));
         blockwrite(f, level, sizeof(level));
         blockwrite(f, _Cardinal[level], sizeof(_Cardinal[level]));
         blockwrite(f, _Section^[level], size)
      end
end; { WriteSectionB }

procedure WriteSectionD(var t           : text;
                            bottom, top : word);
var level, fad, lad, taxa, i, j : word;
begin
   for taxa:=1 to _NTaxa do begin
      i:=(taxa-1) div 256;
      j:=(taxa-1) mod 256;
      level:=bottom;
      while (level<=top) and (not (j in _Section^[level, i])) do
inc(level);
      if level<=top then begin
         fad:=level;
         level:=top;
         while not (j in _Section^[level, i]) do dec(level);
         lad:=level;
         writeln(t, _Taxa[taxa]:{v2.20}SizeOf(Str25)-1, ':', fad:4, '-',
lad:4)
      end
   end
end; { WriteSectionD }

procedure WriteSectionS(var t           : text;
                            bottom, top : word);
var level, taxa : word;
    new         : boolean;
    len         : byte;
begin
   for level:=top downto bottom do
      if _Cardinal[level]<>0 then begin
         write(t, '<', level:4, ': ');
         new:=true;
         len:=5;
         for taxa:=0 to _NTaxa-1 do
            if (taxa mod 256) in _Section^[level, taxa div 256] then
begin
               if len=75 then begin
                  writeln(t, '');
                  write(t, '       ');
                  new:=true;
                  len:=5
               end;
               if new then new:=false else write(t, ', ');
               write(t, _Taxa[taxa+1]:{v2.20}SizeOf(Str25)-1);
               inc(len, 7)
            end;
         writeln(t, '')
      end
end; { WriteSectionS }

procedure FillRange(bottom, top : word);
var level, fad, lad, taxa : word;
    i, j                  : byte;
begin
   for taxa:=1 to _NTaxa do begin
      i:=(taxa-1) div 256;
      j:=(taxa-1) mod 256;
      level:=bottom;
      while (level<=top) and (not (j in _Section^[level, i])) do
inc(level);
      if level<=top then begin
         fad:=level;
         level:=top;
         while not (j in _Section^[level, i]) do dec(level);
         lad:=level;
         for level:=fad to lad do
```

```
            if (_Cardinal[level]<>0) AND
               (not (j in _Section^[level, i])) then begin
               _Section^[level, i]:=_Section^[level, i]+[j];
               inc(_Cardinal[level])
            end
      end
   end
end; { FillRange }

procedure Compile;
var item, section                                      : string;
    bottom, top, fad, lad, level, taxa, size, min, max : word;
    endsection, endlevel                               : boolean;
    f, {v2.10} g                                       : file;
    t                                                  : text;
begin
   if _Cfg^.Run=GoAll then begin
      gotoxy(17, 2); write(', compile...');
      assign(f, _Cfg^.Dpath+_Cfg^.Dname+'.BGA');
      rewrite(f, 1);
      blockwrite(f, _NTaxa, sizeof(_NTaxa));
      blockwrite(f, _NSections, sizeof(_NSections));
      blockwrite(f, _NLevels, sizeof(_NLevels));
{v2.10-}
      assign(g, _Cfg^.Dpath+_Cfg^.Dname+'.BGN');
      rewrite(g, 1);
      blockwrite(g, _NTaxa, sizeof(_NTaxa));
      blockwrite(g, _NSections, sizeof(_NSections));
      blockwrite(g, _NLevels, sizeof(_NLevels))
{-v2.10}
   end
   else begin
      assign(t, _Cfg^.Opath+_Cfg^.Oname+_Cfg^.Oext);
      rewrite(t);
      if _Cfg^.Run=ConvSD then begin
         gotoxy(9, 2); write(', translate into datum...');
         writeln(t, 'DATUM')
      end
      else begin
         gotoxy(9, 2); write(', translate into samples...');
         writeln(t, 'SAMPLES')
      end;
      if 0<length(_Title) then begin
         writeln(t, '');
         writeln(t, 'TITLE:');
         writeln(t, '"', _Title, '"')
      end
   end;

   new(_Section);
   if _Cfg^.Run=GoAll then
      assign(_F, _Cfg^.Dpath+_Cfg^.Dname+_cfg^.Dext)
   else
      assign(_F, _Cfg^.Ipath+_Cfg^.Iname+_cfg^.Iext);
   reset(_F);
   _Line:=1;
   size:=LevelSize(_NTaxa)-sizeof(HEAD_REC);

   { passing by the heading}

   repeat RdF(item) until item='SECTION';

   { sections }

   _NSections:=0;
   repeat

      {heading of section }

      RdF(section);
      inc(_NSections);
      RdF(item);
      RdFNumber(bottom);
      RdF(item);
      RdFNumber(top);
      fillchar(_Section^[bottom], (top-bottom+1)*sizeof(DataSet), 0);
      fillchar(_Cardinal[bottom], (top-bottom+1)*sizeof(word), 0);
      if _Cfg^.Run<>GoAll then begin
         writeln(t, '');
         writeln(t, 'SECTION ', section, ': bottom ', bottom, ' - top ',
top)
      end;
      { content of section under DATUM format}

      if _Datum then begin
         min:=1001;
         max:=0;
         repeat
            RdF(item);
            endsection:=((item='') or (item='SECTION'));
            if not endsection then begin
               taxa:=FindTaxa(item);
               RdFNumber(fad);
               RdFNumber(lad);
               for level:=fad to lad do PutInSet(taxa, level);
               if fad<min then min:=fad;
               if max<lad then max:=lad
            end
         until endsection;
         bottom:=min;
         top:=max
      end

      { content of section under SAMPLES format}

      else begin

         RdF(item);
         repeat
            endsection:=((item='') or (item='SECTION'));
            if not endsection then begin
               RdFNumber(level);
               repeat
                  RdF(item);
                  endlevel:=((item='<') or (item='') or
(item='SECTION'));
                  if not endlevel then PutInSet(FindTaxa(item), level)
               until endlevel
            end
         until endsection;
{<v2.10  FillRange(bottom, top)}
      end;
{<v2.10
      case _Cfg^.Run of
         GoAll  : WriteSectionB(f, size, _NSections, bottom, top);
         ConvSD : WriteSectionD(t, bottom, top);
         ConvDS : WriteSectionS(t, bottom, top)
      end
}
{v2.10-}
      case _Cfg^.Run of
         GoAll  : begin
                     if not _Datum then begin
                        WriteSectionB(g, size, _NSections, bottom, top);
                        FillRange(bottom, top)
                     end;
                     WriteSectionB(f, size, _NSections, bottom, top);
                  end;
         ConvSD : WriteSectionD(t, bottom, top);
         ConvDS : WriteSectionS(t, bottom, top)
      end
{-v2.10}
   until item='';
   close(_F);
   if _Cfg^.Run=GoAll then begin
      close(f);
      close(g)
   end
   else
      close(t)
end; { Compile }

begin
   CheckBreak:=true;
   _UserError:=ErrorMsg;
   InitFatalError(FatalError);
   Initialization;
   Analyze;
   DisplayResult;
   if _Cfg^.Run=GoAll then begin
      Lexicon;
      Report
   end;
   Compile
end. { BG_DATA }
```

## *Program BG_CONV*

Function : conversion of the binary files into text files

```
PROGRAM BG_CONV;

{$A+,B-,D-,E+,F-,G+,I-,L-,N-,O-,R-,S+,V-,X+}

USES BG_Def, Crt, Dos, Error, GaugeWin, Math, Screen, Tools, TP_Win_E;

const
   _Li : ARRAY[0..2, 1..14] OF char
      = (('!', '-', '+', '+', '+', '+', '+', '+', '+', '+', '+', 'X', '.', 'X'),
         ('_', '-', '+', '+', '+', '+', '+', '_', '-', '-', '+', '_', '•', '_'),
         ('_', '-', '+', '+', '+', '+', '+', '_', '-', '-', '+', '_', '•', '_'));
         { 1    2    3    4    5    6    7    8    9   10   11   12   13   14 }
VAR
   _Title                : STRING;
   _NSections, _NTaxa    : WORD;
   _Sections             : SectionLPtr;
   _Taxa                 : TaxaLPtr;
   _T                    : text;
   _L, _W                : BYTE;

{$F+} FUNCTION ErrorMsg(error : WORD) : Str74; {$F-}
VAR msg : Str64;
BEGIN
   CASE error OF                                              {!}
      50 : msg:='invalid command line';
   ELSE ErrorMsg:=''
   END
END; { ErrorMsg }

{$F+} PROCEDURE FatalError(error : INTEGER); {$F-}
BEGIN
   DisplayMsg(ErrorStr(error));
   Tools.Pause;
   ClearMsg
END; { FatalError }

PROCEDURE Initialization;
VAR cmd : ComStr;
BEGIN
   cmd:=paramstr(1);
   IF cmd[1]='-' THEN BEGIN
      new(_Cfg);
      fillchar(_Cfg^, sizeof(_Cfg^), 0);
      IF upcase(cmd[2]) in ['A'..'M'] THEN
         _Cfg^.Output:=1 shl (ord(cmd[2])-65)
      ELSE halt(50);
      _Cfg^.Dname:=paramstr(2);
      _Cfg^.Width:=80;
      _L:=1
   END
   ELSE BEGIN
      _Cfg:=StringToAddr;
      _C:=_Cfg^.color;
      IF ASCII in _Cfg^.Setup THEN
         _L:=0
      ELSE IF Light in _Cfg^.Setup THEN
         _L:=1
      ELSE
         _L:=2
   END;
   SetBackScreen(0, 0, '');
   IF _C THEN BEGIN
      textcolor(white);
      textbackground(magenta)
   END
   ELSE BEGIN
      textcolor(black);
      textbackground(white)
   END;
   gotoxy(9, 2);
   Write(' B I O G R A P H - BG_CONV v2.02  (c) 1990 by J.Savary & J.Guex ');
   gotoxy(9, 3);
   Write(' Institute OF Geology,   University OF Lausanne,    Switzerland ');
   OpenGauge(15, 6);
   OpenWin(17, 10, 64, 23, _Dwfg, _Dwbg, _Drfg, _Drbg, _Dffg, _Dfbg, _MFrame)
```

```
END; { Initialization }

PROCEDURE Wr(s : STRING);
BEGIN
   gotoxy(2, wherey);
   writeln(s)
END; { Wr }

PROCEDURE LoadLexicon;
VAR f : file;
    t : WORD;
BEGIN
   Wr('Read lexicon');
   assign(f, _Cfg^.Dpath+_Cfg^.Dname+'.LEX');
   reset(f, 1);
   blockread(f, _Title, sizeof(_Title));
   blockread(f, _NSections, sizeof(_NSections));
   blockread(f, _NTaxa, sizeof(_NTaxa));
   getmem(_Sections, _NSections*sizeof(Str25));
   blockread(f, _Sections^, _NSections*sizeof(Str25));
   getmem(_Taxa, _NTaxa*sizeof(Str5));
   blockread(f, _Taxa^, _NTaxa*sizeof(Str5));
   close(f);
   FOR t:=1 TO _NTaxa DO
      CASE length(_Taxa^[t]) OF
         1 : _Taxa^[t]:='    '+_Taxa^[t];
         2 : _Taxa^[t]:='   ' +_Taxa^[t];
         3 : _Taxa^[t]:='  '  +_Taxa^[t];
         4 : _Taxa^[t]:=' '   +_Taxa^[t]
      END
END; { LoadLexicon }

FUNCTION LoadSource(VAR f : file) : boolean;
VAR Ptr   : POINTER;
    count : LONGINT;
    read  : WORD;
p:levelptr;
BEGIN
   count:=filesize(f);
   IF _UserMemory<count THEN BEGIN
      Wr('Not enough memory');
      LoadSource:=FALSE;
      exit
   END;
   Ptr:=_UserHeap;
   WHILE $4000<count DO BEGIN
      BlockRead(f, Ptr^, $4000, read);
      IF read<$4000 THEN BEGIN
         close(f);
         Wr('Not loaded');
         LoadSource:=FALSE;
         exit
      END;
      dec(count, $4000);
      Ptr:=AddAddr(Ptr, $4000)
   END;
   BlockRead(f, Ptr^, count, read);
   close(f);
   IF read<count THEN BEGIN
      Wr('Not loaded');
      LoadSource:=FALSE
   END
   ELSE
      LoadSource:=TRUE
END; { LoadSource }

FUNCTION NotExist(c : char) : boolean;
VAR f : file;
BEGIN
   IF FileExist(_Cfg^.Dpath+_Cfg^.Dname+'.BG'+c) THEN BEGIN
      IF FileExist(_Cfg^.Dpath+_Cfg^.Dname+'.TG'+c) THEN
         Wr(_Cfg^.Dname+'.BG'+c+'  =>  '+ _Cfg^.Dname+'.TG'+c+' overwrite')
      ELSE
         Wr(_Cfg^.Dname+'.BG'+c+'  =>  '+ _Cfg^.Dname+'.TG'+c+' create');
      Assign(f, _Cfg^.Dpath+_Cfg^.Dname+'.BG'+c);
      reset(f, 1);
      IF not LoadSource(f) THEN exit;
      Assign(_T, _Cfg^.Dpath+_Cfg^.Dname+'.TG'+c);
      rewrite(_T);
      IF c='I' THEN writeln(_T, '{');
      Writeln(_T, 'BIOGRAPH v2.02');
      Writeln(_T, '(c) 1990 by J.Savary & J.Guex');
      Writeln(_T, '');
      IF c='I' THEN BEGIN
         writeln(_T, 'Numerical ranges OF the taxa in UA (TGJ)');
         writeln(_T, '}')
      END
      ELSE
         IF length(_Title)<>0 THEN
            writeln(_T, _Title)
         ELSE
            writeln(_T, 'Untitled');
      writeln(_T, '');
      NotExist:=FALSE
   END
   ELSE BEGIN
      Wr(_Cfg^.Dname+'.BG'+c+' not found !');
      NotExist:=TRUE
   END
END; { NotExist }

FUNCTION Inset(t : WORD;
               s : LevelPtr) : boolean;
BEGIN
   dec(t);
   Inset:=(t mod 256) in s^.Taxa[t div 256]
END; { Inset }

FUNCTION Dont(a1, a2, b1, b2 : WORD) : boolean;
BEGIN
   IF (a1=a2) and (b1=b2) THEN
      Dont:=FALSE
   ELSE BEGIN
      Wr('Incompatible files');
      Close(_T);
      erase(_T);
      Dont:=TRUE
   END
END; { Dont }

PROCEDURE bgA(c : char);
VAR s, t, nc, p, np, i, j, size : WORD;
    g                           : LONGINT;
    sptr, ptr                   : LevelPtr;
BEGIN
   IF NotExist(c) THEN exit;
   sptr:=_UserHeap;
   nc:=(_Cfg^.Width-12) div 2;
   np:=_NTaxa div nc;
   IF _NTaxa mod nc<>0 THEN inc(np);
   InitGauge(sptr^.Cardinal*np);
   IF Dont(sptr^.Section, _NTaxa, sptr^.Level, _NSections) THEN exit;
   Writeln(_T, 'Local range charts');
   Writeln(_T, '');
   Writeln(_T, sptr^.Section:4 , ' taxa');
   Writeln(_T, sptr^.Level:4   , ' sections');
   Writeln(_T, sptr^.Cardinal:4, ' horizons');
   sptr:=AddAddr(sptr, 6);
   size:=LevelSize(_Ntaxa);
   g:=0;
   FOR s:=1 TO _NSections DO BEGIN
      writeln(_T, '');
      writeln(_T, 'Section ', _Sections^[s]);
      t:=1;
      FOR p:=1 TO np DO BEGIN
         FOR i:=1 TO 5 DO BEGIN
            IF i<5 THEN
               write(_T, '          ')
            ELSE
               write(_T, '   L   n  ');
            FOR j:=t TO IMin(_NTaxa, t+nc-1) DO
               write(_T, ' ', _Taxa^[j,i]);
            writeln(_T, '')
         END;
         write(_T, '         ', _Li[_L, 3]);
         FOR i:=t TO IMin(_NTaxa, t+nc-1) DO write(_T, _Li[_L, 2], _Li[_L, 2]);
         writeln(_T, _Li[_L, 2], _Li[_L, 4]);

         ptr:=sptr;
         WHILE ptr^.Section=s DO BEGIN
            write(_T, ptr^.Level:4, ptr^.Cardinal:4, ' ', _Li[_l, 1]);
            FOR j:=t TO IMin(_NTaxa, t+nc-1) DO
               IF Inset(j, ptr) THEN
                  write(_T, ' ', _Li[_L, 12])
               ELSE
                  write(_T, ' ', _Li[_L, 13]);
            writeln(_T, ' ', _Li[_L, 1]);
            ptr:=AddAddr(ptr, size);
            inc(g);
            Gauge(g)
         END;

         write(_T, '         ', _Li[_L, 5]);
         FOR i:=t TO IMin(_NTaxa, t+nc-1) DO write(_T, _Li[_L, 2], _Li[_L, 2]);
         writeln(_T, _Li[_L, 2], _Li[_L, 6]);
         t:=j+1;
         writeln(_T, '')
      END;
      sptr:=ptr
   END;
   close(_T)
END; { bgA }

PROCEDURE bgB(c : char);
VAR s, t, size, n, nl : WORD;
    g                 : LONGINT;
    ptr               : LevelPtr;
BEGIN
   IF NotExist(c) THEN exit;
   ptr:=_UserHeap;
   nl:=ptr^.Cardinal;
   InitGauge(nl);
   IF Dont(ptr^.Section, _NTaxa, ptr^.Level, _NSections) THEN exit;
   writeln(_T, 'Local maximal horizons');
   Writeln(_T, '');
   Writeln(_T, ptr^.Section:4 , ' taxa');
   Writeln(_T, ptr^.Level:4   , ' sections');
   Writeln(_T, nl:4           , ' local maximal horizons');
   ptr:=AddAddr(ptr, 6);
   size:=LevelSize(_NTaxa);
   g:=0;
   FOR s:=1 TO _Nsections DO BEGIN
      writeln(_T, '');
      writeln(_T, '');
      writeln(_T, 'Section ', _Sections^[s]);
      WHILE (s=ptr^.Section) and (g<nl) DO BEGIN
         n:=0;
         FOR t:=1 TO _NTaxa DO
            IF Inset(t, ptr) THEN BEGIN
               IF n=0 THEN BEGIN
                  writeln(_T, '');
                  write(_T, ptr^.Level:3, ' [', ptr^.Cardinal:3, ']:');
                  n:=10
               END
               ELSE IF _Cfg^.Width<n+6 THEN BEGIN
                  writeln(_T, '');
                  write(_T, '             ');
                  n:=10
               END;
               write(_T, _Taxa^[t]:6);
               inc(n, 6)
            END;
         ptr:=AddAddr(ptr, size);
         inc(g);
         Gauge(g)
      END
   END;
   close(_T)
END; { bgB }

PROCEDURE bgC(c : char);
VAR l, nl, t, size, n : WORD;
    g                 : LONGINT;
    ptr               : LevelPtr;
BEGIN
   IF NotExist(c) THEN exit;
   ptr:=_UserHeap;
   InitGauge(ptr^.Cardinal);
   IF Dont(ptr^.Section, _NTaxa, ptr^.Level, _NSections) THEN exit;
   writeln(_T, 'Maximal residual horizons');
   writeln(_T, 'sorted by cardinality');
   Writeln(_T, '');
   Writeln(_T, ptr^.Section:4 , ' taxa');
   Writeln(_T, ptr^.Level:4   , ' sections');
   Writeln(_T, ptr^.Cardinal:4, ' maximal residual horizons');
   nl:=ptr^.Cardinal;
   ptr:=AddAddr(ptr, 6);
   size:=LevelSize(_NTaxa);
   g:=0;
   FOR l:=1 TO nl DO BEGIN
      writeln(_T, '');
      write(_T, ptr^.Section:3, '.', ptr^.Level:3, ' [', ptr^.Cardinal:3, ']:');
      n:=14;
      FOR t:=1 TO _NTaxa DO
         IF Inset(t, ptr) THEN BEGIN
            IF _Cfg^.width<n+6 THEN BEGIN
               writeln(_T, '');
               write(_T, '              ');
               n:=14
            END;
            write(_T, _Taxa^[t]:6);
            inc(n, 6)
         END;
      ptr:=AddAddr(ptr, size);
      inc(g);
      Gauge(g)
   END;
   close(_T)
END; { bgC }

FUNCTION Matrix(Mat        : POINTER;
                i, j, Size : LONGINT) : IntPtr;
VAR l : LONGint;
BEGIN
   IF j<i THEN BEGIN
      l:=i;
      i:=j;
      j:=l
   END;
   l:=((i-1)*Size+j-1-(i*(i+1) div 2))*2;
   WHILE $FFFF<l DO BEGIN
      Mat:=AddAddr(Mat, $FFFF);
      DEC(l, $FFFF)
   END;
   Matrix:=AddAddr(Mat, l)
END; { Matrix }

PROCEDURE bgD(c : char);
VAR sptr, ptr : IntPtr;
    t1, t2, n : WORD;
    g         : LONGINT;
BEGIN
   IF NotExist(c) THEN exit;
   sptr:=_UserHeap;
   InitGauge(_NTaxa*2-1);
   IF Dont(sptr^, _NTaxa, 0, 0) THEN exit;
   writeln(_T, 'BIOSTRATIGRAPHIC GRAPH');
   writeln(_T, 'with the reproducibility');
   writeln(_T, 'OF the superpositions');
   writeln(_T, '');
   writeln(_T, sptr^:3, ' taxa');
   writeln(_T, '');
   writeln(_T, 'Edges:');
   sptr:=AddAddr(sptr, 2);
   ptr:=sptr;
   g:=0;
   FOR t1:=1 TO _Ntaxa-1 DO BEGIN
      write(_T, _Taxa^[t1]:5, ':');
      n:=6;
      FOR t2:=t1+1 TO _NTaxa DO BEGIN
         IF ptr^=EDGE THEN BEGIN
            IF _Cfg^.Width<n+6 THEN BEGIN
               writeln(_T, '');
               write(_T, '      ');
               n:=6
            END;
            write(_T, _Taxa^[t2]:6);
            inc(n, 6)
         END;
         ptr:=AddAddr(ptr, 2)
      END;
      writeln(_T, '');
      inc(g);
      Gauge(g)
   END;
   writeln(_T, '');
   writeln(_T, 'Arcs (->):');
   ptr:=sptr;
   FOR t1:=1 TO _Ntaxa DO BEGIN
      write(_T, _Taxa^[t1]:5, ':');
      n:=6;
      FOR t2:=1 TO _NTaxa DO
         IF t1<>t2 THEN BEGIN
            ptr:=Matrix(sptr, t1, t2, _NTaxa);
            IF (ptr^<>EDGE) and
               (((t1<t2) and (ptr^<0)) or ((t1>t2) and (ptr^>0))) THEN BEGIN
               IF _Cfg^.Width<n+11 THEN BEGIN
                  writeln(_T, '');
                  write(_T, '      ');
                  n:=6
               END;
               write(_T, _Taxa^[t2]:6, '[', abs(ptr^):3, ']');
               inc(n, 11)
         END
      END;
      writeln(_T, '');
      inc(g);
      Gauge(g)
   END;
   close(_T)
END; { bgD }

FUNCTION Clique(section, level : WORD) : Str8;
VAR s, l : Str3;
    u    : char;
BEGIN
   IF section and $8000=$8000 THEN BEGIN
      str(section xor $8000, s);
      u:='*'
   END
   ELSE BEGIN
      str(section, s);
      u:=' '
   END;
   str(level, l);
   Clique:=s+'.'+l+u
END; { Clique }

PROCEDURE bgE(c : char);
VAR l, nl, t, size, n : WORD;
    g                 : LONGINT;
    ptr               : LevelPtr;
BEGIN
   IF NotExist(c) THEN exit;
   ptr:=_UserHeap;
   InitGauge(ptr^.Cardinal);
   IF Dont(ptr^.Section, _NTaxa, ptr^.Level, _NSections) THEN exit;
   writeln(_T, 'Maximal cliques sorted by cardinality');
   Writeln(_T, '');
   Writeln(_T, ptr^.Section:4 , ' taxa');
   Writeln(_T, ptr^.Level:4    , ' sections');
   Writeln(_T, ptr^.Cardinal:4, ' cliques');
   nl:=ptr^.Cardinal;
   ptr:=AddAddr(ptr, 6);
   size:=LevelSize(_NTaxa);
   g:=0;
   FOR l:=1 TO nl DO BEGIN
      writeln(_T, '');
      write(_T, l:3, Clique(ptr^.Section, ptr^.Level):9, '[', ptr^.Cardinal:3, ']:');
      n:=18;
      FOR t:=1 TO _NTaxa DO
         IF Inset(t, ptr) THEN BEGIN
            IF _Cfg^.width<n+6 THEN BEGIN
               writeln(_T, '');
               write(_T, '                  ');
               n:=18
            END;
            write(_T, _Taxa^[t]:6);
            inc(n, 6)
         END;
      ptr:=AddAddr(ptr, size);
      inc(g);
      Gauge(g)
   END;
   close(_T)
END; { bgE }

FUNCTION NotLoadCliques(nk  : WORD;
                        ptr : CliquePtr) : boolean;
VAR f     : file;
```

```
    count : WORD;
BEGIN
   IF FileExist(_Cfg^.Dpath+_Cfg^.Dname+'.BG0') THEN BEGIN
      assign(f, _Cfg^.Dpath+_Cfg^.Dname+'.BG0');
      reset(f, 1);
      blockread(f, ptr^, nk*sizeof(CliqueRec), count);
      IF count<nk*sizeof(CliqueRec) THEN BEGIN
         Wr(_Cfg^.Dname+'.BG0 not loaded');
         Close(_T);
         Erase(_T);
         NotLoadCliques:=TRUE;
         exit
      END;
      close(f);
      NotLoadCliques:=FALSE
   END
   ELSE BEGIN
      Wr(_Cfg^.Dname+'.BG0 not found !');
      close(_T);
      erase(_T);
      NotLoadCliques:=TRUE
   END
END; { NotLoadCliques }

PROCEDURE bgF(c : char);
VAR cliques       : CliquePtr;
    nk, n, k1, k2 : WORD;
    i             : BYTE;
    g             : LONGINT;
    ptr           : IntPtr;
    arc           : String[11];
BEGIN
   IF NotExist(c) THEN exit;
   ptr:=_UserHeap;
   nk:=ptr^;
   InitGauge((nk*(nk-1)) div 2);
   g:=0;
   ptr:=AddAddr(ptr, 2);
   cliques:=AddAddr(ptr, nk*(nk-1));
   IF NotLoadCliques(nk, cliques) THEN exit;
   Writeln(_T, 'Relationships between the cliques');
   Writeln(_T, '');
   Writeln(_T, '>> : i below j whithout contradiction');
   Writeln(_T, '-> : i below j whith contradiction');
   Writeln(_T, '<- : i above j whith contradiction');
   Writeln(_T, '<< : i above j whithout contradiction');
   Writeln(_T, '?? : undetermined');
   Writeln(_T, '');
   Writeln(_T, nk, ' cliques');
   Writeln(_T, '');
   FOR k1:=1 TO nk-1 DO BEGIN
      write(_T, Clique(cliques^[k1].Section, cliques^[k1].Level):8, ':');
      n:=9;
      FOR k2:=k1+1 TO nk DO BEGIN
         IF _Cfg^.Width<n+11 THEN BEGIN
            writeln(_T, '');
            write(_T, '         ');
            n:=9
         END;
         IF ptr^ AND $4000=$4000 THEN
            IF ptr^ XOR $4000=0 THEN
               arc:='??'
            ELSE
               arc:='!!'
         ELSE IF ptr^<0 THEN
            IF ptr^ AND $1000=$1000 THEN
               arc:='>>'
            ELSE
               arc:='->'
         ELSE IF ptr^ AND $1000=$1000 THEN
            arc:='<<'
         ELSE
            arc:='<-';
         arc:=' '+arc+Clique(cliques^[k2].Section, cliques^[k2].Level);
         FOR i:=length(arc)+1 TO 11 DO arc:=arc+' ';
         write(_T, arc);
         ptr:=AddAddr(ptr, 2);
         inc(n, 11);
         inc(g);
         Gauge(g)
      END;
      writeln(_T, '')
   END;
   close(_T)
END; { bgF }

PROCEDURE bgG(c : char);
VAR cliques                      : CliquePtr;
    k, nk, o, nc, na, n, size, i : WORD;
    g                            : LONGINT;
    conn                         : LevelPtr;
    arc                          : STRING[11];
BEGIN
   IF NotExist(c) THEN exit;
   conn:=_UserHeap;
   nk:=conn^.Section;
   nc:=conn^.Level;
   na:=conn^.Cardinal;
   size:=LevelSize(nk);
   IF na=0 THEN InitGauge(100) ELSE InitGauge(na);
   g:=0;
   cliques:=AddAddr(_UserHeap, 8+(nc+1)*size+na*4);
   IF NotLoadCliques(nk, cliques) THEN exit;
   Writeln(_T, 'List OF the strongly connected components');
   Writeln(_T, 'and OF the undetermined relationships');
   Writeln(_T, '');
   IF nc=0
      THEN BEGIN
         Writeln(_T, 'No strongly connected component');
         Gauge(100)
      END
      ELSE BEGIN
         Writeln(_T, nc:5, ' strongly connected components');
         Writeln(_T, na:5, ' undetermined relationships');
         conn:=AddAddr(conn, 6);
         FOR o:=1 TO nc DO BEGIN
            writeln(_T, '');
            writeln(_T, 'Strongly connected component ', o, ':');
            n:=0;
            FOR k:=1 TO nk DO
               IF Inset(k, conn) THEN BEGIN
                  IF _Cfg^.Width<n+9 THEN BEGIN
                     writeln(_T, '');
                     n:=0
                  END;
                  write(_T, Clique(cliques^[k].Section, cliques^[k].Level):9);
                  inc(n, 9)
               END;
            writeln(_T, '');
            writeln(_T, 'Undetermined relationships:');
            conn:=AddAddr(conn, size);
            n:=0;
            REPEAT
               IF _Cfg^.Width<n+20 THEN BEGIN
                  writeln(_T, '');
                  n:=0;
               END;
               write(_T, Clique(cliques^[conn^.Section].Section,
                         cliques^[conn^.Section].Level):9);
               arc:=' - '+Clique(cliques^[conn^.Level].Section,
                                 cliques^[conn^.Level].Level);
               FOR i:=length(arc)+1 TO 11 DO arc:=arc+' ';
               write(_T, arc);
               inc(n, 20);
               conn:=AddAddr(conn, 4);
               inc(g);
               Gauge(g)
            UNTIL conn^.Section=0;
            writeln(_T, '')
         END
      END;
   close(_T)
END; { bgG }

PROCEDURE bgH(c : char);
VAR l, nl, t, nc, p, np, i, j, size : WORD;
    g                               : LONGINT;
    sptr, ptr                       : LevelPtr;
BEGIN
   IF NotExist(c) THEN exit;
   sptr:=_UserHeap;
   nl:=sptr^.Cardinal;
   nc:=(_Cfg^.Width-12) div 2;
   np:=_NTaxa div nc;
   IF _NTaxa mod nc<>0 THEN inc(np);
   InitGauge(sptr^.Cardinal*np);
   IF Dont(sptr^.Section, _NTaxa, 0, 0) THEN exit;
   Writeln(_T, 'Adjacency matrix OF L');
   Writeln(_T, '');
   Writeln(_T, sptr^.Section:4 , ' taxa');
   Writeln(_T, sptr^.Cardinal:4, ' unitary associations');
   sptr:=AddAddr(sptr, 6);
   size:=LevelSize(_Ntaxa);
   g:=0;
   t:=1;
   FOR p:=1 TO np DO BEGIN
      writeln(_T, '');
      FOR i:=1 TO 5 DO BEGIN
         IF i<5 THEN
            write(_T, '          ')
         ELSE
            write(_T, '  UA   n  ');
         FOR j:=t TO IMin(_NTaxa, t+nc-1) DO
            write(_T, ' ', _Taxa^[j,i]);
         writeln(_T, '')
      END;
      write(_T, '         ', _Li[_L, 3]);
      FOR i:=t TO IMin(_NTaxa, t+nc-1) DO write(_T, _Li[_L, 2], _Li[_L, 2]);
      writeln(_T, _Li[_L, 2], _Li[_L, 4]);

      ptr:=sptr;
      FOR l:=1 TO nl DO BEGIN
         write(_T, ptr^.Level:4, ptr^.Cardinal:4, ' ', _Li[_l, 1]);
         FOR j:=t TO IMin(_NTaxa, t+nc-1) DO
            IF Inset(j, ptr) THEN
               write(_T, ' ', _Li[_L, 12])
            ELSE
               write(_T, ' ', _Li[_L, 13]);
         writeln(_T, ' ', _Li[_L, 1]);
         ptr:=AddAddr(ptr, size);
         inc(g);
         Gauge(g)
      END;

      write(_T, '         ', _Li[_L, 5]);
      FOR i:=t TO IMin(_NTaxa, t+nc-1) DO write(_T, _Li[_L, 2], _Li[_L, 2]);
      writeln(_T, _Li[_L, 2], _Li[_L, 6]);
      t:=j+1
   END;
   close(_T)
END; { bgH }

PROCEDURE bgI(c : char);
VAR sptr, ptr                 : LevelPtr;
    t, l, nl, fad, lad, size : WORD;
BEGIN
   IF NotExist(c) THEN exit;
   sptr:=_UserHeap;
   InitGauge(_NTaxa);
   IF Dont(sptr^.Section, _NTaxa, 0, 0) THEN exit;
   writeln(_T, 'DATUM');
   writeln(_T, '');
   writeln(_T, 'TITLE:');
   IF length(_Title)<>0 THEN writeln(_T, '"', _Title, '"');
   writeln(_T, '');
   writeln(_T, 'Section UA : bottom 1 - top ', sptr^.Cardinal);
   nl:=sptr^.Cardinal;
   sptr:=AddAddr(sptr, 6);
   size:=LevelSize(_NTaxa);
   FOR t:=1 TO _NTaxa DO
      BEGIN
         write(_T, _Taxa^[t]:5, ':');
         ptr:=sptr;
         fad:=nl+1;
         Lad:=0;
         FOR l:=nl downto 1 DO BEGIN
            IF Inset(t, ptr) THEN BEGIN
               IF l<fad THEN fad:=l;
               IF lad<l THEN lad:=l
            END;
            ptr:=AddAddr(ptr, size)
         END;
         writeln(_T, fad:3, ' - ', lad:3);
         Gauge(t)
      END;
   close(_T)
```

```
END; { bgI }

PROCEDURE bgJ(c : char);
VAR sptr, ptr                       : AUptr;
    l, nl, t, p, np, nc, min, max : WORD;
    g                              : LONGINT;
BEGIN
   sptr:=_UserHeap;
   IF NotExist(c) THEN exit;
   nl:=sptr^.Fad;
   nc:=(_Cfg^.Width-9) div 2;
   np:=nl div nc;
   IF nl mod nc<>0 THEN inc(np);
   InitGauge(np*_NTaxa);
   IF Dont(sptr^.Taxa, _NTaxa, 0, 0) THEN exit;
   Writeln(_T, 'SORTED UNITARY ASSOCIATIONS');
   writeln(_T, '');
   writeln(_T, sptr^.Taxa:3, ' taxa');
   writeln(_T, sptr^.Fad:3, ' unitary associations');
   g:=0;
   sptr:=AddAddr(sptr, 4);
   min:=1;
   FOR p:=1 TO np DO BEGIN
      max:=IMin(min+nc-1, nl);
      writeln(_T, '');
      FOR t:=1 TO 3 DO BEGIN
         write(_T, '       ');
         FOR l:=min TO max DO
            CASE t OF
               1 : IF 99<l THEN write(_T, l div 100:2) ELSE write(_T, '  ');
               2 : IF  9<l THEN
                      write(_T, (l mod 100) div 10:2)
                   ELSE
                      write(_T, '  ');
               3 : write(_T, l mod 10:2)
            END;
       writeln(_T, '')
     END;
      write(_T, '      ', _Li[_L, 3]);
      FOR l:=min TO max DO write(_T, _Li[_L,2], _Li[_L,9]);
      writeln(_T, _Li[_L, 2], _Li[_L, 4]);

      ptr:=sptr;
      FOR t:=1 TO _NTaxa DO BEGIN
         write(_T, _Taxa^[ptr^.Taxa]:5, ' ', _Li[_L, 7]);
         FOR l:=min TO IMin(max, ptr^.Fad-1) DO
            write(_T, _Li[_L, 2], _Li[_L, 11]);
         FOR l:=IMax(min, ptr^.Fad) TO IMin(max, ptr^.Lad) DO
            write(_T, ' ', _Li[_L, 14]);
         FOR l:=IMax(min, ptr^.Lad+1) TO max DO
            write(_T, _Li[_L, 2], _Li[_L, 11]);
         IF (ptr^.Fad<=max) and (max<=ptr^.Lad) THEN
            write(_T, ' ')
         ELSE
            write(_T, _Li[_L, 2]);
         writeln(_T, _Li[_L, 8]);
         ptr:=AddAddr(ptr, 6);
         inc(g);
         Gauge(g)
      END;

      write(_T, '      ', _Li[_L, 5]);
      FOR l:=min TO max DO write(_T, _Li[_L,2], _Li[_L, 10]);
      writeln(_T, _Li[_L, 2], _Li[_L, 6]);
      inc(min, nc)
   END;
   close(_T)
END; { bgJ }

PROCEDURE bgK(c : char);
VAR ptr      : CorrPtr;
    l, nl, s : WORD;
BEGIN
   IF NotExist(c) THEN exit;
   ptr:=_UserHeap;
   nl:=ptr^.Section;
   ptr:=AddAddr(ptr, 2);
   InitGauge(nl);
   writeln(_T, 'CORRELATION TABLE');
   writeln(_T, '');
   writeln(_T, nl, ' horizons');
   s:=0;
   FOR l:=1 TO nl DO BEGIN
      IF s<>ptr^.Section THEN BEGIN
         s:=ptr^.Section;
         writeln(_T, '');
         writeln(_T, 'Section ', _Sections^[s])
      END;
      writeln(_T, ptr^.Level:3, ':', ptr^.Fau:4, ' -', ptr^.Lau:4);
      ptr:=AddAddr(ptr, sizeof(ptr^));
      Gauge(l)
   END;
   close(_T)
END; { bgK }

PROCEDURE bgL(c : char);
VAR l, nl, s, ns, nc, p, np, i, j, k, size : WORD;
    g                                      : LONGINT;
    sptr, ptr                              : LevelPtr;
    ok                                     : boolean;
BEGIN
   IF NotExist(c) THEN exit;
   sptr:=_UserHeap;
   ns:=sptr^.Section;
   nl:=sptr^.Cardinal;
   nc:=(_Cfg^.Width-12) div 2;
   np:=_NSections div nc;
   IF _NSections mod nc<>0 THEN inc(np);
   InitGauge(sptr^.Cardinal*np);
   IF Dont(ns, _NSections, 0, 0) THEN exit;
   Writeln(_T, 'REPRODUCIBILITY OF THE UA');
   Writeln(_T, '');
   Writeln(_T, ns:4, ' sections');
   Writeln(_T, nl:4, ' unitary associations');
   sptr:=AddAddr(sptr, 6);
   size:=LevelSize(_NSections);
   g:=0;
   s:=1;
   FOR i:=1 TO _NSections DO
      FOR j:=length(_Sections^[i])+1 TO 25 DO
         _Sections^[i]:=' '+_Sections^[i];
   FOR p:=1 TO np DO BEGIN
      writeln(_T, '');
      ok:=FALSE;
      FOR i:=1 TO 25{3} DO BEGIN
         IF not ok THEN
            FOR j:=s TO IMin(_NSections, s+nc-1) DO
               IF _Sections^[j,i]<>' ' THEN ok:=TRUE;
         IF ok THEN BEGIN
            IF i<25{3} THEN
               write(_T, '         ')
            ELSE
               write(_T, ' UA   n  ');
            FOR j:=s TO IMin(_NSections, s+nc-1) DO
               write(_T, _Sections^[j,i]:2);
            writeln(_T, '')
         END
      END;
      write(_T, '         ', _Li[_L, 3]);
      FOR i:=s TO IMin(_NSections, s+nc-1) DO write(_T, _Li[_L, 2], _Li[_L, 2]);
      writeln(_T, _Li[_L, 2], _Li[_L, 4]);

      ptr:=sptr;
      FOR l:=1 TO nl DO BEGIN
         write(_T, ptr^.Level:4, ptr^.Cardinal:4, ' ', _Li[_l, 1]);
         FOR j:=s TO IMin(_NSections, s+nc-1) DO
            IF Inset(j, ptr) THEN
               write(_T, ' ', _Li[_L, 12])
            ELSE
               write(_T, ' ', _Li[_L, 13]);
         writeln(_T, ' ', _Li[_L, 1]);
         ptr:=AddAddr(ptr, size);
         inc(g);
         Gauge(g)
      END;

      write(_T, '         ', _Li[_L, 5]);
      FOR i:=s TO IMin(_NSections, s+nc-1) DO write(_T, _Li[_L, 2], _Li[_L, 2]);
      writeln(_T, _Li[_L, 2], _Li[_L, 6]);
      s:=j+1
   END;
   close(_T)
END; { bgL }
```

```
 PROCEDURE bgM(c : char);
 VAR ptr        : IntPtr;
     t1, t2, n : WORD;
     i          : BYTE;
     g, a       : LONGINT;
     t          : Str5;
 BEGIN
    IF NotExist(c) THEN exit;
    ptr:=_UserHeap;
    InitGauge((LONGINT(_NTaxa)*LONGINT(_NTaxa-1)) div 2);
    IF Dont(ptr^, _NTaxa, 0, 0) THEN exit;
    ptr:=AddAddr(ptr, 2);
    g:=0;
    n:=0;
    a:=0;
    writeln(_T, 'Virtual residual edges');
    writeln(_T, '');
    FOR t1:=1 TO _NTaxa-1 DO
       FOR t2:=t1+1 TO _NTaxa DO BEGIN
          IF ptr^<>0 THEN BEGIN
             inc(a);
             IF _Cfg^.Width<n+14 THEN BEGIN
                writeln(_T, '');
                n:=0
             END;
             write(_T, _Taxa^[t1]:6, ' - ');
             t:=_Taxa^[t2];
             WHILE t[1]=' ' DO delete(t, 1, 1);
             FOR i:=length(t)+1 TO 5 DO t:=t+' ';
             write(_T, t);
             inc(n, 14);
          END;
          ptr:=AddAddr(ptr, 2);
          inc(g);
          Gauge(g)
       END;
    IF a=0 THEN
       write(_T, 'No')
    ELSE BEGIN
       writeln(_T, '');
       writeln(_T, '');
       write(_T, a)
    END;
    write(_T, ' residual edge');
    IF 1<a THEN write(_T, 's');
    writeln(_T, ' found');
    close(_T)
 END; { bgM }

 VAR t : BYTE;
 BEGIN
    CheckBreak:=TRUE;
    _UserError:=ErrorMsg;
    InitFatalError(FatalError);
    Initialization;
    LoadLexicon;
    GetMemory(MaxAvail, 10000);
    gotoxy(2, wherey);
    Writeln('Memory available: ', _UserMemory, ' bytes');
    FOR t:=0 TO 12 DO
       IF (_Cfg^.Output shr t) and 1=1 THEN
          CASE t OF
              0 : bgA('A');
              1 : bgB('B');
              2 : bgC('C');
              3 : bgD('D');
              4 : bgE('E');
              5 : bgF('F');
              6 : bgG('G');
              7 : bgH('H');
              8 : bgI('I');
              9 : bgJ('J');
             10 : bgK('K');
             11 : bgL('L');
             12 : bgM('M')
          END
 END. { BG_CONV }
```

## *Program BG_DRAW*

Function : graphic display of the graphs

```
PROGRAM BG_DRAW;

{$A+,B-,D-,E+,F-,G+,I-,L-,N-,O-,R-,S+,V-,X+}
{$M 32768, 0, 655360}

USES Crt, Dos, Printer, Graph, Tools, Mouse, Math,
     Button, GPrint, Video, Error, GrWin, InitGra, HP, GrTools,
     BG_Def;

CONST
   VER : STRING[10] = 'DRAW v2.01';

TYPE
   ArcSet    = SET OF BYTE;
   GraphType = ARRAY[1..255] OF
                  RECORD
                     r    : BOOLEAN;
                     x, y : INTEGER;
                     a, e : ArcSet
                  END;
   GType     = (Gb, Gk, Gr, Gt);

   DrawProc  = PROCEDURE(x1, y1, x2, y2 : INTEGER);
   DrawSet   = (Ln, Rec);
   DrawType  = RECORD
                  x1, y1, x2, y2 : INTEGER;
                  d              : DrawSet
               END;

CONST
   X1draw =  110;
   X2draw =  639;
   dXdraw = 1440;
   _Xmin   : INTEGER = 0;
   _Xmax   : INTEGER = X2draw-1;
   _Ymin   : INTEGER = 0;

   _PageLeft : INTEGER = 0;
   _PageHome : INTEGER = 0;

   X1butt =   5;
   X2butt = 102;
   X1dir  = X1draw+28;
   X2dir  = X1dir+150;

   dYbutt = 21;

   _Background       : ScreenColor = (Black    ,  Blue     );
   _ButtonFrame      : ScreenColor = (LightGray,  DarkGray );
   _BackgroundButton : ScreenColor = (LightGray,  Green    );
   _ArcColor         : ScreenColor = (LightGray,  Cyan     );
   _VertexFrame      : ScreenColor = (LightGray,  LightGray);
   _VertexShadow     : ScreenColor = (Black    ,  DarkGray );
   _VertexGround     : ScreenColor = (Black    ,  LightRed );
   _VertexNumber     : ScreenColor = (LightGray,  White    );
   _TitleColor       : ScreenColor = (Black    ,  Yellow   );
   _LineColor        : ScreenColor = (LightGray,  Yellow   );
   _HelpColor        : ScreenColor = (LightGray,  DarkGray );
   _HelpBackColor    : ScreenColor = (Black    ,  White    );

   _P  : ARRAY[0..4, 0..1] OF BYTE
       = ((Black    , Blue),
          (Green    , Green),
          (Cyan     , Cyan),
          (LightRed , LightRed),
          (LightCyan, LightCyan));
   _PI : BYTE = 0;

   _Nvertex : BYTE = 0;
   _Marked  : BYTE = 0;

   _Tx : INTEGER = X1draw+15;
   _Ty : INTEGER = 30;

   _Lin    : BOOLEAN = FALSE;
   _Rec   : BOOLEAN = FALSE;
   _L     : BOOLEAN = FALSE;
   _R     : BOOLEAN = FALSE;
   _GamaP : BOOLEAN = FALSE;
   _GamaS : BOOLEAN = FALSE;
   _SCC   : BOOLEAN = FALSE;
   _Grid  : BOOLEAN = FALSE;
   _HP    : BOOLEAN = FALSE;
   _Dlast : BYTE =  0;
   _S     : BYTE = 12;
   _A     : BYTE =  0;

   EDGE = -$8000;

VAR
   _Path                   : DirStr;
   _Name                   : NameStr;
   _Ext                    : ExtStr;
   _G                      : GraphType;
   _Gtype                  : GType;
   _Succ, _Pred            : ArcSet;
   _dXv, _dYv,
   _Lx1, _Ly1, _Lx2, _Ly2 : INTEGER;
   _V, _M                  : POINTER;
   _D                      : ARRAY[1..255] OF DrawType;
   _Panel                  : ButtonPtr;
   _Dline                  : DrawSet;
   _Father, _Son           : BYTE;
   _Pratio                 : BYTE;
   _Plotter                : REAL;
   _RecMask                : RecArrayPtr;
   _VM                     : WORD;
   _Title                  : STRING;

{$F+} PROCEDURE FatalError(ExitCode : INTEGER); {$F-}
BEGIN
   RestoreCrtMode;
   TextColor(black);
   textbackground(lightgray);
   Writeln;
   CASE ExitCode OF
        0 : ;
      601 : Writeln('Error ', ExitCode, ': Bad command line => BG_DRAW -<B|K|R|T>
            <filename> [p] [/BW]');
      602 : Writeln('Error ', ExitCode, ': Graph with too many vertices (>255)');
      603 : Writeln('Error ', ExitCode, ': More than 50 vertices');
      ELSE Writeln(ErrorStr(ExitCode))
   END;
   IF ExitCode<>0 THEN BEGIN
      writeln;
      writeln('Press a key TO continue...');
      Pause
   END
END; { FatalError }

PROCEDURE Extrema(VAR xmin, ymin, xmax, ymax : INTEGER);

   PROCEDURE MinMax(x, y : INTEGER);
      BEGIN
         IF x<xmin THEN xmin:=x;
         IF y<ymin THEN ymin:=y;
         IF xmax<x THEN xmax:=x;
         IF ymax<y THEN ymax:=y
      END; { MinMax }

VAR p : BYTE;
    v : BYTE;
BEGIN { Extrema }
   xmin:=maxint;
   ymin:=maxint;
   xmax:=-maxint;
   ymax:=-maxint;
   FOR p:=1 TO _Dlast DO
      BEGIN
         MinMax(_D[p].x1, _D[p].y1);
         MinMax(_D[p].x2, _D[p].y2)
      END;
   FOR v:=1 TO _Nvertex DO
      IF NOT _G[v].r THEN
         MinMax(_G[v].x, _G[v].y);
   xmin:=xmin-_Dxv;
   ymin:=ymin-_Dyv;
   xmax:=xmax+_Dxv;
   ymax:=ymax+_Dyv
END; { Extrema }

FUNCTION px(x : INTEGER) : INTEGER;
BEGIN
   px:=_Pratio*(x-_Xmin)
END; { px }

FUNCTION py(y : INTEGER) : INTEGER;
BEGIN
   py:=_Pratio*(y-_Ymin)
END; { py }

FUNCTION hx(x : REAL) : REAL;
BEGIN
   hx:=_Plotter*x
END; { hx }

FUNCTION hy(y : REAL) : REAL;
BEGIN
   hy:=-_Plotter*y
END; { hy }

PROCEDURE PArrow(x1, y1, x2, y2 : INTEGER);
VAR xa, ya, da, da2   : INTEGER;
    DX, dy, r, rx, ry : REAL;
    p                 : ARRAY[1..3] OF
                           RECORD
                              x, y : INTEGER
                           END;
BEGIN
   DX:=x2-x1;
   dy:=y2-y1;
   IF _A=0 THEN
      BEGIN
         xa:=(x1+x2) DIV 2;
         ya:=(y1+y2) DIV 2
      END
      ELSE BEGIN
         xa:=((x1+x2) DIV 2)+((x2-x1) DIV _A);
         ya:=((y1+y2) DIV 2)+((y2-y1) DIV _A)
      END;
   r:=Sqrt(Sqr(DX)+Sqr(dy));
   IF r<1E-5
      THEN EXIT
      ELSE BEGIN
         rx:=DX/r;
         ry:=dy/r
      END;
   da :=_Pratio*3;
   da2:=_Pratio*6;
   p[1].x:=xa+Round(da2*rx);
   p[1].y:=ya+Round(da2*ry);
   xa    :=xa-Round(da2*rx);
   ya    :=ya-Round(da2*ry);
   p[2].x:=xa-Round(da *ry);
   p[2].y:=ya+Round(da *rx);
   p[3].x:=xa+Round(da *ry);
   p[3].y:=ya-Round(da *rx);
   GPrint.Line(p[1].x, p[1].y, p[2].x, p[2].y);
   GPrint.Line(p[1].x, p[1].y, p[3].x, p[3].y)
END; { PArrow }

PROCEDURE RasterVertex;
VAR v, a : BYTE;
BEGIN
   IF _S<>11 THEN
      FOR v:=1 TO _Nvertex DO
         IF NOT _G[v].r THEN
            FOR a:=1 TO _Nvertex DO
               IF (NOT _G[a].r) AND (a IN _G[v].a) THEN
                  BEGIN
                     PArrow(px(_G[v].x), py(_G[v].y),
                            px(_G[a].x), py(_G[a].y));
                     GPrint.Line(px(_G[v].x), py(_G[v].y),
                                 px(_G[a].x), py(_G[a].y))
                  END;
   IF _S<>13 THEN
      FOR v:=1 TO _Nvertex DO
         IF NOT _G[v].r THEN
            FOR a:=Succ(v) TO _Nvertex DO
```

```
               IF (NOT _G[a].r) AND (a IN _G[v].e) THEN
                     GPrint.Line(px(_G[v].x), py(_G[v].y),
                                 px(_G[a].x), py(_G[a].y));
   FOR v:=1 TO _Nvertex DO
      IF NOT _G[v].r THEN
         BEGIN
            GPrint.Rectangle(px(_G[v].x-_dXv), py(_G[v].y-_dYv),
                             px(_G[v].x+_dXv), py(_G[v].y+_dYv));
            FillRectangle(px(_G[v].x-_dXv), py(_G[v].y-_dYv),
                          px(_G[v].x+_dXv), py(_G[v].y+_dYv));
            GPrint.OutTextXY(px(_G[v].x), py(_G[v].y), IStr(v, 0))
         END
END; { RasterVertex }

PROCEDURE RasterLine;
VAR p : BYTE;
BEGIN
   FOR p:=1 TO _Dlast DO
      IF _D[p].d=Ln
         THEN GPrint.Line(px(_D[p].x1), py(_D[p].y1),
                          px(_D[p].x2), py(_D[p].y2))
         ELSE GPrint.Rectangle(px(_D[p].x1), py(_D[p].y1),
                               px(_D[p].x2), py(_D[p].y2))
END; { RasterLine }

{$F+} PROCEDURE Draw; {$F-}
BEGIN
   RasterVertex;
   RasterLine;
END; { Draw }

FUNCTION HPArrow(x1, y1, x2, y2 : REAL) : BYTE;
VAR xa, ya, da, da2   : REAL;
    DX, dy, r, rx, ry : REAL;
    p                 : ARRAY[1..3] OF
                           RECORD
                              x, y : REAL
                           END;
    error             : BYTE;
BEGIN
   DX:=x2-x1;
   dy:=y2-y1;
   IF _A=0 THEN
      BEGIN
         xa:=(x1+x2)/2;
         ya:=(y1+y2)/2
      END
      ELSE BEGIN
         xa:=((x1+x2)/2)+((x2-x1)/_A);
         ya:=((y1+y2)/2)+((y2-y1)/_A)
      END;
   r:=Sqrt(Sqr(DX)+Sqr(dy));
   IF r<1E-5
      THEN EXIT
      ELSE BEGIN
         rx:=DX/r;
         ry:=dy/r
      END;
   da :=_Plotter*3;
   da2:=_Plotter*6;
   p[1].x:=xa+da2*rx;
   p[1].y:=ya+da2*ry;
   xa    :=xa-da2*rx;
   ya    :=ya-da2*ry;
   p[2].x:=xa-da *ry;
   p[2].y:=ya+da *rx;
   p[3].x:=xa+da *ry;
   p[3].y:=ya-da *rx;
   error:=HP.Line(p[1].x, p[1].y, p[2].x, p[2].y);
   IF error=0 THEN error:=HP.Line(p[1].x, p[1].y, p[3].x, p[3].y);
   HPArrow:=error
END; { HPArrow }

PROCEDURE Plot;
VAR v, a : BYTE;
BEGIN
   GetMem(_RecMask, _NVertex*SizeOf(RecType));
   _VM:=0;
   FOR v:=1 TO _NVertex DO
      IF NOT _G[v].r THEN BEGIN
         _RecMask^[v].x1:=Round(hx(_G[v].x-_dXv));
         _RecMask^[v].y1:=Round(hy(_G[v].y-_dYv));
         _RecMask^[v].x2:=Round(hx(_G[v].x+_dXv));
         _RecMask^[v].y2:=Round(hy(_G[v].y+_dYv));
         INC(_VM)
      END;
   IF _S<>11 THEN
      FOR v:=1 TO _Nvertex DO
         IF NOT _G[v].r THEN
            FOR a:=1 TO _Nvertex DO
               IF (NOT _G[a].r) AND (a IN _G[v].a) THEN
                  BEGIN
                     IF 0<HPArrow(hx(_G[v].x), hy(_G[v].y),
                                 hx(_G[a].x), hy(_G[a].y)) THEN EXIT;
                     MultiRecMask(Round(hx(_G[v].x)), Round(hy(_G[v].y)),
                                 Round(hx(_G[a].x)), Round(hy(_G[a].y)),
                                 _VM, _RecMask, HP.ILine)
                  END;
   IF _S<>13 THEN
      FOR v:=1 TO _Nvertex DO
         IF NOT _G[v].r THEN
            FOR a:=Succ(v) TO _Nvertex DO
               IF (NOT _G[a].r) AND (a IN _G[v].e) THEN
                  MultiRecMask(Round(hx(_G[v].x)), Round(hy(_G[v].y)),
                               Round(hx(_G[a].x)), Round(hy(_G[a].y)),
                               _VM, _RecMask, HP.ILine);
   FOR v:=1 TO _Nvertex DO
      IF NOT _G[v].r THEN
         BEGIN
            IF 0<HP.Rectangle(hx(_G[v].x-_dXv), hy(_G[v].y-_dYv),
                              hx(_G[v].x+_dXv), hy(_G[v].y+_dYv)) THEN EXIT;
            IF 0<HP.OutTextXY(hx(_G[v].x), hy(_G[v].y), IStr(v, 0)) THEN EXIT
         END;
   FOR a:=1 TO _Dlast DO
      IF _D[a].d=Ln THEN
         IF 0<HP.Line(hx(_D[a].x1), hy(_D[a].y1),
                      hx(_D[a].x2), hy(_D[a].y2)) THEN
            EXIT
         ELSE
      ELSE IF 0<HP.Rectangle(hx(_D[a].x1), hy(_D[a].y1),
                             hx(_D[a].x2), hy(_D[a].y2)) THEN
         EXIT
END; { Plot }

PROCEDURE Print;
VAR xmax, ymax : INTEGER;
    error      : BYTE;
BEGIN
   Extrema(_Xmin, _Ymin, xmax, ymax);
   _CharSize:=_Pratio;
   IF PrinterOk THEN
      BEGIN
         OpenMsgGrWin('ESC = abort');
         {$I-}
         Writeln(Lst, '');
         Writeln(Lst, ^['@'^O);
         Writeln(Lst, '___________________________________');
         Writeln(Lst, _Title);
         CASE _Gtype OF
            Gb : Writeln(Lst, 'Biostratigraphic graph G*');
            Gk : Writeln(Lst, 'Graph OF cliques Gk');
            Gr : Writeln(Lst, 'Graph OF reproducibility Gk'#39)
         END;
         IF _Gtype IN [Gb, Gt] THEN
            CASE _S OF
               11 : Writeln(Lst, 'Undirected graph');
               12 : IF _Gtype=Gt THEN Writeln('Biostratigraphic graph');
               13 : Writeln(Lst, 'Directed graph')
            END;
         IF _GamaS AND (0<_Father) THEN
            Writeln(Lst, 'Successors OF ', _Father);
         IF _GamaP AND (0<_Son    ) THEN
            Writeln(Lst, 'Predecessors OF ', _Son);
         IF _SCC THEN
            Writeln(Lst, 'Cycles');
         Writeln(Lst, 'BioGraph (c) 1990, J.Savary & J.Guex');
         Writeln(Lst, '');
         {$I+}
         error:=PrintGraph(0, 0, _Pratio*(xmax-_Xmin+1),
                           _Pratio*(ymax-_Ymin+1), High, Draw);
         CloseGrWin;
         IF error=0 THEN
            {$I-} Writeln(Lst, ^['@') {$I+}
         ELSE
```

```
            MsgGrWin(ErrorStr(error))
      END
      ELSE MsgGrWin(ErrorStr(253))
END; { Print }

PROCEDURE Plotter(xpaper, ypaper : REAL);
LABEL STOP;
CONST WC = 0.20;
      HC = 0.30;
      F  = 15.0;
      Y  = 12;
      NL =  7;
VAR xmax, ymax                                   : INTEGER;
    dx, dy, w, h, fx, fy, sx1, sy1, sx2, sy2 : REAL;
    title, ti                                    : STRING;
    i                                            : BYTE;
BEGIN
   Extrema(_Xmin, _Ymin, xmax, ymax);
   dx:=Abs(xmax-_Xmin);
   dy:=Abs(ymax-_Ymin)+(NL*Y);
   xpaper:=xpaper/F;
   ypaper:=ypaper/F;
   IF xpaper<dx THEN fx:=xpaper/dx ELSE fx:=1;
   IF ypaper<dy THEN fy:=ypaper/dy ELSE fy:=1;
   IF fx<fy THEN _Plotter:=fx ELSE _Plotter:=fy;
   IF dx<xpaper THEN BEGIN
      sx1:=_Xmin-(xpaper-dx)/2;
      sx2:=xmax+(xpaper-dx)/2
   END
   ELSE BEGIN
      sx1:=_Xmin;
      sx2:=xmax
   END;
   IF dy<ypaper THEN BEGIN
      sy1:=_Ymin-(ypaper-dy)/2;
      sy2:=ymax+(NL*Y)+(ypaper-dy)/2
   END
   ELSE BEGIN
      sy1:=_Ymin;
      sy2:=ymax+(NL*Y)
   END;
   w:=WC*_Plotter;
   h:=HC*_Plotter;
   fx:=_Xmin+dx/2;
   fy:=ymax+Y;
   IF PrinterOk THEN
      BEGIN
         OpenMsgGrWin('ESC = abort');
         IF 0<HP.InitPlot THEN GOTO STOP;
         IF 0<HP.Orientation(Portrait) THEN GOTO STOP;
         IF 0<HP.Scale(hx(sx1), hy(sy2), hx(sx2), hy(sy1)) THEN GOTO STOP;
         IF 0<HP.SetTextSize(w, h) THEN GOTO STOP;
         IF 0<HP.SetColor(1) THEN GOTO STOP;
         title:=_Title;
         WHILE 0<length(title) DO BEGIN
            ti:=copy(title, 1, pos(#13#10, title)-1);
            IF length(ti)=0 THEN BEGIN
               ti:=title;
               title:=''
            END
            ELSE
               delete(title, 1, pos(#13#10, title)+1);
            IF 0<length(ti) THEN BEGIN
               FOR i:=1 TO length(ti) DO
                  IF not (ti[i] in [#32..#126]) THEN
                     ti[i]:=#32;
               IF 0<HP.OutTextXY(hx(fx), hy(fy), ti) THEN GOTO STOP;
               fy:=fy+Y
            END
         END;
         CASE _Gtype OF
            Gb : BEGIN
                    IF 0<HP.OutTextXY(hx(fx), hy(fy), 'Biostratigraphic graph G*')
                       THEN GOTO STOP;
                    fy:=fy+Y
                 END;
            Gk : BEGIN
                    IF 0<HP.OutTextXY(hx(fx), hy(fy), 'Graph OF cliques Gk')
                       THEN GOTO STOP;
                    fy:=fy+Y
                 END;
            Gr : BEGIN
```

```
                    IF 0<HP.OutTextXY(hx(fx), hy(fy), 'Graph OF reproducibility Gk'#39)
                       THEN GOTO STOP;
                    fy:=fy+Y
                 END
         END;
         IF _Gtype IN [Gb, Gt] THEN
            CASE _S OF
               11 : BEGIN
                       IF 0<HP.OutTextXY(hx(fx), hy(fy), 'Undirected graph') THEN
                          GOTO STOP;
                       fy:=fy+Y
                    END;
               12 : IF _Gtype=Gt THEN
                       BEGIN
                          IF 0<HP.OutTextXY(hx(fx), hy(fy), 'Biostratigraphic graph')
                             THEN GOTO STOP;
                          fy:=fy+Y
                       END;
               13 : BEGIN
                       IF 0<HP.OutTextXY(hx(fx), hy(fy), 'Directed graph') THEN
                          GOTO STOP;
                       fy:=fy+Y
                    END
            END;
         IF _GamaS AND (0<_Father) THEN
         BEGIN
            IF 0<OutTextXY(hx(fx), hy(fy), 'Successors OF '+IStr(_Father, 0)) THEN
               GOTO STOP;
            fy:=fy+Y
         END;
         IF _GamaP AND (0<_Son    ) THEN
         BEGIN
            IF 0<OutTextXY(hx(fx), hy(fy), 'Predecessors OF '+IStr(_Son, 0)) THEN
               GOTO STOP;
            fy:=fy+Y
         END;
         IF _SCC THEN
         BEGIN
            IF 0<OutTextXY(hx(fx), hy(fy), 'Cycles') THEN GOTO STOP;
            fy:=fy+Y
         END;
         IF 0<OutTextXY(hx(fx), hy(fy), 'BioGraph (c) 1990, J.Savary & J.Guex')
            THEN GOTO STOP;
         Plot;
STOP:
         IF 0<HP.SetColor(0) THEN;
         CloseGrWin
      END
      ELSE MsgGrWin(ErrorStr(247))
END; { Plotter }

PROCEDURE CreateVertex;
BEGIN
   _dXv:=TextWidth('12');
   _dYv:=TextHeight('12');
   GetMem(_V, ImageSize(0,0,2*_dXv,2*_dYv));
   GetMem(_M, ImageSize(0,0,2*_dXv,2*_dYv));
   Graph.SetColor(_VertexFrame[_C]);
   SetLineStyle(SolidLn, 0, NormWidth);
   Graph.Rectangle(X1draw, 0, X1draw+2*_dXv, 2*_dYv);
   SetFillStyle(SolidFill, _VertexGround[_C]);
   FloodFill(X1draw+_dXv, _dYv, _VertexFrame[_C]);
   Graph.SetColor(_VertexShadow[_C]);
   Graph.Line(X1draw+1, 2*_dYv-1, X1draw+2*_dXv-1, 2*_dYv-1);
   Graph.Line(X1draw+2*_dXv-1, 1, X1draw+2*_dXv-1, 2*_dYv-1);
   GetImage(X1draw, 0, X1draw+2*_dXv, 2*_dYv, _V^);
   SetMouseField(0+_dXv, 0+_dYv, GetMaxX-2*_dXv, GetMaxY-_dYv-1);
   SetCursorPos(GetMaxX DIV 2, GetMaxY DIV 2)
END; { CreateVertex }

FUNCTION fx(x : INTEGER) : INTEGER;
BEGIN
   fx:=x-_PageLeft
END; { fx }

FUNCTION ffx(x : INTEGER) : INTEGER;
BEGIN
   ffx:=x+_PageLeft-X1draw
END; { ffx }

FUNCTION fy(y : INTEGER) : INTEGER;
BEGIN
```

```
   fy:=y-_PageHome
END; { fy }

FUNCTION ffy(y : INTEGER) : INTEGER;
BEGIN
   ffy:=y+_PageHome
END; { ffy }

PROCEDURE Reverse;
BEGIN
   HideMouse;
   GetImage(fx(_G[_Marked].x)-_dXv, fy(_G[_Marked].y)-_dYv,
            fx(_G[_Marked].x)+_dXv, fy(_G[_Marked].y)+_dYv, _M^);
   PutImage(fx(_G[_Marked].x)-_dXv, fy(_G[_Marked].y)-_dYv, _M^, NOTPut);
   ShowMouse
END; { Reverse }

PROCEDURE Arrow(x1, y1, x2, y2 : INTEGER);
VAR xa, ya             : INTEGER;
    DX, dy, r, rx, ry : REAL;
    p                  : ARRAY[1..3] OF
                            RECORD
                               x, y : INTEGER
                            END;
BEGIN
   DX:=x2-x1;
   dy:=y2-y1;
   IF _A=0 THEN
      BEGIN
         xa:=(x1+x2) DIV 2;
         ya:=(y1+y2) DIV 2
      END
      ELSE BEGIN
         xa:=((x1+x2) DIV 2)+((x2-x1) DIV _A);
         ya:=((y1+y2) DIV 2)+((y2-y1) DIV _A)
      END;
   r:=Sqrt(Sqr(DX)+Sqr(dy));
   IF r<1E-5
      THEN EXIT
      ELSE BEGIN
         rx:=DX/r;
         ry:=dy/r
      END;
   p[1].x:=xa+Round( 6*rx);
   p[1].y:=ya+Round( 6*ry);
   xa    :=xa-Round( 6*rx);
   ya    :=ya-Round( 6*ry);
   p[2].x:=xa-Round( 3*ry);
   p[2].y:=ya+Round( 3*rx);
   p[3].x:=xa+Round( 3*ry);
   p[3].y:=ya-Round( 3*rx);
   SetLineStyle(SolidLn, 0, ThickWidth);
   Graph.Line(p[1].x, p[1].y, p[2].x, p[2].y);
   Graph.Line(p[1].x, p[1].y, p[3].x, p[3].y);
   SetLineStyle(SolidLn, 0, NormWidth)
END; { Arrow }

PROCEDURE PutAllVertex;
VAR v, a : BYTE;
BEGIN
   SetTextStyle(DefaultFont, HorizDir, 1);
   SetTextJustify(CenterText, CenterText);
   Graph.SetColor(_ArcColor[_C]);
   IF _S<>11 THEN
      FOR v:=1 TO _Nvertex DO
         IF NOT _G[v].r THEN
            FOR a:=1 TO _Nvertex DO
               IF (NOT _G[a].r) AND (a IN _G[v].a) THEN
                  BEGIN
                     Arrow(fx(_G[v].x), fy(_G[v].y), fx(_G[a].x), fy(_G[a].y));
                     Graph.Line(fx(_G[v].x), fy(_G[v].y),
                                fx(_G[a].x), fy(_G[a].y))
                  END;
   IF _S<>13 THEN
      FOR v:=1 TO _Nvertex DO
         IF NOT _G[v].r THEN
            FOR a:=Succ(v) TO _Nvertex DO
               IF (NOT _G[a].r) AND (a IN _G[v].e) THEN
                  Graph.Line(fx(_G[v].x), fy(_G[v].y),
                             fx(_G[a].x), fy(_G[a].y));
   Graph.SetColor(_VertexNumber[_C]);
   FOR v:=1 TO _Nvertex DO
```

```
      IF (NOT _G[v].r) AND
         (0<=fx(_G[v].x)-_dXv) AND (fx(_G[v].x)+_dXv<GetMaxX) AND
         (0<=fy(_G[v].y)-_dYv) AND (fy(_G[v].y)+_dYv<GetMaxY)
         THEN BEGIN
            PutImage(fx(_G[v].x)-_dXv, fy(_G[v].y)-_dYv, _V^, NormalPut);
            Graph.OutTextXY(fx(_G[v].x), fy(_G[v].y), IStr(v, 0));
            IF _Marked=v THEN Reverse
         END
END; { PutAllVertex }

PROCEDURE Sketch;
VAR p : BYTE;
BEGIN
   Graph.SetColor(_LineColor[_C]);
   SetLineStyle(SolidLn, 0, NormWidth);
   FOR p:=1 TO _Dlast DO
      IF _D[p].d=Ln
         THEN Graph.Line(fx(_D[p].x1), fy(_D[p].y1),
                         fx(_D[p].x2), fy(_D[p].y2))
         ELSE Graph.Rectangle(fx(_D[p].x1), fy(_D[p].y1),
                              fx(_D[p].x2), fy(_D[p].y2))
END; { Sketch }

PROCEDURE PutDrawings(x, y : INTEGER);
BEGIN
   Graph.SetColor(_LineColor[_C]);
   SetLineStyle(SolidLn, 0, NormWidth);
   SetWriteMode(XORPut);
   HideMouse;
   IF _Dline=Ln
      THEN Graph.Line(fx(_Lx1), fy(_Ly1), fx(x), fy(y))
      ELSE Graph.Rectangle(fx(_Lx1), fy(_Ly1), fx(x), fy(y));
   ShowMouse;
   SetWriteMode(NormalPut)
END; { PutDrawings }

PROCEDURE Refresh;
VAR x, y : INTEGER;
BEGIN
   HideMouse;
   ClearViewPort;
   IF _Grid THEN
      BEGIN
         x:=fx(Round(_PageLeft/_dXv)*_dXv);
         REPEAT
            y:=fy(Round(_PageHome/_dYv/2)*_dYv*2);
            REPEAT
               Graph.PutPixel(x, y, _LineColor[_C]);
               Inc(y, _dYv)
            UNTIL GetMaxY<y;
            Inc(x, _dXv)
         UNTIL X2draw<x
      END;
   PutAllVertex;
   Sketch;
   IF _L THEN PutDrawings(_Lx2, _Ly2);
   Showmouse
END; { Refresh }

PROCEDURE SetTitle;
BEGIN
   FSplit(_Path, _Path, _Name, _Ext);
   HideMouse;
   SetViewPort(X1butt,1,X2butt,dYbutt-1,ClipOn);
   ClearViewPort;
   SetFillStyle(SolidFill, _BackgroundButton[_C]);
   FloodFill(1,1,_BackgroundButton[_C]);
   SetTextStyle(DefaultFont, HorizDir, 1);
   Graph.SetColor(_TitleColor[_C]);
   SetTextJustify(LeftText, CenterText);
   Graph.OutTextXY(1, dYbutt DIV 2, _Name);
   SetTextJustify(RightText, CenterText);
   CASE _Gtype OF
      Gb : Graph.OutTextXY(X2butt-X1butt-1, dYbutt DIV 2, 'G*');
      Gk : Graph.OutTextXY(X2butt-X1butt-1, dYbutt DIV 2, 'Gk');
      Gr : Graph.OutTextXY(X2butt-X1butt-1, dYbutt DIV 2, 'Gk'#39)
   END;
   SetViewPort(X1draw,0,X2draw,GetMaxY,ClipOn);
   ShowMouse
END; { SetTitle }
```

```
PROCEDURE PressArrowButton(a : BYTE);
BEGIN
   CASE a OF
      0 : PressButton(_Panel, 26);
      4 : PressButton(_Panel, 28);
      6 : PressButton(_Panel, 27)
   END
END; { PressArrowButton }

PROCEDURE ArrowAspect(a : BYTE);
BEGIN
   PressArrowButton(_A);
   CASE a OF
      26 : _A:=0;
      27 : _A:=6;
      28 : _A:=4
   END;
   IF _S<>11 THEN Refresh
END; { ArrowAspect }

PROCEDURE Successors(n : BYTE);
VAR v : BYTE;
BEGIN
   IF _G[n].r THEN EXIT;
   _Succ:=_Succ+[n];
   FOR v:=1 TO _Nvertex DO
      IF (v IN _G[n].a) AND NOT (v IN _Succ) THEN
         Successors(v)
END; { Successors }

PROCEDURE Predecessors(n : BYTE);
VAR v : BYTE;
BEGIN
   IF _G[n].r THEN EXIT;
   _Pred:=_Pred+[n];
   FOR v:=1 TO _Nvertex DO
      IF (n IN _G[v].a) AND NOT (v IN _Pred) THEN
         Predecessors(v)
END; { Predecessors }

PROCEDURE LoadMatA;
VAR t1, t2, taxa, a : INTEGER;
    f               : FILE OF INTEGER;
BEGIN
   FillChar(_G, SizeOf(_G), 0);
   Assign(f, Concat(_Path, '.BGD'));
   Reset(f);
   Read(f, taxa);
   IF (taxa=0) OR (255<taxa) THEN HALT(602);
   _Nvertex:=taxa;
   FOR t1:=1 TO Pred(taxa) DO
      FOR t2:=Succ(t1) TO taxa DO
         BEGIN
            Read(f, a);
            IF a=EDGE THEN
               BEGIN
                  _G[t1].e:=_G[t1].e+[t2];
                  _G[t2].e:=_G[t2].e+[t1]
               END
               ELSE IF a<>0 THEN
                  IF a<0
                     THEN _G[t1].a:=_G[t1].a+[t2]
                     ELSE _G[t2].a:=_G[t2].a+[t1]
         END;
   Close(f);
   _S:=12;
   _Gtype:=Gb
END; { LoadMatA }

PROCEDURE LoadMatH;
VAR k1, k2, nk, a : INTEGER;
    f             : FILE OF INTEGER;
BEGIN
   FillChar(_G, SizeOf(_G), 0);
   Assign(f, Concat(_Path, '.BGF'));
   Reset(f);
   Read(f, nk);
   IF (nk=0) OR (255<nk) THEN HALT(602);
   _Nvertex:=nk;
   FOR k1:=1 TO Pred(nk) DO
      FOR k2:=Succ(k1) TO nk DO
         BEGIN
```

```
            Read(f, a);
            IF (a AND $4000=0) AND (a<>0) THEN
               IF a<0
                  THEN _G[k1].a:=_G[k1].a+[k2]
                  ELSE _G[k2].a:=_G[k2].a+[k1]
         END;
   Close(f);
   _S:=13;
   _Gtype:=Gk
END; { LoadMatH }

PROCEDURE LoadMatR;
VAR k, s, nk, ns  : INTEGER;
    buf           : ARRAY[1..3] OF INTEGER;
    f             : FILE;
    r             : ARRAY[1..255] OF ArcSet;
    Size          : WORD;
    n, k1, k2     : BYTE;
BEGIN
   Assign(f, Concat(_Path, '.BGL'));
   Reset(f,1);
   BlockRead(f, buf, SizeOf(buf));
   ns:=buf[1];
   nk:=buf[3];
   IF (nk=0) OR (255<nk) OR (ns=0) OR (255<ns) THEN HALT(602);
   _Nvertex:=nk;
   Size:=2*(((ns-1) SHR 4)+1);
   FOR k:=nk DOWNTO 1 DO
      BEGIN
         BlockRead(f, buf, SizeOf(buf));
         BlockRead(f, r[k], Size)
      END;
   Close(f);
   FillChar(_G, SizeOf(_G), 0);
   FOR k:=1 TO Pred(nk) DO
      FOR s:=0 TO Pred(ns) DO
         IF s IN r[k] THEN
            BEGIN
               n:=k;
               REPEAT Inc(n) UNTIL (n=nk) OR (s IN r[n]);
               IF s IN r[n] THEN _G[k].a:=_G[k].a+[n]
            END;
   { Ne conserve que les arcs transitifs }
   FOR k1:=Pred(Pred(nk)) DOWNTO 1 DO
      FOR k2:=Succ(Succ(k1)) TO nk DO
         IF k2 IN _G[k1].a THEN
            BEGIN
               _G[k1].a:=_G[k1].a-[k2];
               _Succ:=[];
               Successors(k1);
               IF NOT (k2 IN _Succ) THEN _G[k1].a:=_G[k1].a+[k2]
            END;
   _S:=13;
   _Gtype:=Gr
END; { LoadMatR }

PROCEDURE LoadTest;
VAR f      : TEXT;
    v1, v2 : BYTE;
    n      : INTEGER;
    l      : STRING;
BEGIN
   FillChar(_G, SizeOf(_G), 0);
   Assign(f, Concat(_Path, '.GAT'));
   Reset(f);
   Readln(f, _Nvertex);
   Readln(f, l);
   FOR v1:=1 TO Pred(_Nvertex) DO
      BEGIN
         Read(f, n);
         IF n<>v1 THEN HALT(603);
         FOR v2:=Succ(v1) TO _Nvertex DO
            BEGIN
               Read(f, n);
               IF n=2 THEN
                  BEGIN
                     _G[v1].e:=_G[v1].e+[v2];
                     _G[v2].e:=_G[v2].e+[v1]
                  END
                  ELSE IF n=1
                     THEN _G[v1].a:=_G[v1].a+[v2]
                     ELSE IF n=-1
```

```
                        THEN _G[v2].a:=_G[v2].a+[v1]
            END
      END;
   Close(f);
   _S:=12;
   _Gtype:=Gt
END; { LoadTest }

PROCEDURE UnPack(m : ArrayOfBytePtr;
                 s : WORD;
             VAR f : File);
VAR i, n : WORD;
BEGIN
   i:=0;
   REPEAT
      BlockRead(f, m^[i], 1);
      Inc(i);
      IF m^[i-1]=0 THEN BEGIN
         BlockRead(f, n, 2);
         FillChar(m^[i], n, 0);
         Inc(i, n)
      END
   UNTIL s=i
END; { UnPack }

PROCEDURE LoadFile(p : PathStr);
VAR f : FILE;
    v : STRING;
BEGIN
   p:=SetFileExt(p, '.IMG');
   IF NOT FileExist(p)
      THEN BEGIN
         MsgGrWin(ErrorStr(2));
         EXIT
      END;
   Assign(f, p);
   Reset(f,1);
   BlockRead(f, v, SizeOf(VER));
   IF Copy(v, 1, 7)<>Copy(VER, 1, 7)
      THEN BEGIN
         IF Copy(v,1,4)<>Copy(VER,1,4) THEN
            MsgGrWin('ERROR: not .IMG file')
         ELSE
            MsgGrWin('ERROR: bad version');
         EXIT
      END;
   SetGraphicCursor(Mouse.Diskette);
   _Nvertex:=0;
   PressButton(_Panel, _S);
   UnPack(@_G, SizeOf(_G), f);
   BlockRead(f, _Gtype, SizeOf(_Gtype));
   BlockRead(f, _S, SizeOf(_S));
   BlockRead(f, _Nvertex, SizeOf(_Nvertex));
   PressArrowButton(_A);
   BlockRead(f, _A, SizeOf(_A));
   PressArrowButton(_A);
   BlockRead(f, _Dlast, SizeOf(_Dlast));
   UnPack(@_D, SizeOf(_D), f);
   Close(f);
   IF NOT (_S IN [11..13]) THEN _S:=12;
   PressButton(_Panel, _S);
   _Path:=p;
   SetGraphicCursor(Mouse.Arrow);
   SetTitle;
   Refresh
END; { LoadFile }

PROCEDURE Pack(m : ArrayOfBytePtr;
               s : WORD;
           VAR f : File);
VAR none : BOOLEAN;
    i, n : WORD;
BEGIN
   none:=FALSE;
   FOR i:=0 TO s-1 DO
      IF m^[i]=0 THEN
         IF none THEN
            Inc(n)
         ELSE BEGIN
            BlockWrite(f, m^[i], 1);
            none:=TRUE;
            n:=0
         END
      ELSE BEGIN
         IF none THEN BEGIN
            BlockWrite(f, n, 2);
            none:=FALSE
         END;
         BlockWrite(f, m^[i], 1)
      END;
   IF none THEN BlockWrite(f, n, 2)
END; { Pack }

PROCEDURE SaveFile(p : PathStr);
VAR f : FILE;
    r : WORD;
BEGIN
   SetGraphicCursor(Mouse.Diskette);
   p:=SetFileExt(p, '.IMG');
   Assign(f, p);
   Rewrite(f,1);
   BlockWrite(f, VER, SizeOf(VER));
   Pack(@_G, SizeOf(_G), f);
   BlockWrite(f, _Gtype, SizeOf(_Gtype));
   BlockWrite(f, _S, SizeOf(_S));
   BlockWrite(f, _Nvertex, SizeOf(_Nvertex));
   BlockWrite(f, _A, SizeOf(_A));
   BlockWrite(f, _Dlast, SizeOf(_Dlast));
   Pack(@_D, SizeOf(_D), f);
   Close(f);
   SetGraphicCursor(Mouse.Arrow)
END; { SaveFile }

{$F+} PROCEDURE PutDrawing(m, x, y : INTEGER); {$F-}
BEGIN
   IF (NOT _L) OR (ffx(x)<0) THEN EXIT;
   IF _L THEN PutDrawings(_Lx2, _Ly2);
   _Lx2:=ffx(x);
   _Ly2:=ffy(y);
   PutDrawings(_Lx2, _Ly2)
END; { PutDrawing }

PROCEDURE Drawing(x, y : INTEGER);
BEGIN
   IF x<0 THEN EXIT;
   IF _L
      THEN BEGIN
         _L:=FALSE;
         IF _Dlast=255 THEN EXIT;
         Inc(_Dlast);
         _D[_Dlast].x1:=_Lx1;
         _D[_Dlast].y1:=_Ly1;
         _D[_Dlast].x2:=x;
         _D[_Dlast].y2:=y;
         _D[_Dlast].d :=_Dline
      END
      ELSE BEGIN
        _Lx1:=x;
        _Ly1:=y;
        _Lx2:=x;
        _Ly2:=y;
        _L:=TRUE
     END
END; { Drawing }

PROCEDURE SuccOfV(n : BYTE);
VAR v : BYTE;
BEGIN
   _Father:=n;
   _Succ:=[];
   Successors(n);
   FOR v:=1 TO _Nvertex DO
      _G[v].r:=NOT (v IN _Succ)
END; { SuccOfV }

PROCEDURE PredOfV(n : BYTE);
VAR v : BYTE;
BEGIN
   _Son:=n;
   _Pred:=[];
   Predecessors(n);
   FOR v:=1 TO _Nvertex DO
      _G[v].r:=NOT (v IN _Pred)
END; { PredOfV }
```

```
PROCEDURE StronglyConnected;
VAR v : BYTE;
    s : ArcSet;
BEGIN
   IF NOT _SCC THEN EXIT;
   IF _S=11
      THEN PressButton(_Panel, 11)
      ELSE IF _S=12
         THEN PressButton(_Panel, 12)
         ELSE IF _S=13
            THEN PressButton(_Panel, 13);
   _S:=13;
   PressButton(_Panel, 13);
   s:=[];
   FOR v:=1 TO _Nvertex DO
      BEGIN
         _Pred:=[];
         _Succ:=[];
         Predecessors(v);
         Successors(v);
         s:=s+((_Pred*_Succ)-[v])
      END;
   FOR v:=1 TO _Nvertex DO _G[v].r:=NOT (v IN s);
   Refresh
END; { StronglyConnected }

{$F+} PROCEDURE MoveVertex(x, y : INTEGER); {$F-}
BEGIN
   x:=ffx(x);
   y:=ffy(y);
   IF _Lin
      THEN BEGIN
         _Dline:=Ln;
         Drawing(x, y)
      END
      ELSE IF _Rec
         THEN BEGIN
            _Dline:=Rec;
            Drawing(x, y)
         END
         ELSE IF _Marked=0
            THEN BEGIN
               REPEAT Inc(_Marked)
               UNTIL (_Nvertex<_Marked) OR
                     ((NOT _G[_Marked].r)    AND
                      (_G[_Marked].x-_dXv<x) AND
                      (x<_G[_Marked].x+_dXv) AND
                      (_G[_Marked].y-_dYv<y) AND
                      (y<_G[_Marked].y+_dYv));
               IF _Nvertex<_Marked
                  THEN _Marked:=0
                  ELSE IF _R
                     THEN BEGIN
                        _G[_Marked].r:=TRUE;
                        _Marked:=0;
                        Refresh
                     END
                     ELSE IF _GamaS OR _GamaP THEN
                        BEGIN
                           SetGraphicCursor(Hourglass);
                           IF _GamaS
                              THEN SuccOfV(_Marked)
                              ELSE PredOfV(_Marked);
                           _Marked:=0;
                           Refresh;
                           SetGraphicCursor(Mouse.Arrow)
                        END
                        ELSE IF NOT _G[_Marked].r THEN Reverse
            END
            ELSE IF _PageLeft+_dXv<=x THEN
               BEGIN
                  _G[_Marked].x:=x;
                  _G[_Marked].y:=y;
                  _Marked:=0;
                  Refresh
               END
END; { MoveVertex }

PROCEDURE Align;
VAR v  : BYTE;
BEGIN
```

```
   FOR v:=1 TO _Nvertex DO
      WITH _G[v] DO
         BEGIN
           x:=Round(x/_dXv)*_dXv;
           y:=Round(y/_dYv/2)*_dYv*2
         END;
   Refresh
END; { Align }

PROCEDURE PageLeft;
BEGIN
   IF (dXdraw DIV 8)<=_PageLeft
      THEN BEGIN
         Dec(_PageLeft, dXdraw DIV 8);
         Refresh
      END
END; { PageLeft }

PROCEDURE PageRight;
BEGIN
   IF _PageLeft<_Xmax-(dXdraw DIV 8)
      THEN BEGIN
         Inc(_PageLeft, dXdraw DIV 8);
         Refresh
      END
END; { PageRight }

PROCEDURE PageUp;
BEGIN
   IF _PageHome-(GetMaxY DIV 2)<-maxint THEN EXIT;
   Dec(_pageHome, GetMaxY DIV 2);
   Refresh
END; { PageUp }

PROCEDURE PageDown;
BEGIN
   IF maxint<_PageHome+GetMaxY THEN EXIT;
   Inc(_pageHome, GetMaxY DIV 2);
   Refresh
END; { PageDown }

PROCEDURE ClearLine;
BEGIN
   IF 0<_Dlast
      THEN BEGIN
         FillChar(_D[_Dlast], SizeOf(DrawType), 0);
         Dec(_Dlast);
         Refresh
      END
END; { ClearLine }

PROCEDURE Help;
CONST
    NL   = 41;
    Line : ARRAY[1..NL] OF STRING[65]
         = ('B I O G R A P H - BG ',
            'Copyright (c) 1990 by J. Savary & J. Guex',
            'Institute OF Geology, University OF Lausanne',
            '----------------------------------------------------------------',
            'Representing G, G*, G'#31,
            '----------------------------------------------------------------',
            '',
            'The Gk'#39' graph contains only the immediate arcs',
            '',
            'Mouse : moving the vertices',
            '',
            'SAVE  -> saving the image                                       ',
            'LOAD  -> load any image                                         ',
            'PRINT -> print on any IBM/Epson compatible printer              ',
            '         high resolution (240 dpi) (print=narrow, PRINT=large)',
            '         ESC : interrupt "PRINT"                                ',
            '',
            'G   -> non-orented graph  ',
            'G* -> semi-oriented graph',
            'G'#31' -> oriented graph     ',
            '',
            '      ALIGN -> Align horizontally/vertically                     ',
            '      O / o -> vertices on a large/small circle                  ',
            '          # -> grid display                                      ',
            '1/2 2/3 3/4 -> positionning the arrows                           ',
            '        ALL -> display OF all vertices                           ',
            '        DEL -> delete the vertex choosen by the mouse            ',
                '#24#25' -> display the strongly connected components        ',
            '         _- -> display OF the predecessors OF the choosen vertex',
            '         _+ -> display OF the successors OF the choosen vertex  ',
            '3 last options are only valid FOR the display part OF the graph ',
            '',
            'LINE      -> draw a line                ',
            'RECTANGLE -> draw a rectangle           ',
            'CLEAR     -> clear the last line/rectangle',
            '',
            #17#30#31#16' -> moving the graph',
            '',
            'QUIT -> back TO the main menu',
            '',
            'Press a button TO continue...');

VAR om, m, x, y, dy : INTEGER;
    l               : BYTE;
BEGIN
   Line[1]:=Concat(Copy(Line[1], 1, 21), VER);
   IF _HP THEN BEGIN
      Line[14]:='A4/A3 -> graph plotted on a plotter HP7475A                    ';
      Line[15]:='         Select paper format                                   '
   END;
   HideMouse;
   SetBkColor(_HelpBackColor[_C]);
   ClearViewPort;
   SetTextStyle(DefaultFont, HorizDir, 1);
   SetTextJustify(CenterText, CenterText);
   Graph.SetColor(_HelpColor[_C]);
   dy:=TextHeight(' ');
   FOR l:=1 TO NL DO Graph.OutTextXY((X2draw-X1draw) DIV 2, l*dy+dy, Line[l]);
   REPEAT GetPosBut(m, x, y) UNTIL m=0;
   REPEAT
      GetPosBut(m, x, y);
   UNTIL 0<m;
   SetBkColor(_background[_C]);
   Refresh;
   ShowMouse
END; { Help }

PROCEDURE Restore;
VAR v : BYTE;
BEGIN
   FOR v:=0 TO _Nvertex DO _G[v].r:=FALSE;
   _Father:=0;
   _Son:=0;
   IF _SCC THEN
      BEGIN
         _Scc:=FALSE;
         PressButton(_Panel, 21);
      END;
   Refresh
END; { Restore }

PROCEDURE VCircle(a : BYTE);
CONST
   TWOPI  = 6.283185308;
   HALFPI = 1.570796327;
VAR x0, y0, rx, ry : INTEGER;
    alpha          : REAL;
    v, n, nv       : BYTE;
BEGIN
   nv:=0;
   FOR v:=1 TO _Nvertex DO IF NOT _G[v].r THEN Inc(nv);
   IF nv=0 THEN EXIT;
   rx:=Round(5*nv*Sqrt(Sqr(_dXv)+Sqr(_dYv))/Pi/2);
   IF dXdraw DIV 4<rx THEN rx:=dXdraw DIV 4;
   IF (a=29) AND ((GetMaxY-_dYv) DIV 2<rx) THEN rx:=(GetMaxY-_dYv) DIV 2;
   ry:=Round(9*rx/10);
   x0:=rx+_dXv;
   y0:=ry+_dYv;
   IF x0<(X2draw-X1draw) DIV 2 THEN x0:=(X2draw-X1draw) DIV 2;
   IF y0<(1+GetMaxY) DIV 2 THEN y0:=(1+GetMaxY) DIV 2;
   _PageLeft:=0;
   _PageHome:=0;
   n:=0;
   FOR v:=1 TO _Nvertex DO
      IF NOT _G[v].r THEN
         BEGIN
            alpha:=n*TWOPI/nv+HALFPI;
            _G[v].x:=Round(x0+System.Cos(alpha)*rx);
            _G[v].y:=Round(y0+System.Sin(alpha)*ry);
            Inc(n)
         END;
   Refresh
END; { VCircle }

PROCEDURE OneButton(b : BYTE);
BEGIN
   IF 0<_Marked THEN
      BEGIN
         Reverse;
         _Marked:=0
      END;
   IF _L    THEN PutDrawings(_Lx2, _Ly2);
   _L:=FALSE;
   IF (b<> 5) AND _Lin   THEN PressButton(_Panel,  5);
   IF (b<> 6) AND _Rec   THEN PressButton(_Panel,  6);
   IF (b<>16) AND _R     THEN PressButton(_Panel, 16);
   IF (b<>19) AND _GamaS THEN PressButton(_Panel, 19);
   IF (b<>20) AND _GamaP THEN PressButton(_Panel, 20);
   IF (b<>21) AND _SCC   THEN PressButton(_Panel, 21);
   IF b= 5
      THEN _Lin :=NOT _Lin
      ELSE _Lin :=FALSE;
   IF b= 6
      THEN _Rec :=NOT _Rec
      ELSE _Rec :=FALSE;
   IF b=16
      THEN _R   :=NOT _R
      ELSE _R   :=FALSE;
   IF b=19
      THEN _GamaS:=NOT _GamaS
      ELSE _GamaS:=FALSE;
   IF b=20
      THEN _GamaP:=NOT _GamaP
      ELSE _GamaP:=FALSE;
   IF b=21
      THEN _SCC  :=NOT _SCC
      ELSE _SCC  :=FALSE;
   IF (b<> 5) AND _Lin   THEN PressButton(_Panel,  5);
   IF (b<> 6) AND _Rec   THEN PressButton(_Panel,  6);
   IF (b<>16) AND _R     THEN PressButton(_Panel, 16);
   IF (b<>19) AND _GamaS THEN PressButton(_Panel, 19);
   IF (b<>20) AND _GamaP THEN PressButton(_Panel, 20);
   IF (b<>21) AND _SCC   THEN PressButton(_Panel, 21);
   IF _GamaS THEN _Father:=0;
   IF _GamaP THEN _Son   :=0
END; { OneButton }

{$F+} PROCEDURE Events(a : BYTE); {$F-}
VAR p : PathStr;
BEGIN
   IF a IN [3,21,22] THEN SetGraphicCursor(Hourglass);
   CASE a OF
      1 : IF GetGrFilename(_Path+'*.IMG', p, 'Save file [.IMG]: ')
             THEN SaveFile(p);
      2 : IF GetGrFilename(_Path+'*.IMG', p, 'Load file [.IMG]: ')
             THEN LoadFile(p);
      3, 22 : BEGIN
             IF a=3
                THEN _Pratio:=2
                ELSE _Pratio:=3;
             Print
          END;
      4 : Align;
      5, 6, 16, 19, 20 : OneButton(a);
      7 : ClearLine;
      8 : PageDown;
      9 : PageUp;
     10 : Help;
     11, 12, 13 : BEGIN
             IF _S=11
                THEN PressButton(_Panel, 11)
                ELSE IF _S=12
                   THEN PressButton(_Panel, 12)
                   ELSE IF _S=13
                      THEN PressButton(_Panel, 13);
             _S:=a;
             Refresh
          END;
     14 : PageRight;
     15 : PageLeft;
     17 : Restore;
     18, 29 : VCircle(a);
     21 : BEGIN
             OneButton(a);
             StronglyConnected
          END;
     23 : _PI:=(_PI+1) MOD 5;
     24 : BEGIN
             _P[_PI,1]:=(_P[_PI,1]+1) MOD 64;
             SetPalette(_P[_PI,0], _P[_PI,1])
          END;
     25 : BEGIN
             _P[_PI,1]:=(_P[_PI,1]-1) MOD 64;
             SetPalette(_P[_PI,0], _P[_PI,1])
          END;
     26, 27, 28 : ArrowAspect(a);
     30 : BEGIN
             _Grid:=NOT _Grid;
             Refresh
          END;
     31 : Plotter(7200, 10000);
     32 : Plotter(10000, 15200)
   END;
   IF a IN [3,21,22] THEN SetGraphicCursor(Mouse.Arrow)
END; { Events }

PROCEDURE InitScreen;
VAR dx2, dx3, dx4, dy : INTEGER;
BEGIN
   InitGraphicMode;
   HideMouse;
   Graph.SetColor(_ButtonFrame[_C]);
   Graph.Line(X1draw-1,0,X1draw-1,GetMaxY);
   SetFillStyle(SolidFill, _BackgroundButton[_C]);
   FloodFill(1,1,_ButtonFrame[_C]);
   dx2:=(X2butt-X1butt) DIV 2;
   dx3:=(X2butt-X1butt) DIV 3;
   dx4:=(X2butt-X1butt) DIV 4;
   dy:=dYbutt DIV 2;
   CreatePanel(_Panel, 33, Events, MoveVertex, PutDrawing);
   CreateButton(_Panel, X1butt,       1*dYbutt,
                        X1butt+dx2-1           , 'Save'   , 1, FALSE);
   CreateButton(_Panel, X1butt+dx2+1, 1*dYbutt,
                        X2butt                 , 'Load'   , 2, FALSE);
   IF _HP THEN
   BEGIN
      CreateButton(_Panel, X1butt,       2*dYbutt,
                           X1butt+dx2-1           , 'A4', 31, FALSE);
      CreateButton(_Panel, X1butt+dx2+1, 2*dYbutt,
                           X2butt                 , 'A3', 32, FALSE)
   END
   ELSE BEGIN
      CreateButton(_Panel, X1butt,       2*dYbutt,
                           X1butt+dx2-1           , 'print', 3, FALSE);
      CreateButton(_Panel, X1butt+dx2+1, 2*dYbutt,
                           X2butt                 , 'PRINT',22, FALSE)
   END;
   CreateButton(_Panel, X1butt,       3*dYbutt+dy,
                        X1butt-1+dx3           , 'G'      ,11, TRUE );
   CreateButton(_Panel, X1butt+1+dx3, 3*dYbutt+dy,
                        X2butt-1-dx3           , 'G*'     ,12, TRUE );
   CreateButton(_Panel, X2butt+1-dx3, 3*dYbutt+dy,
                        X2butt                 , 'G'#31   ,13, TRUE );
   CreateButton(_Panel, X1butt,       5*dYbutt,
                        X1butt+dx2-1           , 'Align'  , 4, FALSE);
   CreateButton(_Panel, X1butt+dx2+1, 5*dYbutt,
                        X2butt-dx4-1           , 'O'      ,18, FALSE);
   CreateButton(_Panel, X2butt-dx4+1, 5*dYbutt,
                        X2butt                 , 'o'      ,29, FALSE);
   CreateButton(_Panel, X1butt,       6*dYbutt,
                        X1butt-1+dx3           , '1/2'    ,26, TRUE );
   CreateButton(_Panel, X1butt+1+dx3, 6*dYbutt,
                        X2butt-1-dx3           , '2/3'    ,27, TRUE );
   CreateButton(_Panel, X2butt+1-dx3, 6*dYbutt,
                        X2butt                 , '3/4'    ,28, TRUE );
   CreateButton(_Panel, X1butt      , 7*dYbutt,
                        X1butt+dx3-1           , 'All'    ,17, FALSE);
   CreateButton(_Panel, X1butt+dx3+1, 7*dYbutt,
                        X2butt-dx3-1           , 'Del'    ,16, TRUE );
   CreateButton(_Panel, X2butt-dx3+1, 7*dYbutt,
                        X2butt                 , '#'      ,30, TRUE );
   CreateButton(_Panel, X1butt      , 8*dYbutt,
                        X1butt+dx3-1           , #24#25   ,21, TRUE );
   CreateButton(_Panel, X1butt+dx3+1, 8*dYbutt,
                        X2butt-dx3-1          , '_-'     ,20, TRUE );
   CreateButton(_Panel, X2butt-dx3+1, 8*dYbutt,
                        X2butt                , '_+'     ,19, TRUE );
   CreateButton(_Panel, X1butt,       9*dYbutt+dy,
                        X1butt+dx2-1          , 'Line'   , 5, TRUE );
   CreateButton(_Panel, X1butt+dx2+1, 9*dYbutt+dy,
                        X2butt                , 'Rect'   , 6, TRUE );
   CreateButton(_Panel, X1butt      ,10*dYbutt+dy,
                        X2butt                , 'Clear'  , 7, FALSE);
   CreateButton(_Panel, X1butt+dx3+1,12*dYbutt,
                        X2butt-dx3-1          , #30      , 8, FALSE);
   CreateButton(_Panel, X1butt+dx3+1,13*dYbutt,
                        X2butt-dx3-1          , #31      , 9, FALSE);
   CreateButton(_Panel, X1butt      ,12*dYbutt+dy,
                        X1butt+dx3-1          , #17      ,14, FALSE);
   CreateButton(_Panel, X2butt-dx3+1,12*dYbutt+dy,
                        X2butt                , #16      ,15, FALSE);
   CreateButton(_Panel, X1butt      ,15*dYbutt,
                        X1butt+dx3-1          , '?'      ,10, FALSE);
   CreateButton(_Panel, X1butt+dx3+1,15*dYbutt,
                        X2butt                , 'QUIT'   , 0, FALSE);
   CreateButton(_Panel, X1butt,      21*dYbutt+dy,
                        X1butt-1+dx3          , 'C'      ,23, FALSE);
   CreateButton(_Panel, X1butt+1+dx3,21*dYbutt+dy,
                        X2butt-1-dx3          , #24      ,24, FALSE);
   CreateButton(_Panel, X2butt+1-dx3,21*dYbutt+dy,
                        X2butt                , #25      ,25, FALSE);
   PressButton(_Panel, 26);
   SetBkColor(_Background[_C]);
   CreateVertex;
   SetViewPort(X1draw,0,X2draw,GetMaxY,ClipOn);
   SetGraphicCursor(HourGlass);
   ShowMouse
END; { InitScreen }

PROCEDURE Initialization;
VAR cmd : ComStr;
    f   : file;
BEGIN
   cmd:=ParamStr(1);
   IF cmd[1]='-' THEN BEGIN
      new(_Cfg);
      IF cmd[2]='B' THEN      _Cfg^.Run:=Grb
      ELSE IF cmd[2]='K' THEN _Cfg^.Run:=Grk
      ELSE IF Cmd[2]='R' THEN _Cfg^.Run:=Grr
      ELSE IF Cmd[2]='T' THEN _Cfg^.Run:=GoAll
      ELSE HALT(601);
      _Path:=UpString(paramstr(2));
      cmd:=UpString(ParamStr(3));
      _HP:=cmd[1]='P';
      _Title:=_Path
   END
   ELSE BEGIN
      _Cfg:=StringToAddr;
      _C:=_Cfg^.color;
      _Path:=_Cfg^.Epath+_Cfg^.Ename;
      _HP:=BG_Def.Plot in _Cfg^.Setup;
      assign(f, _Path+'.LEX');
      reset(f, 1);
      blockread(f, _Title, sizeof(_Title));
      close(f)
   END;
   FillChar(_D, SizeOf(_D), 0)
END; { Initialization }

BEGIN { BG_DRAW }
   InitFatalError(FatalError);
   ClrScr;
   Initialization;
   InitScreen;
   CASE _Cfg^.Run OF
      Grb   : LoadMatA;
      Grk   : LoadMatH;
      Grr   : LoadMatR;
      GoAll : LoadTest
   END;
   SetTitle;
   PressButton(_Panel, _S);
   VCircle(29);
   SetGraphicCursor(Mouse.Arrow);
   ActivePanel(_Panel);
   ClearPanel(_Panel, _BackgroundButton[_C]);
   CloseGraph;
END. { BG_DRAW }
```

## *Library BG_DEF*

Function : definitions used in BG, BG_MENU, BG_DATA, BG_CONV and BG_DRAW

```
UNIT BG_DEF;

{$A+,B-,D-,E+,F-,G+,I-,L-,N-,O-,R-,S+,V-,X+}

INTERFACE

uses Dos;

TYPE

        Menu

   OPTIONS    = (All, Trans, ArRe, Ar, Re, First, Last, Print,
                 Plot, ASCII, Light, Dark, Gb, Gk, Gr);
   OPTIONSET = SET OF OPTIONS;
   RUNTYPE    = (Quit, GoAll, GoTrans, Edit,
                 Grb, Grk, Grr, ConvDS, ConvSD);
   CFG        = RECORD
                   DPath, IPath, OPath, EPath : PathStr;
                   DName, IName, OName, Ename : NameStr;
                   DExt,  IExt,  OExt , Eext  : ExtStr;
                   InitStr                    : PathStr;
                   Width                      : BYTE;
                   Output                     : WORD;
                   Setup                      : OPTIONSET;
                   Run                        : RUNTYPE;
                   ToolsFlag, Color           : BOOLEAN
                END;

   CFGPTR     = ^CFG;

        Types of data

   HEAD_REC = RECORD
                 Section, Level, Cardinal : WORD
              END;
   HEAD_PTR = ^HEAD_REC;

   Str5          = String[ 5];
   Str25         = String[25];
   TaxaList      = ARRAY[1..500] OF Str5;
   TaxaLPtr      = ^TaxaList;
   SectionList   = ARRAY[1..1000] OF Str25;
   SectionLPtr   = ^SectionList;
   DataSet       = ARRAY[0..1] OF set OF BYTE;
   LevelType     = record
                      Section, Level, Cardinal : WORD;
                      Taxa                     : DataSet
                   END;
   LevelPtr      = ^LevelType;
   IntPtr        = ^INTEGER;
   AUrec         = record
                      taxa, fad, lad : WORD
                   END;
   AUptr         = ^AUrec;
   CorrRec       = record
                      Section, Level, Fau, Lau : WORD;
                   END;
   CorrPtr       = ^CorrRec;
   CliqueRec     = record
                      Section, Level : WORD
                   END;
   CliqueList    = ARRAY[1..$3FFF] OF CliqueRec;
   CliquePtr     = ^CliqueList;

const
```

```
   EDGE = -32768;

VAR
   {$F+} _Cfg : CFGPTR; {$F-}
```

Interface

```
FUNCTION AddrToString(VAR address) : String;
FUNCTION StringToAddr : POINTER;
```

Memory manager

```
Function LevelSize(taxa : WORD) : WORD;

IMPLEMENTATION

FUNCTION AddrToString(VAR address) : String;
VAR segment, offset : STRING;
BEGIN
   Str(Seg(address), segment);
   Str(Ofs(address), offset );
   AddrToString:=Concat(segment, ' ', offset)
END; { AddrToString }

{$F+}
FUNCTION StringToAddr : POINTER;
VAR segment, offset, code : WORD;
BEGIN
   Val(ParamStr(1), segment, code);
   IF code<>0 THEN HALT(50);
   Val(ParamStr(2), offset, code);
   IF code<>0 THEN HALT(50);
   StringToAddr:=Ptr(segment, offset)
END; { StringToAddr }
{$F-}

Function LevelSize(taxa : WORD) : WORD;
BEGIN
   LevelSize:=SizeOf(WORD)*(((taxa-1) SHR 4)+1)+SizeOf(HEAD_REC)
END; { LevelSize }

END. { BG_DEF }
```

## *Library BG_SCR*

Function : screen manager of BG_MENU

```
UNIT BG_SCR;

{$A+,B-,D-,E+,F-,G+,I-,L-,N-,O-,R-,S+,V-,X+}
{$M 1024, 0, 0 }

INTERFACE

USES Crt, Dos, Video;

PROCEDURE Frontispice(Color  : BOOLEAN);

IMPLEMENTATION

TYPE STR78     = STRING[78];
     ARROW_REC = RECORD
                    X, Y : BYTE;
                    A, B : CHAR;
                    C    : BYTE
                 END;
```

Tatoo of the copy

```
CONST License : ARRAY[1..3] OF STR78 =
      ('kièÅèÅöïÇ; Ç;eIn|æ|iöG;dëÄÅäÅEÅ; Ç;b¥èçèeäÇG;pëäæÇiÄäÅ¥; Ç;g|EÄ|ëëÇ',
       'lÄïEïEòîü<Çü<fJo}Æ}ÄòH<eèÅEàEæE<Çü<cPïêïâàüH<qèàÆüÄÅàEP<Çü<h}æÅ}èèü',
       'Prototype de J.Savary, Institut de Geologie, Universite de Lausanne');
      Serial : STRING[10] = '0000F-1.00';
```

```
Pirate : STRING[20] = ^U^N^A^U^T^H^O^R^I^Z^E^D'_'^C^O^P^Y'_ß';

Disk     : STRING[3] = 'C:\';
DiskTime : LONGINT   = $FEFEFEFE;
```

## Animation

```
Arrow : ARRAY[0..116] OF ARROW_REC =
        ((X:75; Y:15; A:^P; B:'-'; C:Red       ),
         (X:76; Y:15; A:^P; B:'+'; C:Red       ),
         (X:77; Y:15; A:^P; B:'-'; C:Red       ),
         (X:78; Y:15; A:^P; B:'+'; C:Blue      ),
         (X:78; Y:14; A:^^; B:'_'; C:Blue      ),
         (X:78; Y:13; A:^^; B:'_'; C:Blue      ),
         (X:78; Y:12; A:^^; B:'_'; C:Blue      ),
         (X:78; Y:11; A:^^; B:^^ ; C:Blue      ),
         (X:78; Y:10; A:^^; B:'_'; C:Blue      ),
         (X:78; Y: 9; A:^^; B:'_'; C:Blue      ),
         (X:78; Y: 8; A:^^; B:'_'; C:Blue      ),
         (X:78; Y: 7; A:^^; B:'_'; C:Blue      ),
         (X:78; Y: 6; A:^^; B:'_'; C:Blue      ),
         (X:78; Y: 5; A:^^; B:'_'; C:Blue      ),
         (X:78; Y: 4; A:^Q; B:'+'; C:Blue      ),
         (X:77; Y: 4; A:^Q; B:'-'; C:Blue      ),
         (X:76; Y: 4; A:^Q; B:'-'; C:Blue      ),
         (X:75; Y: 4; A:^Q; B:'-'; C:Blue      ),
         (X:74; Y: 4; A:^Q; B:' '; C:Blue      ),
         (X:73; Y: 4; A:^Q; B:^Q ; C:Blue      ),
         (X:72; Y: 4; A:^Q; B:' '; C:Blue      ),
         (X:71; Y: 4; A:^Q; B:'-'; C:Blue      ),
         (X:70; Y: 4; A:^Q; B:'-'; C:Blue      ),
         (X:69; Y: 4; A:^Q; B:'-'; C:Blue      ),
         (X:68; Y: 4; A:^Q; B:'+'; C:Blue      ),
         (X:68; Y: 5; A:^_; B:'_'; C:Blue      ),
         (X:68; Y: 6; A:^_; B:'+'; C:Cyan      ),
         (X:68; Y: 7; A:^Q; B:'-'; C:Cyan      ),
         (X:67; Y: 7; A:^Q; B:'-'; C:LightGray),
         (X:66; Y: 7; A:^Q; B:'+'; C:LightGray),
         (X:65; Y: 7; A:^Q; B:'-'; C:LightGray),
         (X:64; Y: 7; A:^Q; B:'3'; C:LightGray),
         (X:63; Y: 7; A:^Q; B:'-'; C:LightGray),
         (X:62; Y: 7; A:^Q; B:'+'; C:LightGray),
         (X:61; Y: 7; A:^Q; B:'-'; C:LightGray),
         (X:61; Y: 8; A:^_; B:'+'; C:LightGray),
         (X:61; Y: 9; A:^_; B:'_'; C:Cyan      ),
         (X:61; Y:10; A:^P; B:'+'; C:Blue      ),
         (X:62; Y:10; A:^P; B:'-'; C:Blue      ),
         (X:63; Y:10; A:^P; B:^P ; C:Blue      ),
         (X:64; Y:10; A:^P; B:' '; C:Blue      ),
         (X:65; Y:10; A:^P; B:'i'; C:Blue      ),
         (X:66; Y:10; A:^P; B:' '; C:Blue      ),
         (X:67; Y:10; A:^P; B:^P ; C:Blue      ),
         (X:68; Y:10; A:^P; B:'-'; C:Blue      ),
         (X:69; Y:10; A:^P; B:'+'; C:Blue      ),
         (X:69; Y:11; A:^_; B:'-'; C:Blue      ),
         (X:68; Y:11; A:^Q; B:'-'; C:Blue      ),
         (X:67; Y:11; A:^Q; B:'-'; C:Blue      ),
         (X:66; Y:11; A:^Q; B:'-'; C:Blue      ),
         (X:65; Y:11; A:^Q; B:'-'; C:Blue      ),
         (X:64; Y:11; A:^Q; B:'+'; C:Blue      ),
         (X:64; Y:12; A:^_; B:'+'; C:Magenta   ),
         (X:64; Y:13; A:^_; B:'+'; C:Brown     ),
         (X:64; Y:14; A:^_; B:'8'; C:Brown     ),
         (X:64; Y:15; A:^_; B:'+'; C:Brown     ),
         (X:64; Y:16; A:^_; B:'+'; C:Red       ),
         (X:64; Y:17; A:^Q; B:'-'; C:Blue      ),
         (X:63; Y:17; A:^Q; B:'-'; C:Blue      ),
         (X:62; Y:17; A:^Q; B:'-'; C:Blue      ),
         (X:61; Y:17; A:^Q; B:'-'; C:Blue      ),
         (X:60; Y:17; A:^Q; B:'-'; C:Blue      ),
         (X:59; Y:17; A:^Q; B:'-'; C:Blue      ),
         (X:58; Y:17; A:^Q; B:'-'; C:Blue      ),
         (X:57; Y:17; A:^Q; B:'-'; C:Blue      ),
         (X:56; Y:17; A:^Q; B:' '; C:Blue      ),
         (X:55; Y:17; A:^Q; B:^Q ; C:Blue      ),
         (X:54; Y:17; A:^Q; B:' '; C:Blue      ),
         (X:53; Y:17; A:^Q; B:'-'; C:Blue      ),
         (X:52; Y:17; A:^Q; B:'-'; C:Blue      ),
         (X:51; Y:17; A:^Q; B:'-'; C:Blue      ),
         (X:50; Y:17; A:^Q; B:'+'; C:Blue      ),
         (X:50; Y:16; A:^^; B:'_'; C:Blue      ),
        (X:50; Y:16; A:^^; B:'_'; C:Blue      ),
        (X:50; Y:15; A:^^; B:'_'; C:Blue      ),
        (X:50; Y:14; A:^^; B:'_'; C:Blue      ),
        (X:50; Y:13; A:^^; B:'_'; C:Blue      ),
        (X:50; Y:12; A:^^; B:'_'; C:Blue      ),
        (X:50; Y:11; A:^^; B:'_'; C:Blue      ),
        (X:50; Y:10; A:^^; B:^^ ; C:Blue      ),
        (X:50; Y: 9; A:^^; B:'_'; C:Blue      ),
        (X:50; Y: 8; A:^^; B:'_'; C:Blue      ),
        (X:50; Y: 7; A:^^; B:'_'; C:Blue      ),
        (X:50; Y: 6; A:^P; B:'+'; C:Blue      ),
        (X:51; Y: 6; A:^P; B:'-'; C:Green     ),
        (X:52; Y: 6; A:^P; B:'+'; C:Green     ),
        (X:53; Y: 6; A:^P; B:'-'; C:Green     ),
        (X:54; Y: 6; A:^P; B:'k'; C:Green     ),
        (X:54; Y: 7; A:^P; B:'+'; C:Green     ),
        (X:55; Y: 7; A:^P; B:'-'; C:Green     ),
        (X:56; Y: 7; A:^P; B:'-'; C:Green     ),
        (X:57; Y: 7; A:^P; B:'-'; C:Green     ),
        (X:58; Y: 7; A:^P; B:'-'; C:Green     ),
        (X:59; Y: 7; A:^P; B:'-'; C:Green     ),
        (X:60; Y: 7; A:^P; B:'-'; C:Green     ),
        (X:61; Y: 7; A:^P; B:'-'; C:LightGray),
        (X:61; Y: 8; A:^_; B:'+'; C:LightGray),
        (X:61; Y: 9; A:^_; B:'_'; C:Cyan      ),
        (X:61; Y:10; A:^P; B:'+'; C:Blue      ),
        (X:62; Y:10; A:^P; B:'-'; C:Blue      ),
        (X:63; Y:10; A:^P; B:^P ; C:Blue      ),
        (X:64; Y:10; A:^P; B:' '; C:Blue      ),
        (X:65; Y:10; A:^P; B:'i'; C:Blue      ),
        (X:66; Y:10; A:^P; B:' '; C:Blue      ),
        (X:67; Y:10; A:^P; B:^P ; C:Blue      ),
        (X:68; Y:10; A:^P; B:'-'; C:Blue      ),
        (X:69; Y:10; A:^P; B:'+'; C:Blue      ),
        (X:69; Y:11; A:^_; B:'-'; C:Blue      ),
        (X:70; Y:11; A:^P; B:'-'; C:Blue      ),
        (X:71; Y:11; A:^P; B:'-'; C:Blue      ),
        (X:72; Y:11; A:^P; B:'-'; C:Blue      ),
        (X:73; Y:11; A:^P; B:'-'; C:Blue      ),
        (X:74; Y:11; A:^P; B:'+'; C:Blue      ),
        (X:74; Y:12; A:^_; B:'_'; C:Blue      ),
        (X:74; Y:13; A:^_; B:'+'; C:Red       ),
        (X:74; Y:14; A:^_; B:'_'; C:Red       ),
        (X:74; Y:15; A:^_; B:'j'; C:Red       ));
```

## Object files containing the screen image

```
PROCEDURE TITLE_C; EXTERNAL;
{$L BG_Scr_C.obj}

PROCEDURE TITLE_M; EXTERNAL;
{$L BG_Scr_M.obj}

PROCEDURE Frontispice;
VAR i   : BYTE;
    ok  : BOOLEAN;
    msg : STR78;
    dt  : SearchRec;
BEGIN
   CheckBreak:=FALSE;
   CheckSnow:=FALSE;
   IF ParamStr(3)<>'S' THEN EXIT;
   FindFirst(Concat(Disk, '*.*'), VolumeID, dt);
   IF (0<DosError) OR (dt.Time<>DiskTime)
      THEN HALT(399)
      ELSE BEGIN
         ok:=(                 0<Length(License[3])) AND
             (Length(License[1])=Length(License[2])) AND
             (Length(License[1])=Length(License[3])) AND
             (Length(License[2])=Length(License[3]));
         FOR i:=1 TO Length(License[1]) DO
            IF (Ord(License[1][i])<>Ord(License[2][i])- 1) OR
               (Ord(License[1][i])<>Ord(License[3][i])+27)
               THEN ok:=FALSE;
         IF Color
            THEN BEGIN
               Move(Ptr(Seg(Title_C), Ofs(Title_C))^, _S^, 4000);
               TextColor(Yellow);
               TextBackGround(LightGray)
            END
            ELSE BEGIN
```

```
                  Move(Ptr(Seg(Title_M), Ofs(Title_M))^, _S^, 4000);
                  TextColor(White);
                  TextBackGround(Black)
            END;
         IF ok
            THEN BEGIN
               GotoXY((82-Length(License[3])) DIV 2,22);
               Write(License[3]);
               IF Color THEN TextColor(White);
               i:=0;
               REPEAT
                  IF Color THEN TextBackGround(Arrow[i].C);
                  GotoXY(Arrow[i].X, Arrow[i].Y);
                  Write(Arrow[i].B);
                  i:=(i+1) MOD 117;
                  IF Color THEN TextBackGround(Arrow[i].C);
                  GotoXY(Arrow[i].X, Arrow[i].Y);
                  Write(Arrow[i].A);
                  IF Arrow[i].A IN [^^, ^_]
                     THEN Delay(40)
                     ELSE Delay(20)
               UNTIL (i=200) OR (Keypressed and (ReadKey<>#0));
               FOR i:=1 TO 25 DO
                  BEGIN
                     CASE i OF
                       6 : msg:='Serial number #'+Serial
                       ELSE msg:=''
                     END;
                     GotoXY(1, i);
                     InsLine;
                     GotoXY((82-Length(msg)) DIV 2,i-1);
                     Write(msg);
                     Delay(20)
                  END;
               Delay(500)
            END
            ELSE BEGIN
               FOR i:=1 TO Length(Pirate) DO Pirate[i]:=Chr(Ord(Pirate[i])+64);
               GotoXY(31,22); Write(Pirate);
               REPEAT
                  FOR i:=1 TO 255 DO
                     BEGIN
                        Sound(i*12);
                        Delay(2)
                     END;
                  FOR i:=255 DOWNTO 1 DO
                     BEGIN
                        Sound(i*12);
                        Delay(2)
                      END;
               UNTIL 1<i
            END
      END
END; { Frontispice }

END.
```

## 5.4 Interfaces Pascal Library

### *Library TOOLS*

Function : librairies for the tools, for the management of inputs/outputs and for the memory management

```
UNIT Tools;

INTERFACE

USES Crt, Dos;

TYPE
   Str3      = STRING[ 3];
   Str5       = STRING[ 5];
   Str8       = STRING[ 8];
   Str15      = STRING[15];
   Str35      = STRING[35];
   Str64      = STRING[64];
   Str74      = STRING[74];
   Str80      = STRING[80];
   StrPtr     = ^STRING;

   AnyChar    = SET OF CHAR;
   AnyByte    = SET OF BYTE;

   LngIntPtr = ^LONGINT;
   RealPtr    = ^REAL;
```

Type of procedures of sorting to be used with HSort.

```
   LessFunc   = FUNCTION( X, Y : INTEGER): BOOLEAN;
```

Type of procedures of permutation, to be used with HSort.

```
   SwapProc   = PROCEDURE(X, Y : INTEGER);

   ArrayOfByte     = ARRAY[0..$FFFE] OF BYTE;
   ArrayOfBytePtr = ^ArrayOfByte;

   FileOfChar  = FILE OF CHAR;
```

Types of untyped files where:
f : handle
fsize : usual size on the HD
ptr : pointer in the buffer
bsize : size of the buffer
blen : content of the buffer
buffer : buffer

```
   UntypedFile = RECORD
                    f                  : FILE;
                    fsize              : LONGINT;
                    Ptr, bsize, blen : WORD;
                    buffer             : ArrayOfBytePtr
                 END;

CONST
```

Key code

```
   NUL  = $00;  BS   = $08;  TAB  = $09;  CR   = $0D;  ESC  = $1B;
```

Key scan code

```
   STAB = $0F;

   HOME = $47;  UP   = $48;  PGUP = $49;  CHOM = $77;  CPUP = $84;
   LEFT = $4B;               RGHT = $4D;  CLFT = $73;  CRHT = $74;
   ENDK = $4F;  DOWN = $50;  PGDN = $51;  CEND = $75;  CPDN = $76;
   INS  = $52;  DEL  = $53;

                { SHIFT + }  { CTRL + }   { ALT + }
   F1   = $3B;  SF1  = $54;  CF1  = $5E;  AF1  = $68;
   F2   = $3C;  SF2  = $55;  CF2  = $5F;  AF2  = $69;
   F3   = $3D;  SF3  = $56;  CF3  = $60;  AF3  = $6A;
   F4   = $3E;  SF4  = $57;  CF4  = $61;  AF4  = $6B;
   F5   = $3F;  SF5  = $58;  CF5  = $62;  AF5  = $6C;
   F6   = $40;  SF6  = $59;  CF6  = $63;  AF6  = $6D;
   F7   = $41;  SF7  = $5A;  CF7  = $64;  AF7  = $6E;
   F8   = $42;  SF8  = $5B;  CF8  = $65;  AF8  = $6F;
   F9   = $43;  SF9  = $5C;  CF9  = $66;  AF9  = $70;
   F10  = $44;  SF10 = $5D;  CF10 = $67;  AF10 = $71;
```

Size of the buffers in the untyped files

```
   BufferSize : WORD = $8000;
```

Variable containing the code number of the error occurring within the unit

```
   _TError : BYTE = 0;
```

```
VAR
   _UserMemory : LONGINT;
   _UserHeap   : POINTER;

PROCEDURE FillWord( VAR V; Num, Value : WORD );
PROCEDURE Exchange( VAR S, D; L : WORD);
FUNCTION Inkey : WORD;
PROCEDURE Pause;
PROCEDURE WrRepChar( x, y : BYTE;
                     c    : CHAR;
                     n    : BYTE );
FUNCTION UpString( s : STRING ) : STRING;
FUNCTION SetFileExt(p : PathStr;
                    e : ExtStr) : PathStr;
FUNCTION PSplit( Path : STRING ) : STRING;
FUNCTION NSplit( Path : STRING ) : STRING;
FUNCTION FileExist( Path : PathStr ) : BOOLEAN;
FUNCTION TextSize( path : PathStr ) : LONGINT;
FUNCTION PrinterOk : BOOLEAN;
PROCEDURE HSort( Less : LessFunc;
                 Swap : SwapProc;
                 Max  : INTEGER );
PROCEDURE GetMemory( Get, Let : LONGINT );
FUNCTION AddAddr( p : POINTER;
                  s : LONGINT ) : POINTER;
FUNCTION RdItem(VAR f                : FileOfChar;
                    token            : AnyChar;
                    begC, endC, Str  : CHAR;
                VAR nline            : WORD;
                VAR pos              : LONGINT) : STRING;
PROCEDURE ROpenUF(VAR f : UntypedFile;
                      p : PathStr);
PROCEDURE WOpenUF(VAR f : UntypedFile;
                      p : PathStr);
PROCEDURE RCloseUF(VAR f : UntypedFile);
PROCEDURE WCloseUF(VAR f : UntypedFile);
PROCEDURE ReadUF(VAR f      : UntypedFile;
                 VAR b      ;
                     count  : WORD;
                 VAR result : WORD);
PROCEDURE WriteUF(VAR f      : UntypedFile;
                  VAR b      ;
                      count  : WORD;
                  VAR result : WORD);
FUNCTION EoUF(f : UntypedFile) : BOOLEAN;

IMPLEMENTATION
```

Utilities

Fill any variable Num times with the value Value. Equivalent to FillChar.

```
PROCEDURE FillWord;
BEGIN
   INLINE($C4/$BE/V          { LES DI, V[BP]       }
         /$8B/$8E/Num        { MOV CX, [Num+BP]    }
         /$8B/$86/Value      { MOV AX, [Value+BP]  }
         /$FC                { CLD                 }
         /$F3/$AB)           { REPNZ STOSW         }
END; { FillWord }
```

Fast permutation of the content of two variables S and D. L is the number of bytes to be exchanged. Complementary to Move.

```
PROCEDURE Exchange;
BEGIN
   INLINE($1E                { PUSH DS              }
         /$C5/$B6/S          { LDS SI, S[BP]        }
         /$C4/$BE/D          { LES DI, D[BP]        }
         /$8B/$8E/L          { MOV CX, [L+BP]       }
         /$FC                { CLD                  }
         /$26/$8A/$05        { L: MOV AL, ES: [DI]  }
         /$86/$04            { EXCH [SI], AL        }
         /$46                { INC SI               }
         /$AA                { STOSB                }
         /$E2/$F7           { LOOP L                }
         /$1F)              { POP DS                }
END; { Exchange }
```

Inkey return the typed key:
  LO(word) : normal characters
  HI(word) :  special characters
              (cursors, functions, ALT+, etc..) ;
              See the pre-defined constants

```
FUNCTION Inkey;
VAR Key : CHAR;
BEGIN
   Key:=ReadKey;
   IF Key=#0
      THEN BEGIN
         Key:=ReadKey;
         Inkey:=Ord(Key) SHL 8
      END
      ELSE Inkey:=Ord(Key)
END; { Inkey }
```

Generates a pause until a key is typed.

```
PROCEDURE Pause;
BEGIN
   REPEAT UNTIL ReadKey<>#0
END; { Pause }
```

Writes n times the character c in the position x, y.

```
PROCEDURE WrRepChar;
BEGIN
   GotoXY(x, y);
   FOR x:=1 TO n DO Write(c)
END; { WrRepChar }
```

Converts a string of characters into capitals.

```
FUNCTION UpString;
VAR p : BYTE;
BEGIN
   FOR p:=1 TO Length(s) DO s[p]:=UpCase(s[p]);
   UpString:=s
END; { UpString }
```

Restitutes the path <p> to which an extension <e> is added if the path contains no extension.

```
FUNCTION SetFileExt(p : PathStr;
                    e : ExtStr) : PathStr;
VAR d : DirStr;
    n : NameStr;
    x : ExtStr;
BEGIN
   FSplit(p, d, n, x);
   IF x=''
      THEN SetFileExt:=Concat(d, n, e)
      ELSE SetFileExt:=p
END; { SetFileExt }
```

Extract only the path part of a file path.

```
FUNCTION PSplit;
BEGIN
   REPEAT
      Delete(Path, Length(Path), 1);
   UNTIL Path[Length(Path)]='\';
   PSplit:=Path
END; { PSplit }
```

Extract the name of a file from a complete path.

```
FUNCTION NSplit;
VAR n : PathStr;
```

```
BEGIN
   n:=Path;
   Path:=PSplit(Path);
   NSplit:=Copy(n, Length(Path)+1, 12)
END; { NSplit }
```

Returns TRUE if the file does exist and FALSE otherwise. Uses the variable PATH of the environment.

```
FUNCTION FileExist;
BEGIN
   FileExist:=FSearch(Path, GetEnv('PATH'))<>''
END; { FileExist }
```

Returns the text file size in bytes.
Be sure that this file does exist before calling that function.

```
FUNCTION TextSize;
VAR f : FILE;
BEGIN
   Assign(f, path);
   Reset(f, 1);
   TextSize:=FileSize(f);
   {$I-} Close(f); {$I+}
   _TError:=IOResult;
   IF _TError<>0 THEN EXIT
END; { TextSize }
```

Tells you if the printer on port 1 is ready

```
FUNCTION PrinterOk;
VAR r : Registers;
BEGIN
   r.AH:=2;
   r.DX:=0;
   Intr($17, r);
   PrinterOk:=r.AH=$90
END; { PrinterCheck }
```

Sorting procedure using the HeapSort algorithm

```
PROCEDURE HSort;

VAR Level : INTEGER;

   PROCEDURE DownLevel(Node, Max : INTEGER);
   VAR Branch  : INTEGER;
       NotLeaf : BOOLEAN;
   BEGIN
      Branch:=2*Node;
      NotLeaf:=TRUE;
      WHILE (Branch<=Max) AND NotLeaf DO
         BEGIN
            IF (Branch<Max) AND Less(Branch, Branch+1) THEN Inc(Branch);
            IF Less(Node, Branch)
               THEN BEGIN
                  Swap(Node, Branch);
                  Node:=Branch;
                  Branch:=2*Branch
               END
               ELSE NotLeaf:=FALSE
         END
   END; { DownLevel }

BEGIN { HSort }
   IF Max<2 THEN EXIT;
   FOR Level:=Max DIV 2 DOWNTO 1 DO DownLevel(Level, Max);
   FOR Level:=Max DOWNTO 3 DO
      BEGIN
         Swap(1, Level);
         DownLevel(1, Level-1)
      END;
   Swap(1, 2)
END; { HSort }
```

Manager of the memory and of the pointers

```
PROCEDURE GetMemory;
VAR i     : BYTE;
    Ptr   : POINTER;
    avail : LONGINT;
BEGIN
   Mark(_UserHeap);
   IF Get+Let<MaxAvail
      THEN _UserMemory:=Get
      ELSE _UserMemory:=MaxAvail-Let;
   avail:=_UserMemory;
   i:=1;
   WHILE (i<=9) AND ($FFFF<avail) DO
      BEGIN
         GetMem(Ptr, $FFFF);
         Dec(avail, $FFFF);
         Inc(i)
      END;
   GetMem(Ptr, avail)
END; { GetMemory }
```

## Incrementing an address.

```
FUNCTION AddAddr;
TYPE Prec = RECORD
               off, Seg : WORD
            END;
VAR Add : Prec ABSOLUTE p;
BEGIN
   Inc(Add.off, s MOD $10);
   Inc(Add.Seg, s DIV $10);
   Inc(Add.Seg, Add.off DIV $10);
   Add.off:=Add.off MOD $10;
   AddAddr:=p
END; { AddAddr }
```

## Syntaxic reading of a file

Reads one word in a file <f> (in principle a text declared as FileOfChar). The reading is sequential. The words separator (ASCII 0..32 and punctuation) are given in <token>. The comments contained between <begC> and <endC> are ignored. <begC> and <endC> must be different, e.g.: '{' and '}'. All the characters, including the separators and the delimitors of the comments, intercalated between two <Str> (e.g.: '"') are restituted as a single article, including the delimitors <Str>, by RdItem. If the string is larger than 255 characters, it is truncated. It is thus possible to know if it is a word of a string by comparing the first character with <Str>. If the end of the file is reached, RdItem returns a null string. <nline> returns the number of the line of the file where the word was found. Must be initialized to 1 before the first call to that function.

```
FUNCTION RdItem(VAR f                : FileOfChar;
                    token            : AnyChar;
                    begC, endC, Str  : CHAR;
                VAR nline            : WORD;
                VAR pos              : LONGINT) : STRING;

VAR c : CHAR;

   PROCEDURE RdComment; FORWARD;

   FUNCTION RdChar1 : BOOLEAN;
   BEGIN
      IF Eof(f) THEN
         RdChar1:=FALSE
      ELSE BEGIN
         Read(f, c);
         Inc(pos);
         IF c=#13 THEN
            Inc(nline)
         ELSE IF c=begC THEN
            RdComment;
         RdChar1:=TRUE
      END
   END; { RdChar1 }

   PROCEDURE RdComment;
   BEGIN
      REPEAT UNTIL (NOT RdChar1) OR (c=endC);
      IF c=endC THEN c:=#0
   END; { RdComment }

   FUNCTION RdChar2 : BOOLEAN;
   BEGIN
      IF Eof(f) THEN
         RdChar2:=FALSE
      ELSE BEGIN
         Read(f, c);
         Inc(pos);
         IF c=#13 THEN Inc(nline);
         RdChar2:=TRUE
      END
   END; { RdChar2 }

VAR item : STRING;
BEGIN
   REPEAT UNTIL (NOT RdChar1) OR (NOT (c IN token));
   IF Eof(f) THEN
      RdItem:=''
   ELSE BEGIN
      item:='';
      IF c=Str THEN BEGIN
         REPEAT item:=item+c UNTIL (NOT RdChar2) OR (c=Str);
         IF c=Str THEN item:=item+c
      END
      ELSE BEGIN
         REPEAT item:=item+c UNTIL (NOT RdChar1) OR (c IN token);
         IF c=#13 THEN BEGIN
            Dec(nline);
            Dec(pos);
            Seek(f, FilePos(f)-1)
         END
      END;
      RdItem:=item
   END
END; { RdItem }
```

## Untyped files with buffer

### Opens a file for reading.

```
PROCEDURE ROpenUF(VAR f : UntypedFile;
                      p : PathStr);
BEGIN
   Assign(f.f, p);
   {$I-} Reset(f.f, 1); {$I+}
   _TError:=IOResult;
   IF _TError<>0 THEN EXIT;
   f.fsize:=FileSize(f.f);
   IF BufferSize<f.fsize
      THEN IF MaxAvail<BufferSize
         THEN BEGIN
            {$I-} Close(f.f); {$I+}
            _TError:=8;
            EXIT
         END
         ELSE f.bsize:=BufferSize
      ELSE f.bsize:=f.fsize;
   GetMem(f.buffer, f.bsize);
   {$I-} BlockRead(f.f, f.buffer^, f.bsize, f.blen); {$I+}
   _TError:=IOResult;
   IF _TError<>0 THEN EXIT;
   f.Ptr:=0
END; { ROpenUF }
```

### Opens a file for writing.

```
PROCEDURE WOpenUF(VAR f : UntypedFile;
                      p : PathStr);
```

```
BEGIN
   Assign(f.f, p);
   {$I-} Rewrite(f.f, 1); {$I+}
   _TError:=IOResult;
   IF _TError<>0 THEN EXIT;
   IF MaxAvail<BufferSize
      THEN BEGIN
         {$I-} Close(f.f); {$I+}
         _TError:=8
      END
      ELSE BEGIN
         f.bsize:=BufferSize;
         GetMem(f.buffer, f.bsize);
         f.fsize:=0;
         f.blen:=0;
         f.Ptr:=0
      END
END; { ROpenUF }
```

Closes a file in reading and the buffer

```
PROCEDURE RCloseUF(VAR f : UntypedFile);
BEGIN
   {$I-} Close(f.f); {$I+}
   _TError:=IOResult;
   FreeMem(f.buffer, f.bsize)
END; { RCloseUF }
```

Closes a file in writing and the buffer is put on the disk

```
PROCEDURE WCloseUF(VAR f : UntypedFile);
BEGIN
   IF 0<f.blen
      THEN BEGIN
         {$I-} BlockWrite(f.f, f.buffer^, f.blen); {$I+}
         _TError:=IOResult;
         IF _TError<>0 THEN EXIT;
         Inc(f.fsize, f.blen)
      END;
   {$I-} Close(f.f); {$I+}
   _TError:=IOResult;
   FreeMem(f.buffer, f.bsize)
END; { WCloseUF }
```

Reads in <f> <count> bytes and transfers them in <b>. The number which is really trasnferred is returned into <result>. Equivalent to BlockRead.

```
PROCEDURE ReadUF(VAR f      : UntypedFile;
                 VAR b      ;
                     count  : WORD;
                 VAR result : WORD);
VAR p : POINTER;
BEGIN
   p:=@b;
   result:=0;
   REPEAT
      IF count<=f.blen-f.Ptr
         THEN BEGIN
            Move(f.buffer^[f.Ptr], p^, count);
            Inc(f.Ptr, count);
            Inc(result, count);
            count:=0
         END
         ELSE BEGIN
            Move(f.buffer^[f.Ptr], p^, f.blen-f.Ptr);
            Inc(result, f.blen-f.Ptr);
            Dec(count, f.blen-f.Ptr);
            p:=AddAddr(p, f.blen-f.Ptr);
            f.Ptr:=0;
            {$I-} BlockRead(f.f, f.buffer^, f.bsize, f.blen); {$I+}
            _TError:=IOResult;
            IF _TError<>0 THEN EXIT
         END
   UNTIL (count=0) OR (f.blen=0)
END; { ReadUF }
```

Writes in <f> <count> bytes read <b>. The number really trasferred is retrurned into <result>. Equivalent to BlockWrite.

```
PROCEDURE WriteUF(VAR f       : UntypedFile;
                  VAR b       ;
                      count   : WORD;
                  VAR result  : WORD);
VAR p : POINTER;
BEGIN
   p:=@b;
   result:=0;
   REPEAT
      IF count<=f.bsize-f.Ptr
         THEN BEGIN
            Move(p^, f.buffer^[f.Ptr], count);
            Inc(f.Ptr, count);
            Inc(f.blen, count);
            Inc(result, count);
            count:=0
         END
         ELSE BEGIN
            Move(p^, f.buffer^[f.Ptr], f.bsize-f.Ptr);
            f.blen:=f.bsize;
            Inc(result, f.bsize-f.Ptr);
            Dec(count, f.bsize-f.Ptr);
            p:=AddAddr(p, f.bsize-f.Ptr);
            {$I-} BlockWrite(f.f, f.buffer^, f.bsize, f.blen); {$I+}
            _TError:=IOResult;
            IF _TError<>0 THEN EXIT;
            f.Ptr:=0;
            Inc(f.fsize, f.blen);
            f.blen:=0
         END
   UNTIL count=0
END; { WriteUF }
```

Says if the end of the file being read is reached in the buffer

```
FUNCTION EoUF(f : UntypedFile) : BOOLEAN;
BEGIN
   EoUF:=f.blen=f.Ptr
END; { EoUF }

END.
```

## *Library MATH*

Function : Library of the mathematical functions

```
UNIT Math;

INTERFACE

CONST
   MAXINT  =       32767;
   MININT  =      -32768;
   MAXWORD =       65535;
   MAXLONG =  2147483647;
   MINLONG = -2147483647;

   TWOPI   = 6.283185308;
   HALFPI  = 1.570796327;

FUNCTION InDegMode : BOOLEAN;
FUNCTION InRadMode : BOOLEAN;
PROCEDURE ToDegMode;
PROCEDURE ToRadMode;
FUNCTION Sin(x : REAL) : REAL;
FUNCTION Cos(x : REAL) : REAL;
FUNCTION Tan(x : REAL) : REAL;
FUNCTION ArcSin(x : REAL) : REAL;
FUNCTION ArcCos(x : REAL) : REAL;
FUNCTION ArcTan(x : REAL) : REAL;
FUNCTION Slope(x, y : REAL) : REAL;
FUNCTION Rad(x : REAL) : REAL;
FUNCTION Deg(x : REAL) : REAL;
FUNCTION Sign(x : REAL) : INTEGER;
FUNCTION Float(lng : LONGINT) : REAL;
FUNCTION IMin(x, y : INTEGER) : INTEGER;
FUNCTION IMax(x, y : INTEGER) : INTEGER;
FUNCTION RMin(x, y : REAL) : REAL;
FUNCTION RMax(x, y : REAL) : REAL;
FUNCTION IStr(x : LONGINT;
              f : BYTE) : STRING;
FUNCTION RStr(x     : REAL;
              f, d : BYTE) : STRING;
PROCEDURE IntSwap(VAR x, y : INTEGER);
PROCEDURE LngIntSwap(VAR x, y : LONGINT);
PROCEDURE RealSwap(VAR x, y : REAL);

IMPLEMENTATION

CONST
   _DegMode : BOOLEAN = FALSE;
   _DegRad  = 1.745329252E-2;   { PI/180 }
```

True if the mode degree is active.

```
FUNCTION InDegMode : BOOLEAN;
   BEGIN
      InDegMode:=_DegMode
   END; { InDegMode }
```

True if the radians mode is active

```
FUNCTION InRadMode : BOOLEAN;
   BEGIN
      InRadMode:=NOT _DegMode
   END; { InRadMode }
```

Stirs up the mode degree

```
PROCEDURE ToDegMode;
   BEGIN
      _DegMode:=TRUE
   END; { _degMode }
```

Stirs up the mode radians.

```
PROCEDURE ToRadMode;
   BEGIN
      _DegMode:=FALSE
   END; { ToRadMode }
```

Trigonometrical functions

```
FUNCTION Sin(x : REAL) : REAL;
   BEGIN
      IF _DegMode THEN
         Sin:=System.Sin(_DegRad*x)
      ELSE
         Sin:=System.Sin(x)
   END; { Sin }

FUNCTION Cos(x : REAL) : REAL;
   BEGIN
      IF _DegMode THEN
         Cos:=System.Cos(_DegRad*x)
      ELSE
         Cos:=System.Cos(x)
   END; { Cos }

FUNCTION Tan(x : REAL) : REAL;
   BEGIN
      IF _DegMode THEN
         x:=_degRad*x;
      Tan:=System.Sin(x)/System.Cos(x)
   END; { Tan }
```

```
FUNCTION ArcSin(x : REAL) : REAL;
   BEGIN
      x:=System.ArcTan(x/Sqrt(1-Sqr(x)));
      IF _DegMode THEN
         ArcSin:=x/_DegRad
      ELSE
         ArcSin:=x
   END; { ArcSin }

FUNCTION ArcCos(x : REAL) : REAL;
   BEGIN
      IF _DegMode THEN
         ArcCos:=90-ArcSin(x)
      ELSE
         ArcCos:=HALFPI-ArcSin(x)
   END; { ArcCos }

FUNCTION ArcTan(x : REAL) : REAL;
   BEGIN
      IF _DegMode THEN
         ArcTan:=System.ArcTan(x)/_DegRad
      ELSE
         ArcTan:=System.ArcTan(x)
   END; { ArcTan }

FUNCTION Slope(x, y : REAL) : REAL;
   VAR
      r : REAL;
   BEGIN
      r:=Sqrt(Sqr(x)+Sqr(y));
      IF r<1E-5 THEN Exit; { pente inconnue }
      r:=2*ArcTan(y/(r+Abs(x)));
      IF x>0 THEN
         Slope:=r
      ELSE
         IF _DegMode THEN
            Slope:=180-r
         ELSE
            Slope:=PI-r
   END; { Slope }

FUNCTION Rad(x : REAL) : REAL;
   BEGIN
      Rad:=x/_DegRad
   END; { Rad }

FUNCTION Deg(x : REAL) : REAL;
   BEGIN
      Deg:=X*_DegRad
   END; { Deg }
```

## Mathematical functions

```
FUNCTION Sign(x : REAL) : INTEGER;
   BEGIN
      IF x<0 THEN
         Sign:=-1
      ELSE
         IF x=0 THEN
            Sign:=0
         ELSE
            Sign:=1
   END; { Sign }

FUNCTION Float(lng : LONGINT) : REAL;
   BEGIN
      Float:=lng
   END; { Float }

FUNCTION IMin(x, y : INTEGER) : INTEGER;
   BEGIN
      IF x<y
         THEN IMin:=x
         ELSE IMin:=y
   END; { IMin }

FUNCTION IMax(x, y : INTEGER) : INTEGER;
   BEGIN
      IF x<y
         THEN IMax:=y
         ELSE IMax:=x
   END; { IMax }

FUNCTION RMin(x, y : REAL) : REAL;
   BEGIN
      IF x<y
         THEN RMin:=x
         ELSE RMin:=y
   END; { RMin }

FUNCTION RMax(x, y : REAL) : REAL;
   BEGIN
      IF x<y
         THEN RMax:=y
         ELSE RMax:=x
   END; { RMax }

FUNCTION IStr(x : LONGINT;
              f : BYTE) : STRING;
   VAR
      s : STRING;
   BEGIN
      Str(x:f, s);
      IStr:=s
   END; { IStr }

FUNCTION RStr(x    : REAL;
              f, d : BYTE) : STRING;
   VAR
      s : STRING;
   BEGIN
      Str(x:f:d, s);
      RStr:=s
   END; { RStr }

PROCEDURE IntSwap(VAR x, y : INTEGER);
   VAR
      i : INTEGER;
   BEGIN
      i:=x;
      x:=y;
      y:=i
   END; { IntSwap }

PROCEDURE LngIntSwap(VAR x, y : LONGINT);
   VAR
      l : LONGINT;
   BEGIN
      l:=x;
      x:=y;
      y:=l
   END; { LngIntSwap }

PROCEDURE RealSwap(VAR x, y : REAL);
   VAR
      r : REAL;
   BEGIN
      r:=x;
      x:=y;
      y:=r
   END; { RealSwap }

END.
```

## *Library ERROR*

Function : Library of the error messages

```
UNIT Error;

INTERFACE

USES Tools, Math;

TYPE
   UserErrorFunc = FUNCTION (Error : WORD) : Str74;
```

Procedure ending the Program.

```
   UserExitProc   = PROCEDURE (Error : INTEGER);

VAR
```

Pointer on the function restituting the messages of personal errors.

```
   _UserError : UserErrorFunc;

FUNCTION ErrorStr(Error : WORD) : Str74;
PROCEDURE InitFatalError(UserExit : UserExitProc);
PROCEDURE NoUserExit(Error : INTEGER);

IMPLEMENTATION

VAR
   _ExitSave : POINTER;
   _UserExit : UserExitProc;
```

Returns the corresponding error message.

```
FUNCTION ErrorStr(Error : WORD) : Str74;
VAR msg : Str74;
    n   : Str5;
BEGIN
   CASE Error OF
        0 : msg:='Normal termination';
        2 : msg:='File not found';
        3 : msg:='Path not found';
        4 : msg:='Too many open files';
        5 : msg:='File access denied';
        6 : msg:='Invalid file handle';
        7 : msg:='Memory control blocks destroyed';
        8 : msg:='Insufficient memory';
        9 : msg:='Invalid memory block address';
       10 : msg:='Invalid environment';
       11 : msg:='Invalid format';
       12 : msg:='Invalid file access code';
       13 : msg:='Invalid data';
       14 : msg:='RESERVED';
       15 : msg:='Invalid drive';
       16 : msg:='Cannot remove current directory';
       17 : msg:='Cannot rename accross drives';
       18 : msg:='File not found';                  {DOS: 'No more files'}

      100 : msg:='Disk read error';
      101 : msg:='Disk write error';
      102 : msg:='File not assigned';
      103 : msg:='File not open';
      104 : msg:='File not open FOR input';
      105 : msg:='File not open FOR output';
      106 : msg:='Invalid numeric format';

      150 : msg:='Disk is write-protected';
      151 : msg:='Unknown unit';
      152 : msg:='Drive not ready';
      153 : msg:='Unknown command';
      154 : msg:='CRC error in data';
      155 : msg:='Bad drive request structure length';
      156 : msg:='Disk seek error';
      157 : msg:='Unknown media TYPE';
      158 : msg:='Sector not found';
      159 : msg:='Printer out OF paper';
      160 : msg:='Device write fault';
      161 : msg:='Device read fault';
      162 : msg:='Hardware failure';

      200 : msg:='Division by zero';
      201 : msg:='Range check error';
      202 : msg:='Stack overflow error';
      203 : msg:='Heap overflow error';
      204 : msg:='Invalid POINTER operation';
      205 : msg:='Floating point overflow';
      206 : msg:='Floating point underflow';
      207 : msg:='Invalid floating point operation';
      208 : msg:='Overlay manager not installed';
      209 : msg:='Overlay file read error';

      210 : msg:='File already exist';
      240 : msg:='Disk full';
```

Internal errors OF JS_LIB

```
      246 : msg:='Help file not found';
      247 : msg:='Plotter device not ready';
      248 : msg:='Invalid graphic hardware';
      249 : msg:='Mouse not found';
      250 : msg:='Too many options in the menu window -> FATAL ERROR';
      251 : msg:='User break';
      252 : msg:='Search STRING not found';
      253 : msg:='Printer device not ready';
      254 : msg:='Insufficient memory TO load the whole file';
      255 : msg:='User break'
      ELSE BEGIN
          msg:=_UserError(Error);
          IF Length(msg)=0 THEN msg:='Unexpected error '+IStr(Error, 0)
      END
   END;
   Str(Error, n);
   IF 1000<=Error
      THEN ErrorStr:=Concat('WARNING ', n, ': ', msg)
      ELSE IF (Error=0) OR (Error=255)
         THEN ErrorStr:=msg
         ELSE ErrorStr:=Concat('ERROR ', n, ': ', msg)
END; { ErrorStr }

{$F+} PROCEDURE FatalError; {$F-}
BEGIN
   ErrorAddr:=NIL;
   ExitProc:=_ExitSave;
   _UserExit(ExitCode)
END; { FatalError }
```

Elimination of "runtime error" and definition of the personal procedure “end of program”

```
PROCEDURE InitFatalError(UserExit : UserExitProc);
BEGIN
   _ExitSave:=ExitProc;
   ExitProc:=@FatalError;
   _UserExit:=UserExit
END; { InitFatalError }

{$F+} FUNCTION NoUserError(Error : WORD) : Str74; {$F-}
BEGIN
   NoUserError:=''
END; { NoUserError }
```

Procedure for InitFatalError.

```
{$F+} PROCEDURE NoUserExit(Error : INTEGER); {$F-}
BEGIN
END; { NoUserExit }

BEGIN
   _UserError:=NoUserError
END.
```

## ***Library VIDEO***

Function : screen tool

UNIT VIDEO;

```
INTERFACE

USES Dos;
```

```
TYPE
```

Variables allowing to keep the ad hoc colors for monochrome of color sreen

```
   ScreenColor = ARRAY[BOOLEAN] OF BYTE;
   ScreenChar  = RECORD
                    C : CHAR;
                    A : BYTE
                 END;
   ScreenLine  = ARRAY[1..80] OF ScreenChar;
   ScreenPage  = ARRAY[1..25] OF ScreenLine;
   ScreenPtr   = ^ScreenPage;

   CursorType  = (On, Off, Expand);

VAR
```

monochrom screen =FALSE / couleur=TRUE

```
   _C : BOOLEAN;
```

Pointer on the screen buffer

```
   _S : ScreenPtr;

PROCEDURE SaveScreen(screen : ScreenPtr);
PROCEDURE RestoreScreen(screen : ScreenPtr);
PROCEDURE SetCsr( Size : CursorType );
FUNCTION  IsHercules : Boolean;

IMPLEMENTATION

VAR
   _R : BOOLEAN;

PROCEDURE GetVideo;
VAR Regs : Registers;
    Cmd  : ComStr;
    i, p : BYTE;
BEGIN
   Intr($11,Regs);
   _R:=Lo(Regs.AX) AND $30<>$30;
   _S:=Ptr($B000+Ord(_R)*$800,$0);
   i:=1;
   REPEAT
      Cmd:=ParamStr(i);
      Inc(i);
      FOR p:=1 TO Length(Cmd) DO Cmd[p]:=UpCase(Cmd[p])
   UNTIL (0=Length(Cmd)) OR (0<Pos('/BW', Cmd));
   IF _R
      THEN _C:=Pos('/BW', Cmd)=0
      ELSE _C:=_R
END; { GetVideo }

PROCEDURE SaveScreen(screen : ScreenPtr);
BEGIN
   Move(_S^, screen^, SizeOf(ScreenPage))
END; { SaveScreen }

PROCEDURE RestoreScreen(screen : ScreenPtr);
BEGIN
   Move(screen^, _S^, SizeOf(ScreenPage))
END; { RestoreScreen }

PROCEDURE SetCsr( Size : CursorType );
CONST CsrSize : ARRAY[BOOLEAN, CursorType] OF INTEGER
              = (($0C0D,$0E00,$000D),($0607,$2000,$0007));
VAR Regs : Registers;
BEGIN
   Regs.AX:=$0100;
   Regs.CX:=CsrSize[_R, Size];
   Intr($10,Regs)
END; { SetCsr }

FUNCTION IsHercules : Boolean;
BEGIN
   IsHercules:=NOT _R
END; { IsHercules }
```

```
BEGIN
   GetVideo
END.
```

## *Library MOUSE*

Function :   Library for the mouse management

```
UNIT Mouse;

INTERFACE

USES Dos;

TYPE
```

The screenMask is first ANDed into the display, THEN the cursorMask is XORed into the display. The hot spot coordinates are relative TO the upper-left corner OF the cursor image, AND define where the cursor actually 'points TO'.

```
   GraphicCursor = RECORD
                      screenMask, cursorMask: ARRAY [0..15] OF WORD;
                      hotX, hotY             : INTEGER
                   END;

CONST
   Arrow : GraphicCursor
     = (screenMask : ($1FFF,$0FFF,$07FF,$03FF,$01FF,$00FF,$007F,$003F,
                      $003F,$003F,$00FF,$00FF,$107F,$F07F,$F87F,$F87F);
        cursorMask : ($0000,$4000,$6000,$7000,$7800,$7C00,$7E00,$7F00,
                      $7F80,$7C00,$6C00,$4600,$0600,$0300,$0300,$0000);
        hotX       : 0;
        hotY       : 0);
   Arrow_1 : GraphicCursor
     = (screenMask : ($1FFF,$0FFF,$03FF,$80FF,$803F,$C00F,$C003,$E001,
                      $E000,$F00F,$F007,$F803,$F841,$FC60,$FEF1,$FFFB);
        cursorMask : ($0000,$6000,$7800,$3E00,$3F80,$1FE0,$1FF8,$0FFC,
                      $0FC0,$07E0,$0770,$0338,$031C,$010E,$0004,$0000);
        hotX       : 0;
        hotY       : 0);
   SquarePoint : GraphicCursor
     = (screenMask : ($0000,$0000,$0000,$1FF8,$1FF8,$1FF8,$1C38,$1C38,
                      $1C38,$1C38,$1FF8,$1FF8,$1FF8,$0000,$0000,$0000);
        cursorMask : ($0000,$7FFE,$4002,$4002,$4002,$4002,$4002,$4182,
                      $4182,$4002,$4002,$4002,$4002,$4002,$7FFE,$0000);
        hotX       : 7;
        hotY       : 7);
   Diskette : GraphicCursor
     = (screenMask : ($0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000,
                      $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000);
        cursorMask : ($0000,$7FFE,$4002,$4002,$4002,$4002,$4182,$43C2,
                      $4182,$4002,$4002,$4182,$4182,$4182,$7FFE,$0000);
        hotX       : 7;
        hotY       : 7);
   SquareCross : GraphicCursor
     = (screenMask : ($0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000,
                      $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000);
        cursorMask : ($0000,$7FFE,$6006,$500A,$4812,$4422,$4242,$4182,
                      $4182,$4242,$4422,$4812,$500A,$6006,$7FFE,$0000);
        hotX       : 7;
        hotY       : 7);
   SquareCrossGround : GraphicCursor
     = (screenMask : ($0000,$0000,$0000,$07E0,$03C0,$1188,$1818,$1C38,
                      $1C38,$1818,$1188,$03C0,$07E0,$0000,$0000,$0000);
        cursorMask : ($0000,$7FFE,$6006,$500A,$4812,$4422,$4242,$4182,
                      $4182,$4242,$4422,$4812,$500A,$6006,$7FFE,$0000);
        hotX       : 7;
        hotY       : 7);
   LittleSquare : GraphicCursor
     = (screenMask : ($01FF,$01FF,$01FF,$01FF,$01FF,$01FF,$01FF,$FFFF,
                      $FFFF,$FFFF,$FFFF,$FFFF,$FFFF,$FFFF,$FFFF,$FFFF);
        cursorMask : ($0000,$7C00,$4400,$5400,$4400,$7C00,$0000,$0000,
                      $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000);
```

```
         hotX       : 3;
         hotY       : 3);

    Hourglass : GraphicCursor
      = (screenMask : ($0000,$0000,$0000,$C003,$C003,$E007,$F00F,$F00F,
                       $F00F,$F00F,$E007,$C003,$C003,$0000,$0000,$0000);
         cursorMask : ($0000,$7FFE,$1008,$1C38,$0E70,$07E0,$07E0,$03C0,
                       $03C0,$0420,$05A0,$0990,$13C8,$17E8,$7FFE,$0000);
         hotX       : 3;
         hotY       : 3);
```

## Standard functions

```
FUNCTION DriverInstalled : BOOLEAN;
PROCEDURE FlagReset(VAR mouseStatus, numberOfButtons : WORD);
PROCEDURE ShowMouse;
PROCEDURE HideMouse;
PROCEDURE GetPosBut(VAR buttonStatus, horizontal, vertical : INTEGER);
PROCEDURE SetCursorPos(horizontal, vertical : INTEGER);
PROCEDURE GetButPres(button           : INTEGER;
                 VAR buttonStatus,
                     buttonPressCount,
                     horizontal,
                     vertical         : INTEGER);
PROCEDURE GetButRel(button               : INTEGER;
                VAR buttonStatus,
                    buttonReleaseCount,
                    horizontal ,vertical : INTEGER);
PROCEDURE SetHorizontalLimits(minPos, maxPos : INTEGER);
PROCEDURE SetVerticalLimits(minPos, maxPos : INTEGER);
PROCEDURE SetGraphicCursor(VAR cursor : GraphicCursor);
PROCEDURE SetTextCursor(selectedCursor,
                        screenMaskORscanStart,
                        cursorMaskORscanStop  : INTEGER);
PROCEDURE ReadMotionCounters(VAR horizontal, vertical : INTEGER);
PROCEDURE SetMickeysPerPixel(horPix, verPix : INTEGER);
PROCEDURE ConditionalOff(left, top,
                         right, bottom : INTEGER);
PROCEDURE SetSpeedThreshold(threshold : INTEGER);
```

## Non standard functions

```
FUNCTION InitMouse : BOOLEAN;
PROCEDURE SetMouseField(xmin, ymin, xmax, ymax : INTEGER);
PROCEDURE StillMouse;

IMPLEMENTATION
```

## Standard functions

```
FUNCTION DriverInstalled : BOOLEAN;
CONST iret = 207;
VAR driverOff, driverSeg : WORD;
BEGIN
   driverOff:=MemW[0000:0204];
   driverSeg:=MemW[0000:0206];
   IF (driverSeg=0) OR (driverOff=0)
      THEN DriverInstalled:=FALSE
      ELSE DriverInstalled:=Mem[driverSeg:driverOff]<>iret
END; { DriverInstalled }
```

Microsoft Mouse Driver System Call 0
Input : AX = 0 System Call 0
Output : AX : mouse status
0 (FALSE) : mouse hardware AND software NOT installed
-1 (TRUE) : mouse hardware AND software installed
BX : number OF mouse buttons

```
PROCEDURE FlagReset(VAR mouseStatus, numberOfButtons : WORD);
VAR regSet : Registers;
BEGIN
   regSet.AX:=0;
   Intr($33, regSet);
   mouseStatus:=regSet.AX;
   numberOfButtons:=regSet.BX
END; { FlagReset }
```

Microsoft Mouse Driver System Call 1

Input : AX = 1 System Call 1

```
PROCEDURE ShowMouse;
VAR regSet : Registers;
BEGIN
   regSet.AX:=1;
   Intr($33, regSet)
END; { ShowMouse }
```

Microsoft Mouse Driver System Call 2

Input : AX = 2 System Call 2

```
PROCEDURE HideMouse;
VAR regSet : Registers;
BEGIN
   regSet.AX:=2;
   Intr ($33, regSet)
END; { HideMouse }
```

```
Microsoft Mouse Driver System Call 3
Input :         AX = 3 System Call 3
Output :        BX :    mouse button status
           CX : horizontal cursor position
           DX : vertical cursor position
```

```
PROCEDURE GetPosBut(VAR buttonStatus, horizontal, vertical : INTEGER);
VAR regSet : Registers;
BEGIN
   regSet.AX:=3;
   Intr($33, regSet);
   horizontal:=regSet.CX;
   vertical:=regSet.DX;
   buttonStatus:=regSet.BX
END; { GetPosBut }
```

```
Microsoft Mouse Driver System Call 4
Input :         AX = 4 System Call 4
           CX : horizontal mouse cursor position
           DX : vertical mouse cursor position
```

```
PROCEDURE SetCursorPos(horizontal, vertical : INTEGER);
VAR regSet : Registers;
BEGIN
   regSet.AX:=4;
   regSet.CX:=horizontal;
   regSet.DX:=vertical;
   Intr($33, regSet)
END; { SetCursorPos }
```

```
Microsoft Mouse Driver System Call 5
Input :         AX = 5 System Call 5
           BX : button
Output :        AX :    current button status
           BX : count OF button presses since last call TO this
           FUNCTION
           CX : horizontal cursor position at last press
           DX : vertical cursor position at last press
```

```
PROCEDURE GetButPres(button           : INTEGER;
                 VAR buttonStatus,
                     buttonPressCount,
                     horizontal,
                     vertical         : INTEGER);
VAR regSet : Registers;
BEGIN
   regSet.AX:=5;
   regSet.BX:=button;
   Intr($33, regSet);
   buttonStatus:=regSet.AX;
   buttonPressCount:=regSet.BX;
   horizontal:=regSet.CX;
   vertical:=regSet.DX
END; { GetButPres }
```

Microsoft Mouse Driver System Call 6

Input : AX = 6 System Call 6
BX : button
Output : AX : current button status
BX : count OF button releases since last call TO this FUNCTION
CX : horizontal cursor position at last press
DX : vertical cursor position at last press

```
PROCEDURE GetButRel(button                   : INTEGER;
                VAR buttonStatus,
                    buttonReleaseCount,
                    horizontal ,vertical : INTEGER);
VAR regSet : Registers;
BEGIN
  regSet.AX:=6;
  regSet.BX:=button;
  Intr($33, regSet);
  buttonStatus:=regSet.AX;
  buttonReleaseCount:=regSet.BX;
  horizontal:=regSet.CX;
  vertical:=regSet.DX
END; { GetButRel }
```

Microsoft Mouse Driver System Call 7
Input : AX = 7 System Call 7
CX : minimum horizontal position
DX : maximum horizontal position

```
PROCEDURE SetHorizontalLimits(minPos, maxPos : INTEGER);
VAR regSet : Registers;
BEGIN
   regSet.AX:=7;
   regSet.CX:=minPos;
   regSet.DX:=maxPos;
   Intr($33, regSet)
END; { SetHorizontalLimits }
```

Microsoft Mouse Driver System Call 8
Input : AX = 8 System Call 8
CX : minimum vertical position
DX : maximum vertical position

```
PROCEDURE SetVerticalLimits(minPos, maxPos : INTEGER);
VAR regSet : Registers;
BEGIN
   regSet.AX:=8;
   regSet.CX:=minPos;
   regSet.DX:=maxPos;
   Intr($33, regSet)
END; { SetVerticalLimits }
```

Microsoft Mouse Driver System Call 9
Input : AX = 9 System Call 9
BX : cursor hot spot (horizontal)
CX : cursor hot spot (vertical)
ES:DX : POINTER TO screen AND cursor masks

```
PROCEDURE SetGraphicCursor(VAR cursor : GraphicCursor);
VAR regSet : Registers;
BEGIN
   regSet.AX:=9;
   regSet.DX:=Ofs(cursor);
   regSet.ES:=Seg(cursor);
   regSet.BX:=cursor.hotX;
   regSet.CX:=cursor.hotY;
   Intr($33, regSet)
END; { SetGraphicCursor }
```

Microsoft Mouse Driver System Call 10
Input : AX = 10 System Call 10
BX : cursor select
0 : Software TEXT cursor
1 : Hardware TEXT cursor
CX : screen mask value OR scan Line start
DX : cursor mask value OR scan Line stop
FOR the software TEXT cursor, the second two parameters specify the screen AND cursor masks. The screen mask is first ANDed into the display, THEN the cursor mask is XORed into the display. FOR the hardware TEXT cursor, the second two parameters contain the Line numbers OF the first AND last scan Line IN the cursor TO be shown on the screen.

```
PROCEDURE SetTextCursor(selectedCursor,
                        screenMaskORscanStart,
                        cursorMaskORscanStop  : INTEGER);
VAR regSet : Registers;
BEGIN
   regSet.AX:=10;
   regSet.BX:=selectedCursor;
   regSet.CX:=screenMaskORscanStart;
   regSet.DX:=cursorMaskORscanStop;
   Intr($33, regSet)
END; { SetTextCursor }
```

Microsoft Mouse Driver System Call 11
Input : AX = 11 System Call 11
CX : horizontal count
DX : vertical count

```
PROCEDURE ReadMotionCounters(VAR horizontal, vertical : INTEGER);
VAR regSet : Registers;
BEGIN
   regSet.AX:=11;
   regSet.CX:=horizontal;
   regSet.DX:=vertical;
   Intr($33, regSet)
END; { ReadMotionCounters }
```

Microsoft Mouse Driver System Call 15
Input : AX = 15 System Call 15
CX : horizontal mickey/pixel ratio
DX : vertical mickey/pixel ratio

```
PROCEDURE SetMickeysPerPixel(horPix, verPix : INTEGER);
VAR regSet : Registers;
BEGIN
   regSet.AX:=15;
   regSet.CX:=horPix;
   regSet.DX:=verPix;
   Intr($33, regSet)
END; { SetMickeysPerPixel }
```

Microsoft Mouse Driver System Call 16
Input : AX = 16 System Call 16
CX : left
DX : top
SI : right
DI : bottom

```
PROCEDURE ConditionalOff(left, top,
                         right, bottom : INTEGER);
VAR regSet : Registers;
BEGIN
  regSet.AX:=16;
  regSet.CX:=left;
  regSet.DX:=top;
  regSet.SI:=right;
  regSet.DI:=bottom;
  Intr($33, regSet)
END; { ConditionalOff }
```

Microsoft Mouse Driver System Call 19
Input : AX = 19 System Call 19
DX : threshold in mickeys/second

```
PROCEDURE SetSpeedThreshold(threshold : INTEGER);
VAR regSet : Registers;
BEGIN
   regSet.AX:=19;
   regSet.DX:=threshold;
   Intr($33, regSet)
END; { SetSpeedThreshold }
```

### Non standard functions

```
FUNCTION InitMouse : BOOLEAN;
VAR mouseStatus, numberOfButtons : WORD;
BEGIN
   IF DriverInstalled
      THEN BEGIN
         FlagReset(mouseStatus, numberOfButtons);
         InitMouse:=mouseStatus<>0
      END
      ELSE InitMouse:=FALSE
END; { InitMouse }

PROCEDURE SetMouseField(xmin, ymin, xmax, ymax : INTEGER);
BEGIN
   SetHorizontalLimits(xmin, xmax);
   SetVerticalLimits(ymin, ymax)
END; { SetMouseField }

PROCEDURE StillMouse;
VAR button, x, y : INTEGER;
BEGIN
   REPEAT GetPosBut(button, x, y) UNTIL button=0
END; { StillMouse }

END.
```

## *Library ENJOY*

### Function : resting screen

```
unit Enjoy;

interface

PROCEDURE Mondrian(M : boolean);
PROCEDURE Puzzle(M : boolean);
PROCEDURE ScreenPeace(M : boolean);

implementation

uses CRT, Video, Mouse;

TYPE moveproc = PROCEDURE(i, x, y : BYTE);

PROCEDURE Mondrian;

TYPE
   ScreenType = ARRAY[0..81,0..26] OF BYTE;

VAR
   Scr         : ScreenType;
   s           : ScreenPage;
   C1, C2      : BYTE;
   mb, mx, my,
   nb, nx, ny  : INTEGER;
   OK          : boolean;

FUNCTION Must(X,Y : BYTE) : BYTE;
VAR C : BYTE;
BEGIN
   C:=0;
   IF 1<y THEN
   CASE Scr[X,Y-1] OF
      $B3,$B4,$B5,$B8,$BF,$C2,$C3,$C5,$C6,$D1,$D5,$D8,$DA : C:=C+$01;
      $B6,$B7,$B9,$BA,$BB,$C7,$C9,$CB,$CC,$CE,$D2,$D6,$D7 : C:=C+$02
   END;
   IF y<25 THEN
   CASE Scr[X,Y+1] OF
      $B3,$B4,$B5,$BE,$C0,$C1,$C3,$C5,$C6,$CF,$D4,$D8,$D9 : C:=C+$04;
      $B6,$B9,$BA,$BC,$BD,$C7,$C8,$CA,$CC,$CE,$D0,$D3,$D7 : C:=C+$08
   END;
   IF 1<x THEN
   CASE Scr[X-1,Y] OF
      $C0,$C1,$C2,$C3,$C4,$C5,$C7,$D0,$D2,$D3,$D6,$D7,$DA : C:=C+$10;
      $C6,$C8,$C9,$CA,$CB,$CC,$CD,$CE,$CF,$D1,$D4,$D5,$D8 : C:=C+$20
   END;
   IF x<80 THEN
   CASE Scr[X+1,Y] OF
      $B4,$B6,$B7,$BD,$BF,$C1,$C2,$C4,$C5,$D0,$D2,$D7,$D9 : C:=C+$40;
      $B5,$B8,$B9,$BB,$BC,$BE,$CA,$CB,$CD,$CE,$CF,$D1,$D8 : C:=C+$80
   END;
   Must:=C
END; { Must }

FUNCTION NoMust(X,Y : BYTE) : BYTE;
VAR C : BYTE;
BEGIN
   C:=0;
   IF Y=1 THEN C:=C+$03 ELSE
   IF Scr[X,Y-1] in [$20,$BC,$BD,$BE,$C0,$C1,$C4,$C8,$CA,$CD,$CF,$D0,$D3,$D4,$D9]
      THEN C:=C+$03;
   IF Y=25 THEN C:=C+$0C ELSE
   IF Scr[X,Y+1] in [$20,$B7,$B8,$BB,$BF,$C2,$C4,$C9,$CB,$CD,$D1,$D2,$D5,$D6,$DA]
      THEN C:=C+$0C;
   IF X=1 THEN C:=C+$30 ELSE
   IF Scr[X-1,Y] in [$20,$B3,$B4,$B5,$B6,$B7,$B8,$B9,$BA,$BB,$BC,$BD,$BE,$BF,$D9]
      THEN C:=C+$30;
   IF X=80 THEN C:=C+$C0 ELSE
   IF Scr[X+1,Y] in [$20,$B3,$BA,$C0,$C3,$C6,$C7,$C8,$C9,$CC,$D3,$D4,$D5,$D6,$DA]
      THEN C:=C+$C0;
   NoMust:=C
END; { NoMust }

PROCEDURE Exist(VAR C : BYTE);
BEGIN
   CASE C OF
      $00 : C:=$20;
      $05 : C:=$B3;   $15 : C:=$B4;   $25 : C:=$B5;   $1A : C:=$B6;
      $18 : C:=$B7;   $24 : C:=$B8;   $2A : C:=$B9;   $0A : C:=$BA;
      $28 : C:=$BB;   $22 : C:=$BC;   $12 : C:=$BD;   $21 : C:=$BE;
      $14 : C:=$BF;   $41 : C:=$C0;   $51 : C:=$C1;   $54 : C:=$C2;
      $45 : C:=$C3;   $50 : C:=$C4;   $55 : C:=$C5;   $85 : C:=$C6;
      $4A : C:=$C7;   $82 : C:=$C8;   $88 : C:=$C9;   $A2 : C:=$CA;
      $A8 : C:=$CB;   $8A : C:=$CC;   $A0 : C:=$CD;   $AA : C:=$CE;
      $A1 : C:=$CF;   $52 : C:=$D0;   $A4 : C:=$D1;   $58 : C:=$D2;
      $42 : C:=$D3;   $81 : C:=$D4;   $84 : C:=$D5;   $48 : C:=$D6;
      $5A : C:=$D7;   $A5 : C:=$D8;   $11 : C:=$D9;   $44 : C:=$DA
      ELSE  C:=$00;
   END
END; { Exist }

FUNCTION Can(X,Y : BYTE) : boolean;
VAR C   : ARRAY[1..4] OF BYTE;
    I   : BYTE;
    May : boolean;
BEGIN
   C[1]:=Must(X,Y-1);
   C[2]:=Must(X,Y+1);
   C[3]:=Must(X-1,Y);
   C[4]:=Must(X+1,Y);
   I:=0;
   REPEAT I:=I+1;
      May:=not((((C[I] and $01)=$01) and ((C[I] and $08)=$08)) or
               (((C[I] and $02)=$02) and ((C[I] and $04)=$04)) or
               (((C[I] and $10)=$10) and ((C[I] and $80)=$80)) or
               (((C[I] and $20)=$20) and ((C[I] and $40)=$40)))
   UNTIL (I=4) or not May;
   Can:=May
END; { Can }

FUNCTION DisplayChar(Alea,X,Y : BYTE) : boolean;
VAR I,C,N : BYTE;
BEGIN
   N:=0;
   C:=NoMust(X,Y);
   REPEAT
      N:=N+1;
      Scr[X,Y]:=Must(X,Y);
      FOR I:=0 TO 7 DO
         IF ((C and (1 shl I))=0) and ((Scr[X,Y] and (1 shl I))=0)
            THEN IF random(Alea)=0
               THEN Scr[X,Y]:=Scr[X,Y]+(1 shl I);
      Exist(Scr[X,Y]);
      IF M THEN BEGIN
         GetPosBut(nb, nx, ny);
         OK:=(mb<>nb) or (mx<>nx) or (my<>ny)
      END;
      IF keypressed THEN ok:=TRUE
   UNTIL ((Scr[X,Y]<>$00) and Can(X,Y)) or (N=$FF) or ok;
   _S^[Y,X].c:=chr(Scr[X,Y]);
   IF Scr[X,Y]=$20 THEN _S^[Y,X].a:=C1 ELSE _S^[Y,X].a:=C2;
   DisplayChar:=ok
END; { DisplayChar }

PROCEDURE Blackboard;
VAR Alea,I,X,Y,X1,X2,Y1,Y2 : BYTE;
    Key                    : char;
BEGIN
   Alea:=random(30)+2;
   IF _C THEN BEGIN
      I:=random(6)+9;
      C1:=((random(7)+1) shl 4) or I;
      C2:=((random(7)+1) shl 4) or I
   END
   ELSE BEGIN
      C1:=$07;
      C2:=$07
   END;
   fillchar(Scr,sizeof(ScreenType),0);
   CASE random(6) OF
      0 : BEGIN
             X1:=1;
             X2:=80;
             Y1:=1;
             Y2:=25;
             REPEAT
                FOR X:=X1 TO X2 DO
                   IF DisplayChar(Alea,X,Y1) THEN exit;
                IF X2=56 THEN exit;
                Y1:=Y1+1;
                FOR Y:=Y1 TO Y2 DO FOR I:=0 TO 1 DO
                   IF DisplayChar(Alea,X2-I,Y) THEN exit;
                X2:=X2-2;
                FOR X:=X2 downto X1 DO
                   IF DisplayChar(Alea,X,Y2) THEN exit;
                Y2:=Y2-1;
                FOR Y:=Y2 downto Y1 DO FOR I:=0 TO 1 DO
                   IF DisplayChar(Alea,X1+I,Y) THEN exit;
                X1:=X1+2
             UNTIL x=0
          END;
      1 : BEGIN
             X1:=25;
             X2:=56;
             Y1:=13;
             Y2:=13;
             REPEAT
                FOR X:=X1 TO X2 DO
                   IF DisplayChar(Alea,X,Y1) THEN exit;
                IF X2=80 THEN exit;
                X2:=X2+2;
                FOR Y:=Y1 TO Y2 DO FOR I:=1 downto 0 DO
                   IF DisplayChar(Alea,X2-I,Y) THEN exit;
                Y2:=Y2+1;
                FOR X:=X2 downto X1 DO
                   IF DisplayChar(Alea,X,Y2) THEN exit;
                X1:=X1-2;
                FOR Y:=Y2 downto Y1 DO FOR I:=1 downto 0 DO
                   IF DisplayChar(Alea,X1+I,Y) THEN exit;
                Y1:=Y1-1
             UNTIL X=0
          END;
      2 : FOR Y:=1 TO 25 DO IF (Y and 1)=1
            THEN FOR X:=1 TO 80 DO
               IF DisplayChar(Alea,X,Y) THEN exit ELSE
            ELSE FOR X:=80 downto 1 DO
               IF DisplayChar(Alea,X,Y) THEN exit;
      3 : FOR Y:=25 downto 1 DO IF (Y and 1)=1
            THEN FOR X:=1 TO 80 DO
```

```
                IF DisplayChar(Alea,X,Y) THEN exit ELSE
             ELSE FOR X:=80 downto 1 DO
                IF DisplayChar(Alea,X,Y) THEN exit;
      4 : FOR X:=1 TO 80 DO IF (X and 1)=1
             THEN FOR Y:=1 TO 25 DO
                IF DisplayChar(Alea,X,Y) THEN exit ELSE
             ELSE FOR Y:=25 downto 1 DO
                IF DisplayChar(Alea,X,Y) THEN exit;
      5 : FOR X:=80 downto 1 DO IF (X and 1)=1
             THEN FOR Y:=1 TO 25 DO
                IF DisplayChar(Alea,X,Y) THEN exit ELSE
             ELSE FOR Y:=25 downto 1 DO
                IF DisplayChar(Alea,X,Y) THEN exit
   END
END; { Blackboard }

BEGIN
   IF M THEN GetPosBut(mb, mx, my);
   OK:=FALSE;
   SaveScreen(@s);
   REPEAT BlackBoard UNTIL OK;
   WHILE keypressed DO C1:=ord(readkey);
   RestoreScreen(@s)
END; { Mondrian }

{$F+} PROCEDURE left(i, x, y : BYTE); {$F-}
VAR j : BYTE;
BEGIN
   FOR j:=0 TO 4 DO BEGIN
      move(_S^[y+j, x-2*i], _S^[y+j, x-2*(i+1)], 20);
      fillchar(_S^[y+j, x-2*i+8], 4, 0)
   END
END; { left }

{$F+} PROCEDURE right(i, x, y : BYTE); {$F-}
VAR j : BYTE;
BEGIN
   FOR j:=0 TO 4 DO BEGIN
      move(_S^[y+j, x+2*i], _S^[y+j, x+2*(i+1)], 20);
      fillchar(_S^[y+j, x+2*i], 4, 0)
   END
END; { right }

{$F+} PROCEDURE up(i, x, y : BYTE); {$F-}
VAR j : BYTE;
BEGIN
   FOR j:=0 TO 4 DO move(_S^[y-i+j, x], _S^[y-i-1+j, x], 20);
   fillchar(_S^[y-i+4, x], 20, 0)
END; { up }

{$F+} PROCEDURE down(i, x, y : BYTE); {$F-}
VAR j : BYTE;
BEGIN
   FOR j:=4 downto 0 DO move(_S^[y+i+j, x], _S^[y+i+1+j, x], 20);
   fillchar(_S^[y+i, x], 20, 0)
END; { down }

PROCEDURE Puzzle(M : boolean);
VAR a, b, d, x, y, dir : BYTE;
    dx, dy             : shortint;
    slip               : moveproc;
    s                  : ScreenPage;
    mb, mx, my,
    nb, nx, ny         : INTEGER;
    ok                 : boolean;
BEGIN
   ok:=FALSE;
   IF M THEN GetPosBut(mb, mx, my);
   SaveScreen(@s);
   a:=random(8);
   b:=random(5);
   dir:=0;
   REPEAT
      REPEAT
         REPEAT d:=random(4) UNTIL d<>dir;
         dx:=0;
         dy:=0;
         CASE d OF
            0 : dx:=-1;
            1 : dx:= 1;
            2 : dy:=-1;
            3 : dy:= 1
         END
      UNTIL (0<=a+dx) and (a+dx<8) and (0<=b+dy) and (b+dy<5);
      IF odd(d) THEN dir:=d-1 ELSE dir:=d+1;
      inc(a, dx);
      inc(b, dy);
      x:=10*a+1;
      y:= 5*b+1;
      CASE d OF
         0 : slip:=right;
         1 : slip:=left;
         2 : slip:=down;
         3 : slip:=up
      END;
      FOR d:=0 TO 4 DO BEGIN
         slip(d, x, y);
         delay(100)
      END;
      IF M THEN BEGIN
         GetPosBut(nb, nx, ny);
         OK:=(mb<>nb) or (mx<>nx) or (my<>ny)
      END;
      IF keypressed THEN ok:=TRUE
   UNTIL ok;
   WHILE keypressed DO a:=ord(readkey);
   RestoreScreen(@s)
END; { Puzzle }

PROCEDURE ScreenPeace(M : boolean);
BEGIN
   IF random(2)=0 THEN Mondrian(M) ELSE Puzzle(M)
END; { ScreenPeace }

BEGIN
   randomize
END.
```

## *Library TP_WIN_E*

### Function : Library for windows, masks and menus

```
UNIT TP_Win_E;

INTERFACE

USES Crt, Dos, Tools, Math, Error, Mouse, Enjoy;

TYPE
   CursorType     = (On, Off, Expand);
   MarkerType     = (Normal, Reverse);
```

Type of variable allowing to keep the state of the current window. To be used with GetScreenAspect, SetScreenAspect

```
   ScreenStatus   = RECORD
                       Wmin, Wmax   : WORD;
                       Watt, Wx, Wy : BYTE
                    END;
```

Type of mask for the screen

```
   ScreenFill     = ARRAY[BOOLEAN] OF WORD;
   ScreenColor    = ARRAY[BOOLEAN] OF BYTE;
   ScreenFrame    = ARRAY[BOOLEAN] OF Str8;
```

Definition of a window: use WinDef only. MaskSet = (Imask, Rmask, Smask); It is declared here but is used only internally by the procedures of masks management.

```
   WinDef         = ^WindowDef;
   WindowDef      = RECORD
                       xmin, ymin, xmax, ymax,
                       wfg, wbg, rfg, rbg, ffg, fbg : BYTE;
                       frame                        : Str8;
                       buffer, shadow               : ArrayOfbytePtr;
```

```
                      below                          : WinDef
                    END;

MaskSet          = (Imask, Rmask, Smask);
```

Pointer preserving the parameters of the masks.

```
MaskDef          = ^MaskRec;
MaskRec          = RECORD
                      last, next : MaskDef;
                      help       : WORD;
                      l, x, y, w : BYTE;
                      cast       : MaskSet;
                      CASE MaskSet OF
                         Imask : (Ivalue      : LngIntPtr;
                                  Imin, Imax  : LONGINT);
                         Rmask : (Rvalue      : RealPtr;
                                  Rmin, Rmax  : REAL);
                         Smask : (s           : StrPtr;
                                  mask        : AnyChar;
                                  upchar      : BOOLEAN)
                   END;
```

Type of the procedures of management of the menus written by the user

```
ArrayOfByte     = ARRAY[0..$FFFE] OF BYTE;
ArrayOfBytePtr  = ^ArrayOfbyte;

ActionProc      = PROCEDURE(Action : BYTE);
```

Type controling the action of the trees in the menu. To be uised with SetOption.

Root : option "root" of the tree, does'nt allow exit by ESC.

Stay : once the option is exexcuted, the window of the menu stays open.

GoBack : once the option is exexcuted, the window of the menu closes and we search a position in the preceding window

GoRoot : once the option is exexcuted, all the window s of the menu are closed down to the root

```
ActionDef       = (Root, Stay, GoBack, GoRoot);

CONST
```

SetBackScreen: Mask of the background of the screen in black, white and blue

- ■ - bits 0..7 : characters
- - bits 8..11 : foreground color
- ■ - bits 12..15: background color

```
ScreenGround : ScreenFill = ($70B0, $39B1);  { blue cobalt              }
```

Color of the title

```
_Tfg    : ScreenColor = (LightGray, White    );
_Tbg    : ScreenColor = (Black    , Magenta  );
```

Declaration of the aspect of the menus window

```
_Mwfg    : ScreenColor = (LightGray, LightCyan);
_Mwbg    : ScreenColor = (Black    , Blue     );
_Mrfg    : ScreenColor = (Black    , White    );
_Mrbg    : ScreenColor = (LightGray, Green    );
_Mffg    : ScreenColor = (LightGray, Yellow   );
_Mfbg    : ScreenColor = (Black    , Blue     );
_MFrame  : ScreenFrame = ('+-+__+-+', '_¯______');
_NoFrame : ScreenFrame = ('+-+__+-+', '¨ ¨  ¨ ¨');
```

Declaration of the aspect of the directory window

```
_Dwfg    : ScreenColor = (LightGray, LightCyan );
_Dwbg    : ScreenColor = (Black    , Cyan      );
_Drfg    : ScreenColor = (Black    , White     );
_Drbg    : ScreenColor = (LightGray, Blue      );
_Dffg    : ScreenColor = (LightGray, LightGreen);
```

```
_Dfbg   : ScreenColor = (Black    , Cyan      );
_DFrame : ScreenFrame = ('+-+__+-+', '_¯______');
_Dxmin  : BYTE        = 11;
_Dymin  : BYTE        = 10;
```

Declaration of the aspect of the seizure masks

```
_Qwfg   : ScreenColor = (LightGray, LightCyan );
_Qwbg   : ScreenColor = (Black    , Green     );
_Qrfg   : ScreenColor = (Black    , White     );
_Qrbg   : ScreenColor = (LightGray, Blue      );
_Qffg   : ScreenColor = (LightGray, LightGreen);
_Qfbg   : ScreenColor = (Black    , Green     );
_QFrame : ScreenFrame = ('+-+__+-+', '_¯______');
```

Declaration of the aspect of the error messages window

```
_Ewfg   : ScreenColor = (Black    , Yellow    );
_Ewbg   : ScreenColor = (LightGray, Red       );
_Effg   : ScreenColor = (Black    , LightRed  );
_Efbg   : ScreenColor = (LightGray, Red       );
_EFrame : ScreenFrame = ('+-+__+-+', '_¯______');
_Eymin  : BYTE        = 9;
```

Declaration of the aspect of the help window

```
_Hwfg   : ScreenColor = (LightGray, Black     );
_Hwbg   : ScreenColor = (Black    , Cyan      );
_Htfg   : ScreenColor = (Black    , White     );
_Htbg   : ScreenColor = (LightGray, Magenta   );
_Hrfg   : ScreenColor = (LightGray, Yellow    );
_Hrbg   : ScreenColor = (Black    , Red       );
_Hffg   : ScreenColor = (LightGray, LightGreen);
_Hfbg   : ScreenColor = (Black    , Cyan      );
_HFrame : ScreenFrame = ('+-+__+-+', '_¯______');
_Hxmin  : BYTE        = 10;
_Hymin  : BYTE        =  5;
_Hxmax  : BYTE        = 70;
_Hymax  : BYTE        = 20;
```

Declaration of the aspect of the reading window

```
_Swfg   : ScreenColor = (LightGray, LightCyan );
_Swbg   : ScreenColor = (Black    , Black     );
_Srfg   : ScreenColor = (Black    , Yellow    );
_Srbg   : ScreenColor = (LightGray, Red       );
_Sffg   : ScreenColor = (LightGray, LightGreen);
_Sfbg   : ScreenColor = (Black    , Black     );
_SFrame : ScreenFrame = ('+-+__+-+', '_¯______');
_Sxmin  : BYTE        =  3;
_Symin  : BYTE        =  2;
_Sxmax  : BYTE        = 76;
_Symax  : BYTE        = 22;

_HelpPath : PathStr = '';

_AllChar   : AnyChar = [#32..#255];

_GetMacro : Str15   = '';
_PutMacro : Str15   = '';

_Mouse     : BOOLEAN = FALSE;

_ScreenDelay : WORD = 5;

VAR
```

Screen: monochrome/color

```
_C         : BOOLEAN;
```

Procedure of the user treatment of the responses of the menu

```
_Action    : ActionProc;
```

Table of messages defined by the user

```
_HelpMsg : ARRAY[0..20] OF Str80;
```

```
PROCEDURE SetCsr( Size : CursorType );
PROCEDURE SetColor( ForeGround, BackGround : ScreenColor );
PROCEDURE GetScreenAspect( VAR ScrAsp : ScreenStatus );
PROCEDURE SetScreenAspect( VAR ScrAsp : ScreenStatus );
PROCEDURE CloseJob;

PROCEDURE Pause;

PROCEDURE DisplayMsg( Msg : Str64 );
PROCEDURE ClearMsg;
FUNCTION Question( Msg : Str64 ) : BOOLEAN;
PROCEDURE ErrorMsg( Error : WORD );
PROCEDURE HelpLine( help : BYTE );
PROCEDURE DisplayHelp( h : WORD );
PROCEDURE DisplayText( Path : PathStr );

PROCEDURE FillBox( xmin, ymin, xmax, ymax : BYTE;
                   Fill                   : WORD );
PROCEDURE ColorBox( xmin, ymin, xmax, ymax, color : BYTE );
PROCEDURE FrameBox( xmin, ymin, xmax, ymax : BYTE;
                    frame                  : Str8;
                    ffg, fbg               : BYTE );

PROCEDURE OpenWin( xmin, ymin, xmax, ymax       : BYTE;
                   wfg, wbg, rfg, rbg, ffg, fbg : ScreenColor;
                   frame                        : ScreenFrame );
FUNCTION NewWin( xmin, ymin, xmax, ymax       : BYTE;
                 wfg, wbg, rfg, rbg, ffg, fbg : ScreenColor;
                 frame                        : ScreenFrame ) : WinDef;
PROCEDURE PutOnTopWin(w : WinDef);
PROCEDURE CloseWin;

PROCEDURE SetTitleWin( title : STRING );

FUNCTION EditLngInt( xmin, ymin : BYTE;
                     title      : Str80;
                     value      : LngIntPtr;
                     vmin, vmax : LONGINT;
                     l          : BYTE;
                     help       : WORD ) : BOOLEAN;
FUNCTION EditReal( xmin, ymin : BYTE;
                   title      : Str80;
                   value      : RealPtr;
                   vmin, vmax : REAL;
                   l, d       : BYTE;
                   help       : WORD ) : BOOLEAN;
FUNCTION EditString( xmin, ymin, xmax : BYTE;
                     title            : Str80;
                     s                : StrPtr;
                     l                : BYTE;
                     mask             : AnyChar;
                     upchar           : BOOLEAN;
                     help             : WORD ) : BOOLEAN;

PROCEDURE NewMask( VAR m : MaskDef );
PROCEDURE SetIMask( m        : MaskDef;
                    v        : LngIntPtr;
                    min, max : LONGINT;
                    l, x, y  : BYTE;
                    help     : WORD );
PROCEDURE SetRMask( m          : MaskDef;
                    v          : RealPtr;
                    min, max   : REAL;
                    l, d, x, y : BYTE;
                    help       : WORD );
PROCEDURE SetSMask( m          : MaskDef;
                    s          : StrPtr;
                    l, x, y, w : BYTE;
                    mask       : AnyChar;
                    upchar     : BOOLEAN;
                    help       : WORD );
PROCEDURE DisposeMask( VAR m : MaskDef );
PROCEDURE UseMask( m : MaskDef );

FUNCTION DisplayList(list       : StrPtr;
                     len        : BYTE;
                     nfile      : INTEGER;
                 VAR item       : INTEGER;
                     x1, y1, nl : INTEGER;
                     title      : STRING;
                     help       : WORD    ) : BOOLEAN;
```

```
FUNCTION GetFileName( VAR Path  : PathStr;
                          Title : Str35;
                          Help  : WORD ) : BOOLEAN;
FUNCTION ChangeDir( VAR Path : PathStr;
                        Help : WORD ) : BOOLEAN;

PROCEDURE SetMenu( m, x, y, o : BYTE );
PROCEDURE SetOption( option       : Str64;
                     menu, action : BYTE;
                     out          : ActionDef);
FUNCTION GetOption : StrPtr;
PROCEDURE MenuWin( m : INTEGER );

PROCEDURE SetBackScreen(x, y  : BYTE;
                        title : Str80);

IMPLEMENTATION

CONST
```

Declaration of the size of the directory window: rowsxcolumns

```
   DV   = 8;
   DH   = 4;
   PAGE = DV*DH;
```

Maximal number-1  of options allowed by menu

```
   MAX_O = 19;
```

Maximal number -1 of allowed menus

```
   MAX_M = 31;
```

Size of the buffer for the text-files

```
   LINEMAX = 10000;

   PathChar : AnyChar = ['!'..'*',',','-'..';','=','?'..'{','}','~','Ç'..'_'];

TYPE
   BytePtr    = ^BYTE;
   ScreenChar = RECORD
                   C : CHAR;
                   A : BYTE
                END;
   ScreenLine = ARRAY[1..80] OF ScreenChar;
   ScreenPage = ARRAY[1..25] OF ScreenLine;

   StrBuffer  = ARRAY[1..LINEMAX] OF ^STRING;
   StrBuffPtr = ^StrBuffer;
   DirArray   = ARRAY[1..512] OF Str15;

   OptionDef  = RECORD
                   Option       : StrPtr;
                   Menu, Action : BYTE;
                   Out          : ActionDef
                END;
   OptionPtr  = ^OptionDef;
   OptionList = ARRAY[0..MAX_O] OF OptionPtr;
   MenuDef    = RECORD
                   x, y, o, w : BYTE;
                   n          : SHORTINT;
                   Options    : OptionList
                END;
   MenuPtr    = ^MenuDef;

   MoveProc   = PROCEDURE(VAR source, dest; count : WORD);

CONST
   _W    : WinDef          = NIL;
   _Mask : MaskDef         = NIL;
```

Pile of menus

```
   _MW   : String[MAX_M+1] = '';
   _MM   : Boolean         = FALSE;
   _ME   : Boolean         = FALSE;
```

```
VAR
   _S                            : ^ScreenPage;
   _R                            : BOOLEAN;
   _Dir                          : ^DirArray;
   _M, _Wx, _Wy                  : BYTE;
   _Wmin, _Wmax, _Watt, _T       : WORD;
   _Menu                         : ARRAY[0..MAX_M] OF MenuPtr;
```

Keyboard and mouse management with copyright

```
PROCEDURE ShowMouse;
BEGIN
   IF _Mouse THEN Mouse.ShowMouse
END; { ShowMouse }

PROCEDURE HideMouse;
BEGIN
   IF _Mouse THEN Mouse.HideMouse
END; { HideMouse }

PROCEDURE NullMouse;
VAR mouse, x, y : INTEGER;
BEGIN
   IF _Mouse THEN REPEAT GetPosBut(mouse, x, y) UNTIL mouse=0
END; { NullMouse }
```

Waiting a pressure on the keyboard or a touch of the mouse. Analogous to Pause of the Tools but with the mouse in extra. To avoid any confusion between the two procedures you must point to the correct one by using the chosen Library.

```
PROCEDURE Pause;
VAR mouse, x, y : INTEGER;
BEGIN
   NullMouse;
   mouse:=0;
   REPEAT
      IF _Mouse THEN GetPosBut(mouse, x, y)
   UNTIL keypressed or (mouse<>0);
   IF keypressed THEN x:=ord(readkey)
END; { Pause }

FUNCTION Inkey : WORD;
VAR Key          : CHAR;
    s, t, attr   : WORD;
    mouse, x, y  : INTEGER;
    w            : WinDef;
    i            : BYTE;
BEGIN
   GetTime(s, _T, s, s);
   NullMouse;
   IF _MM THEN ShowMouse;
   mouse:=0;
   WHILE (NOT KeyPressed) and (mouse=0) DO
      BEGIN
         GetTime(s, t, s, s);
         IF t-_T=_ScreenDelay THEN BEGIN
            attr:=TextAttr;
            IF _MM THEN HideMouse;
            ScreenPeace(_MM);
            GetTime(s, _T, s, s);
            IF _MM THEN ShowMouse;
            TextAttr:=attr
         END;
         IF _Mouse and (_MM or _ME) THEN BEGIN
            GetPosBut(mouse, x, y);
            x:=(x div 8)+1;
            y:=(y div 8)+1
         END
      END;
   IF _MM THEN HideMouse;
   IF _Mouse and (_MM or _ME) and (mouse<>0) THEN
      IF mouse=2 THEN
         Inkey:=ESC
      ELSE IF _ME THEN
         Inkey:=CR
      ELSE BEGIN
         w:=_W;
         i:=length(_MW);
         WHILE (w<>NIL) and
               ((x<w^.xmin) or (w^.xmax<x) or (y<w^.ymin) or (w^.ymax<y))
         DO BEGIN
            _PutMacro:=_PutMacro+CHR(ESC);
            dec(i);
            w:=w^.below
         END;
         IF w<>NIL THEN
            IF w=_W THEN
               Inkey:=ord(_Menu[ORD(_MW[i])]^.Options[y-w^.ymin-1]^.Option^[1])
            ELSE BEGIN
               _PutMacro:=_PutMacro+
                          _Menu[ORD(_MW[i])]^.Options[y-w^.ymin-1]^.Option^[1]+
                          CHR(CR);
               Inkey:=ESC
            END
         ELSE BEGIN
            _PutMacro:='';
            Inkey:=0
         END
      END
   ELSE BEGIN
      Key:=ReadKey;
      IF Key=#0
         THEN BEGIN
            Key:=ReadKey;
            Inkey:=Ord(Key) SHL 8
         END
         ELSE Inkey:=Ord(Key)
   END
END; { Inkey }
```

Screen management

```
PROCEDURE GetVideo;
VAR Regs : Registers;
    Cmd  : ComStr;
    i, p : BYTE;
BEGIN
   Intr($11,Regs);
   _R:=Lo(Regs.AX) AND $30<>$30;
   _S:=Ptr($B000+Ord(_R)*$800,$0);
   i:=1;
   REPEAT
      Cmd:=ParamStr(i);
      Inc(i);
      FOR p:=1 TO Length(Cmd) DO Cmd[p]:=UpCase(Cmd[p])
   UNTIL (0=Length(Cmd)) OR (0<Pos('/BW', Cmd));
   IF _R
      THEN _C:=Pos('/BW', Cmd)=0
      ELSE _C:=_R
END; { GetVideo }
```

Definition of the aspect of the cursor:
- On : normal form
- Off : extinction
- Expand : cursor under the form of a slab to mark the insertion

```
PROCEDURE SetCsr;
CONST CsrSize : ARRAY[BOOLEAN, CursorType] OF INTEGER
              = (($0C0D,$0E00,$000D),($0607,$2000,$0007));
VAR Regs : Registers;
BEGIN
   Regs.AX:=$0100;
   Regs.CX:=CsrSize[_R, Size];
   Intr($10,Regs)
END; { SetCsr }
```

Defining the colors in function of the screen type

```
PROCEDURE SetColor;
BEGIN
   TextColor(ForeGround[_C]);
   TextBackGround(BackGround[_C])
END; { SetColor }
```

Recall the state of the open window

```
PROCEDURE GetScreenAspect;
BEGIN
```

```
   ScrAsp.Wmin:=WindMin;
   ScrAsp.Wmax:=WindMax;
   ScrAsp.Watt:=TextAttr;
   ScrAsp.Wx  :=WhereX;
   ScrAsp.Wy  :=WhereY;
   Window(1, 1, 80, 25)
END; { GetScreenAspect }
```

Restore the state of a window

```
PROCEDURE SetScreenAspect;
BEGIN
   Window(Lo(ScrAsp.Wmin)+1, Hi(ScrAsp.Wmin)+1,
          Lo(ScrAsp.Wmax)+1, Hi(ScrAsp.Wmax)+1);
   TextAttr:=ScrAsp.Watt;
   GotoXY(ScrAsp.Wx, ScrAsp.Wy)
END; { SetScreenAspect }
```

Makes the output of the screen and put it back into its initial state

```
PROCEDURE CloseJob;
BEGIN
   TextColor(LightGray);
   TextBackGround(Black);
   Window(1,1,80,25);
   GotoXY(1, 25);
   ClrEol;
   GotoXY(1, 24);
   SetCsr(On)
END; { CloseJob }
```

## Management of the different zones of the screen

The procedures FillBox and ColorBox can generate some parasites on some screens

Fills a ectangle given by xmin, ymin, xmax, ymax with the sign Fill:
- bits 0..7 : character
- bits 8..11 : foreground color
- bits 12..15: background color

```
PROCEDURE FillBox;
BEGIN
   Dec(xmax, xmin-1);
   FOR ymin:=ymin TO ymax DO FillWord(_S^[ymin, xmin], xmax, Fill)
END; { FillBox }
```

Puts the color on a rectangle given by xmin, ymin, xmax, ymax with the color without changing the content of the characters

```
PROCEDURE ColorBox;
VAR x, y : BYTE;
BEGIN
   FOR y:=ymin TO ymax DO
      FOR x:=xmin TO xmax DO
         _S^[y, x].A:=color
END; { ColorBox }
```

Frames a rectangle given by xmin, ymin, xmax, ymax with the defined frame and it colors.

```
PROCEDURE FrameBox;
VAR i : BYTE;
    s : ScreenStatus;
BEGIN
   GetScreenAspect(s);
   TextColor(ffg);
   TextBackGround(fbg);
   GotoXY(xmin, ymin); Write(frame[1]);
   FOR i:=xmin+1 TO xmax-1 DO Write(frame[2]);
   Write(frame[3]);
   FOR i:=ymin+1 TO ymax-1 DO
      BEGIN
```

```
         GotoXY(xmin, i); Write(frame[4]);
         GotoXY(xmax, i); Write(frame[5])
      END;
   GotoXY(xmin, ymax); Write(frame[6]);
   FOR i:=xmin+1 TO xmax-1 DO Write(frame[7]);
   Write(frame[8]);
   SetScreenAspect(s)
END; { FrameBox }
```

Windows manager

The system manages a simple pile of windows. These procedures do not verifiy if the windows are running over the screen

```
{$F+} PROCEDURE Save(VAR Screen, Buffer; Count : WORD); {$F-}
BEGIN
   Move(Screen, Buffer, Count)
END; { Save }

{$F+} PROCEDURE Restore(VAR Screen, Buffer; Count : WORD); {$F-}
BEGIN
   Move(Buffer, Screen, Count)
END; { Restore }
```

StoreWin can create parasites

```
PROCEDURE StoreWin(w     : WinDef;
                   Store : MoveProc);
VAR x, DX, y : WORD;
BEGIN
   DX:=2*(w^.xmax-w^.xmin+3);
   x:=0;
   FOR y:=w^.ymin TO w^.ymax+1 DO
      BEGIN
         Store(_S^[y, w^.xmin], w^.Buffer^[x], DX);
         Inc(x, DX)
      END
END; { StoreWin }

PROCEDURE DropWin;
VAR x, y : BYTE;
BEGIN
   IF _W=NIL THEN EXIT;
   x:=2*(_W^.xmax-_W^.xmin+1);
   Move(_W^.shadow^, _S^[_W^.ymax+1, _W^.xmin+2], x);
   FOR y:=_W^.ymin+1 TO _W^.ymax DO
      BEGIN
         Move(_W^.shadow^[x], _S^[y, _W^.xmax+1], 4);
         Inc(x, 4)
      END;
   FrameBox(_W^.xmin, _W^.ymin, _W^.xmax, _W^.ymax,
            _NoFrame[_C], _W^.ffg, _W^.fbg)
END; { DropWin }
```

Quickens the window defined by <w> by putting it on top of the pile. Must be open by NewWin.

```
PROCEDURE PutOnTopWin;
VAR below, above : WinDef;
BEGIN
   IF w=NIL THEN BEGIN
      _W:=w;
      EXIT
   END;
   above:=NIL;
   below:=_W;
   WHILE (below<>NIL) and (below<>w) DO BEGIN
      above:=below;
      below:=below^.below
   END;
   IF (below<>NIL) and (below<>_W) THEN BEGIN
      DropWin;
      IF above<>NIL THEN above^.below:=below^.below;
      w^.below:=_W
   END;
   _W:=w;
   FrameBox(w^.xmin, w^.ymin, w^.xmax, w^.ymax, w^.frame, w^.ffg, w^.fbg);
   ColorBox(w^.xmin+2, w^.ymax+1, w^.xmax+2, w^.ymax+1, $07);
   ColorBox(w^.xmax+1, w^.ymin+1, w^.xmax+2, w^.ymax  , $07);
   Window(w^.xmin+1, w^.ymin+1, w^.xmax-1, w^.ymax-1);
   TextColor(w^.wfg);
   TextBackGround(w^.wbg)
END; { PutOnTopWin }
```

Set a title to the active window.

```
PROCEDURE SetTitleWin;
BEGIN
   Window(1,1,80,25);
   TextColor(_W^.ffg);
   TextBackGround(_W^.fbg);
   title:=Concat(_W^.frame[3], #32, Title, #32, _W^.frame[1]);
   title:=Copy(title, 1, _W^.xmax-_W^.xmin-3);
   FrameBox(_W^.xmin, _W^.ymin, _W^.xmax, _W^.ymax, _W^.frame, _W^.ffg,
            _W^.fbg);
   GotoXY(_W^.xmin+2, _W^.ymin);
   Write(title);
   Window(_W^.xmin+1, _W^.ymin+1, _W^.xmax-1, _W^.ymax-1);
   TextColor(_W^.wfg);
   TextBackGround(_W^.wbg);
   GotoXY(1,1)
END; { SetTitleWin }
```

Opens a window of coordinates xmin, ymin, xmax, ymax
with colors given by:
- wfg, wbg : content of window
- rfg, rbg : reverse video
- ffg, fbg : type ScreenColor

```
PROCEDURE OpenWin;
VAR w    : WinDef;
    x, y : BYTE;
BEGIN
   New(w);
   w^.xmin :=xmin;
   w^.ymin :=ymin;
   w^.xmax :=xmax;
   w^.ymax :=ymax;
   w^.wfg  :=wfg[_C];
   w^.wbg  :=wbg[_C];
   w^.rfg  :=rfg[_C];
   w^.rbg  :=rbg[_C];
   w^.ffg  :=ffg[_C];
   w^.fbg  :=fbg[_C];
   w^.frame:=frame[_C];
   GetMem(w^.buffer, 2*(xmax-xmin+3)*(ymax-ymin+3));
   GetMem(w^.shadow, 2*(xmax-xmin+1+2*(ymax-ymin)));
   w^.below:=_W;
   DropWin;
   StoreWin(w, Save);
   x:=2*(w^.xmax-w^.xmin+1);
   Move(_S^[w^.ymax+1, w^.xmin+2], w^.shadow^, x);
   FOR y:=w^.ymin+1 TO w^.ymax DO
      BEGIN
         Move(_S^[y, w^.xmax+1], w^.shadow^[x], 4);
         Inc(x, 4)
      END;
   PutOnTopWin(w);
   ClrScr
END; { OpenWin }
```

Same as OpenWin but return the pointer of a window

```
FUNCTION NewWin;
BEGIN
   OpenWin(xmin, ymin, xmax, ymax,
           wfg, wbg, rfg, rbg, ffg, fbg, frame);
   NewWin:=_W
END; { NewWin }
```

Closes the active window.

```
PROCEDURE CloseWin;
VAR W : WinDef;
BEGIN
```

```
    IF _W=NIL THEN EXIT;
    w:=_W;
    StoreWin(_W, Restore);
    _W:=_W^.below;
    PutOnTopWin(_W);
    FreeMem(w^.buffer, 2*(w^.xmax-w^.xmin+3)*(w^.ymax-w^.ymin+3));
    FreeMem(w^.shadow, 2*(w^.xmax-w^.xmin+1+2*(w^.ymax-w^.ymin)));
    Dispose(w)
END; { CloseWin }
```

Messages and questions

Opens a window containing the message.

```
PROCEDURE DisplayMsg;
VAR x : BYTE;
BEGIN
   x:=((80-Length(Msg)) DIV 2)-1;
   OpenWin(x, _Eymin, x+Length(msg)+3, _Eymin+2,
           _Ewfg, _Ewbg, _Ewfg, _Ewbg, _Effg,_Efbg, _EFrame);
   Write(' '+Msg)
END; { DisplayMsg }
```

Closes the active window open by DisplayMsg.

```
PROCEDURE ClearMsg;
BEGIN
   CloseWin
END; { ClearMsg }
```

Opens a window containing the error message

```
PROCEDURE ErrorMsg;
VAR oldline : ScreenLine;
BEGIN
   oldline:=_S^[25];
   HelpLine(201);
   DisplayMsg(ErrorStr(Error));
   Pause;
   ClearMsg;
   _S^[25]:=oldline
END; { ErrorMsg }
```

Opens a window with the question + '[Y/N] ?' and waits for the response Y, N or ESC; Is true if the answer is positive

```
FUNCTION Question;
VAR Answer      : CHAR;
    mouse, x, y : INTEGER;
BEGIN
   NullMouse;
   DisplayMsg(Msg+' [Y/N] ?');
   mouse:=0;
   Answer:=#0;
   REPEAT
      IF _Mouse THEN getPosBut(mouse, x, y);
      IF keypressed THEN Answer:=UpCase(ReadKey)
   UNTIL (Answer IN [#27, 'N', 'Y']) or (mouse in [1, 2]);
   ClearMsg;
   Question:=(Answer='Y') or (mouse=1)
END; { Question }
```

Displays help at the bottom of the screen

```
PROCEDURE HelpLine;
VAR msg : STRING;
    s   : ScreenStatus;
BEGIN
   CASE Help OF
    0..20 : msg:=_HelpMsg[Help];
      201 : msg:='Strike a key TO continue...';
      202 : msg:='F1: help _ '#30#31': point _ First letter or ENTER: select';
      203 : msg:='F1: help _ '#30#31': point _ '
                 +'First letter or ENTER: select _ ESC: main menu';
      204 : msg:='F1: help _ HOME '#17#16' END INS DEL <= ^HOME ^END: '
                 +'edit _ ESC: undo';
      205 : msg:='F1: help _ HOME PGUP '#30#17#16#31' PGDN END: point _ ENTER:'
```

```
                +' select _ ESC: undo';

      207 : msg:='F2:find _ F3:find next _ PGUP HOME '#17#30#31#16' END PGDN '
                +'^PGUP ^PGDN:scroll _ ESC:quit';
      208 : msg:='HOME '#17#16' END INS DEL <= ^HOME ^END: edit _ '
                +'ESC: undo';
      209 : msg:='TAB '#31#30' '#24'TAB :point _ '
                +'HOME '#17#16' END INS DEL <= ^HOME ^END: edit _ '
                +'ENTER: quit'
   END;
   GetScreenAspect(s);
   TextColor(Black);
   TextBackGround(LightGray);
   GotoXY(1, 25);
   ClrEol;
   GotoXY((82-Length(msg)) DIV 2, 25);
   Write(msg);
   SetScreenAspect(s)
END; { HelpLine }
```

## Display of an ASCII file and Help

```
FUNCTION LoadTextFile(Path         : PathStr;
                      Line         : StrBuffPtr;
                  VAR N, Len, Top  : INTEGER;
                      Height       : INTEGER;
                      h            : BYTE      ) : BOOLEAN;
VAR f     : TEXT;
    s     : STRING;
    index : Str5;
BEGIN
   IF FSearch(Path, GetEnv('PATH'))=''
      THEN BEGIN
         IF h=0
            THEN ErrorMsg(2)
            ELSE ErrorMsg(246);
         LoadTextFile:=FALSE;
         EXIT
      END
      ELSE LoadTextFile:=TRUE;
   DisplayMsg('Loading...');
   GetMemory(TextSize(Path), 10000);
   Assign(f, Path);
   Reset(f);
   index:=Concat('@', IStr(h,0));
   N:=1;
   Line^[N]:=_UserHeap;
   Len:=0;
   Top:=0;
   WHILE (NOT Eof(f)) AND (0<_UserMemory) AND (N<LINEMAX) DO
      BEGIN
         Readln(f, s);
         IF s[1]=#64
            THEN IF (s=index) AND (Top=0)
               THEN Top:=N
               ELSE
            ELSE IF Length(s)+1<=_UserMemory
               THEN BEGIN
                  Move(s, Line^[N]^, Length(s)+1);
                  IF Len<Length(Line^[N]^) THEN Len:=Length(Line^[N]^);
                  Inc(N);
                  Line^[N]:=AddAddr(Line^[N-1], Length(s)+1);
                  Dec(_UserMemory, Length(s)+1)
               END
               ELSE _UserMemory:=0
      END;
   IF NOT Eof(f) THEN ErrorMsg(254);
   Close(f);
   Dec(N);
   IF (Top=0) THEN Top:=1;
   Top:=IMin(Top, IMax(1, N-Height));
   ClearMsg;
   LoadTextFile:=0<N
END; { LoadTextFile }

PROCEDURE DisplayTextFile(Line                         : StrBuffPtr;
                          N, Len, Top, Height, Width   : INTEGER;
                          Title                        : STRING;
                          wfg, wbg, rfg, rbg, tfg, tbg : ScreenColor);
VAR Margin, Sx, Sy, mouse, x, y : INTEGER;
    Key                         : WORD;
```

```
     Search                         : STRING;
     Find                           : BOOLEAN;

    PROCEDURE SearchString(Where : INTEGER);
    VAR s : STRING;
    BEGIN
       Sy:=Where;
       IF N<Sy
          THEN BEGIN
             Sx:=1;
             Sy:=1
          END
          ELSE Inc(Sx);
       s:=Copy(Line^[Sy]^, Sx, 255);
       WHILE (Sy<=N) AND (Pos(Search, s)=0) DO
          BEGIN
             Sx:=1;
             Inc(Sy);
             s:=Line^[Sy]^
          END;
       Find:=(Sy<=N) AND (0<Pos(Search, s));
       IF Find
          THEN BEGIN
             Sx:=Pos(Search, s)+Sx-1;
             IF Sx<Margin
                THEN Margin:=Sx
                ELSE IF Margin+Width<=Sx
                   THEN Margin:=IMin(Sx, Len-Width+1);
             IF Sy<Top
                THEN Top:=Sy
                ELSE IF Top+Height<=Sy
                   THEN Top:=IMin(Sy, N-Height)
          END
          ELSE BEGIN
             ErrorMsg(252);
             SetTitleWin(Title)
          END
    END; { SearchString }

    PROCEDURE Page;
    VAR L : WORD;
    BEGIN
       FOR L:=Top TO IMin(Top+Height, N) DO
          BEGIN
             IF Line^[L]^[1]=#254 THEN SetColor(tfg, tbg);
             GotoXY(2, L-Top+1);
             Write(Copy(Line^[L]^, Margin, Width));
             IF Line^[L]^[1]=#254 THEN SetColor(wfg, wbg);
             ClrEol
          END;
       IF Find AND
          (Margin<=Sx) AND (Sx+Length(Search)-1<Margin+Width) AND
          (Top<=Sy) AND (Sy<=Top+Height)
          THEN BEGIN
             SetColor(rfg, rbg);
             GotoXY(Sx-Margin+2, Sy-Top+1);
             Write(Search);
             SetColor(wfg, wbg)
          END
    END; { Page }

BEGIN { DisplayTextFile }
    SetTitleWin(Title);
    Search:='';
    Find:=FALSE;
    Margin:=1;
    Key:=0;
    REPEAT
       Page;
       _ME:=TRUE;
       ShowMouse;
       IF _Mouse THEN BEGIN
          IF (hi(Key)<>UP  ) and (hi(Key)<>DOWN) and
             (hi(Key)<>LEFT) and (hi(Key)<>RGHT) THEN StillMouse;
          Key:=0;
          REPEAT
             IF keypressed THEN
                Key:=Inkey
             ELSE BEGIN
                GetPosBut(mouse, x, y);
                x:=(x div 8)+1;
                y:=(y div 8)+1;
```

```
               IF mouse=2 THEN
                  Key:=ESC
               ELSE IF (mouse=1) and (y=25) THEN
                  CASE x OF
                      1.. 9 : Key:=F2   shl 8;
                     11..24 : Key:=F3   shl 8;
                     27..30 : Key:=PGUP shl 8;
                     32..35 : Key:=HOME shl 8;
                     37     : Key:=LEFT shl 8;
                     38     : Key:=UP   shl 8;
                     39     : Key:=DOWN shl 8;
                     40     : Key:=RGHT shl 8;
                     42..44 : Key:=ENDK shl 8;
                     46..49 : Key:=PGDN shl 8;
                     51..55 : Key:=CPUP shl 8;
                     57..61 : Key:=CPDN shl 8;
                     71..80 : Key:=ESC
                  END
            END
         UNTIL Key<>0
      END
      ELSE Key:=Inkey;
      HideMouse;
      _ME:=FALSE;
      CASE Hi(Key) OF
         LEFT : IF 2<Margin
                   THEN Dec(Margin, 2)
                   ELSE Margin:=1;
         RGHT : IF Margin<Len-Width
                   THEN Inc(Margin, 2)
                   ELSE Margin:=Len-Width+1;
         CLFT : IF 0<Margin-Width
                   THEN Dec(Margin, Width)
                   ELSE Margin:=1;
         CRHT : IF Margin+Width<Len-Width
                   THEN Inc(Margin, Width)
                   ELSE Margin:=IMax(1, Len-Width+1);
         HOME : Margin:=1;
         ENDK : Margin:=IMax(1, Len-Width+1);
         UP   : IF 1<Top THEN Dec(Top);
         DOWN : IF Top<N-Height THEN Inc(Top);
         PGUP : IF 1<Top-Height
                   THEN Dec(Top, Height+1)
                   ELSE Top:=1;
         PGDN : IF Top+Height<N-Height
                   THEN Inc(Top, Height+1)
                   ELSE Top:=IMax(1, N-Height);
         CPUP : Top:=1;
         CPDN : Top:=IMax(1, N-Height);
         F2   : BEGIN
                   IF EditString(_W^.xmin+4 , _W^.ymin+2, _W^.xmin+44,
                                 'Search STRING', @Search,
                                 SizeOf(Search)-1, _AllChar, FALSE, 0)
                      THEN SearchString(N+1);
                   SetTitleWin(Title)
                END;
         F3   : SearchString(Sy)
      END
   UNTIL Key=ESC;
   CloseWin;
   Release(_UserHeap)
END; { DisplayTextFile }
```

Displays an ASCII file as help. That file is declared by the global variable _HelpPath.

```
PROCEDURE DisplayHelp(h : WORD);
VAR Line                : StrBuffPtr;
    N, Top, Len, Height : INTEGER;
    oldline             : ScreenLine;
BEGIN
   IF h=0 THEN EXIT;
   oldline:=_S^[25];
   New(Line);
   IF LoadTextFile(_HelpPath, Line, N, Len, Top, _Hymax-_Hymin-2, h)
      THEN BEGIN
         HelpLine(207);
         OpenWin(_Hxmin, _Hymin, _Hxmax, _Hymax, _Hwfg, _Hwbg, _Hwfg, _Hwbg,
                 _Hffg, _Hfbg, _HFrame);
         DisplayTextFile(Line, N, Len, Top, _Hymax-_Hymin-2, _Hxmax-_Hxmin-3,
                         'H E L P', _Hwfg, _Hwbg, _Hrfg, _Hrbg, _Htfg, _Htbg)
   END;
   Dispose(Line);
   _S^[25]:=oldline
END; { DisplayHelp }
```

Displays an ASCII file indicated by Path the rows of which are not exceeding 255 characters.

```
PROCEDURE DisplayText;
VAR Line                  : StrBuffPtr;
    N, Top, Len, Height   : INTEGER;
    oldline               : ScreenLine;
BEGIN
   oldline:=_S^[25];
   New(Line);
   IF LoadTextFile(Path, Line, N, Len, Top, _Symax-_Symin-2, 0)
      THEN BEGIN
         HelpLine(207);
         OpenWin(_Sxmin, _Symin, _Sxmax, _Symax, _Swfg, _Swbg, _Swfg, _Swbg,
                  _Sffg, _Sfbg, _SFrame);
         DisplayTextFile(Line, N, Len, Top, _Symax-_Symin-2, _Sxmax-_Sxmin-3,
                         Path, _Swfg, _Swbg, _Srfg, _Srbg, _Swfg, _Swbg)
   END;
   Dispose(Line);
   _S^[25]:=oldline
END; { DisplayText }
```

## Mask

| | | | |
|---|---|---|---|
| s | : StrPtr | : | variable to be edited |
| l | : Byte | : | maximal length of the string |
| x, y | : Byte | : | position of the edition zone in the window |
| w | : Byte | : | width of the edition zone |
| mask | : AnyChar | : | allowed characters |
| upchar | : Boolean | : | conversion into capitals |
| loQ, hiQ | : AnyByte | : | keys allowing an exit equivalent ot ENTER which is always defined by default, excluding ESC. |
| key | : Word | : | key of exit |
| fore, back | : ScreenColor | : | color of the mask |
| help | : Word | : | index of the help |

The function returns FALSE and the variable s keeps unchanged if the exit is done by ESC.

```
FUNCTION EditLine( s           : StrPtr;
                   l, x, y, w  : BYTE;
                   mask        : AnyChar;
                   upchar      : BOOLEAN;
                   loQ, hiQ    : AnyByte;
               VAR key         : WORD;
                   help        : WORD        ) : BOOLEAN;
VAR old             : STRING;
    c, p            : WORD;
    i               : BYTE;
    mode            : CursorType;
    CH              : CHAR;
    mouse, mx, my   : INTEGER;
BEGIN
   SetColor(_Qrfg, _Qrbg);
   old:=s^;
   c:=Length(s^)+1;
   p:=1;
   mode:=On;
   loQ:=loQ+[13, 27];
   REPEAT
      IF c<p THEN p:=c;
      IF p+w-1<=c THEN p:=c-w+1;
      GotoXY(x, y);
      SetCsr(Off);
      Write(Copy(s^, p, w-1));
      FOR i:=WhereX TO x+w-1 DO Write(' ');
      SetCsr(mode);
      GotoXY(x+c-p, y);
      NullMouse;
      _ME:=TRUE;
      key:=Inkey;
      _ME:=FALSE;
      CASE Hi(key) OF
         LEFT : IF 1<c THEN Dec(c);
         RGHT : IF (c<l) AND (c<=Length(s^)) THEN Inc(c);
         HOME : c:=1;
         ENDK : c:=Length(s^)+1;
         CHOM : BEGIN
                   Delete(s^, 1, c-1);
                   c:=1
                END;
         CEND : BEGIN
                   Delete(s^, c, 255);
                   c:=Length(s^)+1
                END;
         INS  : IF mode=On
                   THEN mode:=Expand
                   ELSE mode:=On;
         DEL  : Delete(s^, c, 1);
         F1   : BEGIN
                   SetCsr(Off);
                   DisplayHelp(help);
                   SetColor(_Qrfg, _Qrbg);
                   SetCsr(mode)
                END
      END;
      IF (Lo(key)=BS) AND (1<c)
         THEN BEGIN
            Dec(c);
            Delete(s^, c, 1)
         END;
      IF (Chr(key) IN mask) AND (c<=l)
         THEN BEGIN
            CH:=Chr(key);
            IF upchar THEN CH:=UpCase(CH);
            IF mode=Expand
               THEN BEGIN
                  Insert(CH, s^, c);
                  s^:=Copy(s^, 1, l)
               END
               ELSE IF Length(s^)<c
                  THEN s^:=s^+CH
                  ELSE s^[c]:=CH;
            Inc(c)
         END
   UNTIL (Lo(key) IN loQ) OR (Hi(key) IN hiQ);
   Setcsr(Off);
   IF Lo(key)=ESC
      THEN BEGIN
         s^:=old;
         EditLine:=FALSE
      END
      ELSE EditLine:=TRUE
END; { EditLine }

FUNCTION EditValue( value      : RealPtr;
                    vmin, vmax : REAL;
                    l, d, x, y : BYTE;
                    loQ, hiQ   : AnyByte;
                VAR key        : WORD;
                    help       : WORD ) : BOOLEAN;
VAR new   : REAL;
    err   : INTEGER;
    s     : STRING;
    out   : BOOLEAN;
    digit : AnyChar;
BEGIN
   Str(value^:0:d, s);
   digit:=['0'..'9'];
   IF vmin<0 THEN digit:=digit+['-'];
   IF d<>0 THEN digit:=digit+['.'];
   REPEAT
      out:=EditLine(@s, l, x, y, l+1, digit, FALSE, loQ, hiQ, key, help);
      Val(s, new, err);
      IF out
         THEN IF err<>0
            THEN ErrorMsg(106)
            ELSE IF (new<vmin) OR (vmax<new)
               THEN ErrorMsg(201)
   UNTIL (out=FALSE) OR ((vmin<=new) AND (new<=vmax) AND (err=0));
   IF out THEN value^:=new;
   EditValue:=out
END; { EditValue }
```

## Edition of an integer.

```
FUNCTION EditLngInt;
VAR oldline : ScreenLine;
    key     : WORD;
    v       : REAL;
BEGIN
   oldline:=_S^[25];
   IF help=0
      THEN HelpLine(208)
      ELSE HelpLine(204);
   OpenWin(xmin, ymin, xmin+Length(title)+l+4, ymin+2,
           _Qwfg, _Qwbg, _Qrfg, _Qrbg, _Qffg, _Qfbg, _QFrame);
   GotoXY(2, 1); Write(title);
   v:=Float(value^);
   EditLngInt:=EditValue(@v, vmin, vmax, l, 0, Length(title)+2, 1,
                         [], [], key, help);
   value^:=Trunc(v);
   CloseWin;
   _S^[25]:=oldline
END; { EditLngInt }
```

## Edition of a real

```
FUNCTION EditReal;
VAR oldline : ScreenLine;
    key     : WORD;
BEGIN
   oldline:=_S^[25];
   IF help=0
      THEN HelpLine(208)
      ELSE HelpLine(204);
   OpenWin(xmin, ymin, xmin+Length(title)+l+4, ymin+2,
           _Qwfg, _Qwbg, _Qrfg, _Qrbg, _Qffg, _Qfbg, _QFrame);
   GotoXY(2, 1); Write(title);
   EditReal:=EditValue(value, vmin, vmax, l, d, Length(title)+2, 1,
                       [], [], key, help);
   CloseWin;
   _S^[25]:=oldline
END; { EditReal }
```

## Edition of a string

```
FUNCTION EditString;
VAR oldline : ScreenLine;
    key     : WORD;
BEGIN
   oldline:=_S^[25];
   IF help=0
      THEN HelpLine(208)
      ELSE HelpLine(204);
   OpenWin(xmin, ymin, xmax, ymin+3,
           _Qwfg, _Qwbg, _Qrfg, _Qrbg, _Qffg, _Qfbg, _QFrame);
   GotoXY(2, 1); Write(Copy(title, 1, xmax-xmin-3));
   EditString:=EditLine(s, l, 2, 2, xmax-xmin-3, mask, upchar,
                        [], [], key, help);
   CloseWin;
   _S^[25]:=oldline
END; { EditString }
```

## Management of multiple masks

## Create a new mask which must be defined by SetMask.

```
PROCEDURE NewMask;
BEGIN
   New(m);
   m^.l:=0;
   _Mask:=m
END; { NewMask }
```

## Invariant part of the masks

```
PROCEDURE MaskList( m          : MaskDef;
                    cast       : MaskSet;
                    l, x, y, w : BYTE;
```

```
                        help        : WORD );
VAR last : MaskDef;
BEGIN
   last:=_Mask;
   IF 0<_Mask^.l THEN New(_Mask);
   last^.next:=_Mask;
   _Mask^.l    :=l;
   _Mask^.x    :=x;
   _Mask^.y    :=y;
   _Mask^.w    :=w;
   _Mask^.help :=help;
   _Mask^.cast :=cast;
   _Mask^.last :=last;
   _Mask^.next :=m;
   m^.last     :=_Mask
END; { MaskList }
```

Defining the seizure mask m to get an integer.
- v pointer on the integer to be seized
- min, max the interval in which it must be
- l the total number of digits authorized in the field
- x, y position in the active window
- help index of the help file

```
PROCEDURE SetIMask;
BEGIN
   MaskList(m, Imask, l, x, y, 0, help);
   _Mask^.Ivalue :=v;
   _Mask^.Imin   :=min;
   _Mask^.Imax   :=max
END; { SetSMask }
```

Defining the seizure mask m for a string
- s pointer on the string of characters to be edited
- l its length
- x, y position in the active window
- w width of the seizure field
- mask of the authorized character set
- upchar switch in capitals
- help index of the help file

```
PROCEDURE SetSMask;
BEGIN
   MaskList(m, Smask, l, x, y, w, help);
   _Mask^.s      :=s;
   _Mask^.mask   :=mask;
   _Mask^.upchar:=upchar
END; { SetSMask }
```

Dispose a group of masks

```
PROCEDURE DisposeMask;
VAR last, next : MaskDef;
BEGIN
   last:=m;
   REPEAT
      next:=m^.next;
      Dispose(m);
      m:=next
   UNTIL m=last
END; { DisposeMask }
```

Management of the seizures in the active window. m is the mask where the pointer is placed at the beginning. The displacement from one field to the other is done by the TAB or by the arrows up down. Exit: use ENTER.

```
PROCEDURE UseMask;
VAR key      : WORD;
    oldline  : ScreenLine;
    last     : MaskDef;
    i        : BYTE;
    rfg, rbg : ScreenColor;
    v         : REAL;
BEGIN
   oldline:=_S^[25];
   HelpLine(209);
   SetColor(_Qrfg, _Qrbg);
   last:=m;
   REPEAT
      GotoXY(m^.x, m^.y);
      CASE m^.cast OF
         Imask : Write(m^.Ivalue^);
         Rmask : Write(m^.Rvalue^:0:m^.w);
         Smask : Write(Copy(m^.s^, 1, m^.w-1))
      END;
      IF m^.cast=Smask
         THEN FOR i:=WhereX TO m^.x+m^.w-1 DO Write(' ')
         ELSE FOR i:=WhereX TO m^.x+m^.l   DO Write(' ');
      m:=m^.next
   UNTIL m=last;
   REPEAT
      CASE m^.cast OF
         Imask : BEGIN
                    v:=Float(m^.Ivalue^);
                    IF EditValue(@v, m^.Imin, m^.Imax, m^.l, m^.w, m^.x, m^.y,
                                 [TAB], [STAB, UP, DOWN], key, m^.help) THEN;
                    m^.Ivalue^:=Trunc(v)
                 END;
         Rmask : IF EditValue(m^.Rvalue, m^.Rmin, m^.Rmax,
                              m^.l, m^.w, m^.x, m^.y,
                              [TAB], [STAB, UP, DOWN], key, m^.help) THEN;
         Smask : IF EditLine(m^.s, m^.l, m^.x, m^.y, m^.w, m^.mask, m^.upchar,
                             [TAB], [STAB, UP, DOWN], key, m^.help) THEN;
      END;
      IF (Lo(key)=TAB) OR (Hi(key)=DOWN)
         THEN m:=m^.next
         ELSE IF (Hi(key)=STAB) OR (Hi(key)=UP)
            THEN m:=m^.last
   UNTIL Lo(key)=CR;
   _S^[25]:=oldline
END; { UseMask }
```

Displays the list of strings in column in a window and allows the seizure of an element.
- list : list of strings (= ARRAY [1..nfile] OF STRING[len])
- len : length of the strings nfile
- item : returns the chosen element by ENTER
- x1, y1, nl : upper left angle and number of rows of the window
- title help : number in the help file

```
FUNCTION DisplayList(list          : StrPtr;
                     len           : BYTE;
                     nfile         : INTEGER;
                 VAR item          : INTEGER;
                     x1, y1, nl    : INTEGER;
                     title         : String;
                     help          : WORD    ) : BOOLEAN;

VAR top     : INTEGER;
    l       : StrPtr;
    p, old  : INTEGER;

    PROCEDURE DisplayLine(item    : BYTE;
                          reverse : BOOLEAN);
    BEGIN
       IF reverse

          THEN SetColor(_Drfg, _Drbg)
          ELSE SetColor(_Dwfg, _Dwbg);
       p:=item-top;
       GotoXY(1, (p MOD nl)+1);
       l:=AddAddr(list, (item-1)*len);
       Write(' ', l^);
       clreol
    END; { DisplayLine }

    PROCEDURE DisplayPage;
    VAR i : INTEGER;
    BEGIN
       old:=top;
       IF item<top THEN top:=item;
       IF top+nl<=item THEN top:=IMax(item-nl+1, 1);
       IF old<>top THEN
          FOR i:=top TO IMin(top+nl-1, nfile) DO
             DisplayLine(i, FALSE)
    END; { DisplayPage }

VAR key     : WORD;
    oldline : ScreenLine;
BEGIN { DisplayDir }
   oldline:=_S^[25];
   HelpLine(205);
   OpenWin(x1, y1, x1+len+2, y1+nl+1,
           _Dwfg, _Dwbg, _Drfg, _Drbg, _Dffg, _Dfbg, _DFrame);
   IF 0<length(title) THEN SetTitleWin(title);
   top:=maxint;
   item:=1;
   REPEAT
      DisplayPage;
      DisplayLine(item, TRUE);
      key:=Inkey;
      DisplayLine(item, FALSE);
      CASE Hi(key) OF
         UP, LEFT    : IF 1<item THEN Dec(item);
         DOWN, RGHT  : IF item<nfile THEN Inc(item);
         HOME        : item:=1;
         ENDK        : item:=nfile;
         PGUP        : IF 0<item-nl+1
                            THEN Dec(item, nl-1)
                            ELSE item:=1;
         PGDN        : IF item+nl-1<nfile
                            THEN Inc(item, nl-1)
                            ELSE item:=nfile;
         F1          : DisplayHelp(help)
      END
   UNTIL Lo(key) IN [CR, ESC];
   CloseWin;
   _S^[25]:=oldline;
   DisplayList:=Lo(key)=CR
END; { DisplayList }
```

## Directory window

```
{$F+} FUNCTION LessEntry(x, y : INTEGER) : BOOLEAN; {$F-}
BEGIN
   IF ((_Dir^[x,1]<>'\') AND (_Dir^[y,1]<>'\')) OR
      ((_Dir^[x,1]='\') AND (_Dir^[y,1]='\'))
      THEN LessEntry:=_Dir^[x]<_Dir^[y]
      ELSE LessEntry:=_Dir^[x,1]<>'\'
END; { LessEntry }

{$F+} PROCEDURE SwapEntry(x, y : INTEGER); {$F-}
VAR z : Str15;
BEGIN
   z:=_Dir^[x];
   _Dir^[x]:=_Dir^[y];
   _Dir^[y]:=z
END; { SwapEntry }

PROCEDURE GetDirectory(Path  : PathStr;
                   VAR nfile : INTEGER);
VAR entry : SearchRec;
    f     : INTEGER;
BEGIN
   nfile:=0;
   FindFirst(Path, $21, entry);
   WHILE DosError=0 DO
      BEGIN
         Inc(nfile);
         _Dir^[nfile]:=entry.Name;
         FindNext(entry)
      END;
   Path:=Concat(PSplit(Path), '*.*');
   FindFirst(Path, Directory, entry);
   WHILE DosError=0 DO
      BEGIN
         IF (entry.Attr AND Directory=Directory) AND (entry.Name<>'.')
            THEN BEGIN
               Inc(nfile);
               IF entry.Name='..'
                  THEN _Dir^[nfile]:='\____________'
```

```
                ELSE _Dir^[nfile]:=Concat('\', entry.Name)
            END;
         FindNext(entry)
      END;
   IF 0<nfile
      THEN BEGIN
         HSort(LessEntry, SwapEntry, nfile);
         IF _Dir^[nfile]='\____________' THEN _Dir^[nfile]:='\..';
         FOR f:=1 TO nfile DO
            BEGIN
               _Dir^[f]:=Concat(' ', _Dir^[f]);
               WHILE Length(_Dir^[f])<15 DO
                  _Dir^[f]:=Concat(_Dir^[f], ' ')
            END
      END
END; { GetDirectory }

FUNCTION DisplayDir(nfile : INTEGER;
                VAR f     : INTEGER;
                    help  : WORD   ) : BOOLEAN;

VAR top : INTEGER;

    PROCEDURE Lift(f : INTEGER);
    VAR i : BYTE;
    BEGIN
       FOR i:=1 TO DV DO BEGIN
          gotoxy(15*DH+3, i);
          write(#178)
       END;
       f:=round(DV*f/nfile);
       IF f=0 THEN f:=1;
       gotoxy(15*DH+3, f);
       write(#32)
    END; { Lift }

    PROCEDURE DisplayLine(f       : INTEGER;
                          reverse : BOOLEAN);
    VAR p : INTEGER;
    BEGIN
       IF reverse
          THEN SetColor(_Drfg, _Drbg)
          ELSE SetColor(_Dwfg, _Dwbg);
       p:=f-top;
       GotoXY(15*(p DIV DV)+2, (p MOD DV)+1);
       Write(_Dir^[f])
    END; { DisplayLine }

    PROCEDURE DisplayPage;
    VAR i, old : INTEGER;
    BEGIN
       old:=top;
       IF f<top THEN top:=f;
       IF top+PAGE<=f THEN top:=IMax(f-PAGE+1, 1);
       IF old<>top THEN
          FOR i:=top TO IMin(top+PAGE-1, nfile) DO
             DisplayLine(i, FALSE)
    END; { DisplayPage }

VAR key          : WORD;
    mouse, x, y : INTEGER;
BEGIN { DisplayDir }
   top:=maxint;
   f:=1;
   REPEAT
      DisplayPage;
      DisplayLine(f, TRUE);
      Lift(f);
      key:=0;
      NullMouse;
      ShowMouse;
      REPEAT
         IF _Mouse THEN
            IF Keypressed THEN
               key:=Inkey
            ELSE BEGIN
               GetPosBut(mouse, x, y);
               x:=(x div 8)+1;
               y:=(y div 8)+1;
               IF mouse=2 THEN
                  key:=ESC
               ELSE IF mouse=1 THEN
```

```
                  IF (_W^.xmin+1<x) and (x<_W^.xmax-3) and
                     (_W^.ymin  <y) and (y<_W^.ymax  ) THEN BEGIN
                     mouse:=top+DV*((x-_W^.xmin-3) div 15)+y-_W^.ymin-1;
                     IF mouse<=nfile THEN BEGIN
                        DisplayLine(f, FALSE);
                        f:=mouse;
                        key:=CR
                     END
                  END
                  ELSE IF (_W^.xmax-3<=x) and (x<_W^.xmax) and
                          (_W^.ymin<y   ) and (y<_W^.ymax) THEN BEGIN
                       x:=round(DV*f/nfile);
                       IF x=0 THEN x:=1;
                       IF (y-_W^.xmin=1) or (y-_W^.ymin<x) THEN
                          key:=PGUP shl 8
                       ELSE IF x<y-_W^.ymin THEN
                          key:=PGDN shl 8
                  END
            END
         ELSE key:=Inkey
      UNTIL key<>0;
      HideMouse;
      DisplayLine(f, FALSE);
      CASE Hi(key) OF
         UP   : IF 1<f THEN Dec(f);
         DOWN : IF f<nfile THEN Inc(f);
         LEFT : IF top<=f-DV THEN Dec(f, DV);
         RGHT : IF f+DV<IMin(top+PAGE, nfile+1) THEN Inc(f, DV);
         HOME : f:=1;
         ENDK : f:=nfile;
         PGUP : IF 0<f-PAGE+1
                   THEN Dec(f, PAGE-1)
                   ELSE f:=1;
         PGDN : IF f+PAGE-1<nfile
                   THEN Inc(f, PAGE-1)
                   ELSE f:=nfile;
         F1   : DisplayHelp(help)
      END
   UNTIL Lo(key) IN [CR, ESC];
   DisplayDir:=Lo(key)=CR
END; { DisplayDir }

FUNCTION NewPath(VAR Path : PathStr;
                     New  : DirStr) : PathStr;
VAR dir  : DirStr;
    Name : NameStr;
    ext  : ExtStr;
BEGIN
   FSplit(Path, dir, Name, ext);
   IF New='\..'
      THEN BEGIN
         New:='';
         dir:=PSplit(dir)
      END;
   Delete(dir, Length(dir), 1);
   NewPath:=Concat(dir, New, '\', Name, ext)
END; { NewPath }

FUNCTION ShowDir(VAR Path : PathStr;
                     Help : WORD) : BOOLEAN;

VAR nfile, f : INTEGER;
    old      : PathStr;
    EndLoop  : BOOLEAN;
    oldline  : ScreenLine;

BEGIN { ShowDir }
   oldline:=_S^[25];
   HelpLine(205);
   OpenWin(_Dxmin, _Dymin, _Dxmin+15*DH+5, _Dymin+DV+1,
           _Dwfg, _Dwbg, _Drfg, _Drbg, _Dffg, _Dfbg, _DFrame);
   old:=Path;
   New(_Dir);
   EndLoop:=FALSE;
   REPEAT
      GetDirectory(Path, nfile);
      ClrScr;
      SetTitleWin(Path);
      IF nfile=0
         THEN BEGIN
            CloseWin;
            Dispose(_Dir);
            ErrorMsg(DosError);
            ShowDir:=FALSE;
            EXIT
         END
         ELSE IF DisplayDir(nfile, f, Help)
            THEN BEGIN
               Delete(_Dir^[f],1,1);
               WHILE _Dir^[f,Length(_Dir^[f])]=' ' DO
                   Delete(_Dir^[f], Length(_Dir^[f]), 1);
               IF _Dir^[f,1]='\'
                  THEN Path:=NewPath(Path, _Dir^[f])
                  ELSE BEGIN
                     Path:=Concat(PSplit(Path), _Dir^[f]);
                     EndLoop:=TRUE
                  END;
               ShowDir:=EndLoop
            END
            ELSE BEGIN
               Path:=old;
               EndLoop:=TRUE;
               ShowDir:=FALSE
            END
   UNTIL EndLoop;
   Dispose(_Dir);
   CloseWin;
   _S^[25]:=oldline
END; { ShowDir }
```

Opens a sequence of interactive windows to chose a file. Presentation of those windows is defined by constants typed _Dxxx.

- Path and name, complete or not, with or without replacement characters.
- Title of the seizure window
- Help index

Returns TRUE if the sequence is terminated by ENTER. Returns FALSE if the sequence is terminated by ESC, then path keeps unchanged.

```
FUNCTION MaskName(Path : PathStr) : PathStr;
VAR dir  : DirStr;
    Name : NameStr;
    ext  : ExtStr;
BEGIN
   FSplit(Path, dir, Name, ext);
   IF dir ='' THEN GetDir(0, dir);
   IF dir[Length(dir)]<>'\' THEN dir:=Concat(dir, '\');
   IF Name='' THEN Name:='*';
   IF ext ='' THEN ext :='.*';
   MaskName:=Concat(dir, Name, ext)
END; { MaskName }

FUNCTION GetFileName;
VAR old  : PathStr;
BEGIN
   old:=Path;
   Path:=MaskName(Path);
   IF EditString(_Dxmin, _Dymin-2, _Dxmin+40,
                 Title, @Path, SizeOf(Path)-1, PathChar, TRUE,
                 Help)
      THEN BEGIN
         Path:=MaskName(Path);
         IF (0<Pos('*', Path)) OR (0<Pos('?', Path))
            THEN IF ShowDir(Path, Help)
               THEN GetFileName:=TRUE
               ELSE BEGIN
                  Path:=old;
                  GetFileName:=FALSE
               END
            ELSE GetFileName:=TRUE
      END
      ELSE BEGIN
         Path:=old;
         GetFileName:=FALSE
      END
END; { GetFileName }
```

Opens a seizure window to edit a directory name, Path, and changes the active directory. Is ended by ENTER or ESC.

```
FUNCTION ChangeDir;
VAR old : PathStr;
    err : WORD;
BEGIN
   old:=Path;
   IF EditString(_Dxmin, _Dymin-4, _Dxmin+35,
                 'Active Directory', @Path, SizeOf(Path)-1, PathChar,
                 TRUE, Help)
      THEN BEGIN
         IF Path='' THEN GetDir(0, Path);
         IF (Path[Length(Path)]='\') AND (Path[Length(Path)-1]<>':')
            THEN Delete(Path, Length(Path), 1);
         IF Path[Length(Path)]=':' THEN Path:=Concat(Path, '\');
         {$I-} ChDir(Path); {$I+}
         err:=IOResult;
         IF err=0
            THEN ChangeDir:=TRUE
            ELSE BEGIN
               Path:=old;
               ErrorMsg(err);
               ChangeDir:=FALSE
            END
      END
      ELSE ChangeDir:=FALSE
END; { ChangeDir }
```

Menus management

```
{$F+} PROCEDURE NullAction(Action : BYTE); {$F-}
BEGIN
END; { NullAction }
```

Definition of a menu must be immediately followed by the definitions of the options
m :	index of menu, with root equal to 0
x, y :	upper left angle of the window
o :	initial option
That procedure must be immediately followed by the next which makes its content

```
PROCEDURE SetMenu;
BEGIN
   _M:=m;
   New(_Menu[m]);
   _Menu[m]^.x:=x;
   _Menu[m]^.y:=y;
   _Menu[m]^.o:=o;
   _Menu[m]^.n:=-1;
   _Menu[m]^.w:=0;
   FillChar(_Menu[m]^.Options, SizeOf(_Menu[_M]^.Options), 0)
END; { SetMenu }
```

Definition of options must follow definition of menu
option :	display, if empty a row is displayed
menu :	number of the sub-menu to be called
action :	number of the action to be returned and
	reference to help
out :	the window keeps active after execution of the
	option (Stay), or is closed (GoBack), or is closed
	with all others (GoRoot) down to the root menu
	(Root).

```
PROCEDURE SetOption;
BEGIN
   IF _Menu[_M]^.n=MAX_O
      THEN BEGIN
         ErrorMsg(250);
         HALT
      END;
   Inc(_Menu[_M]^.n);
   New(_Menu[_M]^.Options[_Menu[_M]^.n]);
   IF Length(option)=0
      THEN _Menu[_M]^.Options[_Menu[_M]^.n]^.Option:=NIL
      ELSE BEGIN
         IF _Menu[_M]^.w<Length(option)+2 THEN
            _Menu[_M]^.w:=Length(option)+2;
         GetMem(_Menu[_M]^.Options[_Menu[_M]^.n]^.Option, Length(option)+1);
         _Menu[_M]^.Options[_Menu[_M]^.n]^.Option^:=option
      END;
   _Menu[_M]^.Options[_Menu[_M]^.n]^.Menu   :=menu;
   _Menu[_M]^.Options[_Menu[_M]^.n]^.Action:=action;
   _Menu[_M]^.Options[_Menu[_M]^.n]^.Out    :=out
END; { SetOption }
```

That function seizes the place of the displayed string of an option thanks to a pointer. It allows the modification of the option, under the condition that its length does'nt exceed the length declared with SetOption.

```
FUNCTION GetOption;
BEGIN
   GetOption:=_Menu[_M]^.Options[_Menu[_M]^.n]^.Option
END; { GetOption }
```

Opens the menu m. Remark: The menus must be used with a procedure declared by _Action which should contain a CASE followed by the different options corresponding to the numbers assigned to the variable action of SetOption.

```
PROCEDURE MenuWin;

   FUNCTION Menu( m : INTEGER ) : ActionDef;
   VAR key : WORD;
      i   : BYTE;
      out : ActionDef;
      s   : ScreenStatus;

      PROCEDURE Display(Attr : MarkerType);
      BEGIN
         IF Attr=Reverse THEN SetColor(_Mrfg, _Mrbg);
         GotoXY(2, _Menu[m]^.o+1);
         Write(' ', _Menu[m]^.Options[_Menu[m]^.o]^.Option^);
         WrRepChar(WhereX, WhereY, #32,
                  _Menu[m]^.w-
                  Length(_Menu[m]^.Options[_Menu[m]^.o]^.Option^)-2);
         IF _Menu[m]^.Options[_Menu[m]^.o]^.Menu=0
            THEN Write(#32)
            ELSE Write(#16);
         SetColor(_Mwfg, _Mwbg)
      END; { Display }

   BEGIN { Menu }
      _MW:=_MW+CHR(m);
      OpenWin(_Menu[m]^.x,
            _Menu[m]^.y,
            _Menu[m]^.x+_Menu[m]^.w+3,
            _Menu[m]^.y+_Menu[m]^.n+2,
            _Mwfg, _Mwbg, _Mrfg, _Mrbg, _Mffg, _Mfbg, _MFrame);
      FOR i:=0 TO _Menu[m]^.n DO
         IF _Menu[m]^.Options[i]^.Option=NIL
            THEN WrRepChar(2, i+1, #196, _Menu[m]^.w)
            ELSE BEGIN
               GotoXY(3, i+1);
               Write(_Menu[m]^.Options[i]^.Option^);
               IF 0<_Menu[m]^.Options[i]^.Menu
                  THEN BEGIN
                     GotoXY(_Menu[m]^.w+1, i+1);
                     Write(#16)
                  END
            END;
      IF m=0
         THEN HelpLine(202)
         ELSE HelpLine(203);
      out:=Stay;
      REPEAT
         Display(Reverse);
```

```
        IF 0<length(_PutMacro)
           THEN key:=Ord(_PutMacro[1])
           ELSE BEGIN
              _MM:=TRUE;
              key:=Inkey;
              _MM:=FALSE
           END;
        Display(Normal);
        CASE Hi(key) OF
           HOME, PGUP : _Menu[m]^.o:=0;
           ENDK, PGDN : _Menu[m]^.o:=_Menu[m]^.n;
           UP   , LEFT : REPEAT _Menu[m]^.o:=(_Menu[m]^.o+_Menu[m]^.n)
                                           MOD (_Menu[m]^.n+1)
                       UNTIL _Menu[m]^.Options[_Menu[m]^.o]^.Option<>NIL;
          DOWN, RGHT : REPEAT _Menu[m]^.o:=(_Menu[m]^.o+1) MOD (_Menu[m]^.n+1)
                       UNTIL _Menu[m]^.Options[_Menu[m]^.o]^.Option<>NIL;
           F1         : BEGIN
                          Display(Reverse);
                          DisplayHelp(_Menu[m]^.Options[_Menu[m]^.o]^.Action)
                        END
        END;
        IF ($20<Lo(key)) AND (Lo(key)<$FF)
           THEN BEGIN
              i:=0;
              WHILE (i<=_Menu[m]^.n) AND
                    ((_Menu[m]^.Options[i]^.Option=NIL) OR
                    (UpCase(Chr(key))<>_Menu[m]^.Options[i]^.Option^[1])) DO
                 Inc(i);
              IF i<=_Menu[m]^.n
                 THEN BEGIN
                    _Menu[m]^.o:=i;
                    key:=CR
                 END
           END;
        IF 0<Length(_PutMacro)
           THEN BEGIN
              IF Length(_PutMacro)=1 THEN key:=0;
              Delete(_PutMacro, 1, 1)
           END;
        IF key=CR
           THEN BEGIN
              _GetMacro:=Concat(_GetMacro,
                        _Menu[m]^.Options[_Menu[m]^.o]^.Option^[1]);
              IF _Menu[m]^.Options[_Menu[m]^.o]^.Menu=0
                 THEN BEGIN
                    Display(Reverse);
                    GetScreenAspect(s);
                    _Action(_Menu[m]^.Options[_Menu[m]^.o]^.Action);
                    SetScreenAspect(s);
                    out:=_Menu[m]^.Options[_Menu[m]^.o]^.Out
                 END
                 ELSE BEGIN
                    Display(Reverse);
                    out:=Menu(_Menu[m]^.Options[_Menu[m]^.o]^.Menu);
                    IF (out=GoBack) OR ((out=GoRoot) AND
                       (_Menu[m]^.Options[_Menu[m]^.o]^.Out=Root))
                       THEN out:=Stay
                 END;
              IF m=0
                 THEN HelpLine(202)
                 ELSE HelpLine(203)
           END
           ELSE IF (key=ESC) AND (0<m)
              THEN BEGIN
                 out:=GoBack;
                 Delete(_GetMacro, Length(_GetMacro), 1)
              END
      UNTIL out IN [GoBack, GoRoot];
      CloseWin;
      delete(_MW, length(_MW), 1);
      Menu:=out
   END; { Menu }

BEGIN { MenuWin }
   IF Menu(m)=Root THEN
END; { MenuWin }
```

## Library management

Put color on the screen by means of the constant typed ScreenGround. Displays the title in position x, y with the color defined by _Tfg and _Tbg. To be used when initializing the Program.

```
PROCEDURE SetBackScreen;
BEGIN
   FillBox(1, 1, 80, 25, ScreenGround[_C]);
   SetColor(_Tfg, _Tbg);
   GotoXY(x, y);
   Write(title)
END; { SetBackScreen }

BEGIN
   CheckSnow:=FALSE;
   CheckBreak:=FALSE;
   GetVideo;
   SetCsr(Off);
   FillChar(_HelpMsg, SizeOf(_HelpMsg), 0);
   FillChar(_Menu, SizeOf(_Menu), 0);
   _Action:=NullAction;
   _Mouse:=InitMouse
END.
```

## *Library GAUGEWIN*

Function :   barre de progression d'un tâche

```
UNIT GaugeWin;

INTERFACE

USES Crt, Tools, TP_Win_E;

PROCEDURE OpenGauge(x, y : BYTE);
PROCEDURE InitGauge(sum   : LONGINT);
PROCEDURE Gauge(r : LONGINT);

IMPLEMENTATION

VAR
   _Gstatus  : ScreenStatus;
   _Gauge    : BYTE;
   _Rate     : LONGINT;
```

Insertion of the gauge in position x, y of the screen

```
PROCEDURE OpenGauge(x, y : BYTE);
VAR scr : ScreenStatus;
BEGIN
   GetScreenAspect(scr);
   OpenWin(x-3, y-1, x+54, y+2,
           _Mwfg, _Mwbg, _Mrfg, _Mrbg, _Mffg, _Mfbg, _MFrame);
   GotoXY(3, 1);
   Write('0       20        40        60        80        100 %');
   WrRepChar(3, 2, 'ù', 50);
   GetScreenAspect(_Gstatus);
   SetScreenAspect(scr)
END; { OpenGauge }
```

Initializing the gauge window. Upper left angle given by <x, y> and normalization values given by <sum>. Must be closed by CloseWin if on top of the pile.

```
PROCEDURE InitGauge(sum : LONGINT);
VAR scr : ScreenStatus;
BEGIN
   GetScreenAspect(scr);
   SetScreenAspect(_Gstatus);
   TextColor(_Mwfg[_C]);
   WrRepChar(3, 2, 'ù', 50);
   TextColor(_Mffg[_C]);
   GetScreenAspect(_Gstatus);
   SetScreenAspect(scr);
   _Gauge:=3;
   _Rate:=sum
END; { InitGauge }
```

Fills the gauge in function of the value <r> which must be smaller than or equal to <sum> of InitGauge.

```
PROCEDURE Gauge(r : LONGINT);
VAR scr : ScreenStatus;
    p   : BYTE;
BEGIN
   GetScreenAspect(scr);
   SetScreenAspect(_Gstatus);
   p:=Round(100*(r/_Rate));
   if 100<p then begin
      p:=100;
      gotoxy(4, 2);
      write(' overflow ! ')
   end;
   WrRepChar(_Gauge, 2, 'Û', (p DIV 2)-_Gauge+3);
   IF Odd(p) THEN Write('_');
   _Gauge:=(p DIV 2)+3;
   SetScreenAspect(scr)
END; { Gauge }

END.
```

## Library INITGRA

Function :   initializing the graphic mode

```
UNIT INITGRA;

{$A+,B-,D-,E+,F-,G+,I-,L-,N-,O-,R-,S+,V-,X+}

INTERFACE

USES Mouse, Video, Graph;

CONST
   _Background : ScreenColor = (Black, Blue);

PROCEDURE InitGraphicMode;

IMPLEMENTATION

PROCEDURE EGAVGADriverProc; EXTERNAL;
{$L C:\TP\LIB\EGAVGA.OBJ }

PROCEDURE InitGraphicMode;
VAR driver, mode, error : INTEGER;
BEGIN
   DetectGraph(driver, mode);
   IF RegisterBGIDriver(@EGAVGADriverProc)<0 THEN HALT(248);
   driver:=Detect;
   InitGraph(driver, mode, '');
   error:=GraphResult;
   IF error<>GrOK THEN HALT(248);
   IF NOT InitMouse THEN BEGIN
      CloseGraph;
      HALT(249)
   END;
   SetBkColor(_BackGround[_C]);
   SetGraphicCursor(Mouse.Arrow);
   SetMouseField(0, 0, GetMaxX, GetMaxY);
   SetCursorPos(GetMaxX DIV 2, GetMaxY DIV 2);
   ShowMouse
END; { InitGraphicMode }

END.
```

## *Library GRTOOLS*

Function : Library of graphic masks

```
UNIT GrTools;

INTERFACE

USES Math;

TYPE
```

Coordinates (upper left angle and lower right) of a rectangle.

```
    RecType      = RECORD
                     x1, y1, x2, y2 : INTEGER
                   END;
```

Liste of rectangle.

```
    RecArray     = ARRAY[1..8191] OF RecType;
    RecArrayPtr  = ^RecArray;
```

Procedure of drawing a line (eg Graph.Line ou HP.Line).

```
    LineProc     = PROCEDURE(x1, y1, x2, y2 : INTEGER);

PROCEDURE MultiRecMask(x1, y1, x2, y2 : INTEGER;
                       n              : WORD;
                       rec            : RecArrayPtr;
                       line           : LineProc);

IMPLEMENTATION
```

| | | |
|---|---|---|
| INPUT | x1, y1, x2, y2 : | line |
| | rx1, ry1, rx2, ry2 : | rectangle |
| OUTPUT | ax1, ay1, ax2, ay2 : | first segment |
| | bx1, by1, bx2, by2 : | second segment |
| RETURN | | number of segments |

```
FUNCTION RecMask( x1,  y1,  x2,  y2,
                 rx1, ry1, rx2, ry2 : INTEGER;
             VAR ax1, ay1, ax2, ay2,
                 bx1, by1, bx2, by2 : INTEGER)
                 : BYTE;

   PROCEDURE XSwap;
   BEGIN
      IF x2<x1 THEN BEGIN
         IntSwap(x1, x2);
         IntSwap(y1, y2)
      END
   END; { XSwap }

   PROCEDURE YSwap;
   BEGIN
      IF y2<y1 THEN BEGIN
         IntSwap(x1, x2);
         IntSwap(y1, y2)
      END
   END; { YSwap }

TYPE
   EdgeCutType   = (Te, Be, Le, Re);
   EdgeCutSet    = SET OF EdgeCutType;
   CornerCutType = (TL, TR, BR, BL);
   CornerCutSet  = SET OF CornerCutType;

VAR
   DX, dy, a, b, c, d : INTEGER;
   m, h               : REAL;
   edge               : EdgeCutSet;
   corner             : CornerCutSet;
   Seg                : BYTE;
BEGIN
   IF rx2<rx1 THEN IntSwap(rx1, rx2);
IF ry2<ry1 THEN IntSwap(ry1, ry2);
IF (rx1<=x1) AND (x1<=rx2) AND (rx1<=x2) AND (x2<=rx2) AND
   (ry1<=y1) AND (y1<=ry2) AND (ry1<=y2) AND (y2<=ry2) THEN BEGIN
   RecMask:=0;
   EXIT
END;
DX:=x2-x1;
dy:=y2-y1;
```

Vertical

```
IF DX=0 THEN BEGIN
   IF  y2< y1 THEN IntSwap( y1,  y2);
   ax1:=x1;
   ax2:=x2;
   IF (x1<rx1) OR (rx2<x2) OR (y2<ry1) OR (ry2<y1) THEN BEGIN
      ay1:=y1;
      ay2:=y2;
      Seg:=1
   END
   ELSE IF y2<=ry2 THEN BEGIN
      ay1:= y1;
      ay2:=ry1;
      Seg:=1
   END
   ELSE IF ry1<=y1 THEN BEGIN
      ay1:=ry2;
      ay2:= y2;
      Seg:=1
   END
   ELSE BEGIN
      ay1:= y1;
      ay2:=ry1;
      bx1:= x1;
      by1:=ry2;
      bx2:= x2;
      by2:= y2;
      Seg:=2
   END
END
```

Horizontal

```
ELSE IF dy=0 THEN BEGIN
   IF  x2< x1 THEN IntSwap( x1,  x2);
   ay1:=y1;
   ay2:=y2;
   IF (y1<ry1) OR (ry2<y2) OR (x2<rx1) OR (rx2<x1) THEN BEGIN
      ax1:=x1;
      ax2:=x2;
      Seg:=1
   END
   ELSE IF x2<=rx2 THEN BEGIN
      ax1:= x1;
      ax2:=rx1;
      Seg:=1
   END
   ELSE IF rx1<=x1 THEN BEGIN
      ax1:=rx2;
      ax2:= x2;
      Seg:=1
   END
   ELSE BEGIN
      ax1:= x1;
      ax2:=rx1;
      bx1:=rx2;
      by1:= y1;
      bx2:= x2;
      by2:= y2;
      Seg:=2
   END
END
```

Diagonal

```
ELSE BEGIN
   m:=dy/DX;
   h:=y1-x1*m;
   a:=Round((ry1-h)/m);
   b:=Round((ry2-h)/m);
```

```
c:=Round(m*rx1+h);
d:=Round(m*rx2+h);
edge:=[];
YSwap;
IF (rx1<=a) AND (a<=rx2) AND
   (y1<=ry1) AND (ry1<=y2) THEN edge:=[Te];
IF (rx1<=b) AND (b<=rx2) AND
   (y1<=ry2) AND (ry2<=y2) THEN edge:=edge+[Be];
XSwap;
IF (ry1<=c) AND (c<=ry2) AND
   (x1<=rx1) AND (rx1<=x2) THEN edge:=edge+[Le];
IF (ry1<=d) AND (d<=ry2) AND
   (x1<=rx2) AND (rx2<=x2) THEN edge:=edge+[Re];
corner:=[];
IF (c=ry1) AND (x1<=rx1) AND (rx1<=x2) THEN corner:=corner+[TL];
IF (c=ry2) AND (x1<=rx1) AND (rx1<=x2) THEN corner:=corner+[BL];
IF (d=ry1) AND (x1<=rx2) AND (rx2<=x2) THEN corner:=corner+[TR];
IF (d=ry2) AND (x1<=rx2) AND (rx2<=x2) THEN corner:=corner+[BR];
YSwap;
IF (a=rx1) AND (y1<=ry1) AND (ry1<=y2) THEN corner:=corner+[TL];
IF (a=rx2) AND (y1<=ry1) AND (ry1<=y2) THEN corner:=corner+[TR];
IF (b=rx1) AND (y1<=ry2) AND (ry2<=y2) THEN corner:=corner+[BL];
IF (b=rx2) AND (y1<=ry2) AND (ry2<=y2) THEN corner:=corner+[BR];
IF TL IN corner THEN BEGIN
   edge:=edge+[Te, Le];
   IF edge-[Te, Le]<>[] THEN
      edge:=edge-[Te]
   ELSE IF x2<=rx1 THEN
      edge:=[]
END;
IF TR IN corner THEN BEGIN
   edge:=edge+[Te, Re];
   IF edge-[Te, Re]<>[] THEN
      edge:=edge-[Te]
   ELSE IF rx2<=x2 THEN
      edge:=[]
END;
IF BL IN corner THEN BEGIN
   edge:=edge+[Be, Le];
   IF edge-[Be, Le]<>[] THEN
      edge:=edge-[Be]
   ELSE IF x1<=rx1 THEN
      edge:=[]
END;
IF BR IN corner THEN BEGIN
   edge:=edge+[Be, Re];
   IF edge-[Be, Re]<>[] THEN
      edge:=edge-[Be]
   ELSE IF rx2<=x1 THEN
      edge:=[]
END;
Seg:=0;
IF Te IN edge THEN BEGIN
   ax1:=x1;
   ay1:=y1;
   ax2:=a;
   ay2:=ry1;
   Seg:=1
END;
IF Be IN edge THEN BEGIN
   IF Seg=0 THEN BEGIN
      ax1:=b;
      ay1:=ry2;
      ax2:=x2;
      ay2:=y2
   END
   ELSE IF Seg=1 THEN BEGIN
      bx1:=b;
      by1:=ry2;
      bx2:=x2;
      by2:=y2
   END;
   Inc(Seg)
END;
XSwap;
IF Le IN edge THEN BEGIN
   IF Seg=0 THEN BEGIN
      ax1:=x1;
      ay1:=y1;
      ax2:=rx1;
      ay2:=c
   END
```

```
         ELSE IF Seg=1 THEN BEGIN
            bx1:=x1;
            by1:=y1;
            bx2:=rx1;
            by2:=c
         END;
         Inc(Seg)
      END;
      IF Re IN edge THEN BEGIN
        IF Seg=0 THEN BEGIN
           ax1:=rx2;
           ay1:=d;
           ax2:=x2;
           ay2:=y2
        END
        ELSE IF Seg=1 THEN BEGIN
           bx1:=rx2;
           by1:=d;
           bx2:=x2;
           by2:=y2
        END;
        Inc(Seg)
      END;
      IF Seg=0 THEN BEGIN
         ax1:=x1;
         ay1:=y1;
         ax2:=x2;
         ay2:=y2;
         Seg:=1
      END
      ELSE IF 2<Seg THEN
         Seg:=2
   END;
   RecMask:=Seg
END; { RecMask }
```

Draws with the procedure <line> the segments of line <x1, y1, x2, y2> which are not masked by the rectangles defined in the list <rec>. Rectangles are numbered 1 toi <n>.

```
PROCEDURE MultiRecMask(x1, y1, x2, y2 : INTEGER;
                       n              : WORD;
                       rec            : RecArrayPtr;
                       line           : LineProc);

   PROCEDURE Mask(VAR x1, y1, x2, y2 : INTEGER;
                      i              : WORD);
   VAR ax1, ay1, ax2, ay2,
       bx1, by1, bx2, by2 : INTEGER;
       Seg                : BYTE;
   BEGIN
      IF n<i THEN
         Line(x1, y1, x2, y2)
      ELSE CASE RecMask(x1, y1, x2, y2,
                        rec^[i].x1, rec^[i].y1, rec^[i].x2, rec^[i].y2,
                        ax1, ay1, ax2, ay2,
                        bx1, by1, bx2, by2) OF
         1 : Mask(ax1, ay1, ax2, ay2, i+1);
         2 : BEGIN
                Mask(ax1, ay1, ax2, ay2, i+1);
                Mask(bx1, by1, bx2, by2, i+1)
             END
      END
   END; { Mask }

BEGIN
   Mask(x1, y1, x2, y2, 1)
END; { MultiRecMask }

END.
```

## *Library BUTTON*

Function : Management of the buttons in graphic mode

```
UNIT Button;

INTERFACE

USES Mouse, Graph, Tools, Video;

CONST
    _Frame  : ScreenColor = (LightGray, LightCyan);
    _Fill   : ScreenColor = (LightGray, Cyan     );
    _Shadow : ScreenColor = (Black    , DarkGray );
    _Text   : ScreenColor = (Black    , White    );

TYPE
```

Pointer on the procedure managing the actions associated to the differents buttons.

```
   ActionType = PROCEDURE(action : BYTE);

   MouseProc1 = PROCEDURE(x, y : INTEGER);
   MouseProc2 = PROCEDURE(m, x, y : INTEGER);
```

Dimension (absolute coordinates) of the button, displayed name, number of reference and if it must or not keep pushed

```
   ButtonType = RECORD
                   xmin, ymin, xmax, ymax : INTEGER;
                   Name                   : str64;
                   action                 : BYTE;
                   stay                   : BOOLEAN
                END;
```

Definition of a panel

```
   ButtonList = ARRAY[1..255] OF ButtonType;
   ButtonPtr  = RECORD
                   s, n    : BYTE;
                   action  : ActionType;
                   events1 : MouseProc1;
                   events2 : MouseProc2;
                   view    : ViewPortType;
                   b       : ^ButtonList
                END;

PROCEDURE CreatePanel(VAR Panel   : ButtonPtr;
                          Nb      : BYTE;
                          action  : ActionType;
                          events1 : MouseProc1;
                          events2 : MouseProc2);
PROCEDURE CreateButton(VAR Panel            : ButtonPtr;
                           xmin, ymin, xmax : INTEGER;
                           Name             : Str64;
                           action           : BYTE;
                           stay             : BOOLEAN);
PROCEDURE ClearButton(VAR Panel         : ButtonPtr;
                          action, color : BYTE);
PROCEDURE ClearPanel(VAR Panel : ButtonPtr;
                         color : BYTE);
PROCEDURE PressButton(Panel  : ButtonPtr;
                      button : BYTE);
PROCEDURE ActivePanel(Panel : ButtonPtr);
PROCEDURE NoEvents1(x, y : INTEGER);
PROCEDURE NoEvents2(m, x, y : INTEGER);

IMPLEMENTATION

VAR
   V : ViewPortType;

PROCEDURE SetWindow;
BEGIN
   SetViewPort(V.x1, V.y1, V.x2, V.y2, V.Clip);
   ShowMouse
END; { SetWindow }

PROCEDURE SetScreen(Panel : ButtonPtr);
BEGIN
   GetViewSettings(V);
   SetViewPort(Panel.view.x1, Panel.view.y1,
               Panel.view.x2, Panel.view.y2, Panel.view.Clip);
   HideMouse
END; { SetScreen }
```

Suppression of the button referenced by <action> from the panel, <color> is the color of erasing

```
PROCEDURE ClearButton(VAR Panel         : ButtonPtr;
                          action, color : BYTE);
VAR b : BYTE;
BEGIN
   b:=1;
   WHILE (b<=Panel.n) AND (action<>Panel.b^[b].action) DO Inc(b);
   IF Panel.n<b THEN EXIT;
   SetScreen(Panel);
   SetColor(color);
   SetLineStyle(SolidLn, 0, ThickWidth);
   Rectangle(Panel.b^[b].xmin, Panel.b^[b].ymin,
             Panel.b^[b].xmax, Panel.b^[b].ymax);
   SetLineStyle(SolidLn, 0, NormWidth);
   SetFillStyle(SolidFill, color);
   FloodFill((Panel.b^[b].xmin+Panel.b^[b].xmax) DIV 2,
             (Panel.b^[b].ymin+Panel.b^[b].ymax) DIV 2, color);
   IF b<Panel.n THEN
      Move(Panel.b^[b+1], Panel.b^[b],
           (Panel.n-b)*SizeOf(ButtonType));
   Dec(Panel.n);
   SetWindow
END; { ClearButton }
```

Suppression of a board from the screen.

```
PROCEDURE ClearPanel(VAR Panel : ButtonPtr;
                         color : BYTE);
VAR b : BYTE;
BEGIN
   SetScreen(Panel);
   SetColor(color);
   SetLineStyle(SolidLn, 0, ThickWidth);
   FOR b:=1 TO Panel.n DO
      BEGIN
         Rectangle(Panel.b^[b].xmin, Panel.b^[b].ymin,
                   Panel.b^[b].xmax, Panel.b^[b].ymax);
         SetFillStyle(SolidFill, color);
         FloodFill((Panel.b^[b].xmin+Panel.b^[b].xmax) DIV 2,
                   (Panel.b^[b].ymin+Panel.b^[b].ymax) DIV 2, color)
      END;
   SetLineStyle(SolidLn, 0, NormWidth);
   FreeMem(Panel.b, Panel.s*SizeOf(ButtonType));
   Panel.s:=0;
   Panel.n:=0;
   Panel.b:=NIL;
   SetWindow
END; { ClearPanel }

PROCEDURE SetButton(Panel : ButtonPtr;
                    b     : BYTE);
BEGIN
   SetScreen(Panel);
   SetColor(_Frame[_C]);
   SetLineStyle(SolidLn, 0, NormWidth);
   Rectangle(Panel.b^[b].xmin, Panel.b^[b].ymin,
             Panel.b^[b].xmax, Panel.b^[b].ymax);
   Rectangle(Panel.b^[b].xmin+1, Panel.b^[b].ymin+1,
             Panel.b^[b].xmax-1, Panel.b^[b].ymax-1);
   SetFillStyle(SolidFill, _Fill[_C]);
   FloodFill((Panel.b^[b].xmin+Panel.b^[b].xmax) DIV 2,
             (Panel.b^[b].ymin+Panel.b^[b].ymax) DIV 2, _Frame[_C]);
   SetColor(_Shadow[_C]);
   SetLineStyle(SolidLn, 0, ThickWidth);
   Line(Panel.b^[b].xmin, Panel.b^[b].ymax,
        Panel.b^[b].xmax, Panel.b^[b].ymax);
   Line(Panel.b^[b].xmax, Panel.b^[b].ymin,
        Panel.b^[b].xmax, Panel.b^[b].ymax);
   SetLineStyle(SolidLn, 0, NormWidth);
   SetColor(_Text[_C]);
   SetTextStyle(DefaultFont, HorizDir, 1);
   SetTextJustify(CenterText, CenterText);
   OutTextXY((Panel.b^[b].xmin+Panel.b^[b].xmax) DIV 2,
             (Panel.b^[b].ymin+Panel.b^[b].ymax) DIV 2, Panel.b^[b].Name);
   SetWindow
```

```
END; { SetButton }

PROCEDURE PushButton(Panel  : ButtonPtr;
                     b      : BYTE);
VAR p : POINTER;
BEGIN
   GetMem(p, ImageSize(Panel.b^[b].xmin, Panel.b^[b].ymin,
                       Panel.b^[b].xmax, Panel.b^[b].ymax));
   SetScreen(Panel);
   GetImage(Panel.b^[b].xmin, Panel.b^[b].ymin,
            Panel.b^[b].xmax, Panel.b^[b].ymax, p^);
   PutImage(Panel.b^[b].xmin, Panel.b^[b].ymin, p^, NOTPut);
   SetWindow;
   FreeMem(p, ImageSize(Panel.b^[b].xmin, Panel.b^[b].ymin,
                        Panel.b^[b].xmax, Panel.b^[b].ymax))
END; { PushButton }
```

Push or pull the button of <Panel>

```
PROCEDURE PressButton(Panel   : ButtonPtr;
                      button  : BYTE);
VAR b : BYTE;
BEGIN
   b:=1;
   WHILE (b<=Panel.n) AND (button<>Panel.b^[b].action) DO Inc(b);
   IF Panel.n<b THEN EXIT;
   PushButton(Panel, b)
END; { PressButton }
```

Creating a new panel.
Panel : reference to panel
Nb : number of buttons (between 1 and 255)
action : management of the actions of the buttons
events1 : procedure managing the actions when the left button is pushed alone
events2 : procedure managing all othe events

```
PROCEDURE CreatePanel(VAR Panel    : ButtonPtr;
                          Nb       : BYTE;
                          action   : ActionType;
                          events1  : MouseProc1;
                          events2  : MouseProc2);
BEGIN
   IF MaxAvail<SizeOf(ButtonType)*Nb
      THEN BEGIN
         Panel.b:=NIL;
         Panel.s:=0
      END
      ELSE BEGIN
         GetMem(Panel.b, SizeOf(ButtonType)*Nb);
         Panel.s:=Nb;
         Panel.action:=action;
         Panel.events1:=events1;
         Panel.events2:=events2;
         GetViewSettings(Panel.view)
      END;
   Panel.n:=0;
END; { CreatePanel }
```

Creating a buton in the <Panel>.
xmin, ymin, xmax : dimensions (ymax:=ymin+20)
name : name of the button
action : reference to the events, 0 generates the exit of theActivePanel
stay : TRUE, the button keeps pushed otherwise FALSE.

```
PROCEDURE CreateButton(VAR Panel              : ButtonPtr;
                           xmin, ymin, xmax   : INTEGER;
                           Name               : Str64;
                           action             : BYTE;
                           stay               : BOOLEAN);
BEGIN
   IF Panel.n=Panel.s THEN EXIT;
   Inc(Panel.n);
   Panel.b^[Panel.n].xmin:=xmin;
   Panel.b^[Panel.n].ymin:=ymin;
   Panel.b^[Panel.n].xmax:=xmax;
   Panel.b^[Panel.n].ymax:=ymin+20;
   Panel.b^[Panel.n].Name:=Name;
   Panel.b^[Panel.n].action:=action;
   Panel.b^[Panel.n].stay:=stay;
   SetButton(Panel, Panel.n)
END; { CreateButton }
```

## Activation of a panel of buttons

```
PROCEDURE ActivePanel(Panel         : ButtonPtr);
VAR mouse, oldmouse, x, y : INTEGER;
    button, old           : BYTE;
    ok                    : BOOLEAN;
BEGIN
   oldmouse:=0;
   REPEAT
      ok:=FALSE;
      GetPosBut(mouse, x, y);
      IF (mouse=1) AND (oldmouse<>mouse)
         THEN BEGIN
            button:=0;
            REPEAT Inc(button)
            UNTIL (Panel.n<button) OR
                  (
                  (Panel.b^[button].xmin<x) AND
                  (Panel.b^[button].ymin<y) AND
                  (x<Panel.b^[button].xmax) AND
                  (y<Panel.b^[button].ymax)
                  );
            ok:=button<=Panel.n;
            IF ok
               THEN BEGIN
                  PushButton(Panel, button);
                  old:=Panel.n;
                  Panel.action(Panel.b^[button].action);
                  IF (old=Panel.n) AND NOT Panel.b^[button].Stay
                     THEN PushButton(Panel, button)
                     ELSE ok:=FALSE
               END
               ELSE Panel.events1(x, y)
         END
         ELSE Panel.events2(button, x, y);
      oldmouse:=mouse
   UNTIL ok AND (Panel.b^[button].action=0)
END; { ActivePanel }

{$F+} PROCEDURE NoEvents1(x, y : INTEGER); {$F-}
BEGIN
END; { NoEvents }

{$F+} PROCEDURE NoEvents2(m, x, y : INTEGER); {$F-}
BEGIN
END; { NoEvents }

BEGIN
   IF InitMouse
      THEN ShowMouse
      ELSE BEGIN
         Writeln('Mouse not found');
         HALT(249)
      END
END.
```

## *Library GRWIN*

Function :  Management of windows in graphic mode

```
UNIT GRWIN;

INTERFACE

USES Video, Tools, Graph, Dos, Mouse, Crt, Error;
```

```
TYPE
```

Type of variable allowing to save all the parameters in graphic mode.

```
   StatusType = RECORD
                   c : WORD;
                   v : ViewPortType;
                   t : TextSettingsType;
                   l : LineSettingsType;
                   f : FillSettingsType
                END;

PROCEDURE GetStatus(VAR s : StatusType);
PROCEDURE SetStatus(    s : StatusType);

FUNCTION fx(x : BYTE) : INTEGER;
FUNCTION fy(y : BYTE) : INTEGER;
PROCEDURE OutXY(x, y : BYTE;
                t    : STRING);

PROCEDURE SetWinColors(frame, light, shadow, foreground, background : WORD);
PROCEDURE OpenGrWin(x, y, nx, ny : INTEGER);
PROCEDURE CloseGrWin;

PROCEDURE OpenMsgGrWin(m : STRING);
PROCEDURE MsgGrWin(m : STRING);
FUNCTION EditGrWin(x, y    : INTEGER;
                   title   : STRING;
                   s       : StrPtr;
                   l       : BYTE;
                   mask    : AnyChar;
                   upchar  : BOOLEAN) : BOOLEAN;
FUNCTION GetGrFileName(path  : PathStr;
                   VAR New   : PathStr;
                       title : STRING) : BOOLEAN;

IMPLEMENTATION

TYPE
   PointerType = RECORD
                    p : POINTER;
                    s : WORD
                 END;

   DirArray = ARRAY[0..512] OF STRING[13];


CONST
   _np : BYTE = 0;

   _c0 : ScreenColor = (Black    , Green        ); { Fond }
   _c1 : ScreenColor = (LightGray, Cyan         ); { Cadre }
   _c2 : ScreenColor = (LightGray, DarkGray     ); { Bord dans l'ombre }
   _c3 : ScreenColor = (LightGray, LightCyan    ); { Bord illumine }
   _c4 : ScreenColor = (LightGray, White        ); { Texte }

   PathChar : AnyChar = ['!'..'*',','..';','=','?'..'{','}','~','Ç'..'_'];

VAR
   _dx, _dy, _dxw, _dyw : INTEGER;
   _s                   : StatusType;
   _plist               : ARRAY[1..5] OF PointerType;
   Dir                  : ^DirArray;
```

## Directory

```
{$F+} FUNCTION LessEntry(x, y : INTEGER) : BOOLEAN; {$F-}
BEGIN
   IF ((Dir^[x,1]<>'\') AND (Dir^[y,1]<>'\')) OR
      ((Dir^[x,1]='\') AND (Dir^[y,1]='\'))
      THEN LessEntry:=Dir^[x]<Dir^[y]
      ELSE LessEntry:=Dir^[x,1]<>'\'
END; { LessEntry }

{$F+} PROCEDURE SwapEntry(x, y : INTEGER); {$F-}
VAR z : Str15;
BEGIN
   z:=Dir^[x];
   Dir^[x]:=Dir^[y];
   Dir^[y]:=z
END; { SwapEntry }

PROCEDURE GetDirectory(Path   : PathStr;
                   VAR nfile : INTEGER);
VAR entry : SearchRec;
    f     : INTEGER;
BEGIN
   nfile:=0;
   FindFirst(Path, $21, entry);
   WHILE DosError=0 DO
      BEGIN
         Inc(nfile);
         Dir^[nfile]:=entry.Name;
         FindNext(entry)
      END;
   Path:=Concat(PSplit(Path), '*.*');
   FindFirst(Path, Directory, entry);
   WHILE DosError=0 DO
      BEGIN
         IF (entry.Attr AND Directory=Directory) AND (entry.Name<>'.')
            THEN BEGIN
               Inc(nfile);
               IF entry.Name='..'
                  THEN Dir^[nfile]:='\____________'
                  ELSE Dir^[nfile]:=Concat('\', entry.Name)
            END;
         FindNext(entry)
      END;
   IF 0<nfile
      THEN BEGIN
         HSort(LessEntry, SwapEntry, nfile);
         IF Dir^[nfile]='\____________' THEN Dir^[nfile]:='\..'
      END
END; { GetDirectory }

FUNCTION NewPath(VAR Path : PathStr;
                     New  : DirStr) : PathStr;
VAR dir  : DirStr;
    Name : NameStr;
    ext  : ExtStr;
BEGIN
   FSplit(Path, dir, Name, ext);
   IF New='\..'
      THEN BEGIN
         New:='';
         dir:=PSplit(dir)
      END;
   Delete(dir, Length(dir), 1);
   NewPath:=Concat(dir, New, '\', Name, ext)
END; { NewPath }
```

## Tools fo windows management

## Recording the graphic parameters

```
PROCEDURE GetStatus(VAR s : StatusType);
BEGIN
   s.c:=GetColor;
   GetViewSettings(s.v);
   GetTextSettings(s.t);
   GetLineSettings(s.l);
   GetFillSettings(s.f)
END; { GetStatus }
```

## Restore the graphic parameters

```
PROCEDURE SetStatus(s : StatusType);
BEGIN
   SetColor(s.c);
   SetViewPort(s.v.x1, s.v.y1, s.v.x2, s.v.y2, s.v.clip);
   SetTextStyle(s.t.font, s.t.direction, s.t.charsize);
   SetTextJustify(s.t.horiz, s.t.vert);
   SetLineStyle(s.l.linestyle, s.l.pattern, s.l.thickness);
   SetFillStyle(s.f.pattern, s.f.color)
END; { SetStatus }

FUNCTION GetPointer(s : WORD) : POINTER;
```

```
BEGIN
   Inc(_np);
   _plist[_np].s:=s;
   GetMem(_plist[_np].p, _plist[_np].s);
   GetPointer:=_plist[_np].p
END; { GetPointer }

PROCEDURE FreePointer(p : POINTER);
VAR i : BYTE;
BEGIN
   IF _np=0 THEN EXIT;
   i:=1;
   WHILE (i<_np) AND (_plist[i].p<>p) DO Inc(i);
   IF _plist[i].p=p THEN
      BEGIN
         FreeMem(_plist[i].p, _plist[i].s);
         Move(_plist[i+1], _plist[i], (_np-i)*SizeOf(PointerType));
         Dec(_np)
      END
END; { FreePointer }

PROCEDURE FreeAllPointers;
VAR i : BYTE;
BEGIN
   FOR i:=1 TO _np DO
      FreeMem(_plist[i].p, _plist[i].s);
   _np:=0
END; { FreeAllPointers }

PROCEDURE SetCharSize;
BEGIN
   SetTextStyle(DefaultFont, HorizDir, 1);
   SetTextJustify(LeftText, TopText);
   _dx:=TextWidth('1');
   _dy:=Round(6*TextWidth('1')/5);
END; { SetCharSize }
```

## Converting the coordinates of characters into coordinates of the current window

```
FUNCTION fx(x : BYTE) : INTEGER;
BEGIN
   fx:=(x-1)*_dx
END; { fx }

FUNCTION fy(y : BYTE) : INTEGER;
BEGIN
   fy:=(y-1)*_dy
END; { fy }
```

## Writing text while erasing what is already in that place

```
PROCEDURE OutXY(x, y : BYTE;
                t    : STRING);
VAR i, x1, y1, y2 : INTEGER;
BEGIN
   y1:=fy(y);
   y2:=fy(y+1)-1;
   FOR i:=1 TO Length(t) DO
      BEGIN
         SetColor(_c0[_C]);
         FOR x1:=fx(x+i-1) TO fx(x+i) DO Line(x1, y1, x1, y2);
         SetColor(_c4[_C]);
         OutTextXY(fx(x+i-1), y1, t[i])
      END
END; { OutXY }
```

## Windows management

## Redefinition of the colors in monochrome or active color mode.

```
PROCEDURE SetWinColors(frame, light, shadow, foreground, background : WORD);
BEGIN
   _c0[_C]:=background;
   _c1[_C]:=frame;
   _c2[_C]:=shadow;
   _c3[_C]:=light;
   _c4[_C]:=foreground
END; { SetWinColors }

PROCEDURE OpenGWin(x, y, nx, ny : INTEGER;
                   normal       : BOOLEAN);
VAR w : POINTER;
    s : LONGINT;
BEGIN
   GetStatus(_s);
   HideMouse;
   SetCharSize;
   IF normal
      THEN _dxw:=(nx+4)*_dx
      ELSE _dxw:=(nx+6)*_dx;
   _dyw:=(ny+4)*_dy;
   IF GetMaxX<_dxw THEN
      BEGIN
         _dxw:=GetMaxX;
         x:=0
      END;
   IF GetMaxY<_dyw THEN
      BEGIN
         _dyw:=GetMaxY;
         y:=0
      END;
   IF GetMaxX<x+_dxw THEN x:=GetMaxX-_dxw;
   IF GetMaxY<y+_dyw THEN y:=GetMaxY-_dyw;
   s:=(LONGINT(_dxw)+1)*(LONGINT(_dyw)+1);
   IF $FFFF<s THEN HALT(200);
   w:=GetPointer(ImageSize(0, 0, _dxw, _dyw));
   SetViewPort(x, y, x+_dxw, y+_dyw, ClipOn);
   GetImage(0, 0, _dxw, _dyw, w^);
   ClearViewPort;
   SetColor(_c1[_C]);
   Rectangle(0, 0, _dxw, _dyw);
   IF normal
      THEN Rectangle(_dx, _dy, _dxw-_dx, _dyw-_dy)
      ELSE Rectangle(_dx, _dy, _dxw-2*_dx, _dyw-_dy);
   IF _C
      THEN SetFillStyle(SolidFill, _c1[_C])
      ELSE SetFillStyle(InterleaveFill, _c1[_C]);
   FloodFill(1, 1, _c1[_C]);
   SetFillStyle(SolidFill, _c0[_C]);
   FloodFill(_dx+1, _dy+1, _c1[_C]);
   IF _C
      THEN SetLineStyle(SolidLn, 0, ThickWidth)
      ELSE SetLineStyle(SolidLn, 0, NormWidth);
   SetColor(_c2[_C]);
   IF normal
      THEN Line(_dx, _dy, _dxw-_dx, _dy)
      ELSE Line(_dx, _dy, _dxw-2*_dx, _dy);
   Line(_dx, _dy, _dx, _dyw-_dy);
   Line(0, _dyw, _dxw, _dyw);
   Line(_dxw, 0, _dxw, _dyw);
   SetLineStyle(SolidLn, 0, ThickWidth);
   SetColor(_c3[_C]);
   Line(0, 0, _dxw, 0);
   Line(0, 0, 0, _dyw);
   IF normal THEN
      BEGIN
         Line(_dxw-_dx, _dy, _dxw-_dx, _dyw-_dy);
         Line(_dx, _dyw-_dy, _dxw-_dx, _dyw-_dy)
      END
   ELSE
      BEGIN
         Line(_dxw-2*_dx, _dy, _dxw-2*_dx, _dyw-_dy);
         Line(_dx, _dyw-_dy, _dxw-2*_dx, _dyw-_dy)
      END;
   SetLineStyle(SolidLn, 0, NormWidth);
   IF normal THEN
      SetViewPort(x+2*_dx, y+2*_dy, x+_dxw-2*_dx, y+_dyw-2*_dy, ClipOn)
   ELSE BEGIN
      IF _C
         THEN SetColor(_c4[_C])
         ELSE SetColor(_c0[_C]);
      SetTextJustify(CenterText, CenterText);
      Line(_dxw-_dx, _dy, _dxw-_dx, _dyw-_dy);
      OutTextXY(_dxw-_dx, _dy, #30);
      OutTextXY(_dxw-_dx, _dyw-_dy, #31);
      OutTextXY(_dxw-_dx, 2*_dy, #219)
   END;
   SetColor(_c4[_C]);
   SetTextJustify(LeftText, TopText);
   ShowMouse
END; { OpenGWin }
```

Graphic window with upper left edge in x and y and with a width (nx) and height (ny) are expressed in number of characters

```
PROCEDURE OpenGrWin(x, y, nx, ny : INTEGER);
BEGIN
   OpenGWin(x, y, nx, ny, TRUE)
END; { OpenGrWin }

PROCEDURE OpenDirWin;
BEGIN
   OpenGWin(GetMaxX DIV 5, GetMaxY DIV 8, 13, 20, FALSE)
END; { OpenGrWin }

PROCEDURE OpenCenterGrWin(DX, dy : INTEGER);
BEGIN
   GetStatus(_s);
   SetCharSize;
   SetStatus(_s);
   OpenGrWin((GetMaxX-(DX+4)*_dx) DIV 2, (GetMaxY-(dy+4)*_dy) DIV 2, DX, dy)
END; { OpenCenterGrWin }
```

Closing the graphic window

```
PROCEDURE CloseGrWin;
VAR v : ViewPortType;
BEGIN
   HideMouse;
   GetViewSettings(v);
   SetViewPort(v.x1-2*_dx, v.y1-2*_dy, v.x2+2*_dx, v.y2+2*_dy, ClipOn);
   PutImage(0, 0, _plist[1].p^, NormalPut);
   FreeAllPointers;
   SetStatus(_s);
   ShowMouse
END; { CloseGrWin }

PROCEDURE CloseDirGrWin;
BEGIN
   HideMouse;
   PutImage(0, 0, _plist[1].p^, NormalPut);
   FreeAllPointers;
   SetStatus(_s);
   ShowMouse
END; { CloseGrWin }
```

Window displaying the message (m) in the middle of the screen

```
PROCEDURE OpenMsgGrWin(m : STRING);
VAR c : WORD;
BEGIN
   c:=_c0[_C];
   IF _C THEN _c0[_C]:=LightRed;
   OpenCenterGrWin(Length(m), 1);
   OutTextXY(0,0,m);
   _c0[_C]:=c
END; { MsgGrWin }
```

Window displaying the message (m) in the middle of the screen. Closes when a key or a button of the mouse is pushed.

```
PROCEDURE MsgGrWin(m : STRING);
VAR x, y, b : INTEGER;
BEGIN
   OpenMsgGrWin(m);
   StillMouse;
   REPEAT GetPosBut(b, x, y) UNTIL (b<>0) OR KeyPressed;
   IF KeyPressed THEN Pause;
   CloseGrWin
END; { MsgGrWin }
```

Edition window with upper left angle is defined by x and y.
title :title on the left
s : string to be edited
l : length of the string
mask : authorized characters
upchar : Upper case alone or not
ESC or left button of the mouse: cancellation
ENTER or right button of the mouse: validation.
Only the correction key works for the edition

```
FUNCTION EditGrWin(x, y    : INTEGER;
                   title   : STRING;
                   s       : StrPtr;
                   l       : BYTE;
                   mask    : AnyChar;
                   upchar  : BOOLEAN) : BOOLEAN;
VAR n          : STRING;
    p, c       : BYTE;
    k          : CHAR;
    b, mx, my  : INTEGER;
BEGIN
   p:=Length(title)+1;
   n:=s^;
   IF 70<p+l THEN
      BEGIN
         l:=70-p;
         Delete(n, l+1, Length(n)-l)
      END;
   OpenGrWin(x, y, p+l, 1);
   HideMouse;
   OutTextXY(fx(1), fy(1), Concat(title, n, #22));
   c:=p+Length(n);
   StillMouse;
   REPEAT
      GetPosBut(b, mx, mx);
      IF KeyPressed
         THEN k:=ReadKey
         ELSE k:=#0;
      IF upchar THEN k:=UpCase(k);
      IF (k IN mask) AND (c-p<l) THEN
         BEGIN
            n:=Concat(n, k);
            OutXY(c, 1, Concat(k,#22));
            Inc(c)
         END
      ELSE IF k=#8 THEN
         BEGIN
            Delete(n, Length(n), 1);
            IF 0<(c-p) THEN Dec(c);
            OutXY(c, 1, #22#32)
         END
   UNTIL (k IN [#13, #27]) OR (b IN [1, 2]);
   IF (k=#13) OR (b=1)
      THEN BEGIN
         s^:=n;
         EditGrWin:=TRUE
      END
      ELSE EditGrWin:=FALSE;
   ShowMouse;
   CloseGrWin
END; { EditGrWin }
```

## Openfile dialog box

```
FUNCTION DirGrWin(VAR path : PathStr) : BOOLEAN;

TYPE
   DirArray = ARRAY[1..512] OF Str15;

VAR
   n, home, csr, a, wx, wy : INTEGER;
   r, d                    : POINTER;
Reverse;
BEGIN
   IF (csr<=home) OR (21<csr-home) THEN EXIT;
   HideMouse;
   GetImage(2*_dx, (csr-home+1)*_dy-1, 15*_dx, (csr-home+2)*_dy-2, r^);
   PutImage(2*_dx, (csr-home+1)*_dy-1, r^, NOTPut);
   ShowMouse
END; { Reverse }

PROCEDURE DisplayPage(b : INTEGER);
VAR i : INTEGER;
BEGIN
   HideMouse;
   SetTextJustify(CenterText, CenterText);
   IF _C
      THEN BEGIN
         SetColor(_c1[_C]);
         OutTextXY(_dxw-_dx, a, #219);
      END
      ELSE BEGIN
         SetFillStyle(InterleaveFill, _c1[_C]);
         FloodFill(_dxw-_dx, a, _c1[_C])
      END;
   a:=b;
   IF _C
      THEN SetColor(_c4[_C])
      ELSE SetColor(_c0[_C]);
   OutTextXY(_dxw-_dx, a, #219);
   Line(_dxw-_dx, _dy, _dxw-_dx, _dyw-_dy);
   SetTextJustify(LeftText, TopText);
   SetColor(_c4[_C]);
   FOR i:=home+1 TO home+20 DO
      BEGIN
         PutImage(2*_dx, (i-home+1)*_dy-1, d^, NormalPut);
         IF i<=n  THEN OutTextXY(2*_dx, (i-home+1)*_dy, Dir^[i]);
         IF i=csr THEN
            BEGIN
               ShowMouse;
               Reverse;
               HideMouse
            END
      END;
   ShowMouse
END; { DisplayPage }

FUNCTION IsInField(x, y, x1, y1, x2, y2 : INTEGER) : BOOLEAN;
BEGIN
   x:=x-wx;
   y:=y-wy;
   IsInField:=(x1<x) AND (x<x2) AND (y1<y) AND (y<y2)
END; { IsInField }

FUNCTION DoubleClick(x1, y1 : INTEGER) : BOOLEAN;
VAR b, x, y : INTEGER;
BEGIN
   b:=0;
   REPEAT
      GetPosBut(b, x, y)
   UNTIL (x<>x1) OR (y<>y1) OR (b<>1);
   REPEAT
      GetPosBut(b, x, y)
   UNTIL (x<>x1) OR (y<>y1) OR (b<>0);
   DoubleClick:=(x=x1) AND (y=y1) AND (b=1)
END; { DoubleClick }

FUNCTION Select : BOOLEAN;
VAR b, x, y, newc, a : INTEGER;
    ok               : BOOLEAN;
BEGIN
   ok:=FALSE;
   REPEAT
      GetPosBut(b, x, y);
      IF b=1 THEN
         IF IsInField(x, y, 2*_dx, 2*_dy, 16*_dx, _dyw-2*_dy) THEN
            BEGIN
               newc:=home+((y-wy) DIV _dy)-1;
               IF (csr<>newc) AND (newc<=n) THEN
                  BEGIN
                     Reverse;
                     csr:=newc;
                     Reverse;
                     ok:=DoubleClick(x, y)
                  END
               ELSE ok:=(csr=newc) AND (csr<>0) AND DoubleClick(x, y)
            END
         ELSE IF IsInField(x, y, _dxw-3*_dx, 0, _dxw, _dyw) THEN
            BEGIN
```

```
               a:=y-wy;
               IF a<2*_dy THEN
                  BEGIN
                     a:=2*_dy;
                     home:=0
                  END
               ELSE IF _dyw-2*_dy<a THEN
                  BEGIN
                    a:=_dyw-2*_dy;
                    home:=n-1
                  END
               ELSE home:=Round((a-2*_dy)/(20*_dy/n));
               IF home<0 THEN home:=0;
               IF n-1<home THEN home:=n-1;
               DisplayPage(a)
            END
   UNTIL (b=2) OR (ok AND (0<csr));
   Select:=ok AND (0<csr)
END; { Select }

VAR ok, get : BOOLEAN;
    New     : PathStr;
BEGIN
   OpenDirWin;
   wx:=GetMaxX DIV 5;
   wy:=GetMaxY DIV 8;
   r:=GetPointer(ImageSize(2*_dx, 2*_dy, 15*_dx, 3*_dy-1));
   d:=GetPointer(ImageSize(2*_dx, 2*_dy, 15*_dx, 3*_dy-1));
   HideMouse;
   GetImage(2*_dx, 2*_dy, 15*_dx, 3*_dy-1, d^);
   ShowMouse;
   Dir:=GetPointer(SizeOf(DirArray));
   New:=path;
   get:=FALSE;
   REPEAT
      GetDirectory(New, n);
      IF n=0 THEN
         BEGIN
            CloseDirGrWin;
            DirGrWin:=FALSE;
            MsgGrWin(ErrorStr(DosError));
            EXIT
         END;
      home:=0;
      csr:=0;
      ok:=FALSE;
      DisplayPage(2*_dy);
      get:=Select;
      IF get THEN
         IF Dir^[csr,1]='\' THEN
            New:=NewPath(New, Dir^[csr])
         ELSE ok:=TRUE
      ELSE ok:=TRUE
   UNTIL ok;
   CloseDirGrWin;
   IF get THEN path:=Concat(PSplit(New), Dir^[csr]);
   DirGrWin:=get
END; { DirGrWin }
```

Allows the edition of the path of a file. If there are replacement characters it opens a directory window with a lift to go through the list. To take a file click twice on the chosen name. TRUE is when a name ha been chose. It is restituted in <new>.

```
FUNCTION GetGrFileName(path  : PathStr;
                   VAR New   : PathStr;
                       title : STRING) : BOOLEAN;
VAR dir   : DirStr;
    Name  : NameStr;
    ext   : ExtStr;
BEGIN
   New:=path;
   IF EditGrWin(GetMaxX DIV 5, GetMaxY DIV 8,
                title, @New, 36, PathChar, TRUE) THEN
      BEGIN
         FSplit(New , dir, Name, ext);
         IF dir='' THEN GetDir(0, dir);
         IF dir[Length(dir)]<>'\' THEN dir:=Concat(dir, '\');
         IF ((Name='') AND (ext='')) OR ((Name='*') AND (ext='')) THEN
```

```
            New:=Concat(dir, '*.*')
         ELSE New:=Concat(dir, Name, ext);
         IF (Pos('*', New)<>0) OR (Pos('?', New)<>0) THEN
            IF DirGrWin(New) THEN
               GetGrFileName:=TRUE
            ELSE BEGIN
               New:=path;
               GetGrFileName:=FALSE
            END
         ELSE GetGrFileName:=TRUE
      END
   ELSE BEGIN
      New:=path;
      GetGrFileName:=FALSE
   END
END; { GetGrFileName }

END.
```

## *Library GPRINT*

### Function : Generating graphics on printer

```
UNIT GPrint;

INTERFACE

USES Dos, Crt, Printer, Tools;

TYPE
```

Choice of the density of printing: resolution X / Y
- Low : 60 / 72 dots/inch
- Medium : 120 / 144 dots/inch
- High : 240 / 216 dots/inch

If you chose an isotropic aspect you must chose the adequate ratio: y/x = 5/6, 5/6, 10/9.

```
   GraphicDensitySet = (Low, Medium, High);
```

Routine of the user's drawing

```
   GraphProc         = PROCEDURE;

CONST
   Empty = 0;
   Solid = 1;

   _FillStyle : BYTE = Empty;
   _CharSize  : BYTE = 1;

PROCEDURE Line(x, y, x2, y2 : INTEGER);
PROCEDURE Rectangle(x1, y1, x2, y2 : INTEGER);
PROCEDURE FillRectangle(x1, y1, x2, y2 : INTEGER);
PROCEDURE OutTextXY(x, y : INTEGER;
                    s    : STRING);
FUNCTION PrintGraph(xmin, ymin, xmax, ymax : INTEGER;
                    density                : GraphicDensitySet;
                    Graphic                : GraphProc) : BYTE;

IMPLEMENTATION

TYPE
   CharPtr   = ^CHAR;
   BytePtr   = ^BYTE;
   CharArray = ARRAY[0..$FFFE] OF CHAR;
   ByteArray = ARRAY[0..$FFFE] OF BYTE;
   CharArrP  = ^CharArray;
   ByteArrP  = ^ByteArray;

   PlotProc  = PROCEDURE(x, y : INTEGER);

   LineType    = RECORD
                    p    : BOOLEAN;
                    x, y : INTEGER
                 END;
   LineTypeArr = ARRAY[1..100] OF LineType;
   LineArrPtr  = ^LineTypeArr;
   CharType    = RECORD
                    n : BYTE;
                    l : LineArrPtr
                 END;

CONST
   LMAX  = 2880;
   bits : ARRAY[0..7] OF BYTE = ($80,$40,$20,$10,$08,$04,$02,$01);
   F1MIN = 48;
   F1MAX = 57;

VAR
   _Rxmin, _Rymin,
   _Rxmax, _Rymax,
   _Rdx  , _Rdy     : INTEGER;
   _R               : ByteArrP;
   _Font1           : ARRAY[F1MIN..F1MAX] OF CharType;

FUNCTION RasterGraphic(density : GraphicDensitySet) : BYTE;

   FUNCTION Wr(out : STRING) : BOOLEAN;
   BEGIN
      {$I-} Write(Lst, out); {$I+}
      IF IOResult<>0
         THEN BEGIN
            RasterGraphic:=253;
            Wr:=TRUE
         END
         ELSE IF KeyPressed AND (ReadKey=#27)
            THEN BEGIN
               RasterGraphic:=251;
               Wr:=TRUE
            END
            ELSE Wr:=FALSE
   END; { Wr}

   FUNCTION WrC(out : STRING) : BOOLEAN;
   BEGIN
      {$I-} Write(Lst, out); {$I+}
      IF IOResult<>0
         THEN BEGIN
            RasterGraphic:=253;
            WrC:=TRUE
         END
         ELSE WrC:=FALSE
   END; { WrC }

LABEL
   END_print;

CONST
   LineSpacing : ARRAY[Low..High, Low..High] OF CHAR
               = ((#24, #0, #0), (#1, #23, #0), (#1, #1, #22));
   DotsPerInch : ARRAY[Low..High] OF CHAR
               = ('K', 'L', 'Z');
   CR : String[2] = #13#10;

VAR
   x, y : INTEGER;
   i    : WORD;
   k    : GraphicDensitySet;
   b, l : BYTE;
   Line : ARRAY[0..LMAX-1] OF BYTE;

BEGIN { RasterGraphic }
   RasterGraphic:=0;
   y:=0;
   REPEAT
      FOR k:=Low TO density DO
         BEGIN
            i:=y*_Rdx;
            FillChar(Line, SizeOf(Line), 0);
            FOR x:=0 TO _Rdx-1 DO
               BEGIN
                  FOR b:=0 TO 7 DO
                     BEGIN
                        l:=Ord(k)+b*(Ord(density)+1);
                        IF (_R^[i+(l Div 8)*_Rdx] AND bits[l Mod 8])<>0
```

```
                           THEN Line[x]:=Line[x] OR bits[b]
                     END;
                  Inc(i)
               END;
            x:=_Rdx-1;
            WHILE (Line[x]=0) AND (0<x) DO Dec(x);
            IF 0<Line[x]
               THEN BEGIN
                  IF WrC(^[+'3'+LineSpacing[density,k]
                        +^[+DotsPerInch[density]+Chr(Lo(x+1))+Chr(Hi(x+1)))
                     THEN GOTO END_print;
                  FOR i:=0 TO x DO IF WrC(Chr(Line[i])) THEN GOTO END_print
               END
               ELSE IF Wr(^[+'3'+LineSpacing[density,k]) THEN GOTO END_print;
            IF Wr(CR) THEN GOTO END_print
         END;
      INC(y, ord(density)+1)
   UNTIL _Rdy-1<=y;
   IF Wr(^['2') THEN;
END_print:
END; { RasterGraphic }

FUNCTION Sign(x : REAL) : INTEGER;
BEGIN
   IF x<0 THEN
      Sign:=-1
   ELSE
      Sign:=1
END; { Sign }

PROCEDURE Swap(VAR a, b : INTEGER);
VAR
   c : INTEGER;
BEGIN
   c:=a;
   a:=b;
   b:=c
END; { Swap }

{$F+} PROCEDURE PlotDot(x, y : INTEGER); {$F-}
VAR
   i : WORD;
BEGIN
   IF (x<_Rxmin) OR (_Rxmax<x) OR (y<_Rymin) OR (_Rymax<y) THEN EXIT;
   i:=(x-_Rxmin)+_Rdx*((y-_Rymin) DIV 8);
   _R^[i]:=_R^[i] OR bits[((y-_Rymin) MOD 8)]
END; { PlotDot }

{$F+} PROCEDURE NotPlotDot(x, y : INTEGER); {$F-}
VAR
   i : WORD;
BEGIN
   IF (x<_Rxmin) OR (_Rxmax<x) OR (y<_Rymin) OR (_Rymax<y) THEN EXIT;
   i:=(x-_Rxmin)+_Rdx*((y-_Rymin) DIV 8);
   IF (_R^[i] AND bits[((y-_Rymin) MOD 8)])<>0
      THEN _R^[i]:=_R^[i] XOR bits[((y-_Rymin) MOD 8)]
END; { NotPlotDot }
```

## Creating a line in memory (Bresenham algorithm)

```
PROCEDURE Line(x, y, x2, y2 : INTEGER);
   { Note: IF line is steep, meanings OF x and y are swapped throughout }
VAR
   i, steps, sx, sy, DX, dy, e : INTEGER;
   steep                        : BOOLEAN;
BEGIN
   DX:=Abs(x2-x); sx:=sign(x2-x);
   dy:=Abs(y2-y); sy:=sign(y2-y);
   steep:=dy>DX;
   IF steep THEN
      BEGIN
         Swap( x,  y);
         Swap(DX, dy);
         Swap(sx, sy)
      END;
   e:=2*dy-DX;
   FOR i:=1 TO DX DO
      BEGIN
         IF steep THEN
            PlotDot(y, x)
         ELSE
```

```
            PlotDot(x, y);
         WHILE e>=0 DO
            BEGIN
               y:=y+sy;
               e:=e-2*DX
            END;
         x:=x+sx;
         e:=e+2*dy
      END;
   PlotDot(x2, y2)
END; { Line }
```

### Creating a rectangle in memory.

```
PROCEDURE Rectangle(x1, y1, x2, y2 : INTEGER);
BEGIN
   Line(x1, y1, x2, y1);
   Line(x2, y1, x2, y2);
   Line(x2, y2, x1, y2);
   Line(x1, y2, x1, y1)
END; { Rectangle }
```

### Filling a rectangle with a motif defined by <_FillStyle>.

```
PROCEDURE FillRectangle(x1, y1, x2, y2 : INTEGER);
VAR x, y : INTEGER;
    plot : PlotProc;
BEGIN
   IF _FillStyle=Solid
      THEN plot:=PlotDot
      ELSE plot:=NotPlotDot;
   FOR y:=succ(y1) TO pred(y2) DO
      FOR x:=succ(x1) TO pred(x2) DO
         plot(x, y)
END; { FillRectangle }
```

### Printing the graphic with the resolution <density>, included in the coordinates xmin, ymin and xmax, ymax. Returns 0 if everything OK, otherwise the error message appears. <Graphic> is the drawing routine defined by the user. Needs 64kb of memory to generate the graphic.

```
FUNCTION PrintGraph(xmin, ymin, xmax, ymax : INTEGER;
                    density                : GraphicDensitySet;
                    Graphic                : GraphProc) : BYTE;
VAR
   error : BYTE;
   dy    : INTEGER;
BEGIN
   _Rdx:=Abs(xmax-xmin)+1;
   _Rdy:=$FFFF DIV _Rdx;
   _Rdy:=_Rdy-(_Rdy MOD 3);
   _Rxmin:=xmin;
   _Rxmax:=xmax;
   _Rymin:=ymin;
   _Rymax:=ymin;
   IF (MaxAvail<$FFFF) OR (LMAX<_Rdx)
      THEN BEGIN
         PrintGraph:=8;
         Exit
      END;
   GetMem(_R, $FFFF);
   REPEAT
      FillChar(_R^, $FFFF, 0);
      IF _Rymax+8*_Rdy-1<ymax
         THEN Inc(_Rymax, 8*_Rdy-1)
         ELSE BEGIN
            _Rymax:=ymax;
            _Rdy:=1+(Abs(ymax-_Rymin) DIV 8);
            _Rdy:=_Rdy+3-(_Rdy MOD 3)
         END;
      Graphic;
      Error:=RasterGraphic(density);
      Inc(_Rymin, 8*_Rdy)
   UNTIL (ymax=_Rymax) or (Error<>0);
   FreeMem(_R, $FFFF);
   PrintGraph:=Error
END; { PrintGraph }
```

```
PROCEDURE PlotChar(c    : BYTE;
                   x, y : INTEGER);
VAR
   i : BYTE;
BEGIN
   FOR i:=1 TO pred(_Font1[c].n) DO
     IF _Font1[c].l^[succ(i)].p THEN
        Line(x+_CharSize*_Font1[c].l^[i].x,
             y+_CharSize*_Font1[c].l^[i].y,
             x+_CharSize*_Font1[c].l^[succ(i)].x,
             y+_CharSize*_Font1[c].l^[succ(i)].y)
END; { PlotChar }
```

## Writing a string of characters in x, y.

```
PROCEDURE OutTextXY(x, y : INTEGER;
                    s    : STRING);
VAR
   dx : INTEGER;
   c  : BYTE;
BEGIN
   dx:=7*_CharSize;
   x:=x-((length(s)*dx-2*_CharSize) div 2);
   y:=y-((9*_CharSize) div 2);
   FOR c:=1 TO length(s) DO
      BEGIN
         IF ord(s[c]) in [F1MIN..F1MAX] THEN PlotChar(ord(s[c]), x, y);
         inc(x, dx)
      END
END; { OutTextXY }

PROCEDURE LoadFont;
VAR c : BYTE;
BEGIN
   getmem(_Font1[48].l, sizeof(LineType)* 9);
   getmem(_Font1[49].l, sizeof(LineType)* 5);
   getmem(_Font1[50].l, sizeof(LineType)* 8);
   getmem(_Font1[51].l, sizeof(LineType)*15);
   getmem(_Font1[52].l, sizeof(LineType)* 4);
   getmem(_Font1[53].l, sizeof(LineType)* 9);
   getmem(_Font1[54].l, sizeof(LineType)*12);
   getmem(_Font1[55].l, sizeof(LineType)* 5);
   getmem(_Font1[56].l, sizeof(LineType)*17);
   getmem(_Font1[57].l, sizeof(LineType)*12);
   _Font1[48].n:= 9;
   _Font1[49].n:= 5;
   _Font1[50].n:= 8;
   _Font1[51].n:=15;
   _Font1[52].n:= 4;
   _Font1[53].n:= 9;
   _Font1[54].n:=12;
   _Font1[55].n:= 5;
   _Font1[56].n:=17;
   _Font1[57].n:=12;
   _Font1[48].l^[ 1].p:=FALSE; _Font1[48].l^[ 1].x:=1; _Font1[48].l^[ 1].y:=0;
   _Font1[48].l^[ 2].p:=TRUE ; _Font1[48].l^[ 2].x:=3; _Font1[48].l^[ 2].y:=0;
   _Font1[48].l^[ 3].p:=TRUE ; _Font1[48].l^[ 3].x:=4; _Font1[48].l^[ 3].y:=1;
   _Font1[48].l^[ 4].p:=TRUE ; _Font1[48].l^[ 4].x:=4; _Font1[48].l^[ 4].y:=7;
   _Font1[48].l^[ 5].p:=TRUE ; _Font1[48].l^[ 5].x:=3; _Font1[48].l^[ 5].y:=8;
   _Font1[48].l^[ 6].p:=TRUE ; _Font1[48].l^[ 6].x:=1; _Font1[48].l^[ 6].y:=8;
   _Font1[48].l^[ 7].p:=TRUE ; _Font1[48].l^[ 7].x:=0; _Font1[48].l^[ 7].y:=7;
   _Font1[48].l^[ 8].p:=TRUE ; _Font1[48].l^[ 8].x:=0; _Font1[48].l^[ 8].y:=1;
   _Font1[48].l^[ 9].p:=TRUE ; _Font1[48].l^[ 9].x:=1; _Font1[48].l^[ 9].y:=0;
   _Font1[49].l^[ 1].p:=FALSE; _Font1[49].l^[ 1].x:=1; _Font1[49].l^[ 1].y:=2;
   _Font1[49].l^[ 2].p:=TRUE ; _Font1[49].l^[ 2].x:=2; _Font1[49].l^[ 2].y:=0;
   _Font1[49].l^[ 3].p:=TRUE ; _Font1[49].l^[ 3].x:=2; _Font1[49].l^[ 3].y:=8;
   _Font1[49].l^[ 4].p:=FALSE; _Font1[49].l^[ 4].x:=1; _Font1[49].l^[ 4].y:=8;
   _Font1[49].l^[ 5].p:=TRUE ; _Font1[49].l^[ 5].x:=3; _Font1[49].l^[ 5].y:=8;
   _Font1[50].l^[ 1].p:=TRUE ; _Font1[50].l^[ 1].x:=0; _Font1[50].l^[ 1].y:=4;
   _Font1[50].l^[ 2].p:=TRUE ; _Font1[50].l^[ 2].x:=0; _Font1[50].l^[ 2].y:=1;
   _Font1[50].l^[ 3].p:=TRUE ; _Font1[50].l^[ 3].x:=1; _Font1[50].l^[ 3].y:=0;
   _Font1[50].l^[ 4].p:=TRUE ; _Font1[50].l^[ 4].x:=3; _Font1[50].l^[ 4].y:=0;
   _Font1[50].l^[ 5].p:=TRUE ; _Font1[50].l^[ 5].x:=4; _Font1[50].l^[ 5].y:=1;
   _Font1[50].l^[ 6].p:=TRUE ; _Font1[50].l^[ 6].x:=4; _Font1[50].l^[ 6].y:=4;
   _Font1[50].l^[ 7].p:=TRUE ; _Font1[50].l^[ 7].x:=0; _Font1[50].l^[ 7].y:=8;
   _Font1[50].l^[ 8].p:=TRUE ; _Font1[50].l^[ 8].x:=4; _Font1[50].l^[ 8].y:=8;
   _Font1[51].l^[ 1].p:=FALSE; _Font1[51].l^[ 1].x:=0; _Font1[51].l^[ 1].y:=2;
   _Font1[51].l^[ 2].p:=TRUE ; _Font1[51].l^[ 2].x:=0; _Font1[51].l^[ 2].y:=1;
   _Font1[51].l^[ 3].p:=TRUE ; _Font1[51].l^[ 3].x:=1; _Font1[51].l^[ 3].y:=0;
   _Font1[51].l^[ 4].p:=TRUE ; _Font1[51].l^[ 4].x:=3; _Font1[51].l^[ 4].y:=0;
   _Font1[51].l^[ 5].p:=TRUE ; _Font1[51].l^[ 5].x:=4; _Font1[51].l^[ 5].y:=1;
   _Font1[51].l^[ 6].p:=TRUE ; _Font1[51].l^[ 6].x:=4; _Font1[51].l^[ 6].y:=3;
   _Font1[51].l^[ 7].p:=TRUE ; _Font1[51].l^[ 7].x:=3; _Font1[51].l^[ 7].y:=4;
   _Font1[51].l^[ 8].p:=TRUE ; _Font1[51].l^[ 8].x:=2; _Font1[51].l^[ 8].y:=4;
   _Font1[51].l^[ 9].p:=FALSE; _Font1[51].l^[ 9].x:=3; _Font1[51].l^[ 9].y:=4;
   _Font1[51].l^[10].p:=TRUE ; _Font1[51].l^[10].x:=4; _Font1[51].l^[10].y:=5;
   _Font1[51].l^[11].p:=TRUE ; _Font1[51].l^[11].x:=4; _Font1[51].l^[11].y:=7;
   _Font1[51].l^[12].p:=TRUE ; _Font1[51].l^[12].x:=3; _Font1[51].l^[12].y:=8;
   _Font1[51].l^[13].p:=TRUE ; _Font1[51].l^[13].x:=1; _Font1[51].l^[13].y:=8;
   _Font1[51].l^[14].p:=TRUE ; _Font1[51].l^[14].x:=0; _Font1[51].l^[14].y:=7;
   _Font1[51].l^[15].p:=TRUE ; _Font1[51].l^[15].x:=0; _Font1[51].l^[15].y:=6;
   _Font1[52].l^[ 1].p:=FALSE; _Font1[52].l^[ 1].x:=3; _Font1[52].l^[ 1].y:=8;
   _Font1[52].l^[ 2].p:=TRUE ; _Font1[52].l^[ 2].x:=3; _Font1[52].l^[ 2].y:=0;
   _Font1[52].l^[ 3].p:=TRUE ; _Font1[52].l^[ 3].x:=0; _Font1[52].l^[ 3].y:=6;
   _Font1[52].l^[ 4].p:=TRUE ; _Font1[52].l^[ 4].x:=4; _Font1[52].l^[ 4].y:=6;
   _Font1[53].l^[ 1].p:=FALSE; _Font1[53].l^[ 1].x:=4; _Font1[53].l^[ 1].y:=0;
   _Font1[53].l^[ 2].p:=TRUE ; _Font1[53].l^[ 2].x:=0; _Font1[53].l^[ 2].y:=0;
   _Font1[53].l^[ 3].p:=TRUE ; _Font1[53].l^[ 3].x:=0; _Font1[53].l^[ 3].y:=3;
   _Font1[53].l^[ 4].p:=TRUE ; _Font1[53].l^[ 4].x:=3; _Font1[53].l^[ 4].y:=3;
   _Font1[53].l^[ 5].p:=TRUE ; _Font1[53].l^[ 5].x:=4; _Font1[53].l^[ 5].y:=4;
   _Font1[53].l^[ 6].p:=TRUE ; _Font1[53].l^[ 6].x:=4; _Font1[53].l^[ 6].y:=7;
   _Font1[53].l^[ 7].p:=TRUE ; _Font1[53].l^[ 7].x:=3; _Font1[53].l^[ 7].y:=8;
   _Font1[53].l^[ 8].p:=TRUE ; _Font1[53].l^[ 8].x:=1; _Font1[53].l^[ 8].y:=8;
   _Font1[53].l^[ 9].p:=TRUE ; _Font1[53].l^[ 9].x:=0; _Font1[53].l^[ 9].y:=7;
   _Font1[54].l^[ 1].p:=FALSE; _Font1[54].l^[ 1].x:=4; _Font1[54].l^[ 1].y:=0;
   _Font1[54].l^[ 2].p:=TRUE ; _Font1[54].l^[ 2].x:=2; _Font1[54].l^[ 2].y:=1;
   _Font1[54].l^[ 3].p:=TRUE ; _Font1[54].l^[ 3].x:=1; _Font1[54].l^[ 3].y:=2;
   _Font1[54].l^[ 4].p:=TRUE ; _Font1[54].l^[ 4].x:=0; _Font1[54].l^[ 4].y:=5;
   _Font1[54].l^[ 5].p:=TRUE ; _Font1[54].l^[ 5].x:=0; _Font1[54].l^[ 5].y:=7;
   _Font1[54].l^[ 6].p:=TRUE ; _Font1[54].l^[ 6].x:=1; _Font1[54].l^[ 6].y:=8;
   _Font1[54].l^[ 7].p:=TRUE ; _Font1[54].l^[ 7].x:=3; _Font1[54].l^[ 7].y:=8;
   _Font1[54].l^[ 8].p:=TRUE ; _Font1[54].l^[ 8].x:=4; _Font1[54].l^[ 8].y:=7;
   _Font1[54].l^[ 9].p:=TRUE ; _Font1[54].l^[ 9].x:=4; _Font1[54].l^[ 9].y:=5;
   _Font1[54].l^[10].p:=TRUE ; _Font1[54].l^[10].x:=3; _Font1[54].l^[10].y:=4;
   _Font1[54].l^[11].p:=TRUE ; _Font1[54].l^[11].x:=1; _Font1[54].l^[11].y:=4;
   _Font1[54].l^[12].p:=TRUE ; _Font1[54].l^[12].x:=0; _Font1[54].l^[12].y:=5;
   _Font1[55].l^[ 1].p:=FALSE; _Font1[55].l^[ 1].x:=0; _Font1[55].l^[ 1].y:=0;
   _Font1[55].l^[ 2].p:=TRUE ; _Font1[55].l^[ 2].x:=4; _Font1[55].l^[ 2].y:=0;
   _Font1[55].l^[ 3].p:=TRUE ; _Font1[55].l^[ 3].x:=0; _Font1[55].l^[ 3].y:=8;
   _Font1[55].l^[ 4].p:=FALSE; _Font1[55].l^[ 4].x:=1; _Font1[55].l^[ 4].y:=4;
   _Font1[55].l^[ 5].p:=TRUE ; _Font1[55].l^[ 5].x:=3; _Font1[55].l^[ 5].y:=4;
   _Font1[56].l^[ 1].p:=FALSE; _Font1[56].l^[ 1].x:=1; _Font1[56].l^[ 1].y:=0;
   _Font1[56].l^[ 2].p:=TRUE ; _Font1[56].l^[ 2].x:=3; _Font1[56].l^[ 2].y:=0;
   _Font1[56].l^[ 3].p:=TRUE ; _Font1[56].l^[ 3].x:=4; _Font1[56].l^[ 3].y:=1;
   _Font1[56].l^[ 4].p:=TRUE ; _Font1[56].l^[ 4].x:=4; _Font1[56].l^[ 4].y:=3;
   _Font1[56].l^[ 5].p:=TRUE ; _Font1[56].l^[ 5].x:=3; _Font1[56].l^[ 5].y:=4;
   _Font1[56].l^[ 6].p:=TRUE ; _Font1[56].l^[ 6].x:=1; _Font1[56].l^[ 6].y:=4;
   _Font1[56].l^[ 7].p:=TRUE ; _Font1[56].l^[ 7].x:=0; _Font1[56].l^[ 7].y:=5;
   _Font1[56].l^[ 8].p:=TRUE ; _Font1[56].l^[ 8].x:=0; _Font1[56].l^[ 8].y:=7;
   _Font1[56].l^[ 9].p:=TRUE ; _Font1[56].l^[ 9].x:=1; _Font1[56].l^[ 9].y:=8;
   _Font1[56].l^[10].p:=TRUE ; _Font1[56].l^[10].x:=3; _Font1[56].l^[10].y:=8;
   _Font1[56].l^[11].p:=TRUE ; _Font1[56].l^[11].x:=4; _Font1[56].l^[11].y:=7;
   _Font1[56].l^[12].p:=TRUE ; _Font1[56].l^[12].x:=4; _Font1[56].l^[12].y:=5;
   _Font1[56].l^[13].p:=TRUE ; _Font1[56].l^[13].x:=3; _Font1[56].l^[13].y:=4;
   _Font1[56].l^[14].p:=TRUE ; _Font1[56].l^[14].x:=1; _Font1[56].l^[14].y:=4;
   _Font1[56].l^[15].p:=TRUE ; _Font1[56].l^[15].x:=0; _Font1[56].l^[15].y:=3;
   _Font1[56].l^[16].p:=TRUE ; _Font1[56].l^[16].x:=0; _Font1[56].l^[16].y:=1;
   _Font1[56].l^[17].p:=TRUE ; _Font1[56].l^[17].x:=1; _Font1[56].l^[17].y:=0;
   _Font1[57].l^[ 1].p:=FALSE; _Font1[57].l^[ 1].x:=0; _Font1[57].l^[ 1].y:=8;
   _Font1[57].l^[ 2].p:=TRUE ; _Font1[57].l^[ 2].x:=2; _Font1[57].l^[ 2].y:=7;
   _Font1[57].l^[ 3].p:=TRUE ; _Font1[57].l^[ 3].x:=3; _Font1[57].l^[ 3].y:=6;
   _Font1[57].l^[ 4].p:=TRUE ; _Font1[57].l^[ 4].x:=4; _Font1[57].l^[ 4].y:=3;
   _Font1[57].l^[ 5].p:=TRUE ; _Font1[57].l^[ 5].x:=4; _Font1[57].l^[ 5].y:=1;
   _Font1[57].l^[ 6].p:=TRUE ; _Font1[57].l^[ 6].x:=3; _Font1[57].l^[ 6].y:=0;
   _Font1[57].l^[ 7].p:=TRUE ; _Font1[57].l^[ 7].x:=1; _Font1[57].l^[ 7].y:=0;
   _Font1[57].l^[ 8].p:=TRUE ; _Font1[57].l^[ 8].x:=0; _Font1[57].l^[ 8].y:=1;
   _Font1[57].l^[ 9].p:=TRUE ; _Font1[57].l^[ 9].x:=0; _Font1[57].l^[ 9].y:=3;
   _Font1[57].l^[10].p:=TRUE ; _Font1[57].l^[10].x:=1; _Font1[57].l^[10].y:=4;
   _Font1[57].l^[11].p:=TRUE ; _Font1[57].l^[11].x:=3; _Font1[57].l^[11].y:=4;
   _Font1[57].l^[12].p:=TRUE ; _Font1[57].l^[12].x:=4; _Font1[57].l^[12].y:=3;
END; { LoadFont }

BEGIN
   LoadFont
END.
```

## *Library HP*

Function : Generating graphics on plotter

```
UNIT HP;

INTERFACE

USES Printer, Crt;

TYPE
    OrientationType = (Portrait, Landscape);

FUNCTION InitPlot : BYTE;
FUNCTION Orientation(o : OrientationType) : BYTE;
FUNCTION Scale(x1, y1, x2, y2 : REAL) : BYTE;

FUNCTION SetColor(color : BYTE) : BYTE;
FUNCTION Line(x1, y1, x2, y2 : REAL) : BYTE;
PROCEDURE ILine(x1, y1, x2, y2 : INTEGER);
FUNCTION Rectangle(x1, y1, x2, y2 : REAL) : BYTE;
FUNCTION Circle(x, y, r : REAL) : BYTE;
FUNCTION SetTextSize(w, h : REAL) : BYTE;
FUNCTION OutTextXY(x, y : REAL;
                   s    : STRING) : BYTE;

IMPLEMENTATION
```

A4 Format

```
CONST
   _W : REAL = 0.187;
   _H : REAL = 0.269;

FUNCTION Error : BYTE;
BEGIN
   IF IOResult<>0 THEN
      Error:=247
   ELSE IF KeyPressed AND (ReadKey=#27) THEN
   BEGIN
      {$I-} Write(Lst, 'SP0;'); {$I+}
      Error:=251
   END
   ELSE
      Error:=0
END; { Error }
```

Reinitializing the plotter.

```
FUNCTION InitPlot : BYTE;
BEGIN
   {$I-} Write(Lst, 'IN;IP;IW;'); {$I+}
   InitPlot:=Error
END; { InitPlot }
```

Determining the orientation.

```
FUNCTION Orientation(o : OrientationType) : BYTE;
BEGIN
   IF o=Portrait THEN
      {$I-} Write(Lst, 'RO90;IP;IW;')  {$I+}
   ELSE
      {$I-} Write(Lst, 'RO0;IP;IW;'); {$I+}
   Orientation:=Error
END; { Orientation }
```

Defining the user's coordinates in relation with the default surface of the page

```
FUNCTION Scale(x1, y1, x2, y2 : REAL) : BYTE;
BEGIN
   {$I-} Write(Lst, 'SC', x1:0:0, ',', x2:0:0, ',',
                              y1:0:0, ',', y2:0:0, ';'); {$I+}
  Scale:=Error
END; { Scale }
```

### Selection of the color

```
FUNCTION SetColor(color : BYTE) : BYTE;
BEGIN
  {$I-} Write(Lst, 'SP', color, ';'); {$I+}
  SetColor:=Error
END; { SetColor }
```

### Tracing a line in the absolute coordinates defined by the user

```
FUNCTION Line(x1, y1, x2, y2 : REAL) : BYTE;
BEGIN
  {$I-} Write(Lst, 'PAPU', x1:0:4, ',', y1:0:4,
                   'PD'  , x2:0:4, ',', y2:0:4, 'PU;'); {$I-}
  Line:=Error
END; { Line }

PROCEDURE ILine(x1, y1, x2, y2 : INTEGER);
BEGIN
  {$I-} Write(Lst, 'PAPU', x1, ',', y1:4,
                   'PD'  , x2, ',', y2:4, 'PU;'); {$I-}
END; { ILine }
```

### Tracing a rectangle in the absolute coordinates defined by the user

```
FUNCTION Rectangle(x1, y1, x2, y2 : REAL) : BYTE;
BEGIN
  {$I-} Write(Lst, 'PAPU', x1:0:4, ',', y1:0:4,
                   'PD'  , x2:0:4, ',', y1:0:4, ',',
                           x2:0:4, ',', y2:0:4, ',',
                           x1:0:4, ',', y2:0:4, ',',
                           x1:0:4, ',', y1:0:4, 'PU;'); {$I-}
  Rectangle:=Error
END; { Rectangle }
```

### Tracing a circle centered on x,y of radius r in the absolute coordinates defined by the user

```
FUNCTION Circle(x, y, r : REAL) : BYTE;
BEGIN
  {$I-} Write(Lst, 'PAPU', x:0:4, ',', y:0:4, 'CI', r:0:4, 'PU;'); {$I-}
  Circle:=Error
END; { Circle }
```

### Defining the dimensions of the characters in cm; default = 0.187, 0.269.

```
FUNCTION SetTextSize(w, h : REAL) : BYTE;
BEGIN
  _W:=w;
  _H:=h;
  {$I-} Write(Lst, 'DI1,0;SI', w:0:3, ',', h:0:3, ';'); {$I+}
  SetTextSize:=Error
END; { SetTextSize }
```

### Writing a string s centered in position x, y.

```
FUNCTION OutTextXY(x, y : REAL;
                   s    : STRING) : BYTE;
BEGIN
  {$I-} Write(Lst, 'PAPU', x:0:4, ',', y:0:4,
                   ';CP-', Length(s)/2-1/6:0:4, ',-0.25;',
                   ';LB', s, #3'PU;'); {$I+}
  OutTextXY:=Error
END; { OutTextXY }

END.
```

# 5.5 Utilities (Pascal)

## *Program BG_MARK*

Function : inscription of the serial number and of the user's name in the distributed version

```
PROGRAM BG_MARK;

{$A+,B-,D-,E+,F-,G+,L-,N-,O-,R-,S+,V-,X+}

USES Crt, Dos;

TYPE StrPtr = ^STRING;
     Str78  = STRING[78];

CONST BS = ^H;
      CR = ^M;

      p1 = 131 * 512 + 136;
      p2 =         p1 +  79;
      p3 =         p1 + 158;
      p4 =         p1 + 238;

      Serial : String[10] = '2.03F-0000';

VAR ExitSave      : POINTER;
    Owner, o1, o2 : Str78;
    Disk          : CHAR;
    i             : BYTE;

{$F+} PROCEDURE FatalError; {$F-}
VAR Msg : PathStr;
BEGIN
   CASE ExitCode OF
          0 : Msg:='Normal termination';
          1 : Msg:='Invalid FUNCTION code';
          2 : Msg:='File not found';
          3 : Msg:='Path not found';
          4 : Msg:='Too many open files';
          5 : Msg:='File access denied';
          6 : Msg:='Invalid file handle';
          7 : Msg:='Memory control blocks destroyed';
          8 : Msg:='Not enough memory FOR the child process';
          9 : Msg:='Invalid memory block address';
         10 : Msg:='Invalid environment';
         11 : Msg:='Invalid format';
         12 : Msg:='Invalid file access code';
         13 : Msg:='Invalid data';

         15 : Msg:='Invalid drive';
         16 : Msg:='Cannot remove current Directory';
         17 : Msg:='Cannot Rename accross different devices';
         18 : Msg:='No more files';

        100 : Msg:='Disk Read error';
        101 : Msg:='Disk Write error';
        102 : Msg:='File not assigned';
        103 : Msg:='File not open';
        104 : Msg:='File not open FOR Input';
        105 : Msg:='File not open FOR Output';
        106 : Msg:='Invalid numeric format';

        150 : Msg:='Disk is Write-protected';
        151 : Msg:='Unknown unit';
        152 : Msg:='Drive not ready';
        153 : Msg:='Unknown command';
        154 : Msg:='CRC error in data';
        155 : Msg:='Bad drive request structure Length';
        156 : Msg:='Disk Seek error';
        157 : Msg:='Unknown media TYPE';
        158 : Msg:='Sector not found';
        159 : Msg:='Printer out OF paper';
```

```
         160 : Msg:='Device Write fault';
         161 : Msg:='Device Read fault';
         162 : Msg:='Hardware failure';

         200 : Msg:='Division by zero';
         201 : Msg:='Range check';
         202 : Msg:='Stack overflow';
         203 : Msg:='Heap overflow';
         204 : Msg:='Invalid POINTER operation';
         205 : Msg:='Floating point overflow';
         206 : Msg:='Floating point underflow';
         207 : Msg:='Invalid floating point operation';
         208 : Msg:='Overlay manager not installed';
         209 : Msg:='Overlay file Read error';

         240 : Msg:='Disk full';

         255 : Msg:='User break'
       ELSE Msg:='Unexpected error'
       END;
   Writeln;
   IF NOT (ExitCode IN [0, 255]) THEN Write('Error ', ExitCode, ' : ');
   Writeln(Msg);
   ErrorAddr:=NIL;
   ExitProc:=ExitSave
END; { FatalError }

PROCEDURE PatchFile(Path  : PathStr;
                    Patch : StrPtr;
                    Pos   : LONGINT;
                    Len   : BYTE);
VAR i : BYTE;
    f : FILE OF CHAR;
    t : LONGINT;
BEGIN
   Assign(f, Path);
   Reset(f);
   GetFTime(f, t);
   Seek(f, Pos);
   FOR i:=0 TO Len DO Write(f, Patch^[i]);
   SetFTime(f, t);
   Close(f)
END; { PatchFile }

FUNCTION YesNo(Question : STRING) : BOOLEAN;
VAR Key : CHAR;
BEGIN
   Write(Question, ' [Y/N] ?');
   REPEAT
      Key:=UpCase(ReadKey);
   UNTIL Key IN ['Y', 'N'];
   GotoXY(1, WhereY);
   ClrEol;
   YesNo:=Key='Y'
END; { YesNo }

PROCEDURE Edit(VAR Str  : Str78;
                   L, Y : BYTE);
VAR c : CHAR;
BEGIN
   REPEAT
      GotoXY(1,Y);
      Write('>', Str);
      ClrEol;
      c:=ReadKey;
      CASE c OF
         BS                     : Delete(Str,Length(Str),1);
         #32..#126, #128..#255  : IF Length(Str)<L THEN Str:=Str+c
      END
   UNTIL c=CR;
   GotoXY(1,Y);
   ClrEol;
   HighVideo;
   Writeln('>', Str);
   NormVideo;
   ClrEol
END; { Edit }

FUNCTION Loop(Q : BYTE) : BOOLEAN;
TYPE CharSet = SET OF CHAR;
VAR Msg    : PathStr;
    Answer : CharSet;
    Key     : CHAR;
BEGIN
   CASE Q OF
      0 : Msg:='Est-ce juste ?';
   END;
   HighVideo; Write('A'); NormVideo; Write('bort, ');
   HighVideo; Write('R'); NormVideo; Write('etry');
   IF 1<Q
   THEN BEGIN
      Answer:=['A', 'R'];
      Write(' [A, R')
   END
   ELSE BEGIN
      Answer:=['A', 'R', 'C'];
      Write(', '); HighVideo; Write('C');
      NormVideo; Write('ontinue [A, R, C')
   END;
   Write('] ?');
   REPEAT
      Key:=UpCase(ReadKey);
   UNTIL Key IN Answer;
   GotoXY(1, WhereY);
   ClrEol;
   CASE Key OF
      'A' : HALT(255);
      'R' : Loop:=FALSE;
      'C' : Loop:=TRUE
   END
END; { Loop }

BEGIN
   ExitSave := ExitProc;
   ExitProc := @FatalError;
   ClrScr;
   HighVideo;
   Write('+------------------------------------------------------------------------------
            -+');
   Write('_ B I O G R A P H : BG_MARK v2.03    Copyright (c) 1990 by J. Savary & J. Guex
            _');
   Write('_ Institute OF Geology, University OF Lausanne, UNIL - BFSH2, CH-1015 Lausanne
            _');
   Write('+------------------------------------------------------------------------------
            -+');
   NormVideo;
   Writeln;
   Writeln('Marque le proprietaire et le numero de serie dans BIOGRAPH');
   Writeln;

   Write('Disquette [A/B] ? ');
   REPEAT Disk:=Upcase(ReadKey) UNTIL Disk IN ['A', 'B'];

   Owner:='';
   GotoXY(1, 8);
   Write('Proprietaire:');
   ClrEol;
   REPEAT
      Edit(Owner, SizeOf(Owner)-1, 9);
      Write('Est-ce juste ? ')
   UNTIL Loop(0);
   Writeln;

   Writeln('Numero de serie:');
   REPEAT
      Edit(Serial, SizeOf(Serial)-1, 12);
      Write('Est-ce juste ? ')
   UNTIL Loop(0);
   Writeln;

   o1[0]:=Owner[0];
   o2[0]:=Owner[0];
   FOR i:=1 TO Length(Owner) DO BEGIN
      o1[i]:=Chr(Ord(Owner[i])+27);
      o2[i]:=Chr(Ord(Owner[i])+28)
   END;
   Randomize;
   FOR i:=Length(Owner)+1 TO SizeOf(Owner)-1 DO BEGIN
      Owner[i]:=Chr(Random(256));
         o1[i]:=Chr(Random(256));
         o2[i]:=Chr(Random(256))
   END;

   PatchFile(Disk+':\BG_MENU.EXE', @o1     , p1, SizeOf(o1   ));
   PatchFile(Disk+':\BG_MENU.EXE', @o2    , p2, SizeOf(o2   ));
   PatchFile(Disk+':\BG_MENU.EXE', @Owner , p3, SizeOf(Owner));
   PatchFile(Disk+':\BG_MENU.EXE', @Serial, p4, 11           );

   HighVideo;
   Writeln('N'#39'oublie pas de noter le nom du proprietaire, son adresse,');
   Writeln('la date d'#39'envoi et le numero de serie')
END.
```

## *Program INSTALL*

Function : installation de Biograph

```
PROGRAM INSTALL;

{$A+,B-,D-,E+,F-,G+,I+,L-,N-,O-,R-,S+,V-,X+}
{$M 32768, 0, 0}

USES Crt, Dos;

CONST BS = ^H;
      CR = ^M;

      Ppos =  14 *512 + 302; { BIOGRAPH path > BG       }
      Epos =      Ppos +  80; { EDITOR   path > BG       }
      Dpos =   0 *512 +   0; { DATA     path > BG_CFG   }
      Cpos =   0 *512 + 463; { COLOR         > BG_CFG   }
      Gpos = 130 *512 + 488; { CONFIG   path > BG_MENU  }
      Wpos =      Gpos +  80; { EDITOR   path > BG_MENU  }
      Lpos =      Gpos + 432; { LOCK          > BG_MENU  }

      BUFFSIZE = 16384;
      DiskName = ':\BIOGRAPH.203';

      EndLoop : BOOLEAN = TRUE;

      Path : PathStr = 'C:\BIOGRAPH';
      Ed   : PathStr = 'C:\WORD\WORD.EXE';

TYPE StrPtr = ^STRING;

VAR ExitSave : POINTER;
    C        : CHAR;
    Disk     : PathStr;
    BGDisk   : CHAR;

{$F+} PROCEDURE FatalError; {$F-}
VAR Msg : PathStr;
BEGIN
   CASE ExitCode OF
          0 : Msg:='Installation completed';
          1 : Msg:='Invalid FUNCTION code';
          2 : Msg:='File not found';
          3 : Msg:='Path not found';
          4 : Msg:='Too many open files';
          5 : Msg:='File access denied';
          6 : Msg:='Invalid file handle';
          7 : Msg:='Memory control blocks destroyed';
          8 : Msg:='Not enough memory FOR the child process';
          9 : Msg:='Invalid memory block address';
         10 : Msg:='Invalid environment';
         11 : Msg:='Invalid format';
         12 : Msg:='Invalid file access code';
         13 : Msg:='Invalid data';

         15 : Msg:='Invalid drive';
         16 : Msg:='Cannot remove current Directory';
         17 : Msg:='Cannot Rename accross different devices';
         18 : Msg:='No more files';

        100 : Msg:='Disk Read error';
        101 : Msg:='Disk Write error';
        102 : Msg:='File not assigned';
        103 : Msg:='File not open';
        104 : Msg:='File not open FOR Input';
        105 : Msg:='File not open FOR Output';
```

```
         106 : Msg:='Invalid numeric format';

         150 : Msg:='Disk is Write-protected';
         151 : Msg:='Unknown unit';
         152 : Msg:='Drive not ready';
         153 : Msg:='Unknown command';
         154 : Msg:='CRC error in data';
         155 : Msg:='Bad drive request structure Length';
         156 : Msg:='Disk Seek error';
         157 : Msg:='Unknown media TYPE';
         158 : Msg:='Sector not found';
         159 : Msg:='Printer out OF paper';
         160 : Msg:='Device Write fault';
         161 : Msg:='Device Read fault';
         162 : Msg:='Hardware failure';

         200 : Msg:='Division by zero';
         201 : Msg:='Range check';
         202 : Msg:='Stack overflow';
         203 : Msg:='Heap overflow';
         204 : Msg:='Invalid POINTER operation';
         205 : Msg:='Floating point overflow';
         206 : Msg:='Floating point underflow';
         207 : Msg:='Invalid floating point operation';
         208 : Msg:='Overlay manager not installed';
         209 : Msg:='Overlay file Read error';

         240 : Msg:='Disk full';

         253 : Msg:='A tiny problem: you must retry';
         254 : Msg:='Device missing';
         255 : Msg:='User break'
      ELSE Msg:='Unexpected error'
      END;
   Writeln;
   IF NOT (ExitCode IN [0, 255]) THEN Write('Error ', ExitCode, ' : ');
   Writeln(Msg);
   ErrorAddr:=NIL;
   ExitProc:=ExitSave
END; { FatalError }

FUNCTION GetVideo : BOOLEAN;
VAR Regs : Registers;
BEGIN
   Intr($11,Regs);
   GetVideo:=Lo(Regs.AX) AND $30=$30
END; { GetVideo }

FUNCTION FindFile(Path : PathStr;
                  Attr : BYTE) : BOOLEAN;
VAR Entry : SearchRec;
BEGIN
   FindFirst(Path, Attr, Entry);
   FindFile:=DosError=0
END; { FindFile }

PROCEDURE DeleteFile(Path : PathStr);
VAR f : File;
BEGIN
   IF FindFile(Path, AnyFile) THEN
   BEGIN
      Assign(f, Path);
      Erase(f)
   END
END; { DeleteFile }

PROCEDURE GetDiskTime(VAR Disk : PathStr);
VAR Entry  : SearchRec;
    Regs   : Registers;
BEGIN
   Delete(Disk, 4, SizeOf(Disk));
   FindFirst(Disk+'*.*', VolumeID, Entry);
   IF DosError<>0 THEN BEGIN
      Disk:=Disk+'NONE'#0;
      Regs.AH:=$3C;
      Regs.CX:=$08;
      Regs.DS:=seg(Disk[1]);
      Regs.DX:=ofs(Disk[1]);
      intr($21, Regs);
      IF DosError<>0 THEN Halt(253)
   END;
   Move(Entry.Time, Disk[4], SizeOf(Entry.Time))
```

```
END; { GetDiskTime }

PROCEDURE PatchFile(Path  : PathStr;
                    Patch : StrPtr;
                    Pos   : LONGINT;
                    Len   : BYTE);
VAR i : BYTE;
    f : FILE OF CHAR;
    t : LONGINT;
BEGIN
   Assign(f, Path);
   Reset(f);
   GetFTime(f, t);
   Seek(f, Pos);
   FOR i:=0 TO Len-1 DO Write(f, Patch^[i]);
   SetFTime(f, t);
   Close(f)
END; { PatchFile }

FUNCTION YesNo(Question : STRING) : BOOLEAN;
VAR Key : CHAR;
BEGIN
   Write(Question, ' [Y/N] ?');
   REPEAT
      Key:=UpCase(ReadKey);
   UNTIL Key IN ['Y', 'N'];
   GotoXY(1, WhereY);
   ClrEol;
   YesNo:=Key='Y'
END; { YesNo }

PROCEDURE Edit(VAR Str : PathStr;
                   Y   : BYTE);
VAR c : CHAR;
BEGIN
   REPEAT
      GotoXY(1,Y);
      Write('>>> ', Str);
      ClrEol;
      c:=UpCase(ReadKey);
      CASE c OF
         BS                      : Delete(Str,Length(Str),1);
         '.','0'..':', '@'..'_' : Str:=Str+c
      END
   UNTIL c=CR;
   GotoXY(1,Y-1);
   ClrEol;
   HighVideo;
   Writeln('>>> ', Str);
   NormVideo;
   ClrEol
END; { Edit }

FUNCTION Loop(Q : BYTE) : BOOLEAN;
TYPE CharSet = SET OF CHAR;
VAR Msg    : PathStr;
    Answer : CharSet;
    Key    : CHAR;
BEGIN
   CASE Q OF
      0 : Msg:='Directory already existing';
      1 : Msg:='Editor not found';
      2 : Msg:='The BioGraph disk not found';
      3 : Msg:='BioGraph must be installed upon a hard disk'
   END;
   Write(Msg, ' ! ');
   HighVideo; Write('A'); NormVideo; Write('bort, ');
   HighVideo; Write('R'); NormVideo; Write('etry');
   IF 1<Q
   THEN BEGIN
      Answer:=['A', 'R'];
      Write(' [A, R')
   END
   ELSE BEGIN
      Answer:=['A', 'R', 'C'];
      Write(', '); HighVideo; Write('C');
      NormVideo; Write('ontinue [A, R, C')
   END;
   Write('] ?');
   REPEAT
      Key:=UpCase(ReadKey);
   UNTIL Key IN Answer;
   GotoXY(1, WhereY);
   ClrEol;
   Loop:=TRUE;
   CASE Key OF
      'A' : HALT(255);
      'R' : GotoXY(1, WhereY-1);
      'C' : Loop:=FALSE
   END
END; { Loop }

PROCEDURE CopyFile(Message, Mask : PathStr);
VAR sf, tf                   : FILE;
    NumRead, NumWritten, N : WORD;
    Buffer                 : ARRAY[1..BUFFSIZE] OF CHAR;
    Entry                  : SearchRec;
    t                      : LONGINT;
BEGIN
   HighVideo; Write('_ ', Message, ':'); NormVideo;
   N:=0;
   FindFirst(BGDisk+':\'+Mask, AnyFile, Entry);
   WHILE DosError=0 DO
   BEGIN
      Assign(sf, BGDisk+':\'+Entry.Name);
      Reset(sf, 1);
      GetFTime(sf, t);
      Assign(tf, Path+Entry.Name);
      Rewrite(tf, 1);
      Inc(N);
      GotoXY(22, WhereY);
      Write(N:3, '  ', Entry.Name, ' is being copied...');
      ClrEol;
      REPEAT
         BlockRead(sf, Buffer, BUFFSIZE, NumRead);
         BlockWrite(tf, Buffer, NumRead, NumWritten);
         IF NumRead<>NumWritten THEN HALT(101)
      UNTIL (NumRead=0) OR (NumWritten<>NumRead);
      Close(sf);
      Reset(tf);
      SetFTime(tf, t);
      Close(tf);
      FindNext(Entry)
   END;
   GotoXY(22, WhereY); Write(N:3, ' file');
   IF 1<N THEN Write('s');
   ClrEol;
   Writeln
END; { CopyFile }

BEGIN
   ExitSave := ExitProc;
   ExitProc := @FatalError;
   ClrScr;
   HighVideo;
   Write('+-------------------------------------------------------------------------------
         -+');
   Write('_ B I O G R A P H : INSTALL v2.03    Copyright (c) 1990 by J. Savary & J. Guex
         _');
   Write('_ Institute OF Geology, University OF Lausanne, UNIL - BFSH2, CH-1015 Lausanne
         _');
   Write('+-------------------------------------------------------------------------------
         -+');
   Writeln;
   Write('Version 2.03');
   NormVideo;
   Writeln(': Installation from the distribution disk upon hard disk');
   Writeln;

   REPEAT
      Writeln('Device and Directory where BIOGRAPH must be installed:');
      Edit(Path, 9);
      IF Pos(':', Path)=0 THEN HALT(254);
      IF Path[Length(Path)]='\' THEN Delete(Path, Length(Path), 1);
      IF Path[1] IN ['A','B']
         THEN EndLoop:=NOT Loop(3)
         ELSE EndLoop:=TRUE;
      IF Path[Length(Path)]<>':' THEN
         IF FindFile(Path, Directory)
            THEN EndLoop:=NOT Loop(0)
            ELSE MkDir(Path)
   UNTIL EndLoop;
   IF Path[Length(Path)]=':' THEN Path:=Path+'\';
   Disk:=Path;
```

```
  GetDiskTime(Disk);
  ChDir(Path);
  IF Path[Length(Path)]<>'\' THEN Path:=Path+'\';

  IF GetVideo THEN
  BEGIN
     C:=#0;
     HighVideo;
     Writeln('>>> Monochrom screen');
     NormVideo
  END
  ELSE BEGIN
     HighVideo; Write('M');
     NormVideo; Write('onochrom or ');
     HighVideo; Write('C');
     NormVideo; Write('olor Screen [M/C] ?');
     REPEAT
        C:=UpCase(ReadKey);
     UNTIL C IN ['C', 'M'];
     GotoXY(1, WhereY);
     HighVideo;
     IF C='M' THEN
     BEGIN
        C:=#0;
        Write('>>> Monochrom screen')
     END
     ELSE BEGIN
        C:=#1;
        Write('>>> Color screen')
     END;
     NormVideo;
     ClrEol;
     Writeln
  END;

  REPEAT
     Writeln('Full pathname OF your favorite editor:');
     Edit(Ed, 11);
     IF FindFile(Ed, AnyFile)
        THEN EndLoop:=TRUE
        ELSE EndLoop:=NOT Loop(1)
  UNTIL EndLoop;

  REPEAT
     Writeln;
     BGDisk:='A';
     IF FindFile(BGDisk+DiskName, VolumeID)
        THEN EndLoop:=TRUE
        ELSE BEGIN
           BGDisk:='B';
           IF FindFile(BGDisk+DiskName, VolumeID)
              THEN EndLoop:=TRUE
              ELSE EndLoop:=NOT Loop(2)
        END
  UNTIL EndLoop;

  CopyFile('PROGRAM FILES', 'BG*.*');

  DeleteFile(Path+'INSTALL.EXE');

  PatchFile(Path+'BG.EXE'     , @Path, Ppos, SizeOf(Path));
  PatchFile(Path+'BG.EXE'     , @Ed  , Epos, SizeOf(Ed  ));
  PatchFile(Path+'BG.CFG'     , @Path, Dpos, SizeOf(Path));
  PatchFile(Path+'BG.CFG'     , @C   , Cpos, SizeOf(C)   );
  PatchFile(Path+'BG_MENU.EXE', @Path, Gpos, SizeOf(Path));
  PatchFile(Path+'BG_MENU.EXE', @Ed  , Wpos, SizeOf(Ed  ));
  PatchFile(Path+'BG_MENU.EXE', @Disk, Lpos, 8           );

  Writeln;
  IF YesNo('Do you wish example data files')
     THEN CopyFile('EXAMPLE DATA FILES', '*.DAT');

  Writeln;
  Writeln('Type BG TO use BIOGRAPH')
END.
```

## 5.6 Analytical Utilities (Modula 2)

### *Program BG_CON*

Function : List of the arcs and of their weight between the cliques in Gk

```
MODULE BG_CON;

IMPORT IO, FIO;

FROM Str     IMPORT Append;
FROM Lib     IMPORT IncAddr, AddAddr, ParamCount, ParamStr;
FROM BG_ERR  IMPORT Error;
FROM BG_MEM  IMPORT INT_PTR, WORD_LIST, New, Release, PointSet, ClearMemory,
                    Difference, SizeOfDiagMat, DiagIndex, EDGE, SetRange,
                    GetRange, SetSegment, Min, Max;

CONST BUFFER = 0FFF8H;

TYPE EXTENSION  = ARRAY[0..3] OF CHAR;

VAR Taxa, _Size       : CARDINAL;
    Cliques           : ADDRESS;
    Mat, _Arcs, _Sum  : INT_PTR;
    _F                : FIO.File;
    Filename          : FIO.PathStr;
```

Files management

```
PROCEDURE CreateFileName(Name : FIO.PathStr;
                         Ext  : EXTENSION) : FIO.PathStr;
BEGIN
   Append(Name, Ext);
   RETURN Name
END CreateFileName;

PROCEDURE LoadFile(ptr  : ADDRESS;
                   size : LONGCARD);
VAR read : CARDINAL;
BEGIN
   WHILE BUFFER<size DO
      read:=FIO.RdBin(_F, ptr^, BUFFER);
      IncAddr(ptr, read);
      DEC(size, LONGCARD(read))
   END;
   read:=FIO.RdBin(_F, ptr^, CARDINAL(size));
   IF read<CARDINAL(size) THEN Error(100) END;
   FIO.Close(_F)
END LoadFile;

PROCEDURE Load_Levels(Path        : FIO.PathStr;
                      Ext         : EXTENSION;
                  VAR Taxa, Size  : CARDINAL;
                  VAR Level       : ADDRESS);
VAR NSec : CARDINAL;
BEGIN
   _F:=FIO.Open(CreateFileName(Path, Ext));
   IF FIO.RdBin(_F, Taxa, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   IF FIO.RdBin(_F, NSec, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   IF FIO.RdBin(_F, Size, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   SetRange(Taxa);
   IO.WrStr('Number of taxa: ');
   IO.WrCard(Taxa, 0);
   IO.WrLn;
   IO.WrStr('Number of cliques: ');
   IO.WrCard(Size, 0);
   IO.WrLn;
   IO.WrLn;
   New(Level, LONGCARD(Size)*LONGCARD(GetRange())+SIZE(CARDINAL));
   LoadFile(Level, LONGCARD(Size)*LONGCARD(GetRange()))
END Load_Levels;

PROCEDURE Load_Matrix(Path : FIO.PathStr;
                    Ext   : EXTENSION;
                VAR Mat   : ADDRESS;
                VAR Size  : CARDINAL);
BEGIN
   _F:=FIO.Open(CreateFileName(Path, Ext));
   IF FIO.RdBin(_F, Size, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   New(Mat, SizeOfDiagMat(Size));
   LoadFile(Mat, SizeOfDiagMat(Size))
END Load_Matrix;
```

## Tools

```
PROCEDURE GetCmdLine(VAR Name : FIO.PathStr);
BEGIN
   IO.WrStr("| BG_TOOL 5:");                                              IO.WrLn;
   IO.WrStr("————————————");                                              IO.WrLn;
   IO.WrStr("BG_CON v1.01 Copyright (c) 1990 by J. Savary & J. Guex"); IO.WrLn;
   IO.WrStr('Count arcs between the cliques');                            IO.WrLn;
   IO.WrLn;

   IF ParamCount()=0 THEN
      IO.WrStr("Usage: BG_CON <file> [>file | LPT1:]");
      IO.WrLn;
      HALT
   END;

   ParamStr(Name, 1);
   Append(Name, '.');
   Name:=CreateFileName(Name, 'BGD');
   IO.WrStr(Name); IO.WrLn;
   IF NOT FIO.Exists(Name) THEN
      IO.WrStr("File not found !"); IO.WrLn;
      HALT
   END;
   ParamStr(Name, 1);
   Append(Name, '.');
   Name:=CreateFileName(Name, 'BGE');
   IO.WrStr(Name); IO.WrLn;
   IF NOT FIO.Exists(Name) THEN
      IO.WrStr("File not found !"); IO.WrLn;
      HALT
   END;

   ParamStr(Name, 1);
   Append(Name, '.')
END GetCmdLine;

PROCEDURE IndexA(i, j : CARDINAL) : INT_PTR;
BEGIN
   RETURN AddAddr(_Arcs, 2*((i-1)+(j-1)*_Size))
END IndexA;

PROCEDURE IndexS(i, j : CARDINAL) : INT_PTR;
BEGIN
   RETURN AddAddr(_Sum, 2*((i-1)+(j-1)*_Size))
END IndexS;
```

## Construction of the matrix adjacent to the cliques graph Gk

```
PROCEDURE Count(Taxa, Size : CARDINAL;
                Mat        : INT_PTR;
                Cliques    : ADDRESS);
VAR mat, aij, aji, sij, sji : INT_PTR;
    i, j, N1, n1, N2, n2    : CARDINAL;
    Level, Arc              : ADDRESS;
    Arc1, Arc2              : WORD_LIST;
BEGIN
   New(Arc1 , LONGCARD(Taxa*SIZE(CARDINAL)));
   New(Arc2 , LONGCARD(Taxa*SIZE(CARDINAL)));
   SetSegment(Cliques);
   FOR i:=1 TO Size-1 DO
      Level:=PointSet(i);
      FOR j:=i+1 TO Size DO
         Difference(Level, PointSet(j), Arc1, Arc2, N1, N2);
         FOR n1:=0 TO N1-1 DO
            FOR n2:=0 TO N2-1 DO
               mat:=DiagIndex(Mat, Arc1^[n1], Arc2^[n2], Taxa);
               IF (mat^<>0) AND (mat^<>EDGE) THEN
                  aij:=IndexA(i, j);
                  aji:=IndexA(j, i);
                  sij:=IndexS(i, j);
                  sji:=IndexS(j, i);
                  IF Arc1^[n1]<Arc2^[n2] THEN
                     IF mat^<0 THEN
                        INC(aij^);
                        INC(sij^, ABS(mat^))
                     ELSE
                        INC(aji^);
                        INC(sji^, ABS(mat^))
                     END
                  ELSE
                     IF mat^<0 THEN
                        INC(aji^);
                        INC(sji^, ABS(mat^))
                     ELSE
                        INC(aij^);
                        INC(sij^, ABS(mat^))
                     END
                  END
               END
            END
         END
      END
   END;
   Release(Arc1)
END Count;

PROCEDURE ListArcs(Size : CARDINAL);
VAR i, j                   : CARDINAL;
    aij, aji, sij, sji : INT_PTR;
BEGIN
   IO.WrStr('Arcs:');
   IO.WrLn;
   FOR i:=1 TO Size-1 DO
      FOR j:=i+1 TO Size DO
         aij:=IndexA(i, j);
         aji:=IndexA(j, i);
         sij:=IndexS(i, j);
         sji:=IndexS(j, i);
         IF ((aij^=0) OR (aji^=0)) AND (aij^<>aji^) THEN
            IF aji^=0 THEN
               IO.WrCard(i, 3);
               IO.WrChar('>');
               IO.WrCard(j, 3);
               IO.WrChar(':');
               IO.WrInt(aij^, 5);
               IO.WrChar('+');
               IO.WrInt(sij^, 5)
            ELSE
               IO.WrCard(i, 3);
               IO.WrChar('<');
               IO.WrCard(j, 3);
               IO.WrChar(':');
               IO.WrInt(aji^, 5);
               IO.WrChar('+');
               IO.WrInt(sji^, 5)
            END;
            IO.WrLn
         END
      END
   END;
   IO.WrLn
END ListArcs;

PROCEDURE ListContradictions(Size : CARDINAL);
VAR i, j                   : CARDINAL;
    aij, aji, sij, sji : INT_PTR;
BEGIN
   IO.WrStr('Contradictions:');
   IO.WrLn;
   FOR i:=1 TO Size-1 DO
      FOR j:=i+1 TO Size DO
         aij:=IndexA(i, j);
         aji:=IndexA(j, i);
         sij:=IndexS(i, j);
         sji:=IndexS(j, i);
         IF ((aij^<>0) AND (aji^<>0)) OR (aij^=aji^) THEN
            IO.WrCard(i, 3);
            IF aij^+sij^=aji^+sji^ THEN
               IO.WrChar('?')
            ELSIF aij^+sij^>aji^+sji^ THEN
```

```
               IO.WrChar('>')
            ELSE
               IO.WrChar('<')
            END;
            IO.WrCard(j, 3);
            IO.WrChar(':');
            IO.WrInt(aij^, 5);
            IO.WrStr('> +');
            IO.WrInt(sij^, 5);
            IO.WrStr(' &');
            IO.WrInt(aji^, 5);
            IO.WrStr('< +');
            IO.WrInt(sji^, 5);
            IO.WrChar('=');
            IO.WrInt(TRUNC(1000.0*FLOAT(Min(aij^+sij^, aji^+sji^))/
                                  FLOAT(Max(aij^+sij^, aji^+sji^))), 5);
            IO.WrLn
         END
      END
   END;
   IO.WrLn
END ListContradictions;

BEGIN
   GetCmdLine(Filename);
   Load_Levels(Filename, 'BGE', Taxa, _Size, Cliques);
   Load_Matrix(Filename, 'BGD', Mat, Taxa);
   New(_Arcs, LONGCARD(2*_Size*_Size));
   ClearMemory(_Arcs, LONGCARD(2*_Size*_Size));
   New(_Sum, LONGCARD(2*_Size*_Size));
   ClearMemory(_Sum, LONGCARD(2*_Size*_Size));
   Count(Taxa, _Size, Mat, Cliques);
   ListArcs(_Size);
   ListContradictions(_Size);
END BG_CON.
```

## *Program BG_CS3*

### Function :  List of $S_3$ and  $C_3$ in G*

```
MODULE BG_CS3;

IMPORT IO, FIO;

FROM Str    IMPORT Append;
FROM Lib    IMPORT IncAddr, AddAddr, ParamCount, ParamStr, WordFill;
FROM BG_ERR IMPORT Error;
FROM BG_MEM IMPORT INT_PTR, New, Release, SizeOfDiagMat, DiagIndex, EDGE,
                   ClearMemory;

CONST BUFFER = 0FFF8H;

TYPE EXTENSION  = ARRAY[0..3] OF CHAR;

VAR _Taxa      : CARDINAL;
    _Mat, _Arc : INT_PTR;
    _Count     : ARRAY[1..500] OF CARDINAL;
    _F         : FIO.File;
    Filename   : FIO.PathStr;
```

### Files management

```
PROCEDURE CreateFileName(Name : FIO.PathStr;
                         Ext  : EXTENSION) : FIO.PathStr;
BEGIN
   Append(Name, Ext);
   RETURN Name
END CreateFileName;

PROCEDURE LoadFile(ptr  : ADDRESS;
                   size : LONGCARD);
VAR read : CARDINAL;
BEGIN
   WHILE BUFFER<size DO
      read:=FIO.RdBin(_F, ptr^, BUFFER);
      IncAddr(ptr, read);
      DEC(size, LONGCARD(read))
   END;
   read:=FIO.RdBin(_F, ptr^, CARDINAL(size));
   IF read<CARDINAL(size) THEN Error(100) END;
   FIO.Close(_F)
END LoadFile;

PROCEDURE Load_Matrix(Path : FIO.PathStr;
                      Ext  : EXTENSION);
BEGIN
   _F:=FIO.Open(CreateFileName(Path, Ext));
   IF FIO.RdBin(_F, _Taxa, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   IO.WrStr('Number of taxa: ');
   IO.WrCard(_Taxa, 0);
   IO.WrLn;
   IO.WrLn;
   New(_Mat, SizeOfDiagMat(_Taxa));
   LoadFile(_Mat, SizeOfDiagMat(_Taxa))
END Load_Matrix;
```

## Tools

```
PROCEDURE GetCmdLine(VAR Name : FIO.PathStr);
BEGIN
   IO.WrStr("| BG_TOOL 8:");                                           IO.WrLn;
   IO.WrStr("────────────");                                           IO.WrLn;
   IO.WrStr("BG_CS3 v1.01 Copyright (c) 1990 by J. Savary & J. Guex"); IO.WrLn;
   IO.WrStr('C3 and S3 in G*');                                        IO.WrLn;
   IO.WrLn;

   IF ParamCount()=0 THEN
      IO.WrStr("Usage: BG_CS3 <file> [>file | LPT1:]");
      IO.WrLn;
      HALT
   END;

   ParamStr(Name, 1);
   Append(Name, '.');
   Name:=CreateFileName(Name, 'BGD');
   IO.WrStr(Name); IO.WrLn;
   IF NOT FIO.Exists(Name) THEN
      IO.WrStr("File not found !"); IO.WrLn;
      HALT
   END;

   ParamStr(Name, 1);
   Append(Name, '.')
END GetCmdLine;
```

## List of $S_3$ and $C_3$

```
PROCEDURE A(i, j : CARDINAL) : BOOLEAN;
VAR mat : INT_PTR;
BEGIN
   IF i=j THEN RETURN FALSE END;
   mat:=DiagIndex(_Mat, i, j, _Taxa);
   IF mat^=EDGE THEN
      RETURN FALSE
   ELSIF i<j THEN
      RETURN mat^<0
   ELSE
      RETURN 0<mat^
   END
END A;

PROCEDURE E(i, j : CARDINAL) : BOOLEAN;
VAR mat : INT_PTR;
BEGIN
   IF i=j THEN RETURN FALSE END;
   mat:=DiagIndex(_Mat, i, j, _Taxa);
   RETURN mat^=EDGE
END E;

PROCEDURE WrA(i : CARDINAL);
BEGIN
   INC(_Count[i]);
   IO.WrCard(i, 3);
   IO.WrChar('>')
END WrA;
```

```
PROCEDURE WrE(i : CARDINAL);
BEGIN
   INC(_Count[i]);
   IO.WrCard(i, 3);
   IO.WrChar('-')
END WrE;

PROCEDURE IncArc(i, j : CARDINAL);
VAR a : INT_PTR;
BEGIN
   a:=DiagIndex(_Arc, i, j, _Taxa);
   IF i<j THEN
      DEC(a^)
   ELSE
      INC(a^)
   END
END IncArc;

PROCEDURE ListCS;
VAR i, j, k : CARDINAL;
    ok      : BOOLEAN;
BEGIN
   FOR i:=1 TO _Taxa-2 DO
      FOR j:=i+1 TO _Taxa-1 DO
         FOR k:=j+1 TO _Taxa DO
            ok:=FALSE;
            IF A(i,j) THEN
               IF A(j,k) THEN
                  IF A(k,i) THEN
                     ok:=TRUE;
                     IO.WrStr('C3:');
                     WrA(i);
                     WrA(j);
                     WrA(k);
                     IncArc(i,j);
                     IncArc(j,k);
                     IncArc(k,i)
                  ELSIF E(k,i) THEN
                     ok:=TRUE;
                     IO.WrStr('S3:');
                     WrA(i);
                     WrA(j);
                     WrE(k);
                     IncArc(i,j);
                     IncArc(j,k)
                  END
               ELSIF E(j,k) AND A(k,i) THEN
                  ok:=TRUE;
                  IO.WrStr('S3:');
                  WrA(i);
                  WrE(j);
                  WrA(k);
                  IncArc(i,j);
                  IncArc(k,i)
               END
            ELSIF E(i,j) AND A(j,k) AND A(k,i) THEN
               ok:=TRUE;
               IO.WrStr('S3:');
               WrE(i);
               WrA(j);
               WrA(k);
               IncArc(j,k);
               IncArc(k,i)
            END;
            IF ok=FALSE THEN
               IF A(i,k) THEN
                  IF A(k,j) THEN
                     IF A(j,i) THEN
                        ok:=TRUE;
                        IO.WrStr('C3:');
                        WrA(i);
                        WrA(k);
                        WrA(j);
                        IncArc(i,k);
                        IncArc(k,j);
                        IncArc(j,i)
                     ELSIF E(j,i) THEN
                        ok:=TRUE;
                        IO.WrStr('S3:');
                        WrA(i);
                        WrA(k);
                        WrE(j);
                        IncArc(i,k);
                        IncArc(k,j)
                     END
                  ELSIF E(k,j) AND A(j,i) THEN
                     ok:=TRUE;
                     IO.WrStr('S3:');
                     WrA(i);
                     WrE(k);
                     WrA(j);
                     IncArc(i,k);
                     IncArc(j,i)
                  END
               ELSIF E(i,k) AND A(k,j) AND A(j,i) THEN
                  ok:=TRUE;
                  IO.WrStr('S3:');
                  WrE(i);
                  WrA(k);
                  WrA(j);
                  IncArc(k,j);
                  IncArc(j,i)
               END
            END;
            IF ok THEN IO.WrLn END
         END
      END
   END
END ListCS;

PROCEDURE Vertices;
VAR v : CARDINAL;
BEGIN
   IO.WrLn;
   FOR v:=1 TO _Taxa DO
      IO.WrCard(v, 3);
      IO.WrChar(':');
      IO.WrCard(_Count[v], 5);
      IO.WrLn
   END
END Vertices;

PROCEDURE Arcs;
VAR i, j : CARDINAL;
    a    : INT_PTR;
BEGIN
   IO.WrLn;
   FOR i:=1 TO _Taxa-1 DO
      FOR j:=i+1 TO _Taxa DO
         a:=DiagIndex(_Arc, i, j, _Taxa);
         IF a^<>0 THEN
            IO.WrCard(i, 3);
            IF a^<0 THEN
               IO.WrChar('>')
            ELSE
               IO.WrChar('<')
            END;
            IO.WrCard(j, 3);
            IO.WrChar(':');
            IO.WrInt(ABS(a^), 5);
            IO.WrLn
         END
      END
   END
END Arcs;

BEGIN
   GetCmdLine(Filename);
   Load_Matrix(Filename, 'BGD');
   WordFill(ADR(_Count), 500, 0);
   New(_Arc, SizeOfDiagMat(_Taxa));
   ClearMemory(_Arc, SizeOfDiagMat(_Taxa));
   ListCS;
   Arcs;
   Vertices
END BG_CS3.
```

## Program BG_CSZ4

Function :   List of $C_4$, $S_4$, $S'_4$, $Z_4$, $Z'_4$ and $Z''_4$ in G*

```
MODULE BG_CSZ4;

IMPORT IO, FIO;

FROM Str    IMPORT Append, Caps, Pos;
FROM Lib    IMPORT IncAddr, AddAddr, ParamCount, ParamStr, WordFill;
FROM BG_ERR IMPORT Error;
FROM BG_MEM IMPORT INT_PTR, New, Release, SizeOfDiagMat, DiagIndex, EDGE,
                   ClearMemory;

CONST BUFFER = 0FFF8H;

TYPE EXTENSION = ARRAY[0..3] OF CHAR;
     ARCTYPE   = (Nul, Edge, Forward, Backward);
     ARCSET    = SET OF ARCTYPE;

VAR _Taxa      : CARDINAL;
    _Mat, _Arc : INT_PTR;
    _Flag      : BITSET;
    _F         : FIO.File;
    _Count     : ARRAY[1..500] OF CARDINAL;
    Filename   : FIO.PathStr;
```

### Files management

```
PROCEDURE CreateFileName(Name : FIO.PathStr;
                         Ext  : EXTENSION) : FIO.PathStr;
BEGIN
   Append(Name, Ext);
   RETURN Name
END CreateFileName;

PROCEDURE LoadFile(ptr  : ADDRESS;
                   size : LONGCARD);
VAR read : CARDINAL;
BEGIN
   WHILE BUFFER<size DO
      read:=FIO.RdBin(_F, ptr^, BUFFER);
      IncAddr(ptr, read);
      DEC(size, LONGCARD(read))
   END;
   read:=FIO.RdBin(_F, ptr^, CARDINAL(size));
   IF read<CARDINAL(size) THEN Error(100) END;
   FIO.Close(_F)
END LoadFile;

PROCEDURE Load_Matrix(Path : FIO.PathStr;
                      Ext  : EXTENSION);
BEGIN
   _F:=FIO.Open(CreateFileName(Path, Ext));
   IF FIO.RdBin(_F, _Taxa, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   IO.WrStr('Number of taxa: ');
   IO.WrCard(_Taxa, 0);
   IO.WrLn;
   IO.WrLn;
   New(_Mat, SizeOfDiagMat(_Taxa));
   LoadFile(_Mat, SizeOfDiagMat(_Taxa))
END Load_Matrix;
```

### Tools

```
PROCEDURE GetCmdLine(VAR Name : FIO.PathStr);

   PROCEDURE Usage;
   BEGIN
      IO.WrStr('Usage: BG_CSZ4 <file> [-CSZ] [>file | LPT1:]'); IO.WrLn;
      IO.WrStr('                       -C : C4');                IO.WrLn;
      IO.WrStr("                       -S : S4, S'4");           IO.WrLn;
      IO.WrStr("                       -Z : Z4, Z'4, Z");
                                            IO.WrStr('"4');      IO.WrLn;
      IO.WrStr('                       default : -CSZ');         IO.WrLn;
      HALT
   END Usage;
```

```
VAR flag : ARRAY[0..4] OF CHAR;
BEGIN
   IO.WrStr("| BG_TOOL 10:");                                               IO.WrLn;
   IO.WrStr("———————————");                                              IO.WrLn;
   IO.WrStr("BG_CSZ4 v1.01 Copyright (c) 1990 by J. Savary & J. Guex"); IO.WrLn;
   IO.WrStr("C4, S4, S'4, Z4, Z'4, Z"); IO.WrStr('"4 of G*');            IO.WrLn;
   IO.WrLn;

   IF ParamCount()=0 THEN Usage END;

   ParamStr(Name, 1);
   Append(Name, '.');
   Name:=CreateFileName(Name, 'BGD');
   IO.WrStr(Name); IO.WrLn;
   IF NOT FIO.Exists(Name) THEN
      IO.WrStr("File not found !"); IO.WrLn;
      HALT
   END;

   ParamStr(Name, 1);
   Append(Name, '.');

   _Flag:={};
   ParamStr(flag, 2);
   IF flag[0]='-' THEN
      Caps(flag);
      IO.WrStr('Flag: ');
      IO.WrStr(flag);
      IO.WrLn;
      IF Pos(flag, 'C')<>MAX(CARDINAL) THEN INCL(_Flag, 0) END;
      IF Pos(flag, 'S')<>MAX(CARDINAL) THEN INCL(_Flag, 1) END;
      IF Pos(flag, 'Z')<>MAX(CARDINAL) THEN INCL(_Flag, 2) END;
      IF _Flag={} THEN Usage END
   ELSE
      _Flag:={0,1,2}
   END
END GetCmdLine;
```

## List of $C_4$, $S_4$, $S'_4$, $Z_4$, $Z'_4$ and $Z''_4$

```
PROCEDURE ArcType(i, j : CARDINAL) : ARCTYPE;
VAR mat : INT_PTR;
BEGIN
   mat:=DiagIndex(_Mat, i, j, _Taxa);
   IF mat^=0 THEN
      RETURN Nul
   ELSIF mat^=EDGE THEN
      RETURN Edge
   ELSIF i<j THEN
      IF mat^<0 THEN
         RETURN Forward
      ELSE
         RETURN Backward
      END
   ELSE
      IF 0<mat^ THEN
         RETURN Forward
      ELSE
         RETURN Backward
      END
   END
END ArcType;

PROCEDURE IncArc(i, j : CARDINAL);
VAR a : INT_PTR;
BEGIN
   a:=DiagIndex(_Arc, i, j, _Taxa);
   IF i<j THEN
      DEC(a^)
   ELSE
      INC(a^)
   END
END IncArc;

PROCEDURE WrA(i : CARDINAL);
BEGIN
   INC(_Count[i]);
   IO.WrCard(i, 3);
   IO.WrChar('>')
END WrA;
```

```
PROCEDURE WrA2(i, j : CARDINAL);
BEGIN
   IO.WrCard(i, 3);
   IO.WrChar('>');
   IO.WrCard(j, 3);
   IO.WrChar(';')
END WrA2;

PROCEDURE WrE(i : CARDINAL);
BEGIN
   INC(_Count[i]);
   IO.WrCard(i, 3);
   IO.WrChar('-')
END WrE;

PROCEDURE WrE2(i, j : CARDINAL);
BEGIN
   IO.WrCard(i, 3);
   IO.WrChar('-');
   IO.WrCard(j, 3);
   IO.WrChar(';')
END WrE2;

PROCEDURE CA(i,j,k,l : CARDINAL);
BEGIN
   IF NOT (0 IN _Flag) THEN RETURN END;
   IF (ArcType(i,k) IN ARCSET{Edge, Forward, Backward}) OR
      (ArcType(j,l) IN ARCSET{Edge, Forward, Backward}) THEN RETURN END;
   IncArc(i,j);
   IncArc(j,k);
   IncArc(k,l);
   IncArc(l,i);
   IO.WrStr('C4 :');
   WrA(i);
   WrA(j);
   WrA(k);
   WrA(l);
   IO.WrLn
END CA;

PROCEDURE CB(i,j,k,l : CARDINAL);
BEGIN
   IF NOT (1 IN _Flag) THEN RETURN END;
   IF (ArcType(i,k) IN ARCSET{Edge, Forward, Backward}) OR
      (ArcType(j,l) IN ARCSET{Edge, Forward, Backward}) THEN RETURN END;
   IncArc(i,j);
   IncArc(j,k);
   IncArc(k,l);
   IO.WrStr('S4 :');
   WrA(i);
   WrA(j);
   WrA(k);
   WrE(l);
   IO.WrLn
END CB;

PROCEDURE CD(i,j,k,l : CARDINAL);
VAR ik, jl : ARCTYPE;
BEGIN
   ik:=ArcType(i,k);
   jl:=ArcType(j,l);
   IF (ik IN ARCSET{Forward, Backward}) OR
      (jl IN ARCSET{Forward, Backward}) THEN RETURN END;
   IF (ik=Edge) AND (jl<>Edge) THEN
      IF NOT (1 IN _Flag) THEN RETURN END;
      IncArc(i,j);
      IncArc(k,l);
      IO.WrStr("S'4:");
      WrA(i);
      WrE(j);
      WrA(k);
      WrE(l);
      IO.WrChar(';');
      WrE2(i,k);
      IO.WrLn
   ELSIF (ik<>Edge) AND (jl=Edge) THEN
      IF NOT (1 IN _Flag) THEN RETURN END;
      IncArc(i,j);
      IncArc(k,l);
      IO.WrStr("S'4:");
      WrA(i);
      WrE(j);
      WrA(k);
      WrE(l);
      IO.WrChar(';');
      WrE2(j,l);
      IO.WrLn
   ELSIF (ik=Edge) AND (jl=Edge) THEN
      IF NOT (2 IN _Flag) THEN RETURN END;
      IncArc(i,j);
      IncArc(k,l);
      IO.WrStr('Z"4:');
      WrE(i);
      WrE(k);
      WrE(j);
      WrE(l);
      IO.WrChar(';');
      WrA2(i,j);
      WrA2(k,l);
      IO.WrLn
   ELSE
      IF NOT (1 IN _Flag) THEN RETURN END;
      IO.WrStr("S4 :");
      WrA(i);
      WrE(j);
      WrA(k);
      WrE(l);
      IO.WrLn
   END
END CD;

PROCEDURE CF(i,j,k,l : CARDINAL);
VAR ik, jl : ARCTYPE;
BEGIN
   IF NOT (2 IN _Flag) THEN RETURN END;
   ik:=ArcType(i,k);
   jl:=ArcType(j,l);
   IF (ik=Edge) OR (jl=Edge) THEN RETURN END;
   IF (ik=Nul) AND (jl=Nul) THEN
      IO.WrStr('Z4 :');
      WrE(i);
      WrE(j);
      WrE(k);
      WrE(l)
   ELSIF (ik=Forward) AND (jl=Nul) THEN
      IncArc(i,k);
      IO.WrStr("Z'4:");
      WrE(i);
      WrE(j);
      WrE(k);
      WrE(l);
      IO.WrChar(';');
      WrA2(i,k)
   ELSIF (ik=Backward) AND (jl=Nul) THEN
      IncArc(k,i);
      IO.WrStr("Z'4:");
      WrE(i);
      WrE(j);
      WrE(k);
      WrE(l);
      IO.WrChar(';');
      WrA2(k,i)
   ELSIF (ik=Nul) AND (jl=Forward) THEN
      IncArc(j,l);
      IO.WrStr("Z'4:");
      WrE(i);
      WrE(j);
      WrE(k);
      WrE(l);
      IO.WrChar(';');
      WrA2(j,l)
   ELSIF (ik=Nul) AND (jl=Backward) THEN
      IncArc(l,j);
      IO.WrStr("Z'4:");
      WrE(i);
      WrE(j);
      WrE(k);
      WrE(l);
      IO.WrChar(';');
      WrA2(l,j)
   ELSE
      IO.WrStr('Z"4:');
      WrE(i);
      WrE(j);
      WrE(k);
      WrE(l);
      IO.WrChar(';');
      IF ik=Forward THEN
         IncArc(i,k);
         WrA2(i,k)
      END;
      IF ik=Backward THEN
         IncArc(k,i);
         WrA2(k,i)
      END;
      IF jl=Forward THEN
         IncArc(j,l);
         WrA2(j,l)
      END;
      IF jl=Backward THEN
         IncArc(l,j);
         WrA2(l,j)
      END
   END;
   IO.WrLn
END CF;

PROCEDURE Cast(i,j,k,l : CARDINAL);
VAR ij, jk, kl, li : ARCTYPE;
BEGIN
   ij:=ArcType(i,j);
   jk:=ArcType(j,k);
   kl:=ArcType(k,l);
   li:=ArcType(l,i);
   IF ij=Forward THEN
      IF jk=Forward THEN
         IF kl=Forward THEN
            IF li=Forward THEN
               CA(i,j,k,l)
            ELSIF li=Edge THEN
               CB(i,j,k,l)
            END
         ELSIF (kl=Edge) AND (li=Forward) THEN
            CB(l,i,j,k)
         END
      ELSIF jk=Edge THEN
         IF kl=Forward THEN
            IF li=Forward THEN
               CB(k,l,i,j)
            ELSIF li=Edge THEN
               CD(i,j,k,l)
            END
         END
      END
   ELSIF ij=Edge THEN
      IF jk=Forward THEN
         IF kl=Forward THEN
            IF li=Forward THEN
               CB(j,k,l,i)
            END
         ELSIF (kl=Edge) AND (li=Forward) THEN
            CD(j,k,l,i)
         END
      ELSIF (jk=Edge) AND (kl=Edge) AND (li=Edge) THEN
         CF(i,j,k,l)
      END
   END
END Cast;

PROCEDURE List;
VAR i, j, k, l              : CARDINAL;
    ij, jk, kl, li, ik, jl : ARCTYPE;
    fw, bw                 : ARCSET;
BEGIN
   fw:=ARCSET{Edge, Forward };
   bw:=ARCSET{Edge, Backward};
   FOR i:=1 TO _Taxa-3 DO
      FOR j:=i+1 TO _Taxa-2 DO
         ij:=ArcType(i,j);
         FOR k:=j+1 TO _Taxa-1 DO
            jk:=ArcType(j,k);
            ik:=ArcType(i,k);
            FOR l:=k+1 TO _Taxa DO
               kl:=ArcType(k,l);
               li:=ArcType(l,i);
               jl:=ArcType(j,l);
               IF (ij IN fw) AND (jk IN fw) AND
                  (kl IN fw) AND (li IN fw) THEN
                  Cast(i,j,k,l)
               ELSIF (ij IN bw) AND (jk IN bw) AND
                     (kl IN bw) AND (li IN bw) THEN
                  Cast(i,l,k,j)
               ELSIF (ij IN fw) AND (jl IN fw) AND
                     (kl IN bw) AND (ik IN bw) THEN
                  Cast(i,j,l,k)
               ELSIF (ik IN fw) AND (kl IN fw) AND
                     (jl IN bw) AND (ij IN bw) THEN
                  Cast(i,k,l,j)
               ELSIF (li IN bw) AND (jl IN bw) AND
                     (jk IN fw) AND (ik IN bw) THEN
                  Cast(i,l,j,k)
               ELSIF (ik IN fw) AND (jk IN bw) AND
                     (kl IN fw) AND (li IN fw) THEN
                  Cast(i,k,j,l)
               END
            END
         END
      END
   END
END List;

PROCEDURE Vertices;
VAR v : CARDINAL;
BEGIN
   IO.WrLn;
   FOR v:=1 TO _Taxa DO
      IO.WrCard(v, 3);
      IO.WrChar(':');
      IO.WrCard(_Count[v], 5);
      IO.WrLn
   END
END Vertices;

PROCEDURE Arcs;
VAR i, j : CARDINAL;
    a    : INT_PTR;
BEGIN
   IO.WrLn;
   FOR i:=1 TO _Taxa-1 DO
      FOR j:=i+1 TO _Taxa DO
         a:=DiagIndex(_Arc, i, j, _Taxa);
         IF a^<>0 THEN
            IO.WrCard(i, 3);
            IF a^<0 THEN
               IO.WrChar('>')
            ELSE
               IO.WrChar('<')
            END;
            IO.WrCard(j, 3);
            IO.WrChar(':');
            IO.WrInt(ABS(a^), 5);
            IO.WrLn
         END
      END
   END
END Arcs;

BEGIN
   GetCmdLine(Filename);
   Load_Matrix(Filename, 'BGD');
   WordFill(ADR(_Count),500,0);
   New(_Arc, SizeOfDiagMat(_Taxa));
   ClearMemory(_Arc, SizeOfDiagMat(_Taxa));
   List;
   Arcs;
   Vertices
END BG_CSZ4.
```

## *Program BG_DIA*

Function : Taxa distribution expressed in terms of UAs; providing the diachronism of the datums

```
MODULE BG_DIA;

IMPORT IO, FIO;

FROM Str    IMPORT Append;
FROM Lib    IMPORT IncAddr, DecAddr, ParamCount, ParamStr;
FROM BG_ERR IMPORT Error;
FROM BG_MEM IMPORT New, Release, PointSet, ClearMemory, SetRange, GetRange,
                   SetSegment, SET_PTR, CORR_REC, CORR_PTR;

CONST BUFFER = 0FFF8H;

TYPE EXTENSION  = ARRAY[0..3] OF CHAR;

VAR Taxa, NLev   : CARDINAL;
    Levels, Corr : ADDRESS;
    _F           : FIO.File;
    Filename     : FIO.PathStr;
```

### Files management

```
PROCEDURE CreateFileName(Name : FIO.PathStr;
                         Ext  : EXTENSION) : FIO.PathStr;
BEGIN
   Append(Name, Ext);
   RETURN Name
END CreateFileName;

PROCEDURE LoadFile(ptr  : ADDRESS;
                   size : LONGCARD);
VAR read : CARDINAL;
BEGIN
   WHILE BUFFER<size DO
      read:=FIO.RdBin(_F, ptr^, BUFFER);
      IncAddr(ptr, read);
      DEC(size, LONGCARD(read))
   END;
   read:=FIO.RdBin(_F, ptr^, CARDINAL(size));
   IF read<CARDINAL(size) THEN Error(100) END;
   FIO.Close(_F)
END LoadFile;

PROCEDURE Load_Levels(Path        : FIO.PathStr;
                      Ext         : EXTENSION;
                  VAR Taxa, NLev  : CARDINAL;
                  VAR Level       : ADDRESS);
VAR NSec : CARDINAL;
BEGIN
   _F:=FIO.Open(CreateFileName(Path, Ext));
   IF FIO.RdBin(_F, Taxa, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   IF FIO.RdBin(_F, NSec, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   IF FIO.RdBin(_F, NLev, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   SetRange(Taxa);
   IO.WrStr('Number of taxa: ');
   IO.WrCard(Taxa, 0);
   IO.WrLn;
   IO.WrStr('Number of levels: ');
   IO.WrCard(NLev, 0);
   IO.WrLn;
   IO.WrLn;
   New(Level, LONGCARD(NLev)*LONGCARD(GetRange())+SIZE(CARDINAL));
   LoadFile(Level, LONGCARD(NLev)*LONGCARD(GetRange()))
END Load_Levels;

PROCEDURE Load_Corr(Path : FIO.PathStr;
                    Ext  : EXTENSION;
                VAR NLev : CARDINAL;
                VAR Corr : ADDRESS);
BEGIN
   _F:=FIO.Open(CreateFileName(Path, Ext));
   IF FIO.RdBin(_F, NLev, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   New(Corr, LONGCARD(NLev)*SIZE(CORR_REC));
   LoadFile(Corr, LONGCARD(NLev)*SIZE(CORR_REC))
END Load_Corr;
```

## Tools

```
PROCEDURE GetCmdLine(VAR Name : FIO.PathStr);
BEGIN
   IO.WrStr("| BG_TOOL 7:");                                              IO.WrLn;
   IO.WrStr("————————");                                                  IO.WrLn;
   IO.WrStr("BG_DIA v1.02 Copyright (c) 1990 by J. Savary & J. Guex"); IO.WrLn;
   IO.WrStr('Diachronism');                                               IO.WrLn;
   IO.WrLn;

   IF ParamCount()=0 THEN
      IO.WrStr("Usage: BG_DIA <file> [>file | LPT1:]");
      IO.WrLn;
      HALT
   END;

   ParamStr(Name, 1);
   Append(Name, '.');
   Name:=CreateFileName(Name, 'BGA');
   IO.WrStr(Name); IO.WrLn;
   IF NOT FIO.Exists(Name) THEN
      IO.WrStr("File not found !"); IO.WrLn;
      HALT
   END;
   ParamStr(Name, 1);
   Append(Name, '.');
   Name:=CreateFileName(Name, 'BGK');
   IO.WrStr(Name); IO.WrLn;
   IF NOT FIO.Exists(Name) THEN
      IO.WrStr("File not found !"); IO.WrLn;
      HALT
   END;

   ParamStr(Name, 1);
   Append(Name, '.')
END GetCmdLine;
```

## Diachronism

```
PROCEDURE Display(NLev   : CARDINAL;
                  Levels : SET_PTR);
VAR L : SET_PTR;
    l : CARDINAL;
BEGIN
   SetSegment(Levels);
   FOR l:=1 TO NLev DO
      L:=PointSet(l-1);
      IO.WrCard(L^.H.Section, 3);
      IO.WrCard(L^.H.Level  , 3);
      IO.WrLn
   END
END Display;

PROCEDURE SearchCorr(NLev : CARDINAL;
                     Corr : CORR_PTR;
                     sec, lev : CARDINAL;
                 VAR Fau, Lau : CARDINAL);
BEGIN
   DecAddr(Corr, SIZE(CORR_REC));
   REPEAT
      IncAddr(Corr, SIZE(CORR_REC));
   UNTIL (Corr^.Section=sec) AND (Corr^.Level=lev);
   Fau:=Corr^.Fau;
   Lau:=Corr^.Lau
END SearchCorr;

PROCEDURE Diachronism1(Taxa, NLev   : CARDINAL;
                       Levels, Corr : ADDRESS);
VAR t, l, fau, lau, nl, lmax,
    FadFau, FadLau, LadFau, LadLau : CARDINAL;
    L                              : SET_PTR;
    ok                             : BOOLEAN;
BEGIN
   IO.WrStr('Local diachronism:'); IO.WrLn; IO.WrLn;
   SetSegment(Levels);
   nl:=1;
   REPEAT
      L:=PointSet(nl);
      lmax:=L^.H.Level;
      IO.WrStr('Section ');
      IO.WrCard(L^.H.Section, 0);
      IO.WrLn;
      FOR t:=1 TO Taxa DO
         ok:=FALSE;
         FOR l:=0 TO lmax-1 DO
            L:=PointSet(nl+1);
            IF (t-1) IN L^.Set THEN
               SearchCorr(NLev, Corr, L^.H.Section, L^.H.Level, fau, lau);
               IF ok=FALSE THEN
                  ok:=TRUE;
                  LadFau:=fau;
                  LadLau:=lau
               END;
               FadFau:=fau;
               FadLau:=lau
            END
         END;
         IF ok THEN
            IO.WrCard(t, 3);
            IO.WrChar(':');
            IO.WrCard(FadFau, 3);
            IO.WrChar('-');
            IO.WrCard(FadLau, 3);
            IO.WrCard(LadFau, 4);
            IO.WrChar('-');
            IO.WrCard(LadLau, 3);
            IO.WrLn
         END
      END;
      IO.WrLn;
      INC(nl, lmax)
   UNTIL NLev<nl
END Diachronism1;

PROCEDURE Diachronism2(Taxa, NLev   : CARDINAL;
                       Levels, Corr : ADDRESS);
VAR t, l, fau, lau, nl, lmax,
    FadFau, FadLau, LadFau, LadLau,
    FadFauMin, FadLauMin, FadFauMax, FadLauMax,
    LadFauMin, LadLauMin, LadFauMax, LadLauMax,
    FadSMin, FadSMax, LadSMin, LadSMax         : CARDINAL;
    L                                          : SET_PTR;
    ok                                         : BOOLEAN;
BEGIN
   IO.WrStr('Total diachronism:'); IO.WrLn; IO.WrLn;
   IO.WrStr('  x|FAD min [ s ] FAD max [ s ] = Da');
   IO.WrStr( ' | LAD min [ s ] LAD max [ s ] = Dd |  Dt'); IO.WrLn;
   IO.WrStr('──┼─────────────────────────────────');
   IO.WrStr( '─┼──────────────────────────────────┼───'); IO.WrLn;
   SetSegment(Levels);
   FOR t:=1 TO Taxa` DO
      nl:=1;
      FadFauMin:=MAX(CARDINAL);
      FadLauMax:=0;
      LadFauMin:=MAX(CARDINAL);
      LadLauMax:=0;
      FadSMin:=0;
      FadSMax:=0;
      LadSMin:=0;
      LadSMax:=0;
      REPEAT
         ok:=FALSE;
         L:=PointSet(nl);
         lmax:=L^.H.Level;
         FOR l:=0 TO lmax-1 DO
            L:=PointSet(nl+l);
            IF (t-1) IN L^.Set THEN
               SearchCorr(NLev, Corr, L^.H.Section, L^.H.Level, fau, lau);
               IF ok=FALSE THEN
                  ok:=TRUE;
                  LadFau:=fau;
                  LadLau:=lau
               END;
               FadFau:=fau;
               FadLau:=lau
            END
         END;
         IF ok THEN
            IF FadFau=FadFauMin THEN
               IF FadLau<FadLauMin THEN
                  FadLauMin:=FadLau;
                  FadSMin:=L^.H.Section
               END
            ELSIF FadFau<FadFauMin THEN
               FadFauMin:=FadFau;
               FadLauMin:=FadLau;
               FadSMin:=L^.H.Section
            END;
            IF FadLauMax=FadLau THEN
               IF FadFauMax<FadFau THEN
                  FadFauMax:=FadFau;
                  FadSMax:=L^.H.Section
               END
            ELSIF FadLauMax<FadLau THEN
               FadFauMax:=FadFau;
               FadLauMax:=FadLau;
               FadSMax:=L^.H.Section
            END;
            IF LadFau=LadFauMin THEN
               IF LadLau<LadLauMin THEN
                  LadLauMin:=LadLau;
                  LadSMin:=L^.H.Section
               END
            ELSIF LadFau<LadFauMin THEN
               LadFauMin:=LadFau;
               LadLauMin:=LadLau;
               LadSMin:=L^.H.Section
            END;
            IF LadLauMax=LadLau THEN
               IF LadFauMax<LadFau THEN
                  LadFauMax:=LadFau;
                  LadSMax:=L^.H.Section
               END
            ELSIF LadLauMax<LadLau THEN
               LadFauMax:=LadFau;
               LadLauMax:=LadLau;
               LadSMax:=L^.H.Section
            END
         END;
         INC(nl, lmax)
      UNTIL NLev<nl;
      IF 0<FadSMin THEN
         IO.WrCard(t, 3);
         IO.WrChar('_');
         IO.WrCard(FadFauMin, 3);
         IO.WrChar('-');
         IO.WrCard(FadLauMin, 3);
         IO.WrStr(' [');
         IO.WrCard(FadSMin, 3);
         IO.WrChar(']');
         IO.WrCard(FadFauMax, 4);
         IO.WrChar('-');
         IO.WrCard(FadLauMax, 3);
         IO.WrStr(' [');
         IO.WrCard(FadSMax, 3);
         IO.WrStr('] =');
         IO.WrCard(FadFauMax-FadLauMin, 3);
         IO.WrStr('  _');
         IO.WrCard(LadFauMin, 4);
         IO.WrChar('-');
         IO.WrCard(LadLauMin, 3);
         IO.WrStr(' [');
         IO.WrCard(LadSMin, 3);
         IO.WrChar(']');
         IO.WrCard(LadFauMax, 4);
         IO.WrChar('-');
         IO.WrCard(LadLauMax, 3);
         IO.WrStr(' [');
         IO.WrCard(LadSMax, 3);
         IO.WrStr('] =');
         IO.WrCard(LadFauMax-LadLauMin, 3);
         IO.WrStr('  _');
         IO.WrCard(FadFauMax-FadLauMin+LadFauMax-LadLauMin, 4);
         IO.WrLn
      END
   END
END Diachronism2;

BEGIN
   GetCmdLine(Filename);
   Load_Levels(Filename, 'BGA', Taxa, NLev, Levels);
   Load_Corr(Filename, 'BGK', NLev, Corr);
   Diachronism1(Taxa, NLev, Levels, Corr);
   Diachronism2(Taxa, NLev, Levels, Corr)
END BG_DIA.
```

## *Program BG_UNI*

Function : List of the unions of MRH forming maximal cliques

```
MODULE BG_UNI;

IMPORT IO, FIO;

FROM Str    IMPORT Append;
FROM Lib    IMPORT IncAddr, AddAddr, ParamCount, ParamStr, QSort;
FROM BG_ERR IMPORT Error;
FROM BG_MEM IMPORT INT_PTR, WORD_LIST, New, Release, PointSet, ClearMemory,
                   Difference, SizeOfDiagMat, DiagIndex, EDGE, SetRange,
                   GetRange, SetSegment, Min, Max, PutInSet, SET_PTR, SwapSet,
                   HEAD_PTR, GetSegment, HEAD_REC, WithSetDo, Subtraction;

CONST BUFFER = 0FFF8H;

TYPE EXTENSION  = ARRAY[0..3] OF CHAR;

VAR Taxa, NLev      : CARDINAL;
    Cliques         : ADDRESS;
    Mat             : INT_PTR;
    _F              : FIO.File;
    Filename        : FIO.PathStr;
```

### Files management

```
PROCEDURE CreateFileName(Name  : FIO.PathStr;
                         Ext   : EXTENSION) : FIO.PathStr;
BEGIN
   Append(Name, Ext);
   RETURN Name
END CreateFileName;

PROCEDURE LoadFile(ptr  : ADDRESS;
                   size : LONGCARD);
VAR read : CARDINAL;
BEGIN
   WHILE BUFFER<size DO
      read:=FIO.RdBin(_F, ptr^, BUFFER);
      IncAddr(ptr, read);
      DEC(size, LONGCARD(read))
   END;
   read:=FIO.RdBin(_F, ptr^, CARDINAL(size));
   IF read<CARDINAL(size) THEN Error(100) END;
   FIO.Close(_F)
END LoadFile;

PROCEDURE Load_Levels(Path        : FIO.PathStr;
                      Ext         : EXTENSION;
                  VAR Taxa, NLev  : CARDINAL;
                  VAR Level       : ADDRESS);
VAR NSec : CARDINAL;
BEGIN
   _F:=FIO.Open(CreateFileName(Path, Ext));
   IF FIO.RdBin(_F, Taxa, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   IF FIO.RdBin(_F, NSec, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   IF FIO.RdBin(_F, NLev, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   SetRange(Taxa);
   IO.WrStr('Number of taxa: ');
   IO.WrCard(Taxa, 0);
   IO.WrLn;
   IO.WrStr('Number of HMR: ');
   IO.WrCard(NLev, 0);
   IO.WrLn;
   IO.WrLn;
   New(Level, LONGCARD(NLev)*LONGCARD(GetRange())+SIZE(CARDINAL));
   LoadFile(Level, LONGCARD(NLev)*LONGCARD(GetRange()))
END Load_Levels;

PROCEDURE Load_Matrix(Path : FIO.PathStr;
                      Ext  : EXTENSION;
                  VAR Mat  : ADDRESS;
                  VAR Size : CARDINAL);
BEGIN
   _F:=FIO.Open(CreateFileName(Path, Ext));
   IF FIO.RdBin(_F, Size, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(100) END;
   New(Mat, SizeOfDiagMat(Size));
   LoadFile(Mat, SizeOfDiagMat(Size))
END Load_Matrix;
```

## Tools

```
PROCEDURE GetCmdLine(VAR Name : FIO.PathStr);
BEGIN
   IO.WrStr("| BG_TOOL 6:");                                                     IO.WrLn;
   IO.WrStr("────────────");                                                     IO.WrLn;
   IO.WrStr("BG_UNI v1.01 Copyright (c) 1990 by J. Savary & J. Guex"); IO.WrLn;
   IO.WrStr('Unifications');                                                     IO.WrLn;
   IO.WrLn;

   IF ParamCount()=0 THEN
      IO.WrStr("Usage: BG_UNI <file> [>file | LPT1:]");
      IO.WrLn;
      HALT
   END;

   ParamStr(Name, 1);
   Append(Name, '.');
   Name:=CreateFileName(Name, 'BGC');
   IO.WrStr(Name); IO.WrLn;
   IF NOT FIO.Exists(Name) THEN
      IO.WrStr("File not found !"); IO.WrLn;
      HALT
   END;
   ParamStr(Name, 1);
   Append(Name, '.');
   Name:=CreateFileName(Name, 'BGD');
   IO.WrStr(Name); IO.WrLn;
   IF NOT FIO.Exists(Name) THEN
      IO.WrStr("File not found !"); IO.WrLn;
      HALT
   END;

   ParamStr(Name, 1);
   Append(Name, '.')
END GetCmdLine;
```

## Unions of non maximal cliques

```
PROCEDURE SortByCard(a, b : CARDINAL) : BOOLEAN;
VAR ptr1, ptr2 : HEAD_PTR;
BEGIN
   ptr1:=PointSet(a);
   ptr2:=PointSet(b);
   RETURN ptr1^.Cardinal>ptr2^.Cardinal
END SortByCard;

PROCEDURE Compaction(Taxa, NbLevel : CARDINAL) : CARDINAL;
VAR L, W, Levels, OldLevels, Range, S1, S2 : CARDINAL;
    Segment                                : ADDRESS;
    Ptr1, Ptr2                             : SET_PTR;
BEGIN
  IF 1<NbLevel THEN
      Levels:=NbLevel;
      OldLevels:=Levels-1;
      Segment:=GetSegment();
      Range:=((GetRange()-SIZE(HEAD_REC)) >> 1) - 1;
      QSort(Levels, SortByCard, SwapSet);
      REPEAT
         FOR L:=2 TO Levels DO
            W:=0;
            Ptr1:=PointSet(1);
            Ptr2:=PointSet(L);
            WHILE (W<=Range) AND ((Ptr2^.Bit[W]-Ptr1^.Bit[W])={}) DO INC(W) END;
            IF Range<W THEN
               Ptr2^.H.Cardinal:=0;
               DEC(Levels);
               DEC(NbLevel);
               S1:=Ptr1^.H.Section;
               S2:=Ptr2^.H.Section;
               EXCL(BITSET(S1), 15);
               EXCL(BITSET(S2), 15);
               IO.WrCard(S1, 3);
               IO.WrChar('.');
               IO.WrCard(Ptr1^.H.Level, 3);
               IF 15 IN BITSET(Ptr1^.H.Section) THEN
```

```
                  IO.WrChar('*')
               ELSE
                  IO.WrChar(' ')
               END;
               IO.WrChar('+');
               IO.WrCard(S2, 3);
               IO.WrChar('.');
               IO.WrCard(Ptr2^.H.Level, 3);
               IF 15 IN BITSET(Ptr2^.H.Section) THEN
                  IO.WrChar('*')
               END;
               IO.WrLn
            END
         END;
         DEC(Levels);
         SetSegment(PointSet(2));
         QSort(OldLevels, SortByCard, SwapSet);
         OldLevels:=Levels-1
      UNTIL Levels<2;
      SetSegment(Segment)
   END;
   RETURN NbLevel
END Compaction;
```

## Unions of equivalent cliques

### Construction of the neighborhood of each species, including the species itself

```
PROCEDURE Neighbours(VAR FirstN : ADDRESS;
                         Mat_A  : INT_PTR;
                         Taxa   : CARDINAL);
VAR t1, t2 : CARDINAL;
    Near   : SET_PTR;
    Mat    : INT_PTR;
BEGIN
   New(FirstN, LONGCARD(Taxa*GetRange()));
   ClearMemory(FirstN, LONGCARD(Taxa*GetRange()));
   FOR t1:=1 TO Taxa-1 DO
      FOR t2:=t1+1 TO Taxa DO
         Mat:=DiagIndex(Mat_A, t1, t2, Taxa);
         IF Mat^=EDGE THEN
            Near:=PointSet(t1);
            PutInSet(t2, Near);
            Near:=PointSet(t2);
            PutInSet(t1, Near)
         END
      END
   END;
   FOR t1:=1 TO Taxa DO
      Near:=PointSet(t1);
      PutInSet(t1, Near)
   END
END Neighbours;
```

### Union by means of the neighborhood

```
PROCEDURE Unification(Taxa, NLev  : CARDINAL;
                      Levels      : ADDRESS;
                      Mat_A       : INT_PTR)  : CARDINAL;
VAR NSec, Level, t : CARDINAL;
    Clique, Scrap  : SET_PTR;
    FirstN         : ADDRESS;
BEGIN
   New(Scrap, LONGCARD(GetRange()));
   Neighbours(FirstN, Mat_A, Taxa);
   Level:=1;
   REPEAT
      SetSegment(Levels);
      Clique:=PointSet(Level);
      SetSegment(FirstN);
      FOR t:=1 TO Taxa DO
         IF NOT (t-1 IN Clique^.Set) THEN
            IF WithSetDo(Subtraction, Clique, PointSet(t), Scrap)=0 THEN
               PutInSet(t, Clique);
               INCL(BITSET(Clique^.H.Section), 15)
            END
         END
      END;
      INC(Level)
   UNTIL NLev<Level;
   SetSegment(Levels);
   NLev:=Compaction(Taxa, NLev);
   Release(Scrap);
   RETURN NLev
END Unification;

PROCEDURE List(NLev : CARDINAL);
VAR k, s : CARDINAL;
    p    : SET_PTR;
BEGIN
   IO.WrLn;
   IO.WrStr('Ordinal list of cliques:');
   IO.WrLn;
   FOR k:=1 TO NLev DO
      p:=PointSet(k);
      s:=p^.H.Section;
      EXCL(BITSET(s), 15);
      IO.WrCard(k, 3);
      IO.WrStr(': ');
      IO.WrCard(s, 0);
      IO.WrChar('.');
      IO.WrCard(p^.H.Level, 0);
      IF 15 IN BITSET(p^.H.Section) THEN
         IO.WrChar('*')
      END;
      IO.WrLn
   END;
   IO.WrCard(NLev, 0);
   IO.WrStr(' cliques');
   IO.WrLn
END List;

BEGIN
   GetCmdLine(Filename);
   Load_Levels(Filename, 'BGC', Taxa, NLev, Cliques);
   Load_Matrix(Filename, 'BGD', Mat, Taxa);
   NLev:=Unification(Taxa, NLev, Cliques, Mat);
   List(NLev)
END BG_UNI.
```

## *Program BG_T01*

### Function : geofrequency

```
MODULE BG_T01;

(*# debug    (vid       => off) *)
(*# check    (nil_ptr   => off
              index     => off
              stack     => off
              overflow  => off
              range     => off) *)
(*# optimize (speed     => on
              cse       => on
              const     => on
              peep_hole => on
              jump      => on
              loop      => on
              alias     => off
              regass    => on
              stk_frame => on
              i286      => on
              x87       => off) *)

IMPORT IO;
FROM FIO      IMPORT WrCard, WrLn, WrStr;
FROM Lib      IMPORT Fill, IncAddr, WordFill;
FROM BG_TOOLS IMPORT End, Gauge, InitGauge, LoadLevels, _NLevel, _NTaxa,
                     OpenTextFile, SetPtr, _SizeOfSet, _T, TaxaRange,
                     WriteInOut, WriteTitle, WrTaxa;

CONST
   Tools   = 'T01';
   Program = 'BG_'+Tools+' v2.01: geofrequence';
```

```
VAR
   _Levels : FarADDRESS;
   _GF     : ARRAY TaxaRange OF CARDINAL;

PROCEDURE Count;
VAR s, l  : CARDINAL;
    t     : TaxaRange;
    met   : ARRAY TaxaRange OF BOOLEAN;
    level : SetPtr;
BEGIN
   InitGauge('Count', _NLevel);
   s:=0;
   level:=_Levels;
   FOR l:=1 TO _NLevel DO
      IF level^.H.Section<>s THEN
         s:=level^.H.Section;
         Fill(ADR(met), SIZE(met), ORD(TRUE))
      END;
      FOR t:=1 TO MAX(TaxaRange) DO
         IF ((t-1) IN level^.Set) AND met[t] THEN
            met[t]:=FALSE;
            INC(_GF[t])
         END
      END;
      IncAddr(level, _SizeOfSet);
      Gauge(l)
   END
END Count;

PROCEDURE Output;
VAR t : TaxaRange;
BEGIN
   InitGauge('Write', _NTaxa);
   FOR t:=1 TO _NTaxa DO
      WrTaxa(t);
      WrCard(_T, _GF[t], 6);
      WrLn(_T);
      Gauge(t)
   END
END Output;

BEGIN
   IO.WrStr();
   IO.WrLn;
   LoadLevels('BGB', _Levels);
   WriteInOut('BGB', Tools);
   WriteTitle;
   WordFill(ADR(_GF), 500, 0);
   Count;
   OpenTextFile(Tools, Program);
   Output;
   End
END BG_T01.
```

## *Program BG_T02*

Function :   observed edges

```
MODULE BG_T02;

(*# debug    (vid        => off) *)
(*# check    (nil_ptr    => off
              index      => off
              stack      => off
              overflow   => off
              range      => off) *)
(*# optimize (speed      => on
              cse        => on
              const      => on
              peep_hole  => on
              jump       => on
              loop       => on
              alias      => off
              regass     => on
              stk_frame  => on
              i286       => on
              x87       => off) *)

IMPORT IO;
FROM FIO      IMPORT WrCard, WrChar, WrInt, WrLn, WrLngCard, WrStr;
FROM Lib      IMPORT DecAddr, Fill, IncAddr;
FROM BG_TOOLS IMPORT ClearMem, DiagIndex, End, Gauge, GetMem, InitGauge, IntPtr,
                     LoadLevels, NewDiagMat, _NSection, _NTaxa, OpenTextFile,
                     PointSet, Set, SetPtr, _SizeOfSet, _T, TaxaRange, Union,
                     WriteInOut, WriteTitle, WrTaxa;

CONST
   Tools   = 'T02';
   Program = 'BG_'+Tools+' v2.01: observed edges';

VAR
   _Levels : FarADDRESS;
   _Mat    : IntPtr;

PROCEDURE Count;
VAR s, l, bottom, top : CARDINAL;
    t1, t2            : TaxaRange;
    level, edge       : SetPtr;
    a                 : IntPtr;
BEGIN
   GetMem(edge, LONGCARD(_SizeOfSet));
   bottom:=1;
   InitGauge('Count  ', _NSection);
   FOR s:=1 TO _NSection DO
```

Search the boudaries of the section

```
      top:=bottom;
      level:=PointSet(_Levels, bottom, _SizeOfSet);
      REPEAT
         INC(bottom);
         IncAddr(level, _SizeOfSet)
      UNTIL level^.H.Section<>s;

      FOR t1:=0 TO _NTaxa-2 DO
```

Search presence of species

```
         l:=bottom;
         level:=PointSet(_Levels, l, _SizeOfSet);
         REPEAT
            DEC(l);
            DecAddr(level, _SizeOfSet)
         UNTIL (l=top) OR (t1 IN level^.Set);
```

if species present

```
         IF t1 IN level^.Set THEN
            ClearMem(edge, LONGCARD(_SizeOfSet));
```

Union of the horizons containing the species

```
            REPEAT
               Set(Union, edge, level, edge);
               DEC(l);
               DecAddr(level, _SizeOfSet)
            UNTIL (l<top) OR NOT (t1 IN level^.Set);
```

Recording into the matrix

```
            FOR t2:=t1+1 TO _NTaxa-1 DO
               IF t2 IN edge^.Set THEN
                  a:=DiagIndex(_Mat, _NTaxa, t1+1, t2+1);
                  INC(a^)
               END
            END
         END
      END;
      Gauge(s)
   END
END Count;

PROCEDURE Output;
VAR t1, t2 : TaxaRange;
    n, sum : LONGCARD;
    t      : ARRAY TaxaRange OF BOOLEAN;
    mat    : IntPtr;
    edge   : ARRAY TaxaRange OF LONGCARD;
    ne     : ARRAY TaxaRange OF CARDINAL;
BEGIN
   InitGauge('Write 1', _NTaxa-1);
   mat:=_Mat;
   n:=0;
   sum:=0;
   Fill(ADR(t), _NTaxa, 0);
   FOR t1:=1 TO _NTaxa-1 DO
      FOR t2:=t1+1 TO _NTaxa DO
         IF mat^<>0 THEN
            INC(n);
            INC(sum, LONGCARD(mat^));
            t[t1]:=TRUE;
            t[t2]:=TRUE;
            WrTaxa(t1);
            WrChar(_T, CHR(32));
            WrTaxa(t2);
            WrInt(_T, mat^, 6);
            WrLn(_T)
         END;
         IncAddr(mat, 2)
      END;
      Gauge(t1)
   END;
   WrStr(_T, '"      " "_"');
   WrLngCard(_T, sum, 10);
   WrLn(_T);
   WrLn(_T);
   WrLngCard(_T, n, 0);
   WrStr(_T, ' "/" ');
   WrLngCard(_T, LONGCARD(_NTaxa)*LONGCARD(_NTaxa-1) DIV 2, 0);
   WrStr(_T, ' "edges between" ');
   n:=0;
   FOR t1:=1 TO _NTaxa DO
      IF t[t1] THEN INC(n) END
   END;
   WrLngCard(_T, n, 0);
   WrStr(_T, ' "species /" ');
   WrCard(_T, _NTaxa, 0);
   WrLn(_T);
   WrLn(_T);
   WrStr(_T, '"Reproducibilities and frequencies OF edges"');
   WrLn(_T);
   WrStr(_T, '             "R"   "f"');
   WrLn(_T);
   InitGauge('Write 2', _NTaxa);
   Fill(ADR(edge), SIZE(edge), 0);
   Fill(ADR(ne  ), SIZE(ne  ), 0);
   FOR t1:=1 TO _NTaxa DO
      FOR t2:=1 TO _NTaxa DO
         IF t1<>t2 THEN
            mat:=DiagIndex(_Mat, _NTaxa, t1, t2);
            INC(edge[t1], LONGCARD(mat^));
            IF mat^<>0 THEN INC(ne[t1]) END
         END
      END;
      WrTaxa(t1);
      WrLngCard(_T, edge[t1], 8);
      WrCard(_T, ne[t1], 6);
      WrLn(_T);
      Gauge(t1)
   END
 END Output;

 BEGIN
    IO.WrStr(Program);
    IO.WrLn;
    LoadLevels('BGB', _Levels);
    WriteInOut('BGB', Tools);
    WriteTitle;
    NewDiagMat(_Mat, _NTaxa);
    Count;
    OpenTextFile(Tools, Program);
    Output;
    End
 END BG_T02.
```

## *Program BG_T03*

Function : virtual edges

```
MODULE BG_T03;

(*# debug    (vid        => off) *)
(*# check    (nil_ptr    => off
              index      => off
              stack      => off
              overflow   => off
              range      => off) *)
(*# optimize (speed      => on
              cse        => on
              const      => on
              peep_hole  => on
              jump       => on
              loop       => on
              alias      => off
              regass     => on
              stk_frame  => on
              i286       => on
              x87        => off) *)

IMPORT IO;
FROM FIO      IMPORT WrChar, WrLn, WrLngCard, WrStr;
FROM Lib      IMPORT DecAddr, Fill, IncAddr;
FROM BG_TOOLS IMPORT ClearMem, DiagIndex, EDGE, End, Gauge, GetMem, InitGauge,
                     IntPtr, LoadLevels, NewDiagMat, _NSection, _NTaxa,
                     OpenTextFile, PointSet, Set, SetPtr, _SizeOfSet,
                     Subtraction, _T, TaxaRange, Union, WriteInOut, WriteTitle,
                     WrTaxa;

CONST
   Tools   = 'T03';
   Program = 'BG_'+Tools+' v2.01: virtual edges';

VAR
   _Levels : FarADDRESS;
   _Mat    : IntPtr;

PROCEDURE Count;
VAR s, l, bottom, top : CARDINAL;
    t1, t2            : TaxaRange;
    level, edge, arc  : SetPtr;
    a                 : IntPtr;
BEGIN
   GetMem(edge, LONGCARD(_SizeOfSet));
   GetMem(arc , LONGCARD(_SizeOfSet));
   bottom:=1;
   InitGauge('Count', _NSection);
   FOR s:=1 TO _NSection DO
```

Search boundaries of the section

```
      top:=bottom;
      level:=PointSet(_Levels, bottom, _SizeOfSet);
      REPEAT
         INC(bottom);
         IncAddr(level, _SizeOfSet)
      UNTIL level^.H.Section<>s;

      FOR t1:=0 TO _NTaxa-2 DO
```

Search presence of species

```
         l:=bottom;
         level:=PointSet(_Levels, l, _SizeOfSet);
         REPEAT
            DEC(l);
            DecAddr(level, _SizeOfSet)
         UNTIL (l=top) OR (t1 IN level^.Set);
```

if species present

```
         IF t1 IN level^.Set THEN
            ClearMem(edge, LONGCARD(_SizeOfSet));
            ClearMem(arc , LONGCARD(_SizeOfSet));
```

Union of the horizons containing the species

```
            REPEAT
               Set(Union, edge, level, edge);
               DEC(l);
               DecAddr(level, _SizeOfSet)
            UNTIL (l<top) OR NOT (t1 IN level^.Set);
```

Union of the horizons above

```
            WHILE top<=l DO
               Set(Union, arc, level, arc);
               DEC(l);
               DecAddr(level, _SizeOfSet)
            END;

            Set(Subtraction, arc, edge, arc);
```

Recording the edges into the matrix

```
            FOR t2:=t1+1 TO _NTaxa-1 DO
               IF t2 IN edge^.Set THEN
                  a:=DiagIndex(_Mat, _NTaxa, t1+1, t2+1);
                  a^:=EDGE
               END
            END;
```

Search for virtual edges

```
            FOR t2:=0 TO _NTaxa-1 DO
               IF (t1<>t2) AND (t2 IN arc^.Set) THEN
                  a:=DiagIndex(_Mat, _NTaxa, t1+1, t2+1);
                  IF (a^<>EDGE) AND (a^<>2) THEN
                     IF t1<t2 THEN
                        IF a^=1 THEN
                           a^:=2          (* virtuel *)
                        ELSE
                           a^:=-1
                        END
                     ELSIF a^=-1 THEN
                        a^:=2             (* virtuel *)
                     ELSE
                        a^:=1
                     END
                  END
               END
            END
         END
      END;
      Gauge(s)
   END
END Count;

PROCEDURE Output;
VAR t1, t2 : TaxaRange;
    n      : LONGCARD;
    mat    : IntPtr;
BEGIN
   InitGauge('Write', _NTaxa-1);
   mat:=_Mat;
   n:=0;
   FOR t1:=1 TO _NTaxa-1 DO
      FOR t2:=t1+1 TO _NTaxa DO
         IF mat^=2 THEN
            INC(n);
            WrTaxa(t1);
            WrChar(_T, CHR(32));
            WrTaxa(t2);
            WrLn(_T)
         END;
         IncAddr(mat, 2)
      END;
      Gauge(t1)
   END;
   WrLn(_T);
   IF n=0 THEN
      WrStr(_T, '"No virtual edge"')
   ELSIF n=1 THEN
      WrStr(_T, '1 "virtual edge"')
   ELSE
      WrLngCard(_T, n, 0);
      WrStr(_T, ' "virtual edges"')
   END;
   WrLn(_T)
END Output;

BEGIN
   IO.WrStr(Program);
   IO.WrLn;
   LoadLevels('BGB', _Levels);
   WriteInOut('BGB', Tools);
   WriteTitle;
   NewDiagMat(_Mat, _NTaxa);
   Count;
   OpenTextFile(Tools, Program);
   Output;
   End
END BG_T03.
```

## Program BG_T04

Function : observed arcs

```
MODULE BG_T04;

(*# debug     (vid       => off) *)
(*# check     (nil_ptr   => off
               index     => off
               stack     => off
               overflow  => off
               range     => off) *)
(*# optimize  (speed     => on
               cse       => on
               const     => on
               peep_hole => on
               jump      => on
               loop      => on
               alias     => off
               regass    => on
               stk_frame => on
               i286      => on
               x87       => off) *)

IMPORT IO;
FROM FIO      IMPORT WrCard, WrChar, WrInt, WrLn, WrLngCard, WrStr;
FROM Lib      IMPORT DecAddr, Fill, IncAddr;
FROM BG_TOOLS IMPORT ClearMem, DiagIndex, EDGE, End, Gauge, GetMem, InitGauge,
                     IntPtr, LoadLevels, NewDiagMat, _NSection, _NTaxa,
                     OpenTextFile, PointSet, Set, SetPtr, _SizeOfSet,
                     Subtraction, _T, TaxaRange, Union, WriteInOut, WriteTitle,
                     WrTaxa;

CONST
   Tools   = 'T04';
   Program = 'BG_'+Tools+' v2.01: observed arcs';

VAR
   _Levels : FarADDRESS;
   _Mat    : IntPtr;

PROCEDURE Count;
VAR s, l, bottom, top : CARDINAL;
    t1, t2            : TaxaRange;
    level, edge, arc  : SetPtr;
    a                 : IntPtr;
BEGIN
   GetMem(edge, LONGCARD(_SizeOfSet));
   GetMem(arc , LONGCARD(_SizeOfSet));
   bottom:=1;
   InitGauge('Count    ', _NSection);
   FOR s:=1 TO _NSection DO
```

Search boundaries of the section

```
    top:=bottom;
    level:=PointSet(_Levels, bottom, _SizeOfSet);
    REPEAT
      INC(bottom);
      IncAddr(level, _SizeOfSet)
    UNTIL level^.H.Section<>s;

    FOR t1:=0 TO _NTaxa-2 DO
```

Search presence of species

```
      l:=bottom;
      level:=PointSet(_Levels, l, _SizeOfSet);
      REPEAT
        DEC(l);
        DecAddr(level, _SizeOfSet)
      UNTIL (l=top) OR (t1 IN level^.Set);
```

if species present

```
      IF t1 IN level^.Set THEN
        ClearMem(edge, LONGCARD(_SizeOfSet));
        ClearMem(arc , LONGCARD(_SizeOfSet));
```

Union of the horizons containing the species

```
        REPEAT
          Set(Union, edge, level, edge);
          DEC(l);
          DecAddr(level, _SizeOfSet)
        UNTIL (l<top) OR NOT (t1 IN level^.Set);
```

Union of the upper horizons

```
        WHILE top<=l DO
          Set(Union, arc, level, arc);
          DEC(l);
          DecAddr(level, _SizeOfSet)
        END;

        Set(Subtraction, arc, edge, arc);
```

Recording the edges into the matrix

```
        FOR t2:=t1+1 TO _NTaxa-1 DO
          IF t2 IN edge^.Set THEN
            a:=DiagIndex(_Mat, _NTaxa, t1+1, t2+1);
            a^:=EDGE
          END
        END;
```

marking the arcs

```
        FOR t2:=0 TO _NTaxa-1 DO
          IF (t1<>t2) AND (t2 IN arc^.Set) THEN
            a:=DiagIndex(_Mat, _NTaxa, t1+1, t2+1);
            IF a^<>EDGE THEN
              IF t1<t2 THEN
                IF 0<a^ THEN
                  a^:=EDGE          (* virtuel *)
                ELSE
                  INC(a^, -1)
                END
              ELSIF a^<0 THEN
                a^:=EDGE            (* virtuel *)
              ELSE
                INC(a^)
              END
            END
          END
        END.
      END
    END;
    Gauge(s)
  END
END Count;

PROCEDURE Output;
VAR t1, t2   : TaxaRange;
```

```
    n, sum    : LONGCARD;
    t         : ARRAY TaxaRange OF BOOLEAN;
    mat       : IntPtr;
    arc       : INTEGER;
    up, down  : ARRAY TaxaRange OF LONGCARD;
    nu, nd    : ARRAY TaxaRange OF CARDINAL;
BEGIN
   InitGauge('Write 1', _NTaxa-1);
   mat:=_Mat;
   n:=0;
   sum:=0;
   Fill(ADR(t), _NTaxa, 0);
   FOR t1:=1 TO _NTaxa-1 DO
      FOR t2:=t1+1 TO _NTaxa DO
         IF (mat^<>0) AND (mat^<>EDGE) THEN
            INC(n);
            INC(sum, LONGCARD(ABS(mat^)));
            t[t1]:=TRUE;
            t[t2]:=TRUE;
            WrTaxa(t1);
            IF mat^<0 THEN
               WrStr(_T, ' "->" ')
            ELSE
               WrStr(_T, ' "<-" ')
            END;
            WrTaxa(t2);
            WrInt(_T, ABS(mat^), 6);
            WrLn(_T)
         END;
         IncAddr(mat, 2)
      END;
      Gauge(t1)
   END;
   WrStr(_T, '"     " "  " "_"');
   WrLngCard(_T, sum, 10);
   WrLn(_T);
   WrLn(_T);
   WrLngCard(_T, n, 0);
   WrStr(_T, ' "/" ');
   WrLngCard(_T, LONGCARD(_NTaxa)*LONGCARD(_NTaxa-1) DIV 2, 0);
   WrStr(_T, ' "arcs between" ');
   n:=0;
   FOR t1:=1 TO _NTaxa DO
      IF t[t1] THEN INC(n) END
   END;
   WrLngCard(_T, n, 0);
   WrStr(_T, ' "species /" ');
   WrCard(_T, _NTaxa, 0);
   WrLn(_T);
   WrLn(_T);
   WrStr(_T, '"Reproducibilities and frequencies OF arcs"');
   WrLn(_T);
   WrStr(_T, '          "R ->"  "R <-"   "_R"  "f ->"  "f <-"   "_f"');
   WrLn(_T);
   InitGauge('Write 2', _NTaxa);
   Fill(ADR(up  ), SIZE(up  ), 0);
   Fill(ADR(down), SIZE(down), 0);
   Fill(ADR(nu  ), SIZE(nu  ), 0);
   Fill(ADR(nd  ), SIZE(nd  ), 0);
   FOR t1:=1 TO _NTaxa DO
      FOR t2:=1 TO _NTaxa DO
         IF t1<>t2 THEN
            mat:=DiagIndex(_Mat, _NTaxa, t1, t2);
            arc:=mat^;
            IF (arc<>EDGE) AND (arc<>0) THEN
               IF t2<t1 THEN arc:=-arc END;
               IF arc<0 THEN
                  INC(up[t1], LONGCARD(ABS(arc)));
                  INC(nu[t1])
               ELSE
                  INC(down[t1], LONGCARD(arc));
                  INC(nd[t1])
               END
            END
         END
      END;
      WrTaxa(t1);
      WrLngCard(_T, up[t1], 8);
      WrLngCard(_T, down[t1], 8);
      WrLngCard(_T, up[t1]+down[t1], 8);
      WrCard(_T, nu[t1], 8);
      WrCard(_T, nd[t1], 8);
      WrCard(_T, nu[t1]+nd[t1], 8);
      WrLn(_T);
      Gauge(t1)
   END
END Output;

BEGIN
   IO.WrStr(Program);
   IO.WrLn;
   LoadLevels('BGB', _Levels);
   WriteInOut('BGB', Tools);
   WriteTitle;
   NewDiagMat(_Mat, _NTaxa);
   Count;
   OpenTextFile(Tools, Program);
   Output;
   End
END BG_T04.
```

## *Program BG_T05*

Function :   statistics on the UAs

```
MODULE BG_T05;

(*# debug    (vid       => off) *)
(*# check    (nil_ptr   => off
              index     => off
              stack     => off
              overflow  => off
              range     => off) *)
(*# optimize (speed     => on
              cse       => on
              const     => on
              peep_hole => on
              jump      => on
              loop      => on
              alias     => off
              regass    => on
              stk_frame => on
              i286      => on
              x87       => off) *)

IMPORT IO;
FROM FIO       IMPORT WrCard, WrLn, WrStr;
FROM Lib       IMPORT WordFill;
FROM BG_TOOLS  IMPORT End, Gauge, GetMem, InitGauge, Intersection, LoadLevels,
                      _NLevel, _NTaxa, OpenTextFile, PointSet, Set, SetPtr,
                      _SizeOfSet, Subtraction, _T, TaxaRange, WriteInOut,
                      WriteTitle;

CONST
   Tools   = 'T05';
   Program = 'BG_'+Tools+' v2.02: statistics on UA';

VAR
   _AU : FarADDRESS;
   _S  : ARRAY[1..500] OF RECORD
           au, car, fad, lad, one, com, sfad, slad : CARDINAL
         END;

PROCEDURE Count;
VAR au                                        : CARDINAL;
    top, on, bottom, empty, fad, lad, one, com : SetPtr;
BEGIN
   GetMem(empty, LONGCARD(_SizeOfSet));
   GetMem(fad  , LONGCARD(_SizeOfSet));
   GetMem(lad  , LONGCARD(_SizeOfSet));
   GetMem(one  , LONGCARD(_SizeOfSet));
   GetMem(com  , LONGCARD(_SizeOfSet));
   InitGauge('Count', _NLevel);
   FOR au:=1 TO _NLevel DO
      IF au=1 THEN
         top:=empty
      ELSE
         top:=PointSet(_AU, au-1, _SizeOfSet)
      END;
      on:=PointSet(_AU, au, _SizeOfSet);
      IF au=_NLevel THEN
         bottom:=empty
      ELSE
         bottom:=PointSet(_AU, au+1, _SizeOfSet)
      END;
      Set(Subtraction , on , bottom, fad);
      Set(Subtraction , on , top   , lad);
      Set(Subtraction , on , fad   , com);
      Set(Subtraction , com, lad   , com);
      Set(Intersection, fad, lad   , one);
      Set(Subtraction , fad, one   , fad);
      Set(Subtraction , lad, one   , lad);
      _S[au].au :=on^.H.Level;
      _S[au].car:=on^.H.Cardinal;
      _S[au].fad:=fad^.H.Cardinal;
      _S[au].lad:=lad^.H.Cardinal;
      _S[au].one:=one^.H.Cardinal;
      _S[au].com:=com^.H.Cardinal;
      Gauge(au)
   END;
   _S[_NLevel].sfad:=_S[_NLevel].fad+_S[_NLevel].one;
   _S[_NLevel].slad:=_S[_NLevel].lad+_S[_NLevel].one;
   FOR au:=_NLevel-1 TO 1 BY -1 DO
      _S[au].sfad:=_S[au+1].sfad+_S[au].fad+_S[au].one;
      _S[au].slad:=_S[au+1].slad+_S[au].lad+_S[au].one
   END
END Count;

PROCEDURE Output;
VAR au : CARDINAL;
BEGIN
   InitGauge('Write', _NLevel);
   WrStr(_T, '"Columns: A. UA"'); WrLn(_T);
   WrStr(_T, '"         B. number OF species contained by the UA"'); WrLn(_T);
   WrStr(_T, '"         C. species strictly contained by the UA"'); WrLn(_T);
   WrStr(_T, '"         D. LAD without column C"'); WrLn(_T);
   WrStr(_T, '"         E. FAD without column C"'); WrLn(_T);
   WrStr(_T, '"         F. common species with the previous and the next UA"');
            WrLn(_T);
   WrStr(_T, '"         G. LAD + column C"'); WrLn(_T);
   WrStr(_T, '"         H. FAD + column C"'); WrLn(_T);
   WrStr(_T, '"         I. common species with the previous UA"'); WrLn(_T);
   WrStr(_T, '"         J. common species with the next UA"'); WrLn(_T);
   WrStr(_T, '"         K. sum OF LAD from the first UA"'); WrLn(_T);
   WrStr(_T, '"         L. sum OF FAD from the first UA"'); WrLn(_T);
   WrLn(_T);
   WrStr(_T, ' "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L"'); WrLn(_T);
   FOR au:=1 TO _NLevel DO
      (*A*) WrCard(_T, _S[au].au , 3);
      (*B*) WrCard(_T, _S[au].car, 4);
      (*C*) WrCard(_T, _S[au].one, 4);
      (*D*) WrCard(_T, _S[au].lad, 4);
      (*E*) WrCard(_T, _S[au].fad, 4);
      (*F*) WrCard(_T, _S[au].com, 4);
      (*G*) WrCard(_T, _S[au].lad+_S[au].one, 4);
      (*H*) WrCard(_T, _S[au].fad+_S[au].one, 4);
      (*I*) WrCard(_T, _S[au].lad+_S[au].com, 4);
      (*J*) WrCard(_T, _S[au].fad+_S[au].com, 4);
      (*K*) WrCard(_T, _S[au].slad, 4);
      (*L*) WrCard(_T, _S[au].sfad, 4);
      WrLn(_T);
      Gauge(au)
   END
END Output;

BEGIN
   IO.WrStr(Program);
   IO.WrLn;
   LoadLevels('BGI', _AU);
   WriteInOut('BGI', Tools);
   WriteTitle;
   WordFill(ADR(_S), SIZE(_S) DIV 2, 0);
   Count;
   OpenTextFile(Tools, Program);
   Output;
   End
END BG_T05.
```

## Program BG_T06

Function : comparison of the biostratigraphic graph with the UAs

```
MODULE BG_T06;

(*# debug    (vid       => off) *)
(*# check    (nil_ptr   => off
              index     => off
              stack     => off
              overflow  => off
              range     => off) *)
(*# optimize (speed     => on
              cse       => on
              const     => on
              peep_hole => on
              jump      => on
              loop      => on
              alias     => off
              regass    => on
              stk_frame => on
              i286      => on
              x87       => off) *)

IMPORT IO;
FROM FIO      IMPORT WrChar, WrLn, WrLngCard, WrStr;
FROM Lib      IMPORT IncAddr;
FROM Str      IMPORT Caps;
FROM BG_TOOLS IMPORT ClearMem, DiagIndex, EDGE, End, Gauge, GetMem, InitGauge,
                     IntPtr, LoadDiagMat, LoadLevels, NewDiagMat, _NLevel,
                     _NTaxa, OpenTextFile, PointSet, _Quote, Set, SetPtr,
                     _SizeOfSet, Subtraction, _T, TaxaRange, Union, WriteInOut,
                     WrTaxa, WriteTitle;

CONST
   Tools   = 'T06';
   Program = 'BG_'+Tools+' v2.00: compare G* with UA';

TYPE
   TestProc = PROCEDURE(INTEGER, INTEGER) : BOOLEAN;

VAR
   _AU      : FarADDRESS;
   _Gb, _Gk : IntPtr;
   _N       : LONGCARD;

PROCEDURE MakeGk;
VAR t1, t2         : TaxaRange;
    a              : CARDINAL;
    au, edge, arc  : SetPtr;
    mat            : IntPtr;
BEGIN
   GetMem(edge, LONGCARD(_SizeOfSet));
   GetMem(arc , LONGCARD(_SizeOfSet));
   InitGauge('Matrix OF AU   ', _NTaxa-2);
   FOR t1:=0 TO _NTaxa-2 DO
      ClearMem(edge, LONGCARD(_SizeOfSet));
      ClearMem(arc , LONGCARD(_SizeOfSet));
      a:=1;
      au:=_AU;
      WHILE (a<_NLevel) AND NOT (t1 IN au^.Set) DO
         Set(Union, arc, au, arc);
         INC(a);
         IncAddr(au, _SizeOfSet)
      END;
      REPEAT
         Set(Union, edge, au, edge);
         INC(a);
         IncAddr(au, _SizeOfSet)
      UNTIL (_NLevel<a) OR NOT (t1 IN au^.Set);
      Set(Subtraction, arc, edge, arc);
      FOR t2:=t1+1 TO _NTaxa-1 DO
         IF t2 IN edge^.Set THEN
            mat:=DiagIndex(_Gk, _NTaxa, t1+1, t2+1);
            mat^:=EDGE
         END
      END;
      FOR t2:=0 TO _NTaxa-1 DO
         IF t2 IN arc^.Set THEN
```

```
            mat:=DiagIndex(_Gk, _NTaxa, t1+1, t2+1);
            IF t1<t2 THEN
               mat^:=-1
            ELSE
               mat^:=1
            END
         END
      END;
      Gauge(t1)
   END
END MakeGk;

PROCEDURE GbNorm;
VAR mat    : IntPtr;
    t1, t2 : TaxaRange;
BEGIN
   InitGauge('Normalize A    ', _NTaxa-1);
   mat:=_Gb;
   FOR t1:=1 TO _NTaxa-1 DO
      FOR t2:=t1+1 TO _NTaxa DO
         IF mat^<>EDGE THEN
            IF mat^<0 THEN
               mat^:=-1
            ELSIF mat^>0 THEN
               mat^:=1
            END
         END;
         IncAddr(mat, 2)
      END;
      Gauge(t1)
   END
END GbNorm;

PROCEDURE DestroyedEdge(a, b : INTEGER) : BOOLEAN;
BEGIN
   RETURN (a=EDGE) AND (b<>EDGE)
END DestroyedEdge;

PROCEDURE ReverseArc(a, b : INTEGER) : BOOLEAN;
BEGIN
   RETURN ((a=1) AND (b=-1)) OR ((a=-1) AND (b=1))
END ReverseArc;

PROCEDURE VirtualEdge(a, b : INTEGER) : BOOLEAN;
BEGIN
   RETURN ((ABS(a)=1) OR (a=0)) AND (b=EDGE)
END VirtualEdge;

PROCEDURE Compare(title : ARRAY OF CHAR;
                  test   : TestProc);
VAR t1, t2 : TaxaRange;
    b, k   : IntPtr;
BEGIN
   _N:=0;
   b:=_Gb;
   k:=_Gk;
   InitGauge(title, _NTaxa-1);
   FOR t1:=1 TO _NTaxa-1 DO
      FOR t2:=t1+1 TO _NTaxa DO
         IF test(b^, k^) THEN
            IF _N=0 THEN
               Caps(title);
               WrStr(_T, title);
               WrLn(_T)
            END;
            INC(_N);
            WrTaxa(t1);
            WrChar(_T, CHR(32));
            WrTaxa(t2);
            WrLn(_T)
         END;
         IncAddr(b, 2);
         IncAddr(k, 2)
      END;
      Gauge(t1)
   END
END Compare;

PROCEDURE WrN(title : ARRAY OF CHAR);
BEGIN
   IF _N=0 THEN
      WrStr(_T, 'No ')
   ELSE
      WrLngCard(_T, _N, 0);
      WrChar(_T, CHR(32))
   END;
   WrStr(_T, title);
   IF 1<_N THEN
      WrChar(_T, 's')
   END;
   WrLn(_T);
   WrLn(_T)
END WrN;

BEGIN
   _Quote:=FALSE;
   IO.WrStr(Program);
   IO.WrLn;
   WriteInOut('BGD/BGI', Tools);
   WriteTitle;
   NewDiagMat(_Gk, _NTaxa);
   LoadDiagMat('BGD', _Gb);
   LoadLevels('BGI', _AU);
   MakeGk;
   GbNorm;
   OpenTextFile(Tools, Program);
   Compare('Destroyed edges', DestroyedEdge);
   WrN('destroyed edge');
   Compare('Reverse arcs   ', ReverseArc);
   WrN('reverse arc');
   Compare('Virtual edges  ', VirtualEdge);
   WrN('virtual edge');
   End
END BG_T06.
```

## *Program BG_T07*

Function :   distance between the UAs

```
MODULE BG_T07;

(*# debug    (vid        => off) *)
(*# check    (nil_ptr    => off
              index      => off
              stack      => off
              overflow   => off
              range      => off) *)
(*# optimize (speed      => on
              cse        => on
              const      => on
              peep_hole  => on
              jump       => on
              loop       => on
              alias      => off
              regass     => on
              stk_frame  => on
              i286       => on
              x87        => off) *)

IMPORT IO;
FROM FIO      IMPORT WrCard, WrLn, WrStr;
FROM Lib      IMPORT DecAddr;
FROM Str      IMPORT FixRealToStr;
FROM BG_TOOLS IMPORT End, Gauge, GetMem, InitGauge, LoadLevels,
                     _NLevel, OpenTextFile, PointSet, Set, SetPtr,
                     _SizeOfSet, Subtraction, _T, WriteInOut,
                     WriteTitle;

CONST
   Tools   = 'T07';
   Program = 'BG_'+Tools+' v2.00: Gap ratio between UA';

VAR
   _AU : FarADDRESS;

PROCEDURE Count;
VAR au            : CARDINAL;
    a, b, ab, ba : SetPtr;
    ratio        : REAL;
    r            : ARRAY[0..8] OF CHAR;
    ok           : BOOLEAN;
BEGIN
   GetMem(ab, LONGCARD(_SizeOfSet));
   GetMem(ba, LONGCARD(_SizeOfSet));
   InitGauge('Gap ratio', _NLevel-1);
   b:=PointSet(_AU, _NLevel, _SizeOfSet);
   FOR au:=1 TO _NLevel-1 DO
      a:=b;
      DecAddr(b, _SizeOfSet);
      Set(Subtraction, a, b, ab);
      Set(Subtraction, b, a, ba);
      ratio:=FLOAT(ab^.H.Cardinal)/FLOAT(a^.H.Cardinal)+
             FLOAT(ba^.H.Cardinal)/FLOAT(b^.H.Cardinal);
      FixRealToStr(LONGREAL(ratio), 6, r, ok);
      WrCard(_T, au, 3);
      WrStr(_T, ' \ ');
      WrCard(_T, au+1, 3);
      WrStr(_T, '    ');
      WrStr(_T, r);
      WrLn(_T);
      Gauge(au)
   END
END Count;

BEGIN
   IO.WrStr(Program);
   IO.WrLn;
   LoadLevels('BGI', _AU);
   WriteInOut('BGI', Tools);
   WriteTitle;
   OpenTextFile(Tools, Program);
   Count;
   End
END BG_T07.
```

## *Program BG_T08*

Function : number of real arcs between the UAs

```
MODULE BG_T08;

(*# debug    (vid       => off) *)
(*# check    (nil_ptr   => off
              index     => off
              stack     => off
              overflow  => off
              range     => off) *)
(*# optimize (speed     => on
              cse       => on
              const     => on
              peep_hole => on
              jump      => on
              loop      => on
              alias     => off
              regass    => on
              stk_frame => on
              i286      => on
              x87       => off) *)

IMPORT IO;
FROM FIO      IMPORT WrCard, WrLn, WrLngCard, WrStr;
FROM Lib      IMPORT SubAddr;
FROM BG_TOOLS IMPORT ClearMem, DiagIndex, EDGE, End, Gauge, GetMem, InitGauge,
                     IntPtr, LoadDiagMat, LoadLevels, _NLevel,
                     _NTaxa, OpenTextFile, PointSet, Set, SetPtr,
                     _SizeOfSet, Subtraction, _T, TaxaRange, Union, WriteInOut,
                     WriteTitle;

CONST
   Tools   = 'T08';
   Program = 'BG_'+Tools+' v2.00: real arcs in UA';

VAR
   _AU, _A : FarADDRESS;
```

```
PROCEDURE Count;
VAR l                       : CARDINAL;
    au1, au2, level, arc : SetPtr;
    mat                     : IntPtr;
    count                   : LONGCARD;
    t1, t2                  : TaxaRange;
BEGIN
   GetMem(level, LONGCARD(_SizeOfSet));
   GetMem(arc  , LONGCARD(_SizeOfSet));
   InitGauge('Count', _NLevel-1);
   au1:=PointSet(_AU, _NLevel, _SizeOfSet);
   FOR l:=1 TO _NLevel-1 DO
      au2:=SubAddr(au1, _SizeOfSet);
      Set(Subtraction, au1, au2, level);
      Set(Subtraction, au2, au1, arc  );
      count:=0;
      FOR t1:=0 TO _NTaxa-1 DO
         IF t1 IN level^.Set THEN
            FOR t2:=0 TO _NTaxa-1 DO
               IF t2 IN arc^.Set THEN
                  mat:=DiagIndex(_A, _NTaxa, t1+1, t2+1);
                  IF mat^<>0 THEN
                     INC(count)
                  END
               END
            END
         END
      END;
      WrCard(_T, l  , 3);
      WrStr(_T, ' > ');
      WrCard(_T, l+1, 3);
      WrLngCard(_T, count, 9);
      WrLn(_T);
      au1:=au2;
      Gauge(l)
   END
END Count;

BEGIN
   IO.WrStr(Program);
   IO.WrLn;
   WriteInOut('BGD/BGI', Tools);
   WriteTitle;
   LoadDiagMat('BGD', _A);
   LoadLevels('BGI', _AU);
   OpenTextFile(Tools, Program);
   Count;
   End
END BG_T08.
```

## *Program BG_T09*

Function : relationships between the cliques (conflicting ranges)

```
MODULE BG_T09;

(*# debug    (vid       => off) *)
(*# check    (nil_ptr   => off
              index     => off
              stack     => off
              overflow  => off
              range     => off) *)
(*# optimize (speed     => on
              cse       => on
              const     => on
              peep_hole => on
              jump      => on
              loop      => on
              alias     => off
              regass    => on
              stk_frame => on
              i286      => on
              x87       => off) *)

IMPORT IO;
```

```
FROM FIO      IMPORT WrCard, WrLn, WrStr;
FROM Lib      IMPORT IncAddr;
FROM Str      IMPORT FixRealToStr;
FROM BG_TOOLS IMPORT DiagIndex, EDGE, End, Gauge, GetMem, InitGauge,
                     IntPtr, LoadDiagMat, LoadLevels, _NLevel,
                     _NTaxa, OpenTextFile, PointSet, Set, SetPtr,
                     _SizeOfSet, Subtraction, _T, TaxaRange, WriteInOut,
                     WriteTitle;

CONST
   Tools   = 'T09';
   Program = 'BG_'+Tools+' v2.00: relationships between the maximal cliques';

VAR
   _K, _A : FarADDRESS;

PROCEDURE Min(a, b : CARDINAL) : CARDINAL;
BEGIN
   IF a<b THEN
      RETURN a
   ELSE
      RETURN b
   END
END Min;

PROCEDURE Max(a, b : CARDINAL) : CARDINAL;
BEGIN
   IF a<b THEN
      RETURN b
   ELSE
      RETURN a
   END
END Max;

PROCEDURE Ratio(a, b : CARDINAL) : LONGREAL;
BEGIN
   IF Max(a, b)=0 THEN
      RETURN 0.0
   ELSE
      RETURN LONGREAL(FLOAT(Min(a, b))/FLOAT(Max(a, b)))
   END
END Ratio;

PROCEDURE MakeGk;
VAR t1, t2             : TaxaRange;
    k1, k2             : CARDINAL;
    l1, l2, K1, K2     : SetPtr;
    a                  : IntPtr;
    f12, f21, r12, r21 : CARDINAL;
    s                  : ARRAY[0..6] OF CHAR;
    ok                 : BOOLEAN;
    r1, r2, r3         : SHORTCARD;
BEGIN
   GetMem(K1, LONGCARD(_SizeOfSet));
   GetMem(K2, LONGCARD(_SizeOfSet));
   InitGauge('Make Gk', _NLevel-1);
   l1:=_K;
   FOR k1:=1 TO _NLevel-1 DO
      l2:=l1;
      FOR k2:=k1+1 TO _NLevel DO
         IncAddr(l2, _SizeOfSet);
         Set(Subtraction, l1, l2, K1);
         Set(Subtraction, l2, l1, K2);
         f12:=0;
         f21:=0;
         r12:=0;
         r21:=0;
         FOR t1:=0 TO _NTaxa-2 DO
            IF (t1 IN K1^.Set) OR (t1 IN K2^.Set) THEN
               FOR t2:=t1+1 TO _NTaxa-1 DO
                  IF t2 IN K1^.Set THEN
                     a:=DiagIndex(_A, _NTaxa, t1+1, t2+1);
                     IF (a^<>EDGE) AND (a^<>0) THEN
                        (* t2 _ K1 et t1 _ K2 *)
                        IF a^<0 THEN
                           INC(f21);
                           INC(r21, ABS(a^))
                        ELSE
                           INC(f12);
                           INC(r12, a^)
                        END
                     END
         ELSIF t2 IN K2^.Set THEN
            a:=DiagIndex(_A, _NTaxa, t1+1, t2+1);
            IF (a^<>EDGE) AND (a^<>0) THEN
               (* t1 _ K1 et t2 _ K2 *)
               IF a^<0 THEN
                  INC(f12);
                  INC(r12, ABS(a^))
               ELSE
                  INC(f21);
                  INC(r21, a^)
               END
            END
         END
      END
   END
END;
WrCard(_T, k1, 3);
WrCard(_T, k2, 4);
WrCard(_T, f12, 6);
WrCard(_T, f21, 6);
FixRealToStr(Ratio(f12, f21), 4, s, ok);
WrStr(_T, '  ');
WrStr(_T, s);
IF f12=f21 THEN
   WrStr(_T, ' "??"');
   r1:=0
ELSIF f12>f21 THEN
   IF f21=0 THEN
      WrStr(_T, ' ">>"');
      r1:=1
   ELSE
      WrStr(_T, ' "->"');
      r1:=2
   END
ELSIF f12=0 THEN
   WrStr(_T, ' "<<"');
   r1:=3
ELSE
   WrStr(_T, ' "<-"');
   r1:=4
END;
WrCard(_T, r12, 6);
WrCard(_T, r21, 6);
FixRealToStr(Ratio(r12, r21), 4, s, ok);
WrStr(_T, '  ');
WrStr(_T, s);
IF r12=r21 THEN
   WrStr(_T, ' "??"');
   r2:=0
ELSIF r12>r21 THEN
   IF r21=0 THEN
      WrStr(_T, ' ">>"');
      r2:=1
   ELSE
      WrStr(_T, ' "->"');
      r2:=2
   END
ELSIF r12=0 THEN
   WrStr(_T, ' "<<"');
   r2:=3
ELSE
   WrStr(_T, ' "<-"');
   r2:=4
END;
INC(r12, f12);
INC(r21, f21);
WrCard(_T, r12, 6);
WrCard(_T, r21, 6);
FixRealToStr(Ratio(r12, r21), 4, s, ok);
WrStr(_T, '  ');
WrStr(_T, s);
IF r12=r21 THEN
   WrStr(_T, ' "??"');
   r3:=0
ELSIF r12>r21 THEN
   IF r21=0 THEN
      WrStr(_T, ' ">>"');
      r3:=1
   ELSE
      WrStr(_T, ' "->"');
      r3:=2
   END
```

```
            ELSIF r12=0 THEN
               WrStr(_T, ' "<<"');
               r3:=3
            ELSE
               WrStr(_T, ' "<-"');
               r3:=4
            END;
            IF (r1<>r2) OR (r2<>r3) OR (r3<>r1) THEN
               WrStr(_T, ' "!"')
            END;
            WrLn(_T)
         END;
         IncAddr(l1, _SizeOfSet);
         Gauge(k1)
      END
   END MakeGk;

   BEGIN
      IO.WrStr(Program);
      IO.WrLn;
      WriteInOut('BGD/BGE', Tools);
      WriteTitle;
      LoadDiagMat('BGD', _A);
      LoadLevels('BGE', _K);
      OpenTextFile(Tools, Program);
      WrStr(_T, ' "k" "k"  "A>"  "A<"  "f(A)"  ""   "R>"  "R<"  "f(R)"  ""
               "A+R>""A+R<""f(A+R)"');
      WrLn(_T);
      MakeGk;
      End
   END BG_T09.
```

## *Program BG_T10*

Function : strongly connected components in G*

```
MODULE BG_T10;

(*# debug    (vid       => off) *)
(*# check    (nil_ptr   => off
              index     => off
              stack     => off
              overflow  => off
              range     => off) *)
(*# optimize (speed     => on
              cse       => on
              const     => on
              peep_hole => on
              jump      => on
              loop      => on
              alias     => off
              regass    => on
              stk_frame => on
              i286      => on
              x87       => off) *)

IMPORT IO;
FROM FIO      IMPORT WrCard, WrLn, WrStr, WrStrAdj;
FROM Lib      IMPORT Fill, IncAddr, QSort;
FROM Str      IMPORT FixRealToStr;
FROM BG_TOOLS IMPORT ClearMem, DiagIndex, EDGE, End, Gauge, GetMem, InitGauge,
                     IntPtr, LoadDiagMat, _NTaxa, OpenTextFile, PointSet,
                     PutInSet, Set, SetPtr, _SizeOfSet, SizeOfSet, _T,
                     TaxaRange, WriteInOut, WriteTitle, WrTaxa;

CONST
   Tools   = 'T10';
   Program = 'BG_'+Tools+' v2.00: strongly connected components in G*';

TYPE
   MarkedVertex = ARRAY TaxaRange OF BOOLEAN;
   VertexList   = ARRAY TaxaRange OF TaxaRange;
   CoeffRec     = RECORD
                     t                                        : TaxaRange;
                     fac, fbc, rac, rbc, fat, fbt, rat, rbt : CARDINAL;
                     cfa, cfb, cra, crb, coeff                : REAL
```

```
                  END;
   CoeffList    = ARRAY TaxaRange OF CoeffRec;

VAR
   _Mat       : IntPtr;
   NConnex    : CARDINAL;
   Components : VertexList;
   ConnexList : FarADDRESS;
   _C         : CoeffList;

PROCEDURE DirectedGraph;
VAR mat    : IntPtr;
    t1, t2 : TaxaRange;
BEGIN
   InitGauge('Directed graph               ', _NTaxa-1);
   mat:=_Mat;
   FOR t1:=1 TO _NTaxa-1 DO
      FOR t2:=t1+1 TO _NTaxa DO
         IF mat^=EDGE THEN
            mat^:=0
         END;
         IncAddr(mat, 2)
      END;
      Gauge(t1)
   END
END DirectedGraph;

PROCEDURE Forward(v1, v2 : TaxaRange) : BOOLEAN;
VAR m : IntPtr;
    a : INTEGER;
BEGIN
   m:=DiagIndex(_Mat, _NTaxa, v1, v2);
   a:=m^;
   IF v2<v1 THEN a:=-a END;
   IF a<0 THEN
      RETURN TRUE
   ELSE
      RETURN FALSE
   END
END Forward;

PROCEDURE Backward(v1, v2 : TaxaRange) : BOOLEAN;
VAR m : IntPtr;
    a : INTEGER;
BEGIN
   m:=DiagIndex(_Mat, _NTaxa, v1, v2);
   a:=m^;
   IF v2<v1 THEN a:=-a END;
   IF 0<a THEN
      RETURN TRUE
   ELSE
      RETURN FALSE
   END
END Backward;

PROCEDURE DFS1(v1        : TaxaRange;
           VAR notmarked : MarkedVertex;
           VAR list      : VertexList;
           VAR v         : TaxaRange);
VAR v2 : TaxaRange;
BEGIN
   notmarked[v1]:=FALSE;
   FOR v2:=1 TO _NTaxa DO
      IF notmarked[v2] AND Forward(v1, v2) THEN
         DFS1(v2, notmarked, list, v)
      END
   END;
   INC(v);
   list[v]:=v1
END DFS1;

PROCEDURE DFS2(v1         : TaxaRange;
           VAR notmarked  : MarkedVertex;
           VAR components : VertexList;
               n          : TaxaRange);
VAR v2 : TaxaRange;
BEGIN
   notmarked[v1]:=FALSE;
   components[v1]:=n;
   FOR v2:=1 TO _NTaxa DO
      IF notmarked[v2] AND Backward(v1, v2) THEN
         DFS2(v2, notmarked, components, n)
      END
   END
END DFS2;

PROCEDURE Connex(VAR components : VertexList);
VAR list      : VertexList;
    notmarked : MarkedVertex;
    t, v      : TaxaRange;
BEGIN
   InitGauge('Strongly connected components', 2*_NTaxa-1);
   Fill(ADR(notmarked), SIZE(notmarked), ORD(TRUE));
   v:=0;
   FOR t:=1 TO _NTaxa DO
      IF notmarked[t] THEN
         DFS1(t, notmarked, list, v)
      END;
      Gauge(t)
   END;
   Fill(ADR(notmarked), SIZE(notmarked), ORD(TRUE));
   v:=0;
   FOR t:=_NTaxa TO 1 BY -1 DO
      IF notmarked[list[t]] THEN
         INC(v);
         DFS2(list[t], notmarked, components, v)
      END;
      Gauge(2*_NTaxa-t)
   END
END Connex;

PROCEDURE DoConnex(VAR components : VertexList;
                   VAR nconnex    : CARDINAL;
                       connex     : SetPtr);
VAR t : TaxaRange;
    c : SetPtr;
BEGIN
   InitGauge('List OF components           ', 2*_NTaxa);
   FOR t:=1 TO _NTaxa DO
      c:=PointSet(connex, components[t], _SizeOfSet);
      PutInSet(t-1, c);
      Gauge(t)
   END;
   nconnex:=0;
   FOR t:=1 TO _NTaxa DO
      IF 2<connex^.H.Cardinal THEN
         INC(nconnex)
      END;
      IncAddr(connex, _SizeOfSet);
      Gauge(_NTaxa+t)
   END
END DoConnex;

PROCEDURE Min(a, b : REAL) : REAL;
BEGIN
   IF a<b THEN RETURN a ELSE RETURN b END
END Min;

PROCEDURE Max(a, b : REAL) : REAL;
BEGIN
   IF a>b THEN RETURN a ELSE RETURN b END
END Max;

PROCEDURE Less(a, b : CARDINAL) : BOOLEAN;
BEGIN
   RETURN _C[a].coeff>_C[b].coeff
END Less;

PROCEDURE Swap(a, b : CARDINAL);
VAR buffer : CoeffRec;
BEGIN
   buffer:=_C[a];
   _C[a]:=_C[b];
   _C[b]:=buffer
END Swap;

PROCEDURE WrReal(r : REAL);
VAR s  : ARRAY[0..3] OF CHAR;
    ok : BOOLEAN;
BEGIN
   FixRealToStr(LONGREAL(r), 0, s, ok);
   WrStrAdj(_T, s, 5)
END WrReal;
```

```
PROCEDURE Output(connex : SetPtr;
                 n      : CARDINAL);
VAR c, t, a : TaxaRange;
    m       : IntPtr;
    arc     : INTEGER;
BEGIN
   IF NConnex=0 THEN
      WrStr(_T, '"No strongly connected component"')
   ELSE
      InitGauge('Write components            ', _NTaxa);
      IF NConnex=1 THEN
         WrStr(_T, '"1 strongly connected component"')
      ELSE
         WrCard(_T, NConnex, 0);
         WrStr(_T, ' "strongly connected components"')
      END;
      WrLn(_T);
      n:=0;
      FOR c:=1 TO _NTaxa DO
         IF 2<connex^.H.Cardinal THEN
            Fill(ADR(_C), SIZE(_C), 0);
            INC(n);
            WrLn(_T);
            WrStr(_T, '""       "SCC"   ""    ""    ""  "G*"    ""    ""    ""
            "SCC/G*"');
            WrLn(_T);
            WrStr(_T, '""        "f>"  "r>"  "f<"  "r<"  "f>"  "r>"  "f<"  "r<" "f>"
            "f<" "r>" "r<"  "C"');
            WrLn(_T);
            WrLn(_T);
            WrStr(_T, '"Strongly connected component ');
            WrCard(_T, n, 0);
            WrStr(_T, ': ');
            WrCard(_T, connex^.H.Cardinal, 0);
            WrStr(_T, ' vertices"');
            WrLn(_T);
            FOR t:=1 TO _NTaxa DO
               IF (t-1) IN connex^.Set THEN
                  _C[t].t:=t;
                  FOR a:=1 TO _NTaxa DO
                     IF t<>a THEN
                        m:=DiagIndex(_Mat, _NTaxa, t, a);
                        IF m^<>0 THEN
                           arc:=m^;
                           IF a<t THEN arc:=-arc END;
                           IF arc<0 THEN
                              INC(_C[t].fat);
                              INC(_C[t].rat, ABS(arc));
                              IF (a-1) IN connex^.Set THEN
                                 INC(_C[t].fac);
                                 INC(_C[t].rac, ABS(arc))
                              END
                           ELSE
                              INC(_C[t].fbt);
                              INC(_C[t].rbt, arc);
                              IF (a-1) IN connex^.Set THEN
                                 INC(_C[t].fbc);
                                 INC(_C[t].rbc, arc)
                              END
                           END
                        END
                     END
                  END;
                  _C[t].cfa:=100.0*FLOAT(_C[t].fac)/FLOAT(_C[t].fat);
                  _C[t].cfb:=100.0*FLOAT(_C[t].fbc)/FLOAT(_C[t].fbt);
                  _C[t].cra:=100.0*FLOAT(_C[t].rac)/FLOAT(_C[t].rat);
                  _C[t].crb:=100.0*FLOAT(_C[t].rbc)/FLOAT(_C[t].rbt);
                  _C[t].coeff:=100.0*FLOAT(_C[t].fac+_C[t].fbc+_C[t].rac+_C[t].rbc)
                                    /FLOAT(_C[t].fat+_C[t].fbt+_C[t].rat+_C[t].rbt)
               END
            END;
            QSort(_NTaxa, Less, Swap);
            FOR t:=1 TO _NTaxa DO
               IF _C[t].t<>0 THEN
                  WrTaxa(_C[t].t);
                  WrCard(_T, _C[t].fac, 6);
                  WrCard(_T, _C[t].rac, 6);
                  WrCard(_T, _C[t].fbc, 6);
                  WrCard(_T, _C[t].rbc, 6);
                  WrCard(_T, _C[t].fat, 6);
                  WrCard(_T, _C[t].rat, 6);
                  WrCard(_T, _C[t].fbt, 6);
                  WrCard(_T, _C[t].rbt, 6);
                  WrReal(_C[t].cfa);
                  WrReal(_C[t].cfb);
                  WrReal(_C[t].cra);
                  WrReal(_C[t].crb);
                  WrReal(_C[t].coeff);
                  WrLn(_T)
               END
            END
         END;
         IncAddr(connex, _SizeOfSet);
         Gauge(c)
      END
   END
END Output;

BEGIN
   IO.WrStr(Program);
   IO.WrLn;
   LoadDiagMat('BGD', _Mat);
   _SizeOfSet:=SizeOfSet(_NTaxa);
   GetMem(ConnexList, LONGCARD(_NTaxa)*LONGCARD(_SizeOfSet));
   WriteInOut('BGD', Tools);
   WriteTitle;
   DirectedGraph;
   Connex(Components);
   DoConnex(Components, NConnex, ConnexList);
   OpenTextFile(Tools, Program);
   Output(ConnexList, NConnex);
   End
END BG_T10.
```

## *Program BG_T11*

Function : completing the UA range chart with the complete names of the species

```
MODULE BG_T11;

(*# debug    (vid       => off) *)
(*# check    (nil_ptr   => off
              index     => off
              stack     => off
              overflow  => off
              range     => off) *)
(*# optimize (speed     => on
              cse       => on
              const     => on
              peep_hole => on
              jump      => on
              loop      => on
              alias     => off
              regass    => on
              stk_frame => on
              i286      => on
              x87       => off) *)

IMPORT IO, FIO;
FROM Lib      IMPORT ParamCount, ParamStr;
FROM Str      IMPORT Append, Caps, CHARSET, CharPos, Compare, Copy, Delete,
                     Length, Prepend, Slice;
FROM BG_TOOLS IMPORT Error, _Filename, _NTaxa, CreateFileName, WriteInOut;

CONST
   Program = 'BG_T11 v2.00: full species names in UA';

   NameLength  = 50;
   Separators1 = CHARSET{CHR(0)..CHR(32), '!', '&', "'", '(', ')', '*', '+',
                  ',', '-', '.', '/', ':', ';', '=', '>', '?', '[', '\', ']',
                  '^', '`', '|', '}', '~', CHR(127)};
   Separators2 = CHARSET{CHR(10),CHR(13)};
   Separators3 = CHARSET{CHR(1)..CHR(32)};
   Characters  = CHARSET{CHR(32)..CHR(255)};
   EndOfLine   = CHARSET{CHR(13)};

TYPE
   TaxaCodeT = ARRAY[0..5] OF CHAR;
   TaxaNameT = ARRAY[0..NameLength] OF CHAR;
   TaxaRec   = RECORD
                  code : TaxaCodeT;
                  name : TaxaNameT;
               END;
   TaxaListT = ARRAY[1..500] OF TaxaRec;

VAR
   TaxaList    : TaxaListT;
   Dictionnary : FIO.PathStr;

PROCEDURE ReadTaxaList(path : FIO.PathStr);
VAR f           : FIO.File;
    r           : ARRAY[0..255] OF CHAR;
    first, last : CARDINAL;
BEGIN
   path:=CreateFileName(path, 'DCT');
   IF NOT FIO.Exists(path) THEN Error(1006) END;
   f:=FIO.OpenRead(path);
   IO.WrStr('Dictionnary');
   _NTaxa:=0;
   LOOP
      FIO.Separators:=Separators1;
      FIO.RdItem(f, r);
      IF FIO.EOF THEN EXIT END;
      INC(_NTaxa);
      Caps(r);
      first:=Length(r);
      CASE first OF
      |   1 : Prepend(r, '    ')
      |   2 : Prepend(r, '   ')
      |   3 : Prepend(r, '  ')
      |   4 : Prepend(r, ' ')
      END;
      Copy(TaxaList[_NTaxa].code, r);
      FIO.Separators:=Separators2;
      FIO.RdItem(f, r);
      first:=0;
      WHILE (r[first] IN Separators3) AND (first<255) DO INC(first) END;
      last:=first;
      WHILE (r[last] IN Characters) AND (last<255) DO INC(last) END;
      IF last<255 THEN DEC(last) END;
      WHILE (r[last]=CHR(32)) AND (first<last) DO DEC(last) END;
      Slice(r, r, first, last-first+1);
      Copy(TaxaList[_NTaxa].name, r);
      IO.WrChar('.')
   END;
   FIO.Close(f);
   IO.WrLn
END ReadTaxaList;

PROCEDURE Translate(name1 : FIO.PathStr);
VAR f1, f2 : FIO.File;
    l      : ARRAY[0..255+NameLength] OF CHAR;
    i      : CARDINAL;
    code   : TaxaCodeT;
    name2  : FIO.PathStr;
BEGIN
   name2:=CreateFileName(name1, '$$$');
   name1:=CreateFileName(name1, 'TGJ');
   f1:=FIO.OpenRead(name1);
   f2:=FIO.Create(name2);
   FIO.Separators:=EndOfLine;
   IO.WrStr('Modification');
   LOOP
      FIO.RdItem(f1, l);
      IF FIO.EOF THEN EXIT END;
      IF l[0]=CHR(10) THEN Delete(l, 0, 1) END;
      IF CharPos(l, '+')=6 THEN
         Slice(code, l, 0, 5);
         Delete(l, 0, 6);
         Append(l, ' ');
         i:=1;
         WHILE (Compare(code, TaxaList[i].code)<>0) AND (i<=_NTaxa) DO
            INC(i)
         END;
         IF _NTaxa<i THEN
            Append(l, '? ');
            Append(l, code);
            IO.WrChar('?')
         ELSE
            Append(l, TaxaList[i].name);
            IO.WrChar('.')
         END;
         FIO.WrStr(f2, l)
      ELSE
         i:=0;
         WHILE (l[i]=CHR(32)) AND (i<6) DO INC(i) END;
         IF i=6 THEN
            Delete(l, 0, 6);
            FIO.WrStr(f2, l)
         ELSE
            FIO.WrStr(f2, l)
         END
      END;
      FIO.WrLn(f2)
   END;
   FIO.Close(f1);
   FIO.Close(f2);
   FIO.Erase(name1);
   FIO.Rename(name2, name1);
   Error(0)
END Translate;

BEGIN
   IO.WrStr(Program);
   IO.WrLn;
   IF ParamCount()<>2 THEN Error(1003) END;
   ParamStr(Dictionnary, 2);
   WriteInOut('DCT', 'TGJ');
   ReadTaxaList(Dictionnary);
   Translate(_Filename)
END BG_T11.
```

## *Program BG_T12*

### Function : Application of a false range chart to a true dataset

```
MODULE BG_T12;

IMPORT IO;
FROM FIO      IMPORT PathStr, WrCard, WrChar, WrLn, WrStr;
FROM Lib      IMPORT ParamCount, ParamStr;
FROM BG_TOOLS IMPORT _Filename1, _Filename2, _NLevel, _NTaxa, _Quote,
                     _SizeOfSet, _T, End, Error, Gauge, GetMem, InitGauge,
                     IsCompatibleFiles, LoadLevels, OpenTextFile, PointSet, Rd2CmdLine,
            Set,
                     SetPtr, Subtraction, Write2InOut, WriteTitle;

CONST
   Tools   = 'T12';
   Program = 'BG_'+Tools+' v1.01: UA correlations';

VAR
   Levels, AU : ADDRESS;
   NbOfAU     : CARDINAL;

PROCEDURE Correlations(Levels, AU : ADDRESS;
                       NbOfAU     : CARDINAL);
VAR Section, l, Previous, First, Last : CARDINAL;
    Level, Clique, Test               : SetPtr;
BEGIN
   Section:=0;
   GetMem(Test, LONGCARD(_SizeOfSet));
   InitGauge('Correlations', _NLevel);
   FOR l:=1 TO _NLevel DO
      Level:=PointSet(Levels, l, _SizeOfSet);
      IF Section<>Level^.H.Section THEN
         Previous:=MAX(CARDINAL);
         Section:=Level^.H.Section;
         WrLn(_T);
         WrStr(_T, 'Section ');
         WrCard(_T, Section, 0);
         WrLn(_T)
      END;
      WrCard(_T, Level^.H.Level, 3);
      WrStr(_T, ': ');
      Last:=NbOfAU;
      REPEAT
         Clique:=PointSet(AU, Last, _SizeOfSet);
         Set(Subtraction, Level, Clique, Test);
         DEC(Last)
      UNTIL (Last=0) OR (Test^.H.Cardinal=0);
      IF Last=0 THEN
         IF Test^.H.Cardinal<>0 THEN
            WrStr(_T, 'Unknown correlation !')
         ELSE
            WrCard(_T, Clique^.H.Level, 4);
            Previous:=Clique^.H.Level
         END
      ELSE
         First:=Clique^.H.Level;
         REPEAT
            Clique:=PointSet(AU, Last, _SizeOfSet);
            Set(Subtraction, Level, Clique, Test);
            DEC(Last)
         UNTIL (Last=0) OR (Test^.H.Cardinal<>0);
         Last:=NbOfAU-Last;
         IF Test^.H.Cardinal<>0 THEN
            DEC(Last)
         END;
         IF First=Last THEN
            WrCard(_T, First, 4)
         ELSE
            WrCard(_T, First, 4);
            WrStr(_T, ' ..');
            WrCard(_T, Last, 4)
         END;
         IF Previous<First THEN
            WrStr(_T, ' Inversion !')
         END;
         Previous:=Last
      END;
      WrLn(_T);
      Gauge(1)
   END
END Correlations;

BEGIN
   _Quote:=FALSE;
   IO.WrStr(Program);
   IO.WrLn;
   Rd2CmdLine;
   IsCompatibleFiles(_Filename1, 'BGA', _Filename2, 'BGI');
   LoadLevels(_Filename2, 'BGI', AU);
   NbOfAU:=_NLevel;
   LoadLevels(_Filename1, 'BGA', Levels);
   Write2InOut('BGA', 'BGI', Tools);
   WriteTitle;
   OpenTextFile(_Filename2, Tools, Program);
   WrStr(_T, _Filename2);
   WrStr(_T, 'BGI');
   WrStr(_T, ' => ');
   WrStr(_T, _Filename1);
   WrStr(_T, 'BGA');
   WrLn(_T);
   Correlations(Levels, AU, NbOfAU);
   End
END BG_T12.
```

## *Program BG_T13*

Function : Reproducibility table with the unions of UAs

```
MODULE BG_T13;

(*# debug    (vid       => off) *)
(*# check    (nil_ptr   => off
              index     => off
              stack     => off
              overflow  => off
              range     => off) *)
(*# optimize (speed     => on
              cse       => on
              const     => on
              peep_hole => on
              jump      => on
              loop      => on
              alias     => off
              regass    => on
              stk_frame => on
              cpu       => 286
              copro     => emu) *)

IMPORT IO;
FROM FIO      IMPORT Close, EOF, File, OpenRead, PathStr, RdBin, WrCard, WrChar,
                     WrLn, WrCharRep, WrStr, Create, WrBin;
FROM Lib      IMPORT Fill;
FROM BG_TOOLS IMPORT _Filename1, _NSection, _Quote, _Section, _T,
                     CreateFileName, End, Error, ExtStr, Gauge, GetMem, HeadRec,
                     InitGauge, OpenTextFile, Param, ReadSectionLexicon,
                     SwapCard, WriteInOut, WriteTitle;

CONST
   Tools   = 'T13';
   Program = 'BG_'+Tools+' v1.01: reproducibility';

   SECMAX = 131;
   AUMAX  = 254;

TYPE
   corr = RECORD
             Level, FauE, FauI, LauI, LauE : CARDINAL;
             Strict, Coal                  : BOOLEAN;
          END;

VAR
   _R : ARRAY[1..1001] OF corr;
   _A : BOOLEAN;

PROCEDURE HeadTable(max   : CARDINAL;
                    space : BOOLEAN );
VAR loop, i, j : CARDINAL;
BEGIN
    WrLn(_T);
    WrLn(_T);
    IF max<10 THEN
       loop:=0
    ELSIF max<100 THEN
       loop:=1
    ELSE
       loop:=2
    END;
    FOR i:=loop TO 0 BY -1 DO
       WrCharRep(_T, ' ', 4);
       FOR j:=1 TO max DO
          IF space THEN WrChar(_T, ' ') END;
          CASE i OF
          |   0 : WrCard(_T, j MOD 10, 0)
          |   1 : WrCard(_T, (j DIV 10) MOD 10, 0)
          |   2 : WrCard(_T, j DIV 100, 0)
          END
       END;
       WrLn(_T)
    END
END HeadTable;

PROCEDURE Analyze(filename   : PathStr;
                  ext1, ext2 : ExtStr);
TYPE matrix = ARRAY[1..SECMAX] OF ARRAY[1..AUMAX] OF CHAR;
VAR f                                 : File;
    NbAU, NbL, n, e, preS, S, Lmax, L : CARDINAL;
    l                                 : INTEGER;
    R                                 : POINTER TO matrix;
    buffer                            : corr;
BEGIN
   f:=OpenRead(CreateFileName(filename, ext2));
   IF RdBin(f, NbAU, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(1002) END;
   IF RdBin(f, NbAU, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(1002) END;
   IF RdBin(f, NbAU, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(1002) END;
   Close(f);
   IF NbAU>AUMAX THEN Error(1009) END;
   f:=OpenRead(CreateFileName(filename, ext1));
   IF RdBin(f, NbL, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(1002) END;
   n:=0;
   WrCard(_T, NbL, 5); WrStr(_T, ' levels'); WrLn(_T);
WrCard(_T, NbAU, 5); WrStr(_T, ' united associations'); WrLn(_T);
WrLn(_T);
InitGauge('Correlations', NbL);
GetMem(R, SIZE(matrix));
Fill(R, SIZE(matrix), 249);
preS:=0;
REPEAT
   IF preS=0 THEN
      L:=0
   ELSE
      L:=1;
      _R[L]:=_R[Lmax+1]
   END;
   REPEAT
      IF RdBin(f, S, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(1002) END;
      IF preS=0 THEN preS:=S END;
      INC(L);
      IF RdBin(f, _R[L].Level, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(1002) END;
      IF RdBin(f, _R[L].FauE, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(1002) END;
      IF RdBin(f, _R[L].LauE, SIZE(CARDINAL))<>SIZE(CARDINAL) THEN Error(1002) END;
      _R[L].FauI:=_R[L].FauE;
      _R[L].LauI:=_R[L].LauE;
      _R[L].Strict:=_R[L].FauE=_R[L].LauE;
      _R[L].Coal:=FALSE;
      INC(n)
   UNTIL (S<>preS) OR (n=NbL);
   SwapCard(preS, S);
   Gauge(n);
   IF n<NbL THEN Lmax:=L-1 ELSE Lmax:=L END;
   FOR L:=1 TO Lmax DIV 2 DO
      buffer:=_R[L];
      _R[L]:=_R[Lmax-L+1];
      _R[Lmax-L+1]:=buffer
   END;
```

troncature

```
   FOR L:=1 TO Lmax-1 DO
      FOR l:=L+1 TO Lmax DO
         IF _R[l].LauE<_R[L].LauI THEN
            _R[L].LauI:=_R[l].LauE
         END
      END
   END;
   FOR L:=Lmax TO 2 BY -1 DO
      FOR l:=L-1 TO 1 BY -1 DO
         IF _R[L].FauI<_R[l].FauE THEN
            _R[L].FauI:=_R[l].FauE
         END
      END
   END;
```

coalescence

```
   FOR L:=1 TO Lmax DO
      IF (NOT _R[L].Strict) AND
         (_R[L].FauE=_R[L].FauI) AND (_R[L].LauI=_R[L].LauE) THEN
         _R[L].Coal:=TRUE;
         FOR l:=L-1 TO 1 BY -1 DO
            IF _R[l].Strict AND (_R[l].LauE=_R[L].FauE) THEN
               _R[L].Coal:=FALSE
            END
         END;
         IF _R[L].Coal THEN
            FOR l:=L+1 TO Lmax DO
               IF _R[l].Strict AND (_R[L].LauE=_R[l].FauE) THEN
                  _R[L].Coal:=FALSE
               END
            END
         END
      END
   END;
```

limitation to the included coalescences

```
   FOR L:=1 TO Lmax-1 DO
      l:=L+1;
      WHILE _R[L].Coal AND (CARDINAL(l)<=Lmax) AND (_R[l].FauE<_R[L].LauE) DO
         IF _R[l].Coal THEN
            IF    ((_R[L].FauE> _R[l].FauE) AND (_R[L].LauE<=_R[l].LauE)) OR
```

```
                     ((_R[L].FauE>=_R[l].FauE) AND (_R[L].LauE< _R[l].LauE)) THEN
                     _R[l].Coal:=FALSE
               ELSIF ((_R[L].FauE< _R[l].FauE) AND (_R[L].LauE>=_R[l].LauE)) OR
                     ((_R[L].FauE<=_R[l].FauE) AND (_R[L].LauE> _R[l].LauE)) THEN
                     _R[L].Coal:=FALSE
               END
            END;
            INC(l)
         END
      END;
      FOR L:=1 TO Lmax DO
         IF _R[L].Strict THEN
            R^[S,_R[L].FauE]:='_'
         ELSIF _R[L].Coal THEN
            R^[S, _R[L].FauE]:='-';
            FOR l:=_R[L].FauE+1 TO _R[L].LauE-1 DO
               IF R^[S,l]='•' THEN R^[S,l]:='_' END
            END;
            R^[S, _R[L].LauE]:='-'
         END
      END;
      IF _A THEN
         WrLn(_T);
         WrStr(_T, 'Section ');
         WrCard(_T, S, 0);
         WrStr(_T, ': ');
         WrStr(_T, _Section^[S]);
         HeadTable(NbAU, FALSE);
         FOR L:=Lmax TO 1 BY -1 DO
            WrCard(_T, _R[L].Level, 3);
            WrChar(_T, '_');
            WrCharRep(_T, '•', _R[L].FauE-1);
            WrCharRep(_T, '_', _R[L].FauI-_R[L].FauE);
            IF _R[L].Strict THEN
               WrCharRep(_T, '_', 1)
            ELSIF _R[L].Coal THEN
               WrCharRep(_T, '_', _R[L].LauI-_R[L].FauI+1)
            ELSE
               WrCharRep(_T, '_', _R[L].LauI-_R[L].FauI+1)
            END;
            WrCharRep(_T, '_', _R[L].LauE-_R[L].LauI);
            WrCharRep(_T, '•', NbAU-_R[L].LauE);
            WrChar(_T, '_');
            WrLn(_T)
         END
      END
   UNTIL n=NbL;
   WrLn(_T);
   WrStr(_T, 'Reproducibility table');
   HeadTable(S, TRUE);
   FOR l:=NbAU TO 1 BY -1 DO
      WrCard(_T, l, 4);
      FOR L:=1 TO S DO
         WrChar(_T, ' ');
         WrChar(_T, R^[L,l])
      END;
      WrLn(_T)
   END;
   Close(f);
   f:=Create(CreateFileName(filename, 'B13'));
   WrBin(f, NbAU, SIZE(CARDINAL));
   FOR l:=NbAU TO 1 BY -1 DO
      WrBin(f, S, SIZE(SHORTCARD));
      FOR L:=1 TO S DO WrBin(f, R^[L,l], SIZE(CHAR)) END
   END;
   Close(f)
END Analyze;

BEGIN
   _Quote:=FALSE;
   IO.WrStr(Program);
   IO.WrLn;
   WriteInOut('BGK', Tools);
   _A:=Param(2, 'A');
   IF _A THEN
      IO.WrStr('Reproducibility by section and ')
   END;
   IO.WrStr('Reproducibility table');
   IO.WrLn;
   ReadSectionLexicon(_Filename1);
   WriteTitle;
   IF SECMAX<_NSection THEN Error(1008) END;
   OpenTextFile(_Filename1, Tools, Program);
   Analyze(_Filename1, 'BGK', 'BGI');
   End
END BG_T13.
```

## Program BG_T14

Function : list of contradictions between two given MLH

```
MODULE BG_T14;

(*# debug    (vid       => off) *)
(*# check    (nil_ptr   => off
              index     => off
              stack     => off
              overflow  => off
              range     => off) *)
(*# optimize (speed     => on
              cse       => on
              const     => on
              peep_hole => on
              jump      => on
              loop      => on
              alias     => off
              regass    => on
              stk_frame => on
              cpu       => 286
              copro     => emu) *)

IMPORT IO;
FROM FIO      IMPORT File, Close, Create, Erase, OpenRead, RdBin, WrBin, WrCard,
                     WrChar, WrCharRep, WrLn, WrLngCard, WrStr, WrStrAdj;
FROM Lib      IMPORT AddAddr, Fill, IncAddr, ParamCount, ParamStr;
FROM Str      IMPORT CHARSET, Item, StrToCard;
FROM BG_TOOLS IMPORT EDGE, _Filename1, _NLevel, _NTaxa, _Quote, _T, _Taxa,
                     DiagIndex, End, Error, Gauge, GetMem, HeadRec,
                     Intersection, IsCompatibleFiles, InitGauge, IntPtr,
                     LoadDiagMat, LoadLevels, OpenTextFile, ReadLexicon, Set,
                     SetPtr, SizeOfSet, Subtraction, SwapCard,TaxaRange,
                     WriteInOut, WriteTitle;

CONST
   Tools   = 'T14';
   Program = 'BG_'+Tools+' v1.02: 2 local maximal horizons relationships';

VAR
   L, A   : FarADDRESS;
   L1, L2 : HeadRec;

PROCEDURE GetLevels(VAR l1, l2 : HeadRec);

   PROCEDURE Translate(p : CARDINAL;
                       VAR l : HeadRec);
   VAR s, a, b : ARRAY[0..10] OF CHAR;
       ok      : BOOLEAN;
   BEGIN
      ParamStr(s, p);
      Item(a, s, CHARSET{'.'}, 0);
      Item(b, s, CHARSET{'.'}, 1);
      l.Section:=CARDINAL(StrToCard(a, 10, ok));
      IF NOT ok THEN Error(1007) END;
      l.Level:=CARDINAL(StrToCard(b, 10, ok));
      IF NOT ok THEN Error(1007) END
   END Translate;

VAR s : ARRAY[0..10] OF CHAR;
BEGIN
   IF ParamCount()<3 THEN Error(1003) END;
   Translate(2, l1);
   Translate(3, l2);
   IF l1.Section=l2.Section THEN Error(1007) END;
   IF l2.Section<l1.Section THEN
      SwapCard(l1.Section, l2.Section);
      SwapCard(l1.Level, l2.Level)
   END
END GetLevels;
```

```
PROCEDURE WrHeader(f : File;
                   h : HeadRec);
BEGIN
   WrCard(f, h.Section, 0);
   WrChar(f, '.');
   WrCard(f, h.Level, 0)
END WrHeader;

PROCEDURE WrPair(f      : File;
                 t1, t2 : TaxaRange;
                 c      : CHAR;
             VAR i      : LONGCARD);
BEGIN
   IF i MOD 5=0 THEN WrLn(f) END;
   WrCharRep(f, ' ', 3);
   WrStrAdj(f, _Taxa^[t1], 5);
   WrChar(f, c);
   WrStrAdj(f, _Taxa^[t2], -5);
   INC(i)
END WrPair;

PROCEDURE WrNumber(f : File;
                   i : LONGCARD);
BEGIN
   WrLn(f);
   IF 0<i THEN WrLn(f) END;
   WrStr(f, 'Total: ');
   WrLngCard(f, i, 0);
   WrLn(f)
END WrNumber;

PROCEDURE Append(FName : ARRAY OF CHAR);
CONST S = 4096;
VAR f : File;
    b : ARRAY[0..S] OF BYTE;
    s : CARDINAL;
BEGIN
   f:=OpenRead(FName);
   REPEAT
      s:=RdBin(f, b, S);
      WrBin(_T, b, s)
   UNTIL s=0;
   Close(f);
   Erase(FName)
END Append;

PROCEDURE Analyze(levels, mat : FarADDRESS;
                  h1, h2      : HeadRec);
VAR l1, l2, s  : SetPtr;
    size       : CARDINAL;
    t1, t2     : TaxaRange;
    arc        : IntPtr;
    l          : CARDINAL;
    a, i, e    : LONGCARD;
    fa, fi     : File;
    stat       : ARRAY[1..500] OF RECORD
                    e, a, i : LONGCARD
                 END;
BEGIN
   InitGauge('Check 2 local residual horizons', _NTaxa);
   size:=SizeOfSet(_NTaxa);
   GetMem(s, LONGCARD(size));
   l1:=levels;
   l:=0;
   WHILE ((l1^.H.Section<>h1.Section) OR (l1^.H.Level<>h1.Level)) AND
         (l<_NLevel) DO
      IncAddr(l1, size);
      INC(l)
   END;
   IF _NLevel<=l THEN Error(1007) END;
   l2:=AddAddr(l1, size);
   INC(l);
   WHILE (l2^.H.Section<>h2.Section) OR (l2^.H.Level<>h2.Level) AND
         (l<_NLevel) DO
      IncAddr(l2, size);
      INC(l)
   END;
   IF _NLevel<=l THEN Error(1007) END;
   Set(Intersection, l1, l2, s);
   Set(Subtraction, l1, s, l1);
   Set(Subtraction, l2, s, l2);
```

```
fa:=Create('A.$$$');
fi:=Create('I.$$$');

WrLn(fa);
WrStr(fa, 'List OF arcs: ');
WrHeader(fa, h1);
WrStr(fa, '->');
WrHeader(fa, h2);
WrLn(fa);

WrLn(_T);
WrStr(_T, 'List OF edges: ');
WrHeader(_T, h1);
WrChar(_T, '=');
WrHeader(_T, h2);
WrLn(_T);

WrLn(fi);
WrStr(fi, 'List OF arcs: ');
WrHeader(fi, h1);
WrStr(fi, '<-');
WrHeader(fi, h2);
WrLn(fi);

a:=0; e:=0; i:=0;
Fill(ADR(stat), SIZE(stat), 0);

FOR t1:=1 TO _NTaxa DO
   IF (t1-1) IN l1^.Set THEN
      FOR t2:=1 TO _NTaxa DO
         IF (t2-1) IN l2^.Set THEN
            arc:=DiagIndex(mat, _NTaxa, t1, t2);
            IF arc^=EDGE THEN
               WrPair(_T, t1, t2, '=', e);
               INC(stat[t1].e);
               INC(stat[t2].e)
            ELSIF arc^<>0 THEN
               IF t1<t2 THEN
                  IF arc^<0 THEN
                     WrPair(fa, t1, t2, '>', a);
                     INC(stat[t1].a);
                     INC(stat[t2].a)
                  ELSE
                     WrPair(fi, t1, t2, '<', i);
                     INC(stat[t1].i);
                     INC(stat[t2].i)
                  END
               ELSE
                  IF 0<arc^ THEN
                     WrPair(fa, t1, t2, '>', a);
                     INC(stat[t1].a);
                     INC(stat[t2].a)
                  ELSE
                     WrPair(fi, t1, t2, '<', i);
                     INC(stat[t1].i);
                     INC(stat[t2].i)
                  END
               END
            END
         END
      END
   END;
   Gauge(t1)
END;
IO.WrLn;
IO.WrStr('Writing...');
WrNumber(_T, e);
WrNumber(fa, a);
WrNumber(fi, i);
Close(fa);
Close(fi);
Append('A.$$$');
Append('I.$$$');
WrLn(_T);
WrStr(_T, 'Statistics:');
WrLn(_T);
WrStr(_T, 'Taxa      =    ->    <-');
WrLn(_T);
FOR t1:=1 TO _NTaxa DO
   IF (0<stat[t1].e) OR (0<stat[t1].a) OR (0<stat[t1].i) THEN
      WrStrAdj(_T, _Taxa^[t1], 5);
```

```
         WrLngCard(_T, stat[t1].e, 6);
         WrLngCard(_T, stat[t1].a, 6);
         WrLngCard(_T, stat[t1].i, 6);
         WrLn(_T)
      END
   END;
   WrStr(_T, 'Total');
   WrLngCard(_T, 2*e, 6);
   WrLngCard(_T, 2*a, 6);
   WrLngCard(_T, 2*i, 6);
   WrLn(_T)
END Analyze;

BEGIN
   _Quote:=FALSE;
   IO.WrStr(Program);
   IO.WrLn;
   WriteInOut('BGB/BGD', Tools);
   GetLevels(L1, L2);
   IsCompatibleFiles(_Filename1, 'BGD', _Filename1, 'BGB');
   LoadLevels(_Filename1, 'BGB', L);
   LoadDiagMat(_Filename1, 'BGD', A);
   ReadLexicon(_Filename1, FALSE);
   WriteTitle;
   OpenTextFile(_Filename1, Tools, Program);
   WrHeader(_T, L1);
   WrStr(_T, ' checked with ');
   WrHeader(_T, L2);
   WrLn(_T);
   Analyze(L, A, L1, L2);
   End
END BG_T14.
```

## *Library BG_TOOLS*

Function : common modules

```
DEFINITION MODULE BG_TOOLS;

FROM FIO IMPORT File, PathStr;

TYPE
   TaxaRange   = [1..500];

   ExtStr      = ARRAY[0..  3] OF CHAR;
   TitleStr    = ARRAY[0..255] OF CHAR;
   TaxaStr     = ARRAY[0..  5] OF CHAR;
   TaxaList    = ARRAY TaxaRange OF TaxaStr;
   TaxaLPtr    = POINTER TO TaxaList;
   SectionStr  = ARRAY[0.. 25] OF CHAR;
   SectionList = ARRAY[1..1000] OF SectionStr;
   SectionLPtr = POINTER TO SectionList;

   HeadRec     = RECORD
                   Section, Level, Cardinal : CARDINAL
                 END;
   SetRec      = RECORD
                   H : HeadRec;
                   CASE : BOOLEAN OF
                   | TRUE  : Set : SET OF TaxaRange;
                   | FALSE : Bit : ARRAY[0..31] OF BITSET
                   END
                 END;
   SetPtr      = POINTER TO SetRec;
   SetProc     = PROCEDURE(BITSET, BITSET) : BITSET;

   IntPtr      = POINTER TO INTEGER;

CONST
   EDGE = MIN(INTEGER);

VAR
   _T                                       : File;
   _NTaxa, _NSection, _NLevel, _SizeOfSet : CARDINAL;
   _Taxa                                    : TaxaLPtr;
   _Section                                 : SectionLPtr;
   _Quote                                 : BOOLEAN;
   _Filename1, _Filename2                 : PathStr;

PROCEDURE Error(error : CARDINAL);
PROCEDURE Rd2CmdLine;
PROCEDURE Param(p : CARDINAL;
                c : CHAR) : BOOLEAN;
PROCEDURE SwapCard(VAR a, b : CARDINAL);

PROCEDURE WriteInOut(ext1, ext2 : ARRAY OF CHAR);
PROCEDURE Write2InOut(ext1, ext2, ext3 : ARRAY OF CHAR);
PROCEDURE WriteTitle;

PROCEDURE CreateFileName(name, ext : ARRAY OF CHAR) : PathStr;
PROCEDURE OpenTextFile(filename : PathStr;
                       ext      : ExtStr;
                       s        : ARRAY OF CHAR);
PROCEDURE WrTaxa(t : TaxaRange);
PROCEDURE End;

PROCEDURE InitGauge(s : ARRAY OF CHAR;
                    g : CARDINAL );
PROCEDURE Gauge(g : CARDINAL);

PROCEDURE SizeOfSet(bits : CARDINAL) : CARDINAL;
PROCEDURE PointSet(base      : ADDRESS;
                   set, size : CARDINAL) : ADDRESS;
PROCEDURE PutInSet(Element : CARDINAL;
                   Set     : SetPtr);
PROCEDURE Union(Set1, Set2 : BITSET) : BITSET;
PROCEDURE Intersection(Set1, Set2 : BITSET) : BITSET;
PROCEDURE Subtraction(Set1, Set2 : BITSET) : BITSET;
PROCEDURE Set(Do               : SetProc;
              Set1, Set2, Set : SetPtr);

PROCEDURE NewDiagMat(VAR mat  : ADDRESS;
                         size : CARDINAL);
PROCEDURE DiagIndex(mat         : ADDRESS;
                    size, i, j : CARDINAL) : IntPtr;

PROCEDURE GetMem(VAR ptr   : ADDRESS;
                     count : LONGCARD);
PROCEDURE ClearMem(ptr   : ADDRESS;
                   count : LONGCARD);

PROCEDURE IsCompatibleFiles(filename1, ext1, filename2, ext2 : ARRAY OF CHAR);
PROCEDURE ReadLexicon(filename : PathStr;
                      just     : BOOLEAN);
PROCEDURE ReadSectionLexicon(filename : PathStr);

PROCEDURE LoadLevels(filename : PathStr;
                     ext      : ExtStr;
                 VAR level    : ADDRESS);

PROCEDURE LoadDiagMat(filename : PathStr;
                      ext      : ExtStr;
                  VAR mat      : ADDRESS);

END BG_TOOLS.

IMPLEMENTATION MODULE BG_TOOLS;

IMPORT IO;
FROM FIO     IMPORT Close, Create, Exists, File, GetPos, IOcheck, OpenRead,
                    RdBin, Seek, WrChar, WrLn, WrStr;
FROM Lib     IMPORT AddAddr, IncAddr, Move, ParamCount, ParamStr, RunTimeError,
                    SetReturnCode, SysErrno, WordFill;
FROM Storage IMPORT ALLOCATE, ClearOnAllocate, HeapAllocate, HeapAvail,
                    MainHeap;
FROM Str     IMPORT CHARSET, Append, Caps, CharPos;
FROM Window  IMPORT Black, GotoXY, LightGray, Open, RelCoord, WhereX, WhereY,
                    WinDef, WinType;

TYPE
   ErrorStr     = ARRAY[0.. 64] OF CHAR;
   HeadPtr      = POINTER TO HeadRec;

CONST
   BUFFER    = 4096;
   Copyright = 'BIOGRAPH Copyright (c) 1991 by J.Savary & J.Guex';

VAR
```

```
    _Title         : TitleStr;
    _Gauge, _X, _Y : CARDINAL;
```

## Utilities

```
PROCEDURE ExitOnError(code : CARDINAL;
                      msg  : ARRAY OF CHAR);
BEGIN
   IO.WrLn;
   IF code<>0 THEN
      IO.WrStr('ERROR ');
      IO.WrCard(code, 0);
      IO.WrStr(': ')
   END;
   IO.WrStr(msg);
   IO.WrLn;
```

### Internal error

```
   IF 255<code THEN code:=255 END;
   SetReturnCode(SHORTCARD(code));
   HALT
END ExitOnError;

PROCEDURE FatalError(erraddr : LONGCARD;
                     code    : CARDINAL;
                     msg     : ARRAY OF CHAR);
VAR m : ErrorStr;
BEGIN
   code:=SysErrno();
   CASE code OF
     |    0 : m:='Normal termination'
     |    1 : m:='Invalid FUNCTION number'
     |    2 : m:='File not found'
     |    3 : m:='Path not found'
     |    4 : m:='Too many open files (no handles left)'
     |    5 : m:='Access denied'
     |    6 : m:='Invalid handle'
     |    7 : m:='Memory control blocks destroyed'
     |    8 : m:='Insufficient memory'
     |    9 : m:='Invalid memory block address'
     |   10 : m:='Invalid environment'
     |   11 : m:='Invalid format'
     |   12 : m:='Invalid access code'
     |   13 : m:='Invalid data'
     (*  14 : Reserved *)
     |   15 : m:='Invalid drive was specified'
     |   16 : m:='Attempt TO remove the current directory'
     |   17 : m:='Not same device'
     |   18 : m:='No more files'
     |   19 : m:='Attempt TO write on write-protected diskette'
     |   20 : m:='Unknown unit'
     |   21 : m:='Drive not ready'
     |   22 : m:='Unknown command'
     |   23 : m:='Data error (CRC)'
     |   24 : m:='Bad request structure length'
     |   25 : m:='Seek error'
     |   26 : m:='Unknown media TYPE'
     |   27 : m:='Sector not found'
     |   28 : m:='Printer out OF paper'
     |   29 : m:='Write fault'
     |   30 : m:='Read fault'
     |   31 : m:='General failure'
     |   32 : m:='Sharing Violation'
     |   33 : m:='Lock Violation'
     |   34 : m:='Invalid disk change'
     |   35 : m:='FCB unavailable'
     (*  36..79 : Reserved *)
     |   80 : m:='File exists'
     (*  81 : Reserved *)
     |   82 : m:='Cannot Make'
     |   83 : m:='Fail on INT 24'
     | 0F0H : m:='Disk Full (write failed)' (* JPI internal *)
   ELSE
      m:='Unexpected error'
   END;
   ExitOnError(code, m)
END FatalError;

PROCEDURE Error(code : CARDINAL);
```

```
VAR m : ErrorStr;
BEGIN
   CASE code OF
      |    0 : m:='Normal termination'
      | 1001 : m:='Lexicon file not found'
      | 1002 : m:='Disk read error'
      | 1003 : m:='Invalid command line'
      | 1004 : m:='Not enough memory'
      | 1005 : m:='Not compatible files'
      | 1006 : m:='Dictionnary file not found'
      | 1007 : m:="Invalid levels code"
      | 1008 : m:='Too many sections'
      | 1009 : m:='Too many AU'
   ELSE
       m:='Unexpected error'
   END;
   ExitOnError(code, m)
END Error;

PROCEDURE Initialization;
VAR w : WinType;
BEGIN
   w:=Open(WinDef(9, 4, 70, 20, Black, LightGray, TRUE, TRUE, FALSE, TRUE,
                  '_________', LightGray, Black));
   _Title[0]:=CHR(0);
   _Taxa:=NIL;
   _NTaxa:=0;
   _NSection:=0;
   _NLevel:=0;
   _SizeOfSet:=0;
   _Quote:=TRUE
END Initialization;

PROCEDURE RdCmdLine;
BEGIN
   IF ParamCount()<1 THEN Error(1003) END;
   ParamStr(_Filename1, 1)
END RdCmdLine;

PROCEDURE Rd2CmdLine;
BEGIN
   IF ParamCount()<2 THEN Error(1003) END;
   ParamStr(_Filename2, 2)
END Rd2CmdLine;

PROCEDURE Param(p : CARDINAL;
                c : CHAR) : BOOLEAN;
VAR cmd : ARRAY[0..5] OF CHAR;
BEGIN
   IF ParamCount()<p THEN
      RETURN FALSE
   END;
   ParamStr(cmd, p);
   IF cmd[0]<>'-' THEN Error(1003) END;
   Caps(cmd);
   RETURN CharPos(cmd, c)<>MAX(CARDINAL)
END Param;

PROCEDURE WriteInOut(ext1, ext2 : ARRAY OF CHAR);
BEGIN
   IO.WrLn;
   IO.WrStr(_Filename1);
   IO.WrStr(ext1);
   IO.WrStr(' -> ');
   IO.WrStr(_Filename1);
   IO.WrStr(ext2);
   IO.WrLn
END WriteInOut;

PROCEDURE Write2InOut(ext1, ext2, ext3 : ARRAY OF CHAR);
BEGIN
   IO.WrLn;
   IO.WrStr(_Filename1);
   IO.WrStr(ext1);
   IO.WrStr(' + ');
   IO.WrStr(_Filename2);
   IO.WrStr(ext2);
   IO.WrStr(' -> ');
   IO.WrStr(_Filename2);
   IO.WrStr(ext3);
   IO.WrLn
END Write2InOut;
```

```
PROCEDURE WriteTitle;
BEGIN
   IO.WrStr(_Title);
   IO.WrLn
END WriteTitle;

PROCEDURE SwapCard(VAR a, b : CARDINAL);
VAR c : CARDINAL;
BEGIN
   c:=a; a:=b; b:=c
END SwapCard;

PROCEDURE CreateFileName(name, ext : ARRAY OF CHAR) : PathStr;
BEGIN
   Append(name, ext);
   RETURN PathStr(name)
END CreateFileName;
```

## Output

```
PROCEDURE OpenTextFile(filename : PathStr;
                       ext      : ExtStr;
                       s        : ARRAY OF CHAR);
BEGIN
   _T:=Create(CreateFileName(filename, ext));
   IF _Quote THEN WrChar(_T, '"') END;
   WrStr(_T, Copyright);
   IF _Quote THEN WrChar(_T, '"') END;
   WrLn(_T);
   IF _Quote THEN WrChar(_T, '"') END;
   WrStr(_T, s);
   IF _Quote THEN WrChar(_T, '"') END;
   WrLn(_T);
   WrLn(_T);
   IF _Quote THEN WrChar(_T, '"') END;
   WrStr(_T, _Title);
   IF _Quote THEN WrChar(_T, '"') END;
   WrLn(_T);
   WrLn(_T)
END OpenTextFile;

PROCEDURE WrTaxa(t : TaxaRange);
BEGIN
   IF _Quote THEN WrChar(_T, '"') END;
   WrStr(_T, _Taxa^[t]);
   IF _Quote THEN WrChar(_T, '"') END
END WrTaxa;

PROCEDURE End;
BEGIN
   Close(_T);
   Error(0)
END End;
```

## Progress bar

```
PROCEDURE InitGauge(s : ARRAY OF CHAR;
                    g : CARDINAL );
BEGIN
   IO.WrLn;
   IO.WrStr(s);
   IO.WrStr(':   0 %');
   _X:=WhereX()-5;
   _Y:=WhereY();
   _Gauge:=g
END InitGauge;

PROCEDURE Gauge(g : CARDINAL);
BEGIN
   GotoXY(_X, _Y);
   IF _Gauge<g THEN
      IO.WrStr('Overflow !')
   ELSE
      IO.WrFixReal(100.0*FLOAT(g)/FLOAT(_Gauge), 0, 3)
   END
END Gauge;
```

## Sets functions

```
PROCEDURE SizeOfSet(bits : CARDINAL) : CARDINAL;
BEGIN
   RETURN 2*(((bits-1) >> 4)+1) + SIZE(HeadRec)
END SizeOfSet;

PROCEDURE PointSet(base       : ADDRESS;
                   set, size : CARDINAL) : ADDRESS;
VAR l : LONGCARD;
BEGIN
   l:=LONGCARD(set-1)*LONGCARD(size);
   WHILE MAX(CARDINAL)<l DO
      IncAddr(base, MAX(CARDINAL));
      DEC(l, MAX(CARDINAL))
   END;
   RETURN AddAddr(base, CARDINAL(l))
END PointSet;

PROCEDURE PutInSet(Element : CARDINAL;
                   Set     : SetPtr);
BEGIN
   INC(Set^.H.Cardinal);
   INCL(Set^.Set, Element)
END PutInSet;

PROCEDURE Union(Set1, Set2 : BITSET) : BITSET;
BEGIN
   RETURN Set1+Set2
END Union;

PROCEDURE Intersection(Set1, Set2 : BITSET) : BITSET;
BEGIN
   RETURN Set1*Set2
END Intersection;

PROCEDURE Subtraction(Set1, Set2 : BITSET) : BITSET;
BEGIN
   RETURN Set1-Set2
END Subtraction;

PROCEDURE Set(Do               : SetProc;
              Set1, Set2, Set : SetPtr);
VAR W, bit : CARDINAL;
BEGIN
   Set^.H.Cardinal:=0;
   FOR W:=0 TO ((_SizeOfSet-SIZE(HeadRec)) >> 1)-1 DO
      Set^.Bit[W]:=Do(Set1^.Bit[W], Set2^.Bit[W]);
      FOR bit:=0 TO 15 DO INC(Set^.H.Cardinal, ORD(bit IN Set^.Bit[W])) END
   END
END Set;
```

## Half matrix

```
PROCEDURE SizeOfDiagMat(size : CARDINAL) : LONGCARD;
BEGIN
   RETURN LONGCARD(size)*LONGCARD(size-1)
END SizeOfDiagMat;

PROCEDURE NewDiagMat(VAR mat  : ADDRESS;
                         size : CARDINAL);
BEGIN
   GetMem(mat, SizeOfDiagMat(size))
END NewDiagMat;

PROCEDURE DiagIndex(mat          : ADDRESS;
                    size, i, j : CARDINAL) : IntPtr;
VAR l : LONGCARD;
BEGIN
   IF j<i THEN SwapCard(i, j) END;
   l:=(LONGCARD(i-1)*LONGCARD(size) + LONGCARD(j-1)
      - ((LONGCARD(i)*LONGCARD(i+1)) >> 1)) << 1;
   WHILE MAX(CARDINAL)<l DO
      IncAddr(mat, MAX(CARDINAL));
      DEC(l, MAX(CARDINAL))
   END;
   RETURN AddAddr(mat, CARDINAL(l))
END DiagIndex;
```

## Memory

```
PROCEDURE GetMem(VAR ptr   : ADDRESS;
                     count : LONGCARD);
VAR size : LONGCARD;
BEGIN
   size:=((count-1) >> 4)+1;
   IF HeapAvail(MainHeap)<CARDINAL(size) THEN Error(1004) END;
   HeapAllocate(MainHeap, ptr, CARDINAL(size));
   ClearMem(ptr, count)
END GetMem;

PROCEDURE ClearMem(ptr   : ADDRESS;
                   count : LONGCARD);
BEGIN
   IF ODD(count) THEN INC(count) END;
   WHILE MAX(CARDINAL)-1<count DO
      WordFill(ptr, MAX(CARDINAL)>>1, 0);
      IncAddr(ptr, MAX(CARDINAL)-1);
      DEC(count, MAX(CARDINAL)-1)
   END;
   WordFill(ptr, CARDINAL(count>>1), 0)
END ClearMem;
```

## Files

```
PROCEDURE ReadFile(f : File;
                   v : ADDRESS;
                   s : CARDINAL);
VAR read : CARDINAL;
BEGIN
   read:=RdBin(f, v^, s);
   IF read<>s THEN Error(1002) END
END ReadFile;

PROCEDURE IsCompatibleFiles(filename1, ext1, filename2, ext2 : ARRAY OF CHAR);
VAR f : File;
    n : CARDINAL;
BEGIN
   f:=OpenRead(CreateFileName(filename1, ext1));
   ReadFile(f, ADR(_NTaxa), SIZE(CARDINAL));
   Close(f);
   f:=OpenRead(CreateFileName(filename2, ext2));
   ReadFile(f, ADR(n), SIZE(CARDINAL));
   Close(f);
   IF n<>_NTaxa THEN Error(1005) END
END IsCompatibleFiles;

PROCEDURE DoString(VAR s : ARRAY OF CHAR);
VAR l : CARDINAL;
BEGIN
   l:=CARDINAL(s[0]);
   Move(ADR(s[1]), ADR(s[0]), l);
   s[l]:=CHR(0)
END DoString;

PROCEDURE ReadLexicon(filename : PathStr;
                      just     : BOOLEAN);
VAR f    : File;
    i, j : CARDINAL;
BEGIN
   IF NOT Exists(CreateFileName(filename, 'LEX')) THEN Error(1001) END;
   f:=OpenRead(CreateFileName(filename, 'LEX'));
   ReadFile(f, ADR(_Title), SIZE(_Title));
   DoString(_Title);
   IF _Title[0]=CHR(0) THEN _Title:='Untitled' END;
   ReadFile(f, ADR(_NSection), SIZE(_NSection));
   ReadFile(f, ADR(_NTaxa   ), SIZE(_NTaxa   ));
   Seek(f, GetPos(f)+LONGCARD(_NSection*SIZE(SectionStr)));
   ALLOCATE(_Taxa, _NTaxa*SIZE(TaxaStr));
   ReadFile(f, _Taxa, _NTaxa*SIZE(TaxaStr));
   FOR i:=1 TO _NTaxa DO
      IF just THEN
         FOR j:=ORD(_Taxa^[i,0])+1 TO SIZE(TaxaStr)-1 DO
            _Taxa^[i,j]:=CHR(32)
         END;
         _Taxa^[i,0]:=CHR(SIZE(TaxaStr)-1)
      END;
      DoString(_Taxa^[i])
   END;
   Close(f)
END ReadLexicon;
```

```
PROCEDURE ReadSectionLexicon(filename : PathStr);
VAR f : File;
    i : CARDINAL;
BEGIN
   IF NOT Exists(CreateFileName(filename, 'LEX')) THEN Error(1001) END;
   f:=OpenRead(CreateFileName(filename, 'LEX'));
   ReadFile(f, ADR(_Title), SIZE(_Title));
   DoString(_Title);
   IF _Title[0]=CHR(0) THEN _Title:='Untitled' END;
   ReadFile(f, ADR(_NSection), SIZE(_NSection));
   ReadFile(f, ADR(_NTaxa   ), SIZE(_NTaxa   ));
   ALLOCATE(_Section, _NSection*SIZE(SectionStr));
   ReadFile(f, _Section, _NSection*SIZE(SectionStr));
   FOR i:=1 TO _NSection DO
      DoString(_Section^[i])
   END;
   Close(f)
END ReadSectionLexicon;

PROCEDURE LoadFile(f    : File;
                   ptr  : ADDRESS;
                   size : LONGCARD);
BEGIN
   WHILE BUFFER<size DO
      ReadFile(f, ptr, BUFFER);
      IncAddr(ADDRESS(ptr), BUFFER);
      DEC(size, BUFFER)
   END;
   ReadFile(f, ptr, CARDINAL(size));
   Close(f)
END LoadFile;

PROCEDURE LoadLevels(filename : PathStr;
                     ext      : ExtStr;
                 VAR level    : ADDRESS);
VAR f : File;
    n : CARDINAL;
BEGIN
   f:=OpenRead(CreateFileName(filename, ext));
   ReadFile(f, ADR(n), SIZE(CARDINAL));
   IF n<>_NTaxa THEN Error(1005) END;
   ReadFile(f, ADR(n), SIZE(CARDINAL));
   ReadFile(f, ADR(_NLevel), SIZE(CARDINAL));
   _SizeOfSet:=SizeOfSet(_NTaxa);
   GetMem(level, LONGCARD(_NLevel)*LONGCARD(_SizeOfSet)+SIZE(CARDINAL));
   LoadFile(f, level, LONGCARD(_NLevel)*LONGCARD(_SizeOfSet))
END LoadLevels;

PROCEDURE LoadDiagMat(filename : PathStr;
                      ext      : ExtStr;
                  VAR mat      : ADDRESS);
VAR f : File;
    n : CARDINAL;
BEGIN
   f:=OpenRead(CreateFileName(filename, ext));
   ReadFile(f, ADR(n), SIZE(CARDINAL));
   IF n<>_NTaxa THEN Error(1005) END;
   NewDiagMat(mat, _NTaxa);
   LoadFile(f, mat, SizeOfDiagMat(_NTaxa))
END LoadDiagMat;

BEGIN
   RunTimeError:=FatalError;
   Initialization;
   IO.WrStr(Copyright); IO.WrLn;
   RdCmdLine;
   ClearOnAllocate:=TRUE
END BG_TOOLS.
```

# 6 References

Baumgartner PO (1984) A middle Jurassic-Early Cretaceous low-latitude zonation based on unitary associations and age of Tethyan radiolarites. Eclog. Geol. Helv. 77/3: 729-837

Baumgartner PO, O'Dogherty L, Gorican S, Urquhart E, Pillevuit A, DeWever P (eds) (1995) Middle Jurassic to Early Cretaceous Radiolaria of Tethys: Occurrences, Systematics, Biochronology. Mém. Géol. Lausanne, 23: 1-1172

Berge C (1973) Graphes et Hypergraphes (2nd Edn.). Dunod Paris. 576 p.

Blackham M (1998) The Unitary Association Method of Relative Dating and its Application to Archaeological Data. J. Archaeol. Method and Theory 5/2: 165-207

Buckman SS (1887-1907) Monograph of the ammonites of the Inferior Oolite Series. Paleontogr. Soc. London CCLXII: 456 p

Carre B (1979) Graphs and Networks. Clarendon Press, Oxford. 270 p

Carter ES (1993) Biochronology and paleontology of uppermost Triassic radiolarians, Queen Charlotte Islands, British Columbia, Canada. Mém. Géol. Lausanne 11: 1-175

Carter ES, Whalen P. Guex J (1998) Biochronology and Paleontology of Lower Jurassic (Hettangian and Sinemurian) radiolarians, Queen Charlotte Island, Canada. Geological Survey Canada Bull 496: 1-162

Deboo PB (1965) Biostratigraphic correlation of the type Shubuta Member of the Yazoo Clay and Red Bluff Clay with their equivalent in southwestern Alabama. Alabama Geol. Surv. Bull. 80: 1-84

Drobne K (1977) Alvéolines paléogènes de l'Istrie et de la Slovénie. Mém. Suisses Paléontol. 99: 1-175

Fulkerson DR and Gross OA (1965) Incidence matrices and interval graphs. Pac. J Math.15/3: 835-855

Gilmore PC and Hoffman AJ (1964) A characterization of comparability graphs and of interval graphs. Canad. J Math. 16: 539-548

Godinot M (1981) Usefulness and meaning of the mammalian specific lineages. Acta Geol. Hispanica 16/1: 249-258

Gorican S (1994) Jurassic and Cretaceous radiolarian biostratigraphy and sedimentary evolution of the Budva Zone (Dinarides, Montenegro). Mém. Géol. Lausanne 18: 1-176

Guex J (1967) Contribution à l'étude des blessures chez les ammonites. Bull. Géol. Lausanne, 165: 1-23

Guex J (1968) Sur deux conséquences particulières des traumatismes du manteau des ammonites. Bull. Géol. Lausanne, No 178: 1-8

Guex J (1970) Sur le sexe des ammonites. Bull. Géol. Lausanne, No 178: 1-6

Guex J (1977) Une nouvelle méthode d'analyse biochronologique. Bull. Géol. Lausanne, No 224: 309-322

Guex J (1979) Terminologie et méthodes de la biostratigraphie moderne. Bull. Géol. Lausanne, No 234: 169-216

Guex J (1987) Corrélations biochronologiques et associations unitaires. Presses Polytechniques Romandes. 250 p.

Guex J (1991) Biochrological Correlations. Springer Verlag 252 p.

Jud R (1994) Biochronology and Systematics of Early Cretaceous Radiolaria of the Western Tethys. Mém. Géol. Lausanne 19: 1-147

Lazarus D et al. (1995) Revised chronology of Neogene DSDP holes from the world ocean. O.D.P. Technical Note 24: 1-250

Lekkerkerker CB and Boland JC (1962) Representation of a finite graph by a set of intervals on the real line. Fund. Math. 51: 45-64

Matsuoka A (1995) Middle Jurassic to Early Cretaceous Radiolarian Occurrences in Japan and the Western Tethys. Mém. Géol. Lausanne 23: 937-966

Meinhardt H (1995) The Algorithmic Beauty of Sea Shells. Springer Verlag. 1-204

O'Dogherty L (1994) Biochronology and Paleontology of Mid-Cretaceous Radiolarians from Northern Apennines (Italy) and Betic Cordillera (Spain). Mém. Géol. Lausanne 21: 1-413

Remane J, Basset MG, Cowie JW, Gohrbandt KH, Lane HR, Michelsen O and Wang N (1996) Revised guidelines for the establishment of global chronostratigraphic standards by the International Commission on Stratigraphy (ICS). Episodes 19/3: 77-81

Savary J and Guex J (1991) BioGraph: un nouveau programme de construction des corrélations biochronologiques basées sur les associations unitaires. Bull. Géol. Lausanne. 313: 317-340

Thaler L (1972) Datation, zonation et mammifères. Mém. B.R.G.M. 77/1: 411-424

Westermann GEG (1966) Covariation and taxonomy of the Jurassic ammonite Sonninia adicra (Waagen). N. Jb. Geol. Pal. Abh. 124/3: 289-312

## *Acknowledgements*

The realization of this publication was made possible thanks to the Swiss National Research Foundation which is gratefully acknowledged, together with the 450$^{e}$ Anniversaire Foundation of Lausanne University, the Swiss Academy of Sciences and the Dr Joachim de Giacomi Foundation.

For various reasons we also wish to thank François Reuse, Luis O'Dogherty, Spela Gorican, Beth Carter, Patrick De Wever, Hamal Hubbard, Philip McGough, Grégoire Monin, Mike Cosca and Dave Taylor.

No. 29 DOBMEIER C. 1996. Die variskische Entwicklung des südwestlichen Aiguilles Rouges Massives (Westalpen, Frankreich). 191 pp. 70 text-figs., 18 tables., 1 map.

No. 30 BAUD A., POPOVA I., DICKINS J.M., LUCAS S. and ZAKHAROV Y. 1997. Late Paleozoic and early Mesozoic circum-Pacific events : biostratigraphy, tectonic and ore deposits of Primoryie (far East Russia). IGCP Project 272. 202 pp., 71 text-figs., 48 pls.

No. 31 ARMANDO G. 1999. Intracontinental alkaline magmatism : geology, petrography, mineralogy and geochemistry of the Jebel Hayim Massif (Central High Atlas, Morocco). 106 pp. 51 text-figs., 23 tab., 1 map.

No. 32 DEZES P. 1999. Tectonic and metamorphic evolution of the Centrtal Himalayan Domain in Southeast Zanskar (Kashimr, India). 145 pp., 89 text-figs., 1 map.

No. 33 AMODEO F. 1999. Il Triassico terminale- Giurassico del Bacino Lagonegrese. Studi stratigrafici sugli Scisti Silicei della Basilicata (Italia meridionale). 160 pp., 50 text-figs., 10 pl.

No. 34 SAVARY J. and GUEX J. 1999. Discrete biochronological scales and Unitary Associations: Description of the BioGraph computer program. 282 pp. 21 text-figs.

IMPRIMERIE CHABLOZ SA, 1004 LAUSANNE

IMPRIMÉ EN SUISSE

# Mémoires de Géologie (Lausanne)

No. 1* BAUD A. 1987. Stratigraphie et sédimentologie des calcaires de Saint-Triphon (Trias, Préalpes, Suisse et France). 202 pp., 53 text-figs., 29 pls.

No. 2 ESCHER A., MASSON H. and STECK A. 1988. Coupes géologiques des Alpes occidentales suisses. 11 pp., 1 text-figs., 1 map.

No. 3* STUTZ E. 1988. Géologie de la chaîne Nyimaling aux confins du Ladakh et du Rupshu (NW-Himalaya, Inde). Evolution paléogéographique et tectonique d'un segment de la marge nord-indienne. 149 pp., 42 text-figs., 11 pls. 1 map.

No. 4 COLOMBI A. 1989. Métamorphisme et géochimie des roches mafiques des Alpes ouest-centrales (géoprofil Viège-Domodossola-Locarno). 216 pp., 147 text-figs., 2 pls.

No. 5 STECK A., EPARD J.-L., ESCHER A., MARCHANT R., MASSON H. and SPRING L. 1989 Coupe tectonique horizontale des Alpes centrales. 8 pp., 1 map.

No. 6 SARTORI M. 1990. L'unité du Barrhorn (Zone pennique, Valais, Suisse). 140 pp., 56 text-figs., 3 pls.

No. 7 BUSSY F. 1990. Pétrogenèse des enclaves microgrenues associées aux granitoïdes calco-alcalins: exemple des massifs varisque du Mont-Blanc (Alpes occidentales) et miocène du Monte Capanne (Ile d'Elbe, Italie). 309 pp., 177 text-figs.

No. 8* EPARD J.-L. 1990. La nappe de Morcles au sud-ouest du Mont-Blanc. 165 pp., 59 text-figs.

No. 9 PILLOUD C. 1991. Structures de déformation alpines dans le synclinal de Permo-Carbonifère de Salvan-Dorénaz (massif des Aiguilles Rouges, Valais). 98 pp., 59 text-figs.

No. 10* BAUD A., THELIN P. and STAMPFLI G. 1991. (Eds.). Paleozoic geodynamic domains and their alpidic evolution in the Tethys. IGCP Project No. 276. Newsletter No. 2. 155 pp.

No. 11 CARTER E.S. 1993 Biochronology and Paleontology of uppermost Triassic (Rhaetian) radiolarians, Queen Charlotte Islands, British Columbia, Canada. 132 pp., 15 text-figs., 21 pls.

No. 12* GOUFFON Y. 1993. Géologie de la «nappe» du Grand St-Bernard entre la Doire Baltée et la frontière suisse (Vallée d'Aoste -Italie). 147 pp., 71 text-figs., 2 pls.

No. 13 HUNZIKER J.C., DESMONS J., and HURFORD AJ. 1992. Thirty-two years of geochronological work in the Central and Western Alps: a review on seven maps. 59 pp., 18 text-figs., 7 maps.

No. 14 SPRING L. 1993. Structures gondwaniennes et himalayennes dans la zone tibétaine du Haut Lahul-Zanskar oriental (Himalaya indien). 148 pp., 66 text-figs, 1 map.

No. 15 MARCHANT R. 1993. The Underground of the Western Alps. 137 pp., 104 text-figs.

No. 16 VANNAY J.-C. 1993. Géologie des chaînes du Haut-Himalaya et du Pir Panjal au Haut-Lahul (NW-Himalaya, Inde). Paléogéographie et tectonique. 148 pp., 44 text-figs., 6 pls.

No. 17* PILLEVUIT A. 1993. Les blocs exotiques du Sultanat d'Oman. Evolution paléogéographique d'une marge passive flexurale. 249 pp., 138 text-figs., 7 pls.

No. 18 GORICAN S. 1994. Jurassic and Cretaceous radiolarian biostratigraphy and sedimentary evolution of the Budva Zone (Dinarides, Montenegro). 120 pp., 20 text-figs., 28 pls.

No. 19 JUD R. 1994. Biochronology and systematics of Early Cretaceous Radiolaria of the Western Tethys. 147 pp., 29 text-figs., 24 pls.

No. 20 DI MARCO G. 1994. Les terrains accrétés du sud du Costa Rica. Evolution tectonostratigraphique de la marge occidentale de la plaque Caraïbe. 166 pp., 89 text-figs., 6 pls.

No. 21* O'DOGHERTY L. 1994. Biochronology and paleontology of Mid-Cretaceous radiolarians from Northern Apennines (Italy) and Betic Cordillera (Spain). 415 pp., 35 text-figs., 73 pls.

No. 22 GUEX J. and BAUD A. (Eds.). 1994. Recent Developments on Triassic Stratigraphy. 184 pp.

No. 23 BAUMGARTNER P.O., O'DOGHERTY L., GORICAN S., URQUHART E., PILLEVUIT A. and DE WEVER P. (Eds.). 1995. Middle Jurassic to Lower Cretaceous Radiolaria of Tethys: Occurrences, Systematics, Biochronolgy. 1162 p.

No. 24 REYMOND B. 1994. Three-dimensional sequence stratigraphy offshore Louisiana, Gulf of Mexico (West Cameron 3D seismic data). 215 pp., 169 text-figs., 49 pls.

No. 25 VENTURINI G. 1995. Geology, Geochronology and Geochemistry of the Inner Central Sezia Zone. (Western Alps - Italy). 183 pp. 57 text-figs, 12 pls.

No. 26 SEPTFONTAINE M., BERGER J.P., GEYER M., HEUMANN C., PERRET-GENTIL G. and SAVARY, J. 1995. Catalogue des types paléontologiques déposés au Musée Cantonal de Géologie, Lausanne. 76 pp.

No. 27 GUEX, J. 1995. Ammonites hettangiennes de la Gabbs Valley Range (Nevada, USA). 130 pp., 22 figs., 32 pl.

No. 28 HÜRLIMANN A., BESSON-HURLIMANN A and MASSON H. 1995. Stratigraphie et tectonique de la partie orientale de l'écaille de la Gummfluh (Domaine Briançonnais des Préalpes). 132 pp. 62 text-figs., 39 pl.., 6 maps.

*: out of print

(continued inside)

Order from **Institut de Géologie et Paléontologie,**
**Université de Lausanne. BFSH-2. CH-1015, Switzerland.**
http://www-sst.unil.ch/publications/memoires.htm Fax: (41) 21-692.43.05
Bank Transfer: Banque Cantonale Vaudoise 1002 Lausanne
Account Number: **C.323.52.56** Institut de Géologie, rubrique: Mémoires
Price CHF 30 per volume except volume 23 (CHF 100). The price doesn't include postage and handling.
- Please do not send check -