Institut National Polytechnique de Lorraine

ale Supérieure de Géologie

Centre de Recherche en Informatique de Nancy

DEFINITION ET IMPLANTATION D' UN LANGAGE DE MANIPULATION DE DONNEES

THESE

PRÉSENTÉE POUR L' OBTENTION DU DOCTORAT DE 3e CYCLE EN INFORMATIQUE

SOUTENUE PUBLIQUEMENT LE 12 MARS 1982
PAR

Pierre BOUCHET

DEVANT LE JURY

Président . . . . . J.P. HATON
Examinateurs . . . . . P.Y. CUNIN
. . . . . M. GRIFFITHS
. . . . . J.L. MALLET

Institut National Polytechnique de Lorraine

Ecole Nationale Supérieure de Géologie

Centre de Recherche en Informatique de Nancy

DEFINITION ET IMPLANTATION D' UN LANGAGE DE MANIPULATION DE DONNEES

THESE

PRÉSENTÉE POUR L' OBTENTION DU DOCTORAT DE 3e CYCLE EN INFORMATIQUE

SOUTENUE PUBLIQUEMENT LE 12 MARS 1982
PAR

Pierre BOUCHET

DEVANT LE JURY

Président . . . . . J.P. HATON
Examinateurs . . . . . P.Y. CUNIN
. . . . . M. GRIFFITHS
. . . . . J.L. MALLET

Je remercie MM CUNIN, GRIFFITHS, MALLET, ROYER pour leur soutien et leur aide, M. HATON pour avoir accepté de présider le jury, le service imprimerie du C.R.P.G. et aussi

pour leur complicité.

CHAPITRE I

INTRODUCTION

Introduction

Le champ de l'informatique dans les différents domaines scientifiques s'ètend de plus en plus, ce qui, jusqu'à présent, s'est traduit surtout par un développement du matériel et du logiciel,permettant la résolution des problèmes au coup par coup. C'était aux utilisateurs de s'adapter à l'informatique, c'est à dire, dans la plupart des cas, apprendre un langage tel que FORTRAN, le langage scientifique le plus répandu.

Le développement du logiciel a entrainé une prolifération de bibliothèques de statistiques ou de cartographie °MAL-76§, °LEB-77§. Le rôle de l'utilisateur était d'écrire les programmes principaux appelant ces bibliothèques, dont les paramétres n'étaient pas toujours bien définis. Cette technique présentait le défaut d'être un peu trop figée, car, pour permettre la résolution d'un problème précis, il fallait développer la bibliothèque par les moyens habituels, c'est à dire par la création de sous-programmes répondant aux besoins immédiats.

Aussi certains utilisateurs ont-ils ressenti la nécessité de développer des outils qui simplifient la résolution de leurs problèmes et qui apportent une certaine sécurité, non pas en proposant de nouveaux logiciels, mais en offrant à l'usager des moyens nouveaux, facilitant son travail.

C'est cette démarche qui a été à l'origine de la rencontre de deux équipes de recherche :

- une équipe informatique du Centre de Recherche en Informatique de Nancy (CRIN) : M^{r} GRIFFITHS, M^{r} CUNIN.
- une équipe analyse de données et géologie du Centre de Recherche

Pétrographique et Géochimique (CRPG) : M^r MALLET, M^r ROYER.

Ces deux équipes ont défini les besoins propres aux géologues en approchant le problème d'une manière nouvelle.

Tout le travail des géologues et des statisticiens est centré autour de tableaux de données : un tableau de données est un tableau dont les colonnes représentent un ensemble de mesures ou de valeurs, et dont les lignes correspondent aux points où ont été faites ces mesures.

En dehors des valeurs qu'il contient, un tableau de données représente un ensemble de mesures réparties sur des échantillonnages différents, par exemple :

- une population non régulièrement répartie.
- un forage.
- une carte géologique.

Très souvent, il est essentiel de connaître la position de chaque ensemble de mesures dans un système de coordonnées classiques. Les coordonnées des points peuvent être implicites ou explicites. Les coordonnées implicites correspondent à un espace dans lequel les mesures sont prises de façon régulière, c'est à dire, sur une grille à une, deux ou trois dimensions. Dans ce cas nous parlerons de fonctions r gulières . Les coordonnées explicites sont utilisées dans le cas d'un échantillonnage quelconque sur le terrain. Nous parlerons ici de fonctions irrégulières .

A partir de cette constatation, il nous a paru intéressant de définir un langage permettant :

- une description simple et précise du tableau de données.
- une manipulation aisée du tableau de données, tant du point de vue

du calcul que du point de vue de l'édition.

J'ai rejoint ces équipes l'année de mon D.E.A., alors qu'elles en étaient à l'étape de définition du langage. Ce langage a été défini dans une première phase en essayant de réaliser, pour un cas particulier, la g'ologie, un langage répondant aux différentes normes de fiabilité et de sécurité de programmation, tout en restant un langage algorithmique °CUN-79§.

Aprés l'écriture du compilateur correspondant à la première définition du langage, il s'est avéré intéressant d'englober ce compilateur dans un système conversationnel, facilitant le travail de gestion des différents fichiers de données ou des bibliothèques manipulées par les programmes. Ce système permet à des non-informaticiens d'utiliser les possibilités d'un ordinateur.

Dans une seconde étape, nous avons voulu étendre le champ d'action de ce langage, en étudiant la définition et l'utilisation de "packages génériques", tels que les définit le langage ADA °ADA-81§, non plus dans un contexte purement géologique mais dans le contexte plus vaste de l'analyse de données.

La philosophie de cette thèse consiste en l'étude de l'apport des recherches académiques en informatique appliquées à un cas concret.

CHAPITRE II

LE LANGAGE GEOL

Le langage GEOL

II.1. Introduction

Le langage GEOL °BOU-80$ est un langage conçu de façon à répondre aux ob ectifs suivants :

- permettre une description et une structuration simple et précise de tableaux de données.
- simplifier le stockage et la manipulation des informations contenues dans les tableaux de données.
- faciliter l'utilisation des différents moyens d'édition (bibliothèques ou machines) selon les besoins de l'analyse de données.
- assurer une sécurité lors de l'exécution des programmes :
  * en augmentant les contrôles de sémantique statique, l'implémentation des divers types d'objets manipulés permet une connaissance accrue de leurs caractéristiques.
  * en contrôlant la compatibilité entre les paramètres effectifs et les paramètres formels, y compris dans le cas de sous-programmes de bibliothèques.
- assurer une portabilité des programmes.
- assurer une plus grande lisibilité des programmes.
- permettre une réutilisation aisée des bibliothèques déjà existantes. Cette contrainte entraîne l'acceptation dans un programme écrit en GEOL d'instructions FORTRAN, car les bibliothèques existantes sont généralement écrites en FORTRAN.

Pour les régles syntaxiques qui suivent nous utilisons les conventions d'écriture suivantes :

- Les symboles terminaux sont :
  - + soit des symboles du langage.
  - + soit des mots du langage soulignés.
  - + soit des noms de catégories (idf, cste).
- Les symboles { et } servent à indiquer le choix obligatoire d'une alternative.
- Les répétitions d'alternatives sont notées de la manière suivante :
  - + ° - - § séquence absente ou présente une fois.
  - + ° - - §* séquence absente ou présente plusieurs fois.
  - + ° - - §+ séquence présente une ou plusieurs fois.
- Pour éviter toutes ambiguités les symboles ° et § utilisés en tant que symboles du langage apparaitront soulignés (° et §).
- Le passage à la ligne indique une nouvelle alternative.

II.2. Généralités sur le langage GEOL.

Le tableau de données, autour duquel GEOL a été conçu, demeure une abstraction pour l'utilisateur, car son implantation lui est cachée, et sa manipulation n'est autorisée que par l'intermédiaire d'opérations définies dans le langage.

Le langage GEOL est un langage algorithmique permettant une programmation structurée.

La structure générale d'un programme GEOL est décrite par la syntaxe suivante :

PROGRAMME ::= ° SOUS_PROGRAM §+ ° PROGRAM_PRINCIPAL § ° SOUS_PROGRAM §* ENDGEOL
° SOUS_PROGRAM §* PROGRAM_PRINCIPAL ° SOUS_PROGRAM §* ENDGEOL

PROGRAM_PRINCIPAL ::= PROGRAM ° PARAM_SYS § ; CORPS ENDPROG ;

SOUS_PROGRAM ::= PROCEDURE idf (PARAM ° , PARAM §*) ; CORPS ENDPROC ;
FONCTION TYPE idf (PARAM ° , PARAM §*) ; CORPS ENDFONC ;

CORPS ::= DECLARATION ; °DECLARATION ;§* INSTRUCTION ; ° INSTRUCTION ; §*

Pour des raisons de lisibilité et, à un degré moindre, d'implémentation, nous avons voulu une structure simple de programmes. Il n'y a donc pas de notion de blocs et pas de procédures imbriquées.

Dans ce chapitre, nous présentons les points du langage qui nous sont apparus comme les plus caractéristiques; nous y détaillons aussi la déclaration d'un tableau de données et de ses composantes, mais nous ne parlerons pas des déclarations des variables de travail. Seules apparaîtront les instructions spécifiques du langage.

On trouvera en Annexe I la grammaire complète du langage.

II.3. Objets manipulés par GEOL.

II.3.A. Les tableaux de données

Définition

En géologie, les points de sondage où sont effectuées les mesures sont répartis sur le terrain soit de manière régulière, ce qui donne les réseaux réguliers , soit de manière irrégulière, ce qui donne les réseaux irréguliers .

Soit par exemple un terrain à deux dimensions, figure 1 .
A chaque couple (x_i , y_j) correspond un point où certaines mesures ont été faites. Ces points seront par la suite appelés unités statistiques .
On appelle x_{min} l'abscisse minimale, x_{max} l'abscisse maximale et p_x le pas sur l'axe des abscisses, c'est à dire l'intervalle entre deux abscisses

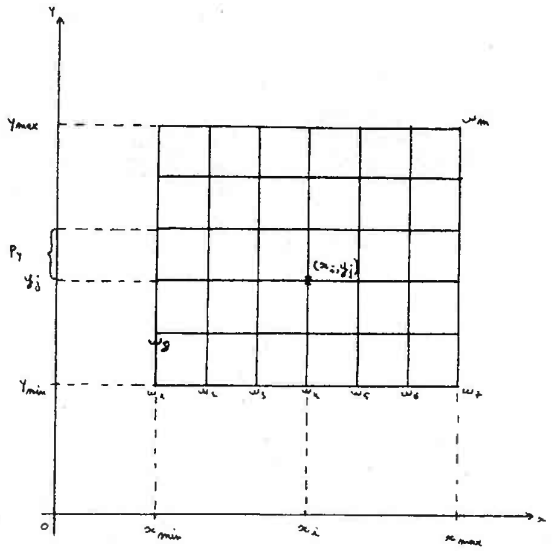

figure 1

consécutives.

On aura la même notation y_{min} , y_{max} , p_y sur l'axe des ordonnées, dans le cas d'un tableau à deux dimensions.

La représentation que nous avons choisie pour visualiser un tableau de données, et qui convient quelque soit la dimension de ce tableau de données, est représentée à la figure 2 .

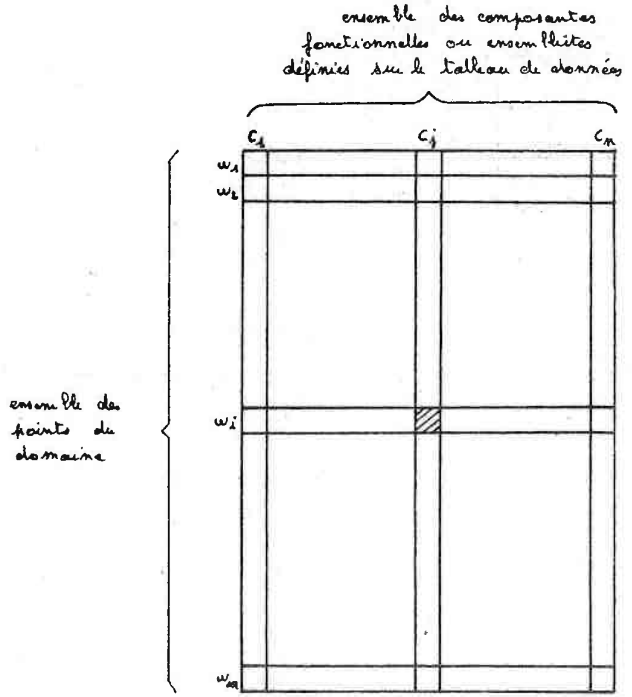

figure 2

Chaque ligne w_l de la figure 2 représente un point de l' espace, c'est à dire une unité stastistique, et correspond à un couple (x_i , y_j) de la

figure 1.

Chaque colonne c_k de la figure 2 représente l'ensemble d'une même mesure en tous points du domaine. Nous attribuerons par la suite, à toute colonne de ce tableau, le nom de composante du tableau de données.

Les unités statistiques ne sont qu'une numérotation ordonnée des points du domaine de définition. Pour un tableau régulier à deux dimensions, connaissant x_{min} , x_{max} , p_x , y_{min} , y_{max} et p_y il est possible à partir d'une unité statistique w_l de connaître le couple (x_i , y_j) correspondant ou d'effectuer le calcul inverse. On note n_x et n_y le nombre de points du domaine sur l'axe des x et l'axe des y. Ces différentes valeurs (x_{min} , x_{max} , n_x ,..) permettent de connaître le domaine de définition du tableau de données. Ceci est possible quelque soit le type du tableau de données.

II.3.B. Déclaration d'un tableau de données.

En GEOL, certaines informations, qui nous ont paru importantes, sont associées à tout tableau de données ; elles correspondent à la définition d'un tableau de données pour les géologues. Il s'agit des points suivants :

- la structure du tableau de données. Est-il régulier ou non?
- le nombre maximal de composantes associées à ce tableau.
- la caractéristique du tableau. Pour un tableau irrégulier, cette caractéristique est le nombre d'unités statistiques, pour un tableau régulier c'est la description du domaine de définition.

Pour la déclaration du nombre de composantes un choix a du être fait entre deux solutions :

- soit considérer que le tableau de données est réellement dynamique, c'est à dire que la place occupée sur disque est fonction de son

nombre de composantes.

- soit considérer que le tableau de données est statique, que la place qu'il occupe sur disque est fixe, et donc imposer une borne supérieure à son nombre de composantes.

Nous avons choisi la deuxième solution, car la première imposait une gestion complexe de l'espace disque. En fait, une fois le tableau créé, l'utilisateur a l'impression d'utiliser un tableau dynamique, car il ne déclare que les composantes qu'il utilise dans le programme, bien que le tableau tout entier soit chargé en mémoire de travail.

Nous considérons que les informations précédentes font partie du tableau de données, et qu'elles sont associées à son nom. Contrairement aux autres langages de programmation, où la déclaration d'une variable correspond seulement à une réservation de place mémoire, ici la déclaration d'un tableau de données correspond à un traitement spécial, qui est développé dans le chapitre III. Il faut, lors de la déclaration, préciser quel sera l'emploi de ce tableau.

Ceci est réalisé par la déclaration suivante :

{ NOUVEAU / LOCAL } TYPETAB idf NCOMP (cste) , OMEGA (CARACTERISTIQUE) ;

{ FORMEL / ANCIEN } TYPETAB idf ;

TYPETAB ::= { DT / DT1 / DT2 / DT3 }

La première option précise l'utilisation du tableau de données :

- NOUVEAU : on crée un tableau de données que l'on sauvegarde à la fin de l'exécution du programme.
- ANCIEN : on utilise un tableau de données déjà existant.
- LOCAL : on crée un tableau de données dont la durée de vie ne dépasse pas celle du programme.
- FORMEL : cette option est la seule utilisée dans un sous-programme et elle indique que la variable idf est un nom de tableau.

La seconde option précise la structure du tableau de données :

- DT : cas d'un tableau irrégulier.
- DT1 : cas d'un tableau régulier à une dimension (par exemple un sondage).
- DT2 : cas d'un tableau régulier à deux dimensions (par exemple un relevé topographique).
- DT3 : cas d'un tableau régulier à trois dimensions (par exemple un relevé dans une mine).

NCOMP indique le nombre maximal de composantes du tableau de données.

OMEGA indique la caractéristique du tableau de données, c'est à dire la nature de la distribution des unités statistiques dans l'espace.

Ces deux dernières options sont inutiles dans le cas d'un tableau de données déclaré ANCIEN ou FORMEL, car il est possible de les retrouver.

Dans la déclaration on ne parle pas du type du tableau, car celui-ci peut contenir des composantes de types différents ; le type apparaît donc dans la déclaration de ses composantes.

Voici un exemple de déclaration de tableau :

```
NOUVEAU DT2 JARNY NCOMP (10), OMEGA (1,1000,1000;1,100,200);
```

Cette déclaration crée un tableau de données de nom JARNY et réserve de la place pour celui-ci. De plus cette déclaration nous indique que :

- ce tableau de données aura au maximum 10 composantes (NCOMP).
- il correspond à une grille à deux dimensions(DT2).
- sur la première dimension x_{min} =1
  x_{max} =1000
  n_x =1000
- sur la deuxième dimension y_{min} =1
  y_{max} =100
  n_y =200

donc nous savons que ce tableau de données a 200000 unités statistiques et que p_x =1 et p_y =0.5

II.3.C. Déclaration d'une composante.

Une fois le tableau de données déclaré, nous pouvons déclarer les composantes qui lui sont associées. Une composante est caractérisée par :

- le nom du tableau auquel elle appartient.
- son type.
- son existence et sa durée de vie.

Sa déclaration sera la suivante :

{ NOUVEAU | ANCIEN | LOCAL | FORMEL } COMP DE idf { COMPOSANTE | (COMPOSANTE ° , COMPOSANTE §*) } ;

COMPOSANTE ::= { ENTIER | REEL | ALPHA | BOOL | ENSEMBLE } { IDF_COMP | (IDF_COMP ° , IDF_COMP §*) }

IDF_COMP ::= idf ° ° cste § §

Exemple :

```
NOUVEAU COMP DE JARNY ENTIER AL203 ° 10 §;
ANCIEN COMP DE JARNY REEL S102;
```

Les combinaisons possibles entre l'utilisation d'un tableau de données et l'existence de ses composantes sont représentées par le tableau de la figure 3.

Tableau / Composante	FORMEL	LOCAL	NOUVEAU	ANCIEN
FORMEL	OUI			
LOCAL		OUI	OUI	OUI
NOUVEAU			OUI	OUI
ANCIEN				OUI

figure 3

Les composantes sont associées à un nom de tableau, ce qui permet d'avoir le même nom de composantes pour des tableaux différents.

Ainsi peut-on déclarer :

ANCIEN COMP DE JARNY REEL SIO2 ;

ANCIEN COMP DE DROITAUMONT REEL SIO2

Le même nom de variable, ici SIO2, apparaît dans deux tableaux de données, à savoir la mine de JARNY et la mine de DROITAUMONT.

Ces variables peuvent être scalaires (idf), dans le cas où le nom repère une seule composante du tableau de données, ou vectorielles (idf ° cste §) si l'on veut que le nom repère un ensemble de composantes.

Ces variables peuvent être d'un type primitif du langage (entier, réel, alpha, booléen), et seront appelées composantes fonctionnelles, ou d'un type spécifique, que nous appellerons composantes ensemblistes. Ces dernières permettent de définir un sous-ensemble d'unités statistiques à partir de l'ensemble de départ. Par exemple (figure 4), pour un tableau à deux dimensions, l'ensemble de départ étant formé des unités statistiques w_1 à w_{16} , nous pouvons définir une composante ensembliste réunissant les unités statistiques w_1 , w_7 , w_{10} , w_{11} , w_{16} , ce qui correspond à la zone hachurée de la figure 4.

Nous verrons au paragraphe II.4 comment sont désignées les composantes dans le tableau de données. Ce repérage, qui se fait par l'intermédiaire du nom propre de la composante, dispense l'utilisateur d'une part de nommer toutes les composantes du tableau de données (il ne précise que celles qui sont utilisées dans la suite du programme), et d'autre part de se souvenir de l'ordre de déclaration de ces composantes.

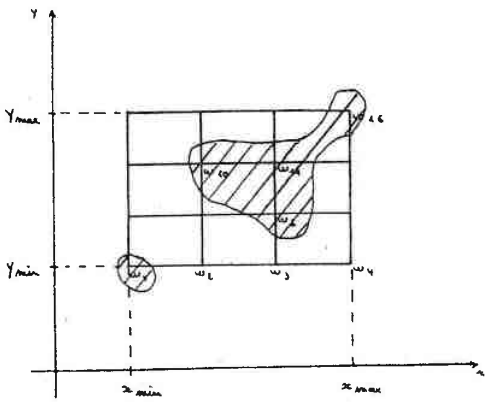

figure 4

Remarque

Pour permettre une réutilisation de programme écrits en GEOL et déjà compilés, nous avons permis une paramétrisation au niveau des déclarations. Les différentes paramétrisations possibles sont les suivantes :

- au niveau du nom d'un tableau de données et de ses caractéristiques, on peut écrire :

NOUVEAU DTl TOTO? NCOMP (?) OMEGA (?);

A l'exécution un dialogue s'établit et c'est à ce moment seulement que l'utilisateur précisera :

+ le nom du tableau de données.
+ le nombre de ses composantes.
+ sa caractéristique.

Signalons que dans tout le programme, le tableau a pour nom TOTO.

- au niveau de la déclaration d'une composante, on peut écrire :

NOUVEAU COMP DE TAB REEL (VAR?) ;

Lors de l'exécution le même procédé se déroulera, l'utilisateur précisant sur quelle composante il veut réellement travailler. Pour tout le reste du programme le nom VAR aura été utilisé.

Ces paramétrisations sont possibles dans tous les cas de déclarations, excepté le cas d'une déclaration FORMEL, car elles sont inutiles.

II.3.D. Réservation de mémoire.

Les réservations sont faites lors de l'exécution du programme objet généré.

+ L'option NOUVEAU correspond à une réservation permanente et doit être utilisé si l'on désire la rémanence des informations associées.
+ L'option LOCAL correspond à une réservation temporaire (locale) c'est à dire qu'il n'y aura pas de sauvegarde des informations à la fin de l'exécution.
+ L'option ANCIEN est utilisée quand on reprend des informations

sauvegardées lors d'un travail antérieur, et que l'on veut les sauvegarder à la fin de l'exécution.

+ L'option FORMEL n'est utilisée qu'à l'intérieur des sous-programmes de type procédure ou fonction, et ne provoque aucune réservation.

Remarque.

Si l'on désire supprimer un tableau de données ou une composante d'un tableau, on peut le faire à l'aide des instructions suivantes :

- pour un tableau

  DETRUIRE idf;

- pour des composantes

  DETRUIRE idf °, idf §* DE idf;

Dans le cas d'un tableau de données ceci libère l'espace mémoire tant physique (espace disque) que de travail (mémoire centrale).

Dans le cas d'une composante l'espace physique n'est pas libéré, mais il est possible d'utiliser cette place pour stocker une nouvelle composante.

Ces instructions ne doivent apparaître qu'au niveau des déclarations et exclusivement dans les programmes principaux. La destruction d'un tableau ou d'une composante déclarés FORMEL est impossible.

II.4. Opérations possibles en GEOL

II.4.A. Désignation d'une composante.

Nous avons vu la déclaration d'une composante. Si maintenant nous voulons la désigner nous écrivons :

{ idf / idf ° EXP § } DE idf

Par exemple :

SIO2 DE JARNY

AL203 ° 5 § DE JARNY

Cette construction permet de désigner une composante scalaire ou vectorielle globalement, c'est à dire pour toutes les unités statistiques. Mais si l'on veut désigner la composante pour une seule unité statistique, on le fera de la manière suivante :

{ idf / idf ° EXP § } (idf) DE idf

Par exemple :

SIO2 (I) DE JARNY

AL203 ° 5 § (I) DE JARNY

Le premier exemple désigne pour la composante SIO2 la $i^{éme}$ unité statistique. Cette notation est un peu lourde, et il serait préférable de rester plus proche de la notation mathématique, c'est à dire d'écrire :

SIO2 (I)

Ceci est possible si la variable I désigne non seulement l'unité statistique, mais aussi le tableau auquel appartient la composante.

Pour permettre cela on a recours à une déclaration spécifique :

STATUS idf DE idf;

Par exemple :

STATUS I DE JARNY ;

A partir de cette déclaration et pour la suite du programme la variable I se trouve liée de manière définitive au tableau JARNY. Elle servira non seulement à désigner l'unité statistique voulue, mais aussi le tableau sur lequel on veut travailler. Nous avons exclu la possibilité d'avoir plus d'un STATUS pour un même tableau, afin d'éviter l'accès à un moment donné à des unités statistiques trop éloignées les unes des autres.

Si XY et I sont deux variables déclarées STATUS pour des tableaux différents, on pourra trouver dans le programme :

SIO2 (XY) = SIO2 (I);

La présence des variables déclarées STATUS enlève toute ambiguïté.

II.4.B. Initialisation d'une composante.

On peut initialiser une composante scalaire ou vectorielle en lui affectant une même valeur pour toute les unités statistiques. Ceci se fait au moyen de l'instruction suivante :

TOUT idf DE idf = cste;

Par exemple :

TOUT SIO2 DE JARNY = 3.15;

Les composantes sur lesquelles on peut faire cette affectation sont d'un type de base du langage (ENTIER, REEL, ALPHA, BOOL), mais non pas du type SET. Une composante ensembliste servant à repérer les unités statistiques ayant une même propriété, son initialisation à une valeur n'a aucun sens, si ce n'est vouloir désigner l'ensemble du tableau ou l'ensemble vide. Pour cela, nous avons deux mots clés OMEGA (ensemble complet) et NULL (ensemble vide).

II.4.C. Parcours d'un tableau de données.

Le principe d'itération sur les unités statistiques demeure proche de celui des autres langages, mais apporte quelques possibilités supplémentaires.

Le corps de la boucle est entouré par les deux mots clés FAIRE , FAIT. Les critères de sélection sont les suivants :

* Parcours de tout le tableau de données.

POUR TOUT idf FAIRE - - - FAIT;

La variable idf est toujours une variable STATUS ce qui permet de définir le tableau sur lequel on travaille.

* Parcours du tableau de données pour certaines unités statistiques.

POUR idf = cstel A cste2 FAIRE - - - FAIT;

Il y a parcours du tableau de données repéré par le STATUS idf pour les unités statistiques appartenant à l'intervallle cstel, cste2.

POUR idf = cstel ° , cste2 §* FAIRE - - - FAIT;

On parcourt le tableau de données pour les unités statistiques prenant les différentes valeurs énumérées cstel, cste2. . .

POUR idf1 DANS idf2 FAIRE - - - FAIT;

Le tableau de données est parcouru pour les unités statistiques appartenant à la composante ensembliste idf2.

II.4.D. Définition d'une composante ensembliste

Une composante ensembliste sert à réunir sous un même nom un ensemble d'unités statistiques, pour lesquelles une ou plusieurs propriétés sont vérifiées. Sa définition s'écrit sous la forme suivante :

DEFENS idf CONTROLE_BOUCLE ° TELQUE (EXP) § ;

Nous appelons CONTROLE_BOUCLE ce qui permet de définir et de parcourir le tableau, et que nous avons vu au paragraphe précédent.

Le tableau est parcouru en fonction du critère CONTROLE_BOUCLE, et un chaînage est établi, reliant entre elles les unités statistiques pour lesquelles l'expression booléenne, qui suit le mot clé TELQUE, est vraie.

Exemple d'utilisation :

```
DEFENS ENS POUR TOUT I TELQUE (SIO2(I) < 0.15 ET AL2O3(I) > 0.45) ;
```

Ainsi est définie une composante ensembliste ENS, qui réunira toutes les unités statistiques du tableau de données précisé par la variable STATUS I, pour lesquelles la variable SIO2 est inférieure à 0.15 et la variable AL203 est supérieure à 0.45.

Quand plusieurs composantes ensemblistes ont été définies de cette manière, il est possible de définir de nouvelles composantes, en utilisant les opérations habituellement définies sur les ensembles :

- ET : intersection.
- OU : union.
- COMP : complémentarité.

Par exemple :

IDSET1 DE idf=(IDSET2 ET IDSET3) OU IDSET4;

Dans ce cas la composante IDSET1 repère toutes les unités statistiques qui appartiennent :

- soit à l'ensemble IDSET4.
- soit à l'ensemble défini comme l'intersection des ensembles IDSET2 et IDSET3.

Lors de l'utilisation de cette instruction, il faut vérifier que toutes les composantes sont définies sur le même tableau idf.

Il existe dans le langage GEOL deux mots clés repérant deux ensembles prédéfinis :

- NULL : ensemble vide.
- OMEGA : ensemble total, c'est à dire l'ensemble désignant toutes les unités statistiques du tableau auquel on s'intéresse.

II.4.E. Définition d'une liste.

Nous venons de voir que les composantes ensemblistes servent à regrouper sous un même nom un certain nombre de lignes d'un tableau de données , c'est à dire un ensemble d'unités statistiques vérifiant une propriété. Pour des raisons de facilité, on peut vouloir regrouper sous un même nom un certain nombre de colonnes d'un tableau de données, c'est à dire une liste de composantes.

Nous avons offert cette possibilité par l'intermédiaire de la déclaration suivante :

DEFLIST TYPE idf DE idf = (idfB ° , idfB §);

idfB ::= idf ° ° { cste | cstel : cste2 } § §

Exemple :

DEFLIST REEL TENEUR DE JARNY = (SIO2 , AL203 ° 3 : 5 §);

Cette déclaration a deux effets :

- elle déclare pour le reste du programme une variable de type liste de nom TENEUR.
- elle précise les composantes qui seront repérées par la variable liste TENEUR.

La notation AL203 ° 3 : 5 § permet d'affecter à la liste TENEUR une partie seulement de la composante vectorielle AL203, et l'on affecte alors à TENEUR les composantes AL203 comprises entre 3 et 5. La variable liste TENEUR, une fois définie s'utilise comme une composante vectorielle, par

exemple :

TENEUR ° 3 § DE JARNY

qui est équivalent à :

AL203 ° 4 § DE JARNY

II.4.F. Déclaration et appel de sous-programmes.

Dans le langage GEOL les procédures sont des sous-programmes secondaires, c'est à dire qu'elles sont compilées séparément ; de plus elles ne peuvent être ni imbriquées, ni déclarées dans un programme principal; elles n'ont donc pas accès à des variables globales.

Le passage des paramètres se fait par adresse, comme dans certaines implémentations du langage FORTRAN.

En GEOL, il existe des sous-programmes de deux types :

- procédure.
- fonction.

La déclaration d'une procédure respecte la syntaxe suivante :

PROCEDURE idf (PARAM ° , PARAM §);

PARAM ::= { idf | idf DE idf }

exemple :

PROCEDURE MOYENNE (X DE Y, MOY);

La déclaration d'une fonction est de la forme :

FONCTION TYPE idf (PARAM ° , PARAM §);

exemple :

FONCTION REEL MOYENNE(X DE Y);

On ne précise pas à ce niveau la taille de la composante, elle peut être scalaire ou vectorielle, ceci est indiqué au niveau de la déclaration de la composante dans le corps de la procédure.

Pour lever toute ambiguïté, il est nécessaire de préciser dans les paramètres formels si ceux-ci sont des composantes ou des variables, car il est possible de rencontrer, dans un sous-programme, la variable SI02 et la composante SI02 d'un tableau de données, d'où la forme syntaxique particulière de PARAM.

Une des particularités importante du langage GEOL est la possibilité de définir une relation entre des paramètres °CUN-79§. Soit la déclaration suivante :

PROCEDURE P (X DE Y , A DE B , A DE Y);

Cette déclaration impose que le 1^{er} et le $3^{ème}$ paramètres soient définis sur le même tableau de données.

L'appel des procédures se fait de la manière suivante :

CALL idf (PARAM_EFFECTIF ° , PARAM_EFFECTIF §);

PARAM_EFFECTIF ::= { EXP | idf ° ° EXP § § DE idf }

Un contrôle de type entre les paramètres effectifs et formels est assuré par le compilateur. Dans le chapître III sont envisagés les problèmes soulevés par la traduction des déclarations et des appels de sous-programme.

II.4.G. Entrées-Sorties.

Les entrées-sorties en GEOL s'effectuent la plupart du temps à l'aide de sous-programmes d'édition, qui permettent aussi bien des éditions de matrices que des dessins de cartes ou de blocs diagrammes.

C'est pour cette raison que la syntaxe des ordres LIRE et ECRIRE a été définie de façon à être la plus simple possible pour l'utilisateur, et non dans le but de permettre des éditions trés élaborées.

C'est ainsi qu'il y a des formats implicites d'édition, associés à chaque type de base, formats que l'on peut redéfinir à tout moment trés simplement à l'aide de l'ordre STYLEDIT.

Par exemple :

```
STYLEDIT (ENTIER(5))
```

Par cette instruction, il est précisé que les entiers qui seront édités par la suite auront cinq chiffres.

Un ordre d'écriture prendra par exemple la forme suivante :

```
ECRIRE <TAUX DE SILICE >, SIO2(XY), 1 LIGNE,
       <TAUX D'OXYDE D'ALUMINIUM>, AL2O3(XY) ;
```

XY est une variable déclarée STATUS, qui désigne une unité statistique. Connaissant le type des variables SIO2, AL2O3, on sait donc quel est le format d'édition associé; et le mot clé LIGNE précise que l'on veut laisser une ligne entre l'édition de la valeur de SIO2 et celle d'AL2O3, pour l'unité statistique correspondant à XY. Les symboles < et > servent à délimiter la chaine de caractères que l'on veut éditer.

Remarque.

S'ajoutant aux fonctions incorporées standard que l'on retrouve dans tous les langages scientifiques (SIN, COS, LOG...), des fonctions spécifiques (à caractère mathématiques, analyse de données, ...) sont proposées dans le langage GEOL. On trouvera en annexe II une liste de ces fonctions et de leurs utilisations.

II.5. Conclusion

L'utilisation du langage GEOL permet de constater que l'écriture de programmes est plus concise, car elle demeure trés proche de la notation mathématique à laquelle sont habitués les géologues, et plus sûre, en raison des contrôles sémantiques réalisés par le compilateur.

Nous avons voulu montrer que l'on pouvait appliquer dans un cas particulier les développements récents concernant la théorie des langages de programmation, ainsi le concept d'abstraction a été concrétisé par l'intermédiaire des tableaux de données, et on voit l'apport de tels concepts, qui se caractérise par une plus grande facilité d'écriture de programmes.

CHAPITRE III

COMPILATEUR DU LANGAGE GEOL

Compilateur du langage GEOL

III.1. Introduction

Au départ du projet GEOL, un débat entre concepteurs et utilisateurs s'est avéré nécessaire pour décider si le langage devait être interprété ou compilé. A l'issue de ce débat, le choix s'est porté sur un compilateur, essentiellement pour les raisons suivantes :

- rapidité lors de l'exécution. Les programmes écrits en Géologie sont souvent réutilisés, il est donc plus intéressant de les conserver compilés.
- sécurité accrue lors de l'exécution des programmes. On verra par exemple que les tableaux de données ne sont modifiés sur le support physique que s'il n'y a eu aucune erreur lors de l'exécution.

Le compilateur du langage GEOL est écrit en langage FORTRAN et génère du FORTRAN, ceci pour des raisons de portabilité qui seront développées au chapitre V.

Il s'articule autour d'une table d'analyse obtenue à partir de la grammaire du langage au moyen d'un transformateur °CUN-78§. Grâce à cette table que le compilateur parcourt, la syntaxe du texte source écrit en GEOL est vérifiée, et les fonctions sémantiques opérant les différents contrôles et génèrant le texte objet écrit en FORTRAN sont activées.

Le compilateur n'autorise l'exécution de ce texte objet que si aucune erreur n'a été détectée.

Pour permettre la réutilisation de programmathèques déjà existantes, nous autorisons à l'intérieur de programmes écrits en GEOL, l'existence d'instructions FORTRAN. Le compilateur effectue alors une recopie sans contrôle de cette partie, entourée par les deux mots clés DEBFOR et

FINFOR.

Plutot que de décrire en détail le compilateur réalisé, nous allons dans la suite de ce chapitre évoquer les solutions retenues et approfondir seulement les points présentant des aspects non classiques.

III.2. Stratégie adoptée pour la traduction des programmes GEOL.

Pour l'utilisateur, le tableau de données demeure une abstraction, son implémentation lui étant cachée. Nous allons exposer ici la démarche que nous avons suivie dans le choix de cette implémentation.

Un programme GEOL est écrit dans le but de manipuler un tableau de données. Cette manipulation peut être réalisée de différentes manières. Si l'on considère un tableau de données déjà existant sur lequel l'utilisateur veut travailler, une première méthode consiste à disposer pour chaque tableau de données d'un fichier, dont chaque enregistrement correspond à une unité statistique. Chaque accès à une unité statistique entraîne une recopie sur le support physique de la dernière unité statistique manipulée, puis un chargement en mémoire centrale des valeurs correspondant à l'unité statistique voulue. Cette solution présente les inconvénients suivants :

- le temps lors de l'exécution est long.
- en cas d'erreur, certaines unités auront subi un traitement alors que d'autres n'auront pas été modifiées.

Une seconde méthode, celle que nous avons privilégiée, consiste à recopier le tableau de données en mémoire centrale lors de sa déclaration. Les manipulations se font sur la copie, et le tableau de données est sauvegardé sur support physique à la fin de l'exécution du programme GEOL. Cette solution est plus sûre, car en cas d'erreur le fichier physique n'est

pas modifié, la recopie sur support externe n'étant pas effectuée. Dans cette solution, tout tableau de données correspond à un enregistrement physique.

Nous avons vu dans le chapitre II qu'au moment de sa création, un tableau est d'une taille fixée par sa déclaration. Le compilateur génèrant un programme FORTRAN dans lequel la taille des tableaux doit apparaître, une première solution consiste à les surdimensionner, mais ceci n'est pas trés performant vis à vis de la gestion de la mémoire. Nous avons préféré faire générer par le compilateur la déclaration d'un tableau unique, dans lequel seront rangés consécutivement les différents tableaux de données. Ce tableau est dimensionné à la taille précisé par l'utilisateur au niveau de l'option PARAM_SYS de la manière suivante :

PARAM (MEM = CSTE) ;

La valeur CSTE indique la taille du tableau de rangement que l'utilisateur juge suffisante pour y ranger les tableaux de données qu'il déclare ensuite. Il est évident qu'à l'exécution on vérifie que cette taille convient.

On ne fera appel à cette démarche que pour la traduction des programmes principaux GEOL, car à l'intérieur des sous-programmes GEOL il n'existe que des tableaux FORMEL.

Cette solution simule une allocation dynamique de mémoire, mais impose une structure spéciale de programmes au niveau du texte FORTRAN généré. Un programme principal GEOL est traduit en un programme principal FORTRAN et en un sous-programme FORTRAN, que nous appelons sous-programme principal.

Le programme principal FORTRAN a la structure suivante :

- la déclaration du tableau de rangement.
- les appels des sous-programmes chargeant les tableaux de données

dans le tableau de rangement.

- les appels des sous-programmes initialisant les pointeurs d'accés aux différentes composantes déclarées.
- l'appel du sous-programme principal FORTRAN qui correspond aux instructions du programme principal GEOL.
- la sauvegarde physique des tableaux de données.

Le sous-programme principal a la structure suivante :

- les déclarations des différents tableaux de données.
- les déclarations des variables permettant l'accès aux composantes déclarées.
- la traduction en FORTRAN des instructions du programme GEOL.

Remarque.

Cette structure présente un avantage supplémentaire. La sauvegarde des tableaux de données se fait par l'intermédiaire d'un sous-programme dont l'appel est généré lors de la rencontre de l'instruction GEOL : STOP. Si au cours de l'exécution le programme s'arréte à la suite d'une erreur de programmation, les tableaux de données n'auront pas été modifiés. Ceci permet de relancer les programmes aprés corrections, sans s'occuper des points de reprise.

III.3. Problèmes rencontrés lors de la déclaration d'un tableau de données ou d'une composante.

III.3.A. Structure d'un tableau de données pour le compilateur.

Nous avons vu au chapitre II qu'un descriptif est associé à chaque tableau de données et à chacune de ses composantes.

La structure d'un tableau de données est de la forme donnée par la figure 5, c'est à dire :

* le descriptif du tableau (32 mots) qui comprend :
  - le type + régulier à 1, 2 ou 3 dimensions.
    + irrégulier, et dans ce cas on ne parle pas de dimensions.
  - le nombre maximum de composantes.
  - le nombre d'unités statistiques de l'ensemble de départ.
  - si le tableau est régulier à n dimensions (n=1, 2 ou 3) il y aura n fois :
    + la coordonnée minimum sur l'axe.
    + le nombre de points sur l'axe.
    + le pas sur l'axe.
  - le nom mnémotechnique du tableau, c'est à dire le nom avec lequel il apparait dans le programme.
  - le titre du tableau (18 mots). Nous entendons par titre du tableau un commentaire que donne l'utilisateur. Par exemple le commentaire associé au tableau JARNY peut être : "caractéristiques géologiques de la mine de JARNY".
  - le reste est inutilisé.

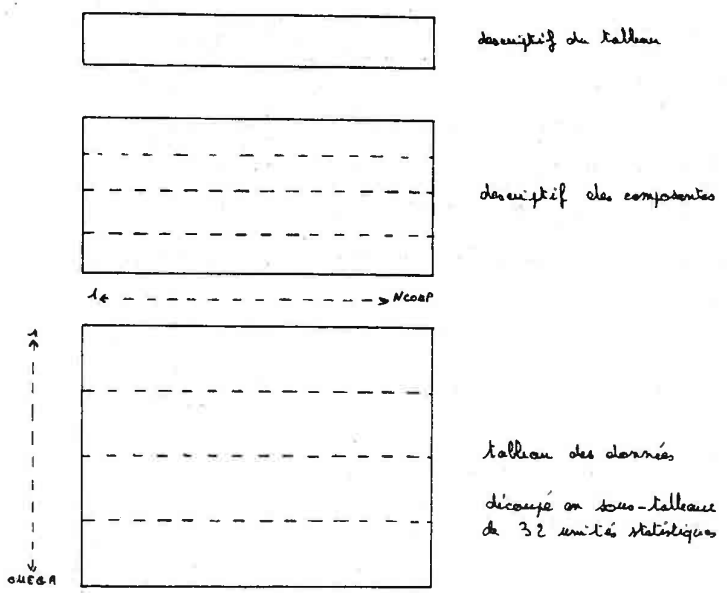

figure 5

* le descriptif de chaque composante (32 mots par composante) qui comprend :

+ le titre de la composante correspondant à un commentaire pour celle-ci.
+ le nom utilisé dans le programme pour cette composante.
+ le type de la composante (au sens entier, réel,..).
+ le type de la composante (au sens scalaire ou vectoriel).

+ le pointeur vers la composante suivante si celle-ci est vectorielle.
+ dans le cas d'une composante ensembliste :
  - la première unité statistique de l'ensemble.
  - la dernière unité statistique de l'ensemble.
  - le cardinal de l'ensemble.
  - si le tableau est régulier et en fonction de sa dimension :
    + x_{set} et x_{mset}
    + y_{set} et y_{mset}
    + z_{set} et z_{mset}
+ le reste est inutilisé.

* le tableau comprenant les valeurs proprement dites.

La figure 6 montre un exemple de composante ensembliste (zone hachurée). Dans cet exemple on remarque que :

- x_{set} (x_{mset}) représente l'abscisse minimale (maximale) correspondant à une unité statistique de la composante ensembliste.
- y_{set} (y_{mset}) représente l'ordonnée minimale (maximale) correspondant à une unité statistique de la composante ensembliste.

Ces informations permettent de définir la partie du domaine dans laquelle est contenu l'ensemble des points repérés par cette composante. Ceci est utile pour certaines fonctions incorporées du langage.

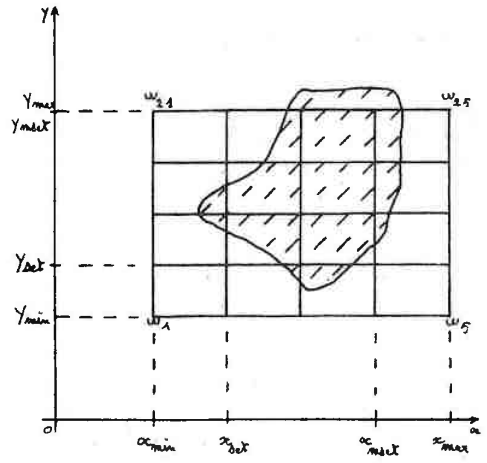

figure 6

Le tableau de données est copié dans le tableau de rangement suivant un découpage précis (figure 5) :

- les 32 mots du descriptif du tableau.
- les 32 mots du descriptif de chaque composante.
- les données proprement dites découpées en sous-tableaux de 32 unités statistiques.

Ce découpage des données par "groupe" de 32 unités statistiques a été guidé par le souci de ne privilégier aucun accès aux données. Nous avons essayé d'avoir, dans un même espace de mémoire, un certain nombre d'unités statistiques avec toutes leurs composantes associées. En effet nous nous sommes aperçus que suivant les traitements à faire il faut :

- soit accéder à toute une composante, par exemple pour le calcul de sa moyenne.
- soit accéder pour une unité statistique à plusieurs composantes, par exemple pour un calcul de corrélation.

Nous avons donc essayé de trouver un compromis ne lésant aucun de ces deux types d'accès.

III.3.B. Cas d'une déclaration d'un tableau de données.

On rappelle la déclaration d'un tableau :

{ NOUVEAU / LOCAL } TYPETAB idf NCOMP (cste) , OMEGA (CARACTERISTIQUE) ;

{ FORMEL / ANCIEN } TYPETAB idf ;

TYPETAB ::= { DT / DT1 / DT2 / DT3 }

Suivant l'option de tête un traitement différent est effectué :

- option FORMEL.

  Quand cette option est rencontrée, le compilateur vérifie que l'on se trouve dans un sous-programe de type procédure ou fonction. Aucun autre contrôle n'est effectué à cet instant car cette déclaration précise seulement que la variable idf sert à désigner un tableau de données.

- options LOCAL et NOUVEAU.

  Le traitement effectué lors de la rencontre d'une de ces options concerne la réservation d'espace dans le tableau de rangement, et la génèration d'un appel à un sous-programme, qui, au moment de l'exécution, créera le tableau de données.

  Une mise à jour du descriptif du tableau s'opère en fonction des caractéristiques NCOMP et OMEGA. Si ces caractéristiques sont paramètrées, il y a alors génèration d'un appel à un sous-programme qui, au moment de l'exécution, demande à l'utilisateur les valeurs de ces paramètres. Si l'option NOUVEAU a été utilisée, il y aura, à la fin de l'exécution, une sauvegarde sur un support physique du tableau de données. Ceci ne s'effectue pas quand l'option LOCAL a été utilisée.

- option ANCIEN.

  Quand cette option est rencontrée il y a génèration d'un appel au même sous-programme que pour les options LOCAL et FORMEL qui, lors de l'exécution, lit sur le support physique le tableau de données, le range dans le grand tableau et fait les contrôles suivants :

  * existence réelle d'un tableau de données de même nom.
  * vérification de la concordance des types (DT, DT1, DT2,

DT3).

  * si les caractéristiques NCOMP et OMEGA sont indiquées, il y aura vérification entre la déclaration et ce qui avait été stocké lors de la création du tableau.

III.3.C. Cas d'une composante d'un tableau de données.

Dans ce cas aussi, le compilateur génère un appel à un sous-programme qui lors de l'exécution fera les contrôles suivants :

- existence d'une composante de même nom si l' option ANCIEN a été employée, et, dans ce cas, vérification entre le type déclaré et le type existant effectivement.
- non-existence d'une composante de même non si l' option NOUVEAU ou l'option LOCAL a été employée, puis recherche de place libre dans le descriptif des composantes pour la création de cette composante, et, dans ce cas, mise à jour du descriptif correspondant. S'il n'y a plus de place un message d'erreur apparait et l'exécution du programme est stoppée. Une composante LOCAL n'est pas sauvegardée sur le support physique à la fin de l'exécution.

L'option FORMEL n'est utilisée que dans les sous-programmes et précise que la variable IDF est une composante d'un tableau de données.

III.3.D. Traduction en langage FORTRAN.

III.3.D.1. Traduction des tableaux de données.

Le rangement d'un tableau de données dans le grand tableau est fait par le sous-programme qui assure les contrôles lors de l'exécution, et ceci a donc lieu dans le programme principal FORTRAN.

Un tableau de données possédant des composantes de types différents, trois déclarations différentes (ENTIER, REEL, BOOL) du tableau de données dans le sous-programme principal sont nécessaires, car en FORTRAN il n'y a que trois types possibles. Ces trois déclarations pointeront sur la même adresse dans le tableau de rangement.

III.3.D.2. Traduction d'une composante

Pour une composante l'accès se fait par l'intermédiaire du nom du tableau de données auquel elle appartient. Pour cela il est nécessaire de connaître :

- la position de cette composante dans le tableau de données. A chaque composante correspond une variable qui précise l'adresse de début de celle-ci dans le tableau. Cette variable est initialisée dans le sous-progamme appelé à l'exécution, qui vérifie l'existence de cette composante.
- le type de la composante, pour connaître quel nom de tableau de même type utiliser.

III.3.D.3. Traduction d'une variable status.

Soit la déclaration suivante :

STATUS I OF JARNY

puis l'affectation :

I=33

L'affectation précise que l'on veut désigner la $33^{ème}$ unité statistique. Comme en FORTRAN les tableaux sont rangés colonne par colonne et comme un tableau de données est découpé en sous-tableaux de 32 unités statistiques, cette unité statistique n'est pas physiquement rangé aprés la $32^{ème}$. Il faut donc calculer l'adresse réelle correspondante.

A la déclaration d'un status correspond la déclaration de deux variables FORTRAN, l'une pour la valeur de la variable déclarée STATUS, et l'autre pour l'adresse réelle.

A chaque affectation de status correspond :

- la même affectation en FORTRAN
- le calcul de l'adresse physique correspondante.

Remarque

On trouvera en annexe III des exemples de traduction des différents cas présentés ci-dessus.

III.4. Différents problèmes rencontrés.

III.4.A. Création d'une composante ensembliste.

Comme nous l'avons vu au chapitre II, une composante ensembliste sert à repérer sous un même nom un ensemble d'unités statistiques vérifiant une même propriété.

La réalisation concrète de cette composante est un chaînage reliant entre elles les unités statistiques devant appartenir au même ensemble. Pour une unité statistique, les valeurs possibles du champ correspondant à la composante ensembliste sont :

* -1 si cette unité statistique n'appartient pas à l'ensemble.
* val qui est le numéro de la prochaine unité statistique appartenant à l'ensemble.
* 0 pour la dernière unité statistique de l'ensemble.

La création d'une composante ensembliste se fait par un parcours complet du tableau de données dans l'ordre croissant des unités statistiques. Le chaînage créé est de sens inverse, c'est à dire que lorsque l'on parcourt l'ensemble, on le fait dans l'ordre décroissant des unités statistiques.

En même temps que la création du chaînage, il y a mémorisation de certaines informations qui serviront à la mise à jour du descriptif de la composante notamment :

- le cardinal de l'ensemble.
- x_{min} , x_{max} , y_{min} , . . .
- la première unité statistique de l'ensemble.
- la dernière unité statistique de l'ensemble.

Prenons l'exemple d'un tableau de données à deux dimensions, figure 7, dans lequel on a défini la composante ensembliste représentant la zone hachurée.

Ce que l'on obtient pour la composante ensembliste apparait à la fois au niveau de son descriptif, et au niveau du chaînage créé pour les parcours ultèrieurs.

Cette méthode de construction a été employée quelle que soit la définition de la composante, c'est à dire :

- soit une définition en fonction d'une propriété.
- soit une définition à partir d'autres composantes ensemblistes.

Dans le $2^{ème}$ cas il aurait été possible de ne parcourir que les unités statistiques mises en cause par les composantes, mais le travail aurait été plus complexe, et surtout le chaînage obtenu aurait été inverse au précédent, ce qui, lors de l'utilisation de cette composante pour la définition d'une autre, aurait posé des problèmes de parcours du tableau. Nous avons donc préféré une uniformisation.

Il est à noter que l'ordre du chaînage créé n'importe pas, car dans les ensembles utilisés en analyse de données, il n'y a pas de relation d'ordre entre les différentes unités statistiques.

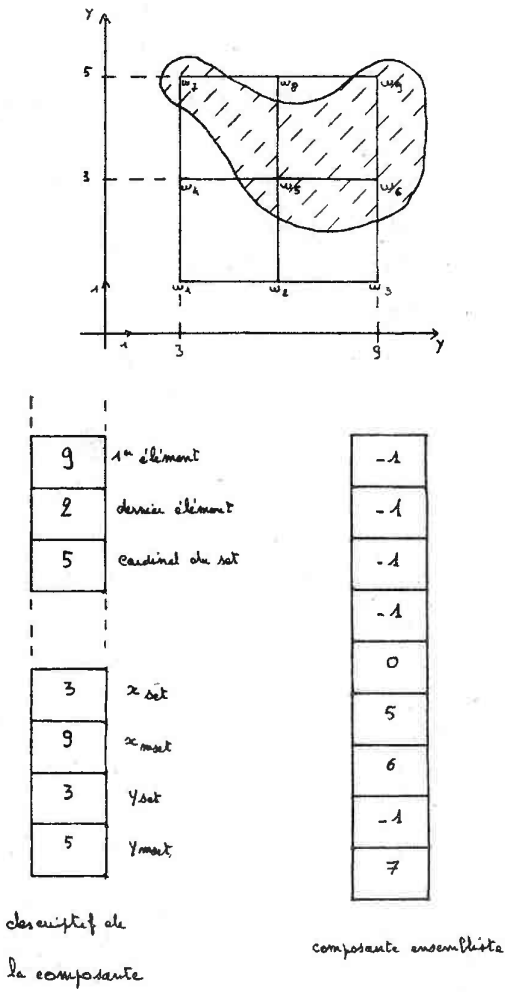

figure 7

III.4.B. Passage des paramètres.

Au niveau des sous-programmes apparaissent deux problèmes :

- le contrôle de concordance entre les paramètres formels et les paramètres effectifs.
- la traduction en langage FORTRAN.

* Contrôle de concordance de type.

Une vérification de la compatibilité entre le type du paramètre formel et celui du paramètre effectif s'impose. Ce contrôle se fait en cours de compilation à chaque appel de sous-programme, si sa définition a déjà été rencontrée, sinon il est réalisé à la fin de la compilation du texte source, au moyen d'un catalogue de bibliothèque. Nous reviendrons sur ce point dans le chapitre IV.

* Contrôle de concordance d'appartenance.

Une autre vérification importante se fait en cours de compilation. Soit la déclaration de sous-programme suivante :

```
PROCEDURE SSP1 ( A OF TAB, B OF TAB1, C OF TAB)
```

Nous avons vu que cette déclaration précise que la première et la troisième composante figurant en arguments, doivent appartenir au même tableau de données. A chaque appel de cette procédure, le compilateur vérifie que le 1^{er} et le $3^{ème}$ paramètres effectifs sont des composantes appartenant à un même tableau de données.

Remarquons que la deuxième composante peut être définie sur le même tableau ou sur un autre.

* Traduction en langage FORTRAN.

Nous avons vu comment étaient repérées les composantes en FORTRAN (paragraphe III.3.D). Ce principe implique une traduction spécifique des paramètres de type composante. En effet, il faut non seulement transmettre le tableau de données auquel ces composantes appartiennent, mais aussi la variable (associée à la déclaration de chaque composante), qui permet l'accés aux valeurs de la composante.

On trouvera en annexe III un exemple de traduction d'une déclaration et d'un appel de sous-programme.

III.4.C. Création de liste

Nous avons montré que la définition d'une liste comportait en même temps deux parties :

- la déclaration de la liste.
- la définition de la liste.

La partie déclaration est traduite par la déclaration d'une variable de type tableau dans laquelle sont rangés les pointeurs permettant l'accès aux composantes.

La partie définition correspond à une initialisation de ce tableau aux valeurs des pointeurs d'accès. Le compilateur vérifie :

- que les composantes définissant la liste sont du même type que celle-ci.
- que lors de la destruction de composantes d'un tableau de données, celles-ci n'étaient pas utilisées pour la définition d'une composante liste.

III.5. Conclusion.

Nous n'avons exposé ici que les points qui nous sont apparus comme les plus importants et caractéristiques du langage GEOL.

L'utilisation du compilateur GEOL permet de constater que le rapport "instructions GEOL / instructions FORTRAN générées" est en moyenne de 1 pour 6. Pour certaines instructions, comme la définition d'une composante ensembliste, le rapport passe à 1 pour 40.

CHAPITRE IV

SYSTEME GEOL ET INTERFACE

Systeme GEOL et interface

IV.1. Introduction.

Le projet initial n'était que la définition d'un langage répondant aux besoins des Géologues et l'écriture du compilateur associé.

L'usage des langages tels que le langage GEOL nous a montré l'utilité d'un environnement de programmation ainsi qu'il est décrit dans °STO-80§. Nous avons élaboré un système qui peut être décomposé en deux parties :

- une partie fabrication de programmes.
- une partie utilisation de programmes.

Il comprend par ailleurs une partie gestion de fichiers (de données ou de bibliothèques), ainsi qu'une partie génération de cartes de commandes. Ces deux éléments sont intégrés dans la fabrication ou l'utilisation de programmes. S'appuyant sur une démarche conversationnelle, le système guide l'utilisateur dans son écriture de programmes, par l'intermédiaire de questions pertinentes. Le dialogue comporte deux étapes :

- la définition du travail que veut accomplir l'utilisateur.
- la définition des différents paramètres utiles au moment de l'exécution du programme, et qui sont décrits au chapitre II.

IV.2. Système GEOL.

L'objectif de ce système est d'alléger la charge de l'utilisateur dans la gestion des fichiers qu'il utilise, et d'assurer l'enchaînement des différentes tâches qui lui sont proposées.

Dans la suite du chapitre, nous appelons dans notre système tâche, une étape minimale que l'on peut faire, c'est à dire :

- soit une compilation.
- soit une édition de liens.
- soit une exécution de programmes.
- soit une mise en bibliothèque.

Nous avons défini, dans le cadre du système comprenant le langage GEOL, cinq tâches de base, qui sont les suivantes :

- G->F : Compilation d'un programme écrit en GEOL, et sauvegarde du programme FORTRAN généré.

- F->OM : Compilation d'un programme écrit en FORTRAN, et sauvegarde du code généré.

- OM->L : Mise en bibliothèque d'un programme sous forme de code objet.

- OM->LM : Construction d'un module exécutable ("load-module") à partir d'un code objet, et sauvegarde du module exécutable.

- LM->EX : Exécution d'un module exécutable.

L'utilisateur a le choix entre neuf travaux qui sont des combinaisons de ces tâches, et que voici :

Geol-Fortran

Compilation d'un programme écrit en GEOL équivalent à la tâche G-F.

Geol-Librairie

Mise en bibliothèque d'un sous-programme écrit en GEOL, équivalent à l'enchaînement : G-F, F-OM, OM-L.

Geol-Exécution

Exécution d'un programme principal écrit en GEOL, équivalent à l'enchaînement : G-F, F-OM, OM-LM, LM-EX.

Geol-Load Module

Sauvegarde d'un load-module créé à partir d'un programme écrit en GEOL, équivalent à l'enchaînement suivant : G-F , F-OM , OM-LM.

Load Module-Exécution

Exécution d'un load module, équivalent à : LM-EX.

Fortran-Module Objet

Compilation d'un programme écrit en FORTRAN et sauvegarde du code généré, équivalent à : F-OM.

Fortran-Librairie

Mise en bibliothèque d'un sous-programme écrit en FORTRAN, équivalent à l'enchaînement : F-OM, OM-L.

Fortran-Exécution

Exécution d'un programme écrit en FORTRAN, équivalent à l'enchaînement : F-OM , OM-LM , LM-EX.

Fortran-Loadmodule

Création d'un module exécutable à partir d'un programme écrit en FORTRAN, équivalent à l'enchaînement : F-OM , OM-LM.

Dans ces différents travaux proposés, seuls les fichiers de départ et d'arrivée sont sauvegardés.

Ce programme système est conçu comme un automate, qui oriente et dirige les différents choix que doit faire un utilisateur en fonction du travail qu'il veut effectuer.

Un dialogue est donc établi et se déroule en deux étapes :

- choix du travail.
- en fonction de ce choix, des questions sont posées par le programme, et leurs réponses permettront le lancement et l'exécution du travail.

Prenons l'exemple d'un utilisateur qui, ayant écrit un programme en GEOL, demande son exécution. Lors du choix du travail, l'utilisateur prendra donc l'option G-E (Geol-Exécution) décrite précédemment. A partir de ce choix, le système posera les questions permettant de préciser :

- le nom du fichier où se trouve le programme source.
- le nom des bibliothèques où se trouvent les différents sous-programmes appelés par le programme que l'on compile, ce qui permettra les contrôles ayant lieu lors des appels de sous-programmes.
- le nom des bibliothèques dans le cas des appels à des sous-programmes écrits en FORTRAN, car la démarche pour ces appels est différente (voir paragraphe IV.3.).
- le mode d'exécution désiré.

Une fois ces différents paramètres sont fixés, il y a création des cartes de commande puis exécution du travail demandé, soit en mode direct, soit en mode différé.

Le mode direct doit être utilisé pour l'exécution d'un programme dans lequel certaines déclarations ont été paramétrées. Un nouveau dialogue sera instauré au moment de l'exécution du programme FORTRAN généré.

IV.3. Gestion des bibliothèques.

Dans ce système, il existe deux types de bibliothèques :

- les bibliothèques créées à partir de sous-programmes écrits en GEOL.
- les bibliothèques que possèdent l'utilisateur et qui ont été créées à partir de sous-programmes écrits en FORTRAN.

Pour les sous-programmes qui appartiennent aux bibliothèques du deuxième type, les appels sont écrits en FORTRAN, donc le compilateur GEOL les ignore. Aucun contrôle de type n'est fait sur les paramètres. Ces bibliothèques n'interviennent qu'au niveau de l'édition de liens, la sécurité n'est donc pas assurée dans ce cas, mais nous pensons que ce type de bibliothèque aura tendance à disparaître avec l'emploi du système GEOL.

Nous allons détailler maintenant la gestion des bibliothèques construites à l'aide du système GEOL.

L'utilisation de ces bibliothèques intervient à deux niveaux :

- dans la compilation d'un programme écrit en GEOL, pour les vérifications de type des paramètres.
- dans l'édition de liens, soit pour exécuter un programme GEOL, soit pour créer un module exécutable.

La structure d'un programme GEOL permet de déclarer les sous-programmes aussi bien avant le programme principal qu'aprés. Quand aux contrôles effectués sur les paramètres, ils sont faits :

- soit lors de l'appel du sous-programme, si sa définition a déjà été rencontrée.
- soit lors de sa définition, pour les appels précédents.
- soit à la fin de la compilation pour les appels de sous-programmes existant dans une bibliothèque.

Pour ce troisième point, il n'est pas nécessaire de connaître le texte du sous-programme, mais il faut posséder un descriptif de sa définition. Par descriptif nous entendons :

- le nombre de paramètres du sous-programme.
- pour chaque paramètre:
  * son type.
  * son appartenance en tant que composante à un tableau de données, Si tel est le cas le descriptif doit aussi indiquer son éventuelle appartenance au même tableau qu'une autre composante figurant en paramètre.

Nous avons regroupé l'ensemble des descriptifs des sous-programmes dans un catalogue, et afin de simplifier les recherches, nous avons établi un catalogue par bibliothèque.

Sans entrer dans le domaine du suivi de programmes °ROY-82§, nous sommes obligés d'aborder certains problèmes de gestion de versions. Nous avons vu que le système n'autorise l'exécution du code FORTRAN généré, que si aucune erreur n'a été détectée. Lorsqu'on sauvegarde un programme GEOL compilé (c'est à dire un programme FORTRAN), on a l'assurance, à cet instant, d'une

parfaite concordance entre un appel à un sous-programme et sa définition. Si par la suite il y a un changement de définition du sous-programme, la concordance disparaît, et la compilation FORTRAN ne permettra pas de le découvrir. Pour éviter ce risque, nous avons décidé de conserver également, lors du stockage d'un programme, le texte FORTRAN des sous-programmes appelés. Ainsi même si une modification est apportée à l'intérieur du sous-programme, nous pouvons assurer que le programme stocké donnera toujours le même résultat. Cette démarche est sûre, mais présente le désavantage de coûter de la place. Ainsi, un sous-programme appelé par quatre programmes existera cinq fois, quatre fois avec les programmes qui l'appellent, et une fois dans la bibliothèque. C'est pour remédier partiellement à ce problème que nous avons décidé d'établir deux types de bibliothéque :

- une bibliothèque standard, fournie avec le système, et qui contient les sous-programmes les plus fréquemment utilisés en cartographie et en analyse de données. L'utilisateur n'a accès à cette bibliothèque qu'en mode lecture, c'est à dire qu'il ne lui est pas permis de modifier un des sous-programmes.
- une ou des bibliothèques personnelles que l'utilisateur se compose lui-même, et dont il prend en charge la gestion.

Selon le type de bibliothèques à laquelle appartient le sous-programme, la démarche effectuée pour la sauvegarde sera différente. Les sous-programmes appartenant à la bibliothèque standard étant figés, ils ne seront pas dupliqués lors du stockage d'un programme. On ne dupliquera donc que les sous-programmes personnels, car eux peuvent être modifiés.

Lors de la mise en bibliothèque d'un sous-programme, deux cas peuvent se présenter:

- le sous-programme n'existe pas, dans ce cas, une mise en place du

descriptif dans le catalogue de la bibliothèque s'effectue, ainsi qu'un stockage du code objet.

- le sous-programme existe, dans ce cas, il y a changement du code objet et mise à jour du descriptif correspondant.

Il faut préciser ici que la modification faite dans un sous-programme de bibliothèque personnelle n'entraîne nul changement pour les programmes GEOL déjà compilés, car ils sont sauvegardés avec une ancienne version de ce sous-programme.

Pour la recherche d'un descriptif, on s'adressera d'abord aux catalogues des bibliothèques personnelles, et ensuite au catalogue de la bibliothèque standard. Ainsi les bibliothèques personnelles sont prioritaires par rapport à la bibliothèque standard.

IV.4. Interface machine et système GEOL

Afin d'exécuter les différentes tâches présentées au paragraphe IV.2., le système GEOL demande de façon conversationnelle un certain nombre de renseignements, précisant entre autres :

- le choix des options du compilateur.
- les noms de fichiers, et les numéros des unités logiques associées, où seront lus ou sauvegardés les tableaux de données.
- le mode d'exécution.

IV.4.A. Choix des options offertes par le compilateur

- Edition d'un listing de la compilation.
- Unité logique où sera lu le programme source.
- Unité logique où sera sauvegardé le programme objet.
- Unité standard pour les entrées-sorties que l'on peut redéfinir dans le programme source.

IV.4.B. Unités logiques

Nous avons indiqué dans le chapitre III, que par l'intermédiaire de l'en-tête d'un programme GEOL l'utilisateur fixe la taille du tableau de rangement des tableaux de données. Ce même en-tête permet de préciser les unités logiques sur lesquelles seront lus ou sauvegardés les tableaux de données. La syntaxe de cet en-tête est la suivante :

PARAM (MEM = CSTE , { IMPR / LECT / IDF } = { CSTE / ? }) ;

Par exemple:

```
PARAM ( MEM =20000 ,  IMPR = 2,  TAB = 10 )
```

Cet en-tête :

- précise la taille du tableau FORTRAN qui permet de simuler l'allocation dynamique de mémoire.
- redéfinit l'unité logique de sortie (IMPR = 2), l'unité logique d'entrée restant la même.
- définit l'unité logique sur laquelle le tableau de données TAB sera

lu. Seul le nom TAB apparaîtra ensuite au niveau de la déclaration des tableaux. Le compilateur vérifie que pour chaque tableau déclaré une unité logique a été définie.

Remarque

Si le caractère ? a été employé, l'unité logique ne sera alors définie qu'au moment de l'exécution du programme (de façon conversationnelle).

CHAPITRE V

PORTABILITE DU SYSTEME

Portabilité du système

Portabilité du système

V.1. Introduction

Dans ce chapitre, il n'est pas question pour nous d'aborder la théorie de la portabilité °SOFT-75§, mais seulement d'exposer la démarche que nous avons adoptée lors de l'écriture du système GEOL, dans le but de faciliter sa portabilité. Nous abordons également les problèmes que nous avons rencontrés lors du transfert du système sur un autre site.

Nous définissons deux niveaux de portabilité :

- la portabilité du système GEOL, donc du compilateur.
- la portabilite des programmes écrits en GEOL.

Nous disons qu'un programme écrit en GEOL est portable, si son exécution sur deux sites différents donne des résultats identiques, et ceci quelles que soient les modifications apportées au compilateur lors de son installation. Ainsi une non-portabilité du compilateur d'un langage n'entraîne pas nécessairement une non-portabilité des programmes écrits dans ce langage.

Un des moyens permettant d'écrire un programme portable °RICH-75§, consiste à utiliser un langage standard, qui de plus est un langage universel, car :

- il permet l'écriture de programmes pour toutes les applications.
- il existe sur la plupart des machines.

C'est ce moyen que nous avons utilisé, le langage standard choisi étant le langage FORTRAN, ou plus exactement le noyau du langage FORTRAN IV que de nombreuses études existant dans la littérature, permettent de cerner de façon précise. Le compilateur et le système sont écrits en FORTRAN, et traduisent un programme GEOL en programme FORTRAN.

Les problèmes les plus fréquents que l'on rencontre lors du transfert d'un compilateur sont liés aux parties dépendantes de la machine :

- le code objet. La partie génération du code objet n'est donc pas portable.
- les entrées-sorties. Les interfaces pour les entrées-sorties ne sont donc pas portables.

Le fait d'écrire un compilateur générant un langage de haut niveau, comme FORTRAN, permet de supprimer ces deux points. Ceux-ci sont pris en charge par le compilateur du langage FORTRAN qui existe déjà sur la machine hôte.

Il se trouve qu'avant d'écrire le compilateur, nous avions l'expérience de l'écriture et du transport d'un logiciel de cartographie. Ce logiciel a été construit sur IRIS 80 et a été installé sur d'autres machines telles que IBM, VAX, CDC... Cette expérience nous a permis de connaître avec plus de précision le noyau du langage FORTRAN commun aux machines que nous prévoyons pour l'installation de ce système.

V.2. Portabilité du système GEOL.

Le système GEOL ne peut pas globalement être considéré comme portable, car nous avons vu dans le chapitre précédent qu'un de ses buts était de générer, en fonction des demandes de l'utilisateur, les cartes de commandes pour l'exécution du travail. Il est bien évident que cette partie est entièrement dépendante de la machine sur laquelle on se trouve. Pour remédier facilement à ce problème nous avons enfermé dans des sous-programmes toutes les parties que l'on sait dépendantes de la machine.

En dehors de ce problème, que l'on connaissait, nous n'en avons rencontré qu'un autre, provenant du codage des caractères. En effet lors de l'analyse lexicographique, nous recherchons dans des tables si l'unité lexicographique rencontrée correspond à un mot clé ou à une fonction prédéfinie. Ces recherches sont faites de manière dichotomique, ce qui impose un classement des mots suivant l'ordre croissant de leur codage. Ces tables étaient initialisés par un ordre DATA en FORTRAN. Sur IRIS 80, par exemple, l'ordre alphabétique des mots correspond à l'ordre croissant de la valeur de leurs codages, alors que sur d'autres machines, comme VAX, la valeur du codage ne respecte pas l'ordre alphabétique. Dans de tels cas les recherches dichotomiques se sont avérées fausses.

Pour résoudre ceci deux solutions s'offraient à nous :

- modifier l'initialisation des tables, faite par un DATA, chaque fois que l'on change de machine.
- écrire un sous-programme qui transforme la table des mots-clés rangés par ordre alphabétique, en table où les mots clés seront rangés par ordre croissant de leur codage. Cette opération s'effectue lors de l'appel du compilateur.

Nous avons préféré la 2ème solution, qui bien que légèrement coûteuse en temps et en place, rend le système plus indépendant de la machine hôte.

Avec cette correction, le transfert du système GEOL se résume à la réécriture des sous-programmes de génération des cartes de commande.

V.3. Portabilité des programmes écrits en GEOL.

En dehors des problèmes liés à la précision des calculs arithmétiques, nous pensons pouvoir assurer la portabilité des programmes écrits en GEOL, une fois que le système a été implanté.

V.4. Conclusion

La portabilité du système nous semble avoir été trés fortement facilitée par l'expérience acquise lors de l'écriture du logiciel de cartographie. Ainsi nous n'avons pas utilisé certaines des facilités du FORTRAN IV installé sur IRIS 80, quand on savait que celles-ci n'existaient pas sur les autres machines.

Cependant nous n'avons pas l'intention d'implanter le système GEOL sur des micro-ordinateurs pour deux raisons :

- le système GEOL est prévu pour la gestion de gros tableaux de données, ce qui impose une mémoire de travail importante.
- le langage FORTRAN existant sur les micro-ordinateurs est souvent un FORTRAN spécifique, c'est à dire qu'il ne correspond pas toujours à

la définition du langage FORTRAN standard.

CHAPITRE VI

ETUDE D'UN PACKAGE DE STATISTIQUE

Etude d'un package de statistique

Etude d'un package de statistique

VI.1. Introduction

Le langage GEOL nous semble n'être qu'une première étape dans la définition d'un langage spécialisé pour l'analyse de données, qu'elles soient géologiques ou statistiques, car en dehors de la simplification et de la sécurité apportées, le langage GEOL ne remet pas en cause l'approche des problèmes et la programmation de leurs résolutions.

Nous avons essayé d'inclure dans le langage GEOL les résultats des différentes recherches faites sur les langages. Aussi allons nous maintenant étudier l'apport, lors de la programmation, de nouveaux concepts tels que les types abstraits.

Nous attachant plus particuliérement aux outils de programmation, nous entreprenons notre étude à partir de la définition de "package" telle qu'elle existe dans le langage ADA °ADA-81§, en raison de la spécificité des problèmes abordés en géologie, nous allons essayer de définir un package statistique.

Nous supposons pour la suite que nous avons un tableau de données, JARNY, tel qu'il a été défini dans le langage GEOL, et que sur celui-ci nous avons défini :

- des composantes ensemblistes ENS1, ENS2.
- des composantes fonctionnelles SIO2, AL2O3, PROBA.

La composante PROBA contient les différentes valeurs de la fonction de pondération associée aux points du domaine, une fonction de pondération désignant la probabilité associée à chaque point du domaine.

VI.2. Définition d'un package

Un package se décompose en deux parties :

- une partie interface qui est visible pour tout utilisateur et qui comprend :
  - * la déclaration du package.
  - * la liste des opérations autorisées par ce package.
- une partie implémentation cachée de l'utilisateur, qui contient les instructions correspondant aux opérations permises.

VI.3. Définition du problème.

En analyse de données, certaines opérations apparaissent très fréquemment ; ainsi est-on amené souvent à calculer la moyenne, la variance, la covariance de variables aléatoires. Une première possibilité s'offre pour simplifier l'écriture de programmes, qui consiste à définir des sous-programmes et à les appeler en temps voulu. Cette solution est généralement adoptée, encore qu'elle n'apporte aucune amélioration notable, que ce soit du point de vue du temps de calcul, de la mémorisation d'informations et de la sécurité, et elle a l'inconvénient d'éloigner l'utilisateur des notations mathématiques habituelles.

Nous choisissons d'aborder le problème différemment. Sachant que les différents calculs statistiques sont tous faits sur des variables aléatoires, l'idée première est de définir un "package variable aléatoire" comprenant :

- la définition d'une variable aléatoire qui est connue par :
  - * l'ensemble sur lequel elle est définie.

  - * la fonction de probabilité qui lui est associée.
  - * les valeurs qu'elle prend aux points de son domaine de définition.
- la liste des opérations possibles, par exemple ;
  - * le calcul de la moyenne.
  - * le calcul de la variance.

Si l'on suppose avoir défini les types suivants :

- type TSET : array of integer
- type TREEL : array of reel
- type TINTEGER : array of integer

la définition d'un tel package en langage ADA est :

```
generic type T1, X:T1, ENS:tset, POND:treel;
    package VAR_ALEA is :
        function ESP return reel;
        function VAR return reel;
    end VAR_ALEA;
```

L'utilisation se fera de la manière suivante :

- déclaration d'une variable aléatoire :

  V_A_SIO2 is new VAR_ALEA(treel,SIO2 de JARNY,ENS1,PROBA);

- utilisation des opérations :

  l'appel V_A_SIO2.ESP va calculer l'espérance mathématique de la composante SIO2 du tableau de données JARNY.

Il est à noter que les variables apparaissant en paramètres sont toutes des composantes appartenant au même tableau de données, ainsi pour ENS1 et PROBA n'est-il pas nécessaire de préciser le tableau de données, car il est indiqué par la variable SIO2.

La partie implémentation a la forme suivante :

```
package body VAR_ALEA is :
    E:reel;
    V:reel;
    VAC : treel;
    function ESP is :
        begin
        --calcul de la moyenne et stockage du résultat dans la
        variable E, puis calcul de la variable aléatoire centrée
        réduite stockée dans le tableau VAC--
        end ESP;
    function VAR is :
        begin
        --calcul de la moyenne si elle n'est pas calculée puis
        calcul de la variance à partir du tableau VAC--
        end VAR;
    end VAR_ALEA;
```

La déclaration d'une variable aléatoire correspond à une instantiation du package, c'est à dire à une recopie de la partie implémentation en fonction des paramètres effectifs. L'utilisation des opérations correspond, elle, à un appel de sous-programmes.

Le fait d'être obligé de préciser le type de la composante lors de la déclaration de la variable aléatoire, provient de la définition même du langage ADA. Cette notation paraît trés lourde aux utilisateurs qui aimeraient ne pas répéter le type de la variable. Une définition plus acceptable peut être proposée ainsi :

```
generic X, ENS : tset, POND : treel;
    package VAR_ALEA is:
    function ESP return reel;
    function VAR return reel;
    end VAR_ALEA;
```

X cette fois symbolise la composante et le type de ses éléments sur laquelle porte le package.

L'utilisation se fera de la manière suivante :

- déclaration d'une variable aléatoire :

  V_A_SIO2 is new VAR_ALEA (SIO2 de JARNY,ENS1,PROBA);

- utilisation des opérations :

  V_A_SIO2.ESP et V_A_SIO2.VAR

permettent de calculer la moyenne ou la variance de la composante SIO2.

Cette facilité d'écriture est envisageable en GEOL, car contrairement au langage ADA, il existe un nombre fixe de types possibles (entier, réel ou logique). Il suffit donc d'avoir les trois formes possibles du package, et au moment de la déclaration, de recopier la version correspondant au type de la composante.

L'apport de ce package intervient à deux niveaux :

- pour le raisonnement : ce package permet de rester proche de l'habitude mathématique, qui est de se définir une variable aléatoire, --ici instantiation du package--, puis d'y travailler dessus, --ici appels des opérations permises--.
- pour le calcul : dans la partie implémentation, qui est cachée à l'utilisateur, certaines informations sont sauvegardées, qui peuvent être réutilisées par la suite. Ainsi quand la moyenne a été calculée sa valeur est conservée, et n'aura plus a être recalculée par la suite.

Cependant une réflexion plus précise sur l'utilisation de ce package peut faire apparaître quelques réticences :

- si l'on veut travailler sur la variable aléatoire correspondant à la variable "2 fois SIO2", on ne peut utiliser le package V_A_SIO2, et il faut alors en redéfinir un nouveau.
- ce package ne permet pas de manipuler en même temps deux variables aléatoires, ce qui interdit le calcul d'une covariance par exemple.
- si l'on respecte la définition du langage ADA, il n'est pas permis de comparer les espérances mathématiques de deux variables aléatoires, car V_A_SIO2.ESP et V_A_AL203.ESP ne sont pas de même type.
- pour améliorer les temps de calcul, nous avons indiqué que l'on

mémorisait certaines informations, ce qui entraîne deux constatations :

* la place utilisée en mémoire centrale est augmentée.
* toutes modifications du tableau de données sont interdites, afin de rendre certain que les résultats conservés restent valables.

Ce sont ces différents points, et notamment les trois premiers, qui nous ont amenés à définir un autre package.

VI.4. Deuxième définition du package.

Il parait plus intéressant de préciser d'abord un espace probabilisé avant de définir les opérations qui seront autorisées. Un espace probabilisé est défini par le domaine de définition des variables aléatoires à étudier, et par la fonction de pondération associée à ce domaine. La nouvelle définition devient ainsi :

```
generic ENS : tset, POND : treel
    package ESP_PROB is
        function ESP(X) return reel;
        function VAR(X) return reel;
        function COVAR(x,y) return treel;
        function CORREL(x,y) return treel;
    end ESP_PROB;
```

X,Y désignent les composantes fonctionnelles sur lesquelles on veut travailler.

L'utilisation en sera la suivante :

- définition d'un espace probabilisé :

  ESPACE is new ESP_PROB(ENSl de JARNY,PROBA);

- utilisation des opérations :

  le calcul de la covariance des deux composantes SIO2 et AL203 s'écrira :

  A=ESPACE.COVAR(SIO2,AL203);

  cette opération suppose que l'on autorise les affectations de matrices.

Là aussi nous nous sommes éloignés du langage ADA, car nous ne désirons pas faire apparaître le type des composantes sur lesquelles nous voulons travailler. La syntaxe du langage ADA nous oblige à écrire :

```
function ESP(type T1,x:T1);      pour la définition de la fonction ESP
ESPACE.ESP(treel,SIO2)           pour l'utilisation de la fonction ESP.
```


VI.4.A. Remarques.

Apportons quelques précisions sur les composantes apparaissant en tant que paramètres.

- Type des composantes.

  On peut avoir des composantes de type ENTIER ou REEL, mais aussi de type BOOL ou SET. Le résultat sera dans tous les cas de type REEL. Dans le cas de composantes de type BOOL ou SET, on donnera à la composante le sens de la fonction indicatrice ; ainsi, par exemple, la moyenne sera la probabilité de cette fonction indicatrice.

- Taille des composantes.

  Si l'on admet que l'on dispose d'instructions permettant les calculs matriciels, la taille des composantes ne pose aucun problème.

  Par exemple :

  soit l'affectation :

  M = ESPACE.ESP (SIO2);

  le compilateur GEOL, connaissant la taille de SIO2 génère après différentes vérification :

  * une affectation entre deux variables scalaires, si les variables SIO2 et M ont été déclarées scalaires.
  * une affectation matricielle, si les variables SIO2 et M ont été déclarées vectorielles.

Ce package répond aux points évoqués précédemment :

- avec le même package il est possible de travailler sur SIO2 ou sur 2*SIO2.
- Par l'intermédiaire des fonctions COVAR ou CORREL, il est permis de

travailler en même temps sur deux variables aléatoires. De plus, on est sûr alors qu'elles seront bien définies sur le même espace probabilisé.

- une comparaison est devenue possible entre les moyennes de variables aléatoires, car elles sont considérées de même type.

VI.5. Implémentation du package.

Le modèle d'implémentation sera le suivant :

```
package ESP_PROB is
    function ESP is
        begin
        --calcul de la moyenne--
        end
        --de même pour VAR,COVAR et CORREL
    end ESP_PROB;
```

Il nous faut maintenant détailler la partie implémentation d'un package de ce type, en prenant l'exemple du calcul de la moyenne.

Quand le compilateur rencontre une instantiation du package, le travail suivant est effectué :

- sélection du tableau de données sur lequel on définit un espace probabilisé.
- sélection d'une composante ensembliste.
- sélection de la composante fonction de pondération.
- recopie des sous-programmes correspondant aux différentes opérations

autorisées par ce package.

Pour ce dernier point s'offrent deux solutions.

* On sait qu'il existe trois types possibles en FORTRAN, et l'on peut, pour chaque sous-programmes, avoir autant de versions que de types. Dans le cas présent, comme il y a quatre sous-programmes et trois types lors de l'instantiation, il y a recopie des douze sous-programmes. Lors de l'utilisation, il y a appel au sous-programme correspondant au type de la composante paramètre.

Dans le cas du calcul de la moyenne, les sous-programmes FORTRAN qui sont appelés, ont la déclaration suivante :

```
function real ESPENT(TAB,VEC1,VEC2,VEC3)
function real ESPREEL(TAB1,TAB2,VEC1,VEC2,VEC3)
```

Pour la fonction ESPENT :

TAB est la version entière du tableau de données.

VEC1 est le vecteur d'accès à la composante ensembliste.

VEC2 est celui de la fonction de pondération.

VEC3 est celui de la composante dont on calcule la moyenne.

Pour la fonction ESPREEL :

TAB,VEC1,VEC2,VEC3 ont la même signification.

TAB2 est la version réelle du tableau de données. Ici les deux versions du tableau de données sont nécessaires, la version entière étant utile pour la composante ensembliste, et la version réelle pour la composante fonctionnelle dont on calcule la moyenne.

* Il est possible de n'avoir qu'une seule version des sous-programmes, et suivant les cas d'exécuter telle ou telle séquence d'instructions. La déclaration du sous-programme est la suivante :

function reel ESP(TAB1,TAB2,TAB3,VEC1,VEC2,VEC3,I)

TAB1,TAB2,TAB3 sont les versions du tableau de données.
VEC1,VEC2,VEC3 sont les trois vecteurs d'accès aux composantes.
I est le "sélecteur d'instructions" prenant les valeurs 1,2 ou 3 suivant le type de la composante fonctionnelle.

La première solution est plus rapide à l'exécution que la seconde, mais présente le désavantage d'occuper plus de place en mémoire centrale. Il est aussi tout à fait possible de combiner les deux, c'est à dire pour certains sous-programmes, d'avoir autant de versions que de types possibles, et pour d'autres, d'avoir recours à un sélecteur d'instructions. Nous pensons qu'il n'y a pas de règles permettant un choix.

Ces deux solutions, qui permettent de réaliser l'implémentation d'un package, ne sont pas très agréables dans leurs réalisations. Il aurait été plus simple de faire apparaître le type des composantes en paramètres, mais cela aurait été au détriment de la facilité d'écriture de programmes. Actuellement, on est sur ce point placé devant l'obligation de faire un choix entre une utilisation agréable et une implémentation rationnelle.

VI.6. Conclusion.

Nous n'avons pas intégré ces possibilités dans le langage GEOL, car leur besoin s'est fait sentir qu'après la première définition du langage, que nous ne considérions que comme une première étape.

Nous avons voulu montrer dans ce chapitre, qu'il était possible de définir un langage conservant les notations mathématiques et la démarche habituellement utilisée en analyse de données, tout en appliquant les résultats des recherches académiques. Un langage de ce genre a deux intérêts :

- faciliter la résolution de problèmes aux utilisateurs non-informaticiens.
- assurer en même temps une sécurité accrue, le compilateur opérant plus de contrôles.

Cette démarche a montré, par ailleurs, que l'implémentation d'un langage de ce type était réalisable, bien que peu rationnelle à l'heure actuelle.

CHAPITRE VII

CONCLUSION

Conclusion

Conclusion

VII.1. Résultats actuels.

Le compilateur et le système associé fonctionnent sur l'ordinateur de l'Institut Universitaire de Calcul Automatique de Lorraine. Par ailleurs, le compilateur a été aussi installé à l'Ecole des Mines de Fontainebleau. Sur ce site son utilisation n'est pas optimum, en raison de la difficulté de dissociation du compilateur et du système attenant, en effet, l'utilisation du système complet, qui gère les bibliothèques des utilisateurs, est plus aisée que celle du compilateur nu.

Plusieurs demandes d'installation du système dans différents centres de calcul nous ont été adressées, ce qui confirme que ce système répond à un besoin auprès des utilisateurs.

VII.2. Développement futur.

Ce système ne constitue qu'une étape importante dans la résolution des problèmes en analyse de données, il est possible, d'imaginer un prolongement futur, schématisé selon la figure 8, en deux blocs :

- tableau de données.
- ensembles de fonctions permettant de le manipuler. Ces ensembles sont découpés en trois parties indépendantes :
  - * une partie saisie du tableau de données.
  - * une partie manipulation du tableau de données.
  - * une partie édition de résultats.

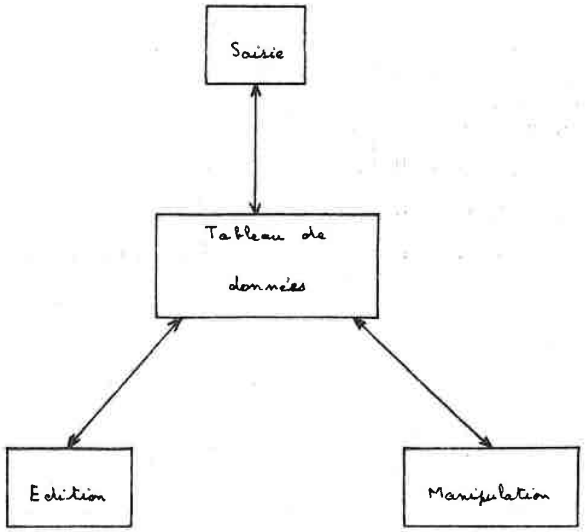

figure 8

Dans le système actuel, pour avoir accés au tableau de données, il faut passer par le langage GEOL, soit en écrivant des programmes, soit en utilisant des commandes cataloguées. Le système que nous décrivons permet d'accéder au tableau de données par des moyens différents.

VII.2.A. Le tableau de données.

Le tableau de données est tel que nous l'avons présenté dans le langage GEOL. Il contient donc les valeurs que l'on manipule et un descriptif, ce descriptif, plus complet que celui existant dans le langage GEOL, sert non seulement à effectuer les contrôles, mais aussi à aider et guider l'utilisateur dans sa démarche.

Avant tout travail, il faut que ce descriptif existe. Les différents types de tableaux de données sont connus à l'avance, ce qui permet de prévoir un automate qui aide l'utilisateur à créer ce descriptif. Ceci se déroule de manière conversationnelle.

VII.2.B. Partie saisie des données.

Dans cette partie deux opérations différentes sont prévues :

- stockage sur support physique des données. Un dialogue est instauré qui, à partir du descriptif du tableau de données, guide, oriente et contrôle les demandes de l'utilisateur. Par exemple, dans le cas d'un tableau irrégulier, il est demandé à l'utilisateur d'indiquer explicitement les coordonnées des points du domaine.
- interrogation et modification d'un tableau de données. Par l'intermédiaire d'un éditeur, l'utilisateur a la possibilté d'interroger le tableau de données. Certaines modifications dans le cas de valeurs erronées sont aussi permises, ce qui entraîne les remarques suivantes :
  * l'utilisateur ne peut pas modifier de composantes ensemblistes.
  * les composantes ensemblistes peuvent s'avérer fausses aprés modification de composantes fonctionnelles.

VII.2.C. Partie modification du tableau de données.

Pour les applications habituelles, l'utilisateur aura à sa disposition, d'une part, des fonctions de calcul prédéfinies, comme le package statistique que nous avons développé au chapitre VI, et d'autre part des fonctions d'accès aux composantes et aux unités statistiques du tableau de données. La combinaison de ces deux types de fonctions permettra de résoudre la plupart des problèmes. Il est envisagé un langage interprété se servant du descriptif du tableau de données pour guider et contrôler les opérations. Pour des applications ponctuelles, on aura recours à un langage compilé comme le langage GEOL.

VII.2.D. Partie édition de résultats.

Les éditions de résultats se résument en général à la manipulation de sous-programmes ayant un grand nombre de paramètres. Ici aussi le descriptif du tableau de données permet de fixer certains de ces paramètres. Les autres paramètres seront fixés au cours d'un dialogue. C'est ce moyen qui est utilisé dans °ROY-79§ pour faire un interface GEOL CARTOLAB.

Certains de ces objectifs ont été réalisés dans le système GEOL par l'intermédiaire de commandes spécifiques. Il existe ainsi une commande INTERP qui permet de transformer un réseau irrégulier en un réseau régulier. Cette commande comprend un dialogue qui permet de fixer les désirs de l'utilisateur, puis de réaliser la trsansformation.

VII.3. Conclusion.

Un tel système est utile, car il permet à tout utilisateur scientifique de se servir d'un ordinateur, en lui épargnant l'obligation d'apprendre des langages qui l'éloignent de ses problèmes.

En ce qui me concerne, ce travail m'aura permis de faire un lien entre la recherche théorique et la recherche appliquée.

ANNEXE

ANNEXE I : GRAMMAIRE DU LANGAGE GEOL

PROGRAMME ::= [SOUS_PROG]$^{+}$ [PROG_PRINCI] [SOUS_PROG]* ENDGEOL

[SOUS_PROG]* PROG_PRINCI [SOUS_PROG]* ENDGEOL

PROG_PRINCI ::= PROGRAM [(PARAM_SYS [,PARAM_SYS]*)];CORPS ENDPROG

SOUS_PROG ::= PROC idf (PARAM [,PARAM]*); CORPS ENDPROC

FUNCTION TYPE idf (PARAM [,PARAM]*); CORPS ENDFUNC

PARAM_SYS ::= TCORE = cste

IMPR = cste

LECT = cste

idf = cste

TYPE ::= REAL

INTEGER

LOGIC

ALPHA

PARAM ::= idf [OF idf]

CORPS ::= DECLARATION ; [DECLARATION ;]* INSTRUCTION ; [INSTRUCTION ;]*

DECLARATION ::= DEC_TAB_DON

DEC_COMPOSANTE

DEC_VARIABLE

DEC_LISTE

DETRUIRE

DEC_TAB_DON ::= AGE TYP_TAB idf [NCOMP (cste), OMEGA (CARACTERISTIQUE)]

TYP_TAB ::= DT

DT1

DT2

DT3

AGE ::= NEW

OLD

LOC

DUM

CARACTERISTIQUE ::= cste [, cste, cste [; cste, cste, cste]$^{0-2}$]

DEC_COMPOSANTE ::= AGE COMP OF idf TYPE1 IDENTIFICATEUR

IDENTIFICATEUR ::= idf [[cste]]

(idf [[cste]] [,idf [[cste]]])

TYPE1 ::= TYPE

SET

DEC_VARIABLE ::= TYPE2 DEC_VARIAB

TYPE2 ::= TYPE

STATUS

DEC_VARIAB ::= FONC idf

DEC_VARIAB2 = cste

DEC_VARIAB2 ::= VARIABLE

(VARIABLE [, VARIABLE]*)

VARIABLE ::= idf [[cste [, cste [, cste]]]]

DEC_LISTE ::= DEFLIST TYPE idf OF idf = (IDFLIST [, IDFLIST]*)

IDFLIST ::= idf [[cste [: cste]]]

VAR_LISTE ::= idf [[cste]]
(idf [[cste]] |, idf [[cste]] [*)

DETRUIRE ::= DELETE QUE_DETRUIRE

QUE_DETRUIRE ::= idf
idf [, idf]* OF idf

INSTRUCTION ::= [cste :] INSTRUCTIONS

INSTRUCTIONS ::= CONDITION
AFFECTATION
DEFINITION_SET
BOUCLE
ENTREE_SORTIE
APPEL
INIT_COMP
STOP
RETURN
CONTINUE

CONDITION ::= IF TEST THEN LIST_INST [ELSE LIST_INST] FI

```
LIST_INST        ::= INSTRUCTIONS ; { INSTRUCTIONS ; }*

AFFECTATION      ::= idf [ [ exp {, EXP {, EXP } } ] ] = EXP
                     AFFECTATION_SET

AFFECTATION_SET  ::= idf [ [ EXP ] ] OF idf = EXP_SET

EXP_SET          ::= VAR_SET1 [ UNION EXP_SET ]

VAR_SET1         ::= VAR_SET2 [ INTER VAR_SET1 ]

VAR_SET2         ::= idf
                     COMP idf
                     ( EXP_SET )
                     OMEGA
                     NULLSET

DEFINITION_SET   ::= DEFSET idf [ [ EXP ] ] FOR VAR_BOUCLE [ SUCHTHAT ( TEST ) ]

BOUCLE           ::= FOR VAR_BOUCLE [ WHILE ( TEST ) ] DO LIST_INST DONE
```

```
VAR_BOUCLE      ::= ALL idf
                    idf IN idf [ [ EXP ] ]
                    idf = EXP [, EXP ]*
                    idf = EXP TO EXP [ STEP EXP ]

ENTREE_SORTIE   ::= READ [ ( [ cste , ] END = cste ) ] QUE_LIRE [, QUE_LIRE ]*
                    WRITE [ ( cste ) QUE_ECRIRE [ , QUE_ECRIRE ]*

QUE_LIRE        ::= idf [ [ EXP [, EXP [, EXP] ] ] ]
                    idf [ [ EXP ] ] ( idf )
                    TITLE OF idf [ [ EXP ] ] [ OF idf ]

QUE_ECRIRE      ::= idf [ [ EXP [, EXP [, EXP ] ] ] ]
                    idf [ [ EXP ] ] ( idf )
                    TITLE OF idf [ [ EXP ] ] [ OF idf ]
                    < chaine de caracteres >
                    NEWP            --saut de page--
                    cste SPACES
                    cste NEWL       --saut de ligne--

APPEL           ::= CALL idf ( EXP [, EXP ]* )

INIT_COMP       ::= ALL idf [ [ EXP ] ] OF idf = cste
```

```
EXP          ::= -- la definition des expressions est celle generalement
                  adoptee dans les langages avec la definition de PRIMAIRE
                  suivante --

PRIMAIRE     ::= ( EXP )
                 idf [ [ EXP [, EXP [, EXP ] ] ] ]
                 idf [ [ EXP ] ] ( idf )
                 FONCT_INCOR ( EXP )

TEST         ::= -- la definition des tests est celle generalement adoptee
                  dans les langages
```

ANNEXE II : FONCTIONS INCORPOREES

- Calcul de modalités.

calcul de modalites par valeur :

MODVAL (exp;exp_1 ,exp_2 ,..,exp_3) = i si exp=exp_i

n+1 sinon

calcul de modalités par intervalle :

MODINT (exp;exp_1 ,exp_2 ,..,exp_3) = i si exp<exp_i

i si exp_{i-1} < exp < exp_i

n+1 si exp > exp_n

- Conversion d'unités statistiques en coordonnées.

Soit idf une variable déclarée STATUS :

XOF (idf) donne la coordonnée réelle de idf sur l'axe des X

YOF (idf) donne la coordonnée réelle de idf sur l'axe des Y

ZOF (idf) donne la coordonnée réelle de idf sur l'axe des Z

IXOF (idf) donne la coordonnée entière de idf sur l'axe des X

IYOF (idf) donne la coordonnée entière de idf sur l'axe des Y

IZOF (idf) donne la coordonnée entière de idf sur l'axe des Z

- Conversion de coordonnées en unités statistiques.

Soit idf une variable déclarée STATUS :

USOF (idf , expl °, exp2 °, exp3 § §) donne l'unité statistique en fonction des coordonnées expl, exp2 et exp3.

- Fonction indicatrice d'une composante de type SET.

Soient idf1 un tableau de données et idf2 une composante de type SET de ce tableau de données :

INOUTS (idf1 , idf2) = 1 si idf1 appartient à idf2

o sinon

- Caractéristique d'une composante de type SET :

Soient idf1 un tableau de données et idf2 une composante de type SET de ce tableau :

CARD (idf2 OF idf1) donne le nombre d'unités statistiques de idf2.

CARDX (idf2 OF idf1) donne le nombre d'unités statistiques de idf2 sur l'axe des X.

il existe de même CARDY et CARDZ.

XMIN (dif2 OF idf1) donne la plus petite coordonnée sur l'axe des X des unités statistiques de idf2.

XMAX (idf2 OF idf1) donne la plus grande coordonnée sur l'axe des X des unités statistiques de idf2.

il existe de même YMIN, YMAX, ZMIN et ZMAX.

- Caractéristiques du quadrillage pour les tableaux réguliers.

Soient p_x ,p_y et p_z le pas sur l'axe des X, Y et Z et idf un tableau de données régulier :

PIXELX (idf) donne la valeur p_x .

PIXELY (idf) donne la valeur p_y .

PIXELZ (idf) donne la valeur p_z .

PIXEL (idf) donne p_x si idf est de type DT1, $p_x * p_y$ si idf est de type DT2 ou $p_x * p_y * p_z$ si idf est de type DT3.

Traduction des déclarations.

```
OLD DT2 MINE ;
```

déclaration d'un tableau de données en GEOL

```
CALL $XSDT(NERR,LOCT,IW,IW,USCORE,CORE,NCORE,
+           NBTDEC,IDTP,1,6HMINE  ,2
+          )
```

code généré dans le programme principal FORTRAN

```
REAL    MINE  (1)
INTEGER IX$101(1)
LOGICAL IX$102(1)
```

déclarations dans le sous-programme principal FORTRAN

```
OLD COMP OF MINE   REAL(FER,EFER,PUISS,CAO,SI02) ;
```

déclarations de composantes en GEOL

```
IX$105=USCORE+1
IW(IX$105)=        0
IX$106=USCORE+3
CALL $XSDC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
+          1,6HMINE  ,6HFER   ,2,IW(IX$105),IW(IX$106))

IX$107=USCORE+1
IW(IX$107)=        0
IX$108=USCORE+3
CALL $XSDC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
+          1,6HMINE  ,6HEFER  ,2,IW(IX$107),IW(IX$108))

IX$109=USCORE+1
IW(IX$109)=        0
IX$110=USCORE+3
CALL $XSDC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
+          1,6HMINE  ,6HPUISS ,2,IW(IX$109),IW(IX$110))

IX$111=USCORE+1
IW(IX$111)=        0
IX$112=USCORE+3
CALL $XSDC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
+          1,6HMINE  ,6HCAO   ,2,IW(IX$111),IW(IX$112))
```

Code généré dans le programme principal FORTRAN

```
INTEGER IX$105(2) , IX$106(1)
INTEGER IX$107(2) , IX$108(1)
INTEGER IX$109(2) , IX$110(1)
INTEGER IX$111(2) , IX$112(1)
```

déclarations dans le sous programme principal FORTRAN

Traduction des sous-programmes.

```
PROC MINEX( QUALIT OF MINE ,
            PUISS  OF MINE ,
            RECUP  OF MINE ,
            ZONEX  OF MINE ,
            WREAL  OF MINE ,
            WSET   OF MINE ,
                        QUALIT OF PLANEX ,
                        SURF   OF PLANEX ,
                        VOLUME OF PLANEX ,
                        FRONT  OF PLANEX ) ;
```

déclaration d'une procédure en GEOL

```
 SUBROUTINE MINEX ( IXS101 , IXS102 , IXS104 , IXS105 , IXS106 , IX
+S107 , IXS108 , IXS109 , IXS110 , IXS111 , IXS112 , IXS113 , IXS11
+4 , IXS115 , IXS117 , IXS118 , IXS119 , IXS120 , IXS121 , IXS122 ,
+ MINE , IXS103 , SXS101 , PLANEX , IXS116 , SXS102 )
```

déclaration de la procédure en FORTRAN

```
CALL MINEX( QUALIT OF MINE ,
            PUISS  OF MINE ,
            RECUP  OF MINE ,
            ZONEX  OF MINE ,
            WREAL  OF MINE ,
            WSET   OF MINE ,
                        QUALIT OF PLANEX ,
                        SURF   OF PLANEX ,
                        VOLUME OF PLANEX ,
                        FRONT  OF PLANEX ) ;
```

appel d'une procédure en GEOL

```
 CALL MINEX ( IXS123 , IXS124 , IXS109 , IXS110 , IXS115 , IXS116 ,
+ IXS117 , IXS118 , IXS119 , IXS120 , IXS121 , IXS122 , IXS141 , IX
+S142 , IXS135 , IXS136 , IXS137 , IXS138 , IXS139 , IXS140 , MINE
+, IXS101 , IXS102 , PLANEX , IXS103 , IXS104 )
```

appel correspondant en FORTRAN

Traduction d'une composante de type SET

Instruction de définition d'un SET en GEOL.

```
DEFSET ZONEX FOR ALL XY
             SUCHTHAT(  INOUTF(FRONP,XOF(XY),YOF(XY)) .GT. 0 );
```

Instructions FORTRAN équivalentes.

```
C****
C****DEBUT DE DEFINITION DU SET ZONEX  PAR UNE INSTRUCTION 'DEFSET'
C****
C
      I$EXPR=1
      N$XMAX=0
      N$XMIN= 10000000
      N$YMAX=0
      N$YMIN= 10000000
      I$NX  =IX$103(5)
      I$ADR =0
      N$CARD=0
      I$XYZ = 10000000
      N$US=IX$103(3)
      XY    =N$US+1
      DO 90009 I$1=1,N$US
      XY    =XY    -1
      IX$124=XY    +(XY    +31)/32*IX$103(13)+32
      I$2=IX$109(I$EXPR)+IX$124
      IX$103(I$2)=-1
      $X$103=$X$CUS( 1,XY    ,MINE  ,IX$103)
      $X$104=$X$CUS( 2,XY    ,MINE  ,IX$103)
      IF(.NOT.( INOUTF ( FRONP (1,1), $X$103 , $X$104 , IX$127 (1,1), IX
     +$127 (1, 1) , IX$127 (1, 2) ) .GT. 0
     +)) GO TO 90010
      IX$103(I$2   )=I$ADR
      J$=(XY    -1)/I$NX
      I$Y=J$+1
      I$X=XY    -J$*I$NX
      IF(I$Y.GE.N$YMIN) GO TO 90011
      N$YMIN=I$Y
      GO TO 90012
90011 IF(I$Y.LE.N$YMAX) GO TO 90012
      N$YMAX=I$Y
90012 IF(I$X.GE.N$XMIN) GO TO 90013
      N$XMIN=I$X
      GO TO 90014
90013 IF(I$X.LE.N$XMAX) GO TO 90014
      N$XMAX=I$X
90014 CONTINUE
      I$ADR=XY
      N$CARD=N$CARD+1
      GO TOS0009
90010 IX$103(I$2   )=-1
C
90009 CONTINUE
C
      I$EXPR=IX$109(I$EXPR)
      I$=I$EXPR+24
      IX$103(I$)=I$ADR
      I$ADR=N$XMIN
     +    +(N$YMIN-1)*IX$103(5)
      I$=I$EXPR+27
      IX$103(I$)=I$ADR
      I$=I$EXPR+28
      N$XMIN=N$XMAX-N$XMIN+1
      IX$103(I$)=N$XMIN
      I$=I$EXPR+29
      N$YMIN=N$YMAX-N$YMIN+1
      IX$103(I$)=N$YMIN
      I$=I$EXPR+26
      IX$103(I$)=N$CARD
      IF(N$CARD.NE.0) GO TO 90016
      I$=I$EXPR+23
      DO 90015 J$=1,7
      I$=I$+1
90015 IX$103(I$)=0
90016 CONTINUE
C
C****
C**** FIN DE DEFINITION DU SET ZONEX
C****
```

```
      PROG(TCORE=60000,MINE=1,PLANEX=2) ;
$
      OLD DT2 MINE ;
      OLD COMP OF MINE  REAL(FER,EFER,PUISS,CAO,SIO2) ,
                        REAL(RECUP),SET(ZONEX)        ;
      LOC COMP OF MINE  REAL(WREAL[2],WSET) ;
$
      OLD DT PLANEX ;
      OLD COMP OF PLANEX REAL(FER,EFER,PUISS,CAO,SIO2) ,
                         REAL(SURF,VOLUME)             ,
                         REAL(FRONT[5])                ;
$
      DEFLIST REAL QUALIT OF MINE   * (FER,EFER,PUISS,CAO,SIO2) ;
      DEFLIST REAL QUALIT OF PLANEX * (FER,EFER,PUISS,CAO,SIO2) ;
$
      CALL MINEX ( QUALIT OF MINE ,
                   PUISS  OF MINE ,
                   RECUP  OF MINE ,
                   ZONEX  OF MINE ,
                   WREAL  OF MINE ,
                   WSET   OF MINE ,
                                  QUALIT OF PLANEX ,
                                  SURF   OF PLANEX ,
                                  VOLUME OF PLANEX ,
                                  FRONT  OF PLANEX ) ;
$
      STOP ; ENDPROG ;
00027.000I
```

```
      PROC MINEX( QUALIT OF MINE ,
                  PUISS  OF MINE ,
                  RECUP  OF MINE ,
                  ZONEX  OF MINE ,
                  WREAL  OF MINE ,
                  WSET   OF MINE ,
                                  QUALIT OF PLANEX ,
                                  SURF   OF PLANEX ,
                                  VOLUME OF PLANEX ,
                                  FRONT  OF PLANEX )  ;
$
$*****************************************************************
$*****************************************************************
$
$
$      DECLARATIONS REMANENTES
$
      DUM DT2 MINE ;
      DUM COMP OF MINE  REAL(QUALIT[5],PUISS,RECUP,WREAL[2]),
                        SET (ZONEX,WSET) ;
      STATUS XY OF MINE ;
$
$
      DUM DT PLANEX ;
      DUM COMP OF PLANEX REAL(QUALIT[5],SURF,VOLUME,FRONT[8]) ;
      STATUS PAN OF PLANEX ;
$
$
$      DECLARATIONS LOCALES
$
      REAL FRONP[2,100] ; INTEGER (I,NQUALI,CARDP,NFRON) ;
      REAL TAUX;
      REAL FUNC (SOMFXY) ;
      INTEGER FUNC INOUTF;
$
$
$*****************************************************************
$
$
                     $***************$
                     $               $
                     $  PROLOGUE     $
                     $               $
                     $***************$
$
$
$....POSITIONNEMENT DU PLAN DE LA MINE SUR LA TABLETTE
$
      CALL INITAB       ;
$
$....INITIALISATION DE LA ZONE NON ENCORE EXPLOITEE
$
      NQUALI=DIM(QUALIT OF MINE , 1) ;
      NFRON =DIM(FRONT  OF PLANEX , 1) ;
 IF ANSWER(<VOULEZ VOUS (RE)DEFINIR LA ZONE NON ENCORE EXPLOITEE))
00086.000I
```

```
 THEN ALL VOLUME OF PLANEX = -1. ;
      ALL SURF   OF PLANEX = -1. ;
      FOR I=1 TO NQUALI DO ALL QUALITE[I] OF PLANEX = -1. ; DONE;
      FOR I=1 TO NFRON  DO ALL FRONT  [I] OF PLANEX = -1. ; DONE;
     WRITE (DESIGNEZ SUR LA TABLE A DIGITALISER LES POINTS) ,
           1 NEWLINE ,
           (FRONTIERE DELIMITANT LA ZONE NON ENCORE EXPLOITEE) ;
     CALL LECTAB(FRONP,0) ;
     DEFSET ZONEX FOR ALL XY
                  SUCHTHAT(  INOUTF(FRONP,XOF(XY),YOF(XY)) .GT. 0 );
  FI ;
$
$....INITIALISATION DU TAUX DE RECUPERATION
$
  IF ANSWER((VOULEZ VOUS (RE)INITIALISER LES TAUX DE RECUPERATION))
     THEN
     ALL RECUP OF MINE = 1. ;
     'ETI1':
     WRITE (DESIGNEZ SUR LA TABLE A DIGITALISER LES POINTS FRONTIERE)
           , 1 NEWLINE ,
           (DELIMITANT UNE ZONE OU L'ON VEUT AFFECTER UN TAUX DE)
           , 1 NEWLINE ,
           (RECUPERATION CONSTANT) ;
     CALL LECTAB(FRONP,0) ;
     WRITE (QUEL TAUX DE RECUPERATION VOULEZ VOUS AFFECTER ?) ;
     READ  TAUX ;
     DEFSET WSET FOR ALL XY
                 SUCHTHAT(  INOUTF(FRONP,XOF(XY),YOF(XY)) .GT. 0 ) ;
     FOR XY IN WSET DO RECUP(XY)=TAUX ; DONE ;
     IF ANSWER((VOULEZ VOUS FIXER LE TAUX DE RECUP. D'UNE NOUV. ZONE))
        THEN GOTO 'ETI1' ; FI ;
 FI ;
$
$
                    $$$$$$$$$$$$$$$
                    $             $
                    $  DIALOGUE   $
                    $             $
                    $$$$$$$$$$$$$$$
$
$
$....INITIALISATIONS
$
      NFRON = NFRON/2  ;
      STYLEDIT(  I(3)  ) ;
$
      FOR ALL XY DO WREAL[1](XY) = RECUP(XY) * PUISS(XY) ;
                    WREAL[2](XY) = 1. ; DONE ;
      FOR ALL PAN WHILE(FRONT[1](PAN).GT.0.) DO CONTINUE ; DONE ;
$
$....LECTURE ET CONTROLE D'UN PANNEAU MINIER
$
     'ETI2' :
      WRITE (DESIGNEZ SUR LA TABLE A DIGITALISER LES POINTS)
           , 1 NEWLINE ,
00141.000$
```

```
          <FRONTIERE DELIMITANT UN NOUVEAU PANNEAU MINIER> ;
      CALL LECTAB(FRONP,0) ;
      IF DIM(FRONP,2).GT.NFRON THEN
         WRITE <UN PANNEAU MINIER NE PEUT AVOIR PLUS DE> ,
               NFRON , <POINTS FRONTIERE> ;
         GOTO 'ETI2' ;
      FI ;
$
      DEFSET WSET FOR ALL XY
                  SUCHTHAT(  INOUTF(FRONP,XOF(XY),YOF(XY)) .GT. 0 );
      CARDP = CARD(WSET OF MINE) ;
      WSET OF MINE  = WSET  .I. ZONEX  ;
      IF CARDP.NE.CARD(WSET OF MINE)
         THEN WRITE <ERREUR: LE PANNEAU DESIGNE N'EST PAS TOUT ENTIER>
                    , 1 NEWLINE ,
                    <CONTENU DANS LA ZONE NON ENCORE EXPLOITEE> ;
              GOTO 'ETI3' ;
      FI ;
$
$....INTEGRATION SUR LE PANNEAU DESIGNE
$
      VOLUME(PAN) = SOMFXY(WREAL[1] OF MINE , FRONP) ;
      SURF  (PAN) = SOMFXY(WREAL[2] OF MINE , FRONP) ;
  FOR I=1 TO NQUALI DO
      QUALIT[I](PAN) = SOMFXY(QUALIT[I] OF MINE,FRONP) / SURF(PAN) ;
  DONE ;
$
$....EDITION DES VALEURS ASSOCIEES AU PANNEAU
$
      I = DIM(FRONP,2) ;
      WRITE  NEWPAGE ,
            <CARACTERISTIQUES GEOMETRIQUES DU PANNEAU MINIER>,
            1 NEWLINE ,
            <.....>,
            3 NEWLINE ,
            <NOMBRE DE POINTS FRONTIERE : >,I,1 NEWLINE,
            <SURFACE DU PANNEAU         : >,SURF(PAN),1 NEWLINE,
            <VOLUME DU PANNEAU          : >,VOLUME(PAN),3NEWLINE,
            <CARACTERISTIQUES DE QUALITE  DU  PANNEAU MINIER>,
            1 NEWLINE,
            <.....>,
            3 NEWLINE;
      FOR I=1 TO NQUALI DO
            WRITE <VALEUR MOYENNE DE>,
                  NAME OF QUALIT[I] OF MINE,QUALIT[I](PAN);
      DONE ;
$
$....PRISE DE DECISION D'EXPLOITATION
$
  IF ANSWER(<VOULEZ VOUS EXPLOITER CE PANNEAU MINIER>)
     THEN
      IF PAN.EQ.CARD(OMEGA  - PLANEX)
         THEN
          WRITE <LE NOMBRE MAXIMUM DE PANNEAUX MINIERS>,
                <AUTORISE EST ATTEINT> ; GOTO 'ETI3' ;
      FI ;
00197.0001
```

```
      ZONEX OF MINE = ZONEX - USET ;
      I = PAN ;
      CALL STOFRT(2,FRONP,FRONT OF PLANEX,I) ;
      IF PAN.LT.CARD(OMEGA OF PLANEX) THEN PAN = PAN+1 ;
      FI ;
  FI ;
@
  'ETI3' :
  IF ANSWER(VOULEZ VOUS INTRODUIRE UN NOUVEAU PANNEAU))
     THEN  GOTO 'ETI2'  ;
  FI ;
@
@
                    @@@@@@@@@@@@@@@
                    @             @
                    @  EPILOGUE   @
                    @             @
                    @@@@@@@@@@@@@@@
@
@
  'ETI4' :
@
@....RECAPITULATION DES DIFFERENTS PANNEAUX MINIERS
@    CONSTITUANT LE PLAN D'EXPLOITATION
@
      URITE  NEUPAGE ,
             (TABLEAU RECAPITULATIF DES PANNEAUX MINIERS ),
             1 NEULINE ,
             (---------------------------------------- ),
             3 NEULINE ;
  FOR ALL PAN WHILE(FRONTE1](PAN).GT.0.) DO
      I = (FRONTE1](PAN)+0.5)/2.  ;
      URITE 3 NEULINE ,
             (PANNEAU NUMERO            : ),PAN,
             1 NEULINE ,
             (NOMBRE DE PONTS FRONTIERE: ),I,
             1 NEULINE ,
             (SURFACE DU PANNEAU        : ),SURF(PAN),
             1 NEULINE ,
             (VOLUME DU PANNEAU         : ),VOLUME(PAN),
             2 NEULINE ;
      FOR I=1 TO NQUALI DO
      URITE (VALEUR MOYENNE DE),
             NAME OF QUALITE[I] OF MINE , QUALITE[I](PAN) ;
      DONE ;
  DONE ;
@
  RETURN ; ENDPROC ;
@OF HIT
@
```

```
      INTEGER   CORE,USCORE
C
C*************************************************************************
C********************* PROGRAMME-PRINCIPAL-GENERE ************************
C*************************************************************************
C
C
      INTEGER   IW
      DIMENSION LOCT(6,10),IDTP(3,10)
      COMMON    IW(    60003)
C
      COMMON /SXSIOS/ IMPRS,LECTS
C
      DATA CORE,USCORE,NCORE /    60000, 0, 0/
      DATA NERR /0/
      DATA NBTDEC /  2 /
C
      DATA IDTP(1, 1),IDTP(2, 1),IDTP(3, 1)/'MINE','    ',      1/
      DATA IDTP(1, 2),IDTP(2, 2),IDTP(3, 2)/'PLAN','EX  ',      2/
C
C
C*************************************************************************
C
C   INITIALISATIONS ET DECLARATIONS
C
      IMPRS=      108
      LECTS=      105
C
      IX$101=USCORE+1
      CALL SXSDT(NERR,LOCT,IW,IW,USCORE,CORE,NCORE,
     +           NBTDEC,IDTP,1,6HMINE  ,2
     +          )
C
      IX$105=USCORE+1
      IW(IX$105)=       0
      IX$106=USCORE+3
      CALL SXSDC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
     +           1,6HMINE  ,6HFER   ,2,IW(IX$105),IW(IX$106))
C
      IX$107=USCORE+1
      IW(IX$107)=       0
      IX$108=USCORE+3
      CALL SXSDC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
     +           1,6HMINE  ,6HEFER  ,2,IW(IX$107),IW(IX$108))
C
      IX$109=USCORE+1
      IW(IX$109)=       0
      IX$110=USCORE+3
      CALL SXSDC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
     +           1,6HMINE  ,6HPUISS ,2,IW(IX$109),IW(IX$110))
C
      IX$111=USCORE+1
      IW(IX$111)=       0
      IX$112=USCORE+3
      CALL SXSDC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
$$$$$$
```

```
     +           1,6HMINE  ,6HCAO   ,2,IW(IX$111),IW(IX$112))
C
      IX$113=USCORE+1
      IW(IX$113)=        6
      IX$114=USCORE+3
      CALL $X$DC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
     +           1,6HMINE  ,6HSIO2  ,2,IW(IX$113),IW(IX$114))
C
      IX$115=USCORE+1
      IW(IX$115)=        8
      IX$116=USCORE+3
      CALL $X$DC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
     +           1,6HMINE  ,6HRECUP ,2,IW(IX$115),IW(IX$116))
C
      IX$117=USCORE+1
      IW(IX$117)=        0
      IX$118=USCORE+3
      CALL $X$DC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
     +           1,6HMINE  ,6HZONEX ,5,IW(IX$117),IW(IX$118))
C
      IX$119=USCORE+1
      IW(IX$119)=        2
      IX$120=USCORE+3
      CALL $X$DC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
     +           3,6HMINE  ,6HWREAL ,2,IW(IX$119),IW(IX$120))
C
      IX$121=USCORE+1
      IW(IX$121)=        0
      IX$122=USCORE+3
      CALL $X$DC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
     +           3,6HMINE  ,6HWSET  ,2,IW(IX$121),IW(IX$122))
      IX$123=USCORE+1
      IX$124=USCORE+3
      IW(IX$123  )=        5
      IW(IX$123+1)=        5
      USCORE=USCORE+2+        5
C
C
      IX$103=USCORE+1
      CALL $X$DT(NERR,LOCT,IW,IW,USCORE,CORE,NCORE,
     +           NBTDEC,IDTP,1,6HPLANEX,0
     +          )
C
      IX$125=USCORE+1
      IW(IX$125)=        0
      IX$126=USCORE+3
      CALL $X$DC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
     +           1,6HPLANEX,6HFER   ,2,IW(IX$125),IW(IX$126))
C
      IX$127=USCORE+1
      IW(IX$127)=        0
      IX$128=USCORE+3
      CALL $X$DC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
     +           1,6HPLANEX,6HEFER  ,2,IW(IX$127),IW(IX$128))
C
      IX$129=USCORE+1
@@1118
```

```
      IW(IX$129)=        0
      IX$130=USCORE+3
      CALL $X$DC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
     +           1,6HPLANEX,6HPUISS ,2,IW(IX$129),IW(IX$130))
C
      IX$131=USCORE+1
      IW(IX$131)=        0
      IX$132=USCORE+3
      CALL $X$DC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
     +           1,6HPLANEX,6HCAO   ,2,IW(IX$131),IW(IX$132))
C
      IX$133=USCORE+1
      IW(IX$133)=        0
      IX$134=USCORE+3
      CALL $X$DC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
     +           1,6HPLANEX,6HSIO2  ,2,IW(IX$133),IW(IX$134))
C
      IX$135=USCORE+1
      IW(IX$135)=        0
      IX$136=USCORE+3
      CALL $X$DC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
     +           1,6HPLANEX,6HSURF  ,2,IW(IX$135),IW(IX$136))
C
      IX$137=USCORE+1
      IW(IX$137)=        0
      IX$138=USCORE+3
      CALL $X$DC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
     +           1,6HPLANEX,6HVOLUME,2,IW(IX$137),IW(IX$138))
C
      IX$139=USCORE+1
      IW(IX$139)=        1
      IX$140=USCORE+3
      CALL $X$DC(NERR,LOCT,IW,USCORE,CORE,NCORE,NBTDEC,
     +           1,6HPLANEX,6HFRONT ,2,IW(IX$139),IW(IX$140))
      IX$141=USCORE+1
      IX$142=USCORE+3
      IW(IX$141  )=        5
      IW(IX$141+1)=        5
      USCORE=USCORE+2+        5
C
C
C*******!*********************************************************************
C
C   CONTROLE DU CORE
C
      WRITE(IMPR$,1) CORE,NCORE
    1 FORMAT(//,
     +1X,52H*** GEOL-EXECUTION : CORE USED BY DATA(S) TABLE(S) :,/,
     +          1X,27H    ALLOCATED CORE : TCORE=,I9,/,
     +          1X,27H    NECESSARY CORE : TCORE=,I9,//)
      IF(NCORE.LE.CORE) GO TO 3
      WRITE(IMPR$,2)
    2 FORMAT(//,1X,41H*** GEOL-RUN-TIME ERROR : MEMORY OVERFLOW )
      CALL $X$ERR(0,0,0)
    3 IF(NERR.NE.0) CALL $X$ERR(0,0,0)
C
@@167$
```

```
C   STOCKAGE SUR MEMOIRE SECONDAIRE
C
      CALL SX$SAV(IU,LOCT,NBTDEC)
C
C   APPEL DU SOUS-PROGRAMME-PRINCIPAL-GENERE
C
      CALL SX$$$$(S,IU(IX$101 ) ,IU(IX$101 ) ,IU(IX$101 ) ,IU(IX$105 ) ,
     +IU(IX$106 ) ,IU(IX$107 ) ,IU(IX$108 ) ,IU(IX$109 ) ,IU(IX$110 ) ,I
     +U(IX$111 ) ,IU(IX$112 ) ,IU(IX$113 ) ,IU(IX$114 ) ,IU(IX$115 ) ,IU
     +(IX$116 ) ,IU(IX$117 ) ,IU(IX$118 ) ,IU(IX$119 ) ,IU(IX$120 ) ,IU(
     +IX$121 ) ,IU(IX$122 ) ,IU(IX$123 ) ,IU(IX$124 ) ,IU(IX$103 ) ,IU(I
     +X$103 ) ,IU(IX$103 ) ,IU(IX$125 ) ,IU(IX$126 ) ,IU(IX$127 ) ,IU(IX
     +$123 ) ,IU(IX$129 ) ,IU(IX$130 ) ,IU(IX$131 ) ,IU(IX$132 ) ,IU(IX$
     +133 ) ,IU(IX$134 ) ,IU(IX$135 ) ,IU(IX$136 ) ,IU(IX$137 ) ,IU(IX$1
     +38 ) ,IU(IX$139 ) ,IU(IX$140 ) ,IU(IX$141 ) ,IU(IX$142 ) )
      STOP
      END
C
C
C*************************************************************************
C***.*************** SOUS-PROGRAMME-PRINCIPAL-GENERE ********************
C***.*************************************************************************
C
C
      SUBROUTINE SX$$$$(S, MINE , IX$101 , IX$102 , IX$105 , IX$106 , IX
     +$107 , IX$108 , IX$109 , IX$110 , IX$111 , IX$112 , IX$113 , IX$11
     +4 , IX$115 , IX$116 , IX$117 , IX$118 , IX$119 , IX$120 , IX$121 ,
     + IX$122 , IX$123 , IX$124 , PLANEX , IX$103 , IX$104 , IX$125 , IX
     +$126 , IX$127 , IX$128 , IX$129 , IX$130 , IX$131 , IX$132 , IX$13
     +3 , IX$134 , IX$135 , IX$136 , IX$137 , IX$138 , IX$139 , IX$140 ,
     + IX$141 , IX$142 )
                            REAL    MINE  (1)
                            INTEGER IX$101(1)
                            LOGICAL IX$102(1)
                          INTEGER IX$105(2) , IX$106(1)
                          INTEGER IX$107(2) , IX$108(1)
                          INTEGER IX$109(2) , IX$110(1)
                          INTEGER IX$111(2) , IX$112(1)
                          INTEGER IX$113(2) , IX$114(1)
                          INTEGER IX$115(2) , IX$116(1)
                          INTEGER IX$117(2) , IX$118(1)
                          INTEGER IX$119(2) , IX$120(1)
                          INTEGER IX$121(2) , IX$122(1)
                          INTEGER IX$123(2) , IX$124(1)
                            REAL    PLANEX(1)
                            INTEGER IX$103(1)
                            LOGICAL IX$104(1)
                          INTEGER IX$125(2) , IX$126(1)
                          INTEGER IX$127(2) , IX$128(1)
                          INTEGER IX$129(2) , IX$130(1)
                          INTEGER IX$131(2) , IX$132(1)
                          INTEGER IX$133(2) , IX$134(1)
                          INTEGER IX$135(2) , IX$136(1)
                          INTEGER IX$137(2) , IX$138(1)
                          INTEGER IX$139(2) , IX$140(1)
                          INTEGER IX$141(2) , IX$142(1)
```

```
C*********************************************************************
C
      REAL PRNAM$(2)
      LOGICAL END$,$X$ANS
      COMMON /$X$IOS/IMPR$,LECT$
      DATA PRNAM$/'$$$$','$$$$'/
C
C*********************************************************************
C
      IX$124(     1)=IX$106(1)
      IX$124(     2)=IX$108(1)
      IX$124(     3)=IX$110(1)
      IX$124(     4)=IX$112(1)
      IX$124(     5)=IX$114(1)
C
      IX$123(2)=          5
      IF(IX$123(1).LT.IX$123(2))
     +CALL $X$ERR(PRNAM$,    16,   5)
      IX$142(     1)=IX$126(1)
      IX$142(     2)=IX$128(1)
      IX$142(     3)=IX$130(1)
      IX$142(     4)=IX$132(1)
      IX$142(     5)=IX$134(1)
C
      IX$141(2)=          5
      IF(IX$141(1).LT.IX$141(2))
     +CALL $X$ERR(PRNAM$,    17,   5)
      CALL MINEX ( IX$123 , IX$124 , IX$109 , IX$110 , IX$115 , IX$116 ,
     + IX$117 , IX$118 , IX$119 , IX$120 , IX$121 , IX$122 , IX$141 , IX
     +$142 , IX$135 , IX$136 , IX$137 , IX$138 , IX$139 , IX$140 , MINE
     +, IX$101 , IX$102 . PLANEX , IX$103   IX$104 )
      CALL $X$FIN
      END
002578
```

```
      SUBROUTINE MINEX ( IX$101 , IX$102 , IX$104 , IX$105 , IX$106 , IX
     +$107 , IX$108 , IX$109 , IX$110 , IX$111 , IX$112 , IX$113 , IX$11
     +4 , IX$115 , IX$117 , IX$118 , IX$119 , IX$120 , IX$121 , IX$122 ,
     + MINE , IX$103 , $X$101 , PLANEX , IX$116 , $X$102 )
                              REAL    MINE  (1)
                              INTEGER IX$103(1)
                              LOGICAL $X$101(1)
                            INTEGER IX$101(2) , IX$102(1)
                            INTEGER IX$104(2) , IX$105(1)
                            INTEGER IX$106(2) , IX$107(1)
                            INTEGER IX$110(2) , IX$111(1)
                            INTEGER IX$108(2) , IX$109(1)
                            INTEGER IX$112(2) , IX$113(1)
           INTEGER XY     ,IX$124
                              REAL    PLANEX(1)
                              INTEGER IX$116(1)
                              LOGICAL $X$102(1)
                            INTEGER IX$114(2) , IX$115(1)
                            INTEGER IX$117(2) , IX$118(1)
                            INTEGER IX$119(2) , IX$120(1)
                            INTEGER IX$121(2) , IX$122(1)
           INTEGER PAN    ,IX$126
                REAL    FRONP (        2,100)
                INTEGER IX$127(2,3)
      INTEGER I
      INTEGER NQUALI
      INTEGER CARDP
      INTEGER NFRON
      REAL    TAUX
      REAL SOMFXY
      INTEGER INOUTF
C
C*************************************************************************
C
      REAL PRNAM$(2)
      LOGICAL END$,$X$ANS
      COMMON /$X$IO$/IMPR$,LECT$
      DATA PRNAM$/'MINE','X   '/
C
C*************************************************************************
C
                  IX$127(1,1)=        2
                  IX$127(2,1)=        2
                  IX$127(1,2)=      100
                  IX$127(2,2)=      100
                  IX$127(1,3)=        0
                  IX$127(2,3)=        0
C
C
C*************************************************************************
C
      CALL INITAB ( TAUX )
      NQUALI = IX$101(2)
      NFRON = IX$121(2)
      IF(.NOT.( $X$ANS( 13, 52HVOULEZ VOUS (RE)DEFINIR LA ZONE NON ENCOR
     +E EXPLOITEE,IMPR$,LECT$)
 000313
```

```
     +1) GO TO 90001
      I$=1
      I$4=IX$116(3)
      DO 90002 I$3=1,I$4
      I$3B  =I$3   +(I$3   +31)/32*IX$116(13)+32
      I$5=IX$120(I$)+I$3B
      PLANEX(I$5)=-  .100000000E+01
90002 CONTINUE
      I$=1
      I$4=IX$116(3)
      DO 90003 I$3=1,I$4
      I$3B  =I$3   +(I$3   +31)/32*IX$116(13)+32
      I$5=IX$118(I$)+I$3B
      PLANEX(I$5)=-  .100000000E+01
90003 CONTINUE
      DO 90004 I      =            1,NQUALI      , 1
      I$1=I
      IF(I$1.LT.1)
     +CALL $X$ERR(PRNAM$,   58,  4)
      I$=I$1
      IF(I$2.GT.IX$114(3))
     +CALL $X$ERR(PRNAM$,   58,  3)
      I$4=IX$116(3)
      DO 90005 I$3=1,I$4
      I$3B  =I$3   +(I$3   +31)/32*IX$116(13)+32
      I$5=IX$115(I$)+I$3B
      PLANEX(I$5)=-  .100000000E+01
90005 CONTINUE
90004 CONTINUE
      DO 90006 I      =            1,NFRON       , 1
      I$1=I
      IF(I$1.LT.1)
     +CALL $X$ERR(PRNAM$,   59,  4)
      I$=I$1
      IF(I$2.GT.IX$121(3))
     +CALL $X$ERR(PRNAM$,   59,  3)
      I$4=IX$119(3)
      DO 90007 I$3=1,I$4
      I$3B  =I$3   +(I$3   +31)/32*IX$116(13)+32
      I$5=IX$122(I$)+I$3B
      PLANEX(I$5)=-  .100000000E+01
90007 CONTINUE
90006 CONTINUE
      WRITE(IMPR$ , 90008)
90008 FORMAT( 1X, 46HDESIGNEZ SUR LA TABLE A DIGITALISER LES POINTS , /
     +, 1X, 49HFRONTIERE DELIMITANT LA ZONE NON ENCORE EXPLOITEE )
      IX$133=        0
      CALL LECTAB ( FRONP (1,1), IX$133 , IX$127 (1,1), IX$127 (1, 1) ,
     +IX$127 (1, 2) )
C
C$$$$
C$$$$DEBUT DE DEFINITION DU SET ZONEX  PAR UNE INSTRUCTION 'DEFSET'
C$$$$
C
      I$EXPR=1
      NEXMAX=0
90359$
```

```
      N$XMIN= 10000000
      N$YMAX=0
      N$YMIN= 10000000
      I$NX  =IX$103(5)
      I$ADR =0
      N$CARD=0
      I$XYZ = 10000000
      N$US=IX$103(3)
      XY    =N$US+1
      DO 90009 I$1=1,N$US
      XY    =XY    -1
      IX$124=XY    +(XY    +31)/32*IX$103(13)+32
      I$2=IX$109(I$EXPR)+IX$124
      IX$103(I$2)=-1
      $X$103=$X$CUS( 1,XY    ,MINE  ,IX$103)
      $X$104=$X$CUS( 2,XY    ,MINE  ,IX$103)
      IF(.NOT.( INOUTF ( FRONP (1,1), $X$103 , $X$104 , IX$127 (1,1), IX
     +$127 (1, 1) , IX$127 (1, 2) ) .GT. 0
     +)) GO TO 90010
      IX$103(I$2   )=I$ADR
      J$=(XY    -1)/I$NX
      I$Y=J$+1
      I$X=XY    -J$*I$NX
      IF(I$Y.GE.N$YMIN) GO TO 90011
      N$YMIN=I$Y
      GO TO 90012
90011 IF(I$Y.LE.N$YMAX) GO TO 90012
      N$YMAX=I$Y
90012 IF(I$X.GE.N$XMIN) GO TO 90013
      N$XMIN=I$X
      GO TO 90014
90013 IF(I$X.LE.N$XMAX) GO TO 90014
      N$XMAX=I$X
90014 CONTINUE
      I$ADR=XY
      N$CARD=N$CARD+1
      GO TO 90009
90010 IX$103(I$2   )=-1
C
90009 CONTINUE
C
      I$EXPR=IX$109(I$EXPR)
      I$=I$EXPR+24
      IX$103(I$)=I$ADR
      I$ADR=N$XMIN
     +    +(N$YMIN-1)*IX$103(5)
      I$=I$EXPR+27
      IX$103(I$)=I$ADR
      I$=I$EXPR+28
      N$XMIN=N$XMAX-N$XMIN+1
      IX$103(I$)=N$XMIN
      I$=I$EXPR+29
      N$YMIN=N$YMAX-N$YMIN+1
      IX$103(I$)=N$YMIN
      I$=I$EXPR+26
      IX$103(I$)=N$CARD
      END$
```

```
      IF(N$CARD.NE.0) GO TO 90016
      I$=I$EXPR+23
      DO 90015 J$=1,7
      I$=I$+1
90015 IX$103(I$)=0
90016 CONTINUE
C
C****
C**** FIN DE DEFINITION DU SET ZONEX
C****
C
90001 CONTINUE
      IF(.NOT.( $X$ANS( 13, 52HVOULEZ VOUS (RE)INITIALISER LES TAUX DE R
     +ECUPERATION,IMPR$,LECT$)
     +)) GO TO 90017
      I$=1
      I$4=IX$103(3)
      DO 90018 I$3=1,I$4
      I$3B  =I$3   +(I$3   +31)/32*IX$103(13)+32
      I$5=IX$107(I$)+I$3B
      MINE  (I$5)=   .100000000E+01
90018 CONTINUE
90019 CONTINUE
      WRITE(IMPR$ , 90020)
90020 FORMAT( 1X, 56HDESIGNEZ SUR LA TABLE A DIGITALISER LES POINTS FRON
     +TIERE , / , 1X, 52HDELIMITANT UNE ZONE OU L'ON VEUT AFFECTER UN TA
     +UX DE , / , 1X, 21HRECUPERATION CONSTANT )
      IX$134=       0
      CALL LECTAB ( FRONP (1,1), IX$134 , IX$127 (1,1), IX$127 (1, 1) ,
     +IX$127 (1, 2) )
      WRITE(IMPR$ , 90021)
90021 FORMAT( 1X, 48HQUEL TAUX DE RECUPERATION VOULEZ VOUS AFFECTER ?)
C****
C**** INSTRUCTION READ
C****
C
      CALL $X$LEC(END$,PRNAM$, 0, 81 , 2, 1, TAUX )
C
C****  FIN DE L'INSTRUCTION READ
C
C
C****
C****DEBUT DE DEFINITION DU SET USET   PAR UNE INSTRUCTION 'DEFSET'
C****
C
      I$EXPR=1
      N$XMAX=0
      N$XMIN=  10000000
      N$YMAX=0
      N$YMIN=  10000000
      I$NX  =IX$103(5)
      I$ADR =0
      N$CARD=0
      I$XYZ =  10000000
      N$US=IX$103(3)
      XY    =N$US+1
 00481*
```

```
      DO 90022 I$1=1,N$US
      XY     =XY     -1
      IX$124=XY     +(XY     +31)/32*IX$103(13)+36
      I$2=IX$113(I$EXPR)+IX$124
      IX$103(I$2)=-1
      $X$105=$X$CUS( 1,XY     ,MINE  ,IX$103)
      $X$106=$X$CUS( 2,XY     ,MINE  ,IX$103)
      IF(.NOT.( INOUTF ( FRONP (1,1), $X$105 , $X$106 , IX$127 (1,1), IX
     +$127 (1, 1) , IX$127 (1, 2) ) .GT. 0
     +)) GO TO 90023
      IX$103(I$2    )=I$ADR
      J$=((Y     -1)/I$NX
      I$Y=J$+1
      I$X=(Y     -J$*I$NX
      IF(I$Y.GE.N$YMIN) GO TO 90024
      N$YMIN=I$Y
      GO TO 90025
90024 IF(I$Y.LE.N$YMAX) GO TO 90025
      N$YMAX=I$Y
90025 IF(I$X.GE.N$XMIN) GO TO 90026
      N$XMIN=I$X
      GO TO 90027
90026 IF(I$X.LE.N$XMAX) GO TO 90027
      N$XMAX=I$X
90027 CONTINUE
      I$ADR=XY
      N$CARD=N$CARD+1
      GO TO 90022
90023 IX$103(I$2    )=-1
C
90022 CONTINUE
C
      I$EXPR=IX$113(I$EXPR
      I$=I$EXPR+24
      IX$103(I$)=I$ADR
      I$ADR=N$XMIN
          +(N$YMIN-1)*IX$103(5)
      I$=I$EXPR+27
      IX$103(I$)=I$ADR
      I$=I$EXPR+28
      N$XMIN=N$XMAX-N$XMIN+1
      IX$103(I$)=N$XMIN
      I$=I$EXPR+25
      N$YMIN=N$YMAX-N$YMIN+1
      IX$103(I$)=N$YMIN
      I$=I$EXPR+26
      IX$103(I$)=N$CARD
      IF(N$CARD.NE.0) GO TO 90029
      I$=I$EXPR+23
      DO 90028 J$=1,7
      I$=I$+1
90028 IX$103(I$)=0
90029 CONTINUE
C
C****
C**** FIN DE DEFINITION DU SET USET
00537*
```

```
C****.
C
      I$=I($113( 1          )+24
      XY    =IX$103(I$)
90030 IF(X/    .EQ.0) GO TO 90031
      IX$124=XY    +(XY    +31)/32*IX$103(13)+32
      IX$135=IX$107(1)+IX$124
      MINE ( IX$135 ) = TAUX
      I$=I($124+IX$113( 1          )
      XY    =IX$103(I$)
      GO TO 90030
90031 CONTINUE
      IF(.NOT.( $X$ANS( 13, 52HVOULEZ VOUS FIXER LE TAUX DE RECUP. D'UNE
     + NOU). ZONE,IMPR$,LECT$)
     +)) GO TO 90032
      GO TO 90019
90032 CONTINUE
90017 CONTINUE
      NFRON = NFRON / 2
      IX$136=IX$103(3)
      DO 90033 XY    =1,IX$136
      IX$124=XY    +(XY    +31)/32*IX$103(13)+32
      IX$137=IX$111(          1)+IX$124
      IX$138=IX$107(1)+IX$124
      IX$139=IX$105(1)+IX$124
      MINE ( IX$137 ) = MINE ( IX$138 ) * MINE ( IX$139 )
      IX$140=IX$111(          2)+IX$124
      MINE ( IX$140 ) = .10000E+01
90033 CONTINUE
      IX$141=IX$116(3)
      DO 90034 PAN   =1,IX$141
      IX$126=PAN   +(PAN   -1)/32*IX$116(13)+32
      IX$142=IX$[illegible]          1)+IX$126
      IF(.NOT.(
     +PLANEX (   IX$142      .GT.       .00000E+00
     +))GO TO 90035
      CONTINUE
90034 CONTINUE
90035 CONTINUE
90036 CONTINUE
      WRITE(IMPR$ , 90037)
90037 FORMAT( 1X, 46HDESIGNEZ SUR LA TABLE A DIGITALISER LES POINTS , /
     +, 1X, 48HFRONTIERE DELIMITANT UN NOUVEAU PANNEAU MINIER )
      IX$143=          0
      CALL LECTAB ( FRONP (1,1), IX$143 , IX$127 (1,1), IX$127 (1, 1) ,
     +IX$127 (1, 2) )
      IF(.NOT.( IX$127(2,2) .GT. NFRON
     +)) GO TO 90038
      WRITE(IMPR$ , 90039) NFRON
90039 FORMAT( 1X, 39HUN PANNEAU MINIER NE PEUT AVOIR PLUS DE , X,I3 , 1X
     +, 16HPOINTS FRONTIERE)
      GO TO 90036
90038 CONTINUE
C
C****
C****DEBUT DE DEFINITION DU SET USET   PAR UNE INSTRUCTION 'DEFSET'
80593*
```

```
C$$$$.
C
      I$EXPR=1
      N$XMAX=0
      N$XMIN=  10000000
      N$YMAX=0
      N$YMIN=  10000000
      I$NX  =IX$103(5)
      I$ADR =0
      N$CARD=0
      I$XYZ =  10000000
      N$US=IX$103(3)
      XY    =N$US+1
      DO 90040 I$1=1,N$US
      XY    =XY    -1
      IX$124=XY    +(XY    +31)/32*IX$103(13)+32
      I$2=IX$113(I$EXPR)+IX$124
      IX$103(I$2)=-1
      $X$107=$X$CUS( 1,XY    ,MINE  ,IX$103)
      $X$108=$X$CUS( 2,XY    ,MINE  ,IX$103)
      IF(.NOT.( INOUTF ( FRONP (1,1), $X$107 , $X$108 , IX$127 (1,1), IX
     +$127 (1, 1) , IX$127 (1, 2) ) .GT. 0
     +)) GO TO 90041
      IX$103(I$2   )=I$ADR
      J$=(XY    -1)/I$NX
      I$Y=J$+1
      I$X=XY    -J$*I$NX
      IF(I$Y.GE.N$YMIN) GO TO 90042
      N$YMIN=I$Y
      GO TO 90043
90042 IF(I$Y.LE.N$YMAX) GO TO 90043
      N$YMAX=I$Y
90043 IF(I$X.GE.N$XMIN) GO TO 90044
      N$XMIN=I$X
      GO TO 90045
90044 IF(I$X.LE.N$XMAX) GO TO 90045
      N$XMAX=I$X
90045 CONTINUE
      I$ADR=XY
      N$CARD=N$CARD+1
      GO TO90040
90041 IX$103(I$2   )=-1
C
90040 CONTINUE
C
      I$EXPR=IX$113(I$EXPR)
      I$=I$EXPR+24
      IX$103(I$)=I$ADR
      I$ADR=N$XMIN
     +    +(N$YMIN-1)*IX$103(5)
      I$=I$EXPR+27
      IX$103(I$)=I$ADR
      I$=I$EXPR+28
      N$XMIN=N$XMAX-N$XMIN+1
      IX$103(I$)=N$XMIN
      I$=I$EXPR+29
000491
```

```
      N$YMIN=N$YMAX-N$YMIN+1
      IX$103(I$)=N$YMIN
      I$=I$EXPR+26
      IX$103(I$)=N$CARD
      IF(N$CARD.NE.0) GO TO 90047
      I$=I$EXPR+23
      DO 90046 J$=1,7
      I$=I$+1
90046 IX$103(I$)=0
90047 CONTINUE
C
C$$$$
C$$$$ FIN DE DEFINITION DU SET USET
C$$$$
C
      NC32$=IX$113(1)
      IX$144=IX$CAR( 1, 1,MINE  ,NC32$)
      CARDP = IX$144
C
C$$$$
C$$$$DEBUT DE DEFINITION DU SET USET   PAR UNE AFFECTATION
C$$$$
C
      I$EXPR=1
      IX$145=IX$113(I$EXPR)
      IX$147=IX$113(1)
      IX$149=IX$109(1)
      N$XMAX=0
      N$XMIN=  10000000
      N$YMAX=0
      N$YMIN=  10000000
      I$NX  =IX$103(5)
      I$ADR =0
      N$CARD=0
      I$XYZ =  10000000
      N$US=IX$103(3)
      I$$$$$=N$US+1
      DO 90048 I$1=1,N$US
      I$$$$$=I$$$$$-1
      IB$$$$=I$$$$$+(I$$$$$+31)/32*IX$103(13)+32
      IX$150=IX$149+IB$$$$
      IX$148=IX$147+IB$$$$
      IX$146=IX$145+IB$$$$
      IF(.NOT.( IX$103 ( IX$148 ).GE.0 .AND. IX$103 ( IX$150 ).GE.0
     +)) GO TO 90049
      IX$103(IX$146)=I$ADR
      J$=(I$$$$$-1)/I$NX
      I$Y=J$+1
      I$X=I$$$$$-J$*I$NX
      IF(I$Y.GE.N$YMIN) GO TO 90050
      N$YMIN=I$Y
      GO TO 90051
90050 IF(I$Y.LE.N$YMAX) GO TO 90051
      N$YMAX=I$Y
90051 IF(I$X.GE.N$XMIN) GO TO 90052
      N$XMIN=I$X
@@705*
```

```
      GO TO 90053
90052 IF(I$X.LE.N$XMAX) GO TO 90053
      N$XMAX=I$X
90053 CONTINUE
      I$ADR=I$$$$$
      N$CARD=N$CARD+1
      GO TOS0048
90049 IX$103(Ix$146)=-1
C
90048 CONTINUE
C
      I$EXPR=IX$113(I$EXPR)
      I$=I$EXPR+24
      IX$103(I$)=I$ADR
      I$ADR=N$XMIN
     +    +(N$YMIN-1)*IX$103(5)
      I$=I$EXPR+27
      IX$103(I$)=I$ADR
      I$=I$EXPR+28
      N$XMIN=N$XMAX-N$XMIN+1
      IX$103(I$)=N$XMIN
      I$=I$EXPR+29
      N$YMIN=N$YMAX-N$YMIN+1
      IX$103(I$)=N$YMIN
      I$=I$EXPR+26
      IX$103(I$)=N$CARD
      IF(N$CARD.NE.0) GO TO 90055
      I$=I$EXPR+23
      DO 90054 J$=1,7
      I$=I$+1
90054 IX$103(I$)=9
90055 CONTINUE
C
C****
C**** FIN DE DEFINITION DU SET WSET
C****
C
      NC32$=IX$113(1)
      IX$145=IX$CAR( 1, 1,MINE  ,NC32$)
      IF(.NOT.( CARDP .NE. IX$145
     +)) GO TO 90056
      WRITE(IMPR$ , 90057)
90057 FORMAT( 1X, 43HERREUR! LE PANNEAU DESIGNE N'EST PAS TOUT ENTIER ,
     +/ , 1X, 41HCONTENU DANS LA ZONE NON ENCORE EXPLOITEE )
      GO TO 90058
90056 CONTINUE
      IX$146=IX$120(1)+IX$126
      IX$147=1
      IX$148=IX$111(           1
     +          )
      PLANEX ( IX$146 ) = SOMFXY ( IX$147 , IX$148 , FRONP (1,1), MINE ,
     + IX$103 , $X$101 , IX$127 (1,1), IX$127 (1, 1) , IX$127 (1, 2) )
      IX$149=IX$118(1)+IX$126
      IX$150=1
      IX$151=IX$111(           2
     +          )
007611
```

```
      PLANEX ( IX$149 ) = SOMFXY ( IX$150 , IX$151 , FRONP (1,1), MINE ,
     + IX$103 , $X$101 , IX$127 (1,1), IX$127 (1, 1) , IX$127 (1, 2) )
      DO 90059 I       =              1,NQUALI       , 1
      IX$152=IX$115(I             )+IX$126
      IX$153=1
      IX$154=IX$102(I
     +             )
      IX$155=IX$118(1)+IX$126
      PLANEX ( IX$152 ) = SOMFXY ( IX$153 , IX$154 , FRONP (1,1), MINE ,
     + IX$103 , $X$101 , IX$127 (1,1), IX$127 (1, 1) , IX$127 (1, 2) ) /
     + PLANEX ( IX$155 )
90059 CONTINUE
      I = IX$127(2,2)
      IX$156  =IX$118  (1)+IX$126
      IX$157  =IX$120  (1)+IX$126
      WRITE(IMPR$ , 90060) I , PLANEX ( IX$156 ) , PLANEX ( IX$157 )
90060 FORMAT( /,1H1 , 1X, 47HCARACTERISTIQUES GEOMETRIQUES DU PANNEAU MI
     +NIER , / , 1X, 47H**********************************************
     +, / / / , 1X, 29HNOMBRE DE POINTS FRONTIERE :   , X,I3 , / , 1X, 29
     +HSURFACE DU PANNEAU          :   , X,E10.3 , / / / , 1X, 29HVOLUME DU PA
     +NNEAU          :   , X,E10.3 , / / / , 1X, 47HCARACTERISTIQUES DE Q
     +UALITE  DU  PANNEAU MINIER , / , 1X, 47H**********************
     +********** , / / / )
      DO 90061 I       =              1,NQUALI       , 1
      IX$158  =IX$102  (I             )+19
      IX$159  =IX$115  (I             )+IX$126
      WRITE(IMPR$ , 90062) MINE (IX$158 ),MINE (IX$158 +1) , PLANEX ( IX
     +$159 )
90062 FORMAT( 1X, 17HVALEUR MOYENNE DE , 1X, 2A4 , X,E10.3 )
90061 CONTINUE
      IF(.NOT.( $X$ANS( 10, 39HVOULEZ VOUS EXPLOITER CE PANNEAU MINIER ,
     +IMPR$,LECT$)
     +)) GO TO 90063
      NC32$=0
      IX$160=IX$CAR( 1, 0,PLANEX,NC32$)
      IF(.NOT.( PAN .EQ. IX$160
     +)) GO TO 90064
      WRITE(IMPR$ , 90065)
90065 FORMAT( 1X, 37HLE NOMBRE MAXIMUM DE PANNEAUX MINIERS , 1X, 20HAUTO
     +RISE EST ATTEINT)
      GO TO 90058
90064 CONTINUE
C
C****
C****DEBUT DE DEFINITION DU SET ZONEX  PAR UNE AFFECTATION
C****
C
      I$EXPR=1
      IX$161=IX$109(I$EXPR)
      IX$163=IX$109(1)
      IX$165=IX$113(1)
      N$XMAX=0
      N$XMIN=  10000000
      N$YMAX=0
      N$YMIN=  10000000
      I$NX  =IX$103(5)
C83178
```

```
      I$ADR =0
      N$CARD=0
      I$XYZ =  10000000
      N$US=IX$103(3)
      I$$$$$=N$US+1
      DO S0066 I$1=1,N$US
      I$$$$$=I$$$$$-1
      IB$$$$=I$$$$$+(I$$$$$+31)/32*IX$103(13)+32
      IX$166=IX$165+IB$$$$
      IX$164=IX$163+IB$$$$
      IX$162=IX$161+IB$$$$
      IF(.NOT.( IX$103 ( IX$164 ).GE.0 .AND..NOT. IX$103 ( IX$166 ).GE.0
     +
     +)) GO TO 90067
      IX$103(IX$162)=I$ADR
      J$=(I$$$$$-1)/I$NX
      I$Y=J$+1
      I$X=I$$$$$-J$*I$NX
      IF(I$Y.GE.N$YMIN) GO TO 90068
      N$YMIN=I$Y
      GO TO 90069
90068 IF(I$Y.LE.N$YMAX) GO TO 90069
      N$YMAX=I$Y
90069 IF(I$X.GE.N$XMIN) GO TO 90070
      N$XMIN=I$X
      GO TO 90071
90070 IF(I$X.LE.N$XMAX) GO TO 90071
      N$XMAX=I$X
90071 CONTINUE
      I$ADR=I$$$$$
      N$CARD=N$CARD+1
      GO TOS0066
90067 IX$103(IX$162)=-1
C
C
90066 CONTINUE
C
      I$EXPR=IX$109(I$EXPR)
      I$=I$EXPR+24
      IX$103(I$)=I$ADR
      I$ADR=N$XMIN
     +    +(N$YMIN-1)*IX$103(5)
      I$=I$EXPR+27
      IX$103(I$)=I$ADR
      I$=I$EXPR+28
      N$XMIN=N$XMAX-N$XMIN+1
      IX$103(I$)=N$XMIN
      I$=I$EXPR+29
      N$YMIN=N$YMAX-N$YMIN+1
      IX$103(I$)=N$YMIN
      I$=I$EXPR+26
      IX$103(I$)=N$CARD
      IF(N$CARD.NE.0) GO TO 90073
      I$=I$EXPR+23
      DO 90072 J$=1,7
      I$=I$+1
90072 IX$103(I$)=0
908733
```

```
90073 CONTINUE
C
C%%%%
C%%%% FIN DE DEFINITION DU SET ZONEX
C%%%%
C
      I = PAN
      IX$161=        2
      CALL $TOFRT ( IX$161 , FRONP (1,1), IX$121 , IX$122 , I , PLANEX ,
     + IX$115 , $X$102 , IX$127 (1,1), IX$127 (1, 1) , IX$127 (1, 2) )
      NC32$=0
      IX$162=IX$CAR( 1, 0,PLANEX,NC32$)
      IF(.NOT.( PAN .LT. IX$162
     +)) GO TO 90074
      PAN = PAN + 1
      IF( (PAN   .LT.1) .OR. (PAN   .GT.IX$116(3)))
     +CALL $X$ERR(PRNAM$,  170,  2)
      IX$126=PAN   +(PAN   +31)/32*IX$116(13)+32
90074 CONTINUE
90063 CONTINUE
90058 CONTINUE
      IF(.NOT.( $X$ANS( 11, 41HVOULEZ VOUS INTRODUIRE UN NOUVEAU PANNEAU
     + ,IMPR$,LECT$)
     +)) GO TO 90075
      GO TO 90036
90075 CONTINUE
90076 CONTINUE
      WRITE(IMPR$ , 90077)
90077 FORMAT( /,1H1 , 1X, 43HTABLEAU RECAPITULATIF DES PANNEAUX MINIERS
     + , / , 1X, 43H=========================================  , / / /
     +)
      IX$163=IX$116(3)
      DO 90078 PAN   =1,IX$163
      IX$126=PAN   +(PAN   +31)/32*IX$116(13)+32
      IX$164=IX$122(           1)+IX$126
      IF(.NOT.(
     +PLANEX  (   IX$164  )   .GT.       .00000E+00
     +))GO TO 90079
      IX$165=IX$122(           1)+IX$126
      I = ( PLANEX ( IX$165 ) + .50000E+00) / .20000E+01
      IX$166  =IX$118  (1)+IX$126
      IX$167  =IX$120  (1)+IX$126
      WRITE(IMPR$ , 90080) PAN , I , PLANEX ( IX$166 ) , PLANEX ( IX$167
     + )
90080 FORMAT( / / / , 1X, 27HPANNEAU NUMERO           :  , X,I3 , / , 1X
     +, 27HNOMBRE DE PONTS FRONTIERE:  , X,I3 , / , 1X, 27HSURFACE DU PA
     +NNEAU        :  , X,E10.3 , / , 1X, 27HVOLUME DU PANNEAU        :
     +, X,E10.3 , / / )
      DO 90081 I     =          1,NQUALI      , 1
      IX$168  =IX$102  (I           )+19
      IX$169  =IX$115  (I           )+IX$126
      WRITE(IMPR$ , 90082) MINE (IX$168 ),MINE (IX$168 +1) , PLANEX ( IX
     +$169 )
90082 FORMAT( 1X, 17HVALEUR MOYENNE DE , 1X, 2A4 , X,E10.3 )
90081 CONTINUE
90078 CONTINUE
00929$
```

```
90078 CONTINUE
90079 CONTINUE
      RETURN
      END
00932x
```

BIBLIOGRAPHIE

°ADA-81§

Groupe ADA de l'AFCET 1981

Rapport d'évaluation du langage ADA

°BOU-79§

BOUCHET P. 1979

Implémentation du langage GEOL.

Rapport de D.E.A.

°BOU-80§

BOUCHET P., CUNIN P.Y., GRIFFITHS M., MALLET J.L., ROYER J.J. 1980

GEOL : a graphical and statical retrival langage for geodata management.

Compstat 1980, VIENNA Physical-Verlag for IASC.

°CUN-78§

CUNIN P.Y., GRIFFITHS M. 1978

Générateur d'analyseur LL(1) sur IRIS 80

CRIN note interne

°CUN-79a§

CUNIN P.Y., GRIFFITHS M., MALLET J.L., ROYER J.J. 1979

GEOL : langage d'anlyse de données géologiques.

Convencion Informatica Latina,Barcelone,Espagne

°CUN-79b§

CUNIN P.Y. 1979

Fiabilité et Sécurité des Programmes.

Thése détat.

°LEB-79§

LEBART L., MORINEAU A., TABARD N. 1979

Techniques de la description statiatique.

°MAL-76§

MALLET J.L. et al. 1976

Programmes de cartographie automatique : CARTOLAB.

Sciences de la Terre,série Informatique-Géologie N°7

°RIC-75§

RICHARDS M. 1975

Portable compiler.

Springer Verlag

°ROY-79§

ROYER A. 1979

Interface GEOL-CARTOLAB.

Rapport de D.E.A.

°ROY-82§

ROYER A. 1982

Développement et Suive de Programmes.

Thése de 3ème cycle à paraître

°SOFT-75§

Software ingenering 1975

Springer Verlag

°STO-80§

STONEMAN 1980

Requirements for ADA Programming Support of Environments.

institut
national
polytechnique
de lorraine

Le Président

N/Réf. : Scol.

AUTORISATION DE SOUTENANCE DE THESE DE DOCTORAT 3ème CYCLE

-:-:-:-:-:-:-:-:-:-:-:-:-:-

VU LE RAPPORT ETABLI PAR :

Monsieur le Professeur GRIFFITHS

le Président de l'Institut National Polytechnique de Lorraine autorise :

Monsieur BOUCHET Pierre

à soutenir, devant l'I.N.P.L., une thèse de Doctorat intitulée :

"DEFINITION ET IMPLANTATION D'UN LANGAGE DE MANIPULATIONS DE DONNEES"

en vue de l'obtention du titre de DOCTEUR 3ème CYCLE

Spécialité "INFORMATIQUE"

Fait à NANCY, le 28 février 1982

Le Président de l'I.N.P.L.

M. LUCIUS