GIBIANE - CASTEM2000

SEMT/LAMS
T. Charras

Table des matières

1 Introduction **2**

2 Presentation générale de Castem2000 **3**
2.1 Structure de Castem2000 3
2.2 Les objets dans Castem2000 4
2.2.1 Liste des règles du superviseur 5
2.3 Quelques remarques sur le fonctionnement 6
2.3.1 Recherche du nom de l'opérateur 6
2.3.2 Position des opérandes 6
2.4 Conclusion du premier chapitre 7

3 Le langage de données : GIBIANE **8**
3.1 Améliorations de convivialité 8
3.1.1 Commentaires 8
3.1.2 résultats multiples 8
3.1.3 Traitement des parenthèses 9
3.2 Addition au langage, structuration des commandes 9
3.2.1 Exécution alternative : SI 9
3.2.2 Exécution répétitive : REPETER 10
3.3 Macro commandes : les PROCEDURES 11
3.3.1 Déclaration d'une procédure 11
3.3.2 Exécution d'une procédure 12
3.3.3 Sortie anticipée d'une procédure 13
3.3.4 Arguments différés, arguments facultatifs 13
3.3.5 Non-initialisation des variables 14
3.3.6 Résultats en cours de procédure 15
3.3.7 Les procédures de Castem2000 15
3.3.8 Les procédures personnelles 16

4 les OBJETS du langage **17**
4.1 Les TEXTES 17
4.2 Les TABLES 18
4.3 Les METHODES 19

4.4 Les objets de type OBJET 20

5 Bibliographie **26**

Chapitre 1

Introduction

Castem2000 est le fruit de plus de vingt ans de développement de programmes de calcul par la méthode des éléments finis (résolution d'équations aux dérivées partielles sur un domaine fini). Il tente d'incorporer toutes les étapes nécessaires pour mener à bien une étude par éléments finis, du maillage au post-traitement. Son champ d'application privilégié est la mécanique mais il traite aussi bien de la thermique, de l'hydraulique ou de l'éléctromagnétisme.

Castem2000 est fondé sur un concept OPERATEUR-OPERANDES et se différencie en cela des programmes classiques d'éléments finis. Les opérateurs sont des outils qui agissent sur des objets spécifiques.

Dans le cadre du projet ELAN, GIBIANE a intégré les concepts modernes d'objets et de méthodes. Une approche de Castem2000 à travers le concept de : "méthodes qui s'appliquent à des objets" est maintenant possible.

Il était donc nécessaire de mettre à jour la présentation de GIBIANE.

Chapitre 2

Presentation générale de Castem2000

2.1 Structure de Castem2000

Quand on démarre un micro-ordinateur, celui-ci lance son programme d'exploitation (par exemple UNIX) et se met en attente des commandes utilisateurs. Le programme d'exploitation a une structure de "Data Flow Control" c'est à dire que ce sont les données utilisateur qui orientent le déroulement des actions, celles-ci n'ayant a priori aucun ordre pré-établi.

Cette structure a été choisie pour Castem2000 et cela nécéssite un superviseur de commandes qui soit capable de :

- lire une commande utilisateur
- reconnaître l'action à faire
- donner la main à l'opérateur et instaurer un dialogue avec lui pour lui fournir les arguments qu'il réclame et récupérer les résultats qu'il fournit
- attendre une nouvelle commande

La syntaxe générale de UNIX est :

OPERATEUR OPERANDE1 OPERANDE2 ...

Celle de Castem2000 est :

RESULTAT = OPERATEUR OPERANDES ;

Par exemple pour copier un fichier sur un autre il faut sous UNIX écrire :

CP FICH1 FICH2

alors que dans la syntaxe de Castem2000 nous devons écrire :

FICH2 = COPIER FICH1 ;

L'action complète dans Castem2000 est décomposée en trois parties :

1. Lecture de la commande par le superviseur qui donne le contrôle à l'opérateur COPIER.
2. COPIER demande au superviseur les opérandes (ici FICH1) puis il exécute le travail et rend au superviseur son résultat.
3. Le superviseur traite la commande "FICH2= résultat". Il affecte le nom FICH2 à ce résultat et le place dans la base des objets déjà crées et nommés.

2.2 Les objets dans Castem2000

Comme nous l'avons vu UNIX est un système d'exploitation de gestion des fichiers, or en éléments finis il s'agit de maillages, de rigidités, de champs de vecteurs déplacements, de champs magnétiques.... Les entités ou objets que reconnaîtra le superviseur sont de natures(types) différentes.

Pour travailler intelligement, on se dote de structures complexes :

1. Les premières sont issues des entités physiques à représenter : MAILLAGE, CHPOINT (champs définis sur les noeuds du maillage), MCHAML (Champs définis sur les éléments finis), CHARGEMENT (description d'un chargement variant dans le temps et dans l'espace)...
2. les deuxième ont un rôle plus mathématiques : RIGIDITE (qui sert pour toutes les matrices), EVOLUTION (décrit l'évolution d'une quantité en fonction d'une autre), NUAGE (généralisation de l'EVOLUTION)...
3. Les derniers sont utilitaires : TABLE, ENTIER, FLOTTANT, MOT...

Ces structures sont les objets de Castem2000. Les instanciations de ces types d'objets sont **nommées** ce qui permet de les manipuler individuellement. Les instanciations sont aussi appelés "objets de Castem2000".

Un objet est une **collection prédéfinie** d'informations. Ainsi un opérateur qui peut travailler sur un type d'objet n'a pas à se soucier de la provenance de l'objet mais seulement de son type qui lui assure toute une série d'informations dans un format prédéfini. Il en découle ainsi l'indépendance des opérateurs entre eux.

Seul l'opérateur manipule les objets et en crée de nouveaux, le superviseur n'est là que pour lui donner la main et lui fournir les opérandes qu'il réclame.

La syntaxe générale et le concept d'objet de Castem2000 sont très intuitifs, ils correspondentbien à une vision "algébrique" de la modélisation (des produits comme MATLAB ou MATHEMATICA ont une approche similaire).

2.2.1 Liste des règles du superviseur

- Le point virgule signale la fin de la commande et demande au superviseur de lancer l'analyse puis l'exécution.
- La commande est analysée seulement sur les 72 premiers caractères de chaque ligne (un reste des antiques cartes perforées !).
- Une commande peut tenir sur plusieurs lignes et il peut y avoir plusieurs commandes par lignes.
- A priori les caractères minuscules sont remplacés par des majuscules. Les quotes, encadrant des caractères, marquent la définition d'une chaîne de caractères à prendre sans modification.
  1. *"res1 = mot 'AB3ef' " ; et " res2 = mot AB3ef " ; ne produisent pas le même résultat, la deuxième instruction est égale à : " res2 = mot AB3EF "*
  2. *dans la séquence : " X = 5 ; Y = x + 3 ; " x est traduit en X et Y vaudra 8*
  3. *" x = 5 ; y = X + 3 ; " produit aussi Y égal 8 car le nom des objets est toujours transcodé en majuscule.*
  4. *Le nom d'un objet ne peut pas faire plus de 8 caractères.*
- Le caractère blanc est le séparateur universel. D'autres caractères sont parfois des séparateurs, par exemple + - *. Néanmoins nous ne saurions trop conseiller d'utiliser systématiquement des blancs entre les données afin d'éviter tout risque d'erreur surtout avec les symboles plus "+", moins "-" et point ".". Par exemple le point "." sert de virgule flottante et de séparateur des indices des objets "tables" il ne faut donc pas confondre TAB.1.2 et TAB . 1 . 2 , dans le premier cas il s'agit de la table TAB et de son indice flottant valant 1.2 et dans l'autre de la table TAB et de son indice 1 qui lui même pointe vers une table dont on prend l'indice 2.
- Un nom d'objet a au plus huit caractères alphanumériques, au delà toute chaîne de caractères devient un objet de type "MOT" qui n'a pas de nom. Ce nom ne doit pas pouvoir être interprété comme une constante numérique (par exemple 1e5). Le type d'un objet est une chaîne de caractères de 8 lettres ou moins (PROCEDUR, CHARGEME, CHPOINT, EVOLUTIO, MOT...).
- Aucun nom n'est réservé, même ceux des opérateurs peuvent être surchargés (attention, c'est une source d'erreur).
- L'instruction est lue de gauche à droite et est exécutée dans ce même ordre sans notion de hiérarchie entre opérateurs. La notion de calcul en chaîne apparaît ici. Elle permet de ne pas nommer les résultats intermédiares.
  - *x = 3 + 5 * 8 ; produit x égal 64*
  - *y = 3 + 5 ; x = y * 8 ; produit aussi x égal 64 mais le résultat y égal 8 est conservé*

2.3 Quelques remarques sur le fonctionnement

2.3.1 Recherche du nom de l'opérateur

Le superviseur lit la commande, située après le signe =, jusqu'à ce qu'il rencontre une chaîne de caractères (donc ni entier ni flottant) qui ne correspond pas à un objet nommé. Il interrompt alors le décodage de la ligne et crée un objet de type MOT. Si cette chaîne ne fait pas plus de huit caractères il lui donne un nom qui est égal au contenu de la chaîne, le nom et le contenu sont identiques. Ensuite le superviseur vérifie si les quatres premières lettres de ce mot correspondent au nom d'un opérateur. Si tel est le cas, le superviseur lui donne le contrôle sinon il exécute le signe égal (=), c'est à dire qu'il affecte des noms aux résultats. Une fois donné le contrôle à un opérateur, le superviseur attend que celui-ci ait fini son travail.

Nous ne parlerons pas ici du traitement des "procédures" qui ressemble à celui des opérateurs.

Remarque : Le message d'erreur dans le signe = du type "Il y a un objet en trop par rapport aux noms à affecter" est souvent dû à une mauvaise orthographe de l'opérateur ou au fait qu'il est précédé par un mot. C'est pour cette raison qu'il est préférable de toujours mettre le nom de l'opérateur avant les opérandes de type "MOT". Ainsi il faut écrire : " EGA MOT1 MOT2 " et non pas " MOT1 EGA MOT2"

2.3.2 Position des opérandes

Nous avons déjà parlé de la position relative du nom de l'opérateur et des opérandes, mais il faut, de temps en temps, faire aussi attention à la position des opérandes les uns avec les autres.

Quand un opérateur demande au superviseur de lui fournir des opérandes, celui-ci parcourt la liste des opérandes jusqu'à ce qu'il en trouve un du bon type et le fournit à l'opérateur. Cette recherche d'un objet du bon type s'arrête si le superviseur rencontre un "MOT", ceci permet de garder un caractère local aux données et de ne pas mélanger les données de deux opérateurs successifs.

Les opérandes d'un opérateur qui ne sont pas des MOTS peuvent donc être mis dans n'importe quel ordre s'ils sont de types différents. Si plusieurs opérandes du même type sont appelés, l'ordre peut bien évidemment avoir de l'importance, par exemple pour l'opérateur de soustraction.

Remarque : quand un opérateur a des données obligatoires et des données *facultatives déclarées à l'aide de mots-clés il peut être nécessaire de placer les données obligatoires avant les facultatives. La notice donne toujours un ordre qui fonctionne*

2.4 Conclusion du premier chapitre

Ces quelques règles sont suffisantes pour posséder un superviseur de commandes de niveau élémentaire qui fonctionne parfaitement. Cependant il est possible de rendre plus confortable ce langage de commande. Au fur et à mesure des besoins, ce langage a été amélioré et il est devenu , peu à peu, un langage plus complet qui a été baptisé "GIBIANE".

Chapitre 3

Le langage de données : GIBIANE

3.1 Améliorations de convivialité

3.1.1 Commentaires

Il est possible de commenter un jeu de données en introduisant des lignes de commandes qui ont comme premier caractère l'étoile "*". Le superviseur lit les lignes et passe aux suivantes.

3.1.2 résultats multiples

Un opératuer peut fournir plussieurs résultats et le superviseur doit leur affecter à chacun un nom. la syntaxe devient :

```
RES1 RES2 RES3 .. = OPERATEUR OPERANDES ;
```

Par exemple pour initialiser plusieurs variables il est possible d'écrire :

```
A B C D E = 1 2 3 4 5 ;
```

Cette séquence est identique à :

```
A=1 ; B=2; C=3 ; D=4 ; E=5 ;
```

En fait, très peu d'opérateurs fournissent plusieurs résultats, par contre, cela arrive plus souvent aux procédures que nous verrons plus tard.

3.1.3 Traitement des parenthèses

Pour s'affranchir de l'exécution des commandes de gauche à droite il est possible de préciser l'ordre d'exécution à l'aide de parenthèses. La règle est d'effectuer d'abord les parenthèses en commençant à chaque fois par la plus interne qui se trouve le plus à gauche. Cela revient à effectuer un groupe entre parenthèses dès qu'on rencontre la parenthèse fermante. Il peut y avoir autant de parenthèses emboitées que nécessaires. Exemples :

```
X   = 3 + ( 5 * 8 ) ;                                  ---> X = 43
LI  = ( 0 0 0 ) DROIT 10 ( 5 0 0 ) ;
SUR = TRANS 5  ( (0 0 0 ) DROIT 10 ( 5 0 0 ) )  ( 0 10 0 );
A   = ( 1 + ( 2 * ( 8 - ( 5 - 3))))                    ---> A = 13
```

3.2 Addition au langage, structuration des commandes

Le travail fait pour améliorer GIBIANE n'a pour but ni d'en faire un langage sophistiqué avec de nombreuses possibilités ni d'en faire un vrai langage de programmation. Il essaye plutôt de rendre le langage plus souple en introduisant une exécution non linéaire des commandes.*Les additions aux langages qui suivent sont traitées à l'aide d'opérateurs et seuls les quatre premières lettres suffisent.*

3.2.1 Exécution alternative : SI

L'opérateur "SI" permet, suivant la valeur de l'objet de type LOGIQUE qui le suit, d'exécuter une séquence de commandes ou une autre. Les deux syntaxes suivantes sont permises.

```
SI    LOG1 ;                         |
    suite1 de commandes              | si LOG1 a la valeur VRAI
SINON;                               | suite1 est exécutée sinon
    suite2 de commandes              | c'est suite2
FINSI;                               |
```

ou bien

```
SI   LOG1  ;                         |  La séquence suite1 est
    suite1 de commandes              |  exécutée si LOG1 est
FINSI;                               |  VRAI
```

Remarque : le logique LOG1 peut bien évidemment être le résultat d'un calcul et l'on rencontre souvent les formes :

```
SI (  AA EGA BB) ; ...

SI ( ( AA EGA BB ) ET ( CC EGA DD) ); ....
```

3.2.2 Exécution répétitive : REPETER

L'opérateur "REPETER" permet de définir une séquence de commandes à exécuter autant de fois que nécessaire.

Syntaxe de REPETER

```
REPETER  NOMBOUCL  NFOIS;
      ...
      ...
   suite de commandes
      ...
FIN  NOMBOUCL ;
```

La suite de commandes comprise entre REPETER et FIN sera ainsi répétée NFOIS. La séquence est toujours exécutée au moins une fois. Il est permis de mettre des SI à l'intérieur des boucles et réciproquement.

Le nombre NFOIS est facultatif, s'il n'est pas précisé la boucle se répéte infiniment et il faut faire appel à un autre mécanisme pour en sortir.

QUITTER une séquence REPETER

Pour sortir d'une séquence REPETER, soit parce qu'elle était infinie, soit de manière anticipée, on peut utiliser l'opérateur QUITTER. La structure logique des données GIBIANE devient :

```
REPETER  NOMBOUCL  NFOIS;
      ...
      ...
      SI LOG1 ; QUITTER NOMBOUCL ; FINSI;
      ...
FIN  NOMBOUCL ;
```

Aprés l'instruction QUITTER l'exécution reprend à la commande située derrière FIN NOMBOUCL;

ITERER une séquence REPETER

A l'intérieur d'une séquence REPETER, on peut sauter la partie finale de la séquence de commandes en utilisant l'opérateur ITERER. Dans ce cas l'exécution de commandes continue à la commande FIN NOMBOUCL, c'est-à-dire que la séquence est répétée si le compteur n'a pas atteint NFOIS.

Exemple :

```
I = 0 ;
REPETER  NOMBOUCL  4;
    I = I + 1;
    SI ( I EGA 2 ) ; ITERER NOMBOUCL ; FINSI;
    MESSAGE ' I VAUT ' I ;
FIN  NOMBOUCL ;
```

Cette séquence produit les impressions suivantes :

```
I VAUT  1
I VAUT  3
I VAUT  4
```

3.3 Macro commandes : les PROCEDURES

Une procédure sert à regrouper une séquence d'instructions en lui donnant un nom. Cette séquence de commandes pourra être invoquée et exécutée en l'appelant par son nom. Un tel regroupement se fait dans un objet de type PROCEDURE. Dès que l'on dispose d'un objet nommé de type PROCEDURE, son utilisation se fait de la même façon que les opérateurs à la différence près que les procédures sont invoquées par leur nom complet (sans faute d'orthographe), tandis que, pour un opérateur, seul les quatre premières lettres comptent.

3.3.1 Déclaration d'une procédure

La procédure est un regroupement de commandes. Avant de l'utiliser, il faut la créer et la nommer par l'opérateur "DEBPROC" (DEBut de PROCédure) et c'est seulement lors de son appel ultérieur qu'elle sera exécutée. La syntaxe en est la suivante :

```
DEBPROC  NOMPROC  ARG1*TYP1  ARG2*TYP2 ...;
      ...
      ...
   suite de commandes
      ...
      ...
FINPROC  RES1  RES2  ....  ;
```

La procédure NOMPROC vient d'être définie. Elle admet des arguments d'entrée ARG1,ARG2... et des arguments de sortie RES1,RES2... Les arguments d'entrée peuvent avoir un type qui est précisé par TYP1,TYP2...

Une procédure est une structure semi-ouverte :
- elle a la connaissance des objets externes même s'ils ne sont pas passés en arguments
- elle ne peut pas modifier les objets externes
- elle ne transmet au monde externe que ses arguments de retour RES1,RES2..
- les objets définis dans la procédure et qui ne sont pas des arguments de retour sont perdus dès la sortie de la procédure.

Nous conseillons de passer en arguments les objets dont la procédure aura besoin, à l'exception des objets de type PROCEDURE.

3.3.2 Exécution d'une procédure

Pour utiliser la procédure il suffit de l'invoquer par son nom comme un opérateur classique.

Exemple complet :

```
DEBPROC  MODULO  II*ENTIER  JJ*ENTIER;
  SI( JJ EGA 0) ; MESSAGE ' NOMBRE ZERO INACCEPTABLE';.... ;FINSI;
  KK = II / JJ;
  MOD = II - ( KK * JJ) ;
FINPROC  MOD ;

X = 35 ; Y = 6 ; KK = 18;
Z = MODULO X Y ;
MESSAGE ' Z VAUT ' Z 'KK VAUT ENCORE ' KK;
```

Cette séquence produit le message :

```
Z VAUT 5  KK VAUT ENCORE 18
```

On remarque bien que la définition de KK à l'intérieur de la procédure n'affecte pas sa valeur à l'extérieur. On aurait pu provoquer une erreur d'exécution de la procédure si avant de l'appeler on avait surchargé une valeur à SI. Par exemple, en supposant la même définition de la procédure MODULO la séquence suivante provoque une erreur.

```
SI = 32; X = 35 ; Y = 6 ; KK = 18;
Z = MODULO X Y ;
```

En effet à l'intérieur de la procédure la commande : "32 FAUX" n'a pas de sens. Pour se prémunir du surchargement des noms d'opérateurs dans les procédures, il suffit de les mettre entre quotes, afin que l'analyseur de syntaxe les considère toujours comme des mots et non comme un nom d'objet (par chance,la plupart du temps cet objet n'existe pas. Une procédure du domaine public à donc tout intérêt à être écrite comme suit :

```
'DEBPROC'  MODULO  II*'ENTIER'  JJ*'ENTIER';
  'SI'( JJ 'EGA' 0) ; 'MESSAGE' ' NOMBRE ZERO INACCEPTABLE';.... ;'FINSI';
  KK = II / JJ;
  MOD = II - ( KK * JJ) ;
'FINPROC'  MOD ;
```

Dans ce cas là, aucune mésaventure ne peut arriver. A la lecture de cette procédure, on remarque des points de suspension. Il y aurait besoin ici d'un opérateur pour sortir de la procédure.

3.3.3 Sortie anticipée d'une procédure

Pour sortir immédiatement d'une procédure, il faut utiliser l'opérateur QUITTER qui fait continuer l'exécution à la commande FINPROC ...

Il faut préciser le nom de la procédure que l'on veut quitter (exemple paragraphe suivant).

3.3.4 Arguments différés, arguments facultatifs

Suivant le déroulement logique de la procédure, il peut être nécessaire d'avoir d'autres informations, l'opérateur ARGUMENT permet en cours de procédure d'appeler de nouveaux arguments. Prenons l'exemple suivant d'une procédure qui additionne autant d'entiers qu'on lui en fournit.

```
   'DEBPROC' ADDIT I*'ENTIER' J*'ENTIER' ;
* petite procédure pour additionnée N entiers
      K = I + J ; I = 24;
      'REPETER' BOU;
         'ARGUMENT' L/'ENTIER';
  'SI' ( 'EXISTE' L) ;
      K = K + L ;
  'SINON';
      'QUITTER' ADDIT;
  'FINSI';
      'FIN'  BOU;
   'FINPROC' K ;

* exemple d'utilisation

  A = 2; B = 5 C = 7 ;
  X = ADDIT A B 18 C -4;
* le résultat attendu est : X = 28 et  A est toujours égal à 2
  X = ADDIT 2  ( ADDIT 5 8 9 )  6 ;
* le résultat attendu est : X = 30;
```

Pour rendre facultatif un argument, il faut remplacer l'étoile qui sépare le nom de l'argument de son type par un slash ("/"). Ceci peut être fait soit pour les opérandes de DEBPROC soit pour ceux de ARGUMENT. L'opérateur EXISTE permet de savoir si un objet, passé en argument facultatif, a été trouvé. Il renvoie alors un logique VRAI ou FAUX.

3.3.5 Non-initialisation des variables

Comme nous l'avons vu, la procédure étant semi-perméable, ell va chercher dans le monde externe la valeur des objets qu'elle utilise. Pour éviter cette recherche il faut faire commencer le nom de l'objet par un point d'exclamation (" !"). Examinons les deux procédures suivantes dans lesquelles une erreur de frappe c'est glissée :

```
   'DEBPROC'  PASBELLE  II*'ENTIER'  JJ*'ENTIER';
      AD  = II + JJ;
```

```
      MU  = II * JJ ;
      RES = MU - ADD;
* la bonne ligne est : RES = MU - AD;
   'FINPROC'  RES ;

   'DEBPROC'  JOLIE  II*'ENTIER' JJ*'ENTIER';
       !AD = II + JJ ;
       !MU = II * JJ ;
       !RES = !MU - !ADD;
* la bonne ligne est : !RES = !MU - !AD;
   'FINPROC'  MOD ;

* utilisons maintenant ces procédures

    ADD = 456 ; !ADD = 456 ;  K = 5  ; I =  3 ;
    X = PASBELLE  K I ;
    Y = JOLIE  K I ;
```

L'appel à PASBELLE fournit un résultat qui vaut -441 ! Alors que l'appel à JOLIE provoque une erreur. Pour les gens très courageux, il est bon de donner un nom commençant par ! pour toutes les variables internes aux procédures, il en résulte une plus grande sureté.

3.3.6 Résultats en cours de procédure

Tout comme l'opérateur ARGUMENT permet de lire des arguments en cours de procédure, l'opérateur RESPRO (RESultat PROcédure) permet de fournir des résultats en cours de procédure en plus de ceux donnés par l'opérateur FINPRO.

3.3.7 Les procédures de Castem2000

Un grand nombre de procédures sont fournies avec le logiciel, par exemple l'algorithmique des calculs non-linéaires est piloté par les procédures PASAPAS, INCREME, TRANSNON. Ces procédures font parties intégrantes de Castem2000 et un message apparaît si elles sont surchargées par l'utilisateur. D'autres procédures, celles commençant par une @, sont fournies à titre d'exemple mais sans garantie d'une grande généralité ni d'un bon fonctionnement. Toutes ces procédures sont regroupées dans un fichier "GIBI.PROC" de même que la notice est dans "GIBI.MASTER".

3.3.8 Les procédures personnelles

L'utilisateur peut s'enregistrer, dans son espace de travail, des procédures personnelles. Ces procédures surchargeant au besoin les procédures fournies avec le logiciel. L'opérateur UTIL permet d'initialiser une bibliothèque de procédures personnelles.

1. créer un fichier contenant les procédures. Chaque procédures doit être précédée par une ligne : $$$$ NOMPROC où NOMPROC (en majuscules) est le nom de la procédure. Le fichier doit se terminer par une carte : $$$$
2. au besoin, suivre la même démarche pour les notices.
3. exécuter Castem2000 en lui demandant d'exécuter les commandes :
   UTILISATEUR PROCEDURE 'nom du fichier des procédures';
   (UTILISATEUR NOTICE 'nom du fichier des notices';
   FIN;

Cela étant fait, les exécutions suivantes de Castem2000 auront en mémoire les procédures standards et les procédures du fichier surchargeant éventuellement les procédures standards.

Remarque : La procédure INITIALE joue un rôle très particulier. Ce n'est pas une procédure dans le sens qu'elle n'est pas encadrée par DEBPROC et FINPROC. Son autre particularité est d'être appelée automatiquement en tête de toute exécution de Castem2000. On peut ainsi se prédéfinir des variables (E = 2.732..), des alias (CIRCLE = MOT CERCLE; STIFF = MOT RIGIDITE;...) ou même se prédéfinir systématiquement un maillage avec un modèle et un matériau etc...

Chapitre 4

les OBJETS du langage

Il existe quatre types d'objets qui ont un traitement spécifiques par GIBIANE, il s'agit du "TEXTE qui est une chaîne de caractère à interpréter au moment de son utilisation, de la TABLE qui est un objet fourre-tout, de l'OBJET qui, conformément aux spécifications des langages orientés objets, ne peut être manipulé qu'à travers des méthodes pour lesquelles sont créés des objets METHODE.

4.1 Les TEXTES

Un texte se déclare par l'opérateur TEXTE qui lit une chaîne de caractères.

```
    TT = TEXTE  ' I + 2 ';
```

Lors de l'utilisation, GIBIANE commence par le remplacer par son contenu et seulement après fait l'analyse de la commande.

```
* exemple d'utilisation
   TT = TEXTE  ' I + 2 ';
   I = 3 ;
   XX = TT + 4 ;                            --> XX = 9
```

L'utilisation des objets TEXTES est assez rare et nous ne la conseillons pas. Il est parfois utile pour, dans des procédures, appeler des opérateurs dont le nombre d'arguments est variable suivant le contexte. On pourra ainsi passer en argument de l'opérateur soit un texte vide soit un véritable objet. Par exemple :

```
* exemple d'utilisation
    TTT = TEXTE ' ';
    SI ( situation ou argument existe ) ; TTT =ARG3 ; FINSI;
    AA =  OPERATEUR ARG1 ARG2  TTT;
```

Dans le cas où l'opérateur n'attend que deux arguments le texte TTT est vide, sinon TTT contient directement le troisième argument attendu.

4.2 Les TABLES

Une table est un sac qui peut contenir des objets de n'importe quel type. Les objets contenus dans la table sont repérés par des indices. Ces indices sont eux-mêmes des objets de n'importe quel type.

Pour créer un objet de type TABLE on utilise l'opérateur TABLE. Les objets servant d'indices d'une table sont séparés du nom de la table par un point (".").

L'objet TABLE est souvent utilisé pour regrouper des objets. Ces regroupements permettent de définir une nouvelle structure d'informations, par exemple on peut regrouper toutes les données nécessaires à une procédure dans une table dont les indices sont des mots qui donnent un sens aux objets qui sont dans la table.

Exemple :

```
* exemple d'utilisation
   TA = TABLE;  I = 2  ; A = 'titi' ; Z = 2.25;
   TB = TABLE;
   TA . I = 28 ; TA. DEUXUNQUART = Z;
   TA . x = TB;  TA.25 = 8;
* comme x n'est pas défini il est pris comme un mot contenant " X "

   TA.A = I + 4 ;
* TA.A doit donc etre égal à 6;

   Y = I + TA.I;
* ici Y vaut 30

   TB.3 = 36; TB . 2.25 = 4;  TB . 2 = TA;
   R = TB.2 . 2 + 5 ;
* nous avons R = 33 (TB.2 vaut TA et TA.2 vaut 28, c'est la valeur
* de l'indice qui compte et pas son nom)
```

```
    S = TB.2.25 ; T = TB . 2 . 25 ;
*   S = 4  , T = 8   attention aux points décimaux
   U = TA.titi ;  V = TA . 'titi' ; W = TA . deuxunquart ;
*  l'instruction U =... provoque une erreur, V = 6  et W = 2.25
* (attention au changement minuscule-majuscule)


  TA . ( TB . 2 ) = TA;
* attention TA se contient lui meme sous un indice qui est lui meme!
* cela permet l'écriture stupide suivante
  AE = TA.TA.TA.TA.TA.TA.TA.2;
* ce qui donne AE = 28
```

Attention : L'objet TABLE est le seul objet du monde extérieur qu'une procédure peut modifier. Pour autant il n'est pas besoin de la déclarée en tant que résultat de la procédure.

Un objet table créé dans une procédure doit être donné en tant que résultat, si on veut l'utiliser à l'extérieur.

4.3 Les METHODES

Ces objets sont très ressemblants aux procédures. La seule différence est que l'opérateur de création est DEBMETH (au lieu de DEBPROC) et l'opérateur de retour est FINMETH (au lieu de FINPROC). Sinon la méthode a un argument qui lui arrive systématiquement ; c'est l'objet sur lequel on applique la méthode. Cet objet est reconnu dans la méthode par le caractère pourcent ("%").

L'objet OBJET ressemble, quant à lui, à une table. Pour lui attribuer des informations (des objets) on fera précéder l'indice par le pourcent. Pour appliquer une autre méthode sur l'objet on écrira aussi le pourcent suivi de la méthode.

Il existe trois méthodes qui sont attribuées à tous les objets.
- la méthode METHODE qui permet d'affecter une méthode à un objet
- la méthode HERITE qui permet à un objet de prendre toutes les méthodes d'un autre objet
- la méthode OUBLIER qui permet d'oublier une méthode

4.4 Les objets de type OBJET

La particularité de ces objets est d'être uniquement manipulables par des objets METHODE. L'opérateur OBJET est chargé de créer un objet de type OBJET. Celui-ci est un collecteur, à la manière d'une table. Il faut préciser la classe de l'objet. Cette classe est un constructeur c'est-à-dire une méthode. C'est le premier accès à l'objet. Il est donc nécessaire de définir dans ce constructeur les entrées et les méthodes caractérisant la classe. Nous vous conseillons de lire l'exemple suivant dans lequel les listings produits, encadrés par de "******", ont été introduits juste après la commande de liste e (voir le fichier objet.dgibi qui est fournit avec Castem2000).

```
*
* creation of object of class "complex number".
*
* define first the constructor
DEBMETH COMPLEX;
%REA=FAUX;
%IMA=FAUX;
%METHODE SET_IMA  IMAG;
%METHODE SET_REA  REAL;
%METHODE GET_IMA  GIMAG;
%METHODE GET_REA  GREAL;
FINMETH;


* define standard methods of the class
DEBMETH IMAG  I*'FLOTTANT';
%IMA = I;
'SI' ('EGA'  ('TYPE'  %REA )  'FLOTTANT');
   %METHODE MODULE  MODUL;
'FINSI';
%OUBLIER SET_IMA;
FINMETH;


DEBMETH REAL RR*'FLOTTANT';
%REA = RR;
'SI' ('EGA'  ('TYPE'  %IMA )  'FLOTTANT');
   %METHODE MODULE  MODUL;
'FINSI';
%OUBLIER SET_REA;
```

```
FINMETH;


DEBMETH GIMAG ;
II = %IMA;
'SI' ( 'NEG' ( 'TYPE' II) 'FLOTTANT');
   'MESSAGE' 'The imaginary value is not yet defined';
   'ERREUR' 19;
'FINSI';
FINMETH II;

DEBMETH GREAL;
RR = %REA;
'SI' ( 'NEG' ( 'TYPE' RR) 'FLOTTANT');
   'MESSAGE' 'The real value is not yet defined';
   'ERREUR' 19;
'FINSI';
FINMETH RR;

DEBMETH MODUL;
* It is not neccessary to check that real and imaginary exist
AA= %REA* %REA + ( %IMA*%IMA) ** 0.5;
FINMETH AA;

*
* creation of a COMPLEX OBJET and use
*
COM = OBJET COMPLEX;
list com;

******************** start of  listing ***********************
 Objet de type OBJET de pointeur 25089
        Indice                              Objet
     Type    Valeur                      Type   Valeur
 MOT       CLASSE                 MOT       COMPLEX
 METHODE   METHODE                MOT       METHODE
 METHODE   HERITE                 MOT       HERI
 METHODE   OUBLIER                MOT       ANNU
 MOT       REA                    LOGIQUE   FAUX
 MOT       IMA                    LOGIQUE   FAUX
 METHODE   SET_IMA                PROCEDUR       294
 METHODE   SET_REA                PROCEDUR       295
 METHODE   GET_IMA                PROCEDUR       296
```

```
 METHODE   GET_REA               PROCEDUR       297
********************* end of listing *************************

COM%SET_REA 2.5;
UU=COM%GET_REA; MESS ' real value ' uu;

******************** start of  listing ***********************
     real value   2.5000
********************* end of listing *************************

COM%SET_IMA 3.5;
list com;
******************** start of listing ************************
      Objet de type OBJET de pointeur 25089
              Indice                          Objet
           Type    Valeur                  Type   Valeur
      MOT       CLASSE                  MOT       COMPLEX
      METHODE   METHODE                 MOT       METHODE
      METHODE   HERITE                  MOT       HERI
      METHODE   OUBLIER                 MOT       ANNU
      MOT       REA                     FLOTTANT    .25000000E+01
      MOT       IMA                     FLOTTANT    .35000000E+01
      METHODE   GET_IMA                 PROCEDUR       296
      METHODE   GET_REA                 PROCEDUR       297
      METHODE   MODULE                  PROCEDUR       298
********************* end of listing *************************

ZZ = COM%MODULE ; list ZZ;

******************** start of  listing ***********************
     Réel valant: 4.3012
********************* end of listing *************************

*
* set a new method MULTIplication
*
DEBMETH MULT RR;
'SI' (( 'EGA' ('TYPE'  RR ) 'ENTIER  ')   'OU'
      ( 'EGA' ('TYPE'  RR ) 'FLOTTANT'));
   %REA=%REA*RR;  %IMA=%IMA*RR;
'SINON';
   'SI'  ( 'EGA' ('TYPE'  RR ) 'OBJET   ') ;
      RE = %REA*(RR%GET_REA) -( %IMA * (RR%GET_IMA));
```

```
    IM = %REA*(RR%GET_IMA) +(%IMA *(RR%GET_REA));
    %REA=RE; %IMA=IM;
  'SINON';
    MESS ' Multiplication impossible';
    ERREUR 19;
  'FINSI';
'FINSI';
FINMETH;
*
*  add the new method to the object COM
*
COM%METHODE MULTI MULT;
list com;

******************** start of  listing ***********************
     Objet de type OBJET de pointeur 25089
              Indice                          Objet
           Type    Valeur                  Type   Valeur
      MOT       CLASSE                MOT       COMPLEX
      METHODE   METHODE               MOT       METHODE
      METHODE   HERITE                MOT       HERI
      METHODE   OUBLIER               MOT       ANNU
      MOT       REA                   FLOTTANT    .25000000E+01
      MOT       IMA                   FLOTTANT    .35000000E+01
      METHODE   GET_IMA               PROCEDUR       296
      METHODE   GET_REA               PROCEDUR       297
      METHODE   MODULE                PROCEDUR       298
      METHODE   MULTI                 PROCEDUR       299
********************* end of listing *************************

COM%MULTI 3.5;
list com;

******************** start of  listing ***********************
    Objet de type OBJET de pointeur 25089
             Indice                          Objet
           Type    Valeur                  Type   Valeur
      MOT       CLASSE                MOT       COMPLEX
      METHODE   METHODE               MOT       METHODE
      METHODE   HERITE                MOT       HERI
      METHODE   OUBLIER               MOT       ANNU
      MOT       REA                   FLOTTANT    .87500000E+01
      MOT       IMA                   FLOTTANT    .12250000E+02
```

```
      METHODE   GET_IMA                 PROCEDUR       296
      METHODE   GET_REA                 PROCEDUR       297
      METHODE   MODULE                  PROCEDUR       298
      METHODE   MULTI                   PROCEDUR       299
********************* end of listing *************************

aa = OBJET complex;
aa%set_ima 2;
aa%set_rea 2;
com%multi aa;
list com;

******************** start of  listing ***********************
      Objet de type OBJET de pointeur 25089
              Indice                              Objet
           Type    Valeur                      Type   Valeur
      MOT       CLASSE                  MOT       COMPLEX
      METHODE   METHODE                 MOT       METHODE
      METHODE   HERITE                  MOT       HERI
      METHODE   OUBLIER                 MOT       ANNU
      MOT       REA                     FLOTTANT   -.70000000E+01
      MOT       IMA                     FLOTTANT    .42000000E+02
      METHODE   GET_IMA                 PROCEDUR       296
      METHODE   GET_REA                 PROCEDUR       297
      METHODE   MODULE                  PROCEDUR       298
      METHODE   MULTI                   PROCEDUR       299
********************* end of listing *************************

yy = com%GET_REA; xx = COM%GET_IMA;

* aa  will get the MULTI method  by
* inherits between objects

aa%HERITE COM;
list aa;

******************** start of  listing ***********************
      Objet de type OBJET de pointeur 25259
              Indice                              Objet
           Type    Valeur                      Type   Valeur
      MOT       CLASSE                  MOT       COMPLEX
      METHODE   METHODE                 MOT       METHODE
      METHODE   HERITE                  MOT       HERI
```

```
      METHODE   OUBLIER                 MOT       ANNU
      MOT       REA                     FLOTTANT    .20000000E+01
      MOT       IMA                     FLOTTANT    .20000000E+01
      METHODE   GET_IMA                 PROCEDUR       296
      METHODE   GET_REA                 PROCEDUR       297
      METHODE   MODULE                  PROCEDUR       298
      METHODE   MULTI                   PROCEDUR       299
********************* end of listing *************************

'SI' (( ABS( YY + 7.00))  > 1.e-5); ERREUR 5;FINSI;
'SI'  (( ABS( XX - 42.0))  > 1.e-5) ; erreur 5; finsi;
FIN;
```

A l'intérieur d'une méthode on applique une autre méthode sur l'objet par :

% METH1 ARG1 ARG2 ..

où METH1 est le nom de la méthode tandis qu'à l'extérieur d'une méthode l'utilisateur peut appliquer une méthode sur un objet par :

NOMOBJ%METH1 ARG1 ARG2 ..

L'emploi des méthodes est encore à découvrir, elles permettent sûrement un contrôle syntaxique beaucoup plus précis que les procédures. La réécriture de certaines procédures de Castem2000 en METHODES et OBJET est envisagée.

Chapitre 5

Bibliographie

- [1] P.VERPEAUX, T.CHARRAS, A.MILLARD : "Castem2000 : une approche moderne du calcul des structures" Calcul des Structures et Intelligences artificielle, p261. Pluralis 1988

- [2] Th.CHARRAS, JJ.LAUTARD, MF.ROBEAU, P.VERPEAUX "Langage GIBIANE : Définition" Rapport DEMT/SERMA/89-

- [3] Th.CHARRAS : "Castem2000 Gibiane" Rapport DMT 94/154

- [4] Th.CHARRAS : "Développer dans Castem2000 : LES OBJETS" Rapport DMT/96-171

- [5] Th.CHARRAS : "Développer dans Castem2000 : ESOPE" Rapport DMT/96-172

- [6] Th.CHARRAS : "Développer dans Castem2000 : LES OPERATEURS" Rapport DMT/96-173

- [7] A.CONSTANTINESCU, M.DRAGON, J.KICHENIN : "Programming Engineering Applications Using the Object Oriented FEM code Castem2000." A paraître SIAM Workshop on Object Oriented Methods for Interoperable Scientific and Engineering Computing 1998 Yorktown Heights, New York