Développer dans CASTEM2000 : ESOPE

T.CHARRAS

0.1 Introduction

Le présent manuel n'a pas la prétention de tout dire sur ESOPE, si le lecteur veut en savoir plus il pourra consulter avec profit les rapports :

- Manuel d'utilisation ESOPE V.10.0 par P.VERPEAUX (DMT/93.617)
- Esope 10.1 dit Esope optimisé par P.VERPEAUX (DMT/94.710)

Notre objectif est juste de décrire les fonctions d'ESOPE utilisées dans le code CASTEM2000. En cela ce manuel est presque complet.

0.2 Présentation générale

ESOPE est une extension du langage fortran 77, il est donc nécessaire de posséder correctement le langage fortran avant de vouloir faire de l'ESOPE. L'objectif est de faciliter la gestion des données et de permettre la notion d'objet par la structuration des données.

Le code source, écrit en ESOPE, est dans un premier temps traduit en fortran 77 avant d'etre envoyé en compilation. La traduction en fortran se fait en éxécutant un petit programme appelé lui aussi ESOPE.

La partie d'ESOPE qui gère le transfert des données mémoire vive ¡-¿ mémoire de stockage ne sera pas exposé ici. C'est la bibliothèque GEMAT, incluse dans ESOPE, qui s'en charge et rares sont les applications où le programmeur doit faire un travail spécial.

Pour disposer d'un langage orienté OBJET il faut associer un ensemble de données à une variable. Il manque au fortran la notion d'ensemble de données et l'entité appelée SEGMENT a été ajoutée. Celle-ci répond aux deux exigences précitées :

- c'est un regroupement de variables fortran défini par le programmeur.
- elle est réferencée par une seule variable appelée POINTEUR. La connaissance du pointeur suffit pour accéder à toutes les variables contenues dans la structure de données .

0.3 Manipulation des structures de données

Peu d'instructions ont été ajoutées à celles du fortran77. Elles servent à manipuler et utiliser les SEGMENTS.

Comme en fortran, on est conduit à avoir une instruction déclarative du SEGMENT puis des instructions qui agissent sur le SEGMENT. Il s'agit avant tout de :

- créer un segment (ou INItialiser) SEGINI
- SUPprimer un segment SEGSUP
- DESactiver un segment SEGDES
- ACTiver un segment SEGACT

– ajuster la taille d'un segment SEGADJ

0.3.1 Déclaration d'une SEGMENT

Avant la première instruction éxécutable d'un sous-programme fortran, il faut déclarer les SEGMENTS qui seront utilisés.

Un SEGMENT peut contenir autant de variables fortran que nécessaire et de tous les types fortran admis. La déclaration se fait comme suit :

```
SEGMENT MONSEG
  INTEGER IJK,NOMB(LL,LC),JLC
  REAL X,XX(N)
  REAL*8 Y,YY(3,MM)
  CHARACTER*NBCA ICHA(8,IK),CHAI
ENDSEGMENT
```

La déclaration est comprise entre les mots SEGMENT et ENDSEGMENT. Rappelons que nous sommes en fortran et que ces instructions doivent etre frappées au delà de la septième colonne.

MONSEG est le nom de la classe de la structure matérialisée par ce segment. Une structure appartenant à la classe MONSEG contiendra un entier IJK, un tableau d'entier NOMB, un entier JLC .... et enfin une chaine de NBCA caractères appelée CHAI.

Dans cet exemple IJK,NOMB et JLC sont des noms de variables de type fortran INTEGER, X ET XX sont de type RELLES, etc... Pour des raisons d'architecture de machines il est préférable de ne pas employer DOUBLE PRECISION mais REAL*8. ICHA et CHAI sont des variables contenant des chaines de NBCA caractères.

Remarquons que certaines dimensions de tableaux sont fixées tandis que d'autres font référence à des noms de variables entières (il est d'usage dans CASTEM2000 de respecter les types implicites du fortran). C'est les valeurs de ces variables fortran au moment de l'initialisation du segment qui définiront les dimensions prises par les tableaux. Les variables dimensionnantes sont dans notre exemple LL,LC,N,MM,IK, elles ne sont pas incluses dans le SEGMENT, ne servant qu'au moment de sa création. Une instruction particulière permettra, si besoin est, de connaitre la longueur d'un tableau dans un SEGMENT déjà crée.

0.3.2 Création et initialisation d'un SEGMENT

Plaçons nous encore à l'intérieur d'un sous-programme fortran. Après y avoir déclaré le segment par l'instruction SEGMENT...ENDSEGMENT il faut définir une variable qui référencera la structure instanciée : la connaissance de cette variable donnant accés à toute la structure. Pour cela un nouveau type de variable fortran est inventée : c'est la variable pointeur dont la déclaration est :

POINTEUR nompointeur.nomsegment

par exemple, pour permettre l'instanciation de la classe MONSEG, on déclare :

POINTEUR MONS1.MONSEG,MONS2.MONSEG,....

Dans cet exemple MONS1 et/ou MONS2 permettent de référencer une structure de la classe MONSEG. MONS1 et MONS2 deviennent des variables fortran de type Integer une fois la traduction Esope-¿Fortran faite.

La création d'une sructure de classe MONSEG est faite par l'appel à la fonction SEGINI en précisant la variable pointeur utilisée.

SEGINI MONS1

Dans notre exemple cela donne :

```
subroutine sp1 (......)
segment monseg
   integer ijk,nomb(ll,lc),jlc
   real x,xx(n)
   real*8 y,yy(3,mm)
   character*nbca icha(8,ik),chai
endsegment
POINTEUR MONS1.MONSEG,MONS2.MONSEG
   .
   .
SEGINI MONS1
   .
   .
return
end
```

Il faut, bien évidemment, que les variables dimensionnantes du segment soient définies préalablement à l'instruction SEGINI (sinon le résultat dépend du compilateur !). Nous aurions pu faire le jeu d'instructions :

```
subroutine sp1 (LL,LC......)
segment monseg
   integer ijk,nomb(ll,lc),jlc
   real x,xx(n)
   real*8 y,yy(3,mm)
   character*nbca icha(8,ik),chai
endsegment
pointeur mons1.monseg
   .
   .
N=5
MM=3
IK=12
NBCA=4
SEGINI MONS1
MONS1.XX(3)=...
MONS1.ICHA(5,2)='BIEN'
   .
   .
return
end
```

Dans lequel le tableau YY est dimensionné à 3 lignes et 3 colonnes, tandis que ICHA est un tableau de chaines de 4 caractères qui à 8 lignes et 12 colonnes.

Une fois l'instruction SEGINI éxécutée et tant que l'on reste dans le sous-programme, tous les éléments du SEGMENT peuvent etre considérés par le programmeur comme des variables fortran normales. Pour les atteindres il faut rappeler qu'elles appartiennent à la structure référencée par la variable pointeur MONS1.

Remarques :

- à la création, toutes les variables contenues dans le segment sont mises à zéro.
- il est possible de créer un segment en en recopiant un autre (confère chapitre 4).

ESOPE offre la possibilité de libérer la place mémoire occupée par le segment. on peut soit supprimer le segment, soit le désactiver.

0.3.3 Suppression d'un SEGMENT

L'ordre de déclaration du segment étant en tete du sous-programme on peut à tout moment éxécuter l'instruction SEGSUP dont la syntaxe est :

SEGSUP nom-variable-pointeur

Dans notre exemple, supposons que la structure de données MONSEG instanciée par MONS1 ne serve qu'à l'intèrieur du sous-programme SP1, alors le segment doit etre supprimé avant de quitter le sous-programme.

```
subroutine sp1 (ll,lc......)
segment monseg
   integer ijk,nomb(ll,lc),jlc
   real x,xx(n)
   real*8 y,yy(3,mm)
   character*nbca icha(8,ik),chai
endsegment
pointeur mons1.monseg
   .
   .
n=5
mm=3
ik=12
nbca=4
segini mons1
mons1.xx(3)=...
mons1.icha(5,2)='BIEN'
   .
   .
SEGSUP MONS1
.
return
end
```

0.3.4 Désactivation d'un SEGMENT

En règle générale, au sortir d'un sous-programme fortran ou dés que possible, on désactive les segments pour rendre utilisable la mémoire vive qu'ils occupent. Cela se fait par l'instruction :

SEGDES nom-variable-pointeur

La mise en attente sur disque de travail ne se fera que si les demandes en mémoire le nécessite et si la place sur disque le permet. De toutes les façons une fois un segment désactivé, il aura besoin d'etre réactivé avant de s'en servir à nouveau.

0.3.5 Activation d'un SEGMENT

Supposons que le segment MONS1 ait été défini puis désactivé dans le sous-programme fortran sp1 qui appelle sp2 et que l'on désire se servir de MONS1 dans ce dernier sous-programme. Sp2 devra contenir l'ordre déclaratif de la classe du segment et de plus il devra connaitre la valeur de la variable pointeur instanciant MONSEG. On peut la lui fournir soit par COMMON soit par argument (soit par un autre segment) puisqu'elle est avant tout de type Integer. Pour activer le segment if faut utiliser l'instruction :

SEGACT nom-variable-pointeur

Cette activation se fait en lecture seule, pour pouvoir modifier les valeurs contenues dans le segment, c'est à dire avoir accés en lecture/écriture, il faut terminer l'instruction par *MOD. L'instruction devient :

SEGACT nom-variable-pointeur*MOD

Par exemple on peut envisager la séquence suivante :

```
subroutine sp1 (ll,lc,...)
segment monseg
   integer ijk,nomb(ll,lc),jlc
   real x,xx(n)
   real*8 y,yy(3,mm)
   character*nbca icha(8,ik),chai
endsegment
pointeur mons1.monseg
   .
   .
segini mons1
INTER=MONS1
SEGDES MONS1
call sp2(INTER,... )
return
end


subroutine sp2(INTER,....)
```

```
segment monseg
   integer ijk,nomb(ll,lc),jlc
   real x,xx(n)
   real*8 y,yy(3,mm)
   character*nbca icha(8,ik),chai
endsegment
pointeur mons1.monseg
   .
   .
MONS1=INTER
SEGACT MONS1
AZ=MONS1.XX(2)
SEGDES MONS1
   .
   .
SEGACT MONS1*MOD
MONS1.XX(1)=MONS1.XX(1)*2
SEGDES MONS1
return
end
```

Une fois un segment activé, toutes les variables qu'il contient sont accessibles en lecture au meme titre que les autres variables fortran, par contre il n'est pas possible de les modifier. En effet ,il n'est pas normal de vouloir modifier un objet qui existe déjà et les structures de mémoires servant transitoirement dans un opérateur doivent etre crées en début d'opérateur et détruites en fin d'opérateur. Il est malgré tout possible de modifier un segment desactivé puis activé en faisant suivre le nom du pointeur à activer de *MOD (SEGACT MONS1*MOD).

Rappelons que LL,LC... ne sont pas des variables contenues dans le segment, l'instruction particulière suivante sert à connaitre les dimensions des tableaux

```
LHJ=NOMS1.NOMB(/i)
```

dans LHJ se trouve la valeur de la ième dimension du tableau NOMB.Les tableaux de chaines de caractères sont traités différemment. NOMS1.ICHA(/1) contient la longueur des chaines du tableau ICHA soit 4 dans notre exemple. NOMS1.ICHA(/i) contient la i-1 ème dimension du tableau ICHA.

Dans un sous-programme, il est possible de connaitre la dimension d'un tableau et Esope permet de changer sa dimension.

0.3.6 Ajustement d'un SEGMENT

L'instruction :

SEGADJ nom-variable-pointeur

s'emploie comme SEGINI. c'est à dire que **toutes** les variables dimensionnantes des tableaux du segment doivent etre définies préalablement à l'appel.

ESOPE, suivant les demandes tronquera ou agrandira les tableaux en conservant les valeurs des variables qui existent avant et après l'instruction.

Exemple :

```
subroutine sp1 (ll,lc,n,mm,inter,...)
segment monseg
   integer ijk,nomb(ll,lc),jlc
   real x,xx(n)
   real*8 y,yy(3,mm)
   character*nbca icha(8,ik),chai
endsegment
pointeur mons1.monseg
   .
   .
MONS1=INTER
N=36
IK=18
SEGADJ MONS1
```

0.4 Les structures de la meme classe

Pour pouvoir travailler simultanéement sur plusieurs structures de meme type, il faut avoir plusieurs variables-pointeurs : MONS1 et des autres. L'appartenance des variables à une structure ou à l'autre se faisant par le préfixe. Dans l'instruction POINTEUR, plusieurs nom-de-variables-pointeur peuvent etre définis.

Pour travailler séquentiellement sur des structures de la meme classe on peut conserver la valeur de la variable-pointeur dans une variable fortran de type INTEGER. Par exemple :

```
subroutine sp1 (ll,lc,...)
segment monseg
```

```
    integer ijk,nomb(ll,lc),jlc
    real x,xx(n)
    real*8 y,yy(3,mm)
    character*nbca icha(8,ik),chai
  endsegment
  pointeur mons1.monseg
  INTEGER IUYT,IUYR
    .
    .
  MM=4
  SEGINI MONS1
    .
  MONS1.XX(3)= 36
    .
    .
  IUYT = MONS1
  MM=8
  SEGINI MONS1
  IUYR = MONS1
    .
  MONS1.XX(3)= 10
    .
    .
  MONS1 = IUYT
  YTR = MONS1.XX(3)
** ytr vaut dans cet exemple 36
  SEGDES MONS1
    .
    .
  MONS1 = IUYR
  TEN = MONS1.XX(3)
** ten vaut dans cet exemple 10
  SEGDES MONS1
  etc....
```

0.5 Remarques

Nous allons donner une liste de remarques et de conseils.

1. Pour avoir dans un sous-programme des variables incluses dans un segment, il est souvent préférable de passer en argument le pointeur de tout le segment. Ceci devient obligatoire, si on veut de nouveau utiliser des instructions SEG... dans le sous-programme.

2. Il ne faut pas utiliser comme variable dimensionnante d'un segment une variable appartenant à ce meme segment.
3. On peut activer ou desactiver ou supprimer plusieurs segment par la meme instruction en séparant le nom des variables-pointeurs par une virgule.

   SEGDES MONS1,MONS2,...

4. Si on veut utiliser fréquemment la notion de segment ajustable il est bon de ne mettre qu'un seul tableau dans le segment.
5. Il est d'usage de ne pas polluer la mèmoire en supprimant les segments de travail temporaires et en désactivant les segments qui ne servent plus localement dans ce sous-programme ou dans l'opérateur.
6. L'existence des segments évite l'emploi de COMMON de travail.
7. On peut dupliquer un segment existant. Pour dupliquer le segment MONS1 en le recopiant dans MONS2 on fera :

   SEGINI,MONS2=MONS1

8. En lisant le source Esope de Castem2000 on remarque des instructions, commençant en colonne 1, du type :

   -INC S***.INC**

   Cette instruction demande de remplacer la ligne par le contenu du fichier S*****.INC. Tous les systèmes d'exploitations ne permettent pas à ESOPE de le faire lui-meme, et dans le cas contraire, il faut faire le travail soi meme avant de soumettre le source à la traduction fortran 77. Nous avons procédé ainsi pour deux raisons principales :
   - La qualité est garantie en ne réécrivant pas les instructions contenues dans le fichier.
   - Un fichier correspond souvent à une structure d'OBJET et sa déclaration sera strictement identique dans tous les sous-programmes l'utilisant.
9. Si vous avez des remarques supplémentaires, prenez la peine de nous les signaler, nous vous en serions reconnaissant.

0.6 Simplification

0.6.1 Nom-de-variable-pointeur

La notion de classe d'objet et de pointeur permettant l'instanciation d'un objet particulier de la classe n'avait pas était complétement dégagée au moment de la spécification du langage Esope. D'un autre coté, pour identifier une variable appartenant à un objet instancié il faut rappeler le nom-de-variable-pointeur comme préfixe de la variable. Une simplification possible est d'attribuer un nom par défaut à ce nom-de-variable-pointeur. La solution la plus simple est de prendre le nom de la classe de la structure. Ainsi, dans les exemples précédents, MONSEG est la variable qui identifie la classe de la structure et qui peut servir de nom par défaut. On peut construire l'exemple suivant :

```
      subroutine sp1 (ll,lc,...)
      segment monseg
        integer ijk,nomb(ll,lc),jlc
        real x,xx(n)
        real*8 y,yy(3,mm)
        character*nbca icha(8,ik),chai
      endsegment
        .
        .
      mm=4
      SEGINI MONSEG
        .
      MONSEG.XX(3)= 36
**** ou encore directement
      XX(3)= 36
        .
        .
      SEGDES MONSEG
        .
        .
      SEGACT MONSEG
etc....
```

0.6.2 Exemple d'utilisation complet

Dans ce dernier exemple nous cherchons à donner un apercu de l'utilisation des segments dans Castem2000. Il s'agit de construire le triangle de Pascal (le pivot initial n'est pas forcement égal à 1). Chaque ligne est représentée par un segment. On veut pouvoir avoir accés directement à la n-iéme ligne. La strucure de notre objet triangle de Pascal peut etre :

```
    SEGMENT MPASC
      INTEGER ILIGN(L)
    ENDSEGMENT
    SEGMENT LIGN
      REAL*8 XLIGN(M)
```

```
ENDSEGMENT
POINTEUR LIG1.LIGN
```

Le tableau ILIGN contient la valeur des pointeurs sur le segment LIGN et les valeurs sont dans le tableau XLIGN

Le premier sous-programme crée un triangle de pascal avec XUN comme pivot et avec LI lignes, et le second sous programme écrira un objet triangle de Pascal reçu en argument.

```
   SUBROUTINE SP11 (XUN,LI,IRET)
   IMPLICIT REAL*8(A-H,O-Z)
   SEGMENT MPASC
     INTEGER ILIGN(L)
   ENDSEGMENT
   SEGMENT LIGN
     REAL*8 XLIGN(M)
   ENDSEGMENT
   POINTEUR LIG1.LIGN
   L = LI
   SEGINI MPASC
   IRET = MPASC
   IF ( LI.EQ.0) THEN
   SEGDES MPASC
   RETURN
   ENDIF
   M=1
   SEGINI LIGN
   XLIGN(1)= XUN
   ILIGN(1)=LIGN
   DO 1 I=2,LI
   M= I
   SEGINI LIG1
   ILIGN(I)= LIG1
   LIG1.XLIGN(1) = XUN
   LIG1.XLIGN(I) = XUN
   DO 2 KO = 1,I-2
   LIG1.XLIGN(KO+1) = XLIGN(KO) + XLIGN(KO+1)
 2 CONTINUE
   SEGDES LIGN
   LIGN=LIG1
 1 CONTINUE
   SEGDES LIGN
   SEGDES MPASC
```

```
      RETURN
      END


      SUBROUTINE SP2(IPASC)
      IMPLICIT REAL*8(A-H,O-Z)
      SEGMENT MPASC
        INTEGER ILIGN(L)
      ENDSEGMENT
      SEGMENT LIGN
        REAL*8 XLIGN(M)
      ENDSEGMENT
      POINTEUR LIG1.LIGN
      MPASC=IPASC
      SEGACT MPASC
      DO 1 I=1,ILIGN(/1)
      LIGN= ILIGN(I)
      SEGACT LIGN
      WRITE(6,*) (XLIGN(K),K=1,XLIGN(/1))
      SEGDES LIGN
    1 CONTINUE
      SEGDES MPASC
      RETURN
      END
```