Universite Batna 2
Faculté de Technologie
Département D'hydraulique

Informatique 3

TP 5 : les Instructions de base et Instructions de contrôle (Boucles for et While, Instructions if et switch)

5.1 INTRODUCTION

Cette partie traite des instructions de base du langage FORTRAN permettant de créer les structures séquentielles, sélectives et itératives. Les instructions nécessaires à la structuration des sous-programmes.

5.2 STRUCTURES SÉQUENTIELLES

Une partie de programme à structure séquentielle est une suite d'instructions FORTRAN traitée ligne après ligne, sans sauts ni répétitions. L'un des reproches fait au langage est l'utilisation abusive de branchements et d'étiquettes afin d'atteindre des instructions déterminées dans le programme.

5.2.1 INSTRUCTION DATA

L'instruction data est une instruction non exécutable qui sert à initialiser les variables par le compilateur, d'où économie de code et de temps par rapport à l'affectation pendant l'exécution d'un programme.

La valeur affectée à une variable ou à un élément de variable indicée par cette instruction peut être modifiée par la suite en cours de programme.

data *varlist1* / *conslist1* / [,*varlist2* / *conslist2* / , . . .]

avec :

varlisti liste des identificateurs de variables simples ou de tableaux, éléments de tableaux ou sous-chaînes de caractères,

conslisti liste de constantes numériques, logiques ou chaînes de caractères, qui doit correspondre en nombre, type et ordre à la liste des identificateurs qui la précède immédiatement. Les constantes sont séparées par la virgule et la liste est placée entre / ... / .

Chaque unité de programme peut comporter un nombre quelconque d'instructions **data** , la même variable ne pouvant évidemment figurer qu'une seule fois. Il est de tradition de placer les instructions **data** après les déclarations des divers types, bien que ces instructions puissent se placer à n'importe quel endroit entre les éventuelles fonctions-formules et l'instruction **end** .

Les listes ***conslisti*** peuvent contenir des variables indicées par des constantes. Un tableau sans indice représente tous les éléments dans l'ordre de rangement en mémoire. Lorsque plusieurs constantes consécutives sont égales, on peut l'indiquer par un facteur de répétition sous la forme : nombre * constante .

Exemples :

```
real*4                  table(10,4),x(8),y(8)
character*18 titre

.....
data table /40*1.0/
data x /0.2,0.4,0.5,0.8,1.0,1.2,1.4,1.6/
data y /2.0,1.0,0.0,1.0,2.0,3.0,4.0,5.0/
data titre /'Ecole d''Ingénieurs'/
```

La première instruction **data** affecte 40 fois 1.0 à tous les éléments de la variable indicée table (10,4). Les deux suivantes affectent des valeurs numériques aux deux vecteurs x (8) et y (8) et la dernière une chaîne de caractères à titre *18.

5.2.2 AFFECTATION ARITHMÉTIQUE

L'instruction d'affectation arithmétique assigne à la variable située à gauche du caractère = la valeur numérique de l'expression située à droite du caractère = . La syntaxe générale est :

var = expr

avec :

var une variable simple ou un élément de variable indicée, de type numérique,
expr une expression arithmétique.

Le caractère **=** placé entre ***var*** et ***expr*** ne signifie pas du tout "égal à" au sens mathématique usuel, mais veut dire : affecte à la variable ***var*** la valeur de l'expression arithmétique ***expr*** .

Pour que la variable ***var*** puisse être affectée correctement, il faut que tous les opérandes de l'expression ***expr*** soient définis au moment de l'évaluation. Si la valeur de l'expression est du même type que la variable à affecter, l'instruction transmet immédiatement cette valeur. Par contre, si l'expression est d'un type différent comme c'est très souvent le cas, l'expression est tout d'abord calculée suivant les règles données précédemment, puis transformée ensuite dans le type de la variable à affecter et finalement transmise à *var* . La valeur de l'expression arithmétique doit pouvoir se représenter dans le type adopté pour ***var*** . Le tableau 3.1 donne les règles de transformation pour les types fondamentaux du langage.

Tableau 3.1

RÈGLES DE CONVERSION DANS L'AFFECTATION ARITHMÉTIQUE

V variable ou élément de tableau	Expression arithmétique ***E***			
	integer ou logical	Real	double precision	complex
integer ou logical	Assigne ***E*** à ***V***	Transforme ***E*** en entier et assigne à ***V***	Transforme ***E*** en entier et assigne à ***V***	Transforme PR de ***E*** en entier et affecte à ***V*** ; PI et ***E*** est perdue
real	Génère la fraction 0.0 dans ***E*** ; assigne ***E*** à ***V***	Assigne PPS de ***E*** à ***V*** ; DPS de ***V*** est des zéros	PPS de ***E*** est arrondie, DPS est perdue ; assigne ***E*** à ***V***	Assigne PR de ***E*** à ***V*** ; PI est perdue
Double Precision	Génère la partie 0.0 dans ***E*** ; DPS de ***V*** est des zéros	Assigne PPS de ***E*** à ***V*** ; DPS de ***V*** est des zéros	Assigne ***E*** à ***V***	Assigne la partie réelle de ***E*** à ***V*** et complète par des zéros ; PI est perdue
complex	Génère la partie 0.0 dans ***E*** ; Affecte à PR	Affecte ***E*** à la PR de ***V*** ; PI de ***V*** est égale à zéro	Affecte PPS de ***E*** à ***V*** ; DPS est arrondie ; PI de ***V*** est perdue	Assigne ***E*** à ***V***

Dans ce tableau, les lettres majuscules signifient :

PR partie réelle de la grandeur complexe,
PI partie imaginaire de la grandeur complexe,
PS partie significative,
PPS première partie significative,
DPS dernière partie significative, ***E*** expression arithmétique,
V variable à affecter.

Exemples :

```
integer*2 index,j real*4
                  a,b,beta,c
.....
beta      = -1.0/(2.5*a*b**c+a*a) index =
index+1
```

L'affectation d'une grandeur complexe à une variable ou à un élément de variable indicée de type **complex** fait intervenir les deux parties réelle et imaginaire, séparée par une virgule et encadrées de parenthèses.

Exemples :

```
complex*8 correc,rare
.....
correc = (0.12324,5.56789) rare  =
(987.654,1.3579E4)
```

5.2.3 AFFECTATION LOGIQUE

L'affectation logique évalue tout d'abord l'expression logique située à droite du caractère = et assigne le résultat logique **.true.** ou **.false.** à la variable simple logique ou l'élément d'une variable indicée logique placé à gauche du caractère = . La syntaxe est :

varl = exprl

avec :
varl une variable ou un élément de tableau de type **logical**,
exprl une expression logique.

La variable ou l'élément de tableau, situé à gauche du caractère = doit être absolument de type **logical** . Tous les opérandes de l'expression logique doivent être définis avant l'évaluation logique.

Exemples :

```
logical          logi,lmax,lpage
integer*2 job
real*4           a,b,c
.....
lpage = .true.
logi      = job .gt. 100 .and. .not. lpage
lmax      = (a .lt. b) .or. ((a .gt. b) .and. (a .le. c))
```

En complément de cette programmation, certaines paires de parenthèses ont été introduites dans le dernier exemple afin de faciliter la lecture des conditions logiques.

5.2.4 AFFECTATION D'UNE CHAÎNE DE CARACTÈRES

L'instruction d'affectation alphanumérique assigne la valeur d'une expression de type **character** à une variable, à une sous-chaîne de caractères ou à un élément de tableau de type **character** . La syntaxe générale est :

vara = expra

avec :
vara une variable, un élément de tableau ou une sous-chaîne de type **character**,
expra une expression de type **character**.

*

Exemple 1

```
*
*  Affectation d'une chaîne de caractères
*  et modification de cette chaîne
*
          program exem02
          character*24 gamme
*
          gamme = 'do re mi fa sol la si do'
          print *,'La gamme complète est : ',gamme gamme(1:2)
                                  = 'ut'
          gamme(23:24) = gamme(1:2)
          print *,'La nouvelle gamme est : ',gamme end
```

Résultat de l'exécution du programme :

La gamme complète est : do re mi fa sol la si do La nouvelle gamme est : ut re mi fa sol la si ut

L'instruction **print** utilisée ici permet d'afficher des données à l'écran : constantes alphanumériques ou numériques, variables de types divers.

L'affectation de caractères à une sous-chaîne, à partir de la même variable, s'effectue caractère après caractère. Si les deux termes à gauche et à droite de l'affectation traitent les mêmes caractères, il est certain qu'il y ait quelques problèmes de transmission correcte de ces caractères. L'exemple suivant montre précisément ce genre d'opération sur des sous-chaînes de la même variable.

Exemple 2

```
*  Affectation d'une chaîne de caractères
*  et modification partielle de la chaîne
*
          program exem03
          character*26 alfab
*
          alfab = 'abcdefghijklmnopqrstuvwxyz' print *,'Alphabet
          initial : ',alfab alfab(7:10) = alfab(9:12)
          print *,'Alphabet modifié : ',alfab alfab(7:10) =
          alfab(6:9)
          print *,'Alphabet modifié : ',alfab end
```

Résultat de l'exécution du programme

Alphabet initial : abcdefghijklmnopqrstuvwxyz Alphabet modifié : abcdefijklklmnopqrstuvwxyz Alphabet modifié : abcdeffiikklmnopqrstuvwxyz

Les modifications apportées portent sur les caractères numérotés 7 à 10 . La première modification s'effectue correctement tandis que la seconde provoque une duplication des caractères.

5.2.5 INSTRUCTION : CONTINUE

Cette instruction spécifie le transfert du déroulement du programme à la prochaine instruction exécutable suivant **continue** , dans le cheminement du programme. La syntaxe est simplement :

continue

C'est une instruction exécutable, mais sans effet sur les données du programme. Elle s'utilise principalement comme instruction de fin de boucle **do** , même si la dernière instruction exécutable de la fin de boucle n'est pas une instruction prohibée par cette structure. Elle est alors accompagnée d'une étiquette.

Exemple :

```
      do 10 i=1,10
           a(i)=0.0
10    continue
```

5.5.6 INSTRUCTION : PAUSE

L'instruction **pause** interrompt temporairement l'exécution du programme afin de permettre quelque action de la part de l'utilisateur et affiche un message à l'écran. La syntaxe est :

pause [*n*]

avec :
n une constante de type **character** ou une constante entière de 0 à 99999.

L'instruction provoque l'affichage d'un message au terminal, dépendant du compilateur, mais contenant en tout cas le mot pause . Si l'argument ***n*** est présent, il est affiché à l'écran. Pour poursuivre le programme, il suffit de presser sur la touche <RET> dans la plupart des implémentations. Le programme continue alors à l'instruction exécutable suivant immédiatement l'instruction **pause**.

Exemple

```
*   Ce programme calcule et affiche des nombres entiers
*   entre 1 et une valeur choisie par l'utilisateur
*   en bloc de 20 valeurs avec arrêt du programme
*
          program exem05 intrinsic
          float,sqrt
          integer*4 i,icompt,inomb real*4
                              anomb,racine
*
          icompt=0
*
          print *,'Donnez un nombre réel positif :' read *,anomb
          inomb = anomb
*
          print *
          do 10 i=1,inomb
               racine = sqrt(float(i))
               write (*,'(1x,a,i5,a,f10.5)')
     +       'La racine carrée de ',i,' est : ',racine icompt = icompt+1
               if ((icompt .eq. 20) .and. (i .ne. inomb)) then print *
                    pause 'Veuillez presser sur <RET>' print *
                    icompt = 0 end
               if
 10       continue
*
          end
```

Résultat de l'exécution du programme :

```
Donnez un nombre réel positif :
99.9
La racine carrée de          1 est :          1.00000
La racine carrée de          2 est :          1.41421
.....
La racine carrée de         20 est :          4.47214

Veuillez presser sur <RET>

etc.
```

Dans cet exemple, l'écriture du texte s'effectue au moyen de l'instruction **write** accompagnée d'une mise en forme. Cette mise en forme est donnée par une constante chaîne de caractères dans laquelle :

1x saut du caractère de contrôle
a texte alphanumérique
i5 écriture d'une entité de type **integer** dans 5 colonnes
f10.5 écriture d'une entité réelle dans 10 colonnes, virgule fixe, avec 5 décimales.

5.2.7 STRUCTURES SÉLECTIVES

Les instructions d'un programme FORTRAN sont exécutées ligne après ligne. Il est souvent désirable de pouvoir rompre le déroulement séquentiel d'un programme au moyen d'ordres de contrôle. Il existe deux catégories d'ordres de contrôle :

1. Ruptures de séquence inconditionnelles,
2. Ruptures de séquence conditionnelles.

Ces instructions nécessitent la présence d'étiquettes, entité que nous voulons si possible laisser de côté.

- #### SAUTS INCONDITIONNELS

Les ruptures inconditionnelles permettent de rompre le déroulement séquentiel du programme vers une ou plusieurs instructions étiquetées, suivant la valeur prise par une expression.

GO TO INCONDITIONNEL

Cette instruction transfère l'exécution du programme vers l'instruction portant le numéro d'étiquette spécifié. La syntaxe est :

go to ***etiq***

avec :
etiq une constante entière, comprise entre 1 et 99999, correspondant à l'étiquette d'une instruction exécutable dans la même unité de programme.

Le déroulement du programme s'effectue de la ligne contenant l'instruction **go to *etiq*** à la ligne débutant par l'étiquette ***etiq*** . Cette instruction peut aussi s'écrire en un seul mot : **goto** .

Exemples :

```
go to 55
.....
go to 875
```

- #### GO TO IMPOSÉ

Deux instructions au moins sont nécessaires pour former cette structure de saut : l'affectation d'un identificateur à une étiquette et l'instruction d'utilisation du saut.

L'instruction **assign** relie une variable de type **integer*4** avec une étiquette de la même unité de programme, permettant d'introduire un identificateur alphanumérique dans l'instruction **go to** plutôt qu'une étiquette numérique. La syntaxe est :

assign *etiq* to *ivar*

avec :
etiq une constante entière non signée correspondant à une étiquette d'une instruction exécutable dans la même unité de programme,
ivar l'identificateur d'une variable de type **integer*4** . La variable ***ivar*** ne contient pas nécessairement la valeur numérique de l'étiquette, mais l'adresse interne correspondante.

La variable ***ivar*** ne peut pas s'introduire dans une expression arithmétique, car elle est indéfinie. Pour donner une valeur à cette variable, il faut utiliser une instruction d'affectation, l'adresse de l'étiquette étant alors perdue.

L'instruction **go to** imposé transfère le déroulement du programme vers l'instruction portant l'étiquette assignée à la variable entière de l'instruction **assign** . La syntaxe est :

go to *ivar* [[,] (*eliste*)]

avec :
ivar une variable de type **integer*4**,
eliste une liste d'une ou de plusieurs étiquettes d'instructions exécutables, séparées par la virgule, placées entre parenthèses, dans la même unité de programme. Cette liste décorative n'a pas d'influence sur le déroulement du programme.

```
*
*   Exemple
*   Utilisation de trois instructions d'assignement d'identificateurs
*   d'étiquette et sauts dans le programme
*
          program exem06
          integer*4 erreur,fini,itexte,nombre
          assign 20 to itexte assign 30
          to erreur assign 9999 to fini
*
  10      print *
          print *,'Introduisez :'
          print *,'- un nombre entier compris entre 1 et 100' read *,nombre
*
          if ((nombre .lt. 1) .or. (nombre .gt. 100)) go to erreur
*
          print *,'Merci pour votre participation !' go to fini
*
  20      format (i5,'            n''est pas compris entre 1 et 100')
  30      print itexte,nombre go to
          10
*
  9999 end
```

Résultat de l'exécution du programme :

```
Introduisez :
- un nombre entier compris entre 1 et 100
123
   123      n'est pas compris entre 1 et 100

Introduisez :
- un nombre entier compris entre 1 et 100
58
Merci pour votre participation !
```

• GO TO CALCULÉ

L'instruction **go to** calculé transfère le déroulement du programme vers l'une des instructions à étiquette selon la valeur prise par une expression arithmétique de type **integer**. La syntaxe est :

go to (*etiq1* , *etiq2* , . . .) [,] *iexpr*

avec :
etiqi une étiquette d'une instruction exécutable dans la même unité de programme, la même étiquette pouvant apparaître plusieurs fois dans liste.
iexpr une expression arithmétique entière, donnant un résultant dans les limites 1 à *n*, si *n* est le nombre d'étiquettes placées entre parenthèses.:

```
*
*   Exemple
*   Introduction d'un nombre entier
*   Affichage si ce nombre est pair ou impair
*
```

```
      program exem07

      intrinsic mod
      integer*4         nombre,npair
      character*1 rep
*
 10   print *
      print *,'Entrez un nombre entier :' read *,nombre
      npair = mod(nombre,2)
*
      go to (20,30) npair+1
*
 20   print *,nombre,' est un nombre pair.' go to 40
 30   print *,nombre,' est un nombre impair.'

*
 40   print *
      print *,'Encore un nombre (o/n) ?' read (*,'(a1)')
      rep
      if ((rep .eq. 'o') .or. (rep .eq. 'O')) go to 10
*
      end
```

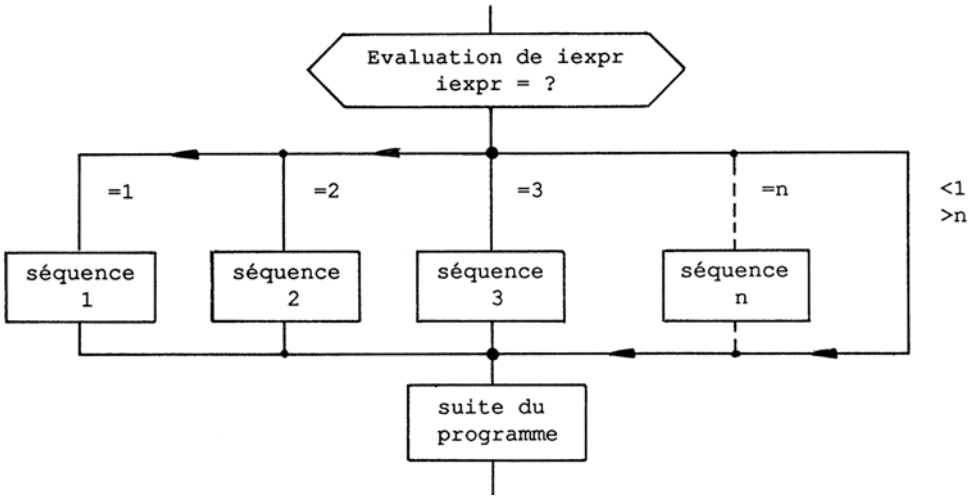
Résultat de l'exécution du programme :

Entrez un nombre entier :
1234
1234 est un nombre pair.

Encore un nombre (o/n) ?
o

Entrez un nombre entier :
-123
-123 est un nombre impair.

Encore un nombre (o/n) ?
n



La fin d'une séquence d'instructions impose un saut dans le programme, action réalisée généralement par une instruction **go to** inconditionnel.

5.3 LES INSTRUCTIONS CONDITIONNELLES IF

5.3.1 INSTRUCTION : IF . . . ARITHMÉTIQUE

Cette instruction appartient aux instructions sélectives de transfert conditionnel du déroulement du programme. L'instruction **if** arithmétique transfère le déroulement du programme vers l'une des trois instructions à étiquette citées, suivant la valeur prise par l'expression arithmétique. La syntaxe de cette instruction est :

if (*expr*) *etiq1* , *etiq2* , *etiq3*

avec :

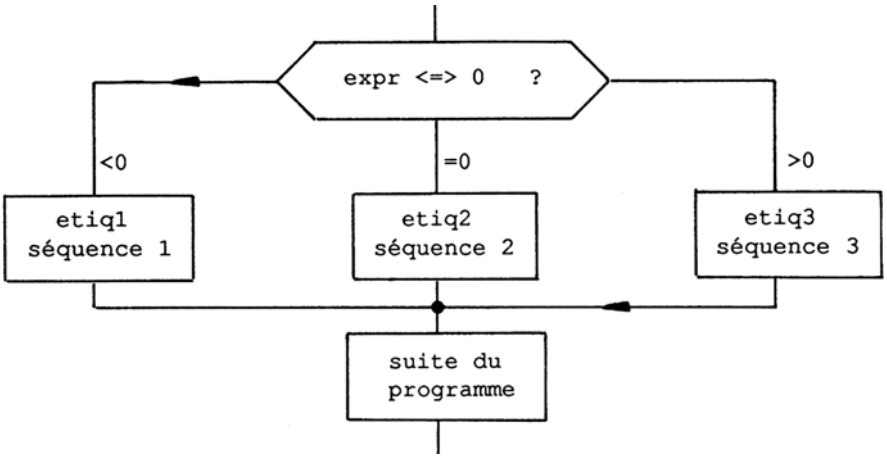
expr une expression arithmétique de type **integer**, **real** ou **double precision**,
etiqi une étiquette d'une instruction exécutable dans la même unité de programme.

Les trois étiquettes ***etiq1*** , ***etiq2*** , ***etiq3*** doivent figurer dans cette instruction. Elles ne doivent pas nécessairement se rapporter à trois instructions différentes. L'expression entre parenthèses ***expr*** est tout d'abord évaluée et le déroulement du programme se poursuit en fonction de la valeur de l'expression arithmétique :

1. si ***expr*** est négative, l'exécution du programme se poursuit à l'instruction portant la première étiquette ***etiq1*** ;
2. si ***expr*** est nulle, l'exécution du programme se poursuit à l'instruction portant la seconde étiquette ***etiq2*** ;
3. si ***expr*** est positive, l'exécution du programme se poursuit à l'instruction portant la troisième étiquette ***etiq3*** .

Cette possibilité de diverger vers trois parties de programme différentes peut aussi s'effectuer au moyen du bloc **if** . . . sans faire intervenir des étiquettes, voir sous 3.2.4.

La fonction de cette instruction peut se représenter sur l'organigramme partiel ci-après. Elle introduit toujours trois étiquettes et des sauts dans le programme.



Cette instruction se retrouvait fréquemment dans les sous-programmes mathématiques et ne devrait plus s'utiliser en programmation structurée. La configuration d'un programme partiel serait :

```
      if (iexpr) 10,20,30
10    { instructions de la séquence 1 si iexpr < 0 } go to 40
20    { instructions de la séquence 2 si iexpr = 0 } go to 40
30    { instructions de la séquence 3 si iexpr > 0 }
40    { suite des instructions du programme ... }
```

5.3.2 INSTRUCTION IF . . .LOGIQUE

L'instruction **if** . . . logique provoque l'exécution conditionnelle d'une seule instruction FORTRAN. La syntaxe est :

IF (*lexpr*) *instr*

avec :

lexpr une expression de type **logical** ,
instr une instruction exécutable FORTRAN , sauf l'une des instructions suivantes :
- boucle **do** ,
- **end** ou **end do**,
- bloc **if** . . . , **else if** , **else** , **end if** ,
- une autre instruction **if** . . . **logique**.

L'expression logique, placée entre parenthèses, est tout d'abord évaluée. Si la valeur de l'expression logique est **.true.** , l'instruction ***instr*** est exécutée. Par contre, si l'expression logique est **.false.** , le programme continue à la ligne suivante sans exécution de l'instruction proposée.

Cette instruction est à utiliser chaque fois que la structure sélective offre un seul choix possible. L'instruction ne comprend jamais le mot **then** .

Exemples :

```
if (i .gt. 3) j = j+1
if (j .lt. 100) print *,'La valeur de j est inférieure à 100'
```

5.3.3 INSTRUCTIONS AVEC BLOC IF

Les instructions se composant de un ou de plusieurs blocs **if** permettent de créer une ou plusieurs structures sélectives sans introduire des sauts et/ou des étiquettes. Les instructions composantes sont :

- if . . . then
- else if . . . then
- else
- end if

5.3.3.1 FORME GENERALE

La combinaison des diverses composantes permet d'obtenir facilement le choix sélectif d'une séquence d'instructions. La forme générale de la structure est la suivante :

```
if ( expr1 ) then
      bloc1
else if ( expr2 ) then
      bloc2
.....
else
      blocn
end if
```

avec :

expri une expression de type **logical** servant de sélecteur,
bloci une suite de zéro à plusieurs instructions FORTRAN , cette séquence étant appelée bloc d'instructions.

La forme générale peut se décrire par la configuration suivante :

```
if (a .gt. b) then
      séquence 1
else if (a .gt. c) then
      séquence 2
else if (a .gt. d) then
      séquence 3
else
      séquence 4
end if
```

5.3.3.2 BLOC IF . . . SIMPLE

Cette structure est une extension des possibilités de l'instruction if ... logique. Elle permet de traiter une séquence d'instructions si l'expression parenthésée prend la valeur logique **.true.** .

```
if ( expr ) then
      séquence d'instructions
end if
```

5.3.3.3 BLOCS IF . . . ET ELSE

Cette syntaxe résout la structure sélective suivante :

si expression logique est vraie **alors**
exécute la séquence 1 d'instructions
sinon
exécute la séquence 2 d'instructions

La séquence 1 d'instructions comprend toutes les instructions comprises entre la ligne **if** (***expr***) **then** et la ligne contenant **else** . La séquence 2 d'instructions part après la ligne **else** et se termine avant la ligne de fin de bloc avec **end if** . Si l'expression ***expr*** prend la valeur logique **.true.** , la séquence 1 est exécutée, sinon c'est la séquence 2 qui est prise en compte. La configuration générale est :

```
      if ( expr ) then
        séquence 1
 else
        séquence 2
  end if
```

Cette structure permet d'effectuer un choix en fonction de la valeur prise par l'expression logique.

5.3.3.4 BLOCS IF . . . ET ELSE IF . . .

Cette structure sélective représente un cas particulier du cas général et peut s'écrire :

si expression 1 est vraie **alors**
exécute la séquence 1 d'instructions
sinon si expression 2 est vraie alors exécute la
séquence 2 d'instructions
sinon
continue à l'instruction exécutable suivant end if

La configuration générale de cette structure se programme par :

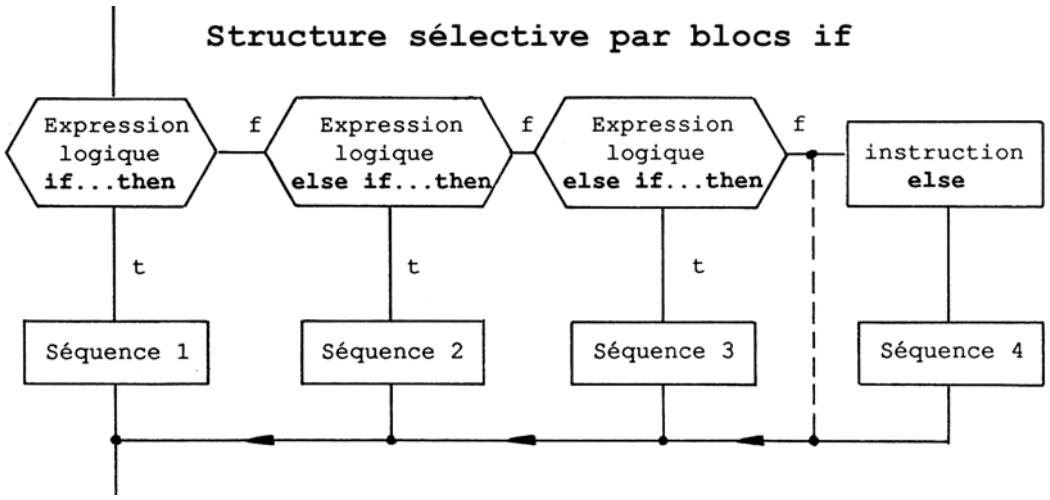
```
if (expr1) then
      séquence 1
else if (expr2) then
      séquence 2
end if
```

5.3.3.5 REMPLACEMENT DE L'INSTRUCTION : IF ARITHMÉTIQUE

L'instruction **if . . .** arithmétique permet de bifurquer dans l'exécution du programme en fonction de la valeur arithmétique d'une expression. Nous pouvons remplacer cette structure à sauts et étiquettes par un ensemble de blocs **if** .

```
if (iexpr .lt. 0) then
      séquence 1
else if (iexpr .eq. 0) then
      séquence 2
else
      séquence 3
end if
```

if ... then , la séquence 3 entre le second **else if ... then** et else , la séquence 4 entre **else** et **end if** .



Remarque importante :

Un seul bloc est évalué et comme l'instruction **else** est présente, le choix porte sur les tests successifs des expressions, sinon le programme se déroule dans la séquence 4 .

5.3.3.6 BLOCS IF . . . IMBRIQUÉS

. Le principe de cette structure imbriquée est donnée dans la configuration suivante :

```
if (expr1) then
 séquence 1.1
        if (expr12) then
      séquence 2.1
        else
      séquence 2.2
        end if
 else
  séquence 1.2
  end iF
```

EXEMPLE DE PROGRAMME TP

Soit à résoudre l'équation algébrique du second degré au moyen de structures sélectives sans introduire une seule étiquette :

$$a\,x^2 + b\,x + c = 0\ .$$

La solution doit tenir compte de tous les cas possibles de solution :

1. équation du second degré si $a \neq 0$, racines réelles et complexes,
2. équation du premier degré si $a = 0$ et $b \neq 0$,
3. équation indéterminée si $a = 0$, $b = 0$ et $c \neq 0$.

La solution de l'équation du second degré est obtenue en normant l'équation primitive par division par le coefficient a et en écrivant :

$$X^2 + 2\,B\,X + C = 0\ .$$

Ce programme introduit d'une façon systématique les blocs **if ...** et déclare toutes les variables utilisées.

```
*
*   Exemple
*   Equation du second degré : a x**2 + b x + c = 0
*   Variables utilisées :
*   a,b,c           coefficients et constante de l'équation primitive
*   bsur2a          b divisé par 2 a
*   csura           c divisé par a
*   discr           discriminant : b**2-c
*   racine          racine carrée du discriminant
*
*   Fonctions arithmétiques :
*                   sqrt        racine carrée
*                   abs         valeur absolue
*
        program exem08
        intrinsic abs,sqrt real*4
        a,b,c
        real*4 bsur2a,csura,discr,racine
*
        print *
        print *,'               EQUATION DU SECOND DEGRE'
        print *;'                 a x**2 + b x + c = 0 .' print *
        print *,'Donnez les coefficients et constante a,b,c' read *,a,b,c
*
*   Bloc extérieur avec test de la valeur du coefficient a
*
        if (a .ne. 0.0) then bsur2a=0.5*b/a
            csura=c/a discr=bsur2a**2-
            csura racine=sqrt(abs(discr))
*
*   Solution de l'équation du second degré
*
            if (discr .gt. 0.0) then
                print *,'Deux racines réelles distinctes' print *,'x1 =',-
                bsur2a+racine
                print *,'x2 =',-bsur2a-racine
            else if (discr .eq. 0.0) then
                print *,'Deux racines réelles égales' print *,'x1 = x2
                =',-bsur2a
            else
                print *,'Deux racines complexes conjuguées' print *,'x1 =',-
                bsur2a,' + j *',racine print *,'x2 =',-bsur2a,' - j *',racine
            end if
*
*   Fin du premier ensemble des blocs intérieurs
*   Calcul de l'équation du premier degré
*   Nouvel ensemble de blocs intérieurs
*
```

```
            else
                if (b .ne. 0.0) then
                    print *,'L''équation est du premier degre' print *,'x1 =',-c/b
                else
                    print *,'L''équation est indeterminée'
                end if
*
*   Fin du second ensemble intérieur
*
            end if
*
*   Fin de l'ensemble extérieur
*   Fin du programme
*
            end
```

Résultats de l'exécution du programme Exemple 1 :

```
          EQUATION DU SECOND DEGRE
           a x**2 + b x + c = 0 .

 Donnez les coefficients et constante a,b,c 1 2 -3

 Deux racines réelles distinctes x1 =    1.0000000
 x2 =                -3.0000000
```

Exemple 2 :

```
          EQUATION DU SECOND DEGRE
           a x**2 + b x + c = 0 .

 Donnez les coefficients et constante a,b,c 1 2 3

 Deux racines complexes conjuguées
 x1 =                -1.0000000 + j *                1.4142140
 x2 =                -1.0000000 - j *                1.4142140
```

5.4 STRUCTURES ITÉRATIVES

Le langage FORTRAN reconnaît deux structures itératives, la première avec un compteur d'ité- rations, la seconde, hors norme ANSI, mais très utile, de genre tant qu'une condition est remplie, exécute une séquence d'instructions.

5.4.1 STRUCTURE A COMPTEUR

L'instruction **do ...** permet l'exécution d'une procédure répétitive d'une séquence d'instructions. Elle provoque le déroulement d'un ensemble d'actions programmées entre la définition du compteur et la fin de la boucle.

La syntaxe de création du compteur est :

do [*etiq* [,]] *index* = *init* , *fini* [, *pas*]

avec :

etiq une constante entière correspondant à l'étiquette d'une instruction exécutable, placée à la suite de l'instruction **do .....** , dans la même unité de programme;

index une variable servant de contrôle de la boucle, de type **integer** , **real** ou **double precision**;

init une expression donnant la valeur initiale à la variable de contrôle ***index*** ; elle peut être une constante ou une expression de type **integer** , **real** ou **double precision** ;

fini une expression fixant la valeur finale pour la variable de contrôle ***index*** ; elle peut être une constante ou une expression de type **integer** , **real** ou **double precision** ;

pas une expression représentant l'incrément qui s'additionne après chaque parcours de la boucle à la variable de contrôle ***index*** . Elle peut être une constante ou une expression de type **integer** , **real** ou **double precision** . L'incrément peut être positif, négatif mais pas nul. Si ce paramètre est absent, l'incrément vaut 1.

L'étiquette peut être omise si l'on utilise **end do** en fin de boucle. Dans les anciennes versions FORTRAN , les expressions de début, de fin et d'incrément ne pouvaient être que de type **integer**. La virgule après l'étiquette est optionnelle.

La variable ***index*** est appelée variable de contrôle. Avant le déroulement de la séquence d'ins- tructions dans la boucle, les expressions sont évaluées et si nécessaire converties dans le type de la variable de contrôle. La fin de la structure est identifiée par une instruction à étiquette, l'instruction **continue** étant habituellement introduite dans ce but. L'instruction **end do** remplace avantageu- sement l'instruction étiquetée, si cette possibilité est comprise par le compilateur. L'instruction étiquetée de fin de boucle ne peut pas contenir les instructions :

- **go to** (inconditionnel ou assigné),
- **if** (arithmétique ou bloc),
- des instruction d'un bloc **if** : **else if** , **else** , **end if** ,
- **end** ,
- **return** ,
- une autre instruction **do** .

Si le compilateur admet la version étendue, l'instruction terminale devient simplement :

end do

L'ensemble des instructions, débutant par l'instruction **do ...** et se terminant par l'instruction à étiquette ou **end do** , fait partie de la structure de la boucle **do** .

5.4.2 CONTROLE DU NOMBRE D'ITÉRATIONS

L'instruction **do ...** évalue tout d'abord les expressions ***init*** , ***fini*** et ***pas*** si c'est nécessaire. La valeur de l'expression ***init*** est alors affectée à la variable de contrôle ***index*** . Le nombre de fois que la boucle **do** est parcourue est calculable par :

[(***fini*** **-** ***init*** **+** ***pas*** **) /** ***pas*** **]**

cette expression étant réduite en entier et le résultat portant le nom de compteur d'itérations. Si le compteur d'itérations vaut zéro ou est négatif, la boucle n'est pas parcourue, contrairement aux ver- sions précédentes FORTRAN où elle était effectuée toujours au moins une fois.

Après chaque parcours de la boucle **do ...** , les opérations suivantes sont effectuées :

1. la valeur de l'expression d'incrémentation est additionnée algébriquement à la variable de con- trôle,
2. le compteur d'incrémentation est décrémenté de 1,
3. si la valeur du compteur d'itérations est supérieure à zéro, le programme continue à la première instruction exécutable après l'instruction **do** ,
4. si la valeur du compteur d'itérations est égale à zéro, l'exécution de la boucle **do** est terminée. Dans ce dernier cas, pour autant qu'il n'y ait pas de boucles imbriquées, le contrôle du déroule-

ment du programme est transféré à la première instruction exécutable suivant directement l'instruc- tion terminale de la boucle **do** .

Exemples :

avec étiquette	**avec end do**
do 10 nombre = 5,50,5	**do** nombre = 5,50,5 a(nombre)=a(nombre-1) a(nombre) = a(nombre-1) b(nombre-2)=b(nombre+2) b(nombre-2) = b(nombre+2)
10 **continue**	**end do**

La boucle proposée ici est parcourue : [(50-5+5)/5] = 10 fois.

5.4.3 BOUCLES IMBRIQUÉES

Une structure itérative do peut contenir une ou plusieurs autres boucles **do** . Les boucles im- briquées peuvent avoir la même étiquette terminale avec l'instruction correspondante. Par contre, l'introduction de l'instruction **end do** nécessite une instruction terminale pour chaque boucle. Ce mode de programmation peut également s'introduire avec les instructions à étiquettes.

Exemples de boucles imbriquées :

avec 3 étiquettes

```
      do 20 j = 1,5
           do 10 k=1,5
                v(i,j)=v(i,j)+x(k,j)
10          continue
20        continue
30    continue
```

avec end do

```
do 30 I = 1,5do i = 1,5
    do j = 1,5 v(i,j)=0.0          v(i,j)=0.0
         do k=1,5)
              v(i,j)=v(i,j)+x(k,j)
          end do
       end do
  end do
```

- **avec une seule étiquette**

```
      do 10 i=1,5
           do 10 j=1,5 v(i,j)=0.0 do 10 k=1,5
                    v(i,j)=v(i,j)+x(k,j)
10    continue
.
```

EXEMPLE DE PROGRAMME TP BOUCLES DO

Cet exemple permet d'introduire une certaine quantité de nombres réels, jusqu'à 100 nombres, de les classer en ordre croissant et de calculer la moyenne arithmétique de ces nombres.

```
* Exemple
* Introduction de nombres réels, au maximum 100 nombres
* Classement en ordre croissant et calcul de la moyenne
*
      program exem09 integer*2 i,j,nombre
      real*4          prov,somme,valeur(100)
           print *
      print *,'DETERMINATION D''UNE MOYENNE ARITHMETIQUE'
           print *
      print *,'Donnez la quantité de nombre à traiter (<=100)' read *,nombre
           print *
      print *,'Introduisez des nombres réels quelconques' print *
      do i=1,nombre
           write (*,'(5x,a,i4,a,$)') 'Nombre numéro :',i,' = ' read *,valeur(i)
      end do
* Rangement en ordre croissant
      do i=1,nombre-1
           do j=i+1,nombre
                if (valeur(i) .gt. valeur(j)) then prov        = valeur(i)
                     valeur(i) = valeur(j) valeur(j) = prov
                end if
           end do

      end do
*
* Calcul de la somme et de la moyenne, affichage des résultats
*
           print *
      print *,'Valeurs rangées en ordre croissant' print *
      somme = 0.0
```

```
      do i=1,nombre
            somme = somme + valeur(i) write (*,'(5x,a,i4,a,f12.4)')
     +      'Valeur classée numéro :',i,' = ',valeur(i)
      end do
            print *
      write (*,'(5x,a,f12.4)') 'Moyenne arithmétique :',somme/nombre
      end
```

Résultat de l'exécution du programme

```
DETERMINATION D'UNE MOYENNE ARITHMETIQUE

Donnez la quantité de nombre à traiter (<=100) 16

Introduisez des nombres réels quelconques Nombre numéro : 1 = 21.2345

etc.Ecrivez ce programme et testez sa fonctionnalité.
```

5.4.4 BOUCLE DO IMPLICITE

Cette structure, utilisée le plus souvent dans les instructions de lecture et d'écriture, est une simplification de la syntaxe fondamentale de la structure itérative, le mot-clé **do** étant omis.

* **Exemple**

```
* Affectation d'un tableau par data
* Utilisation de la boucle do implicite
*
      program exem10 integer*2 i,j,mat(5,5)
      data ((mat(i,j), j=1,1), i=1,5) /5*11/
      data ((mat(i,j), j=2,2), i=1,5) /5*22/
      data ((mat(i,j), j=3,3), i=1,5) /5*33/
      data ((mat(i,j), j=4,4), i=1,5) /5*44/
      data ((mat(i,j), j=5,5), i=1,5) /5*55/
      do i=1,5
            print *,(mat(i,j), j=1,5)
      end do
      end
```

Résultat de l'exécution du programme

```
 11          22          33          44          55
 11          22          33          44          55
 11          22          33          44          55
 11          22          33          44          55
 11          22          33          44          55
```

Remarque : Dans le cas particulier, l'affectation par **data** s'effectue colonne après colonne. Il est aussi possible de simplifier l'affectation en programmant l'instruction suivante :

```
      data mat /5*11,5*22,5*33,5*44,5*55/
```

le résultat étant identique.

5.4.5 STRUCTURE DO WHILE (TANT QUE)

La syntaxe de cette instruction est :

do [*etiq* [,]] while (*exprl*)

avec :

etiq une étiquette optionnelle d'une instruction exécutable dans la même unité de programme, située après l'instruction de définition de la boucle. La virgule est optionnelle. Nous n'utili- serons pas cette possibilité de saut.

exprl une expression de type **logical**, placée entre parenthèses.

Chaque boucle **do while** doit se terminer par sa propre instruction **end do** qui ne nécessite pas d'étiquette. Si **end do** est affectée d'une étiquette, celle-ci doit correspondre à celle citée dans **do while** .

EXEMPLE DE STRUCTURE DO WHILE

Le programme proposé permet à l'utilisateur d'introduire un nombre entier positif et de calculer la somme de ces entiers et de leurs carrés.

```
** Exemple
* Introduction d'un nombre entier positif
* Calcul de la somme des nombres et des carrés
*
      program exem11
      integer*4 carre,nombre,somme character*1 rep
*
      data rep /'o'/
*
      do while ((rep .eq. 'o') .or. (rep .eq. 'O')) print *
        print *,'Donnez un nombre entier positif (<=500) :' read *,nombre
          somme = 0
          carre = 0 do i=1,nombre
          somme = somme + i carre = carre + i*i
        end do
          print *
        print *,'La somme des entiers vaut :',somme print *,'La somme des carrés vaut :
        ',carre
          print *
        print *,'Encore une valeur (o/n) ?' read (*,'(a1)') rep
      end do
*
      end
```

Résultat de l'exécution du programme

```
Donnez un nombre entier positif (<=500) :
100

La somme des entiers vaut :                5050
La somme des carrés vaut :                338350

Encore une valeur (o/n) ? n
```

Si le compilateur n'admet pas cette structure itérative, il est possible d'obtenir le même résultat au moyen d'une structure sélective avec étiquette. Le même programme aurait alors les instructions suivantes:

5.5 L'INSTRUCTION DE CHOIX (SELECT CASE)

A la forme la synthaxe suivante

```
[ref:] SELECT CASE (expression)
       CASE cas_1
            bloc_1
            ...
       CASE cas_n
            bloc_n
       CASE DEFAULT
            bloc_defaut
       END SELECT [ref]
```

Exemple:

```
integer :: flag, res
    ...
 SELECT CASE(flag)
  CASE (1, 2, 3)
    res=1
  CASE (4:6)
    res=2
 DEFAULT
    res=0
 END SELECT
```

EXEMPLE DE PROGRAMME TP DE CASE

Q1 : corrigé les erreurs du programme

Q2 : que donne comme résultat

```
PROGRAM EXEMPLE_case

  INTEGER :: I
  INTEGER, PARAMETER :: N=17 PRINT*, ' ENTRER I'
  READ*, I
  SELECT CASE (I) CASE (2,3,5,7)
     PRINT*, '  I EST PREMIER'
     PRINT*, '  ET PLUS PETIT STRICTEMENT QUE 11'  CASE (N)
     PRINT*, ' I=N'  CASE (8:10)
     PRINT*, ' I EST COMPRIS AU SENS LARGE ENTRE 8 ET 10'  CASE  (26:)
     PRINT*, ' I EST PLUS GRAND OU EGAL A 26'  CASE (:-3)
     PRINT*, ' I EST PLUS PETIT QUE -3'  CASE DEFAULT
     PRINT*, ' COMPRIS ENTRE -2 ET25, MAIS&
         & DIFFERENT DE 2,3,5,7,8,9,10 ...'  END SELECT
END PROGRAM EXEMPLE_case
```