UNIVERSITE BATNA2
FACULTE DE TECHNOLOGIE
DEPARTEMENT DE MECANIQUE

INITIATION AU FORTRAN

Première Partie : COURS

S. DERRADJI
M.K. BELBAHRI

Chapitre 0

Algorithmes, Organigrammes et programmes

0.1 Algorithme

Définition 0.1.1 Un algorithme est un ensemble fini d'instuctions élémentaires non ambiguës amenant (conduisant) à la résolution d'un problème donné.

Ces algorithmes sont écrits en langage humain. Cependant, nous recommandons l'utilisation des mots français suivants :

1) **Affecter**. Le symbole ← est utilisé pour représenter cette instruction. Elle attribue une valeur à une variable.

Exemple

Affecter le contenue de la variable X à la variable Y est symbolisée par Y← X. On dit aussi que Y reçoit la valeur de X.

2) **Ajouter**, **soustraire**, **multiplier**, **diviser** sont représentées respectivement par les symboles : +, -, *, /.

Exemple

Les instructions : ajouter B à A, soustraire B de A, Multiplier B par A et diviser A par B s'écrivent respectivement : A+B, A-B, A*B, A/B.

3) **Inférieur à**, **inférieur ou égal à**, **supérieur à**, **supérieur ou égal à**, **égal à**, **différent de** sont représentés respectivement par les symboles : <, <=, >, >=, =, <>

4) **Lire** . Elle transfert les données de l'exterieur (clavier) vers la mémoire centrale.

Exemple

Lire A. Cette instruction demande la valeur de la variable A.

5) **Ecrire** . Elle transfert les résultats de la mémoire centrale vers l'extérieur (écran).

Exemple

Ecrire A. Cette instruction affiche (imprime) la valeur de la variable A.

6) **Si logique**

Si (condition) **alors**

instruction1

instruction2

…….. .

Instruction m

Sinon

instruction1

instruction2

…….. .

Instruction n

Fsi

où

Fsi : Fin de de Si

Les instruction1,…, instruction m sont exécutées si condition est vraie. Dans le cas contraire, Les instruction1,…, instruction n sont exécutées.

Exemple

Si (d >= 0)alors

Ecrire ‘ d est positif ’

Sinon

Ecrire ‘ d est négatif ’

Fsi

Dans cet exemple, si la condition d >= 0 est vrai alors **Ecrire** ‘ d est positif ’

est exécutée dans le cas contraire l’instruction **Ecrire** ‘ d est négatif ’ est exécutée.

7) Boucle Pour

P**our** i **variant** de n1 à n2 avec un pas de n3 **faire**

. instruction1

instruction2

…….. .

Instruction m

Fpr

où

Fpr : Fin de Pour

Où n1, n2, n3 dont des valeurs réelles désignant respectivement la valeur initiale de l'indice i, sa valeur finale et le pas (incrément). Elle veut dire, tant que i ne dépasse pas la valeur n2 répéter l'exécution des instruction 1,…, instruction m

Exemple

Pour i variant de 1 à 10 avec un pas de 1 **faire**

s ← s+i

Fpr

8) Boucle Tant que

Tant que (condition) **faire**

instruction1

instruction2

…….. .

Instruction m

Ftq

où

Ftq : Fin de tant que

Elle veut dire,faire l'exécution des instruction1,…, instruction m jusqu'à ce condition soit fausse.

Exemple

i ← 1

Tant que (i<=10) **faire**

s ← s+i

i ← i+1

Ftq

Dans cet exemple, ces deux instructions sont exécutées jusqu'à ce que la condition i<=10 soit fausse.

9) **Aller à** n

Où n est une valeur entière désignant le numéro de l'instruction à laquelle on doit se brancher. Elle veut dire, continuer l'exécution à partir de l'instruction dont le numéro est désigné par n

Exemple

s ← 0i

1 s ← s+i

i ← i+1

Si (i <= 10 **) aller à** 1

Fsi

10) Un algorithme commence par le mot **Débu**t et se termine par le mot **Fin**.

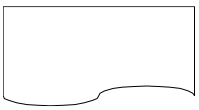
0.2 Organigramme

Définition 0.1.1 Un organigramme est une représentation graphique d'un algorithme d'un problème donné.

Les symboles (formes) constitutifs d'un organigramme sont :

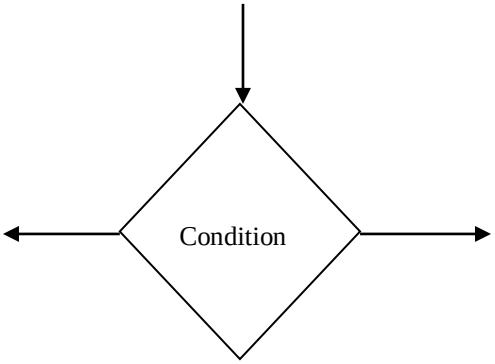
représente le début ou la fin d'un organigramme

représente la lecture des données



représente l'impression (affichage) des résultats

représente le traitement



représente un test logique

représente un renvoi

représente le sens du traitement

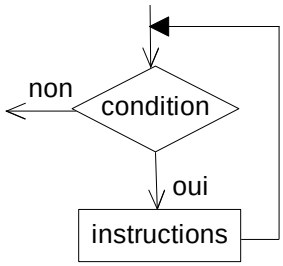
La boucle Tant que :

Tant que (condition) **alors** répéter

Instructions

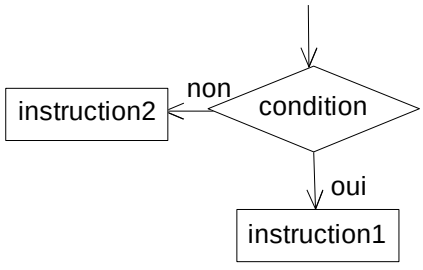
Ftq

peut être représentée schématiquement par :



L'instruction **Si** condition **alors** instruction1

peut être représentée schématiquement par :



0.3 Exemples

Exemple1

a) Ecrire un algorithme qui calcule le maximum de deux nombres A et B

Début

Lire A,B

Si (A<=B) **alors**

Ecrire ‘B est le maximum’

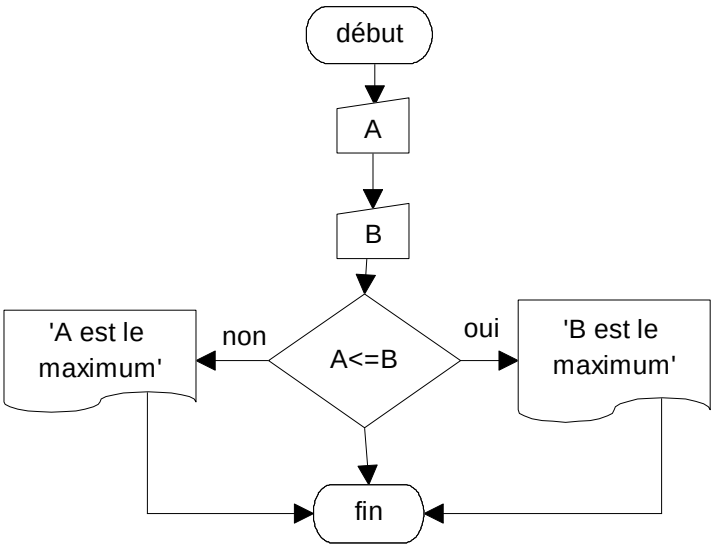
Sinon

Ecrire ’A est le maximum

Fsi

Fin

b) Ecrire un organigramme qui calcule le maximum de deux nombres A et B



Exemple 2

a) Ecrire un algoritme qui détermine la valeur absolue d'un nombre A

Début

Lire A

Si (A<0) **alors**

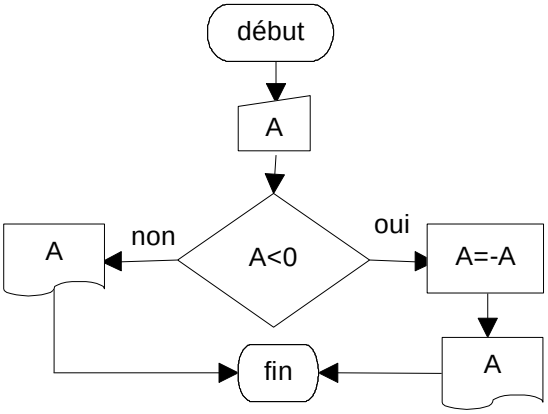
A←-A

Fsi

Ecrire A

Fin

b) Ecrire un organigramme qui détermine la valeur absolue d'un nombre A



Exemple 3

a) Ecrire un algorithme qui détermine la parité d'un entier

Début

Lire A

R← reste de la division par 2

Si (R=0) **alors**

Ecrire 'A est pair'

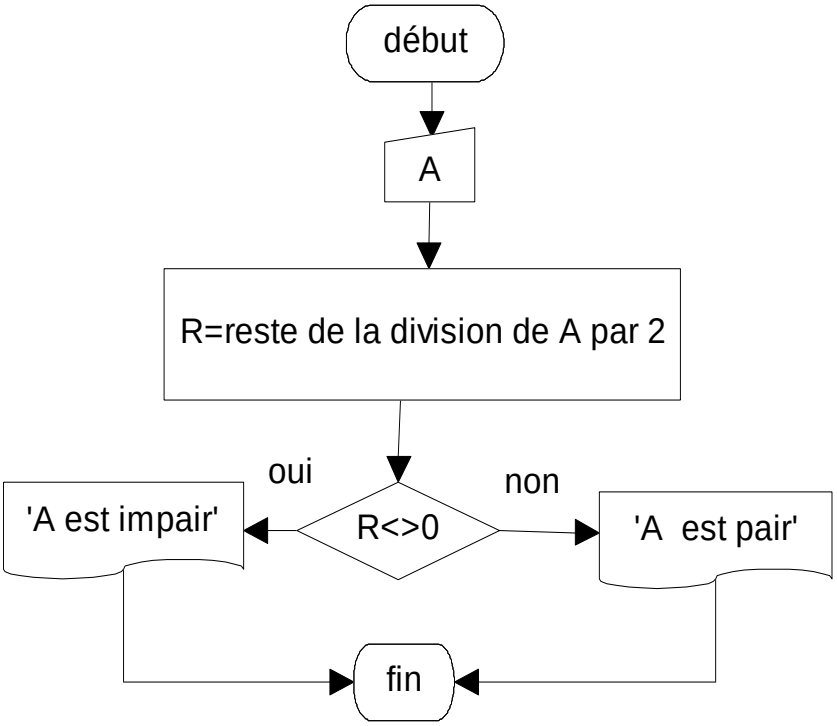
Sinon

Ecrire ‘A est impair’

Fsi

Fin

b) Ecrire un organigramme qui détermine la parité d’un entier



Exemple 4

a) Ecrire un algorithme qui calcule S=1+2+3+…10

Début

S←0

Pour I=1 à 10 avec un pas de 1 **faire**

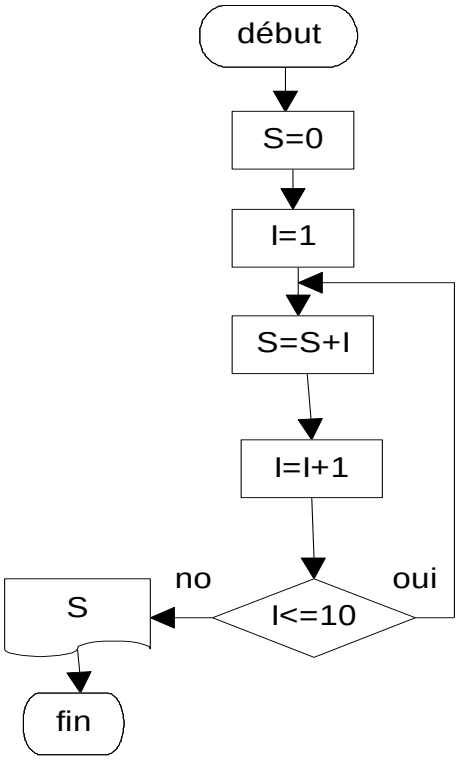
S←S+I

Fpr

Ecrire S

Fin

b) Ecrire un organigramme qui calcule S=1+2+3+…10



Exemple 5

a) Ecrire un algorithme qui calcule le produit p=1*2*3*…*10

Début

P← 1

Pour I=1 à 10 avec un pas de 1 **faire**

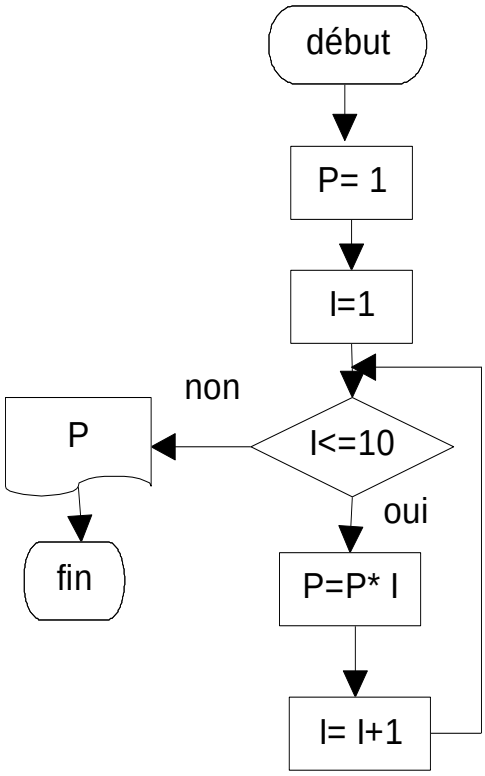
P←P*I

Fpr

Ecrire P

Fin

b) Ecrire un organigramme qui calcule le produit p=1*2*3*…*10



Exemple 6

a) Ecrire un algorithme qui permute deux nombres A et B

Début

Lire A,B

Ecrire A,B, 'AVANT PERMUTATION'

T← A

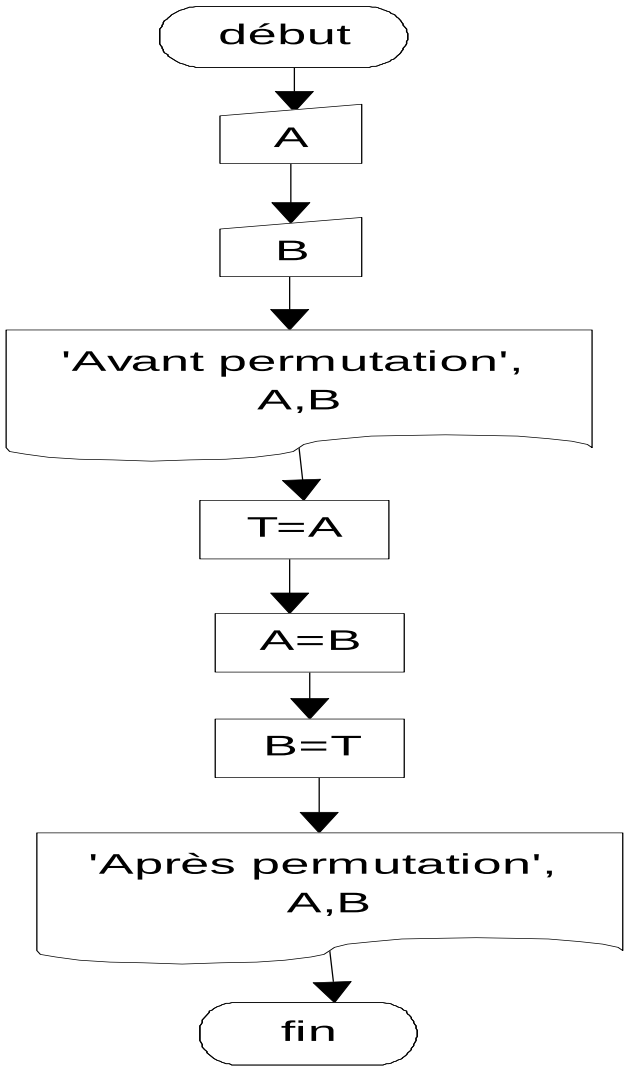
A← B

B← T

Ecrire A,B, 'APRES PERMUTATION'

Fin

b) Ecrire un organigramme qui permute deux nombres A et B



CHAPITRE I : LES ELEMENTS DE BASE DU LANGAGE FORTRAN

1.1. Les constantes

Une constante est une quantité fixée et invariable. Il existe trois types de constantes : numériques, logiques et alphanumériques.

Constantes numériques

Elles se subdivisent en deux catégories : les constantes entières et les constantes réelles.

Les constantes entières sont des nombres entiers dont la magnitude ne dépasse pas 2^n, où n est un entier positif dépendant de l'ordinateur utilisé. Elles ne contiennent pas de point décimal.

Les constantes réelles sont des nombres réels contenant un point décimal. Elles peuvent être écrites sous la forme usuelle avec un point décimal :

<suite de chiffres décimaux>.<suite de chiffres décimaux>

ou sous la forme " exposant " :

<constante numérique réelle>E<constante entière>.

Exemple 1.1.1

Constantes entières	Constantes réelles	
	usuelles	avec exposant
576	98.245	5.4E+2 (5.4×10^2)
2545	-2545.	-67E-3 (-67×10^{-3})
0	0.	12.34E2 (12.34×10^2)

Constantes logiques

Cette classe est constituée des deux valeurs de vérité : .TRUE. (vrai) et .FALSE. (faux). Les deux points précédant et suivant les constantes logiques sont obligatoires.

Constantes alphanumériques

Une constante alphanumérique est une suite de caractères alphabétiques et/ou numériques. Sa forme générale est nH******* où * représente un caractère et n est le nombre de caractères constituant la constante. On peut également écrire une constante sous la forme suivante : '**********'.

Exemple 1.1.2

7HFORTRAN ou 'FORTRAN'

1.2. Les variables

Une variable FORTRAN est une représentation symbolique d'une quantité pouvant prendre différentes valeurs. Un nom de variable est constitué de 1 à 6 caractères alphanumériques dont le premier est obligatoirement alphabétique. Les caractères spéciaux : +, -, *, ., /, $, =, (,), ', la virgule et le blanc ne peuvent pas figurer dans le nom d'une variable.

Exemple 1.2.1

Les noms de variables suivants sont valides :

A, ACI, SOMME

Par contre, les variables suivantes ne sont pas valides :

1A, A$CI, S OMME

Les spécifications de type

Il y a deux manières de spécifier le type d'une variable : explicite et implicite.

Spécification explicite

Si le nom de la variable est précédé de l'instruction de spécification de type : INTEGER, REAL ou LOGICAL, alors la variable est respectivement entière, réelle ou logique.

Spécification implicite

Si une variable n'est pas spécifiée explicitement et si sa première lettre est I,J,K,L,M ou N, alors la variable est entière. Par contre, si la première lettre est autre, la variable est réelle.

Exemple 1.2.2

```
INTEGER AB,AC,SOM

INTEGER X,Y
```

Ces instructions indiquent à l'ordinateur que les variables AB,AC,SOM,X et Y sont entières. Par contre, les instructions suivantes :

```
REAL I,NUM
```

indiquent à l'ordinateur que les variables I et NUM sont réelles. L'instruction

```
LOGICAL TEST
```

indique à l'ordinateur que la variable TEST est une variable logique.

Remarque

Il existe d'autres types de spécification. Par exemple :

DOUBLE PRECISION et COMPLEXE.

1.3. Opérateurs arithmétiques

Les seuls opérateurs permis dans le langage FORTRAN sont :

+	*addition*
-	*soustraction*
*	*multiplication*
/	*division*
**	*élévation à une puissance (exponentiation).*

Tous les autres opérateurs sont construits à l'aide des 5 opérateurs de base ci-dessus. Notons que deux opérateurs ne peuvent jamais se suivre dans une expression. Ainsi, ** représente un seul opérateur.

1.4. Expressions arithmétiques

Une expression arithmétique relie des variables, des constantes, des fonctions, etc. Elle peut être composée d'une constante unique ou d'une variable unique de type entier ou réel. Les expressions arithmétiques sont utilisées en langage FORTRAN pour spécifier le calcul d'une valeur numérique.

Exemple 1.4.1

Les expressions suivantes sont valides :

```
J
56.98
X*Y-A
S**2
```

Par contre, l'expression suivante n'est pas valide :

```
J*-K
```

Certains compilateurs, les moins récents, n'acceptent pas le mélange de type : constantes et variables entières avec constantes et variables réelles.

Exemple 1.4.2

Les expressions suivantes sont à éviter :

```
I*R
23.8-I
```

Elles sont appelées expressions mixtes. L'expression mixte

```
X**2
```

est acceptée. C'est la seule exception.

FONCTIONS

Dans un programme FORTRAN on fait appel à des fonctions élémentaires d'usage courant : La racine carrée, la valeur absolue, le logarithme, le sinus, le cosinus, etc. Le compilateur FORTRAN possède des "sous-programmes " qui évaluent ces fonctions. Bien que le programmeur puisse évaluer directement ces fonctions en utilisant les opérateurs de base +,-,*,/ et **, il est préférable de les utiliser directement. Nous donnons ci-dessous une liste de quelques fonctions élémentaires :

Fonction :	*Signification :*
SQRT(X)	racine carrée de X
ALOG(X)	logarithme népérien de X
ABS(X)	valeur absolue de X
ALOG10(X)	logarithme décimal de X

Remarque

Il est possible que l'argument soit une expression. Par exemple ABS (X-3.8)

1.5. Les opérateurs de relation

Ces opérateurs sont :

opérateur	*signification*
.GT.	supérieur à (>)

.GE.	supérieur ou égal à (≥)
.LT.	inférieur à (<)
.LT.	inférieur ou égal à (≤)
.EQ.	égal à (=)
.NE.	différent (≠)

Ces opérateurs permettent au programmeur de combiner des constantes, des variables, des fonctions ou des expressions arithmétiques pour former des " expressions logiques ".

Exemple 1.5.1

X.LT.1	*(X<1)*	*vrai si X<1 et faux si X≥1*
2*I.EQ.J	*(2I=J)*	*vrai si 2I=J et faux si 2I≠J*
X.NE.Y	*(X≠Y)*	*vrai si X≠Y et faux si X=Y.*

1.6. Les opérateurs logiques

Ces opérateurs sont :

Opérateur	*Signification*
.NOT.	négation
.AND.	intersection
.OR.	réunion

Ils permettent au programmeur de combiner des expressions logiques pour donner un résultat logique vrai (.TRUE.) ou faux (.FALSE.).

Remarque

Deux opérateurs logiques ne peuvent jamais se suivre, sauf si le second opérateur logique est .NOT.

Exemple 1.6.2

.NOT.Y	non Y	*Vrai si Y est faux, faux si Y est vrai*
X.AND.Y	X et Y	*vrai si X et Y sont vrais, faux si X est faux ou si Y est faux*
X.OR.Y	X ou Y	*vrai si X est vrai ou si Y est vrai ou les deux sont vrais, faux si les deux sont faux*

1.7. Hiérarchie des opérateurs arithmétiques

Parenthèses

Le rôle des parenthèses est d'indiquer l'ordre dans lequel les opérations arithmétiques doivent être effectuées. L'expression entre parenthèses est évaluée la première. L'utilisation de parenthèses

superflues n'est pas pénalisée. Il est par conséquent conseillé d'utiliser les parenthèses au maximum.

Exemple 1.7.1

```
(X+1.)/(Y+Z)
1./(X**2)
```

Quand la hiérarchie des opérateurs dans une expression arithmétique n'est pas contrôlée par des parenthèses, ou toute l'expression est dans une seule paire de parenthèses, l'ordinateur exécute ces opérations dans l'ordre suivant :

- 1. Evaluation des fonctions
- 2. Elévation à une puissance (**)
- 3. Multiplication et division (* et /)
- 4. Addition et soustraction (+ et -).

Remarque

S'il y a des opérateurs de hiérarchie égale (par exemple * et /), les opérations sont effectuées de la gauche vers la droite.

Exemple 1.7.2

A/B/C est équivalent à (A/B)/C, c'est à dire que le résultat de la division de A par B est divisé par C. De même, A/B*C est équivalent à (A/B)*C. En d'autres termes, le résultat de la division de A par B est multiplié par C.

Hiérarchie des opérateurs logiques

Si la hiérarchie des opérations dans une expression logique n'est pas contrôlée par des parenthèses, ou si toute l'expression logique est dans une seule paire de parenthèses, l'ordinateur exécute les opérateurs de relations dans l'ordre suivant :

.GT. *.GE.* *.LT.* *.LE.* *.EQ.* *.NE.*

Ensuite, les opérateurs logiques sont exécutés dans l'ordre suivant :

.NOT. *.AND.* *.OR.*

Exemple 1.7.3

```
A.GT.X.AND..NOT.L.OR.N
```

est évaluée da la façon suivante :

1) A.GT.X Soit R1 le résultat
2) .NOT.L Soit R2 le résultat
3) R1.AND.R2 Soit R3 le résultat
4) R3.OR.N

L'expression logique

```
A.GT.X.AND.(.NOT.L.OR.N)
```

est évaluée de la façon suivante :

1) .NOT.L Soit R1 le résultat

2) R1.OR.N Soit R2 le résultat

3) A.GT.X Soit R3 le résultat

4) R3.AND.R2

Remarque

Si dans le corps d'une expression logique figurent des expressions arithmétiques, celles-ci sont préalablement calculées en suivant leur règle de hiérarchie.

1.8. L'instruction d'affectation

Sa forme générale est :

nom de variable = expression arithmétique ou logique

Cette instruction est interprétée par le compilateur de la manière suivante :

Evaluer l'expression à droite du signe = et stocker sa valeur dans l'espace mémoire réservé à la variable gauche du signe =.

Remarques

1. Le signe = dans une instruction d'affectation n'a pas le sens habituel (comme en arithmétique). Ainsi, il est correct d'écrire l'instruction

N=N+1

qui signifie que la nouvelle valeur de N est égale à son ancienne valeur plus 1 (Par exemple si l'ancienne valeur de N est 3, sa nouvelle valeur est 4).

2. Les deux membres de l'égalité doivent être du même type. Cependant, si on écrit :

A=I

la valeur de A est celle de I convertie en réel, tandis que dans :

I=A

la valeur de I est la partie entière de A.

Exemple 1.8.1

```
C=2.0
D=3.0
A=2.
A=C+D
```

En exécutant les trois premières instructions, l'ordinateur affecte respectivement les valeurs 2.0, 3.0 et 2. à C, D et A. La 4eme instruction remplace l'ancienne valeur de A (2.) par la somme C+D, c'est à dire 2.0+3.0. Donc A=5.0.

Les instructions d'affectations suivantes sont valides :

```
ALPHA=-3/X**2+A/2.0*X
Y=EXP(X)
T=ALOG10(X)
```

Les instructions d'affectations suivantes sont non valides :

X=3.Y+2. *manque un opérateur entre 3. et Y*

Z=((X+Y)**2	*le nombre de parenthèses de gauche n'est pas égal au nombre de parenthèses de droite*
3.4=PI	*le nombre de gauche doit être une variable*
X=1./-2.*Y	*deux symboles ne peuvent pas se suivre*

1.9. L'instruction d'entrée (READ)

Cette instruction permet le transfert de l'information de l'unité d'entrée vers la mémoire centrale. Il existe deux méthodes pour introduire des données : Format libre et Format imposé.

a) Format libre

Sa forme générale est :

i) READ,liste de variables

Exemple 1.9.1

```
READ,A,LI,CD
```

ii) READ(*,*)liste de variables

Exemple 1.9.2

```
READ(*,*)A,LI,CD
```

Remarque

L'unité d'entrée est en général le clavier.

b) Format imposé

Sa forme générale est :

```
READ(a,b)liste de variables
```

où a est une constante entière positive désignant le numéro de l'unité d'entrée et b est le numéro de l'instruction FORMAT (voir paragraphe 1.11).

Exemple 1.9.3

```
      READ(105,3)X,TAXE
3     FORMAT(F3.1,F10.3)
```

Dans cet exemple, l'instruction READ indique à l'ordinateur que les données sont transférées de l'unité d'entrée numéro 105 sous le format numéro 3 vers la mémoire centrale.

1.10. instruction de sortie (WRITE)

Cette instruction permet le transfert de l'information de la mémoire centrale vers l'unité de sortie (par exemple l'imprimante). Il existe deux méthodes pour imprimer (ou afficher) des résultats : Format libre et Format imposé.

a) Format libre

Sa forme générale est :

i)

```
PRINT,liste de variables
```

Exemple 1.10.1

```
PRINT,AL,IJ,CD
```

ii)

```
WRITE(*,*)liste de variables
```

Exemple 1.10.2

```
WRITE(*,*)AL,IJ,CD
```

Remarque

Pour imprimer (ou afficher) des messages, on insère ces derniers entre deux apostrophes dans la liste de variables.

Exemple 1.10.3

```
PRINT,'LA VALEUR DE A EST',A
```

b) Format imposé

Sa forme générale est :

```
WRITE(a,b)liste de variables
```

où a est une constante entière positive désignant le numéro de l'unité de sortie (imprimante, écran) et b est le numéro du format associé à l'instruction WRITE.

Exemple

```
      WRITE(108,7)NHRS,DED
7     FORMAT(8X,I3,F10.5)
```

Cette instruction indique à l'ordinateur d'imprimer les valeurs des variables NHRS et DED en utilisant l'unité de sortie numéro 108 selon le format numéro 7.

1.11. L'instruction FORMAT

Cette instruction est utilisée parallèlement avec les instructions READ et WRITE pour indiquer la forme sous laquelle les données doivent être transmises. Cette instruction n'est pas exécutable et peut être placée n'importe où dans le programme. Sa forme générale est :

```
n     FORMAT(C1,C2,...,Cm)
```

où n est un entier positif et ci (i=1,...,m) un code définissant le type, la longueur et la position du point décimal (s'il y a lieu). La forme générale du code ci est

```
aSw.d
```

où a est une constante entière,, non signée et non obligatoire indiquant une répétition du code, S est le type de format, w est une constante entière non signée spécifiant la longueur totale du champ de l'information et d est une constante entière non signée spécifiant le nombre de décimales.

Types de format

Le type I

Il permet de lire ou d'imprimer des nombres entiers.

Exemple 1.11.1

```
      READ(105,4)J,N
4     FORMAT(I3,I2)
```

Ces instructions indiquent à l'ordinateur que la valeur de la variable entière J est placée dans les 3 premières colonnes de l'écran (colonnes 1 à 3) et que la valeur de la variable entière N est placée dans les deux colonnes suivantes (4 et 5).

Exemple 1.11.2

```
      WRITE(108,6)JB,NA
6     FORMAT(I4,I2)
```

Ces instructions indiquent à l'ordinateur d'imprimer la valeur de la variable entière JB dans les quatre premières positions d'une ligne et la valeur de la variable entière NA dans les deux positions suivantes.

Le type F

Il permet de lire ou d'imprimer des nombres réels.

Exemple 1.11.3

```
      READ(105,2)A,B
2     FORMAT(F5.2,F4.2)
```

Ces instructions indiquent à l'ordinateur que la valeur de la variable réelle A est placée dans les 5 premières colonnes de l'écran (colonnes 1 à 5) et comporte 2 chiffres décimaux et que la valeur de la variable réelle B est placée dans les 4 colonnes suivantes (colonnes 6 à 9) et comporte 2 chiffres décimaux.

Exemple 1.11.4

```
      WRITE(108,7)A,B
7     FORMAT(F10.5,F8.3)
```

Ces instructions indiquent à l'ordinateur d'imprimer la valeur de la variable réelle A dans les 10 premières positions d'une ligne et la valeur de la variable réelle B dans les 8 positions suivantes.

Le type E

Le type E permet de lire ou imprimer des nombres réels avec exposant. La donnée associée à ce code se présente comme suit :

Forme normalisée	Forme non normalisée
SGO.xxxxxxxEsgxx	SGxxxxx.xxxEsgxx
←——— d ———→	←——— d ———→
←——— w ———→	←——— w ———→

où

SG = signe du nombre

sg = signe de l'exposant

x = chiffre décimal

d = nombre de décimales

w= longueur du champ

Remarques

SG peut être omis si on a le signe +

Le . peut être omis

La lettre E peut être omise si le signe sg est présent.

Exemple 1.11.5

Soient A=-124.79364 et B=47.1986. En utilisant le format suivant, placer les valeurs de A et de B sur la ligne de l'écran :

```
      READ(105,1)A,B
1     FORMAT(E12.8,E11.8)
```

ligne ⟶ 12345678901234567890123456~~7890~~ N° de colonnes

-12479364+0347198600+02

Soient A=+12.57 et C=1.532. En utilisant le format suivant, placer les valeurs de A et C :

```
      READ(105,3)A,C
3     FORMAT(E10.2,E8.2)
```

12345678901234567890123456~~7890~~ N° de colonnes

12.57E+00 15.32-01

ou bien

125.7-011532E-01

Remarque

Pour que l'impression d'une valeur selon le format Ew.d se fasse correctement, w doit vérifier : w>d+7.

Exemple 1.11.6

Soient A=-124.79364 et B=47.1986. Qu'imprimera l'ordinateur après exécution des deux instructions suivantes ?

```
      WRITE(108,2)A,B
2     FORMAT(E17.8,E15.8)
```

L'ordinateur imprimera :

bb-0.12479364^{E}+03bb0.47198600+02

ou bien

bb-1.24793640^{E}+02b4.71986000^{E}+01

où b=blanc.

Le type X

Le type X permet de sauter des colonnes en entrée et de placer des blancs en sortie. Sa forme générale est nX, où n est une constante entière non signée indiquant le nombre de blancs à mettre en sortie ou le nombre de colonnes à sauter en entrée.

Exemple 1.11.7

```
      READ(105,2)A,B
2     FORMAT(F6.0,6X,E7.0)
```

Ces deux instructions indiquent à l'ordinateur que la valeur de A se trouve entre les colonnes 1 et 6, que le contenu des colonnes 7 à 12 doit être ignoré et que la valeur de B est entre les colonnes 13 et 19.

Exemple 1.11.8

```
      A=12.70
      B=17.0
      WRITE(108,3)A,B
3     FORMAT(1X,F7.2,4X,E10.3)
```

Après exécution de ces instructions, l'ordinateur imprimera :

```
bb12.70bbbbb0.170Eb02      (b=blanc)
```

Remarque

Le premier caractère de chaque instruction FORMAT n'est jamais imprimé. Il sert au contrôle de chariot. En effet, dans l'exemple ci-dessus, le premier blanc n'a pas été imprimé.

Exemple 1.11.9

```
      I=-2
      B=3.33
      WRITE(6,2) I,B
2     FORMAT(5X,I2,4X,F4.2)
```

Après exécution de ces instructions, l'ordinateur imprimera :

```
bbbb-2bbbb3.33
```

Type / (slash)

Le type / permet le passage à la ligne et le saut de lignes.

Exemple 1.11.10

```
      N=12
      H=13.76
      WRITE(6,1)N,H
1     FORMAT(2X,I2,/,1X,F5.2)
```

Après exécution de ces instructions l'ordinateur imprimera :

```
      b12
      13.76
```

Si nous désirons sauter plusieurs lignes, il suffit d'utiliser autant de slashes (/). Ainsi n slashes (immédiatement après la parenthèse de gauche ou immédiatement avant la parenthèse de droite) provoqueront un saut de n lignes. Dans le cas contraire, n slashes causeront l'insertion de (n-1) lignes blanches entre les deux lignes imprimées.

1.12. Les répétitions de type

Une liste de types placés entre parenthèses est appelée groupe.

Exemple 1.12.1

(F5.2,I2) est un groupe.

Un groupe peut être inclus dans un autre groupe.

Exemple 1.12.2

```
3       FORMAT(F5.3,((I4,E13.7),3(I2,4F6.1)))
```

La répétition d'un FORMAT se fait toujours au début de celui-ci s'il ne renferme pas de groupe. Dans le cas contraire, la répétition se fait au début du groupe le plus intérieur. S'il y en a plus d'un, le groupe de droite est répété.

Exemple 1.12.3

```
        DIMENSION A(10)
        A(1)=21
        A(2)=22
        A(3)=23
        A(4)=24
        A(5)=25
        WRITE(108,1)(J,A(J),J=1,5)
1       FORMAT(2(I3,4X,F4.0),(I3,4X,F5.0))
```

Après exécution de ces instructions l'ordinateur imprimera :

b1bbbbb21.bb2bbbbb22.bb3bbbbbb23.
b4bbbbbb24.
b5bbbbbb25.

Mécanisme de l'impression

Caractère de contrôle	**Effet sur l'impression**
blanc	*passage à la ligne*
0	*saut d'une ligne*
1	*passage à la page suivante*
+	*suppression de l'avancement*

CHAPITRE II : LES INSTRUCTIONS DE CONTROLE

2.1. Introduction

Le but de ce chapitre est d'introduire des instructions permettant au programmeur d'en exécuter et d'en sauter sous certaines conditions et de lire de nouvelles données.

Les instructions de branchement permettent au programmeur d'interrompre la séquence d'exécution et de commencer, en un point déterminé, l'exécution du programme.

2.2. Branchement inconditionnel

Sa forme générale est :

```
GO TO n
```

où n est le numéro de l'instruction à laquelle on veut se transférer. Ce numéro est un entier positif dont le nombre de chiffres ne dépasse pas cinq. On le place entre la première et la cinquième colonne. Cette instruction est interprétée par le compilateur de la manière suivante : aller à l'instruction numéro n et commencer l'exécution du programme à partir de celle-ci.

Exemple 2.2.1

```
      READ(105,3)A,B
3     FORMAT(F5.2,F6.2)
      S=A+B
      GO TO 7
      S=2*A
7     WRITE(108,4)S
4     FORMAT(3X,F7.3)
      STOP
      END
```

2.3. Branchement calculé

Sa forme générale est :

```
GO TO(n1,n2,...,nm),I
```

où n1, n2, ..., nm sont des numéros d'instruction et I est une variable entière comprise entre 1 et m. Cette instruction indique à l'ordinateur que la prochaine instruction à exécuter est nj, où j=I et I=1,2,...,m.

Remarque

Les numéros d'instructions n1, n2, ..., nm doivent figurer dans le programme.

Exemple 2.3.1

```
      ISAUT=3
      GO TO(7,23,29),ISAUT
      L=2*ISAUT
      GO TO 1
29    L=ISAUT-1
1     WRITE(108,2)L
```

```
2       FORMAT(10X,I1)
        STOP
        END
```

Dans cet exemple, l'instruction L=ISAUT-1 est exécutée immédiatement après l'exécution du GOTO calculé. La valeur de L est donc 2.

2.4. Branchement imposé

Sa forme générale est :

```
ASSIGN n TO I
GO TO I,(n1,n2,...,nm)
```

où n, n1, n2, ..., nm sont des numéros d'instructions et I est une variable entière dont la valeur est égale à n. Cette instruction permet de transférer le contrôle à l'instruction dont le numéro est n.

Remarque

Les numéros d'instructions n1, n2, ..., nm doivent figurer dans le programme.

Exemple 2.4.1

```
        ASSIGN 4 TO I
             ........
             ........
        GO TO I,(1,4,3,2,5)
            ........
4       AV=(A+B)/2
        WRITE(108,7)AV
7       FORMAT(4X,F5.3)
        STOP
        END
```

Dans cet exemple, l'instruction numéro 4 sera exécutée immédiatement après l'exécution du GO TO.

2.5. IF arithmétique

Sa forme générale est :

```
IF (e) n1,n2,n3
```

où e est une expression arithmétique et n1, n2 et n3 sont des entiers non négatifs désignant des numéros d'instructions. Le compilateur interprète cette instruction de la façon suivante :

Si e<0 alors l'instruction n1 est exécutée

Si e=0 alors l'instruction n2 est exécutée

Si e>0 alors l'instruction n3 est exécutée.

Remarques

a) Les numéros n1,n2 et n3 peuvent être égaux

b) Les numéros d'instructions n1, n2 et n3 doivent figurer dans le programme.

Exemple 2.5.1

```
      READ(105,1)A,B,C
1     FORMAT(3F5.2)
      IF(A) 14,15,14
15     X=-C/B
      WRITE(108,17)X
17     FORMAT(5X,'La sol.de l'équ.du 1er degré est:',F5.2)
      GO TO 6
14     DEL=B**2-4*A*C
      IF(DEL) 5,12,2
2     X1=(-B+SQRT (DEL))/(2*A)
      X2=(-B-SQRT (DEL))/(2*A)
      WRITE(108,7)X1,X2
7     FORMAT(5X,F5.2,3X,F5.2)
      GO TO 6
12     X1=-B/(2*A)
      X2=X1
      WRITE(108,3)X1,X2
3     FORMAT(5X,'L''éq. possède une racine double',F5.2,3X,F5.2)
      GO TO 6
5     WRITE(108,8)
8     FORMAT(5X,'L''éq. ne possède pas de racine réelle')
6     STOP
      END
```

2.6. IF logique

Sa forme générale est :

```
IF(e)s
```

où e est une expression logique pouvant prendre soit la valeur .TRUE. ou bien la valeur .FALSE. et s est une instruction FORTRAN autre que les instructions IF arithmétique, IF logique et DO (voir prochain chapitre). Le compilateur interprète l'instruction IF logique de la manière suivante :

Si la valeur de l'expression logique e est vraie (.TRUE.), alors l'instruction s est exécutée ensuite les instructions qui la suivent, sauf si s est l'instruction GO TO

Si la valeur de l'expression logique e est fausse (.FALSE.), l'instruction immédiatement après le IF logique est exécutée.

Exemple 2.6.1

```
      NSOM=0
      I=1
23    NSOM=NSOM+I
      I=I+1
      IF(I.LE.10) GO TO 23
      WRITE(5,3)NSOM
3     FORMAT(6X,I3)
      STOP
      END
```

CHAPITRE III : BOUCLES DO ET TABLEAUX

Le but de ce chapitre est d'introduire l'instruction DO et les tableaux.

3.1. Boucles

3.1.1 Boucle DO

$1^{ère}$ forme:

```
      DO n I=m1,m2,m3
        Inst1
        Inst2
          ⋮
n       Instp
```

où

- n est le numéro de la dernière ligne sous le contrôle de DO
- I est une variable entière appelée indice de comptage
- m1 est une constante entière non signée ou une variable entière non signée. On l'appelle valeur initiale de l'indice de comptage I
- m2 est une constante entière non signée ou une variable entière non signée. On l'appelle valeur finale de l'indice de comptage I
- m3 est une constante entière non signée ou une variable entière non signée. On l'appelle valeur de l'incrément de l'indice de comptage.

Inst1, Inst1,…, Instp sont des instructions à exécuter

L'instruction DO indique à l'ordinateur d'exécuter d'une manière répétitive les instructions se trouvant entre l'instruction DO et l'instruction dont l'étiquette est n, y compris cette dernière.

Remarque

Les valeurs de m1,m2 et m3 doivent être supérieures à 0. Pour indiquer la fin de la boucle DO, on utilise l'instruction CONTINUE.

Exemple 3.2.1

```
      N=0
      DO 3 I=1,10,3
```

```
      N=N+I
3     WRITE(108,1)N
1     FORMAT(2X,I3)
```

Le programme ci-dessus et le suivant ont le même effet :

```
      N=0
      DO 3 I=1,10,3
      N=N+I
      WRITE(108,1)N
3     CONTINUE
1     FORMAT(2X,I3)
```

L'instruction CONTINUE est optionnelle.

2ème forme:

```
DO   I=m1,m2,m3
       Inst1
       Inst2
         ⋮
      Instp
END DO
```

3.1.2 Boucle DO WHILE

DO WHILE (condition)

```
       Inst1
       Inst2
         ⋮
       Instp
```

END DO

Les insrtructions Inst1, Inst1,…, Instp sont exécutées d'une manière répétitive tant que la condition est vraie.

Exemple 3.2.2

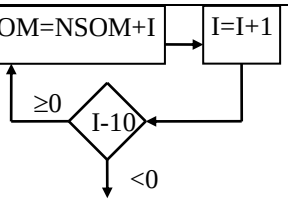
L' Exemple 3.2.1 peut être réécrit à l'aide de la boucle **DO WHILE**

```
      N=0
      I=1
```

```
      DO WHILE (I.LE.10)
      N=N+I
      WRITE(108,1)N
1     FORMAT(2X,I3)
      END DO
```

3.2. Analogie entre le compteur et la boucle DO

Boucle DO		COMPTEUR	
	NSOM=0		NSOM=0
	DO 23 I=1,10,1		I=1
23	NSOM=NSOM+I	23	NSOM=NSOM+I
			I=I+1
			IF(I.LE.10)GO TO 23



Cette analogie entre le compteur et la boucle DO nous montre que :

1. I est initialisée à la valeur m1=1, après avoir exécuté une fois les instructions dans la boucle
2. A chaque itération, l'indice I est incrémenté de la valeur m3=1
3. Lorsque la valeur prise par l'indice I dépasse m2=10, l'exécution de la boucle s'arrête.
4. L'instruction suivant celle dont le numéro est n=23 est exécutée.

Remarques

- **Les paramètres I, m1, m2 et m3 ne doivent pas être changés à l'intérieur de la boucle :**

Exemple 3.2.1

```
      SOM=0.
      DO 7 I=1,10,1
      IF(I.EQ.3) I=I+1      (Cette instruction n'est pas acceptée à
7     SOM=SOM+1             cause de l'instruction d'affectation I=I+1)
```

- **Deux boucles quelconques ne doivent jamais se croiser :**

Exemple 3.2.2

```
      S=O.
      DO 1 I=1,5,1
      A=I*I
      DO 2 J=1,5,1
      S=S+A
1     CONTINUE
      WRITE(6,1)S
2     CONTINUE
3     FORMAT(2X,F6.2)
```

- **Un transfert hors de la boucle est permis à tout moment :**

Exemple 3.2.3

```
      X=0.
      DO 100 J=1,20
      A=X
      X=cos(X)
      IF(ABS (X-A).LE.0.0001) GO TO 200
```

```
100   WRITE(108,10)I,X
10    FORMAT(2X,I2,F5.2)
200   STOP
```

- **Toute entrée dans la boucle ne peut être faite que par l'instruction DO :**

Exemple 3.2.4

```
      I=5
      GO TO 7
      .......
      .......
      DO 6 I=1,12
7     ......
      .......
6     .......
```

- **La dernière instruction d'une boucle DO peut être n'importe quelle instruction exécutable à l'exception de GO TO, IF arithmétique ou une autre instruction DO. La dernière instruction d'une boucle peut être un IF logique. Afin d'éviter que la boucle DO ne se termine par une de ces instructions, on utilise l'instruction CONTINUE comme dernière instruction de la boucle.**

Exemple 3.2.5

```
      DO 7 I=1,5
      IF(t)s
7     CONTINUE
```

Si la valeur de t est vraie (.TRUE.), l'instruction s est exécutée et l'indice I est incrémenté. Si elle est fausse (.FALSE.), l'indice I est incrémenté sans que s soit exécuté.

3.3. Tableau unidimensionnel

Un ensemble d'objets de même espèce est appelé tableau. Par exemple, les températures prélevées quotidiennement constituent un ensemble de nombres de même espèce ; c'est un tableau. Désignons le par T. Pour distinguer une température d'une autre, on les numérote ou indice. En FORTRAN, on utilise les variables indicées.

Une variable indicée est une variable représentant un élément du tableau. En FORTRAN, l'élément numéro I d'un tableau T est noté T(I).

L'instruction DIMENSION

Sa forme générale est :

```
DIMENSION liste
```

où liste est une suite de noms de tableaux. Cette instruction spécifie la taille (nombre d'éléments) des tableaux utilisés dans un programme. Elle est non exécutable. Elle se place au début d'un programme, avant toute autre instruction exécutable.

Exemple 3.3.1

```
DIMENSION A(4),B(3),N(2)
```

Cette instruction est interprétée par le compilateur de la manière suivante :

Réserver 4 espaces-mémoire consécutifs pour les 4 éléments A(1),A(2),A(3) et A(4) du tableau A.

Réserver 3 espaces-mémoire consécutifs pour les 3 éléments B(1),B(2) et B(3) du tableau B.

Réserver 2 espaces-mémoire consécutifs pour les 2 éléments N(1) et N(2) du tableau N.

Remarques

1. La taille du tableau doit être au moins égale au nombre maximum d'éléments qu'on veut enregistrer.
2. La valeur de l'indice doit être un nombre entier au moins égal à 1 et au plus égal à la taille.
3. L'indice ne peut pas être négatif, ni une variable réelle ni une variable indicée. Cependant, l'indice peut être une constante entière, une variable entière ou une expression entière.
4. Les règles de déclaration de type applicables aux variables non indicées restent aussi applicables aux tableaux.

Exemple 3.3.2

Le tableau NA définit des valeurs entières.

Exemple 3.3.3

Le tableau A définit des valeurs réelles.

Les instructions suivantes sont permises :

```
A(I),A(I+2),A(3*L),A(5)
```

Par contre, les instructions suivantes ne sont pas permises :

```
A(-I),A(I(J)),A(5.)
```

3.4. Lecture et écriture d'un tableau

Le moyen le plus direct pour lire et écrire les éléments d'un tableau est de lister explicitement chaque élément du tableau.

Exemple 3.4.1

```
      DIMENSION A(4),B(2)
      READ(105,2)A(1),A(2),A(3),A(4),B(1),B(2)
2     FORMAT(6F10.4)
      STOP
      END
```

Les 4 éléments de A et les 2 éléments de B peuvent être écrits dans n'importe quel ordre. La seule restriction est qu'une fois un ordre est choisi, les valeurs de ces variables indicées doivent être placées dans ce même ordre sur une ligne de l'écran.

Un avantage des tableaux est qu'on peut lire et écrire leurs éléments sans être obligé de les lister individuellement. En d'autres termes, il est suffisant de spécifier le nom du tableau.

Exemple 3.4.2

```
      DIMENSION A(4),B(2)
```

```
      READ(105,2)A,B
2     FORMAT(6F10.4)
      STOP
      END
```

Les instructions dans les exemples 3.4.1 et 3.4.2 sont équivalentes. Les données doivent être placées dans l'ordre suivant : A(1),A(2),A(3),A(4),B(1),B(2).

Exemple 3.4.3

```
      DIMENSION A(10)
      READ(105,2)A
2     FORMAT(F10.2)
      STOP
      END
```

Les dix valeurs du tableau A doivent être placées sur dix lignes comportant chacune une valeur : les valeurs de A(1),A(2),...,A(10) doivent être placées respectivement dans les dix premières colonnes de la première ligne, les dix premières colonnes de la deuxième ligne, etc.

Exemple 3.4.4

```
      DIMENSION X(5)
      READ(105,2)X
2     FORMAT(5F10.4)
      STOP
      END
```

Exemple 3.4.5 Utilisation de la boucle DO

```
      DIMENSION X(5)
      DO 1 I=1,5
1     READ(2,4) X(I)
4     FORMAT (F10.3)
      STOP
      END
```

Les cinq valeurs du tableau X doivent être placées sur une seule ligne. Les valeurs de X(1),...,X(5) doivent être respectivement dans les colonnes 1-10,11-20,...,41-50.

3.5. Variables multidimensionnelles

En FORTRAN, une variable indicée peut avoir un, deux ou trois indices. On dit qu'elle est respectivement unidimensionnelle, bidimensionnelle et tridimensionnelle.

Exemple 3.5.1

A(2,5) est un élément du tableau bidimensionnel A

B(2,4,7) est un élément du tableau tridimensionnel B.

3.6. Taille d'une variable multidimensionnelle

La taille d'une variable indicée est déterminée par le nombre d'indices et la valeur maximale de chaque indice. Cette taille est transmise à l'ordinateur par l'instruction DIMENSION.

Exemple 3.6.1

```
DIMENSION A(2,3),K(2,3,4)
```

Cette instruction indique à l'ordinateur de réserver 2*3=6 cases mémoire pour le tableau A et 2*3*4=24 cases mémoire pour le tableau B.

3.7. Arrangement d'un tableau en mémoire centrale

Les éléments d'un tableau sont enregistrés colonne par colonne en mémoire, en commençant par la première.

Exemple 3.7.1

Soit un tableau B dont le premier indice varie de 1 à 5 et dont le deuxième indice varie de 1 à 3. Alors les éléments de B seront rangés selon l'ordre suivant :

B(1,1), B(2,1), B(3,1), B(4,1), B(5,1), B(1,2), B(2,2), B(3,2), B(4,2), B(5,2), B(1,3), B(2,3), B(3,3), B(4,3), B(5,3).

Soit un tableau C dont le premier indice varie de 1 à 3, le 2 indice de 1 à 2 et le 3 indice de 1 à 3. Alors, les éléments de C seront rangés comme suit :

C(1,1,1), C(2,1,1), C(3,1,1), C(1,2,1), C(2,2,1), C(3,2,1), C(1,1,2), C(2,1,2), C(3,1,2), C(1,2,2), C(2,2,2), C(3,2,2), C(1,1,3), C(2,1,3), C(3,1,3), C(1,2,3), C(2,2,3), C(3,2,3).

3.8. Lecture des éléments d'un tableau multidimensionnel

Exemple 3.8.1

```
DIMENSION A(6,3)
READ(105,1)A
```

Les éléments de A seront lus dans l'ordre selon lequel ils sont rangés en mémoire, c'est à dire colonne par colonne.

3.9. Boucle implicite

Parfois, il est désirable de ne lire ou écrire qu'une partie d'un tableau. Pour ce faire, on utilise la boucle implicite. Sa forme générale est :

```
...(V(I),I=m1,m2,m3)
```

où

- les trois points de suspension désignent soit READ ou WRITE V est le nom d'un tableau
- I est l'indice de comptage
- m1 est la valeur initiale de l'indice de comptage I. C'est soit une constante entière non signée ou une variable entière non signée.
- m2 est la valeur finale de l'indice de comptage I. C'est soit une constante entière non signée ou une variable entière non signée.
- m3 est l'incrément. C'est soit une constante entière non signée ou une variable entière non signée.

Notons que m1>0,m2>0 et m3>0.

Exemple 3.9.1

```
      DIMENSION A(3),B(3)
      READ(105,1)(A(I),B(I),I=1,2,1)
1     FORMAT(F10.2)
```

Ces instructions indiquent à l'ordinateur que la valeur de A(1) est dans les 10 premières colonnes de la première ligne, la valeur de B(1) est dans les 10 premières colonnes de la deuxième ligne, la valeur de A(2) est dans les 10 premières colonnes de la troisième ligne et la valeur de B(2) est dans les 10 premières colonnes de la quatrième ligne.

CHAPITRE IV : LES SOUS-PROGRAMMES

4.1. Introduction

Nous avons déjà introduit et utilisé plusieurs fonctions, par exemple SQRT (racine carrée), SIN (sinus), etc. Ces fonctions sont conçues par le constructeur et fournies avec le compilateur FORTRAN. On les appelle fonctions bibliothèque. Elles peuvent être utilisées dans n'importe quel programme. Cependant, le programmeur peut définir ses propres fonctions en utilisant :

Les formules (fonctions) arithmétiques

Les sous-programmes fonctions

Les sous-programmes sous-routines (subroutines).

4.2. Les fonctions bibliothèque

Afin d'alléger la tâche du programmeur, le constructeur a développé des programmes qui calculent les fonctions mathématiques les plus utilisées : fonctions trigonométriques et logarithmiques, fonction exponentielle, fonction racine carrée, fonction valeur absolue, etc. Chaque fonction est identifiée par un nom que le constructeur lui donne. Ces fonctions (sous-programmes) sont exécutées dès qu'on les appelle par leur nom.

Pour appeler une fonction (sous-programme), le programmeur doit simplement écrire son nom dans une expression.

Exemple 4.2.1

Calcul de la racine carrée d'un nombre :

```
Y=SQRT (X)
```

Dans cette instruction, SQRT désigne le nom de la fonction racine carrée en FORTRAN. L'argument pour lequel elle sera évaluée est X. La présence des parenthèses est obligatoire.

Nous donnons ci-après un tableau des fonctions bibliothèque les plus courantes :

NOM	FONCTION	TYPE ARGUMENT	TYPE FONCTION
SIN	sinus	réel	réelle
COS	cosinus	réel	réelle
EXP	exponentielle	réel	réelle
ALOG	logarithme népérien	réel	réelle
SQRT	racine carrée	réel	réelle
ABS	valeur absolue	réel	réelle
IABS	valeur absolue	entier	entière
FLOAT	conversion d'argument (d'entier à réel)	entier	réelle

Exemple 4.2.2

L'instruction :

```
FLOAT(I)
```

transforme le type entier de la variable I en type réel. Elle a donc le même effet que l'instruction :

```
AI=I
```

Remarque

Le programmeur n'est pas obligé de savoir comment ces fonctions sont construites. Cependant, il peut toujours développer ses propres sous-programmes en utilisant des expressions arithmétiques élémentaires. Par exemple, le programme suivant calcule la valeur absolue d'un nombre X :

```
      X=-12.89
      IF(X)3,4,4
3     VABS=-X
      GO TO 5
4     VABS=X
5     WRITE(6,2)X,VABS
2     FORMAT(5X,'La valeur absolue de',F7.2,2X,4X,'est',5X,F6.2)
      STOP
      END
```

Le résultat de ce programme est : la valeur absolue de -12.89 est 12.89.

4.3. Formule (fonction) arithmétique

Si une série de calculs est utilisée de façon répétée et si elle peut être exprimée par une expression arithmétique, il est avantageux d'utiliser la formule arithmétique. Sa forme générale est :

```
ide(x1,x2,...,xn)=expression arithmétique
```

où ide désigne le nom de la formule arithmétique. Les mêmes règles de formation des noms de variables s'appliquent à la construction des noms de formules arithmétiques. x1,x2,...,xn sont les arguments de la formule arithmétique. Ce sont des variables fictives, non indicées et distinctes.

Voici quelques règles concernant la formule arithmétique :

1) Elle doit être définie par une seule instruction

2) Elle est non exécutable

3) Elle doit être placée avant toute instruction exécutable, mais après les déclarations de type ou DIMENSION s'il y a lieu

4) Elle doit avoir au moins un argument, et si elle contient plus d'un argument, ces derniers doivent être séparés par des virgules

5) Elle peut contenir des variables autres que les arguments.

Exemple 4.3.1

```
RES(T)=A+T*(B+C*T)
F(X)=X**2+X-1
S(X,Y)=X+Y
```

Exemple 4.3.2

```
JTIME(I,J,K)=I*2+J-K
```

ou, de manière équivalente,

```
INTEGER TIME
TIME(I,J,K)=I*2+J-K
```

Exemple 4.3.3

```
AMOY(X,Y,Z)=(X+Y+Z)/3
R=AMOY(12.,13.,11.)
```

La première instruction définit la fonction. La deuxième instruction affecte la valeur de la formule arithmétique AMOY(12.,13.,11.)=(12.+13.+11.)/3=12. à la variable R.

Remarque

La première instruction de cet exemple est exécutée dès que la deuxième l'est.

4.4. Les sous-programmes fonctions

Les sous-programmes fonctions sont utilisés lorsqu'un calcul se répète fréquemment dans un programme et ne peut pas être fait à l'aide d'une seule instruction. Le sous-programme fonction ressemble presque à un programme ordinaire. Il se distingue par deux instructions : FUNCTION et RETURN.

L'instruction FUNCTION

Sa forme est :

```
FUNCTION ide(x1,x2,...,xn)
```

où ide est le nom de la fonction. Il reçoit le résultat du sous-programme fonction. x1,x2,...,xn sont des arguments. Ils peuvent être des tableaux (sans mention de l'indice), des noms de fonctions ou des noms de sous-routines. Ils sont considérés comme des variables fictives (muettes), c'est à dire qu'ils ne prennent de valeurs qu'au moment de l'exécution du sous-programme.

L'instruction RETURN

Elle indique la fin de l'exécution du sous-programme et permet de revenir au programme appelant (principal).

Exemple 4.4.1

Le sous-programme suivant calcule factorielle n (n !) :

```
      FUNCTION IFACT(N)
      IFACT=1
      DO 1 I=1,N
1     IFACT=IFACT*I
      RETURN
```

```
      END
```

Le sous-programme fonction de cet exemple ne peut pas être exécuté seul. Il doit être appelé par un programme appelé programme principal (ou maître). Celui-ci se distingue d'un programme ordinaire par au moins une instruction contenant le nom du sous-programme fonction.

Remarque

Le nom du sous-programme doit apparaître au moins une fois comme variable à gauche du signe =.

Exemple 4.4.2

Pour appeler (exécuter) le sous-programme fonction de l'exemple 4.4.1, nous devons écrire un programme principal. Les instructions suivantes constituent le programme principal appelant IFACT :

```
      S=1.
      DO 1 I=1,15
      S=S+1./IFACT(I)
      WRITE(6,2)I,S
2     FORMAT(2X,I3,2X,E15.10)
1     CONTINUE
      STOP
      END
```

L'exécution de ce programme se fait normalement jusqu'à ce que l'ordinateur arrive à l'instruction contenant le nom du sous-programme IFACT. A cette instruction, l'exécution du programme principal s'arrête temporairement, pendant que le sous-programme IFACT est exécuté. L'exécution de ce dernier achevée, celle du programme principal reprend à partir de l'instruction où s'est arrêtée son exécution.

Remarque

Après exécution du sous-programme IFACT, la valeur de IFACT, calculée en fonction de l'expression à droite du signe =, est retournée au programme principal.

Exemple 4.4.3

Supposons que l'on désire calculer le nombre de combinaisons de n éléments pris k à k, c'est à dire la valeur : $C_k^n = \frac{n!}{k!(n-k)!}$. Le programme principal qui nous permet de faire ce calcul est :

```
      READ(5,1)N,K
1     FORMAT(I3)
      NCOMB=IFACT(N)/(IFACT(K)*IFACT(N-K))
      WRITE(6,2)N,K,NCOMB
2     FORMAT(4X,I3,2X,I3,2X,I8)
      STOP
      END
```

4.5. Les sous-programmes sous-routines

Ils permettent le transfert de plusieurs valeurs au programme principal. Dans une sous-routine (ou subroutine), la première instruction doit être : SUBROUTINE. Sa forme générale est :

```
SUBROUTINE ide(x1,x2,...,xn)
```

où ide est le nom de la sous-routine x1,x2,...,xn sont des arguments qu'on considère comme variables fictives. Ces arguments peuvent être des tableaux (sans mention d'indice), des nom de fonctions ou d'autres sous-routines.

L'appel se fait à l'aide de l'instruction CALL dont la forme générale est :

```
CALL ide(y1,y2,...,yn)
```

où ide est le nom de la sous-routine y1,y2,...,yn sont les variables de données ou de résultats dont la sous-routine dépend. On les appelle variables actuelles. Elles peuvent être :

des variables indicées ou non

des constantes

des expressions arithmétiques

des noms de sous-routines ou des noms de fonctions.

Remarque

Les arguments actuels doivent être conforme en nombre, ordre et type à ceux indiqués dans l'instruction SUBROUTINE.

Lorsqu'un nom de sous-routine ou de fonction est utilisé comme argument actuel dans un programme principal, ce nom doit apparaître dans une instruction EXTERNAL du programme principal. La forme générale de cette dernière est :

```
EXTERNAL a,b,c...
```

où a,b,c,... sont les noms des sous-routines ou des fonctions figurant dans la liste des arguments.

Exemple 4.5.1

```
      EXTERNAL FONCT
      READ(5,1)A,B
1     FORMAT(2F8.3)
      CALL TRINT(A,B,FONCT,50,S)
      WRITE(6,2)S
      FORMAT(F10.5)
      STOP
      END
```

Les instructions ci-dessus constituent un programme principal.

```
      SUBROUTINE TRINT(A,B,F,N,S)
      H=(B-A)/FLOAT(N)
      S=0.
      X=A
      F1=F(A)
      DO 4 I=1,N
```

```
      X=X+H
      F2=F(X)
      S=S+(F1+F2)*H/2
      F1=F2
4     CONTINUE
      RETURN
      END
```

Ces instructions constituent la sous-routine qui calcule la valeur approchée de l'intégrale d'une fonction f par la méthode des trapèzes.

```
      FUNCTION FONCT(X)
      FONCT=1/SQRT (1.+X**5)
      RETURN
      END
```

Ces instructions constituent le sous-programme fonction qui calcule la valeur de la fonction f au point X.

Exemple 4.5.2

```
      DIMENSION A(50)
      READ(5,1)N,(A(I),I=1,N)
1     FORMAT(I2/(F7.2))
      WRITE(6,6)
6     FORMAT(2X,'La liste non triée est:',/,19(H*))
      DO 7 I=1,N
      WRITE(6,5)A(I)
7     CONTINUE
      CALL TRI(N,A)
      WRITE(6,3)
3     FORMAT(2X,'La liste triée est:',/,2X,19(H*))
      DO 4 I=1,N
      WRITE(6,5)A(I)
5     FORMAT(9X,F7.3)
4     CONTINUE
      STOP
      END
```

Ces instructions constituent un programme principal. Les instructions qui suivent constituent une sous-routine qui trie la liste A(1),A(2),...,A(N) :

```
      SUBROUTINE TRI(N,A)
      DIMENSION A(50)
      NM1=N-1
      DO 21 I=1,NM1
      IP1=I+1
      M=I
      DO 31 J=IP1,N
      IF(A(M)-A(J))41,31,31
41    M=J
31    CONTINUE
      TEMP=A(I)
```

```
      A(I)=A(M)
21    A(M)=TEMP
      RETURN
      END
```