Programmation en FORTRAN 77

FORTRAN est le langage de programmation des ordinateurs pour applications scientifiques et techniques.

Dans le cadre de cet ouvrage nous étudierons le FORTRAN 77, c' est à dire le langage adopté par la norme ANSI 1977 et qui est utilisé actuellement dans toutes les machines.

Ce livre est composé de deux parties. Dans la premiére partie, nous rappelons les élements du langage FORTRAN et dans la deuxième partie nous traitons de plusieurs exercices corrigés, permettant à l' étudiant la maîtrise des structures algorithmiques numériques existantes et les structures de données manipulées en FORTRAN.
Afin que l' utilisateur s' entraine efficacement dans la pratique de la programation FORTRAN, les solutions d' exercices sont proposées sous différentes tactiques.

Ce livre s' adresse aux étudiants de toutes les filières scientifiques et techniques ainsi qu' aux praticiens.

N & S TAIBI

Programmation en FORTRAN 77

BERTI Editions

N & S TAIBI

Pratique du FORTRAN 77

COURS ET EXERCICES RÉSOLUS

2éme Edition, revue et corrigée

BERTI Editions

N & S TAIBI

la programmation en FORTRAN 77

COURS ET EXERCICES RÉSOLUS

2ème Edition, revue et corrigée

BERTI
Editions

© BERTI Editions
Rue Ahmed OUAKED , Lot ENNADJAH
DELY IBRAHIM
16 320 ALGER

TABLE DES MATIÈRES

PREMIERE PARTIE: COURS 11

1.GENERALITES 13

2.NOTIONS SUR LES ALGORITHMES ET LES ORGANIGRAMMES 23

3.LES INSTRUCTIONS DE FORTRAN 77 35

4.LES SOUS-PROGRAMMES ET LES INSTRUCTIONS D'EFFICACITE DE PROGRAMMATION 53

5.LES FICHIERS 63

DEUXIEME PARTIE EXERCICES 69

1.EXERCICES DIVERS 71

2.LES VARIABLES INDICEES 167

3.TRAITEMENT DES CHAINES DE CARACTERES 241

4.LES SOUS-PROGRAMMES ET LES INSTRUCTIONS D'EFFICACITE DE PROGRAMMATION 252

5.LES FICHIERS 265

ANNEXE 271

LES FONCTIONS STANDARDS 271

AVANT-PROPOS

Le but recherché à travers cet ouvrage est d'initier l'étudiant à la programmation en FORTRAN 77 , ainsi qu'à la programmation structurée à partir d'algorithmes .

Nous avons mis à sa portée un nombre considérable d'exercices programmés en FORTRAN 77pour lui inculquer des notions de programmation .

Il n'est nullement dans notre intention de proposer des programmes optimums, c'est la raison pour laquelle plusieurs programmes sont donnés pour un même exercice .

Deux types de solutions sont proposés : la première est ALGORITHMIQUE, la seconde utilise des ORGANIGRAMMES .

L'ouvrage est divisé en deux parties : la première est réservée au cours, la seconde aux exercices en 5 chapitres . Le premier chapitre comporte des exercices divers, le second traite les vecteurs et les matrices, le troisième est réservé au type caractère. Dans le quatrième, nous reprenons un grand nombre d'exercices proposés dans les autres chapitres pour les traiter en utilisant des sous-programmes . Vous y trouvez également des exercices relatifs aux instructions d'éfficacité de programmeation . Enfin, le dernier chapitre décrit les fichiers à accés direct et séqentiel .

Tous les programmes ont été compilés et exécutés sur les compilateurs FORTRAN 77de Micro soft .
Les micro-ordinateurs utilisés sont UNISYS et M24, tous deux compatibles IBM, ainsi que le micro-VAX de DIGITAL sur lequel nous avons compilé les programmes utilisant la boucle DO WHILE .

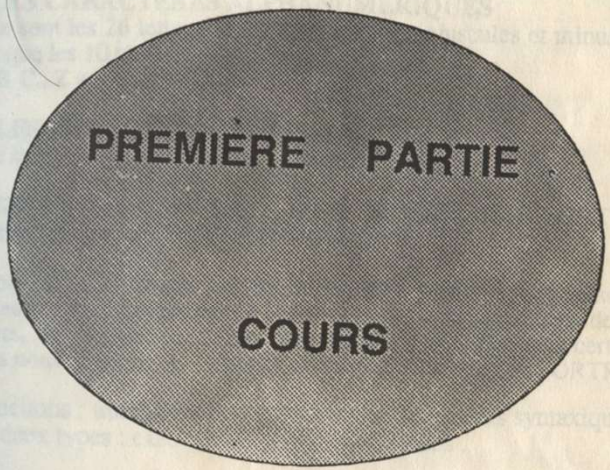
PREPARATION D'UN PROGRAMME

La résolution de n'importe quel problème se ramène à l'écriture d'un certain programme informatique dans un langage spécifique .

La préparation de ce programme se fait toujours en deux étapes complémentaires .

Dans un premier temps, en utilisant les données dont on dispose, une mise en forme de la solution du problème proposé sera réalisée . Elle pourra être élaborée dans un langage naturel, mathématique ou physique, etc, : Cette étape s'appelle l'algorithme de la résolution ; ou s'appuyer sur des représentations graphiques : les organigrammes .

Dans la seconde étape on réalise le codage de l'algorithme dans un langage de programmation (pour nous, ce sera le FORTRAN 77). Une fois le codage réalisé, le programme sera chargé dans la mémoire de l'ordinateur . Il sera compilé puis exécuté.



CHAPITRE.I: GENERALITES

Dans ce premier chapitre, nous décrirons les éléments syntaxiques de FORTRAN .

1. L'ALPHABET FORTRAN

Il est composé de caractères alphanumériques et spéciaux.

1.1 LES CARACTERES ALPHANUMERIQUES

Ce sont les 26 lettres de l'alphabet latin, majuscules et minuscules ainsi que les 10 chiffres

A B C...Z a b c...z 0 1 2..,9

1.2. LES CARACTERES SPECIAUX

Ce sont = + - * / : ( ) ´ . $ et l'espace blanc .

2. STRUCTURE GENERALE D'UN PROGRAMME FORTRAN

Un programme FORTRAN est constitué de :

- Mots : ils sont formés par des lettres, des chiffres et des caractères spéciaux . Ces mots forment à leurs tours des constantes, des identificateurs, des étiquettes et des opérateurs . En appliquant certaines règles nous formons les éléments syntaxiques du langage FORTRAN.

- Instructions : une instruction est une suite d'éléments syntaxiques. Il existe deux types : exécutables et non exécutables .

- Le programme : c' est un ensemble d'instructions et de commentaires . Nous distingons deux types : les programmes principaux et les sous-programmes qui commencent par l'une des instructions FUNCTION ou SUBROUTINE.

Un programme FORTRAN peut avoir la structure suivante:

instructions de déclaration
instructions exécutables et non exécutables
END

3. STRUCTURE D'UNE LIGNE FORTRAN

Chaque ligne contient 80 caractères et est découpée en 4 zonnes :

- Zone A (colonnes 1 à 5) : elle contient un nombre entier qui sert à repérer une instruction; c'est la zone ETIQUETTES.

- Zone B (colonnes 7 à 72) : elle contient l'instruction FORTRAN proprement dite.

- Zone C (colonnes 73 à 80) : généralement elle est utilisée pour numéroter les lignes du programme, elle est ignorée par le compilateur.

- Zone D (colonne 6) : si elle contient un caractère quelconque autre que le zéro, elle spécifie une ligne suite . En effet , si une instruction ne peut être écrite sur une ligne, nous pouvons la faire suivre de 9 lignes au maximum.

Remarques:

- les lignes comportant le caractère C en colonne 1 sont dites lignes commentaires et sont ignorées du compilateur.

- la première ligne d'un programme FORTRAN contient un blanc en colonne 6.

4. LES OPERANDES

Ils sont définis par l'utilisateur en respectant certaines règles.

4.1 LES ETIQUETTES : c'est un nombre sans signe de 1 à 5 chiffre.Elles peuvent apparaitre soit en zone A, soit en zone B.

4.2 LES CONSTANTES : c'est une quantité dont la valeur numérique est fixée et ne peut plus être modifiée pendant l'exécution du programme . Les constantes que nous étudierons sont :

4.2.1 CONSTANTES ENTIERES : elles s'écrivent avec 1 à N chiffres décimaux (N dépend de l'ordinateur utilisé), elles sont précédées ou non d'un signe + ou - . Aucun espace ou point n'est autorisé entre les chiffres composant cette constante .
- exemples : +6 -231 12689

4.2.2 CONSTANTES REELLES SIMPLE PRECISION : deux formes sont possibles, avec exposant (ou point flottant) et sans exposant (ou point fixe).

a- Point fixe : elles s'écrivent avec 1 à N chiffres décimaux avec un point décimal marquant la séparation entre la partie entière et fractionnaire . Cette constante peut être précédée ou non d'un signe .
- exemples : +11.6 -23.12 12. .125

b- Point flottant : dans ce cas la constante possède une mantisse ayant 1 à N chiffres décimaux séparés ou non par un point décimal suivie d'un exposant (base 10) ayant pour symbole la lettre E. Le nombre qui suit E est un entier signé (nous pouvons omettre le signe +) .

- exemples : -17.E+2 115.5E-55 .13E8

4.2.3 CONSTANTES DOUBLE PRECISION : elles s'écrivent en utilisant la forme avec exposant , la lettre E devient D .
-exemples : 55.159874D-5 -.5D+6

4.2.4 CONSTANTES COMPLEXES : elles sont constituées d'un couple de constantes simple précision (avec ou sans exposant) entre parenthèses et séparées par une virgule.
-exemples : (3.1, -1.12) (-12., 5.E+2) (2.E-1, 2.5E+4)

4.2.5 CONSTANTES LOGIQUES : elles sont booléennes (vrai ou faux) . Nous les représentons comme suit :
. TRUE . . FALSE .
Les deux points entourant les mots TRUE et FALSE sont obligatoires.

4.2.6. CONSTANTES CARACTERES : c'est une suite de caractères alphanumériques ou spéciaux encadrés par deux apostrophes. L'apostrophe à l'intérieur d'une chaine de caractères est doublée .
-exemples : ' INFORmatique ' ' électronique ' ' l''élément '

EXERCICE

Donner le type des constantes suivantes :
1220 1.52E+3 -200. +.13 (5.E-2, 0.01)
12.3D+7 -1.665D-5 'F(x)' .TRUE. '.FALSE.'

Réponse :

Constantes entières :1220
Constantes réelles simple précision en point fixe : -200. , +.13
Constantes réelles simple précision en point flottant : 1.52E+3
Constantes complexes : (5.E-2, 0.01)
Constantes logiques : .TRUE.
Constantes doubles précision : 12.3D+7 , -1.665D-5
Constantes caractères : '.FALSE.',' F(x)'

4.3. LES VARIABLES :

Le nom d' une variable est formé de 1 à 6 caractères alphanumériques dont le premier est obligatoirement alphabétique .

- exemples : ALPHA FMIN RMAX BETTA MATRIC
Chaque variable possède un type qui définit la nature des informations que nous voulons y placer . Nous distinguons les variables réelles, entières, complexes, logiques et doubles précisions.

Les variables réelles et entières peuvent être déclarées implicitement : si une variable commence par l'une des lettres suivantes I, J, K, L, M, N, elle est du type entier, sinon elle est du type réel .

Pour les autres types de variables, la déclaration explicite est obligatoire.

4.3.1. DECLARATION EXPLICITE : les variables de type double précision, logique et complexe sont déclarées selon la syntaxe suivante :

DOUBLE PRECISION COMPLEX LOGICAL	Var1 [, Var2,.....,VarN]

avec : DOUBLE PRECISION, LOGICAL ET COMPLEX : mots-clés.

Var1, Var2, ..., VarN : variables .

-Exemples :

```
LOGICAL   debut, fin
COMPLEX   imag, compl
DOUBLE PRECISION max
```

Remarques :

Nous pouvons utiliser la même syntaxe pour déclarer une variable de type réel en entier, et inversement .

-Exemples :

```
INTEGER result, solut
REAL masse, long, nul
```

Les variables de type caractères ont une syntaxe légèrement différente:

CHARACTER [*m] Var1[[*m1], Var2[*m2],...., Varp[*mp]]

avec : m, m1, ..., mp : constantes entières positives, représentant la longueur des chaines de caractères des variables.
Var1, Var2 , ..., varp : variables chaines de caractères.

Remarque :

Dans toutes les syntaxes des instructions, les entités se trouvant entre crochets sont facultatives.

-Exemples :

CHARACTER *5 NOTE1, NOTE2, MOYENNE*3, NOTE3 La longueur des variables chaines NOTE1, NOTE2 ET NOTE3 est de 5, mais MOYENNE est de 3.

CHARACTER ADRESSE*30, PROFESSION*15
Les longueurs des variables chaines ADRESSE et PROFESSION sont respectivement de 30 et 15 .

Remarque :

Dans toutes les syntaxes des instructions, les entités se trouvant entre crochets sont facultatives.

4.3.2 LES VARIABLES INDICEES (TABLEAU)

Une variable indicée est un ensemble de variables simples, de même type.Un vecteur, une matrice, un polynôme constituent des variables indicées. Le nombre d'indices tolérés est au maximum de 3.Toutefois certains compilateurs proposent 7.

Une variable indicée est déclarée avant toute utilisation . Si le nom de ces variables commence par l'une des lettres I, J, K, L, M, N, elles sont dites entières, sinon elles sont réelles: déclaration IMPLICITE.
Les autres variables indicées de types doubles précision, complexe,logique et caratère sont déclarées explicitement . La syntaxe de déclaration pour les 3 premiers types est :

DOUBLE PRECISION COMPLEX LOGICAL	Tab1(i1[, i2, i3]) [,.....,Tabp(m1[, m2, m3])]

avec :

DOUBLE PRECISION, LOGICAL et COMPLEX : mots-clés.
Tab1,.....Tabp : variables indicées.
i_i, ..., m_i : constantes entières positives représentant les dimensions des variables indicées.

Remarque :

Nous pouvons utiliser la même syntaxe pour déclarer une variable indicée de type réel en entier, et inversement.
La syntaxe de déclaration d'une variable indicée de type caractère, est identique à celle de déclaration des variables simples, en remplaçant Var1,....., Varp par des variables indicées.

L'OPERATEUR DE CONCATENATION : il a pour rôle la création d'une chaine de caractères nouvelle à partir des caractères des chaines déjà existantes.

Exemples :

1- 'INFOR' // 'MATIQUE' donnera 'INFORMATIQUE'
2- CHARACTER X*2 , Y*4

Si nous donnons à X la valeur 'PR' et à Y la valeur 'ENOM' , alors X//Y donnera 'PRENOM' , par contre Y//X donnera 'ENOMPR'.

4.3.3. LA DIMENSION D'UNE VARIABLE INDICEE

La déclaration de la dimension d'une variable indicée est obligatoire . Elle se fait par la syntaxe suivante :

DIMENSION VI1(DIM1)[,VI2(DIM2),.....,VIp(DIMp)]

avec:
DIMENSION : mot-clé
VI1 , VI2, ...VIP :noms de variables indicées
DIM1, DIM2, ..., DIMP : Liste des dimensions , avec
DIMi ($1 \leq i \leq N$) peut être représenté comme suit :
DIMi =inf1 : sup1 , inf2 : sup2 , ..., infN : supN avec infi et supi respectivement bornes inférieures et supérieures, de la ième dimension ($1 \leq i \leq N$) . Elles sont généralement des constantes entières positives, nulles ou négatives.

Exemples :

DIMENSION V(-2:1), MATR(2:6)

V et MATR sont des vecteurs respectivement de 4 et 5 éléments numérotés de -2 à 1 pour v et de 2 à 6 pour MATR . Le premier est réel, le second est entier .

Remarques :

a- Lorsque INFi =1, il faut se limiter à la représentation de SUPi : DIM i =sup1 , sup2 ,....., supN
Exemple : DIMENSION V(5), MATR (4, 6)

b- La déclaration de variables indicées chaines de caractères se fait dans la déclaration CHARACTER .

Exemple : CHARACTER *2 C, CH (1:8) *3
est équivalent à :
CHARACTER *2 C , CH*3
DIMENSION C(8), CH (1:8)

c- Remise à zéro d'une variable indicée par DATA

DIMENTION V(5), MATR(4, 6)
DATA V, MATR /29*0./

avec 29 = 5+4*6

d- L'instruction DATA (MATR(1, 2), =1, 4) /4*0./ initialise les 4 éléments de la 2ème colonne de MATR à zéro .

4.3.4. L'INSTRUCTION DE DECLARATION IMPLICIT :

Sa syntaxe est :

IMPLICIT Type1(Liste1)[, Type2(Liste2),.....,TypeN(ListeN)]

avec:

Type1, Type2, ...,TypeN : instructions de déclaration ; elles peuvent être INTEGER, REAL, LOGICAL, COMPLEX, CHARACTER, DOUBLE PRECISION.

Liste1, Liste2, ..., ListeN : Liste de variables simples ou indicées.

La première lettre de chaque liste indique que toutes les variables débutant par cette lettre sont du même type, sauf spécification contraire.

Exemple:

IMPLICIT COMPLEX (W-Z)

Toutes les variables débutant par W, X, Y, Z sont de type complexe

5. LES EXPRESSIONS

5.1. LES EXPRESSIONS ARITHMETIQUES

Nous constuisons une expression arithmétique à partir :
- des constantes , des variables simples ou indicées , des fonctions , des opérateurs arithmétiques (+, -, /, *, **) et des parenthèses .

L'évaluation d'une expression arithmétique se fait selon l'ordre établit suivant :

1) Les appels de fonctions .
2) Les parenthèses : nous commençons par les plus internes .
3) Les opérateurs arithmétiques :

   3.1) l'exponnentiation : à priorité égale, nous commençons de droite à gauche.
   3.2) la multiplication et la division .
   3.3) la soustraction et l'addition .

Pour 3.2 et 3.3 , à priorité égale , nous commençons de gauche à droite.

5.2. LES EXPRESSIONS LOGIQUES

Une expression logique est une expression qui peut être vraie ou fausse . C'est une comparaison entre expressions arithmétiques (généralement de même type) à l'aide d'opérateurs de relation (.LE. , .LT., .GE., .GT., .EQ., NE.) et d'opérateurs logiques (.OR., .NOT., .AND., .EQV., .NEQV.).

Remarque :
L'opérateur .NOT. ne s'applique qu'à une seule expression de relation.
L'ordre d'évaluation des expressions logiques est le suivant :

1) Les appels de fonctions .
2) Les parenthèses : nous commençons par les plus internes .
3) Les opérateurs arithmétiques (voir les expressions arithmétiques).
4) L'opérateur de concaténation //.
5) Les opérateurs de relations : selon l'ordre d'apparition .
6) Les opérateurs logiques : selon l'ordre suivant : .NOT., .AND., .OR. puis .EQV. et .NEQV. ont la même priorité.

Exercice-1 : Ecrire les expressions arithmétiques suivantes en FORTRAN :

$R = T/C\,Ln(\alpha_0/\alpha)$ $\qquad$ $C = U/(\omega R_v(E^2-U^2)^{1/2}$

$V = 4\pi R^3/3$ $\qquad$ $G = S(R_1-R_2)/R_3$

Réponse :

```
R= T/C*ALOG ( ALPHAO/ALPHA)
C= U/(W*Rv*(E**2-U**2)**(1/2.))
V= 4./3*PI*R**3  avec PI connu
G=S*(R1-R2)/R3
```

Exercice-2 Ecrire les expressions logiques suivantes en FORTRAN:

$A \geq B$ $\qquad$ $X<Y$ et $Z>W$ $\qquad$ $X \# Y$

$X^2 \geq 0$ ou $Y<0$ $\qquad$ $(X-Y)^2 \leq 1.2$

Réponse:

```
A .GE. B
X .LT. Y .AND. Z .GT. W
X .NE. Y
X**2 .GE. 0 .OR. Y .LT. 0
(X-Y)**2 .LE. 1.2
```

CHAPITRE II .NOTIONS SUR LES ALGORITHMES ET LES ORGANIGRAMMES

Avant d'écrire un programme en langage évolué comme le FORTRAN , il est préférable, et même conseillé de donner une description détaillée de la marche à suivre afin de résoudre un problème posé .

Cette description peut être faite en utilisant deux formes :
- Les algorithmes
- Les organigrammes

1- LES ALGORITHMES

L'algorithme est une manière de formaliser la solution d'un problème posé . Suivant la solution proposée, plusieurs cas peuvent se présenter .

1.INSTRUCTIONS SEQUENTIELLES

L'algorithme est délimité par :
Début : qui indique le début de l'algorithme.
Fin : qui définit sa fin.

Entre ces deux mots est écrite la solution sous forme d'instruction.

Début
enchainement d'instructions
Fin

Si toutes ces instructions sont exécutées séquentiellement , nous parlons d'une structure séquentielle de l'algorithme.

Exemple :
Pour calculer la surface et le périmétre d'un carré , nous avons besoin de connaitre la valeur du coté C.
Soit S la surface à chercher , et P le périmétre .

Voici l'algorithme qui nous résoudrait le problème :

```
Debut
    Lire C
    *calcul de la surface du carré
    S=C*C
    *calcul du périmétre du carré
    P=4*C
    Ecrire S, P
Fin
```

Afin d'aboutir au résultat attendu, nous avons utilisé une suite d'instructions qui doivent être exécutées séquentiellement .
Parmi ces instructions , nous constatons la présence :

- d'une instruction de lecture : instroduite par lire
- d'une instruction d'écriture : introduite par Ecrire ou imprimer .

Ce sont des instructions d'entrée - sortie .
Nous avons une instruction d'affectation qui affecte la valeur d'une expression à une variable : P = 4 * C
Hormis ces instructions, nous avons des commentaires qui sont introduits pour faciliter la compréhension du traitement , nous les avons précédés d'un astérisque.

En FORTRAN , un commentaire est défini en mettant la lettre C en première colonne de la ligne.

2. INSTRUCTIONS CONDITIONNELLES

2.1. PREMIER CAS

Supposons que lors de notre traitement , une ou un ensemble d'instructions ne sont exécutés que si une condition est vérifiée , nous formulerons ce besoin sous la forme :

```
Si condition
   Alors instruction (s)1
Fsi
```

La condition est une expression logique .
En FORTRAN , cela reviendrait à écrire :

```
IF ( condition ) THEN
  intruction ( s )1
ENDIF
```


exemple1 : Donnons une prime de 4% du salaire à une personne ayant plus de 10 ans d'ancienneté. Ecrivons l'algorithme qui calcule la prime de cette personne :
Posons : P : prime de l'employé ; S : salaire ; A : ancienneté ;
Mat : matricule de l'employé

```
Début
    Lire Mat, S, A
    Si A > 10
      Alors *calcul de la prime
           P = S*0.04
    Fsi
    Ecrire Mat, P
Fin
```

2.2. DEUXIEME CAS

Dans le cas ou la condition n'est pas vérifiée, nous pouvons avoir d'autres instructions à exécuter :

```
Si condition
  Alors instruction ( s )1
  Sinon instruction ( s )2
Fsi
```

Le(s) instruction(s)1 sont exécutées lorsque la condition est vérifiée, dans le cas contraire, ce sont le(s) instruction(s)2 qui le seront .

En FORTRAN

```
IF condition THEN
  instruction(s)1
ELSE
  instruction(s)2
ENDIF
```

Exemple2 : Calculons la prime de l'employé en sachant que s'il a moins de 10 ans d'ancienneté , il a 2% du salaire, dans le cas contraire, il a 4% .

```
Début
  Lire Mat, S , A
  Si A < 10
    Alors P = S * 0.02
    Sinon P= S * 0.04
  Fsi
  Imprimer Mat, P
Fin
```

Remarque : nous pouvons avoir la forme générale suivante :

```
IF   condition1   THEN
  instruction(s)1
ELSE
  IF condition2   THEN
    instruction(s)2
    .....
ENDIF
```

3. INSTRUCTIONS ITERATIVES

3.1. ITERATIONS CONTROLEES PAR DES CONDITIONS

L'exécution d'une ou d'un ensemble d'instructions est répétée tant que la condition posée est vérifiée .

```
Tant que condition          ||  Do while  condition
  Faire                     ||    instruction(s)
    instruction(s)          ||  ENDDO
Fait
```

La condition est une expression logique qui prendra pour valeur: vrai ou faux .
Ces instuctions peuvent ne jamais être exécutées si la condition a pour valeur : faux, dès le départ .

De plus, les instructions peuvent être répétées indéfiniment, si parmi elles, il n'en existe pas une qui modifie la condition .

Exemple :

Ecrire un algorithme qui calcule la valeur de S donnée par :

$$S = 1/(N^2 + 1)$$

La condition d'arrêt sera lorsque S est inférieure à E donné .


```
Début
   Lire E, N
   S=0
   Tant que  S ≥ E
     Faire
       S=S+1/(N^2 +1)
     Fait
   Ecrire S
Fin
```

Une autre forme d'itération controlée est la boucle :

```
Répéter
   instruction(s)
Jusqu'à condition
```

Les instructions seront répétées tant que la condition n'est pas vérifiée, d'ailleurs en utilisant le tant que , nous aurons la formulation suivante :

```
Tant que  non-condition
  Faire
    instruction(s)
  Fait
```

Cette manière d'écrire n'a pas d'équivalent en FORTRAN.

3.2. ITERATIONS CONTROLEES PAR DES INDICES

Lorsque nous avons à tester la valeur d'un compteur dans une instruction itérative, nous pouvons utiliser la forme suivante :

```
Pour compteur de VI à VF pas N
    Faire
      instruction(s)
    Fait
```

Dans ce cas , il faut connaitre les valeurs entières de départ VI et d'arrivée VF du compteur ainsi que la valeur entière N du pas.

Si VI est la valeur minimale du comteur, VF doit être maximale et le pas est alors positif .
Si VI est la valeur maximale du compteur , VF doit être minimale et le pas est alors négatif .
Si N ± 1 , le pas est facultatif .

En FORTRAN , cela serait représenté par :

```
        ||  DO étiq  compteur = val1, val2, pas
        ||           instruction(s)
étiq    ||  instruction exécutable
```

Remarque : L'instruction exécutable ne peut pas être un DO, STOP ou IF .

Exemple :Calculer la prime de 50 employés avec la forme : P = S * 0.04 .Posons : P : prime de l'employé ; S : son salaire ; Mat : son matricule.

```
Début
  Pour I de 1 à 50 pas 1
    Faire
      Lire Mat , S
      P = S * 0.04
      Ecrire Mat , P
    Fait
Fin
```

Remarque :

Le Pour peut être remplacé par Tant que :

Exemple:

```
Début
  I=1
  Tant que  I ≤ 50
    Faire
      Lire  Mat, S
      P = S * 0.04
      Ecrire  Mat , P
      I = I+1
    Fait
  Fin
```


II. LES ORGANIGRAMMES

Une seconde description de la solution d'un problème posé peut être donnée en utilisant la notion d'organigrammes .

Un organigramme utilise les schémas suivants :

1 Début et Fin de l'organigramme

2 Lecture et écriture (E/S)

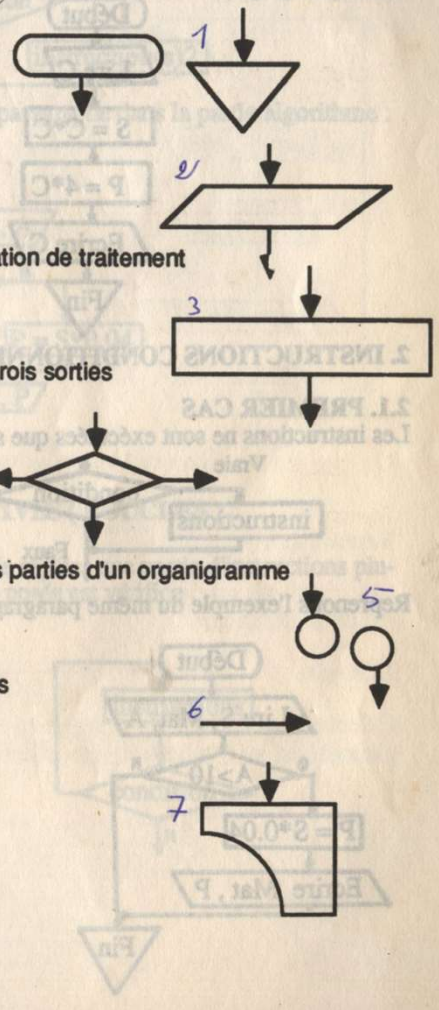
3 Instruction d'affectation et opération de traitement

4 Test à conditions avec deux et trois sorties

5 Noeud pour relier les différentes parties d'un organigramme

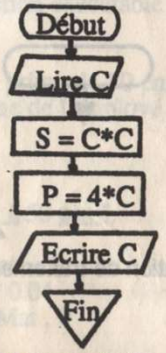
6 Fléche pour relier deux schémas

7 impression sur imprimante



1. INSTRUCTIONS SEQUENTIELLES

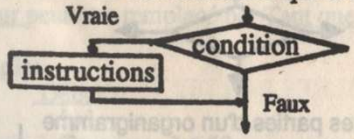
Les instructions sont exécutées dans l'ordre , l'une aprés l'autre .
Reprenons l'exemple donné dans la partie algorithmique , dans le même paragraphe :



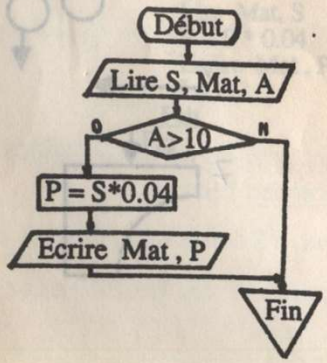
2. INSTRUCTIONS CONDITIONNELLES

2.1. PREMIER CAS

Les instructions ne sont exécutées que si la condition est vérifiée .

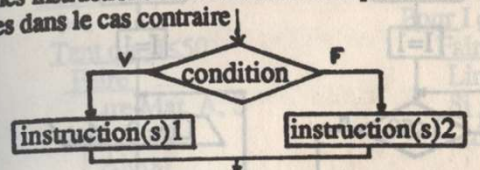


Reprenons l'exemple du même paragraphe dans la partie algorithme :

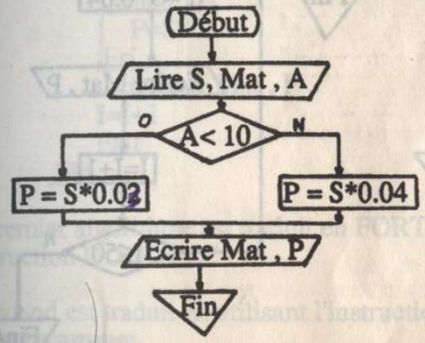


2.2. DEUXIEME CAS

Certaines instructions sont exécutées lorsque la condition est véréfiée, d'autres dans le cas contraire

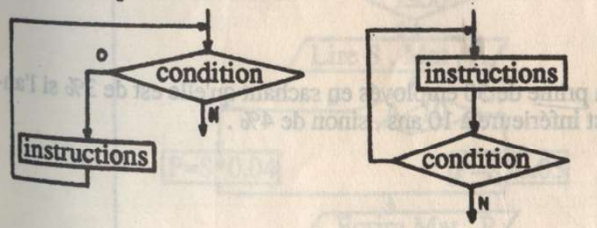


Reprenons l'exemple du même paragraphe dans la partie algorithme :

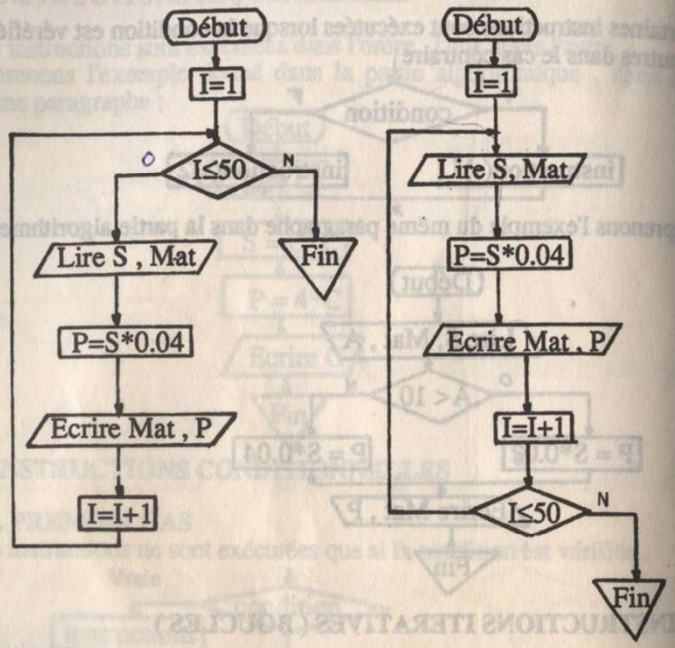


3. INSTRUCTIONS ITERATIVES (BOUCLES)

Quelquefois , il est nécéssaire de répéter une partie d'instructions plusieurs fois tant que la condition posée est vérifiée .
Deux cas se présentent à nous :



Reprenons le calcul de la prime de 50 employés :



Remarque :

Nous pouvons retrouver des instructions conditionnelles dans une instruction itérative .

Exemple :

Calculer la prime de 50 employés en sachant qu'elle est de 3% si l'ancienneté est inférieure à 10 ans , sinon de 4% .

Mat = Matricule
A = ancienneté
P = prime
S = salaire

Algorithmes

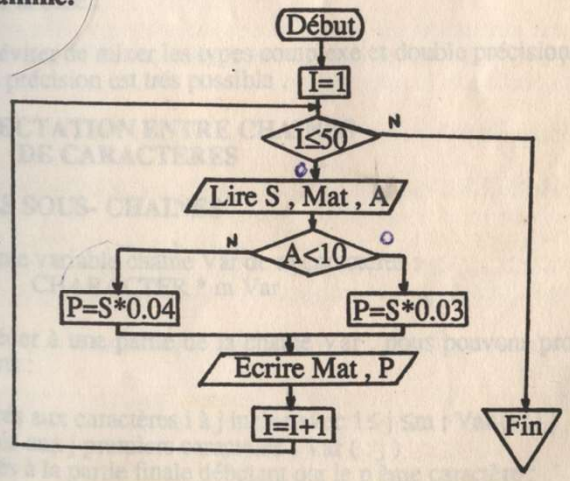
```
Début
  I=1
Tant que I≤50
  Faire
    Lire Mat, A, S
    Si A<10
      Alors
        P=S*0.03
      Sinon
        P=S*0.04
    Fsi
    Imprimer Mat , P
    I=I+1
  Fait
Fin
```

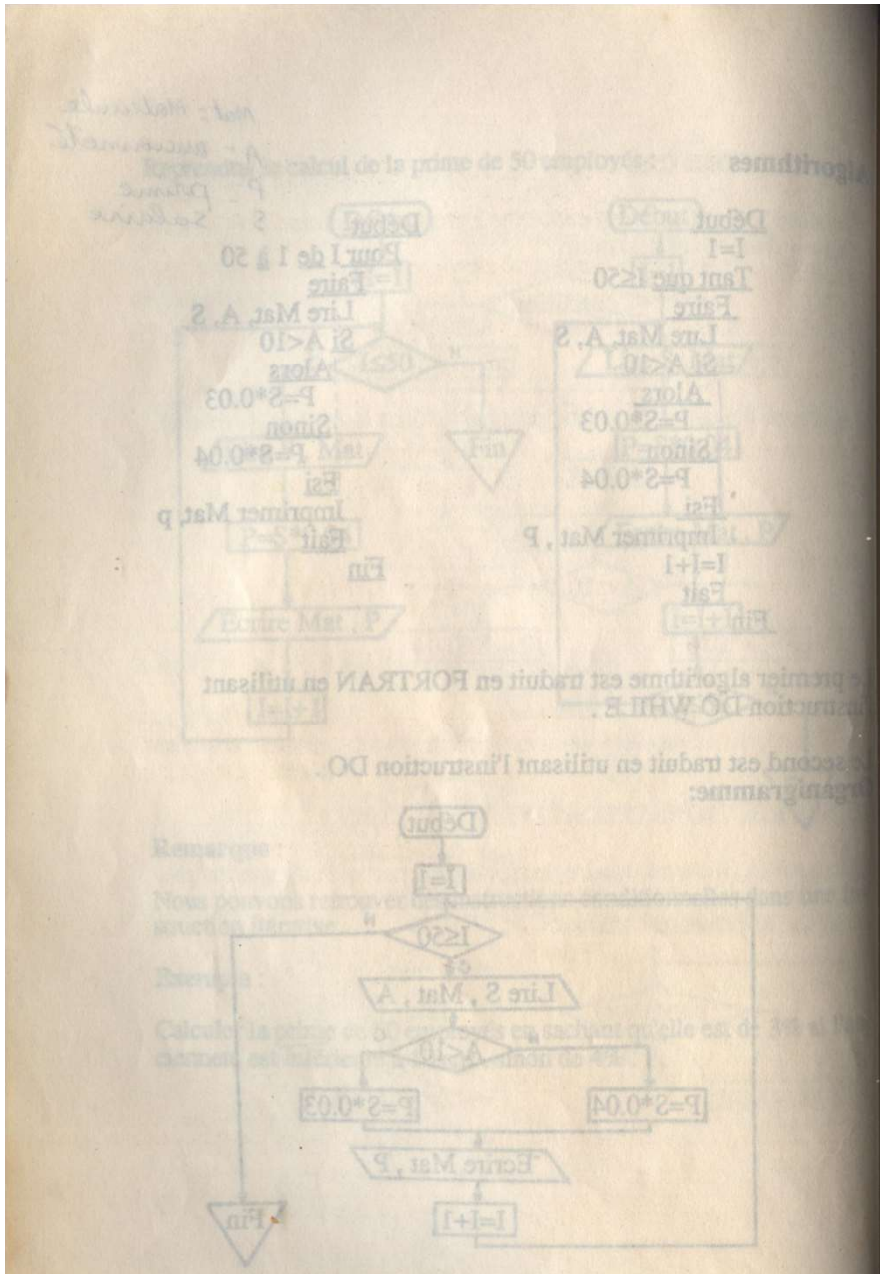
```
Début
  Pour I de 1 à 50
    Faire
      Lire Mat, A, S
      Si A<10
        Alors
          P=S*0.03
        Sinon
          P=S*0.04
      Fsi
      Imprimer Mat, p
    Fait
Fin
```

Le premier algorithme est traduit en FORTRAN en utilisant l'instruction DO WHILE .

Le second est traduit en utilisant l'instruction DO .

Organigramme:





CHAPITRE-III LES INSTRUCTIONS DE FORTRAN 77

1. LES INSTRUCTIONS D'AFFECTATION

1.1. LES INSTRUCTIONS D'AFFECTATION ARITHMETIQUES

La syntaxe est :

Var = expr

avec :

Var : variable numérique simple ou indicée .
expr : expression arithmétique qui est évaluée puis convertie au type de Var .

Remarques :

1- La hiérarchie des types de variables numériques est :

a- complexe
b- double précision
c- réel
d- entier

Pour une opération entre deux opérandes de types différénts , l'opérande de type le plus faible est converti au plus fort , sauf dans le cas de l'exponentiation quand l'exposant est entier, le résultat est toujours du type de la base .

2- Il faut éviter de mixer les types complexe et double précision, un conflit de précision est trés possible .

1.2. AFFECTATION ENTRE CHAINES DE CARACTERES

1.2.1. LES SOUS- CHAINES

Prenons une variable chaine Var de m caractères :

CHARACTER * m Var

Pour accéder à une partie de la chaine Var , nous pouvons procéder de 3 façons :

a- accés aux caractères i à j inclus avec $1 \leq j \leq m$: Var (i : j)
b- accés aux j premiers caractères : Var (: j)
c- accés à la partie finale débutant par le p ème caractère :
Var (p : m)

Exemples :Si la variable BIT vaut 'SCIENCES' , nous aurons :
BIT (4 : 5) vaut 'EN'
BIT (6 : 8) vaut 'CES'
BIT (: 3) vaut 'SCI'

1.2.2. L'AFFECTATION ENTRE SOUS-CHAINES

Sa syntaxe est :

Var = expcar

avec :
Var : variable chaine ou sous-chaine caractère .

Exemple : ALI = 'PROGRAMTION'

avec ALI (8 : 9) = 'MA' , la variable ALI devient **'programmation'**.
Remarques :

a- les caractères de droite en surnombre d'une expression trop longue sont ignorés ;

b- dans le cas ou la chaine expcar est inférieure à la réceptrice , elle est complétée à droite par des blancs .

2-L'INSTRUCTION PARAMETER :

C'est une instruction qui initialise des variables qui seraient utilisées dans des affectations ou dans des déclarations . Ces variables n'existeraient plus aprés la compilation .

Sa syntaxe est :

PARAMETER (Var1=Cst1 [,Var2=Cste2,....,VarN=CsteN)]

avec:
Var1, Var2,.....,VarN : variables . Elles ne devraient pas être déclarées explicitement par les instructions de déclarations .

Cste1, Cste2, ...,CsteN : constantes de types numérique , caractère ou logique .
Exemple:

```
PARAMETER (u=10)
READ(u,*)((a(i,j),i=1,10),=1,5)
```

3.L'INSTRUCTION D'INITIALISATION DES VARIABLES (DATA)

Son rôle est d'alléger le programme en évitant de faire plusieurs affectations.Sa syntaxe est :

DATA VI1, VI2, ...VIN / C1, C2, ...,CN /

avec :
VIi : variables à initialiser .
Ci : constantes .
La variable VI1 est initialisée par C1, VI2 par C2 etc.....
Remarques :
a- Dans le cas ou nous avons plusieurs constantes identiques successives , il est préférable d'utiliser un facteur de répétition .
Exemple : Dans une partie d'un programme FORTRAN , nous rencontrons ceci :

```
I=2
J=2
K=2
L=5
```

Nous utilisons DATA comme suit :
DATA I, J, K, L / 2, 2, 2, 5 /

ou , en utilisant le facteur de répétition :
DATA I, J, K, L, / 3*2, 5 /

b- DATA est la dernière instruction de déclaration à écrire dans un programme .

4. L'INSTRUCTION D'ARRET D'EXECUTION (STOP)

Cette instruction n'est pas obligatoire , néanmoins son utilisation est fort intéréssante dans la mesure ou c'est la dernière instruction exécutable dans un programme FORTRAN .Elles peut être utilisée autant de fois que le programme l'exige .
Sa systaxe générale est :

[étiq] STOP [m]

avec :
étiq : constante entière représentant l'étiquette de STOP .
m : constante entière propre à chaque instruction STOP (dans le cas de plusieurs STOP) .

5. L'INSTRUCTION D'ARRET DE COMPILATION (END)

Sa syntaxe est :

END

Son rôle est d'arrêter la compilation du programme , elle est donc obligatoire . Elle ne doit être suivie d'aucune instruction .

6. LES INSTRUCTIONS DE TEST

Ce sont des instructions d'exécution conditionnelles .

6.1. L'INSTRUCTION IF ARITHMETIQUE

Cette instruction effectue un test sur une expression arithmétique dont le signe renvoie à 3 représentations possibles : positive, négative ou nulle . Elle est dite aussi test à 3 sorties .
Sa syntaxe est :

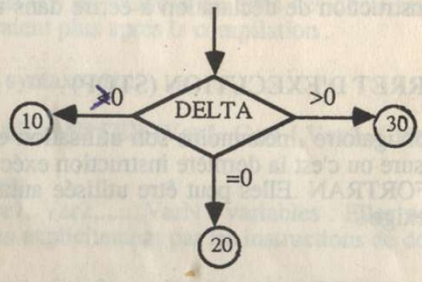
IF (ExpArit) étiq1, étiq2, étiq3

Avec :
ExpArit : expression arithmétique .
étiq1, étiq2, étiq3 : étiquettes numériques correspondant respectivement à ExpArit négative, ExpArit nulle et ExpArit positive.

Exemple :

```
DELTA = B*B - 4. *A*C
IF (DELTA) 10, 20, 30
```

Nous pouvons la représenter comme suit :



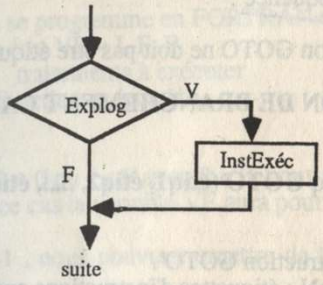
6.2. L'INSTRUCTION IF LOGIQUE

C'est une instruction qui réalise un test logique sur une expression à valeur logique .
Sa syntaxe est :

IF (Explog) instExéc

avec :

Explog : expression logique
InstExéc : instruction exécutable . Elle n'est exécutée que dans le cas ou l'évaluation de l'expression logique est vraie . Elle est décrite par l'organigramme suivant :



Remarque :
L'instruction InstExéc doit être absolument différente des instructions: IF, DO, et END .

6.3. L'INSTRUCTION IF .....THEN.....ELSE / ENDIF

Contrairement aux IF précédents qui conditionnaient une instruction simple , celle-ci conditionne une séquence, c'est à dire une suite d'instructions .
Sa syntaxe est :

```
IF (condition) THEN
   traitement-1
ELSE
   traitement-2
ENDIF
```

Si la condition est vraie , alors seulement le traitement-1 est exécuté , sinon c'est à dire si la condition est fausse , alors seulement le traitement-2 est exécuté .

7. LES INSTRUCTIONS DE BRANCHEMENT

Ces instructions sont utilisées généralement avec les instructions de test précédentes . Elles ont pour rôle d'indiquer le traitement à prendre lors du choix . Pour cela , un numéro qui est une étiquette désigne l'instruction correspondante au traitement .
Il existe deux types d'instructions de branchement :

7.1. L'INSTRUCTION DE BRANCHEMENT INCONDITIONNEL (GOTO)

Sa syntaxe est :

GOTO étiq

avec :
étiq : étiquette de l'instruction ou le branchement doit se faire pour éviter l'exécution en séquence .

Remarque :l'instruction GOTO ne doit pas être étiquettée .

7.2. L'INSTRUCTION DE BRANCHEMENT CALCULEE

Sa syntaxe est :

étiq GOTO (étiq1, étiq2, ..., étiqN)[,] I

avec :
étiq : étiquette de l'instruction GOTO .
étiq1, étiq2,....., étiqN : étiquettes d'instructions exécutables .

I : variable entière positive .
l'exécution de l'instruction GOTO provoque un branchement à étiqK si la valeur de I est K avec I≤ K ≤ N , c'est à dire :

l'étiquette étiq1 si I=1
l'étiquette étiq2 si I=2
.....
l'étiquette étiqN si I=N

8. LES INSTRUCTIONS DE REPETITIONS (LES BOUCLES DO)

8.1. LA BOUCLE DO

Sa syntaxe est :

DO étiq VE = I, F, P

avec :
étiq : l'étiquette d'une instruction exécutable de fin de boucle qui est l'instruction CONTINUE (elle est différente de GOTO , IF, END, STOP et DO).

VE : variable entière simple qui fait le contrôle de la boucle .

I : valeur initial entière positive , négative ou nulle de la variable VE .

F : valeur finale entière positive , négative ou nulle de la variable VE.

P : pas d'incrémention ou de décrémentation de la variable VE (différent de zéro).

Cette instruction se programme en FORTRAN de la façon suivante :

```
      DO étiq VE = I, F, P
            traitements à exécuter
étiq  instruction exécutable
```

Remarques :
1- si I > F avec P > 0 ou I < F avec P < 0, nous ne pouvons accéder à la boucle ; dans ce cas la variable VE aura pour valeur I

2- si P = 1 ou -1 , nous pouvons omettre de le mentionner , il sera pris par défaut .

3- il est possible d'imbriquer les boucles , c'est à dire d'utiliser des boucles à l'intérieur des autres à condition qu'elles soient controlées par des variables différentes .

4- il ne faut jamais modifier la valeur de la variable VE à l'intérieur de la boucle.

5- il est interdit de se brancher directement à l'intérieur d'une boucle DO .

Exemples :

```
      S=0                        Y=1
      DO 100 I=1, 10, 2          DO 50 J=10, 1,-1
100   S=S+V(I)                   Y = Y+ 2*J
100 CONTINUE              50     CONTINUE
```

8.2. LA BOUCLE DO .....WHILE

Cette instruction existe sur beaucoup de compilateurs actuels malgré qu'elle n'a pas été prévue par la norme ANSI .

Sa syntaxe est :

DO [étiq[,]] WHILE explog

avec :

étiq : étiquette de la dernière instruction exécutable de la boucle.
explog : expression logique .

Tant que explog est vraie , un bloc d'instructions compris entre DO WHILE et l'étiquette étiq sera exécuté . ENDDO délimite le traitement répété .

Nous programmons cette instruction de la sorte :

```
DO (étiq(,)) WHILE condition
    bloc d'instructions
ENDDO
```

9. L'INSTRUCTION CONTINUE

Sa syntaxe est :

[étiq] CONTINUE

Cette instruction est surtout utilisée avec la boucle DO . Elle n'a aucun effet , mais elle est exécutable .

10.LES INSTRUCTIONS D'ENTREE / SORTIE

Nous distinguons deux types d'opérations d'E / S en FORTRAN :

a- les E/S en format libre .
b- les E/S controlées par un format.

l'opération d'E/S est caractérisée par :

- le type d'accés (séquentiel ou direct)
- le sens (entrée ou sortie)
- le nom du périphérique ou de l'unité
- la liste des variables à imprimer
- le format des variables

Dans la réalisation d'une E/S , 4 instructions sont utilisées :

PRINT : impression des résultats sur le terminal
READ : lecture des données tapées à partir du clavier vers la mémoire centrale.

WRITE : sortie des données de la mémoire centrale vers un périphérique .
FORMAT : format avec lequel les variables doivent être transmises . Il est utilisé par les instructions précédentes .

1 0.1 L'INSTRUCTION PRINT

a- Sa syntaxe avec format libre :

PRINT *[, liste]

avec :

Liste : liste des donées . Les données peuvent être des variables de n'importe quel type ou des constantes chaines de caractères .

Exemples :

```
PRINT *,'le résultat est ='
PRINT *, I, J, F
PRINT *
```

b- Sa syntaxe avec format imposé :

PRINT étiq, liste

avec :

étiq : l'étiquette d'une instruction FORMAT
liste : liste de résultats

10.2 L' instruction READ

10.2.1. SYNTAXE DE L'INSTRUCTION READ AVEC FORMAT LIBRE

Nous avons la possibilité de l'écrire sous deux formes :

a- PREMIERE SYNTAXE EN FORMAT LIBRE

C'est une lecture à partir du terminal . L'entrée des données se fait par le clavier .

READ *, liste

avec :

liste : liste des variables recevant les valeurs des données, séparées par des virgules .

exemples :

1- READ *,M, epsilon, alpha

2- CHARACTER *17 texte
READ *, texte

Pour ce deuxième exemple , l'entrée au clavier de la valeur (par exemple : VOICI LE RESULTAT :) de la variable TEXTE du type caractère est faite comme suit :

' VOICI LE RESULTAT '

b- DEUXIEME SYNTAXE AVEC FORMAT LIBRE

La lecture se fait à partir de l'unité logique .

READ(u,*[, ERR=étiq1, END=étiq2]) [liste]

avec :
u : numéro d'unité logique , c'est une constante ou variable entière positive .
étiq1 : étiquette d'une instruction exécutable vers laquelle est transféré le contrôle en cas d'erreurs .
étiq2 : étiquette d'une instruction exécutable vers laquelle est transféré le contrôle en cas d'absence d'enregistrements en lecture.
liste : liste de variables simples ou indicées.

Exemples :
Lecture d'une matrice READ(5,*)((A(I, J), I=1, 10), J=1,15)
Lecture d'un vecteur READ(7,*)(X(I), I=1, 100)
Lecture de variables READ(1,*)A, B, C

10.2.2. SYNTAXES DE L'INSTRUCTION READ AVEC FORMAT IMPOSE

Deux formes d'écriture sont possibles :
Les descriptions de cette instruction ont étés faites précédement.

a- PREMIERE SYNTAXE

READ ef, liste

b- DEUXIEME SYNTAXE

READ(u, ef[,ERR=étiq1, END=étiq2]) liste

avec :
ef : étiquette de l'instruction FORMAT .

Exemples :

```
      READ 10, A, B, C, D
10    FORMAT (descripteurs)
```

1-

```
      READ(5, 25)(X(I), I=1, 10)
25    FORMAT(descripteurs)
```

2-

Les descripteurs seront explicités par la suite .

10.3. L'INSTRUCTION WRITE

a-Sa syntaxe en format libre est :

WRITE(u, *[, ERR=étiq]) [liste]

avec :
liste : liste de variables simples ou indicées , de chaines de caractères ou d'expressions .
u , étiq : voir les définitions faites précédement pour l'instruction READ avec format libre .

Exemple :

WRITE(10, *) ' les solutions sont égales à :', I, J, K

b- Sa syntaxe en format imposé est :

WRITE(u, ef [, ERR=étiq])liste

avec :
u : numéro d'unité logique
ef : étiquette de l'instruction FORMAT
liste : liste de variables
Exemple :

```
      WRITE(10, 15)((A(I, J), J=1, 10), I=1,10)
15    FORMAT(descripteurs)
```

10.4. L'INSTRUCTION FORMAT

Cette instruction indique la mise en page pour les états imprimés .
C'est une instruction non exécutable . Elle fait appel à 2 types de descripteurs :

a- descripteurs de traitement des variables dans le cas des conversions, ce sont :

I, E, F, L, D, A

b- descripteurs de mise en page et d'édition du texte , ce sont :
X, ' ', /, H, $

La syntaxe de l'instruction FORMAT est :

étiq FORMAT(descripteurs)

avec :

étiq : étiquette pour se brancher à l'instruction FORMAT
descripteurs : liste de spécification servant à préciser le cadrage et les conversions .

10.5. LES DESCRIPTEURS

10.5.1. LES DESCRIPTEURS DE TRAITEMENT DES VARIABLES

a- le descripteur I

Il est utilisé pour les variables entières.
Sa forme générale est :

[m] I n

avec :

m : répétiteur
n : entier positif indiquant le nombre de caractères à utiliser pour représenter l'entier sur l'enregistrement .

Si le nombre est supérieur aux nombres de positions indiquées par n, il ne sera pas imprimé . Eventuellement , les positions seront remplacées par des astérisques .

Exemples :

```
1-      I=21
        WRITE (3, 10)I
   10   FORMAT(I5)
```

l'édition donnera : 21 précédé de trois blancs


```
2-      INTEGER U, V
        DATA U, V / -21, 125 /
        WRITE(4, 15) U, V
   15   FORMAT(2I6)
```

l'édition donnera : trois blancs suivis de -21 suivis de trois blancs suivis de 125

```
3-      INTEGER X1, X2, X3
        READ(5, 60)X1, X2, X3
   60   FORMAT(I2, I4, I3)
```

Nous faisons entrer par le biais du clavier le nombre : un blanc suivi de 58234651 , l'attribution sera ainsi faite :

X1 = un blanc suivi de 5
X2 = 8234
X3 = 651

b- Le descripteur F

Il est utilisé pour la représentation des nombres de type réel en point fixe simple précision .

Sa forme générale est :

[m]Fn.d

avec :
m : répétiteur
n : représente la longueur totale occupée par le nombre à entrer ou à imprimer .
d : nombre de position qu'occupe la partie fractionnaire (d < n).

Pour ne pas avoir de dépassemnt de capacité , la relation suivante doit être satisfaite :

n = d+p+2

avec :
n et d comme précédement .
p : nombre de caractères de la partie entière .

exemple :

```
   20   READ(*, 20)X, Y
        FORMAT(F10.5, F6.2)
```

Soit à lire : un blanc suivi de 324.5 suivi de quatre blancs suivi de 93.24 .

Nous obtenons :

X = un blanc suivi de 324.5
Y= 93.24

Remarques :

1- Certains nombres réels trés grands ou trés petits (cela dépend de l'ordinateur) ne peuvent pas être représentés par ce descripteur . Ils sont alors représentés par le descripteur E .

2- Lors de l'impression , le signe + est remplacé par un blanc .

c- Le descripteur E

Il est utilisé pour la représentation des nombres de type réel en point flottant simple précision avec exposant en entrée-sortie.

Sa forme générale est :

[m]En.d

avec :
m :répétiteur
n : représente la longueur totale occupée par le nombre à lire ou à imprimer .
d : représente le nombre de chiffres de la partie fractionnaire , avec d < n .

Exemples :

1- représenter 125.13 sous la forme E10.4 : 0.1251E+03

2-
```
    U = -2.25267
    WRITE(3,5)U
5   FORMAT(E11.3)
```

L'édition donnera : un blanc suivi de -0.225E+01

d- Le descripteur L

Il est utilisé pour le traitement des variables logiques .

Sa forme générale est :

[m] L n

avec :
m : répétiteur
n : désigne le nombe de caractères qu'occupe la zone externe. Généralement , n est égal à 1.

A l'édition , à droite de la zone sera édité un T (TRUE) si la variable logique est vraie, sinon il sera édité un F (FALSE) .
Exemple :

```
      LOGICAL I, J
      I= .TRUE.
      J= .FALSE.
      WRITE(*, 10)I, J
10    FORMAT(L2, L3)
```

L'édition donnera : F suivi de deux blancs suivi de T .

e-Le descripteur D

Il est utilisé pour représenter des nombres en double précision .

Sa forme générale est :

[m]Dn.d

avec :
m : répétiteur
n et d : Voir le descripteur E .
La relation suivante doit être satisfaite : n ≥ d+7

Exemple : Donner le format du nombre 1.123456789D+15
D15.9

f- Le descripteur A

Il est utilisé pour les variables de type caractère . Il ne fait aucune conversion. Son intéret est de permettre la lecture et l'édition de chaines de caractères.

Sa forme générale est :

[m]A n

avec :

m : répétiteur

n : longueur de la chaine caractères.

Remarques:

1- En entrée (lecture), si n est inférieur aux nombre de caractères que contient la variable chaine de caractère, elle sera tronquée à gauche . Mais si n est supérieur , les positions de droite seront complétées par des blancs (absence de caractères).

2- En sortie (écriture) , si la zone réservée à l'édition des caractères est plus grande que la chaine , elle sera complétée à gauche par des blancs . Mais, si la zone d'édition est insuffisante, la chaine sera tronquée à droite .

Exemples :

```
1-
        ||WRITE(1, 6) X
      6 ||FORMAT(A6)
```

Si X a pour valeur MATH , il sera édité comme suit : un blanc suivi de MATH. Le premier blanc sera ignoré.

2- Soit à lire 'LA SCIENCE EST NOTRE ISSUE' en utilisant le format suivant :

```
      ||READ(*, 4) X, Y, Z
    4 ||FORMAT(A10, A5, A11)
```

L'attribution sera ainsi faite :

X vaudra : LA SCIENCE
Y vaudra : un blanc suivi de EST suivi d'un blanc
Z vaudra : NOTRE ISSUE

10.5.2. LES DESCRIPTEURS DE MISE EN PAGE ET D'EDITION

a- le descripteur X

Il permet d'insérer n blancs (ou espaces , n dépend de l'ordinateur) consécutifs entre deux zones imprimées d'une même ligne en sortie. Par contre en entrée , il permet d'ignorer le contenu de n caractères consécutifs .
Sa forme générale est :

nX

avec :
n : une constante positive non nulle .

Exemple :

```
      ||WRITE(3, 5) R, P
    4 ||FORMAT(1X, F8.7, 5X, E14.7)
```

L'édition donnera : un blanc qui sera ignoré à l'édition suivi de 8 positions pour la valeur de R , suivi de cinq blancs suivi de 14 positions pour la valeur de P .

b- Le descripteur ''

Il permet l'édition de caractères non interprétés en sortie .
exemple:

```
      ||WRITE(3, 9) R
    9 ||FORMAT(5X,' la valeur de R est : ', F8.7)
```

A l'édition , nous aurons 'la valeur de R est :' suivis de la valeur numérique de R sur 8 positions.

c- Le descripteur / (slach)

Il permet le passage à l'enregistrement suivant . Pour sauter plusieurs enregistrements, il faut répéter le / autant de fois .
Exemple :

```
      ||WRITE(3,51) U, V
   51 ||FORMAT('  U = ', F10.3,5X,/,' V = ',E15.7)
```

l'édition donnera :

U = suivi de la valeur numérique de U 10 positions
V = suivi de la valeur numérique de V 15 positions

d- Le descripteur H (HOLLERITH)

Il permet d'éditer un texte .

Sa forme générale est :

nH

avec :

n : indique le nombre de caractère du texte à éditer aprés le H.

Exemple :

```
      WRITE (1, 5)
5     FORMAT (1X, 8HRESULTAT, /,1H,'10(1H=))
```

L'édition donnera :

RESULTAT
==========

Remarque importante :

Le premier caractère est réservé au positionnement , donc il est recommandé de toujours commencer un FORMAT par le descripteur X. Ceci est trés utile pour ne pas se retrouver avec des résultats dépourvus du premier caractère .

CHAPITRE-IV. LES SOUS- PROGRAMMES ET LES INSTRUCTIONS D'EFFICACITE DE PROGRAMMATION

I. LES SOUS-PROGRAMMES

Deux types de sous programmes sont disponibles dans le langage FORTRAN :

- Les fonctions formules , qui sont des fonctions mathématiques déja définies , c'est à dire reconnues par le compilateur .

- Les sous-programmes , dans lesquels nous retrouvons les fonctions , définies par l'instruction FUNCTION et les subroutines définies par l'instruction SUBROUTINE.

1. LES FONCTIONS FORMULES :

Elles sont également appelées fonctions arithmétiquement définies. Ces fonctions d'un usage facile figurent dans le programme principal .

1.1. SYNTAXE :

La syntaxe générale d'une fonction formule est :

NF(pf1, pf2, ..., pfn) = EAL

avec :

NF : Nom de Fonction , construit selon les règles des variables.

pf1, pf2, ...,pfn : paramètres formels de la fonction ; ce sont des variables non indicées .

EAL : Expression Arithmétique ou logique .

Remarques :

1- L'appel d'une fonction formule provoque , à l'exécution , l'évaluation des paramètres effectifs (noms de variables simples ou indicés , des expressions , etc....) qui doivent correspondre avec les paramètres formels ainsi que l'évaluation de l'expression EAL .

2- Nous pouvons définir le type d'une fonction implicitement et explicitement (déjà vu précédement).

3- Nous devons faire apparaitre la définition de la fonction dans le programme avant toute instuction exécutable.

4- Il doit y avoir correspondance entre paramètres formels et paramètres effectifs : même type , même nombre et l'ordre doit être respecté.

1.2. EXEMPLES

Ecrire les fonctions suivantes sous forme de fonctions formules
a- produit des cubes de deux variables x et y :

PROCUB(X,Y) = X**3*Y**3

b- module de 3 variables :

MODULE(X, Y, Z) = SQRT(X**2+Y**2+Z**2)

2. LES SOUS-PROGRAMMES FUNCTION ET SUBROUTINE

2.1. LA SYNTAXE DE FUCTION

La syntaxe de définition de FUNCTION est :

[déclaration]FUNCTION nomfct(arg1[, arg2, ...argn])

avec :

déclaration : instruction de déclaration explicite de type autre que CHARACTER (voir plus loin) ; s'il est absent, la déclaration est implicite.

nomfct : nom de fonction qui doit être différent de celui des fonctions de la bibliothèque de FORTRAN (voir annexe) . Ce nom doit apparaitre au moins une fois dans un sous-programme FUNCTION , soit à gauche du signe = , soit dans une instruction READ , soit dans une instruction d'appel de sous-programme .

arg1, arg2, ..., argn : arguments; ce sont des noms de variables simples ou indicées, de sous-programmes ou de fonctions. Au moins un argument doit y figurer .

La syntaxe de déclaration des fonctions à valeur de chaines de caractères, diffère un peu de celle donnée :

CHARACTER [*m] FUNCTION nomft (pf1[,.....,])

avec :
m : longueur de la chaine de caractères ,
pf1, ..., : paramètres de la fonction .

Dans l'appel de la fonction de ce type , il faut préciser la longueur exacte de la chaine de caractères .

2.1.1. L'APPEL DE FUNCTION DANS UN PROGRAMME PRINCIPAL

L'appel se fait en écrivant dans n'importe quelle expression mathématique, la syntaxe suivante :

nomfct (par1[, par2 ,....., parn])

avec :
nomfct : nom de fonction .
par1, par2, ...parn : paramètres éffectifs .

2.1.2. LE SOUS-PROGRAMME FUNCTION

Les sous-programmes définis par l'instruction FUNCTION permettent d'écrire des séquences d'instructions différentes de celles du programme principal .
La structure générale est :

[déclaration] FUNCTION nomfct (arg1[, arg2,.....argn])
déclarations
instructions exécutables
END

La dernière instruction exécutable doit être RETURN .

Sa syntaxe est :

[étiq] RETURN

Elle transmet le résultat numérique de la fonction au programme qui lui fait appel ainsi que le retour à ce programme .
L'étiquette est rarement utillisable .

2.1.3. EXEMPLE :

Définir une fonction produit des M (M≥1) premiers nombres entiers .

```
      INTEGER FUNCTION PROD (M)
      PROD = 1
      DO 20 N = 2, M
20    PROD = PROD * N
      RETURN
      END
```

2.2. LA SYNTAXE DE SUBROUTINE :

Sa syntaxe est :

SUBROUTINE nomsub [(arg1, arg2, ..., argn)]

avec :
nomsub : nom de la subroutine , pour lequel s'appliquent les mêmes règles dictées aux variables.
arg1, arg2,....., argn : arguments formels ; ce sont des variables simples ou indicées , de sous-programmes ou de fonctions .

Nous remarquerons que tous les arguments sont facultatifs , contrairement à FUNCTION ou au moins un argument doit figurer .

2.2.1. L'APPEL DE SUBROUTINE DANS UN PROGRAMME PRINCIPAL

L' appel se fait par l'intermédiaire de l'instruction CALL. Cet appel se fait soit dans un programme principal , soit dans un sous-programme.

La syntaxe d'appel est :

CALL nomsub[(par1, par2,....., parn)]

avec :
nomsub : voir précédement .
par1, par2, ..., parn : paramètres effectifs, qui peuvent êtres des constantes , des noms de variables simples ou indicées, des expressions arithmétiques, des noms de sous-programmes.

De même que pour l'instruction SUBROUTINE , les paramètres effectifs peuvent être facultatifs.

2.2.2. LE SOUS-PROGRAMME SUBROUTINE

Les sous-programmes définis par l'instruction SUBROUTINE permettent d'écrire des séquences d'instructions différentes de celles du programme principal .
La structure générale est :

```
SUBROUTINE nomfct[(arg1, arg2, .....arg)]
déclarations
instructions exécutables
END
```

La dernière instruction exécutable est l'instruction RETURN .

2.2.3. EXEMPLE :

Ecrire un sous-programme SUBROUTINE calculant le produit des éléments positifs et la somme des éléments négatifs d'un tableau A(50).

```
      SUBROUTINE TABL (A, PP, SN)
      DIMENSION A(50)
      PP = 1.
      SN = 0.
      DO 10 I = 1, 50
      IF(A(I) .LT. 0.) THEN
        PP = PP * A(I)
      ELSE
        SN = SN + A(I)
      ENDIF
10    CONTINUE
      RETURN
      END
```

2.3. L'INSTRUCTION DE DECLARATION EXTERNAL

C'est une instruction de déclaration non exécutable .

Sa syntaxe est :

EXTERNAL nomef1 [, nomef2, ..., nomefn]

avec :
nomef1, nomef2, ..., nomefn : nom effectifs de sous-programmes ou de fonctions.

Cette instruction doit comporter au moins un nom effectif . Elle doit être écrite au début du programme avant toute instruction exécutable . Elle est nécéssaire quand un sous-programme figure parmi les arguments d'un autre sous-programme .

Pour mieux comprendre son rôle, prenons un exemple :

Soit un sous-programme débutant par : SUBROUTINE DIF (A, B, C, SOM).

Nous supposons que SOM est un sous-programme . Lors de la compilation , le compilateur ne fera pas de distinction entre les variables A, B, C et SOM .

Appellons DIF dans un programe principal : CALL DIF (A, B, C, SOM) .

Lors de l'exécution , SOM aura une adresse mémoire contenant une variable . Pour que le compilateur puisse distinguer SOM d'une variable , nous devons faire la déclaration suivante :

EXTERNAL SOM

II. LES INSTRUCTIONS D'EFFICACITE DE PROGRAMMATION

1. L'INSTRUCTION COMMON :

Cette instruction a pour but de minimiser la perte de temps provoqué par les appels d'arguments dans un sous-programme .

Pour cela , elle crée des zones communes de données et établit une correspondance entre des variables de programmes et sous-programmes différents . Elle se présente sous deux formes : le COMMON blanc et le COMMON étiqueté .

1.1. LE COMMON BLANC

Sa syntaxe est :

COMMON liste

avec :

liste : liste de variables simples ou indicées .

Dans ce cas , les mémoires allouées aux variables sont situées dans une zone commune à partir du début de cete zone . En plus de son rôle d'allocation de mémoire , l'instruction COMMON joue dans ce cas le même rôle que l'instruction DIMENSION puisqu'elle peut déclarer la dimension des variables indicées .

Exemple :

Dans un programme principal , nous rencontrons : COMMON I, J et dans un sous-programme:
```
INTEGER X
COMMON X(3)
```

La correspondance entre ces variables sera :

Au début de la zone commune ., nous aurons : I et X(1) ; dans la zone commune suivante : J et X (2) , dans celles qui suit : X(3) .

1.2. LE COMMON ETIQUETTE

Sa syntaxe est :

COMMON [/[nom1]/]liste1[, /[nom2]/, ..., /[nomN]/listeN]

avec :
nom_1, ...,nomN : noms de zônes de données communes .
liste1,....., listeN : liste de noms de variables simples ou indicées sans paramètres formels .

Cette instruction permet de constituer N zônes communes dans lesquelles peuvent être rangées les variables des listes .

Exemple :
Soit un programme utilisant les variables : I, J, K, L, M, N . Un premier sous-programme utilise les variables J, L, M, N, un second utilise I, J, L .

Les listes auquelles nous faisons appel peuvent être déclarées ainsi :

Dans un programme principal :

```
COMMON J, L /zone1 /M, N /zone2 /I/ zone3 / K
```

Dans le premier sous-programme :

```
COMMON J, L / zone1 / M, N
```

Dans le second sous-programme :

```
COMMON J, L / zone2 / I
```

2. L'INSTRUCTION EQUIVALENCE :

Elle permet de réserver pour un même programme , la même place en mémoire à des variables de noms distincts mais de même types (éventuellement le type peut être distinct dans certains compilateurs).

Nous déduisons du fait qu'une même variable physique peut avoir plusieurs noms distints . Elle permet donc de faire gagner de l'espace mémoire .

Sa syntaxe est :

EQUIVALENCE (liste1) [,(liste2),...(listeN)]

avec:
liste : liste de variable simples ou indicées .Pour ces dernières les indices doivent être des constantes .

exemple:

```
INTEGER VECT(2),TAB(2,2)
EQUIVALENCE (VECT(1),I,J),(TABL(1,2),K)
```

VECT(1), I , J sont des noms différents à la même zone .
TABL(1,2) et K identifient aussi la même zone .

3. L'INSTRUCTION BLOCK DATA

Cette instruction a pour rôle l'initialisation des variables de la zône commune (COMMON) . Elle a l'allure d'un sous-programme .

Sa syntaxe est :

```
BLOCK DATA
instructions de spécification de type
DIMENSION
COMMON
(EQUIVALANCE)
DATA
END
```

Linstruction EQUIVALENCE est facultative .

Exemple :

```
BLOCK DATA
COMMON X, A(2), Y
INTEGER Y
DATA X, A, Y / 2., 3.5, 4., 5 /
END
```

CHPITRE - V . LES FICHIERS

1. NOTION DE FICHIERS

Un fichier est composé d'un enssemble d'enregistrements . Chacun de ces enregistrements est formé d'un ou plusieurs champs.

Il existe deux sortes de fichiers :

- Les fichiers à accés séquentiel
- Les fichiers à accés direct

Un fichier séquentiel est un fichier constitué d'un ensemble d'enregistrements ou d'articles , et où l'accés à ces enregistrements se fait d'une manière séquentielle : pour lire l'enregistrement de rang i, il faut passer par les (i-1) èmes enregistrements précédents .

Un fichier à accés direct a des enregistrements de longueur fixe et comme son nom l'indique , l'accés à un enregistrement donné se fait d'une manière directe : pour lire l'enregistrement i, il n'est pas nécéssaire de passer par ceux qui le précède .

2. DESIGNATION D'UN FICHIER

Un fichier est désigné par un nombre entier naturel , appelé numéro d'unité logique par opposition à l'unité physique , support de fichier .

Un lien est prédéterminé entre ces unités logiques et physiques en utilisant l'instruction OPEN.

3. INSTRUCTION D'OUVERTURE D'UN FICHIER

OPEN (unit [, liste])

avec:

Unit : constante ou variable entière, représentant le numéro d'unité logique du fichier .

Liste : suite d'options du genre ' mot-clé = expression 'séparées par des virgules . L'ordre entre ces options est indifférent .

Les options sont les suivants :

3.1. FILE = nom

Le programmeur donne à son fichier un nom . Ce nom est une chaine de caractères sous forme d'une constante ou d'une variable .

Exemple :

OPEN (5, File = 'fich.dat')

3.2. STATUS = 'expression', avec expression qui peut être :

STATUS = 'NEW'

Indique au système que le fichier n'existe pas, il faut donc le créer .

Exemple :

OPEN (7, STATUS = 'NEW', FILE = 'fichier.res')

STATUS = 'OLD'

Indique au système que le fichier existe déjà .

Exemple :

OPEN (7, STATUS = 'OLD', FILE='fichier.res')

STATUS = 'SCRATCH'

Le système crée un nouveau fichier dit temporaire , et le détruit aprés déconnection . Ce fichier est caractérisé par l'absence de nom.

exemple :

OPEN (1, STATUS = 'SCRATCH')

STATUS = 'UNKNOWN'

Si le fichier existe déjà, le système l'utilise, sinon il le crée .

3.3. ACCES = 'expression' avec expression qui peut être :

ACCES = 'SEQUENTIAL'

C'est une valeur prise par défaut pour considérer le fichier comme étant séquentiel .

ACCES = 'DIRECT'

Précise que le fichier est à accés direct.

3.4. FORM = 'expression' avec expression qui peut être :

FORM = 'UNFORMATTED'

Indique que les enregistrements ne sont pas formatés .

FORM = 'FORMATTED'

Dans ce cas les enregistrements sont formatés .

3.5. RECL = m

Cette option est utilisée en accés direct , et m indique la taille fixe des enregistrements du fichier .

3.6. ERR = étiq

étiq est une étiquette d'une instruction exécutable en cas d'erreurs .

4. INSTRUCTION DE FERMETURE D'UN FICHIER

CLOSE (unit)

Cette instruction permet de rompre le lien entre l'unité logique et le fichier physique associé en provoquant une déconnection de l'unité logique et une fermeture du fichier physique .

S'il s'agit d'un fichier temporaire , il est détruit par :

CLOSE (unit, STATUS = 'DELETE')

5. INSTRUCTION DE REINITIALISATION D'UN FICHIER

REWIND (unit)

Cette instruction provoque un retour en tête du fichier ; donc aprés son exécution , c'est toujours le premier enregistrement du fichier qui est prêt à être traité .

6. INSTRUCTION DE RECUL DANS UN FICHIER

BACKSPACE (unit)

Cette instruction permet de passer à l'enregistrement précédent l'enregistrement courant dans un fichier séquentiel .

7. INSTRUCTION DE FIN DE FICHIER

ENDFILE (unit)

Utilisée dans un fichier séquentiel , cette instruction , écrit un enregistrement 'fin de fichier' .

8. ENTREES-SORTIES DANS UN FICHIER A ACCES DIRECT

8.1. ECRITURE DANS UN FICHIER A ACCES DIRECT

La syntaxe d'écriture dans un fichier à accés direct est :

READ (unit , étf, REC = m[, ERR = étiq]) listVar

avec :

unit : unité logique

étf : étiquette d'un FORMAT ou * pour un format libre

LisVar : liste de variables

8.2. LECTURE DANS FICHIER A ACCES DIRECT

La syntaxe de lecture dans un fichier à accés direct est :

WRITE (unit, étf, REC = m[, ERR=étiq])listDon

avec :

LisDon : liste de données

9. ENTREES-SORTIE DANS UN FICHIER BINAIRE SEQUENTIEL

La syntaxe d'écriture est :

WRITE (unit) LisDon

Cet ordre d'écriture n'ajoute qu'un seul enregistrement au fichier dont la longueur est déterminée par LisDon .

La syntaxe de lecture est :

READ (unit [, END = étiq]) LisVar

Cet ordre de lecture lit un seul enregistrement . La longueur de LisVar ne doit pas être supérieure à celle de l'enregistrement à lire.

DEUXIEME PARTIE

EXERCICES

CHAPITRE-1. EXERCICES DIVERS

EXERCICE 1 .CALCUL DE Y EN FONCTION DE A ET B

Etant donné deux valeurs a et b positives, écrire un programme qui calcule les valeurs de :

$Y = a.b^2$ si $a > b$
$Y = 0$ si $a = b$
$Y = a.b$ si $a < b$

SOLUTION :

Nous résolvons ce problème en utilisant deux méthodes . La première utilise le test à deux sorties, la seconde le test à trois sorties .

PREMIERE SOLUTION :

Le test se fait en comparant les deux variables A et B. La réponse au test logique , en comparant A et B est le résultat du oui ou non : test à deux sorties .

ALGORITHME-1

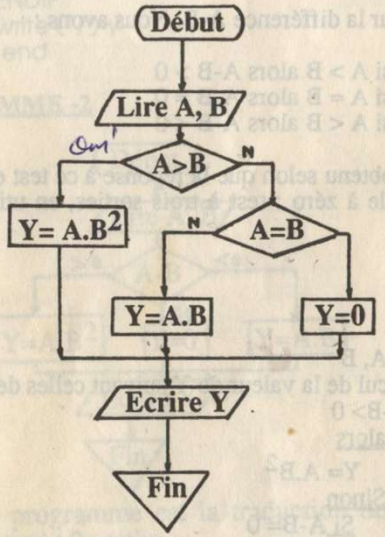
```
Début
  * Lecture des valeurs A et B positives .
  Lire
  * Calcul de la valeur de Y suivant celles de A et B
  Si A>B
    alors
      Y=A.B2
    Sinon
      Si A=B
        Alors
          Y=0
        Sinon
          Y= A.B
      Fsi
  Fsi
  Ecrire Y
Fin
```

Ce premier programme est la traduction de l'algorithme .

PROGRAMME -1

```
      read (*,*)a,b
      if (a.gt.b) then
        y = a*b**2
      else
        if (a.eq.b) then
          y=0
        else
          y=a*b
        endif
      endif
      write (*, 3)y
    3 format (1x,'y = ', f10.4)
      stop
      end
```

Organigramme-1



Ce deuxième programme est la traduction de l'organigramme .

PROGRAMME -2

```
      read (*,*) a,b
      if (a.gt.b) goto 10
      if (a.eq.b) goto 20
      y =a*b
      goto 30
 20   y=0
      goto 30
 10   y=a*b**2
 30   write(*, 3)y
  3   format (1x,'y = ', f10.4)
      stop
      end
```

DEUXIEME SOLUTION :

Le test se fait sur la différence A-B. Nous avons :

si A > B alors A-B > 0
si A = B alors A-B = 0
si A < B alors A-B < 0

Le résultat est obtenu selon que la réponse à ce test est supérieure, inférieure ou égale à zéro : test à trois sorties, en utilisant les organigrammes .

Algorithme-2

```
Début
  Lire A, B
  *Calcul de la valeur de Y suivant celles de A-B.
  Si A-B> 0
     alors
       Y= A.B2
     Sinon
       Si A-B= 0
          Alors
            Y= 0
          Sinon
            Y= A.B
```



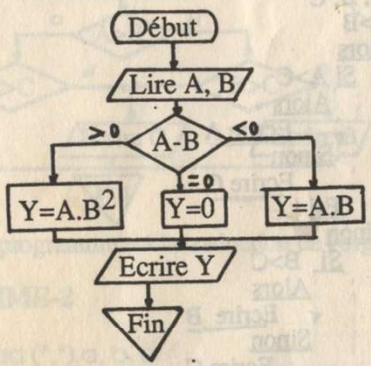
```
       Fsi
  Fsi
  *Affichage de la valeur de Y calculée .
  Ecrire Y
Fin
```

Ce troisième programme est la traduction de l'algorithme .

PROGRAMME -3

```
      read (*,*) a, b
      IF (a-b .GT. 0) THEN
         y = a*b**2
      ELSE
         IF (a-b .EQ. 0) THEN
            y = 0
         ELSE
            y = a*b
      ENDIF
      write (*,*) y
      end
```

ORGANIGRAMME -2



Ce quatrième programme est la traduction de l'organigramme qui répond au test à 3 sorties .

PROGRAMME -4

```
      read (*, *) a, b
      if (a-b) 10, 20, 30
30    y=a*b**2
      goto 5
20    y=0
      goto 5
10    y=a*b
 5    write (*,*) y
      end
```

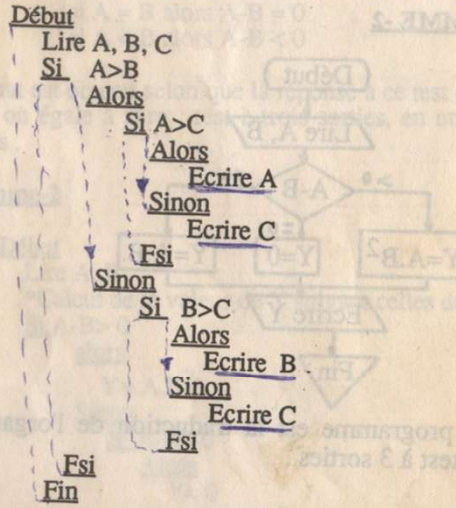
EXERCICE 2 RECHERCHE DE LA PLUS GRANDE VALEUR

Faire un programme qui cherche la plus grande valeur entre A, B et C .

SOLUTION :

Nous utilisons un test logique à deux sorties pour comparer chacune des valeurs A, B et C.

Algorithme

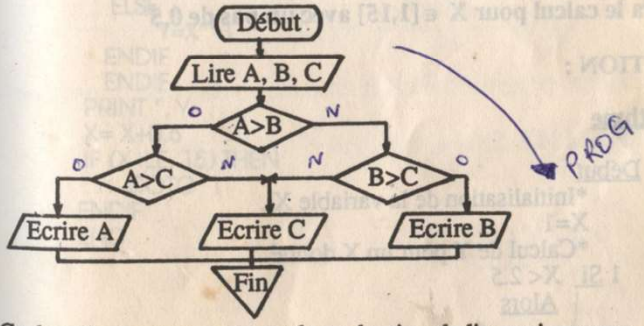


Ce premier programme est la traduction de l'algorithme:

PROGRAMME-1

```
read (*,*) a, b, c
  if (a.gt.b) then
    if (a.gt.c) then
     write (*,*) a
  else
      write (*,*) c
  endif
     else
  if (b.gt.c) then
      write (*,*) b
  else
      write (*,*) c
  endif
endif
end
```

Organigramme



Ce deuxième programme est la traduction de l'organigramme

PROGRAMME-2

```
      read (*,*) a, b, c
      if (a.gt.b) goto 10
      if (b.gt.c) goto 15
25    write (*,*) c
      stop
```

```
15  write (*,*) b
    stop
10  if (a.gt.c) goto 20
    goto 25
20  write (*,*) a
    stop
    end
```

EXERCICE 3. CALCUL DE Y EN FONCTION DES VALEURS DE X.

X étant une variable réelle, écrire un programme qui calcule les valeurs de Y donnée par :

$Y=X^4$ si $X < 2.5$

$Y=X^3$ si $2.5 \leq X \leq 9.5$

$Y = X^2$ si $X > 9.5$

On fera le calcul pour $X \in [1,15]$ avec un pas de 0.5

SOLUTION :

Algorithme

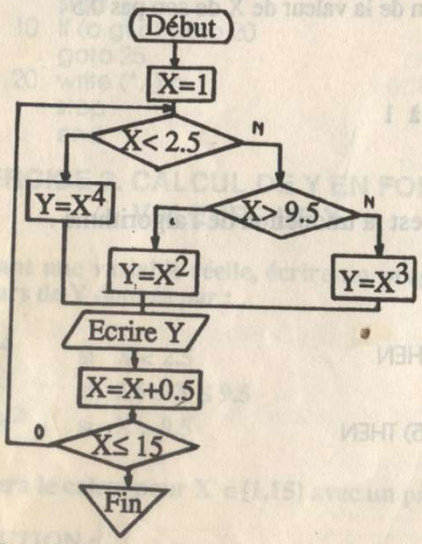
```
Début
    *Initialisation de la variable X .
    X=1
    *Calcul de Y pour un X donné .
1 Si  X< 2.5
      Alors
        Y=X^4
      Sinon
        Si  X> 9.5
          Alors
            Y=X^2
          Sinon
            Y=X^3
        Fsi
    Fsi
    *Affichage de la valeur de Y .
    Ecrire Y
    *Augmentation de la valeur de X de son pas 0.5 .
    X=X+0.5
    Si  X≤ 15
        Alors
            Aller  à  1
    Fsi
Fin
```


Ce premier programme est la traduction de l'algorithme .

PROGRAMME-1

```
      X=1
1     IF (X .LT. 2.5)  THEN
          Y= X**4
      ELSE
        IF (X .GT. 9.5) THEN
            Y= X**2
        ELSE
            Y=X**3
        ENDIF
      ENDIF
      PRINT *, Y
      X= X+0.5
      IF (X .LE. 15) THEN
          GOTO  1
      ENDIF
      END
```

Organigramme



Ce deuxième programme est la traduction de l'organigramme .

PROGRAMME-2

```
      x=1
5     if (x.lt.2.5) goto 10
      if (x.gt.9.5) goto 20
      y=x**3
6     write(*,*) y
      x=x+0.5
      if (x.le.15) goto 5
      stop
20    y=x**2
      goto 6
10    y=x**4
      goto 6
      end
```


EXERCICE 4. RESOLUTION D'UNE EQUATION DU SECOND DEGRE

Les coefficients A, B et C étant connus , résoudre l'équation du second degré dans C : $AX^2 + BX + C = 0$

SOLUTION :

Pour Delta = $B^2 - 4\,A\cdot C$, si delta > 0 , les solutions dans R sont :

$$X_1 = \frac{-B + \sqrt{delta}}{2A} \qquad X_2 = \frac{-b-\sqrt{delta}}{2A}$$

Si delta = 0 , la solution est double dans R : $X_1 = X_2 = \frac{-B}{2A}$

Si delta < 0 ,les solutions dans C sont :

$$X_{c1} = \frac{-B + i\sqrt{delta}}{2A} \qquad X_{c2} = \frac{-b- i\sqrt{delta}}{2A}$$

Algorithme

```
Début
  *Lecture des coefficients de l'équation .
  Lire A, B, C
  Si A= 0
     Alors
  Si B= 0
     Alors
        Si C=0
           Alors
              *Cas ou A= 0, B=0 et C=0 .
              Ecrire "Indéterminé"
           Sinon
              *Cas ou A= 0, B= 0 et C#0 .
              Ecrire "Impossible"
        Fsi
     Sinon
        * Cas ou A= 0, B#0 et C quelconque .
        X= -C/B
        Ecrire X
  Fsi
  Sinon .
```

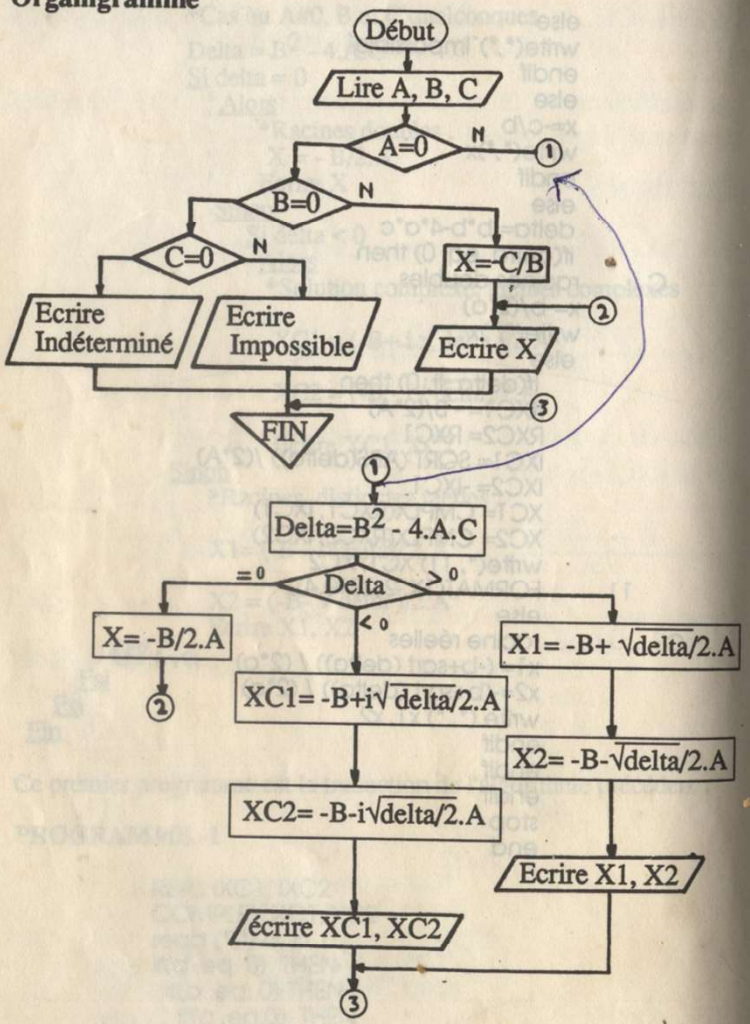
```
*Cas ou A#0, B et C quelconques .
Delta = B2 - 4.A.C
Si delta = 0
  Alors
    *Racines doubles .
     X = - B/2.A
    Ecrire X
  Sinon
    Si delta < 0
      Alors
       *Solution complexe, racines complexes

       XC1 = (-B+ i √delta )/2.A

       XC2 = (-B - i√ delta)/2.A

       Ecrire XC1, XC2
    Sinon
       *Racines  distinctes réelles .

       X1= (-B + √delta)/ 2.A

       X2 = (-B- √ delta )/2.A
       Ecrire X1, X2
    Fsi
  Fsi
Fsi
Fin
```

Ce premier programme est la traduction de l'algorithme précédent .

PROGRAMME -1

```
      REAL IXC1, IXC2
      COMPLEX XC1, XC2
      read (*,*) a, b, c
      if(a .eq. 0) THEN
       if(b .eq. 0) THEN
        if(c .eq.0) THEN
         write(*,*)´ indeterminé´
      else
      write(*,*)´impossible´
      endif
      else
      x=-c/b
      write(*,*)x
      endif
      else
      delta= b*b-4*a*c
      if(delta .eq. 0) then
C     racines doubles
      x=-b/(2*a)
      write(*,*)x
      else
        if(delta .lt. 0) then
        RXC1= - B/(2*A)
        RXC2= RXC1
        IXC1= SQRT (ABS(delta)) /(2*A)
        IXC2= -IXC1
        XC1= CMPLX(RXC1, IXC1)
        XC2= CMPLX(RXC2, IXC2)
        write(*, 11) XC1, XC2
 11     FORMAT(1X, 2(2E10.4))
        else
C       racine réelles
        x1= (-b+sqrt (delta)) / (2*a)
        x2=-(b+sqrt (delta)) / (2*a)
        write (*, *) x1, x2
        endif
        endif
        endif
        stop
        end
```

Organigramme



Ce deuxième programme est la traduction de l'organigramme.

PROGRAMME-2

```
      read(*,*)a, b, c
      if(a.eq.0) goto 1
      delta= b*b-4*a*c
      if(delta)2, 3, 4
C racines doubles
   3  x=-b/(2*a)
   6  write(*, *)x
      stop
C racines complexes
   2  RXC1 = -B/(2*A)
      RXC2 = RXC1
      IXC1 = SQRT (ABS (delta)) / (2*A)
      IXC2 = - IXC1
      XC1 = CMPLX (RXC1, IXC1)
      XC2 = CMPLX (RXC2, IXC2)
      write (*, 11) XC1, XC2
  11  FORMAT (1X, 2 (2E10.4))
      stop
C racines réelles
   4  x1= (-b+sqrt(delta)) / (2*a)
      x2=-(b+sqrt(delta)) / (2*a)
      write (*, *)x1, x2
      stop
   1  if (b.eq.0) goto 7
      x= -c/b
      goto 6
   7  if (c.eq.0) goto 8
      write (*, *) ' impossible '
      stop
   8  write (*, *) ' indeterminé '
      stop
      end
```

EXERCICE 5.CALCUL D'UNE SERIE DE TERME $\frac{1}{N}$

Ecrire un programme qui calcul et imprime la valeur de S donnée par :

$$S = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{20}$$

SOLUTION :

L'expression générale est :

$$S = \sum_{N=1}^{20} \frac{1}{N}$$

Deux algorithmes sont proposés .

Dans le premier , nous utilisons l'instruction de branchement (aller à), dans le second la boucle (pour) .

Algorithme-1

```
Début
   S=0
   N=1
2  S=S+1/N
   N=N+1
   Si N≤ 20
     Alors
       Aller à 2
     Sinon
       Ecrire S
   Fsi
Fin
```


PROGRAMME-1

```
      DATA S, N / 0, 1/
   2  s=s+1. /n
      n=n+1
      if(n.le.20) then
        goto 2
      else
        print *, s
      endif
      stop
      end
```

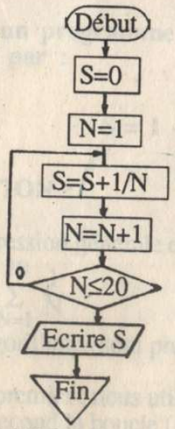
Algorithme-2

```
Début
   S=0
   *Incrémentation
   Pour N=1 à 20 pas 1
      Faire
         S=S+ 1/N

      Fait
   Ecrire S
Fin
```

PROGRAMME-2

```
      s=0
      do 10 n=1, 20
      s=s+1. / n
  10  continue
      write(*, 5) s
   5  format (3x, 'valeur de S', f 7.5)
      stop
      end
```

Organigramme



PROGRAMME-3

```
      s=0.
      n=1
5     s=s+1./n
      n=n+1
      if(n.le.20) goto 5
      write(*, *)s
      stop
      end
```

L'exécution des programmes précédents donne : S = 3.5977400

EXERCICE 6. CALCUL DE LA VALEUR APPROCHEE DE π

Ecrire un programme qui calcule et imprime la valeur de π au moyen de la série suivante (100 termes) :

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots.$$

SOLUTION :

L'expression générale est :

$$S = \frac{\pi}{4} = \sum_{N=0}^{99} \frac{(-1)^N}{2N+1} \quad \text{Ou} \quad S = \frac{\pi}{4} = 1 + \sum_{N=1}^{99} \frac{(-1)^N}{2N+1}$$

Nous vous proposons deux algorithmes et un organigramme .

Algorithme-1

```
Début
   S=0
   N=0
   *K est une variable utilisée pour le signe .
   K=1
1  S=S+K/(2. N+1)
   K= -K
   N=N+1
   Si N≤ 99
      Alors
         * Le calcul n'est pas terminé.
         Aller à 1
      Sinon
         PI = 4 . S
         Ecrire PI
   Fsi
Fin
```

PROGRAMME-1

```
      DATA s, n, k / 2*0, 1 /
 1    s=s+float (k) / (2*n+1)
      k=-k
      n=n+1
      if (n.le.99)  then
        goto  1
      else
        pi=4*s
        write (*, 4) pi
 4    format (1x,´pi = ´, f8.6)
      endif
      end
```

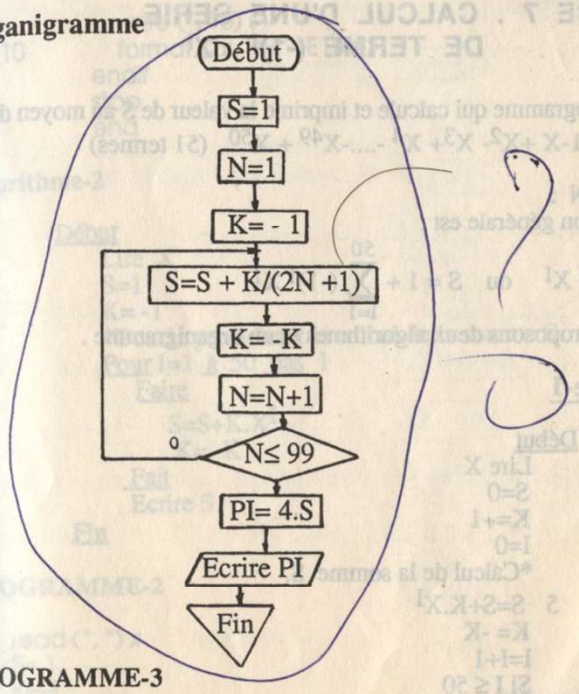
Algorithme-2

```
Début
    S=0
    K=1
    Pour N=0 à 99 pas 1
        Faire
          S= S+K/(2. N+1)
          K= -K
        Fait
        PI = 4.S
        Ecrire PI
Fin
```

PROGRAMME-2

```
      data  s, k / 0, 1 /
      do  15  n=0,99
      s=s+float (k) / (2*n+1 .)
      k=-k
15    continue
      pi= 4*s
      write (*, *) pi
      stop
      end
```


Organigramme



PROGRAMME-3

```
      DATA  s, n, k / 2*1, -1 /
10    s=s+float (k) / (2*n+1)
      k= -k
      n = n+1
      if (n. le. 99)  goto  10
      pi = 4*s
      write (*, 1) pi
 1    format (1x,´pi = ´, f8.6)
      stop
      end
```

L'exécution de ces programmes donne PI = 3.1315930. Nous remarquons que la valeur exacte de π n'est pas atteinte. Pour l'obtenir il faut augmenter la valeur de N .

EXERCICE 7 . CALCUL D'UNE SERIE DE TERME $(-1)^I . X^I$

Ecrire un programme qui calcule et imprime la valeur de S au moyen de la série

$S = 1-X+X^2-X^3+X^4-....-X^{49}+X^{50}$ (51 termes)

SOLUTION :

L'expression générale est :

$$S = \sum_{I=0}^{50} (-1)^I X^I \quad \text{ou} \quad S = 1 + \sum_{I=1}^{50} (-1)^I X^I$$

Nous vous proposons deux algorithmes et un organigramme .

Algorithme-1

```
Début
   Lire X
   S=0
   K=+1
   I=0
   *Calcul de la somme S.
5  S=S+K.X^I
   K= -K
   I=I+1
   Si I ≤ 50
      Alors
         Aller à 5
      Sinon
         *Impression du résultat de la somme.
         Ecrire S
      Fsi
Fin
```

PROGRAMME-1

```
      read (*, *)x
      DATA s,k,i/0,1,0/
5     s=s+k*x**i
      k=-k
      i=i+1
      if (i .le. 50) then
      goto 5
      else
      write (*, 10) s
10    format (2x, 's= ', E13.6)
      endif
      stop
      end
```

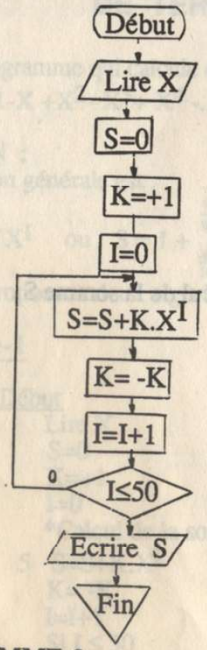
Algorithme-2

```
Début
   Lire X
   S=1
   K= -1
   *Boucle permettant le calcul de la somme S .
   Pour I=1 à 50 pas 1
      Faire
         S=S+K.X^I
         K= -K
   Fait
   Ecrire S
Fin
```

PROGRAMME-2

```
read (*, *) x
S= 1.
K=-1
DO 10 i= 1 , 50
S= S+ K* x ** i
K= -K
10 Continue
Write (*,*) S
STOP
END
```

Organigramme



PROGRAMME-3

```
      read (*, *)x
      s=0
      k=1
      i=0
15    s=s+k*x**i
      k=-k
      i=i+1
      if (i.le.50)  goto 15
      write(*, 10)s
10    format (2x,'s= ',E13.6)
      end
```


EXERCICE 8. CALCUL D'UNE SERIE DE TERME $\frac{N.(-1)^N}{X^N + N}$

Ecrire un programme qui permet de calculer et d'imprimer la valeur de Y en prenant 60 termes de la suite :

$$Y = 1 - \frac{1}{X + 1} + \frac{2}{X^2 + 2} - \frac{3}{X^3 + 3} + \ldots$$

SOLUTION :

L'expression générale est :

$$Y = 1 + \sum_{N=1}^{59} \frac{N.(-1)^N}{X^N + N}$$

Nous vous proposons deux algorithmes et un organigramme .

Algorithme-1

```
Début
   Lire X
   *Y vaut 1 quand N=0.
   Y=1
   N=1
   K= -1
   *Calcul de Y.
1  Y=Y+K.N /( X^N +N)
   K= -K
   N=N+1
   Si N≤ 59
      Alors
         Aller à 1
      Sinon
         Ecrire Y
   Fsi
Fin
```

PROGRAMME-1

```
      read (*, *)x
      DATA y, n, k / 2*1, -1 /
1     y= y+k*n / (x**n+n)
```

```
      k=-k
      n=n+1
      if (n .le. 59) then
         goto 1
      else
         write (*, 2)y
2     format (1x, 17hle résultat est :, f8.6)
      stop
      end
```

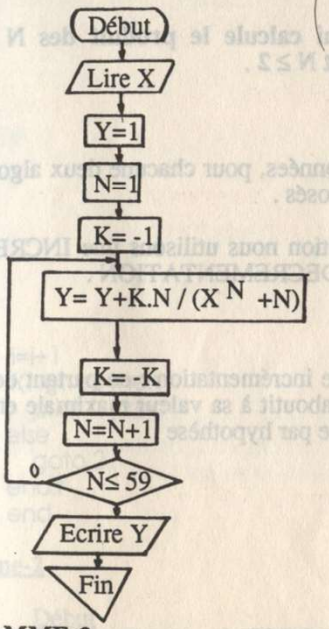
Algorithme-2

```
Début
   Lire X
   Y= 1
   K= -1
  pour N=1 à 59 pas 1
      Faire
        Y = Y+ K.N /(X^N +N)
        K= -K
      Fait
   Ecrire Y
  Fin
```

PROGRAMME-2

```
      read (*, *)x
      y=1
      k=-1
      do 30 n=1, 59
      y=y+k*n / (x**n+n)
30    k= -k
      write(*, *)y
      stop
      end
```


Organigramme



PROGRAMME-3

```
      read (*, *)x
      DATA y, n, k /1, 1, -1 /
20    y=y+k*n / (x**n+n)
      k= -k
      n= n+1
      if (n.le.59) goto 20
      write (*, 1) y
1     format (1x, 17hle résultat est :, f8.6)
      stop
      end
```

L'exécution de ces programmes pour X=4.3872 nous donne :

Y= 8.824736E- 001 .

EXERCICE 9 PRODUIT DE NOMBRES ENTIERS

Ecrire un programme qui calcule le produit des N premiers nombres entiers en prenant N ≥ 2 .

SOLUTION :

Deux solutions sont données, pour chacune deux algorithmes et un organigramme sont proposés .

Dans la première solution nous utilisons une INCREMENTATION, dans la seconde une DECREMENTATION .

PREMIERE SOLUTION :

Cette solution utilise une incrémentation : en partant de la valeur minimale du compteur , on aboutit à sa valeur maximale en rajoutant un pas dont la valeur est fixée par hypothèse .

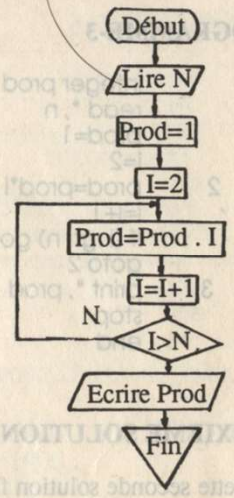
Algorithme-1

```
Début
  Lire N
  Prod=1
  I=2
  *Calcul du produit PROD des premiers nbres entiers
2 Prod = Prod . I
  *Incrémentation de I.
  I=I+1
  Si I>N
     Alors
        Ecrire Prod
     Sinon
        Aller à 2
  Fsi
Fin
```

PROGRAMME-1

```
      integer prod
      read *, n
      prod=1
      i=2
2     prod=prod*i
      i=i+1
      if(i .gt. n) then
        print *, prod
      else
        goto 2
      endif
      end
```


Organigramme-1



Algorithme-2

```
Début
  Lire N
  Prod=1
  *Boucle calculant le produit des N nbres entiers
  Pour I=2 à N pas 1 *écriture d'incrémentation
     Faire
        Prod = prod . I
     Fait
     Ecrire Prod
Fin
```

PROGRAMME-2

```
      integer prod
      read (*, *)n
      prod=1
      do 10 i= 2, n
      prod=prod*i
10    continue
      write (*, *) prod
      end
```

PROGRAMME-3

```
      integer prod
      read *, n
      prod=1
      i=2
 2    prod=prod*i
      i=i+1
      if (i .gt. n) goto 3
      goto 2
 3    print *, prod
      stop
      end
```

DEUXIEME SOLUTION :

Cette seconde solution fait appel à une décrémentation : à partir de la valeur maximale attribuée au compteur, on aboutit à sa valeur minimale en retranchant à chaque fois la valeur fixée d'un pas .

Algorithme-3

```
Début
  Lire N
  Prod=1
1 Prod = Prod . N
  *Décrémentation.
  N=N-1
  Si N≥2
    Alors
      Aller à 1
    Sinon
      Ecrire Prod
  Fsi
Fin
```

PROGRAMME-4

```
      intger prod
      read (*, *)n
      prod=1
15    prod=prod*n
      n=n-1
      if (n .ge. 2) then
        goto 1
      else
        write (*, 2) prod
 2    format (3x, 'prod=', i9)
      endif
      stop
      end
```

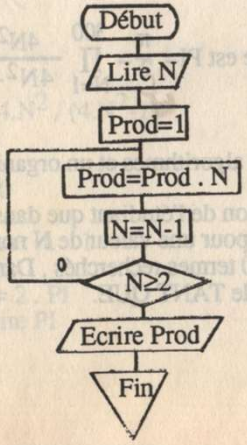

Algorithme-4

```
Début
  Lire N
  Prod=1
  Pour I de N à 2 pas -1
    Faire
      Prod = Prod . I
    Fait
  Ecrire Prod
Fin
```

PROGRAMME-5

```
      integer prod
      read (*, *)n
      prod=1
      do 15 i=n, 2, -1
      prod=prod*n
15    continue
      write(*, 2)prod
 2    format (3x, 'prod=', i9)
      stop
      end
```

Organigramme-2



PROGRAMME-6

```
      integer prod
      read (*, *)n
      prod=1
15    prod=prod*n
      n=n-1
      if (n .ge. 2) goto 15
      write (*, 2) prod
2     format (3x,'prod= ', i9)
      stop
      end
```

L'exécution pour N=12 donne PROD = 479001600

EXERCICE 10 CALCUL DE LA VALEUR APPROCHEE DE π

Ecrire un programme qui calcule et imprime la valeur approchée de π au moyen du produit :

$$\frac{\pi}{2} = \frac{2}{1}\frac{2}{3}\frac{4}{3}\frac{4}{5}\frac{6}{5}\frac{6}{7} \quad \ldots\ldots$$

On prend 600 puis 1200 termes .

SOLUTION :

L'expression générale est $PI = \frac{\pi}{2} = \prod_{N=1}^{300} \frac{4N^2}{4N^2 - 1}$

Nous proposons trois algorithmes et un organigramme .

Nous attirons l'attention de l'étudiant que dans l'expression générale, N varie de 1 à 300 , car pour une valeur de N nous obtenons deux termes, ce qui donnera les 600 termes recherchés . Dans le premier algorithme . nous utilisons la boucle TANT QUE.

Algorithme-1

```
Début
   PI=1
   N=1
   *Boucle calculant les 600 termes donnant π/2
   Tant que  N≤ 300
      Faire
      PI = PI . 4.N^2  / (4.N^2-1)

      N=N+1
      Fait
    *Calcul de π
    PI = 2 . PI
    Ecrire PI
Fin
```

PROGRAMME-1

```
      pi=1
      N=1
      DO WHILE (N .LE. 300)
        pi = pi*4.*n**2 / (4.*n**2-1)
        N=N+1
      ENDDO
      pi = 2*pi
      write(*, *) pi
      stop
      end
```

Algorithme-2

```
Début
   PI=1
   N=1
1  PI = PI . 4.N^2 / (4.N^2-1)

   N=N+1
   Si N≤ 300
      Alors
         Aller  à  1
      Sinon
         PI = 2 . PI
         Ecrire PI
   Fsi
Fin
```

PROGRAMME-2

```
      pi=1
      n=1
1     pi=pi*4.*n**2/(4.*n**2-1.)
      n=n+1
      if(n .le. 300) then
        goto 1
      else
        pi=2*pi
        write (*, *)pi
      endif
      stop
      end
```

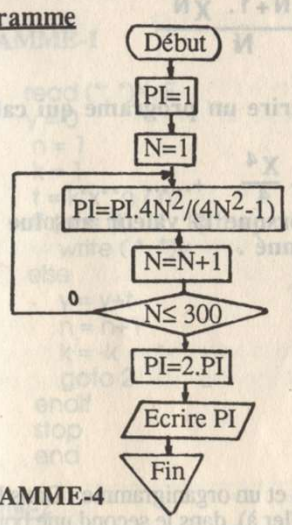
Algorithme-3

Début
PI=1
*Boucle qui calcule les 600 termes de la suite π /2
Pour N de 1 **à** 300
Faire
$PI = PI .4.N^2 / (4.N^2-1)$
Fait
PI = 2 . PI
Ecrire PI
Fin

PROGRAMME-3

```
      pi=1
      do 5 n=1 , 300
5     pi=pi*4*n**2 / ( 4.*n**2-1.)
      pi=2*pi
      write(*, *)pi
      stop
      end
```


Organigramme



PROGRAMME-4

```
      pi=1
      n=1
5     pi=pi*4.*n**2 / (4.*n**2-1.)
      n=n+1
      if(n .le. 300)  goto 5
      pi=2*pi
      write(*, *)pi
      stop
      end
```

L'exécution de tous ces programmes , donne PI = 3.1389800 . Si nous choisissons 1200 termes (n varie de 1 à 600), nous améliorons le résultat :

Pour N variant de 1 à 1200 , l'exécution donne : PI = 3.140284

EXERCICE 11 CALCUL D'UNE SERIE DE TERME $\frac{(-1)^{N+1}.\ X^N}{N}$

Etant donné X (X<1), écrire un programe qui calcule et imprime la valeur de :

$$Y = X - \frac{X^2}{2} + \frac{X^3}{3} - \frac{X^4}{4} + \ldots\ldots$$

Nous arrêtons les calculs lorsque la valeur absolue d'un terme sera inférieure à E donné .

SOLUTION :

L'expression générale est :

$$Y = \sum_{N=1}^{\infty} \frac{(-1)^{N+1}\ X^N}{N}$$

Nous proposons deux algorithmes et un organigramme . Dans le premier nous utilisons un branchement (aller à), dans le second une boucle (tant que) .

Algorithme-1

```
Début
    *Lecture des valeurs de X et E.
    Lire X, E
    Y=0
    N=1
    K=1
    *T : variable utilisée pour définir un terme.
2   T = K . X^N /N
    Si | T | < E
      Alors
        *Impression de Y pour lequel | T | < E.
        Ecrire Y
      Sinon
        Y=Y+T
        N=N+1
        *K : définit le signe du terme.
        K= -K
        Aller à 2
      Fsi
Fin
```


PROGRAMME-1

```
      read (*, *)X, E
      y = 0
      n = 1
      k = 1
2     t = k*x**n / n
      if (abs (t) .LT. E) then
        write ( *, *)y
      else
        y = y+t
        n = n+1
        k = -k
        goto 2
      endif
      stop
      end
```

Algorithme-2

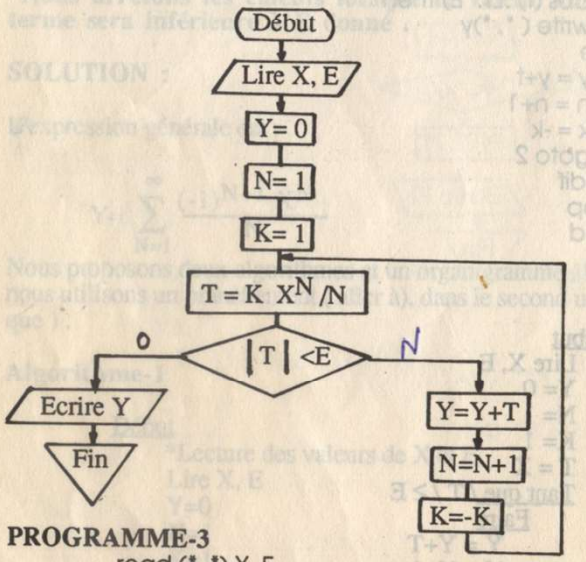
```
Début
    Lire X, E
    Y= 0
    N= 1
    K= 1
    T = X
    Tant que / T / ≥ E
      Faire
        Y = Y+T
        N = N+1
        K = -K
        T = K . X^N /N
      Fait
      Ecrire Y
Fin
```

PROGRAMME-2

```
read (*, *) X, E
DATA y, n, k / 0, 1, 1 /
Do while (abs (t) .GE. E)
  y=y+t
  n=n+1
```

```
      k= -k
      t=k*x**n / n
    enddo
    write (*, *)y
    end
```

Organiramme



PROGRAMME-3

```
      read (*, *) X, E
      data   y, n, k / 0, 1, 1 /
25    t=k*x**n / n
      if (abs (t) .LT. E) goto 15
      y=y+t
      n=n+1
      k= -k
      goto 25
15    write (*, *)y
      stop
      end
```

Exécution :

Pour X = 0.999 et E = 10^{-5} nous avons Y= 6.926416E-001

Pour X = 1 et E = 10^{-4} nous avons Y = 6.93091 7E-001

Pour X = 1.1 et E = 10^{-4}, il n y a pas de résultat

EXERCICE 12 RACINE CARREE D'UN NOMBRE

Ecrire un programme qui calcule la racine carrée d'un nombre réel A en utilisant la suite de Newton :

$$\begin{cases} X_0 = A \\ X_{N+1} = \frac{1}{2}\left(X_N + \frac{A}{X_N} \right) \end{cases}$$

Nous arrêtons le calcul quand $|X_{N+1}-X_N| < \varepsilon$, ε **donné**
Nous posons : $Y=X_{N+1}$ **et** $X=X_N$

SOLUTION :
Nous proposons deux algorithmes et un organigramme .
Nous avons préféré utiliser des variables non indicées pour faciliter la solution .

Algorithme-1

```
Début
   Lire A, E
   X=A
1  Y= 1/2 ( X + A/X )
   Si |Y-X| < ε
     Alors
       Ecrire Y
     Sinon
       X=Y
       Aller à 1
   Fsi
Fin
```

PROGRAMME-1

```
      read (*, *) A, E
      X=A
20    Y=1 / 2.*(X+A / X)
      if ( ABS (Y-X) .lt. E) then
      write (*, *) Y
```

```
        else
           X= Y
           GOTO 20
        endif
        stop
        end
```

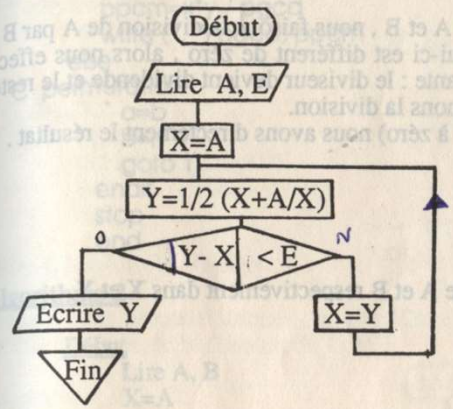
Algorithme-2

```
Début
    Lire A, E
    *Initialisation.
    X=A
    Y = A/2
    Tant que | Y-X | ≥ E

      Faire
        X=Y
        Y = 1/2 ( X+A/X)
      Fait
    Ecrire  Y
  Fin
```

PROGRAMME-2

```
      read (*, *) A, E
      X=A
      Y=A/2
      do while (ABS (Y-X)  .ge.  E)
         X=Y
         Y=1/2.*(X+A / X)
      enddo
      Write (*, *) Y
      stop
      end
```


Organigramme



PROGRAMME-3

```
      read (*, *)A, E
      X=A
20    Y=1 /2.*(X+A / X)
      if (ABS(Y-X) .lt. E)  goto 10
      X=Y
      GOTO 20
10    Write (*, *)Y
      stop
      end
```

Pour A = 17 et E = 10^{-4} , l'exécution donne Y= 4.1231060.

EXERCICE 13. PGCD ET PPCM DE DEUX NOMBRES ENTIERS

Ecrire un programme qui calcule le PGCD de deux nombres a, b tel que a > b ; en déduire le PPCM de ces nombres .

SOLUTION:

Nous vous proposons deux solutions , avec deux algorithmes et un organigramme chacune .

PREMIERE SOLUTION :

Pour trouver le PGCD de A et B , nous faisons la division de A par B , et testons le reste. Si celui-ci est différent de zéro , alors nous effectuons la permutation suivante : le diviseur devient dividende et le reste diviseur , puis nous reprenons la division.
Sinon (si le reste est égal à zéro) nous avons directement le résultat .

Algorithme-1

```
Début
  Lire A, B
  *Sauvegarde de A et B respectivement dans X et Y.
  X=A
  Y=B
1 Q= A/B

  R=A-B . Q
  Si R = 0
    Alors
      PGCD= B

      PPCM = X.Y/PGCD

      Ecrire PGCD, PPCM
    Sinon
      *Permutation.
      A=B
      B=R
      Aller à 1
  Fsi
Fin
```

PROGRAMME-1

```
      integer a, b, x, y, q, r, ppcm, pgcd
      read (*, *) a, b
      x=a
      y=b
1     q=a / b
      r=a-b*q
      if (r.eq.0) then
C résultats
      pgcd=b
      ppcm=x*y / pgcd
      write (*, *) pgcd, ppcm
    else
C permutation
      a=b
      b=r
      goto 1
    endif
    stop
    end
```

Algorithme-2

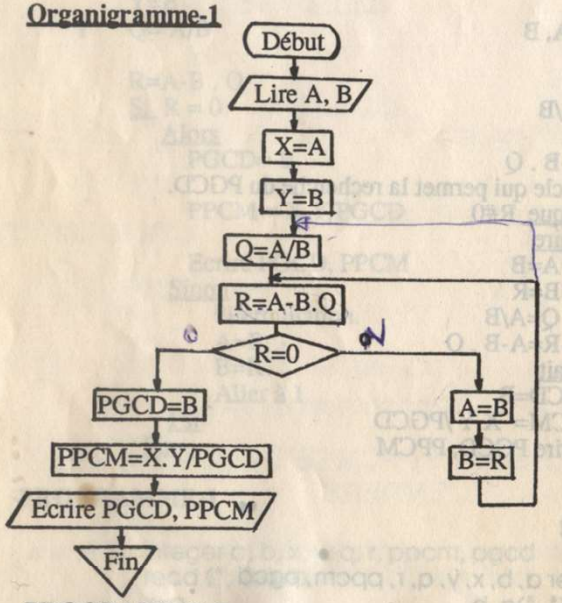
```
Début
    Lire A, B
    X=A
    Y=B
    Q= A/B

    R=A-B . Q
    *Boucle qui permet la recherche du PGCD.
    Tant que R#0
      Faire
        A=B
        B=R
        Q=A/B
        R=A-B . Q
      Fait
    PGCD=B
    PPCM= X.Y /PGCD
    Ecrire PGCD, PPCM
Fin
```

PROGRAMME-2

```
      integer a, b, x, y, q, r, ppcm, pgcd
      read (*, *)a, b
      x=a
      y=b
      q=a / b
```

```
      r=a-b*q
      do while (r .ne. 0)
      a=b
      b=r
      q=a / b
      r=a-b*q
enddo
pgcd=b
ppcm=x*y / pgcd
write (*, *) pgcd, ppcm
stop
end
```

Organigramme-1



PROGRAMME-3

```
      integer a, b, x, y, q, r, ppcm, pgcd
      read (*, *) a, b
      x=a
      y=b
5     q=a / b
      r=a-b*q
      if (r.eq.0) goto 10
C permutation
      a=b
      b=r
      goto 5
C résultat
10    pgcd=b
      ppcm=x*y / pgcd
      write (*, *)pgcd, ppcm
      end
```

DEUXIEME SOLUTION :

Dans cette seconde solution, nous utilisons la fonction MOD pour calculer le reste de la division de I par J .

Algorithme-1

```
Début
  *Utilisons la variable I au lieu de A et J au lieu de B
  Lire I, J
  K=I
  L=J
7 R=MOD (I, J)
  Si R=0
    Alors
      PGCD=J

      PPCM=K.L/PGCD

      Ecrire PGCD, PPCM
    Sinon
      *Permutation
      I=J
      J=R
      Aller à 7
  Fsi
Fin
```

PROGRAMME-4

```
      integer r, pgcd, ppcm
      read(*, *)i, j
      k=i
      L=j
7     r=mod(i, j)
      if(r .eq. 0) then
C résultat
      pgcd=j
      ppcm=k*L / pgcd
      write (*,*) pgcd, ppcm
    else
C permutation
      i=j
      j=r
      goto 7
    endif
    stop
    end
```

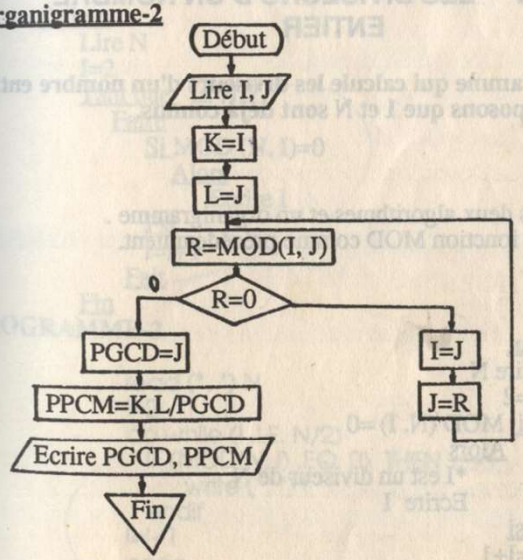
Algorithme-2

```
Début
    Lire I, J
    *Sauvegarde des valeurs de I et J.
    K=I
    L=J
    R=MOD(I, J)
    Tant que R#0
      Faire
        *Permutation.
        I=J
        J=K
        R = MOD (I, J)
      Fait
    PGCD=J
    PPCM = K.L/PGCD

    Ecrire PGCD, PPCM
Fin
```


PROGRAMME-5

```
      integer r, pgcd, ppcm
      read (*, *)i, j
      k=i
      L=j
      r=mod (i, j)
      do while (r .ne. 0) then
C Permutation
      i=j
      j=r
      r=mod (i, j)
    enddo
C résultat
    pgcd=j
    ppcm=k*L/pgcd
    write (*, *)pgcd, ppcm
    stop
    end
```

Organigramme-2



PROGRAMME-6

```
      integer r, pgcd, ppcm
      read (*, *)i, j
      k=i
      L=j
10    r=mod (i, j)
      if (r .eq. 0) goto 5
C permutation
      i=j
      j=r
      goto 10
C résultat
5     pgcd=j
      ppcm=k*L/pgcd
      write (*, *) pgcd, ppcm
      stop
      end
```

EXERCICE 14 LES DIVISEURS D'UN NOMBRE ENTIER

Ecrire un programme qui calcule les diviseurs d'un nombre entier N ≥ 2. Nous supposons que 1 et N sont déjà connus.

SOLUTION :

Nous proposons deux algorithmes et un organigramme .
Nous utilisons la fonction MOD comme précédemment.

Algorithme-1

```
Début
   Lire N
   I=2
1  Si MOD (N, I) =0
      Alors
         *I est un diviseur de N.
         Ecrire I
   Fsi
   I=I+1
```



```
   Si    I ≤ N/2
      Alors
         Aller à 1
   Fsi
Fin
```

PROGRAMME-1

```
      read (*, *)N
      I=2
1     if (MOD(N, I) .EQ. 0) then
         write (*, *) I
      endif
      I=I+1
      IF (I .LE. N/2) then
        GOTO 1
      endif
      end
```

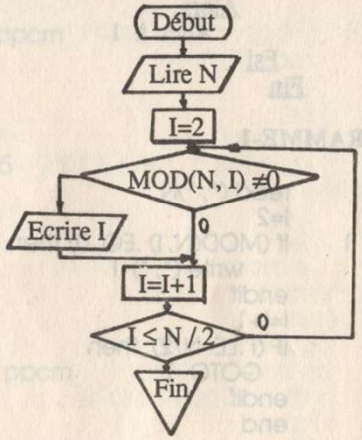
Algorithme-2

```
Début
   Lire N
   I=2
   Tant que I≤ N/2
      Faire
         Si MOD (N, I)=0
            Alors
               Ecrire I
         Fsi
         i=i+1
      Fait
Fin
```

PROGRAMME-2

```
      read (*, *) N
      I=2
      do while (I .LE. N/2)
        IF (MOD(N, I) .EQ. 0) THEN
           write (*, *) I
        endif
      I=I+1
      endo
      stop
      end
```

Organigramme



PROGRAMME-3

```
      read (*, *)N
      I=2
1     if (MOD (N, I). NE.0)  goto 2
      write (*, *) I
2     I=I+1
      IF (I.LE.N/2)  GOTO  1
      stop
      end
```

EXERCICE 15 FACTEURS PREMIERS D'UN NOMBRE ENTIER

Ecrire un programme qui divise un nombre entier en facteurs premiers .

SOLUTION :

Nous appelons facteur premier tout nombre n'admettant la division que par des nombres divisibles par 1 et par eux-même (ex : 2, 3, 5, 7, 11, ...) . Nous utilisons la fonction MOD pour faire cette recherche .

Nous vous proposons un algorithme et un organigramme .

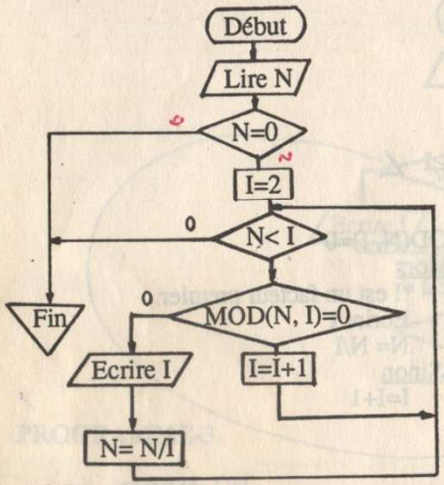
Algorithme

```
Début
   Lire N
   Si N#0
      Alors
         I=2
         Tant que N≥I
            Faire
               Si MOD(N, I)=0
                  Alors
                     *I est un facteur premier.
                     Ecrire I
                     N= N/I
                  Sinon
                     I=I+1
               Fsi
            Fait
   Fsi
Fin
```

PROGRAMME-1

```
      read (*, *)n
      if (n .ne. 0) then
        i=2
        do while (n .ge. i)
          if (mod(n, i) .eq. 0) then
C       I est un facteur premier.
            write (*, *)i
            n=n/i
          else
            i=i+1
          endif
        enddo
      endif
      end
```

Organigramme



PROGRAMME-2

```
      read (*, *)n
      if (n .eq. 0) stop
      i=2
2     if (n .lt, i) stop
      if (mod(n, i) .eq. 0)  goto  1
      i=i+1
      goto  2
C     I est un facteur premier .
1     write (*, *)i
      n=n / i
      goto  2
      end
```

EXERCICE 16 PUISSANCE ET FACTORIELLE D'UN NOMBRE

N étant un entier positif connu, écrire un programme qui calcule la valeur de H donné par :

H= N^{10} si N≥20

H= N! si N<20

SOLUTION :
Nous vous proposons deux algorithmes et un organigramme .
Dans le premier nous utilisons des branchements et dans le second des boucles .

Algorithme-1

```
Début
  Lire N
  H=1
  Si N≥ 20
     Alors
        *Calulons H = N^10
        I=1
     1  H = H . N
        I=I+1
        Si I≤ 10
           Alors
              Aller à 1
        Fsi
     Sinon
        *Calculons H = N !
        I=1
     2  H = H . I
        I=I+1
        Si I≤ N
           Alors
              Aller à 2
        Fsi
  Fsi
  Ecrire H
Fin
```

PROGRAMME-1

```
      read (*, *)n
      h=1
      if (n .ge. 20)  then
C     calcul de H = N^10
      i=1
1     h=h*n
      i=i+1
      if (i .le. 10)  then
         goto  1
      endif
```

```
        else
C       calculons H = N !
      2   h=h*i
          i=i+1
          if (i .le. n) then
            goto 2
          endif
        endif
        write (*, *)H
        stop
        end
```

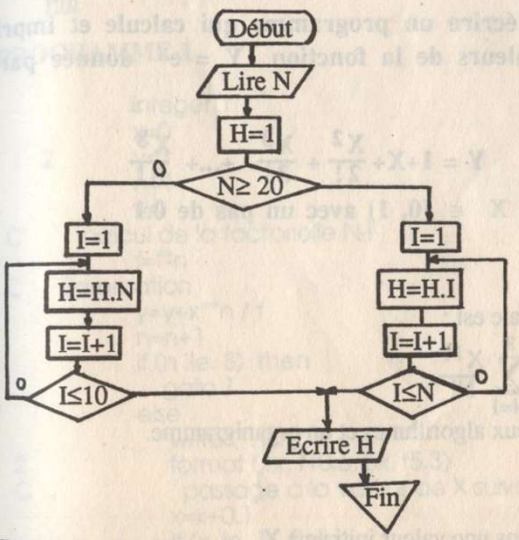
Algorithme-2

```
Début
  Lire N
  H=1
  Si N≥ 20
     Alors
       Pour I de 1 à 10
         Faire
           H = H . N
         Fait
     Sinon
       Pour  I de 1 à N
         Faire
           H = H . I
         Fait
  Fsi
  Ecrire H
Fin
```

PROGRAMME-2

```
      read (*, *)n
      h=1
      if (n .ge. 20) then
        Do 5 i =1, 10
C         calcul de H = N10
          h=h*n
  5     continue
      else
        DO 10 i =1, N
C     calcul de H = N !
          h=h*i
 10     continue
      endif
      write (*, *)h
      stop
      end
```

Organigramme



PROGRAMME-3

```
      read (*, *)n
      h=1
      if (n .ge. 20)  goto  10
      i=1
C  calcul de H = N !
  15  h=h*i
      i=i+1
      if (i.le.n)  goto  15
      goto  20
  10  i=1
C  calcul de H = N10
```

```
25  h=h*n
    i=i+1
    if (i.le.10) goto 25
20  write (*, *)h
    stop
    end
```

EXERCICE 17 . CALCUL DE LA VALEUR APPROCHEE DE e^x

On demande d'écrire un programme qui calcule et imprime une table de valeurs de la fonction $Y = e^x$ donnée par la suite :

$$Y = 1+X+\frac{X^2}{2!}+\frac{X^3}{3!}+\ldots+\frac{X^8}{8!}$$

dans l'intervalle $X \in [0, 1]$ avec un pas de 0.1

SOLUTION :

L'expression générale est :

$$Y=1+\sum_{N=1}^{8}\frac{X^N}{N!}$$

Nous proposons deux algorithmes et un organigramme.

Algorithme-1

```
Début
    *Donnons une valeur initiale à X.
    X=0
2   Y=1
    N=1
    F=1
    *Calcul de la factorielle N !
1   F = F . N
    *Sommation.
    Y = Y + X^N / F
    N=N+1
    Si N≤ 8
      Alors
        *Reprendre le calcul.
        Aller à 1
      Sinon
        Ecrire Y
```



```
        *Passage à la valeur de X suivante.
        X=X+0.1
        Si X≤1
          Alors
            *Reprendre à nouveau le calcul de Y.
            Aller à 2
          Fsi
    Fsi
Fin
```

PROGRAMME-1

```
      integer f
      x=0
2     y=1
      n=1
      f=1
C     calcul de la factorielle N !
1     f=f*n
C     Sommation
      y=y+x**n / f
      n=n+1
      if (n .le. 8) then
        goto 1
      else
        write (*, 2) y, x
2       format (1x, f10.6, 5x, f5.3)
C         passage à la valeur de X suivante
        x=x+0.1
        if (x .le. 1) then
          goto 2
        endif
      endif
      end
```

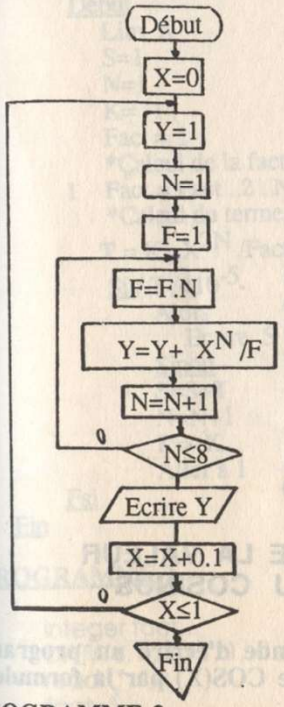
Algorithme-2

```
Début
  X=0
  Tant que  X≤1
    Faire
      Y=1
      N=1
      F=1
      Tant que  N≤8
        Faire
          F= F . N
          Y= Y +   X^N /F
          N=N+1
        Fait
      Ecrire Y
      X=X+0.1
    Fait
Fin
```

PROGRAMME-2

```
      integer f
      x=0
      DO while (x .le. 1)
        y=1
        n=1
        f=1
      DO while (n .le. 8)
         f=f*n
         y=y+x**n / f
         n=n+1
       enddo
       write (*, 2) y, x
2      format (1x, f10.6, 5x, f5. 3)
       x=x+0.1
     enddo
     end
```


Organigramme



PROGRAMME-3

```
            integer f
            x=0
      10    y=1
            n=1
            f=1
C       calcul de la factorielle  N !
      5     f=f*n
C       sommation
            y=y+x**n / f
            n=n+1
            if (n .le. 8)  goto 5
            write (*, 2) y, x
      2     format (1x, f10.6, 5x, f5.3)
```

```
C  passage à la valeur de X suivante
      x=x+0.1
       if (x .le. 1) goto 10
      stop
      end
```

L'exécution donne :

Y	X
1.000000	.000
1.105171	.100
1.221403	.200
1.349859	.300
1.491825	.400
1.648721	.500
1.822119	.600
2.013753	.700
2.225541	.800
2.459602	.900
2.718279	1.000

EXERCICE 18 CALCUL DE LA VALEUR APPROCHEE DU COSINUS

Soit X un réel donné, on demande d'écrire un programme qui calcule et imprime la valeur de COS(X) par la formule :

$$COS(X) = 1-\frac{X^2}{2!}+\frac{X^4}{4!}-\frac{X^6}{6!}+\ldots$$

L'arrêt de calcul est fixé lorsque la valeur absolue d'un terme sera inférieure à 10^{-5}

SOLUTION :

L'expression générale est $S = 1+\sum_{N=1}^{\infty}\frac{(-1)^N \cdot X^{2N}}{(2N)!}$

Nous proposons deux algorithmes et un organigramme .
Nous utilisons la variable K pour le signe, Fact pour le calcul de la factorielle et T pour le terme de la suite, avec

$$T = K.\frac{X^{2N}}{Fact}$$

Algorithme-1

```
Début
   Lire X
   S=1
   N=1
   K= -1
   Fact = 1
   *Calcul de la factorielle (2.N) !
1  Fact = Fact . 2 . N . ( 2 . N - 1)
   *Calcul du terme de la suite .
   T = K . X^2N /Fact
   Si / T /<10^-5
      Alors
         Ecrire S
      Sinon
      S=S+T
      N=N+1
      K= -K
      Aller à 1
   Fsi
Fin
```

PROGRAMME-1

```
      integer fact
      DATA s, n, k / 1, 1, -1 /
      read (*, *)x
      fact=1
1     fact=fact *2*n*(2*n-1)
      t=k*x**(2*n) / fact
      IF (abs(t) .Lt. 1.E-5) THEN
         write (*, 10) s
10       format (2x, 28Hvaleur de COS(X) en radian :, E14.6)
      ELSE
         s=s+t
         n=n+1
         k=-k
         goto 1
      ENDIF
      stop
      end
```

Algorithme-2

Début
Lire X
S=0
N=1
K= -1
Fact=1
T=1
*Boucle qui calcul le COSINUS.

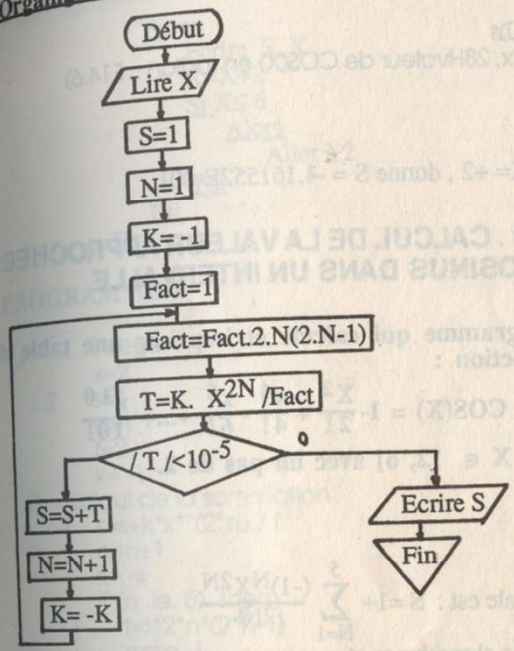
Tant que / T /≥ 10^{-5}
Faire
S=S+T
N=N+1
*K est utilisé pour le signe.
K= -K
Fact = Fact . 2. N . (2.N - 1)

T = K. X^{2N} /Fact
Fait
Ecrire S
Fin

PROGRAMME-2

```
      integer fact
      DATA s, n, k, fact, T / 0, 1, -1, 1, 1 /
      read (*, *)x
      DO WHILE  (abs(t) .GE. 1.E -5)
         s=s+t
         n=n+1
         k=-k
         fact = fact *2*n*(2*n-1)
         t=k*x**(2*n) / fact
      ENDDO
      write (*, 10)s
10    format (2x, 28Hvaleur de COS(X) en radian :, E14.6)
      stop
      end
```


Organigramme



PROGRAMME-3

```
      integer fact
      read (*, *)x
      s=1
      n=1
      k= -1
      fact=1
300   fact=fact*2*n*(2*n-1)
      t=k*x**(2*n) / fact
      if (abs (t).lt.1.E-5)  goto  50
      s=s+t
      n=n+1
```

```
      k=-k
      goto 300
50    write (*, 10)s
10    format (2x, 28Hvaleur de COS(X) en radian :, E14.6)
      stop
      end
```

L'exécution pour X= +2 , donne S = -4.161552E-001.

EXERCICE 19 . CALCUL DE LA VALEUR APPROCHEE DE COSINUS DANS UN INTERVALLE

Ecrire un programme qui calcule et imprime une table de valeurs de la fonction :

$$S = COS(X) = 1 - \frac{X^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots + \frac{x^{10}}{10!}$$

dans l'intervalle X ∈ [2, 6] avec un pas de 2.

SOLUTION :

L'expression générale est : $S = 1 + \sum_{N=1}^{5} \frac{(-1)^N X^{2N}}{(2N)!}$

Nous proposons deux algorithmes et un organigramme .

Algorithme-1

```
Début
   *La valeur initiale de X est 2.
   X=2
2  S=1
   N=1
   F=2
   K= -1
1  S = S+K . X^2N / F
   N=N+1
   K= -K
   Si N≤ 5
   Alors
        F = F . 2 . N . (2 . N - 1)
        Aller à 1
```



```
   Sinon
        Ecrire S, X
        X=X+2
        Si X≤ 6
             Alors
                  Aller à 2
        Fsi
   Fsi
Fin
```

PROGRAMME-1

```
      integer f
      x=2
2     s=1
      n=1
      f=2
      k= -1
C     calcul de la sommation
1     s=s+k*x**(2*n) / f
      n=n+1
      k= -k
      if (n .le. 5) then
        f=f*2*n*(2*n-1)
        goto 1
      else
        write (*, 3)s, x
3       format (1x,'s=', f10.6, 4x,'x= ', f3.1)
        x=x+2
        if (x .le. 6) then
          goto 2
        endif
      endif
      stop
      end
```

Algorithme-2

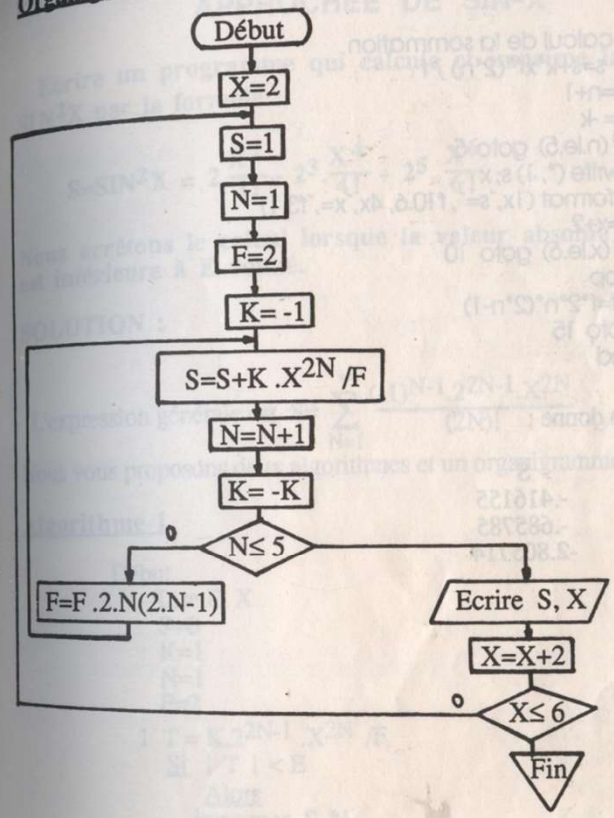
```
Début
   Pour X de 2 à 6 pas 2
        Faire
             *Calcul de la somme S pour un X fixé
             S=1
```

```
        F=2
        k= -1
        Pour N de 1 à 5
          Faire
              S = S+K .X^2N /F
              K= -K
              F = F . 2 . N ( 2 . N - 1)
            Fait
          Ecrire S, X
        Fait
    Fin
```

PROGRAMME-2

```
      integer f, x
      DO 5 X=2, 6, 2
        s=1
        n=1
        f=2
        k= -1
        DO 10 N=1, 5
          s=s+k*x**(2*n) / f
          k= -k
          f=f*2*n*(2*n -1)
10      continue
        write (*, 1) s, x
 1      format (1x,'s=', f10.6, 4x,'x= ', f3.1)
 5    continue
      stop
      end
```


Organigramme



PROGRAMME-3

```
      integer f
      x=2
10    s=1
      n=1
      f=2
      k=-1
```

```
C     calcul de la sommation
 15   s=s+k*x**(2*n) / f
      n=n+1
      k=-k
      if (n.le.5) goto 5
      write (*, 1) s, x
 1    format (1x,'s=', f10.6, 4x,'x=', f3.1)
      x=x+2
      if (x.le.6) goto 10
      stop
 5    f=f*2*n*(2*n-1)
      goto 15
      end
```

L'exécution donne :

X	S
2	-.416155
4	-.685785
6	-2.805714

EXERCICE 22 CALCUL DE LA VALEUR APPROCHEE DE SIN^2X

Ecrire un programme qui calcule et imprime la valeur de SIN^2X par la formule :

$$S=SIN^2X = 2.\frac{X^2}{2!} - 2^3.\frac{X^4}{4!} + 2^5.\frac{X^6}{6!} - \ldots$$

Nous arrêtons le calcul lorsque la valeur absolue d'un terme est inférieure à E donné.

SOLUTION :

L'expression générale est :$S= \sum_{N=1}^{\infty} \frac{(-1)^{N-1}.2^{2N-1}.X^{2N}}{(2N)!}$

Nous vous proposons deux algorithmes et un organigramme .

Algorithme-1

```
Début
    Lire E, X
    S=0
    K=1
    N=1
    F=2
1   T = K.2^(2N-1) .X^2N /F
    Si | T | < E
        Alors
        imprimer S, N
    Sinon
        S=S+T
```

```
            K= -K
            N= N+1
            F = F.2.N.(2.N-1)
            Aller à 1
        Fsi
```

RAMME-1

```
      integer f
      DATA S, K, N, F / 0, 1, 1, 2 /
      read (*, *) E, X
    calcul d'un terme
1     t=k*2*(2*n-1)*x**(2*n) / f
      IF (abs(t) .lt. E) then
         write (*, 50) s, n
50       format (5x,´s= ´, 1x, f10.6, 4x,´n= ´, i2)
      else
C     calcul du sinus carré
         s=s+t
         k= -k
         n= n+1
         f=f*2*n*(2*n-1)
         goto 1
      endif
      stop
      end
```

Algorithme-2

Début

Lire E, X

S=0

K=1

N=1

F=2

$T = X^2$

*Boucle qui calcul le sinus carré.

Tant que / T / ≥E

Faire

S=S+T

K= -K

N=N+1

F = F .2.N.(2.N-1)

$$T = K.2^{2N-1} .X^{2N} /F$$

Fait

imprimer S

Fin

PROGRAMME-2

```
      integer f
      DATA s, k, n, f / 0, 2*1, 2 /
      read (*, *)x, E
      T = X**2
      DO WHILE (abs(t) .GE. E)
         s=s+t
         k= -k
         n=n+1
         f=f*2*n*(2*n-1)
         t=k*2**(2*n-1)*x**(2*n) /f
      enddo
C     impression de la valeur du sinus carré
      write (*, 50) s, n
50    format (5x,´s= ´, 1x, f10.6, 4x,´n=´, i2
      stop
      end
```

PROGRAMME-3

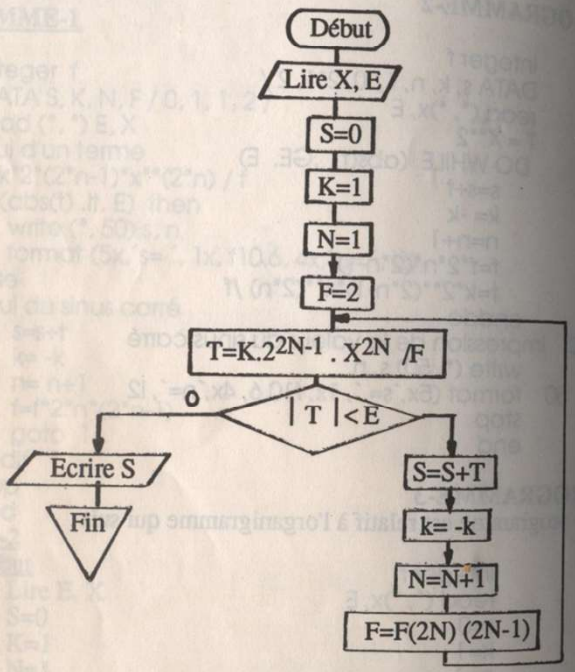
Ce programme est relatif à l'organigramme qui suit .

```
      integer f
      read (*, *)x, E
      s=0
      k=1
      n=1
      f=2
C     calcul d'un terme
200   t=k*2**(2*n-1)*x**(2*n) / f
      if (abs(t) .lt. E) goto 100
C     calcul du sinus carré
      s=s+t
      k= -k
      n=n+1
      f=f*2*n*(2*n-1)
      goto 200
C     impression de la valeur du sinus carré
```

```
100     write (*, 50)s, n
 50     format (5x,'s=', 1x, f10.6, 4x,'n=', i2)
        stop
        end
```

L'exécution donne pour X=2 et E = 1.E-3 :
S = .805079 et N=6

Organigramme



EXERCICE 23 CALCUL DE LA SERIE DE TERME

$$\frac{\frac{N}{10} + N!}{\frac{N+1}{10} + (N!)^2} X^{N-1}$$

Ecrire un programme qui calcule et imprime les valeurs de S donnée par la suite (M terme) :

$$S = \frac{0.1+1!}{0.2+(1!)^2} - \frac{0.2+2!}{0.3+(2!)^2} X + \frac{0.3+3!}{0.4+(3!)^2} . X^2 + \ldots.$$

On prend X dans [4, 20] avec un pas égal à 4.

SOLUTION :
L'espression générale est :

$$S = \sum_{N=1}^{M} H . \frac{\frac{N}{10} + N!}{\frac{N+1}{10} + (N!)^2} X^{N-1}$$

Nous proposons deux algorithmes et un organigramme.
Soit H la variable représentant le signe et F le calcul de la factorielle.

Algorithme-1

```
Début
        Lire M
        X=4
    2   S=0
        N=1
        H=1
        F=1

                    N/10 + F
    1   S = S + H. ----------- X^N-1
                   N+1/10 + F^2

        N=N+1
        Si   N≤ M
             Alors
               H= -H
               F=F.N
               Aller à 1
             Sinon
               Ecrire S
               X=X+4
               Si   X≤ 20
                    Alors
                      Aller à 2
               Fsi
        Fsi
Fin
```

PROGRAMME-1

```
      integer f
        read (*, *)m
      x=4.
  2   s=0
      n=1
      h=1
      f=1
```

```
C  calcul de la somme S
 1        s=s+h*(n / 10.+floa(f)) / ((n+1) / 10. +f**2)
         1*x**(n-1)
          n=n+1
          IF (n .le. m) then
             h=-h
             f=f*n
             goto 1
          else
C   impression de la somme pour un X donné
             write (*, 2)s, x
 2           format (2x, 2hs=, 1x, f16.8, 2 x,´x= ´, f4.1)
             x=x+4
             IF (x .le. 20) then
                goto 2
             endif
       endif
       stop
       end
```

Algorithme-2

Début
INTEGER X
Lire M
Pour X de 4 à 20
Faire
S=0
H=1
F=1
Pour N de 1 à M
Faire
$S = S + H.(N/10+F)/((N+1)/10 +F^2).X^{N-1}$
H= -H
F=F.N
Fait
Ecrire S
Fait
Fin

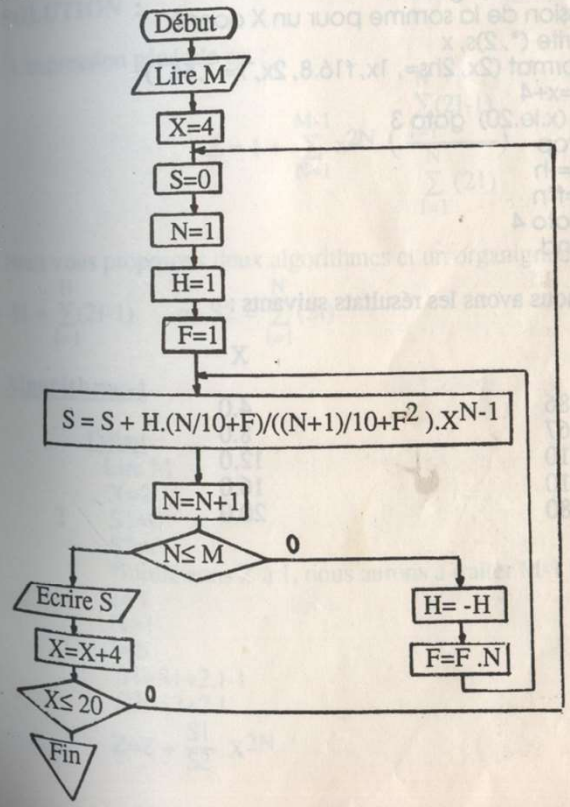
PROGRAMME-2

```
integer f, X
read (*, *)m
do 11 X=4, 20
s=0
h=1
f=1
```



```
C      calcul de la somme S
       s=s+h*(n / 10.+ float (f)) / ((float (n) +1) / 10.+ f**2)
      1*X**(n-1)
       h= -h
       f=f*n
 12    continue
       write (*, 2)s, x
  2    format (2x, 2hs=, 1x, f16. 8, 2x,´x= ´, f4.1)
 11    continue
       stop
       end
```

Organigramme



PROGRAMME-3

```
      integer f
      read (*, *)m
      x=4.
3     s=0
      n=1
      h=1
      f=1
C   calcul de la somme S
4     s=s+h*(n /10.+float (f)) / ((n+1) / 10.+f**2)
     1*x**(n-1)
      n=n+1
      if (n.le.m) goto 1
C   impression de la somme pour un X donné
      write (*, 2)s, x
2     format (2x, 2hs=, 1x, f16.8, 2x,'x= ', f4.1)
      x=x+4
      if (x.le.20) goto 3
      stop
1     h=-h
      f=f*n
      goto 4
      end
```

Pour M = 3, nous avons les résultats suivants :

S	X
1.639386	4.0
7.900567	8.0
19.700210	12.0
37.038310	16.0
59.914880	20.0

EXERCICE 24 — CALCUL DE LA VALEUR D'UNE SERIE

Ecrire un programme qui calcul et imprime les valeurs de Z donnée par :

$$Z = 1 + \frac{1}{2}x^2 + \frac{1+3}{2+4}x^4 + \frac{1+3+5}{2+4+6}x^6 + \ldots$$

Nous prenons en compte M termes .

Le calcul se fait dans l'intervalle $X \in [2, 10]$ avec un pas de 2.

SOLUTION :

L'expression générale est :

$$Z = 1 + \sum_{N=1}^{M-1} x^{2N} \left(\frac{\sum_{I=1}^{N}(2I-1)}{\sum_{I=1}^{N}(2I)} \right)$$

Nous vous proposons deux algorithmes et un organigramme.

$S1 = \sum_{I=1}^{N}(2I-1)$ et $S2 = \sum_{I=1}^{N}(2I)$

Algorithme-1

```
Début
  Lire M
  X=2
2 S1=0
  S2=0
  *Initialisons Z à 1, nous aurons à traiter M-1 termes.
  Z=1
  N=1
1 I=N
  S1=S1+2.I-1
  S2=S2+2.I
  Z=Z + S1/S2 .X^2N
```

```
            N=N+1
            Si    N≤ M-1
                  Alors
                    Aller à 1
                  Sinon
                    Ecrire Z, X
                    X=X+2
                    Si   X≤10
                         Alors
                              Aller à 2
                         Fsi
                  Fsi
            Fin
```

PROGRAMME-1

```
      read (*, *)m
      x=2
 2    s1=0
      s2=0
      z=1
      n=1
 1    i=n
C  calcul de la somme Z
      s1=s1+2*i-1
      s2=s2+2*i
      z=z+s1 / s2*x**(2*n)
      n=n+1
      IF (n .le. m-1) then
        goto 1
      else
C  impression du résultat de la somme
        write (*, 10)z, x
10      format (1x, 2HZ=, 1x, E14.8, 2x,'x='f 5.2)
        x=x+2
        IF (x .le. 10) then
            goto 2
        endif
      endif
      end
```

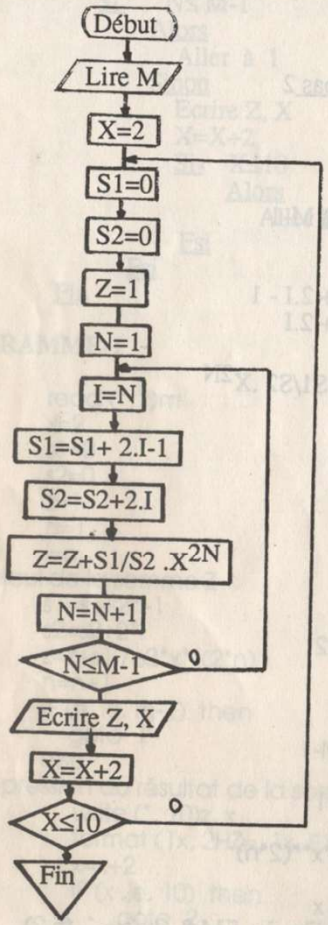

Algorithme-2

```
Début
    Lire M
    Pour X de 2 à 10 pas 2
      Faire
        S1=0
        S2=0
        Z=1
        Pour N de 1 à M-1
          Faire
            I=N
            S1 = S1 + 2.I - 1
            S2 = S2 + 2.I

                Z = Z +S1/S2 .X^2N
          Fait
        Ecrire Z, X
    Fait
Fin
```

PROGRAMME-2

```
      integer X
      read (*, *)m
      DO 100 X=2, 10, 2
        s1=0
        s2=0
        z=1
        DO 90 N=1, M-1
          i=n
          s1= s1+ 2*i-1
          s2= s2+2*i
          z=z+s1 / s2*x**(2*n)
 90     continue
        write (*, 10)z, x
 10     format (1x, 2HZ=, 1x, E14.8, 2x,'x= ', f5.2)
100   continue
      end
```

Organigramme



PROGRAMME-3

```
      read (*, *)m
      x=2
20    s1=0
      s2=0
      z=1
      n=1
5     i=n
C  calcul de la somme Z
      s1=s2+2*i-1
      s2=s2+2*i
      z=z+s1 / s2*x**(2*n)
      n=n+1
      if (n.le.m-1) goto 5
C impression du résultat de la somme
      write (*, 10)z, x
10    format (1x, 2HZ=, 1x, E14.8, 2x,'x= ', f5.2)
      x=x+2
      if (x.le.10)  goto  20
      stop
      end
```

L'exécution de ces programmes pour M = 10, donne :

Z	X
.31315050E+06	2.0
.65915470E+11	4.0
.93982100E+14	6.0
.16467070E+17	8.0
.90897730E+18	10.0

EXERCICE 25 CALCUL DE LA VALEUR D'UNE SERIE

Ecrire un programme qui calcule et imprime la valeur de Y donné

$$Y = 1 + \frac{1}{2} . X^2 + \frac{1.3}{2.4} . X^4 + \frac{1.3.5}{2.4.6} . X^6 + \ldots$$

On prend M termes de la suite précédente. On suppose X donné.

SOLUTION :

L'expression générale est : $Y = 1 + \sum_{I=1}^{M-1} X^{2N} \prod_{I=1}^{N} \frac{2.I-1}{2I}$

Nous vous proposons deux algorithmes avec un organigramme.

Posons : $S = \prod_{I=1}^{N} \frac{2.I-1}{2I}$

Algorithme-1

```
Début
   Lire M, X
   S=1
   Y=1
   N=1
1  I=N
   S = S . (2.I-1)/(2.I)
   Y=Y + S . X^2N
   N=N+1
   Si    N≤M-1
      Alors
         Aller à 1
      Sinon
         Ecrire Y
   Fsi
Fin
```


PROGRAMME-1

```
      data s, y, n / 3*1/
      read (*, *)m, x
C     calcul de la somme Y
C1    i=n
      s=s*(2*i-1) / (2*i)
      y=y+s*x**(2*n)
      n=n+1
      IF (n .le. m-1) then
C     impression de la valeur de Y
        goto 1
      else
        write (*, 100)y
100     format (1x,'y=', E15.8)
      endif
      end
```

Algorithme-2

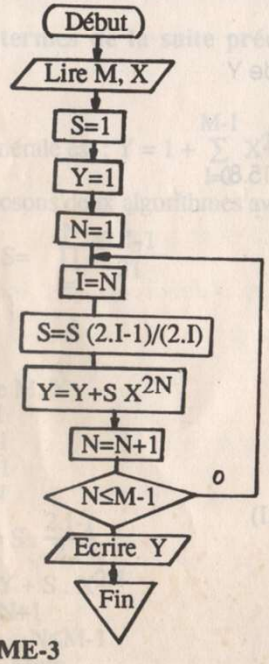
```
Début
   Lire M, X
   S=1
   Y=1
   Pour N de 1 à M-1
      Faire
         I=N
         S = S . (2.I-1)/(2.I)

         Y=Y+S .X^2N
      Fait
   Ecrire Y
Fin
```

PROGRAMME-2

```
      data s, y / 2*1 /
      read (*, *) m, x
C     calcul dela somme Y
      do 16 N=1, M-1
         i=n
         s=s*(2*i-1) / (2*i)
         y=y+s*x**(2*n)
```

```
      16  continue
          write (*, 100)y
     100  format (1x,'y=', E15.8)
          end
```

Organigramme



PROGRAMME-3

```
          data s, n, y / 3*1 /
          read (*, *)m, x
C     calcul de la somme Y
   50     i=n
          s=s*(2*i-1)/(2*i)
          y=y+s*x**(2*N)
          n=n+1
          if (n.le.m-1)  goto  50
          write (*, 100)y
  100     format (1x,'y= ', E15.8)
          stop
          end
```


Pour M = 32 et X = 4, nous obtenons Y = .22920140E+37

EXERCICE 26 CALCUL D'UNE SOMME

Soit l'expression suivante $w = \sum_{I=1}^{M} [\, I \sum_{J=1}^{N} (\prod_{K=1}^{L} (I+J)^K) \,]$

Nous demandons l'écriture d'un programme qui calcule la valeur de w.

SOLUTION :

Posons $S1 = \prod_{K=1}^{L} (I+J)^K$; $S2 = \sum_{J=1}^{N} S1$ et $w = \sum_{I=1}^{M} (I . S2)$

Nous proposons deux algorithmes et deux organigrammes.

Algorithme-1

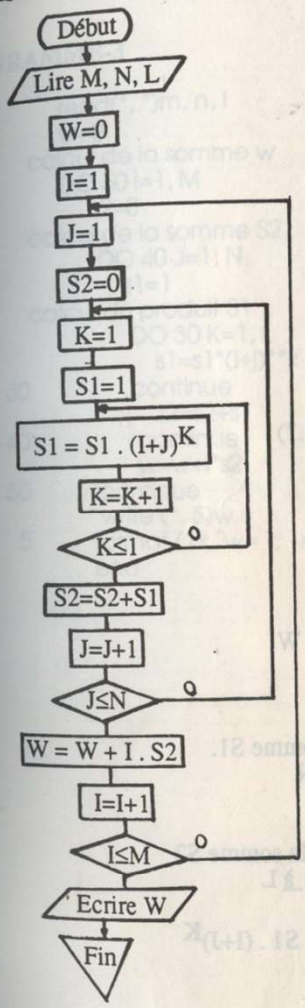
```
Début
    *Lecture des bornes supérieures des sommes.
    Lire M, N, L
    W=0
    I=1
    *Calcul de la somme W.
3   J=1
    S2=0
    *Calcul de la somme S2
2   K=1
    S1=1
    *Calcul du produit S1.

1   S1 = S1 . (I+J)^k
    K=K+1
    Si  K ≤ L
        Alors
            Aller à 1
        Sinon
            S2 = S2 + S1
            J=J+1
            Si  J ≤ N
                Alors
```

```
                Aller à 2
              Sinon
            W = W + I . S2
            I=I+1
            Si I ≤ M
                Alors
                  Aller à 3
                Sinon
                  Ecrire W
            Fsi
        Fsi
    Fsi
Fin
```

PROGRAMME-1

```
      read (*, *)m, n, l
      w=0
      i=1
3     j=1
      s2=0
2     k=1
      s1=1
1     s1=s1*(i+j)**k
      k=k+1
      IF (k . le. L) then
        goto 1
      else
        s2 = s2+s1
        j=j+1
        IF (j .le. n) then
          goto 2
        else
          w=w+i*s2
          i=i+1
          IF (i .le. m) then
            goto 3
          else
            write (*, 5)w
´5          format (1x,´W=´, 1x, E17.7)
          endif
      endif
      stop
      end
```


Organigramme-1



PROGRAMME-2

```
      read (*, *)m, n, l
      w=0
      i=1
30    j=1
      s2=0
20    k=1
      s1=1
10    s1=s1*(i+j)**k
      k=k+1
      if(k.le.l) goto 10
      s2=s2+s1
      j=j+1
      if(j.le.n) goto 20
      w=w+i*s2
      i=i+1
      if(i.le.m) goto 30
      write(*, 5)w
5     format (1x,'w=', 1x, E17.7)
      end
```

Algorithme-2

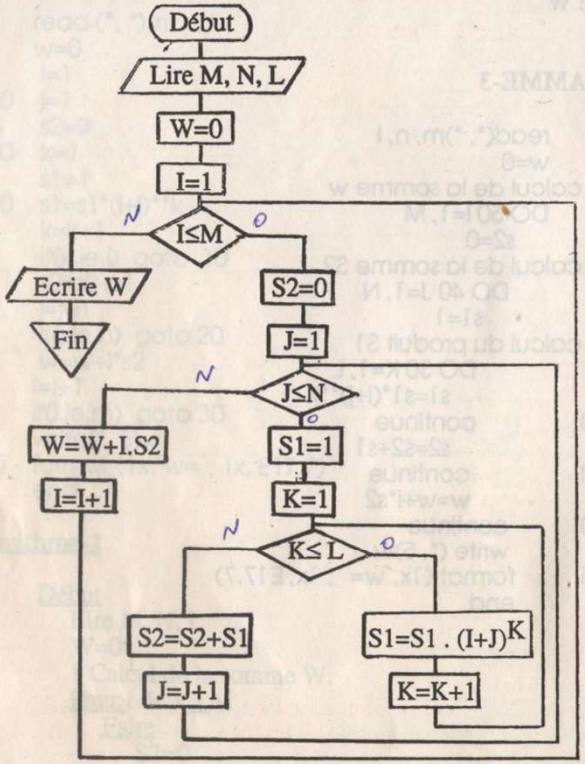
Début
  Lire M, N, L
  W=0
  * Calcul de la somme W.
  Pour I de 1 à M
    Faire
      S2=0
      *Calcul de la somme S1.
      Pour J de 1 à N
        Faire
          S1=1
          *Calcul de la somme S2
          Pour K de 1 à L
            Faire
              $S1 = S1 . S1 . (I+J)^K$
          Fait
        S2=S2+S1
    Fait
  W=W+I.S2

Fait
Ecrire W
Fin

PROGRAMME-3

```
      read(*, *)m, n, l
      w=0
C     calcul de la somme w
      DO 50 I=1, M
        s2=0
C     calcul de la somme S2
        DO 40 J=1, N
          s1=1
C     calcul du produit S1
          DO 30 K=1, L
            s1=s1*(i+j)**k
30        continue
            s2=s2+s1
40        continue
          w=w+i*s2
50      continue
        write (*, 5)w
5       format (1x,'w= ', 1x, E17.7)
        end
```

Organigramme-2



PROGRAMME-4

```
      read (*, *)m, n, l
       w=0
       i=1
 30    if(i.le.m) goto 10
       write (*, 5)w
 5     format (1x,'w= ', 1x, E17.7)
       stop
 10    j=1
       s2=0
 50    if(j.le.n) goto 20
       w=w+i*s2
       i=i+1
       goto 30
 20    k=1
       s1=1
 60    if (k.le.L) goto 40
       s2=s2+s1
       j=j+1
       goto 50
 40    s1=s1* (i+j) **k
       k=k+1
       goto 60
       end
```

L'exécution donne :

Pour M = 32, N = 1 et L = 1, nous avons W = .1196800E+05
Pour M = 2, N = 2 et L = 5, nous avons W = .2190563E+10

CHAPITRE - II . LES VARIABLES INDICEES (VECTEURS ET MATRICES)

Certains organigrammes ne seront pas traduits en programmes, nous laisserons à l'étudiant le soin de le faire.

EXERCICE 27. SOMME DES ELEMENTS DE LA DIAGONALE D'UNE MATRICE

Ecrire un programme qui calcule la somme des éléments de la diagonale d'une matrice carrée (pour l'exécution, on se limitera à une matrice 5X5).

SOLUTION :

Nous vous proposons deux solutions , avec un algorithme et un organigramme chaqu'une.

Soit A la matrice carrée de dimension N ; S est la variable utilisée pour la somme des éléments de la diagonale A (I, I) pour I allant de 1 à N .
Donc :

$$S = A(1, 1) + A(2, 2) + A(3, 3) + \ldots\ldots + A(N, N) = \sum_{I=1}^{N} A(I, I)$$

PREMIERE SOLUTION : Nous utilisons la boucle DO .

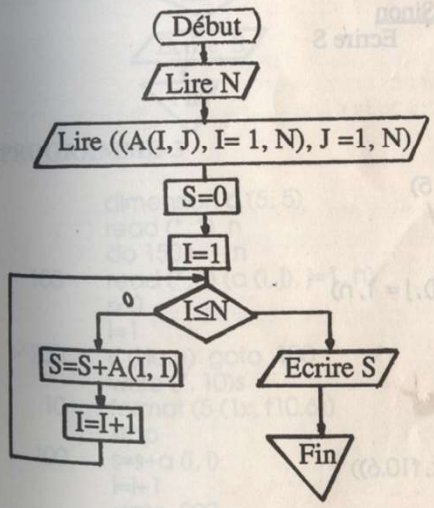
Algorithme-1

Début
*Lecture de la dimension de la matrice.
Lire N
*Lecture de tous les éléments de la matrice.
Lire ((A(I, J), I=1, N), J =1, N)
S=0
*Boucle qui calcule la somme des éléments
*de la diagonale d'une matrice.
Pour I de 1 à N
Faire
*Somme des éléments de la diagonale.
S=S+A(I, I)
Fait
Ecrire S
Fin

PROGRAMME-1

```
      dimension a (5, 5)
      read (*, *) n
      do 15 i =1, n
15    read (*, *) (a (i, j), j =1, n)
      s=0
      DO 20 I=1 , N
         s=s+a (i, i)
20    continue
      write (*, 2)s
2     format (5(1x, f10.6))
      stop
      end
```

Organigramme-1



DEUXIEME SOLUTION : Nous utilisons le test IF...THEN...ELSE

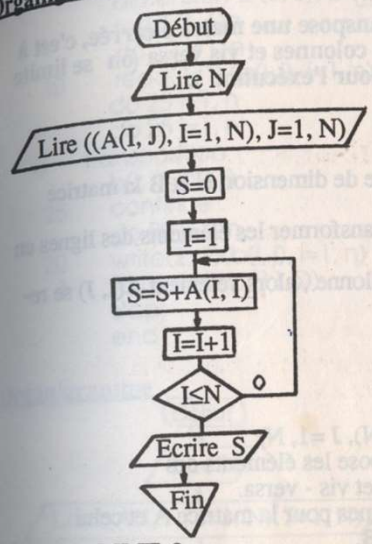
Algorithme-2

```
Début
    Lire N
    Lire ((A(I, J), I=1, N), J = 1, N)
    S=0
    I=1
    *Somme des éléments de la diagonale.
1   S=S+A(I, I)
    I=I+1
    Si  I≤N
        Alors
            *Reprendre le calcul de la somme.
            Aller à 1
        Sinon
            Ecrire S
    Fsi
Fin
```

PROGRAMME-2

```
      dimension a (5, 5)
      DATA S, I / 0, 1 /
      read (*, *) n
      do 150 i=1, n
150   read (*, *) (a (i, j), j = 1, n)
1     s=s+a (i, i)
      i=i+1
      IF (i .le. n) then
          goto 1
      else
          write(*, 10)s
10        format (5 (1x, f10.6))
      endif
      stop
      end
```


Organigramme-2



PROGRAMME-3

```
      dimension a (5, 5)
      read (*, *) n
      do 150 i=1, n
150   read (*, *) (a (i, j), j=1, n)
      s=0
      i=1
200   if (i.le.n) goto 100
      write (*, 10)s
10    format (5 (1x, f10.6))
      stop
100   s=s+a (i, i)
      i=i+1
      goto 200
      end
```

EXERCICE 29. TRANSPOSITION D' UNE MATRICE

Ecrire un programme qui transpose une matrice carrée, c'est à dire les lignes deviennent des colonnes et vis versa (on se limitera à une matrice carrée 5X5 pour l'exécution).

SOLUTION :

Soit A(N, N) une matrice carrée de dimension N et B la matrice transposée de A .
Transposer une matrice , c'est transformer les éléments des lignes en colonnes et vis -versa .
Si I est la ligne dans A et J la colonne , alors l'élément A(I, J) se retrouvera en B(J, I) .

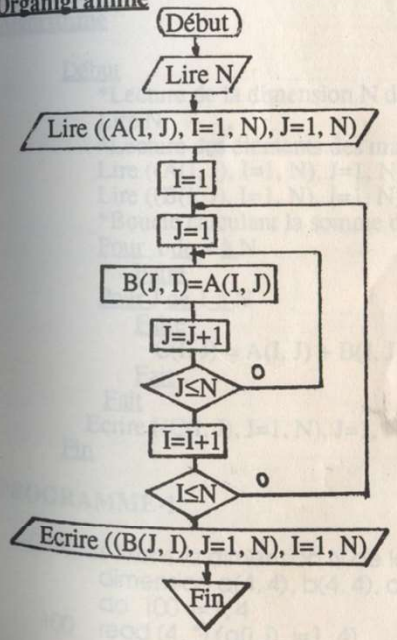
Algorithme

```
Début
    Lire N
    Lire ((A(I, J), I =1, N), J =1, N)
    *Cette boucle transpose les éléments des
    *colonnes en lignes et vis - versa.
    *I est l'indice des lignes pour la matrice A et celui
    *des colonnes pour B.
    Pour I de 1 à N
        Faire
            *J est l'indice des colonnes pour la matrice
            *A est celui des lignes pour B .
            Pour J de 1 à N
                Faire
                    *Opération de transposition.
                    B(J, I) = A(I, J)
                Fait
        Fait
    *Impression de la matrice transposée.
    Ecrire ((B(J, I), J=1, N), I=1, N)
Fin
```


PROGRAMME-1

```
      dimension a (5, 5), b (5, 5)
      read (*, *)n
      do 10 i=1, n
10    read(1, *) (a(i, j), j=1, n)
      do 25 i=1, n
      do 25 j=1, n
C     transposition
      b(j, i)=a(i, j)
25    continue
      do 20 j=1, n
20    write(2, 3)(b(i, j), i=1, n)
3     format (5(1x, f11.5))
      stop
      end
```

Organigramme



PROGRAMME-2

```
      dimension a (5, 5), b (5, 5)
      read (*, *)n
      do 10 i=1, n
10    read (1, *)(a(i, j), j=1, n)
      i=1
 20   j=1
C     transposition
 15   B(j, i)=a(i, j)
      j=j+1
      IF (J .LE. N) goto 15
      i=i+1
      IF (I .LE. N) goto 20
      write (2, 3)(b(i, j), j=1, n), i=1, n)
 3    format (5(1x, f11.5))
      end
```


EXERCICE 31. SOMME DE DEUX MATRICES CARREES

Ecrire un programme qui fait la somme de deux matrices carrées.

SOLUTION :

L'expression mathématique qui donne la somme est :

$$C(I, J) = A(I, J) + B(I, J)$$

Pour obtenir la somme nous procédons comme suit :
Nous fixons I à 1 et nous faisons varier J de 1 à N , puis nous incrémentons I de 1 (I=I+1) et nous répétons l'opération pour J de 1 à N , etc.....

Algorithme

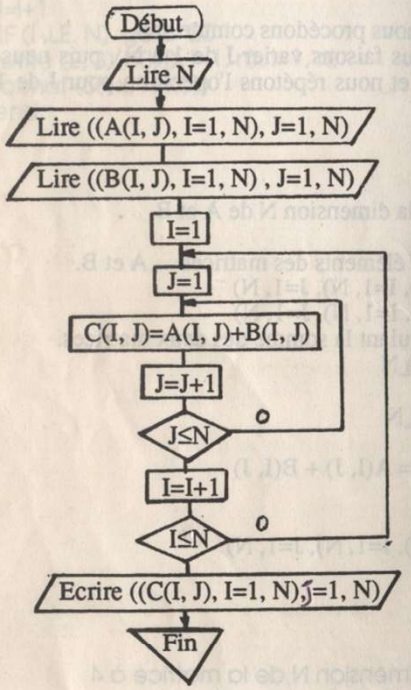
```
Début
    *Lecture de la dimension N de A et B.
    Lire N
    *Lecture des éléments des matrices     A et B.
    Lire ((A(I, J), I=1, N), J=1, N)
    Lire ((B(I, J), I=1, N), J=1, N)
    *Boucle calculant la somme des deux matrices.
    Pour I de 1 à N
        Faire
    Pour J de 1 à N
        Faire
          C(I, J) = A(I, J) + B(I, J)
        Fait
    Fait
  Ecrire ((C(I, J), I=1, N), J=1, N)
Fin
```

PROGRAMME-1

```
C Nous fixons la dimension N de la matrice à 4
      dimension a(4, 4), b(4, 4), c(4, 4)
      do  100  i=1, 4
100   read (4, *) (a(i, j), j=1, 4)
      do 20 i=1, 4
 20   read (5, *) (b(i, j), j=1, 4)
```

```
      do 30 i=1, 4
      do 30 j=1, 4
      c(i, j) = a(i, j) + b(i, j)
30    continue
      do 50 i=1, 4
50    write (6, 40) (c(i, j), j=1, 4)
40    format (4(1x, E14.7))
      stop
      End
```

Organigramme



PROGRAMME-2

```
C     nous fixons la dimension N de la matrice à 4
      dimension a(4, 4), b(4, 4), c(4, 4)
      do 100 i=1, 4
100   read (4, *) (a(i, j) , j=1, 4)
      do 20 i=1, 4
20    read (5,*) (b(i, j), j=1, 4)
C     calcul de la somme des matrices A et B
      i=1
3     j=1
2     c(i, j)=a(i, j)+b(i, j)
      J=J+1
      IF (J .LE. N) goto 2
      I=I+1
      IF (I .LE. N) goto 3
      write (6, 40) (c(i, j), i=1, 4), j=1, 4)
40    format (4(1x, E14.7))
      stop
      end
```

EXERCICE 32 CALCUL D'ELEMENTS D'UN VECTEUR

Soit un vecteur X(N) lu ; écrire un programme qui définit les éléments de Y(N) et les imprime en sachant que :

$$Y_i = \frac{1}{N} \sum_{j=1}^{i} X_j$$

SOLUTION :

Il s'agit de calculer les éléments du vecteur Y à partir de ceux du vecteur X . Pour cela nous devons faire varier J de 1 à I , avec I indice du vecteur Y et J celui de X .

Algorithme

Début

*Lecture de la dimension N des vecteurs X et Y.

Lire N

*Lecture du vecteur X.

Lire (X(I), I=1, N)

*Boucle permettant de définir les éléments de Y.

Pour I de 1 à N

Faire

Y(I)=0

*Calcul de la valeur de l'élément Y(I).

Pour J de 1 à I

Faire

Y(I)=Y(I)+X(J)

Fait

Y(I) = Y(I)/N

*Passage au calcul d'un autre Y(I).

Fait

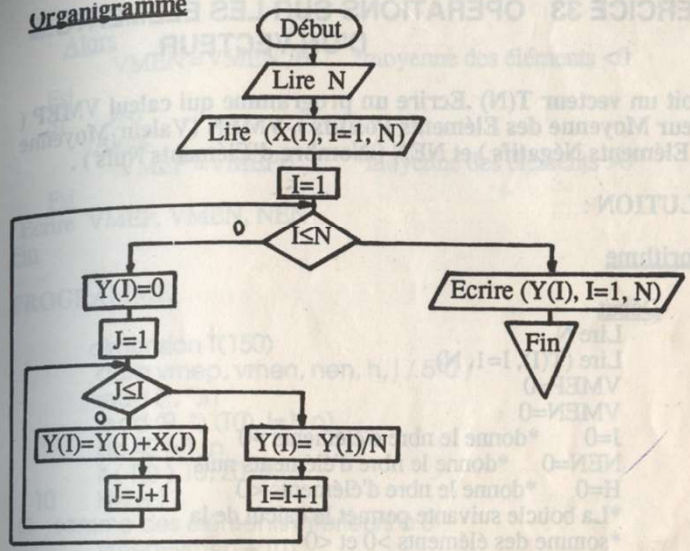
Ecrire (Y(I), I=1, N)

Fin

PROGRAMME-1

```
      dimension X(10), Y(10)
      read (*, *)n, (x(i), i=1, n)
C     calcul des éléments du vecteur Y
      do 5 i=1, n
      y (i)=0
      do 4 j=1, i
      y (i)=y (i)+x (j)
4     continue
      y (i)=y (i) / n
5     continue
C     impression des valeurs du vecteur Y
      do 6 i=1, n
6     write (*, 7) y (i)
7     format (2x,' Y = ', f10.6)
      stop
      end
```


Organigramme



PROGRAMME-2

```
      dimension X(10), Y(10)
      read (*, *)n, (x(i), i=1, n)
C     calcul des éléments du vecteur Y
      i=1
20    if (i .le. n) goto 10
C     impression des valeurs du vecteur Y
      do 6 i=1, n
6     write (*, 7) y (i)
7     format (2x,' Y = ', f10.6)
      stop
10    y (i) =0
      j=1
25    IF (J .LE. I) goto 15
      y (i) = y (i) / n
      I=I+1
      goto 20
15    y (i)= y (i)+x(j)
      J =J+1
      goto 25
      end
```

EXERCICE 33 OPERATIONS SUR LES ELEMENTS D'UN VECTEUR

Soit un vecteur T(N) .Ecrire un programme qui calcul VMEP (Valeur Moyenne des Eléments Positifs), VMEN (Valeur Moyenne des Eléments Négatifs) et NEN (Nombre d'Eléments Nuls) .

SOLUTION :

Algorithme

```
Début
   Lire N
   Lire (T(I), I=1, N)
   VMEP=0
   VMEN=0
   J=0      *donne le nbre d'éléments >0
   NEN=0    *donne le nbre d'éléments nuls
   H=0      *donne le nbre d'éléments <0
   *La boucle suivante permet le calcul de la
   *somme des éléments >0 et <0
   *ainsi que le nbre d'éléments = 0
   Pour I de 1 à N
      Faire
         Si     T(I)>0
               Alors
                  J=J+1
                  *Somme des éléments >0.
                  VMEP=VMEP+T(I)
               Sinon
                  Si   T(i)=0
                     Alors
                        *Nbre d'éléments = 0.
                        NEN=NEN+1
                     Sinon
                        H=H+1
                        * Somme des éléments <0.
                        VMEN=VMEN+T(I)
                  Fsi
      Fsi
   Fait
   *Fin de la boucle
```

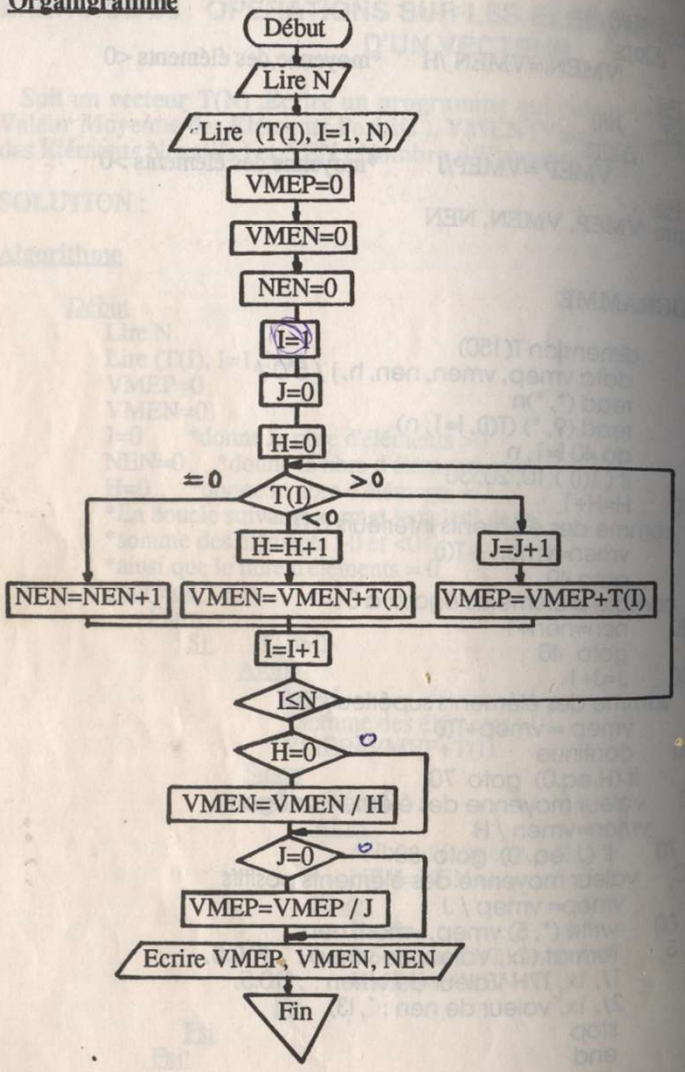


```
Si   H#0
  Alors
     VMEN =VMEN /H     *moyenne des éléments <0

  Fsi
  Si   J#0
     Alors
        VMEP =VMEP/J       *moyenne des éléments >0

  Fsi
  Ecrire  VMEP, VMEN, NEN
Fin
```

PROGRAMME

```
      dimension T(150)
       data vmep, vmen, nen, h, j / 5*0 /
       read (*, *)n
       read (9, *) (T(i), i=1, n)
       do 40 i=1, n
       if ( T(i) ) 10, 20, 30
 10    H=H+1
 C  somme des éléments inférieurs à 0
       vmen=vmen + T(i)
       goto 40
 C  nombre d'éléments égaux à 0
 20    nen=nen+1
       goto  40
 30    J=J+1
 C  somme des éléments supérieurs à 0
       vmep = vmep+T(i)
 40    continue
       if (H.eq.0)  goto  70
 C     valeur moyenne des éléments négatifs
      vmen=vmen / H
  70     if (J .eq. 0)  goto  80
  C    valeur moyenne des éléments positifs
        vmep= vmep / J
  80    write (*, 5) vmep, vmen, nen
  5     format (1x,'Valeur de vmep : ', F10.5,
        1/, 1x, 17H Valeur de vmen : , f10.5,
        2/, 1x,'valeur de nen : ', I3)
        stop
        end
```

Organigramme



EXERCICE 34 . MESURE DE TEMPERATURES

Le vecteur T(50) représente 50 valeurs de températures.

Ecrire un programme qui permet de calculer le nombre de températures inférieures à 20 °C et celui des températures supérieures à 40 °C. Nous demandons de mettre les résultats sans les trier, respectivement dans T1(50) et T2(50).

Nous supposons qu'il existe au moins une valeur > 40 °C et une valeur < 20 °C .
Choisir un format convenable pour l'impression .

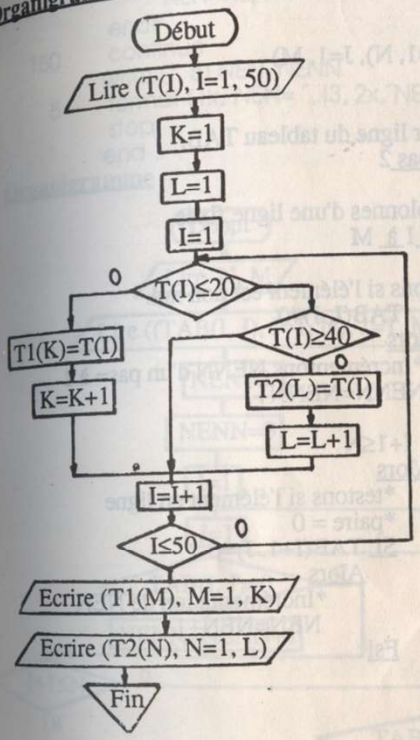
SOLUTION :

```
Algorithme
    Début
      *Lecture des valeurs des éléments de I
      Lire (T(I), I=1, 50)
      K=1  *indice du vecteur T1
      L=1   *indice du vecteur T2
      *I indice du vecteur T
      *Boucle qui traite chaque élément de T.
      Pour I de 1 à 50
        Faire
          Si   T(I)≤20
            Alors
              T1(k)=T(I)  *donne les valeurs ≤ 20  °C
              K=K+1  *donne le nbre de valeurs ≤ 20  °C
            Sinon
              Si   T(I)≥40
                Alors
                  T2(L)=T(I)  * valeur ≥40   °C
                  L=L+1       * nbre de valeurs ≥40   °C
              Fsi
          Fsi
      Fait
  Ecrire (T1(H), H=1, K)
  Ecrire (T2(H), H=1, L)
Fin
```

PROGRAMME

```
      dimension T(50), T1(50), T2(50)
      integer H
      write (2, 1)
 1    format (1x,´vecteur T1´, 12X,´vecteur T2´, / )
      read (5, *)(T(i), i=1, 50)
      K=1
      L=1
      do 40 i=1, 50
        if (T(i) .le. 20) then
           T1 (k)=T(i)
            k=k+1
        else
           if (T(i) .ge. 40) then
             T2(L)=T(i)
              L=L+1
           endif
        endif
40    continue
C  Ecriture des résultats sous le format ci-dessous.
      if (L-K)21, 22, 23
 22   do 15 H=1, K-1
 15   write (2, 9) T1(H), T2(H)
      stop
 21   do 51  H=1, L-1
 51   write (2, 9) T1(H), T2(H)
      do 52 H=L, K-1
 52   write (2, 6) T1(H)
  6   format (4x, F10.5)
      STOP
 23   do 10 H=1, K-1
 10   write (2, 9) T1(H), T2(H)
      do 11 H=K, L-1
 11   write (2, 8) T2(H)
  8   format (18x, F10.5)
  9   format (2(4x, F10.5, 4x))
      stop
      end
```


Organigramme



EXERCICE 35. OPÉRATIONS SUR UN TABLEAU.

Ecrire un programme qui calcule le nombre d'éléments nuls (NEN) se trouvant dans les lignes paires et le nombre d'éléments non nuls (NENN) se trouvant dans les lignes impaires du tableau TAB (N, M) .

Algorithme

Début

Lire N, M

Lire (TAB(I, J), I=1, N), J=1, M)

NEN=0

NENN=0

*Parcours ligne par ligne du tableau TAB.

Pour I de 1 à N pas 2

Faire

*Pour les colonnes d'une ligne fixée

Pour J de 1 à M

Faire

*testons si l'élément est non nul.

Si TAB(I, J)#0

Alors

*Incrémentons NENN d'un pas= à 1

NENN=NENN+1

Fsi

Si I+1≤N

Alors

*testons si l'élément en ligne

*paire = 0

Si TAB(I+1, J)=0

Alors

*Incrémentation de NEN

NEN=NEN+1

Fsi

Fsi

Fait

Fait

Ecrire NEN, NENN

Fin

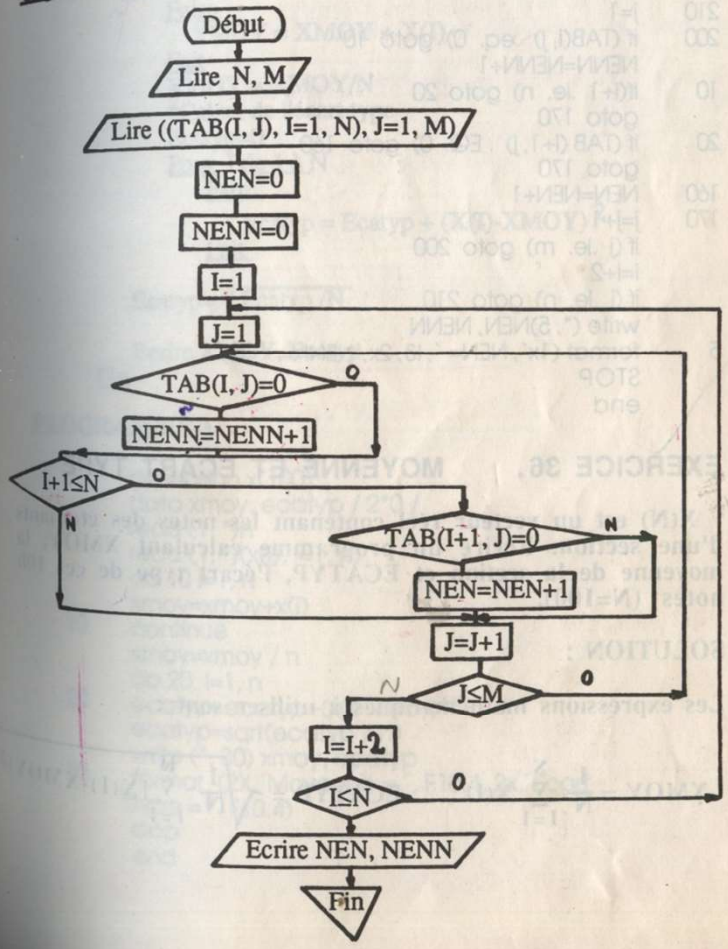
PROGRAMME-1

```
      DIMENSION TAB(30, 40)
      DATA NEN, NENN / 2*0 /
      Read (*, *)n, m
      DO 100 I=1, n
100   read (*, *) (TAB(I, J), J=1, m)
      do 150 i=1, N, 2
      do 150 j=1, M
      if (TAB(i, j) .NE. 0) then
         NENN=NENN+1
      endif
```



```
      if (i+1 .le. n) then
        if (TAB(i+1, j) .EQ. 0) then
           NEN=NEN+1
      endif
150   continue
      write (*, 5) NEN, NENN
5     format (1x,'NEN= ', I3, 2x,'NENN= ', I3)
      stop
      end
```

Organigramme



PROGRAMME-2

```
      DIMENSION TAB(30, 40)
      read (*, *) n, m
      do 100 I=1, n
100   read (*, *) (TAB(I, J), J=1, m)
      NEN=0
      NENN=0
      i=1
210   j=1
200   if (TAB(i, j) .eq. 0) goto 10
      NENN=NENN+1
10    if(i+1 .le. n) goto 20
      goto 170
20    if (TAB (i+1, j) . EQ. 0) goto 160
      goto 170
160   NEN=NEN+1
170   j=j+1
      if (j .le. m) goto 200
      i=i+2
      if (i .le. n) goto 210
      write (*, 5)NEN, NENN
5     format (1x´, NEN= ´, I3, 2x,´NENN= ´, I3)
      STOP
      end
```

EXERCICE 36. MOYENNE ET ECART TYPE

X(N) est un vecteur réel contenant les notes des étudiants d'une section. Ecrire un programme calculant XMOY, la moyenne de la section et ECATYP, l'écart type de ces 100 notes (N=100).

SOLUTION :

Les expressions mathématiques à utiliser sont :

$$XMOY = \frac{1}{N} \sum_{I=1}^{N} X(I) \qquad ECATYP = \sqrt{\frac{1}{N} \sum_{I=1}^{N} [X(I)-XMOY]^2}$$

Algorithme

Début
*Lecture de la dimension et des éléments du vecteur.
Lire N
Lire (X(I), I=1, N)
Xmoy=0 *désigne la moyenne
*Calcul de la moyenne
Pour I de 1 à N
Faire
XMOY = XMOY + X(I)
Fait
XMOY = XMOY/N
*Calcul de l'écart type
Ecatyp=0
Pour I de 1 à N
Faire
Ecatyp = Ecatyp + (X(I)-XMOY) ^{2}
Fait

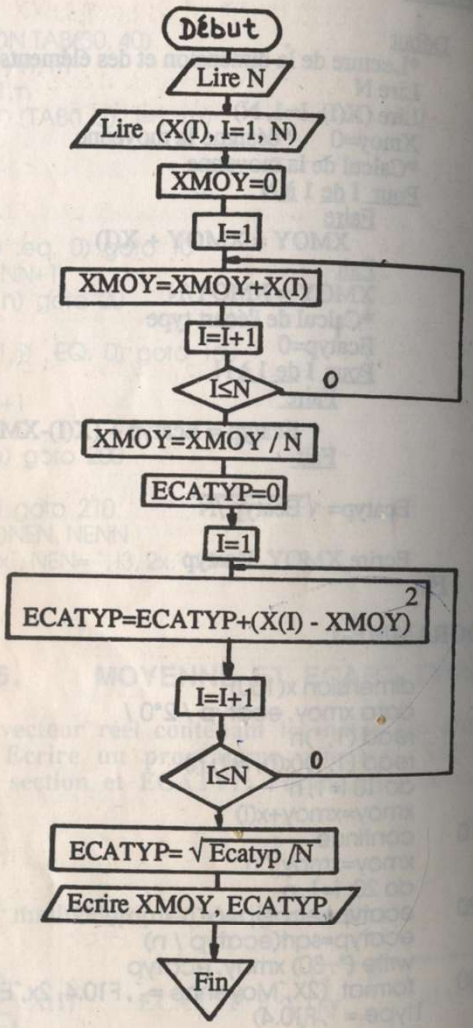
Ecatyp= $\sqrt{Ecatyp/N}$

Ecrire XMOY, Ecatyp
Fin

PROGRAMME-1

```
      dimension x(100)
      data xmoy, ecatyp / 2*0 /
      read (1, *)n
      read (1, *)(x(I), I=1, n)
      do 10 I=1, n
      xmoy=xmoy+x(I)
10    continue
      xmoy=xmoy / n
      do 20 I=1, n
20    ecatyp=ecatyp+(x(I)-xmoy)**2
      ecatyp=sqrt(ecatyp / n)
      write (*, 30) xmoy, ecatyp
30    format (2X,´Moyenne = ´, F10.4, 2x,´Ecart
     1type = ´, F10.4)
      stop
      end
```

Organigramme



EXERCICE 37. CLASSEMENT DES ELEMENTS D'UN VECTEUR : TRI LINEAIRE

Soit un vecteur A (50) . On demande d'écrire un programme qui classe tous les éléments par ordre croissant .

SOLUTION :

Principe : il s'agit de comparer un élément fixé de rang I avec chaqu'un des éléments des rangs qui lui sont supérieurs donc avec le rang J allant de I+1 à 50. Si l'élément A(I) est supérieur à A(J) avec J=I+1, permutons leurs valeurs (c'est à dire que l'élément A(I) prend la place de A(I+1) et vis versa), puis continuons la comparaison de A(I) avec A(I+2) jusqu'à ce que A(I) soit comparé à tout A(J), J allant de I+1 à 50. Sinon terminons la comparaison avec les éléments de rang supérieur.

Ce travail est répété pour I de 1 à 49.

A la fin des tests, nous aurons dans la première case l'élément MIN et dans la dernière l'élément MAX.

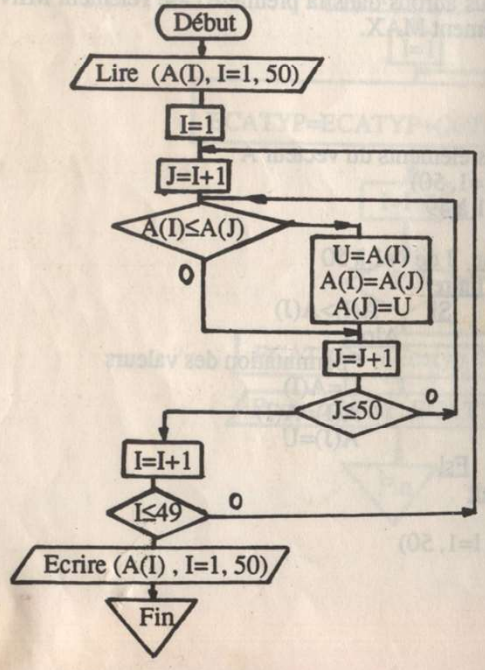
Algorithme

```
Début
   *Lecture des éléments du vecteur A
   Lire (A(I), I=1, 50)
   Pour  I de 1 à 49
      Faire
         Pour  J de I+1 à 50
            Faire
               Si    A(I)>A(J)
                  Alors
                     *permutation des valeurs
                     U=A(I)
                     A(I)=A(J)
                     A(J)=U
               Fsi
            Fait
      Fait
   Ecrire (A(I), I=1, 50)
Fin
```

PROGRAMME-1

```
      dimension a (50)
      read (1, *) (a(i), i=1, 50)
      do 10 i=1, 49
      do 10 j=i+1, 50
       if(a(i) .gt. a(j)) then
C   permutation
        u=a(i)
        a(i)=a(j)
        a(j)=u
       endif
 10  continue
C  impression du vecteur rangé
      write (2, 20) (a(i), i=1, 50)
 20  format (5(1x,f10.4))
      stop
      end
```

Organigramme



PROGRAMME-2

```
      dimension a (50))
      read (1, *) (a(i), i=1, 50)
      do  20  i=1, 49
      N=I+1
      do 20  j=N, 50
      IF (a(i) - a(j)) 20, 20, 30
 30   u=a(i)
      a(i)=a(j)
      a(j)=u
 20   continue
      write (2, 40) (a(i), i=1, 50)
 40    format (2(1x, f10.4))
       stop
       end
```

EXERCICE 38. ELEMENT MAX ET MIN D'UN VECTEUR

Soit un vecteur réel A(40). Ecrire un programme qui cherche l'élément max et l'élément min . Les résultats seront mis respectivement dans RMAX et RMIN .

SOLUTION :

Principe : Initialisons RMAX et RMIN en leur attribuant la première valeur du vecteur (qui est A(I) , I=1).

Si RMAX est inférieure à A(I), sauvegardons la valeur de A(I) dans RMAX, sinon comparons RMIN et A(I) et si RMIN lui est supérieure , alors sauvegardons la valeur de A(I) dans RMIN .

Nous continuons le test avec les autres éléments du vecteur A.

A la fin du test les valeurs max et min du vecteur seront respectivement dans RMAX et RMIN.

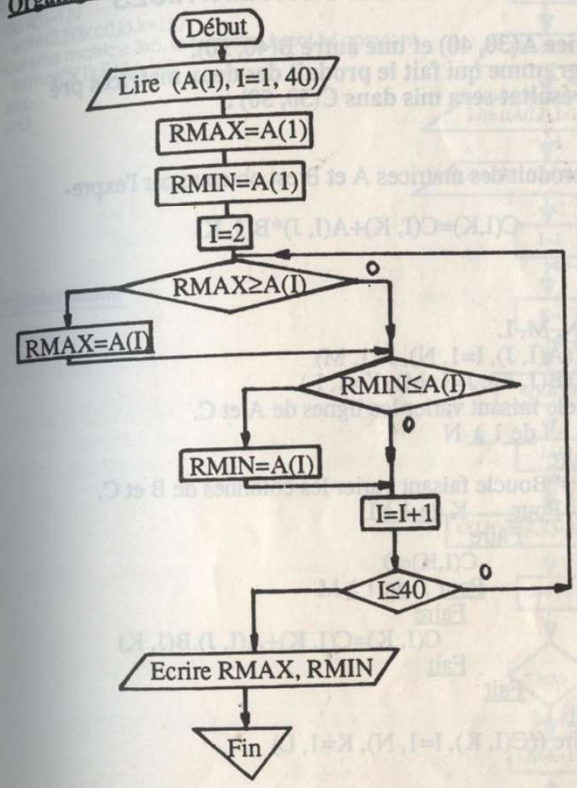
Algorithme

```
Début
    Lire (A(I), I=1, 40)
    *Initialisation de RMAX et RMIN .
    RMAX=A(1)
    RMIN=A(1)
    Pour   I de 2 à 40
        Faire
          Si   RMAX<A(I)
            Alors
              *A(I) valeur supérieure à cell
              RMAX=A(I)
          Sinon
              Si    RMIN≥A(I)
                 Alors
                    *A(I)  valeur inférieure à RMIN.
                    RMIN=A(I)
              Fsi
        Fsi
    Fait
    Ecrire RMAX, RMIN
Fin
```

PROGRAMME

```
      dimension a(40)
      read (1, *)(a(i), i=1, 40)
      rmax=a(1)
      rmin=a(1)
C recherche du MIN et du MAX
      do 35 i=2, 40
      if (rmax .le. a(i)) then
        rmax=a(i)
      else
        if(rmin .ge. a(i)) then
          rmin=a(i)
        endif
        endif
   35   continue
        write(*, 40) rmax, rmin
   40   format (1x, 6Hrmax= , f10.4, 2x,'rmin=', F10.4 )
        stop
        end
```


Organigramme



EXERCICE 39 PRODUIT DE DEUX MATRICES

Soit une matrice A(30, 40) et une autre B(40, 50).
Ecrire un programme qui fait le produit des deux matrices précédentes . Le résultat sera mis dans C(30, 50) .

SOLUTION :
La matrice C, produit des matrices A et B est obtenue par l'expression suivante :

$$C(I,K)=C(I, K)+A(I, J)*B(J, K)$$

Algorithme

Début
Lire N, M, L
Lire ((A(I, J), I=1, N), J=1, M)
Lire ((B(J, K), J=1, M), K=1, L)
*Boucle faisant varier les lignes de A et C.
Pour I de 1 à N
Faire
*Boucle faisant varier les colonnes de B et C.
Pour K de 1 à L
Faire
C(I,K)=0
Pour J de 1 à M
Faire
C(I, K)=C(I, K)+A(I, J).B(J, K)
Fait
Fait
Fait
Ecrire ((C(I, K), I=1, N), K=1, L)
Fin

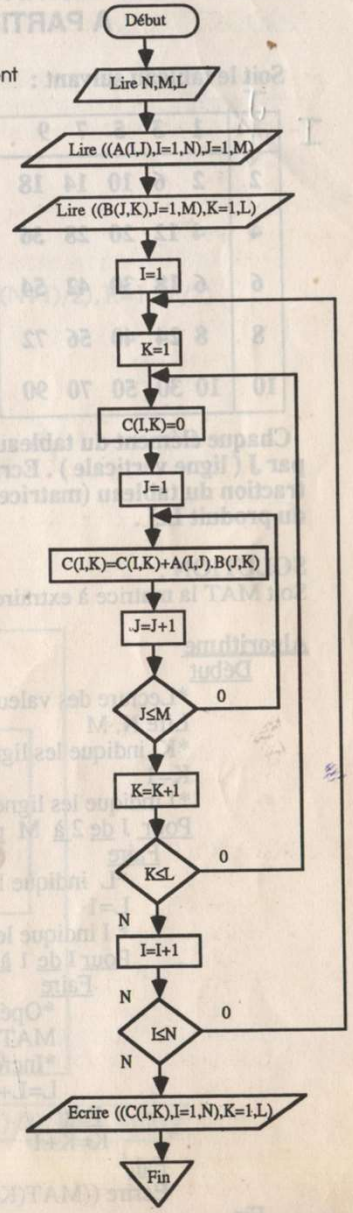
PROGRAMME

```
      dimension a(30, 40), b(40, 50), c(30, 50)
      read (*, *)N, M, L
C  Lecture des éléments des matrices A et B
      do 10 i=1, N
10    read (1, *)(a(i, j), j=1, M)
      do 20 j=1, M
20    read (2, *)(b(j, k), k=1, L)
C  calcul de la matrice produit C
      do 30 i=1, N
      do 30 k=1, L
      c(i, k)=0
      do 30 j=1, M
      c(i, k)=c(i, k)+a(i, j)*b(j, k)
```



```
C  30 continue
C     impression de la matrice produit
      do 40 i=1,N
   40 write(3,50)(c(i,k),k=1,L)
C     pour une matrice 3x5, le format suivant lui convient
   50 format(5(1x,E13.6,1x))
      stop
      end
```

Organigramme



EXERCICE 40 EXTRACTION D'UNE MATRICE A PARTIR D'UN TABLEAU

Soit le tableau suivant :

I \ J	1	3	5	7	9
2	2	6	10	14	18
4	4	12	20	28	36
6	6	18	30	42	54
8	8	24	40	56	72
10	10	30	50	70	90

Chaque élément du tableau est le produit de I (ligne horizontale) par J (ligne verticale) . Ecrire un programme qui permet l'extraction du tableau (matrice) constitué de tous les éléments issus du produit IxJ .

SOLUTION :
Soit MAT la matrice à extraire du tableau .

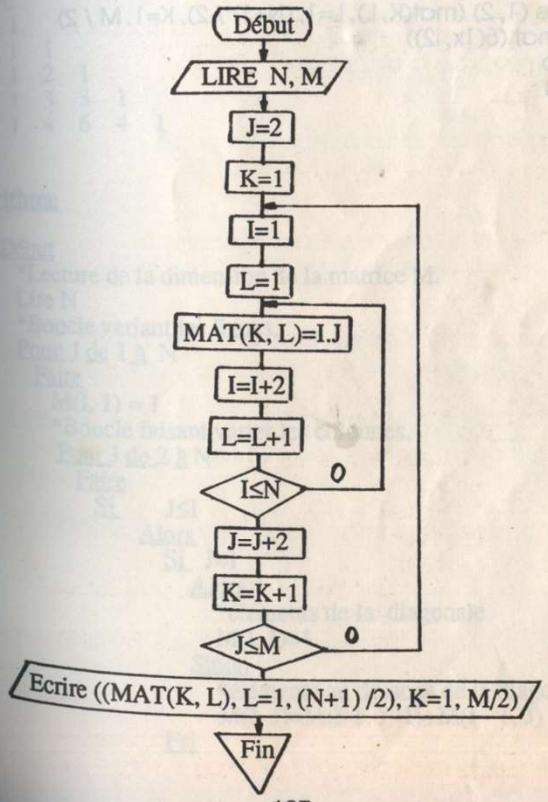
Algorithme

```
Début
    *Lecture des valeurs des dimensions du tableau.
    Lire N, M
    *K indique les lignes de la matrice à trouver MAT.
    K=1
    *J indique les lignes du tableau.
    Pour J de 2 à M pas 2
        Faire
            *L indique les colonnes de la matrice MAT
            L=1
            * I indique les colonnes du tableau
            Pour I de 1 à N pas 2
                Faire
                    *Opération définissant les éléments de MAT
                    MAT (K, L) = I . J
                    *Incrémentation de pas = 1 pour la matrice
                    L=L+1
                Fait
            K=K+1
        Fait
        Ecrire ((MAT(K, L), L=1, (N+1)/2), K=1, M/2)
Fin
```


PROGRAMME-1

```
      dimension mat (5, 5)
C pour établir la matrice recherché,on doit prendre
C N=9 et M=10
      read (*, *)N, M
      k=1
      DO 17 j=2, M, 2
       L=1
        DO 16 i=1, N, 2
         mat (k, L) = i*j
         L=L+1
16     continue
       K=K+1
17     continue
      write (1, 2) ((mat (k, L), L=1, (N+1)/2), K=1, M/2)
2     format (5(1x, i2))
      stop
      end
```

Organigramme



PROGRAMME-2

```
      dimension mat(5, 5)
C  pour établir la matrice recherché,
C  on doit prendre N=9 et M=10.
      read (*, *)N, M
      k=1
30    i=1
      L=1
10    mat (k, L)=i*j
      i=i+2
      L=L+1
      if (i .le. N) goto 10
      j=j+2
      k=k+1
      if (j .le. M) goto 30
      write (1, 2) (mat(K, L), L=1, (N+1) / 2), K=1, M / 2)
2     format (5(1x, i2))
      stop
      end
```


EXERCICE 41 TRIANGLE DE PASCAL

On demande l'écriture d'un programme qui permet la représentation du tiangle de PASCAL .

SOLUTION :

Le triangle de PASCAL s'obtient ainsi :

- Tous les éléments de la première colonne sont égaux à l'unité .
- M(I, J) = 1 pour tout I=J
- Le reste des éléments s'obtient par :

M(I, J)=M(I-1, J-1) + M(I-1, J)

Nous aboutirons au triangle suivant :

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
```

ect...

Algorithme

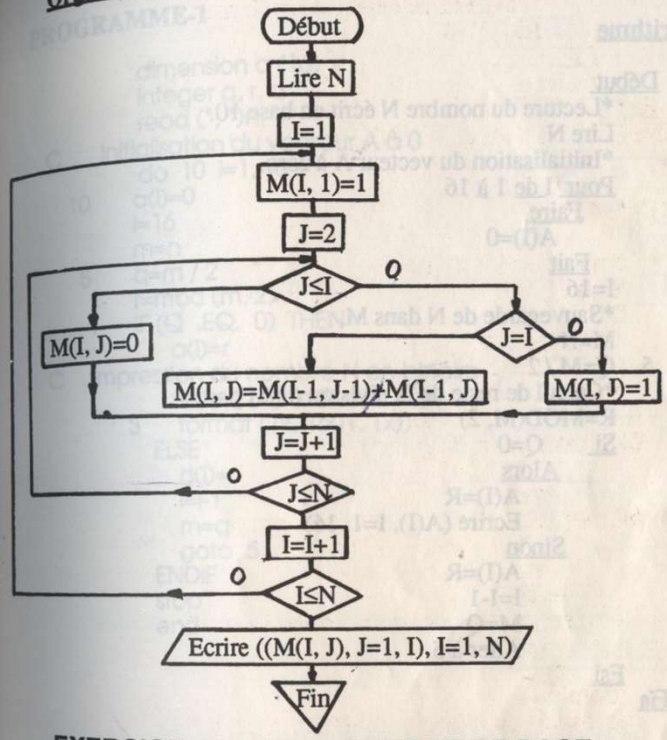
```
Début
  *Lecture de la dimension de la matrice M.
  Lire N
  *Boucle variant les lignes.
  Pour I de 1 à N
    Faire
      M(I, 1) = 1
      *Boucle faisant varier les colonnes.
      Pour J de 2 à N
        Faire
          Si     J≤I
              Alors
                 Si  J=I
                    Alors
                       *éléments de la  diagonale
                       M(I, J)=1
                    Sinon
                       *éléments au dessous de la diagonale.
                       M(I, J)=M(I-1, J -1)+M(I - 1, J)
                 Fsi
```

```
        Sinon
            *éléments au dessus de la diagonale.
            M(I, J)=0
        Fsi
      Fait
    Fait
Ecrire ((M(I, J), J=1, I), I=1, N)
Fin
```

PROGRAMME-1

```
      dimension M(10, 10)
      read (*, *)N
      DO 24 I=1, N
        M(I, 1)=1
        DO 24 J=2, N
          if(J .LE. I) then
            if(j .eq. i) then
C     éléments de la diagonale
              M(i, j)=1
            else
C     éléments au dessous de la diagonale
              M(i, j)=M(i-1, j-1)+M(i-1, j)
            endif
          else
C     éléments au dessus de la diagonale
            M(I, J)=0
          endif
   24   continue
C   impression du triangle de PASCAL pour N=8
        do 30 i=1, n
   30   write (1, 35) (M(i, j), j=1, i)
C   ce format convient pour n=8
   35   format (8(2x, i3))
        stop
        end
```


Organigramme



EXERCICE 42. CHANGEMENT DE BASE

Ecrire un programme qui permet de transformer un nombre N d'une base 10 à une base binaire (base 2). On demande la représentation sur 16 cases (2 octets).

Soit A (16) la représentation binaire de 16 cases.

SOLUTION :

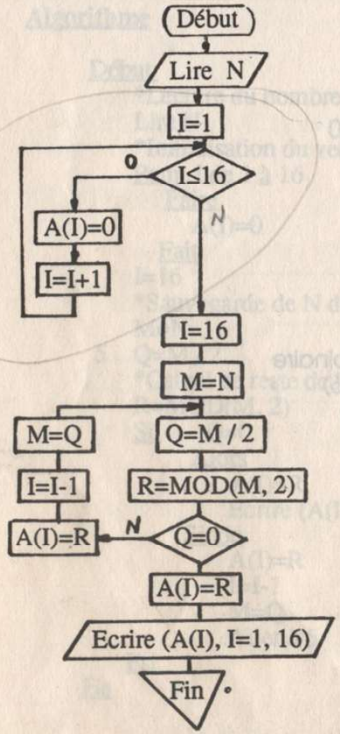
Algorithme

```
Début
    *Lecture du nombre N écrit en base 10.
    Lire N
    *Initialisation du vecteur A à zéro.
    Pour I de 1 à 16
        Faire
            A(I)=0
        Fait
    I=16
    *Sauvegarde de N dans M.
    M=N
5   Q=M / 2
    *Calcul de reste de la division de M par 2.
    R=MOD(M, 2)
    Si      Q=0
        Alors
            A(I)=R
            Ecrire (A(I), I=1, 16)
        Sinon
            A(I)=R
            I=I-1
            M=Q
            Aller à 5
    Fsi
Fin
```


PROGRAMME-1

```
      dimension a (16)
      integer q, r, a
      read (*, *)n
C     initialisation du vecteur A à 0
      do 10 i=1, 16
10    a(i)=0
      i=16
      m=n
5     q=m / 2
      r=mod (m, 2)
      IF (Q .EQ. 0) THEN
        a(i)=r
C     impression du nombre N en binaire
        write (*, 3) (a(i), i=1, 16)
3       format (1x, 16(I1, 1x))
      ELSE
        a(i)=r
        i=i-1
        m=q
        goto 5
      ENDIF
      stop
      end
```

Organigramme



PROGRAMME-2

```
      dimension a (16)
      integer q, r, a
      read (*, *)n
C     intialisation du vecteur A à 0
      do 10 i=1, 16
 10   a (i)=0
      i=16
      m=n
 25   q=m / 2
      r=mod (m, 2)
      if (q. EQ. 0) goto 20
      a (i)=r
      i=i-1
      m=q
      goto 25
 20   a (i)=r
C     impression du nombre N en binaire
      write (*, 3) (a (i), i=1, 16)
    3 format (1x, 16 (1, 1x))
      stop
      end
```

EXERCICE 43. RECHERCHE DE L' ÉLÉMENT MAX DANS CHACUN DES COLONNES D' UNE MATRICE.

Ecrire un programme qui permet la recherche de l'élément max en valeur absolue dans chaqu'une des colonnes d'une matrice A (N, N+1) . Positionner l'élément en indiquant la ligne et la colonne .

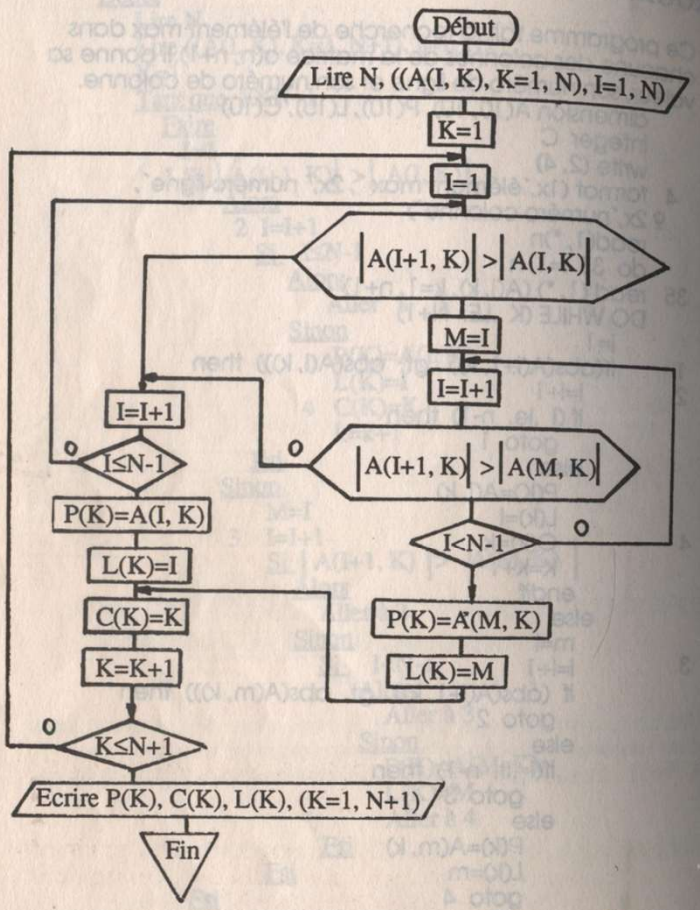
SOLUTION :

```
Algorithme
    Début
        Lire N
        Lire ((A(I, K), K=1, N+1), I=1, N)
        K=1
        Tant que K≤N+1
          Faire
            I=1
          1 Si | A(I+1, K)| >| A(I, K)|
                Alors
                  2 I=I+1
                    Si  I≤N-1
                      Alors
                        Aller à 1
                      Sinon
                        P(K)=A(I, K)
                        L(K)=I
                      4 C(K)=K
                        k=k+1
                    Fsi
                Sinon
                    M=I
                  3 I=I+1
                    Si | A(I+1, K) |> |A(M, K) |
                      Alors
                        Aller à 2
                      Sinon
                        Si  I<N-1
                          Alors
                            Aller à 3
                          Sinon
                            P(K)=A(M, K)
                            L(K)=M
                            Aller à 4
                        Fsi
                    Fsi
            Fsi
        Fait
Ecrire P(K), C(K), L(K), (K=1, N+1)
Fin
```


PROGRAMME 1

```
C  Ce programme fait la recherche de l'élément max dans
C  chacune des colonnes de la matrice a(n, n+1).Il donne sa
C  valeur, son numéro de ligne et son numéro de colonne.
       dimension A(10, 10), P(10), L(10), C(10)
       integer C
       write (2, 4)
    4  format (1x,'élément max ', 2x,' numéro ligne',
    9 2x,'numéro colonne')
       read(1, *)n
       do 35 i=1, n
   35  read (1, *) (A(i, k), k=1, n+1)
       DO WHILE (K .LE. N+1)
          i=1
    1     if(abs(A(i+1, k)) .gt. abs(A(i, k))) then
    2       i=i+1
            if (i .le. n-1) then
              goto 1
            else
              P(K)=A(i, k)
              L(k)=i
    4         C(k)=k
              K=K+1
            endif
          else
            m=i
    3       i=i+1
            if (abs(A(i+1, k)) .gt. abs(A(m, k))) then
              goto 2
            else
              if(i .lt. n-1) then
                 goto 3
              else
                 P(k)=A(m, k)
                 L(k)=m
                 goto 4
              endif
          endif
       endif
        enddo
       DO 19 K=1, N+1
   19  write (2, *) P(k), L(k), C(k)
       end
```

Organigramme



PROGRAMME-2

```
      dimension A(10, 10), P(10), L(10), C(10)
      integer C
      write (2, 4)
    4 format (11x,´élément max ´,2x,´numéro ligne´,
     96x,´numéro colonne´)
      read (1, *)n
      do 35 i=1, n
   35 read (1, *) (A(i, k), k=1, n+1)
      do 5 k=1, n+1
      i=1
   15 if (abs(A(i+1, k)).gt.abs(A(i, k))) goto 10
      m=i
   20 i=i+1
      if (abs(A(i+1, k)).gt.abs(A(m, k))) goto 10
      if(i .lt. n-1) goto 20
      P(k)=A(m, k
      L(k)=m
      goto 6
   10 i=i+1
      if(i .le. n-1) goto 15
      P(k)=A(I, K)
      L(k)=i
    6 C(k)=k
    5 continue
      do 11 k=1, N+1
   11 write(2, *) P(k), L(k), C(k)
      stop
      end
```

EXERCICE 44. TRIANGULARISATION SUPERIEURE D' UNE MATRICE.

Ecrire un programme qui fait la triangularisation supérieure d'une matrice en utilisant l'algorithme de GAUSS. On supposera que les éléments de la diagonale sont différents de zéro.

SOLUTION :

L'algorithme de résolution est le suivant :

$$A(I,J) = A(I,J) - A(I,K)/A(K,K)*A(K,J) \quad \left| \begin{array}{l} K=1, N-1 \\ I=K+1, N \\ J=K+1, N+1 \end{array} \right.$$

$A(I, L) = 0$ pour $L=1, K$

Cet algorithme est applicable à une matrice A(N, N+1) dont la colonne (n+1) désigne le vecteur B(I) solution du système :

$$A(I, J) * X(I) = B(I)$$

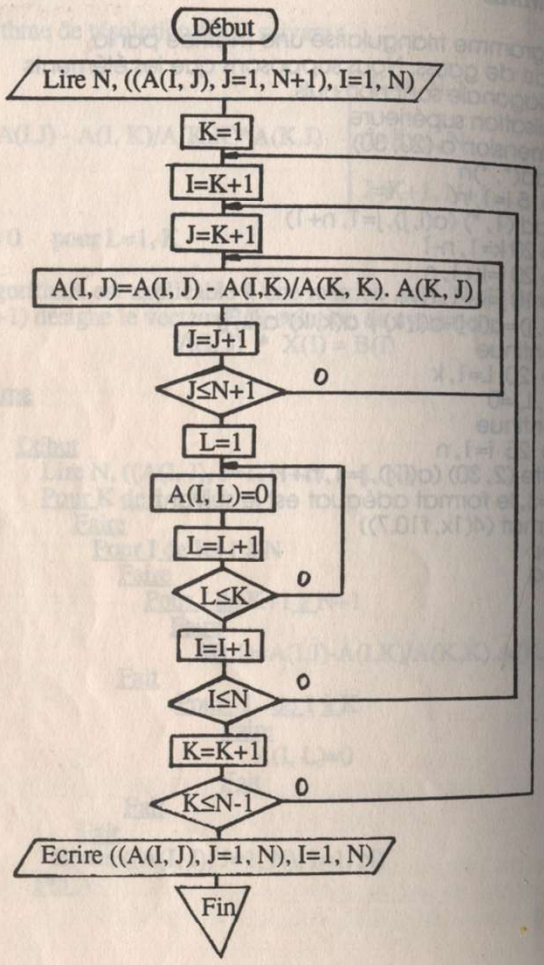
Algorithme

```
Début
  Lire N, ((A(I, J), J=1, N+1), I=1, N)
  Pour K de 1 à N-1
    Faire
      Pour I de K+1 à N
        Faire
          Pour J de K+1 à N+1
            Faire
              A(I,J)=A(I,J)-A(I,K)/A(K,K).A(K,J)
        Fait
            Pour  L  de 1 à K
              Faire
                A(I, L)=0
              Fait
      Fait
    Fait
  Ecrire ((A(I, J), J=1, N), I=1, N)
Fin
```


PROGRAMME

```
C Ce programme triangularise une matrice par la
C méthode de gauss. Nous supposons que les éléments
C de la diagonale sont non nuls.
C Triangulisation supérieure
      dimension a (20, 30)
      read(*, *)n
      do 5 i=1, n
    5 read (1, *) (a(i, j), j=1, n+1)
      do 20 k=1, n-1
      do 20 i=k+1, n
      do 10 j=k+1, n+1
      a(i, j)=a(i, j)-a(i, k) / a(k, k)*a(k, j)
   10 continue
      do 20 L=1, k
      a(i, L)=0
   20 continue
      do 25 i=1, n
   25 write (2, 30) (a(i, j), j=1, n+1)
C pour n=3, le format adéquat est le suivant.
   30 format (4(1x, f10.7))
      stop
      end
```

Organigramme



EXERCICE 45.RESOLUTION D'UN SYSTEME D'EQUATIONS SOUS FORME D'UNE MATRICE TRIANGULAIRE SUPERIEURE PAR L'ALGORITHME DE CRAMER

Soit le système suivant à N équations :

$$A(I, J) .X(I) = B(I)$$

La matrice A(I, J) est une matrice triangulaire supérieure . Nous demandons l'écriture d'un programme résolvant ce système en s'appuyant sur l'algorithme de CRAMER .

SOLUTION:

L'algorithme est le suivant :

$$X(I)=1/A(I, I).[\ B(I) - \sum_{J=I+1}^{N} A(I, J) . X(J)\ ],\ I = N, N-1,...., 1$$

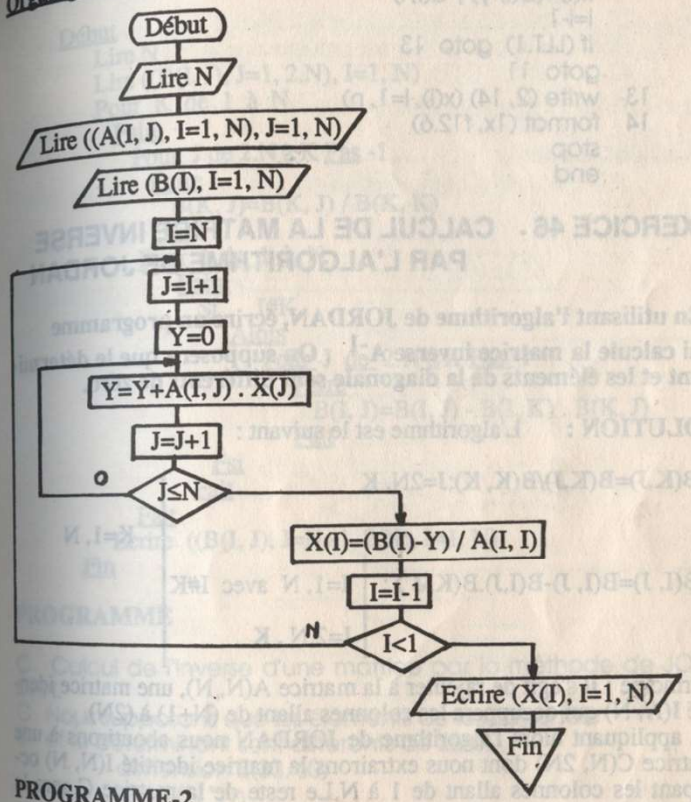
Algorithme

```
Début
   Lire N
   Lire ((A(I, J), I=1, N), J=1, N)
   Lire (B(I), I=1,, N)
   Pour I de N à 1 pas -1
      Faire
         Y=0                                   N
          *Calcul de la somme  Y = Σ A(I, J) . X(J)
                                  J=I+1
         Pour J de I+1 à N
            Faire
               Y=Y+A(I, J) . X(J)
            Fait
          *Calcul de X(I).
          X(I)=(B(I)-Y) / A(I, I)
      Fait
    Ecrire (X(I), I=1, N)
Fin
```

PROGRAMME-1

```
C  Méthode sans pivotation.
C  Résolution d'un système d'équations par la méthode
C  de cramer.
C  Le système est de la forme : a(i, j)*x(i)=b(i).
C  nous suposons que la matrice A est triangulaire supérieure
C et que les éléments de la diagonale sont non nuls.
       dimension a(10, 10), x(10), b(10)
       read(1, *)n
       do 10 i=1, n
   10  read (1, *) (a(i, j), j=1, n)
       read (1, *) (b(i), i=1, n)
       DO 100 i=N, 1, -1
            y=0
            DO  90  J=i+1, N
                 y=y+a(i, j)*x(j)
   90       continue
                 x(i)=(b(i)-y) / a (i, i)
  100       continue
            write (2, 14) (x(i), i=1, n)
   14       format (1x, f12. 6)
            end
```


Organigramme



PROGRAMME-2

```
       dimension a(10, 10), x(10), b(10)
       read (1, *)n
       do 10  i=1, n
   10  read (1, *) (A(i, j), j=1, n)
       Read (1, *)(b(i), i=1, n)
       i=n
   11  j=i+1
       y=0
   15  y=y+a(i, j)*x(j)
       j=j+1
```

```
12  if (j.le.n) goto 15
    x(i)=(b(i)-y) / a(i, i)
    i=i-1
    if (i.LT.1) goto 13
    goto 11
13  write (2, 14) (x(i), i=1, n)
14  format (1x, f12.6)
    stop
    end
```

EXERCICE 46 . CALCUL DE LA MATRICE INVERSE PAR L'ALGORITHME DE JORDAN

En utilisant l'algorithme de JORDAN, écrire un programme qui calcule la matrice inverse A^{-1} . On supposera que le déterminant et les éléments de la diagonale sont différents de zéro.

SOLUTION : L'algorithme est le suivant :

B(K,J)=B(K,J)/B(K, K);J=2N, K		K=1, N
B(I, J)=B(I, J)-B(I,J).B(K,J) ;	I=1, N avec I#K J=2.N , K	

Principe : Il s'agit de rajouter à la matrice A(N, N), une matrice identité I(N, N) qui occuppera les colonnes allant de (N+1) à (2N).
En appliquant alors l'algorithme de JORDAN nous aboutirons à une matrice C(N, 2N) dont nous extrairons la matrice identité I(N, N) occupant les colonnes allant de 1 à N.Le reste de la matrice C sera la matrice inverse recherchée.

Pour aboutir à ce résultat, nous appliquons les deux étapes de l'algorithme :

La première étape fait la normalisation : nous fixons K à 1 et nous varions J de 2.N à K en le decrémentant.

La deuxième étape fait la réduction : Nous maintenons la valeur de K pour laquelle nous avons fait la normalisation, nous varions I de 1 à N sans prendre la valeur de I=K et nous faisons varier J de 2.N à K pour chaque valeur de I .

Nous refaisons les deux étapes pour une autre valeur de K, etc..

Algorithme

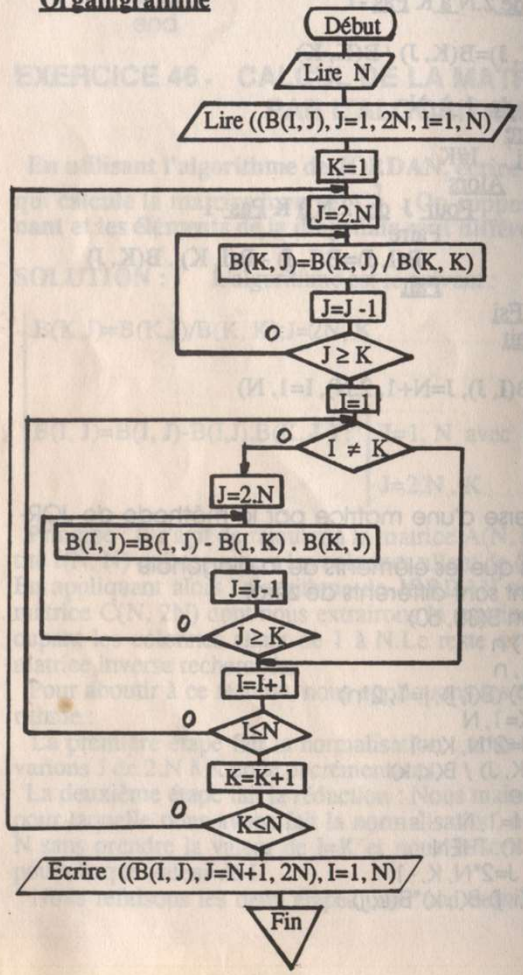
```
Début
   Lire N
   Lire ((B(I, J), J=1, 2.N), I=1, N)
   Pour K de 1 à N
     Faire
       Pour J de 2.N à K Pas -1
         Faire
           B(K, J)=B(K, J) / B(K, K)
         Fait
       Pour I de 1 à N
         Faire
           Si  I#K
             Alors
               Pour J de 2.N à K Pas -1
                 Faire
                   B(I, J)=B(I, J) - B(I, K) . B(K, J)
                 Fait
           Fsi
         Fait
     Fait
   Ecrire ((B(I, J), J=N+1, 2.N), I=1, N)
Fin
```

PROGRAMME

```
C  Calcul de l'inverse d'une matrice par la méthode de JOR-
DAN.
C  Nous supposons que les éléments de la diagonale
C  et le déterminant sont différents de zéro.
       dimension B(30, 60)
       read (*, *) n
       do 5 i=1, n
5      read (1, *) (B(i, j), j=1, 2*n)
       DO 30 K=1, N
       DO 10 J=2*N, K, -1
       B(k, j)=B(K, J) / B(k, k)
10     continue
       DO 30 I=1, N
       IF(I .NE. K) THEN
       DO 15 J=2*N, K, -1
15     B(i, j)=B(i, j)-B(i, k)*B(k, j)
       endif
```

```
 30     continue
        do 35 i=1, n
 35     write (2, 40) (B(i, j), j=n+1, 2*n)
C Pour  N=3, ce format est adéquat.
 40     format (3(1x, f12.4))
        stop
        end
```

Organigramme



EXERCICE 47. OPÉRATIONS SUR UN VECTEUR.

Soit un vecteur X(100) ne contenant que des valeurs entières comprises entre 1 et 20 .

Calculer le nombre de fois qu'apparait chaque valeur du vecteur X .

Donner la valeur qui a la plus grande apparition dans X .

SOLUTION :

Algorithme

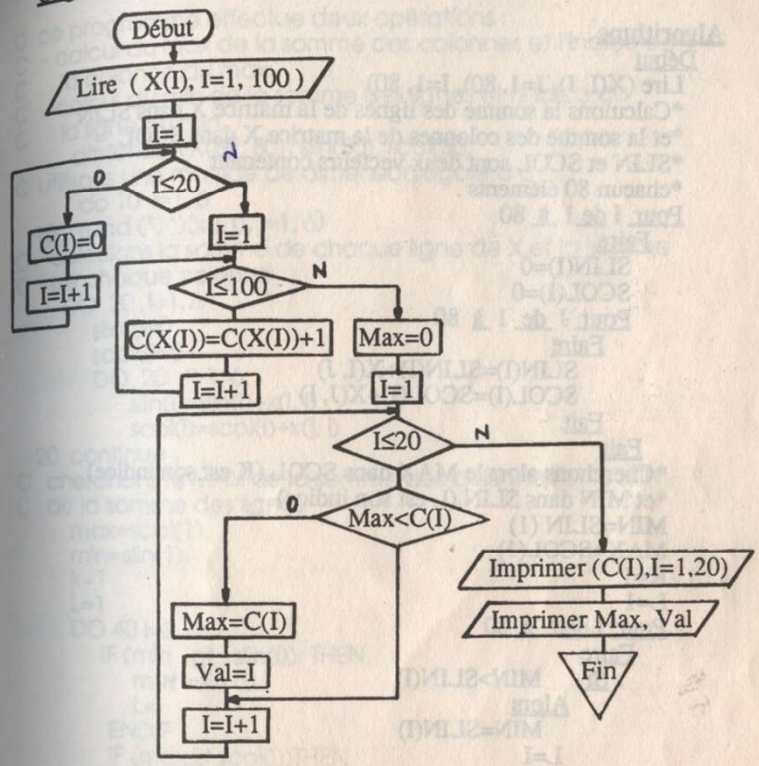
```
Début
    Lire (X(I), I=1, 100 )
    *Nous utiliserons un vecteur C (20) qui donnera le
    *nombre d'apparitions de chaque valeur de X .
    Pour I de 1 à 20
      Faire
        *Initialisons chaque élément de C à zéro.
        C(I)=0
      Fait
    *Cherchons le nombre de fois qu'apparait
    *chaque valeur de X.
    Pour I de 1 à 100
      Faire
        C(X(I))=C(X(I))+1
      Fait
    *Cherchons la valeur qui a la plus grande apparition
    *dans X que nous sauvegardons dans MAX .
    Max=0
    Pour I de 1 à 20
      Faire
        Si Max<C(I)
          Alors
            Max=C(I)
            *Indice de l'élément dans le vecteur C
            Val=I
          Fsi
      Fait
```

Ecrire (C(I), I=1, 20)
Ecrire Max, Val
Fin

PROGRAMME

```
      dimension x(100), c(20)
      integer x
C     prenons 30 valeurs du vecteur X
      read (1, *) (x(i), i=1, 30)
C     initialisation du vecteur C à 0
      do 10 i=1, 20
 10   c(i)=0
C     recherche du nombre de fois qu'apparait chaque
C     valeur de X
      do 20 i=1, 30
 20   c(x(i))=c(x(i))+1
C     recherche de la valeur qui a la plus grande apparition
      max=0
      do 40 i=1, 20
        if(max .lt. c(i)) then
          max=c(i)
          Val=i
        endif
 40   continue
      do 50 i=1, 20
 50   write (2, *)c(i)
      write (2, *)max, val
      end
```


Organigramme



EXERCICE 48 : OPERATIONS SUR UNE MATRICE

Soit une matrice X(80, 80) .

Chercher

- Le maximum de la somme des colonnes (SCOL)
- Le maximum de la somme des lignes (SLIN)

et trouver l'indice :

- de la colonne représentant le max
- de la ligne représentant le min.

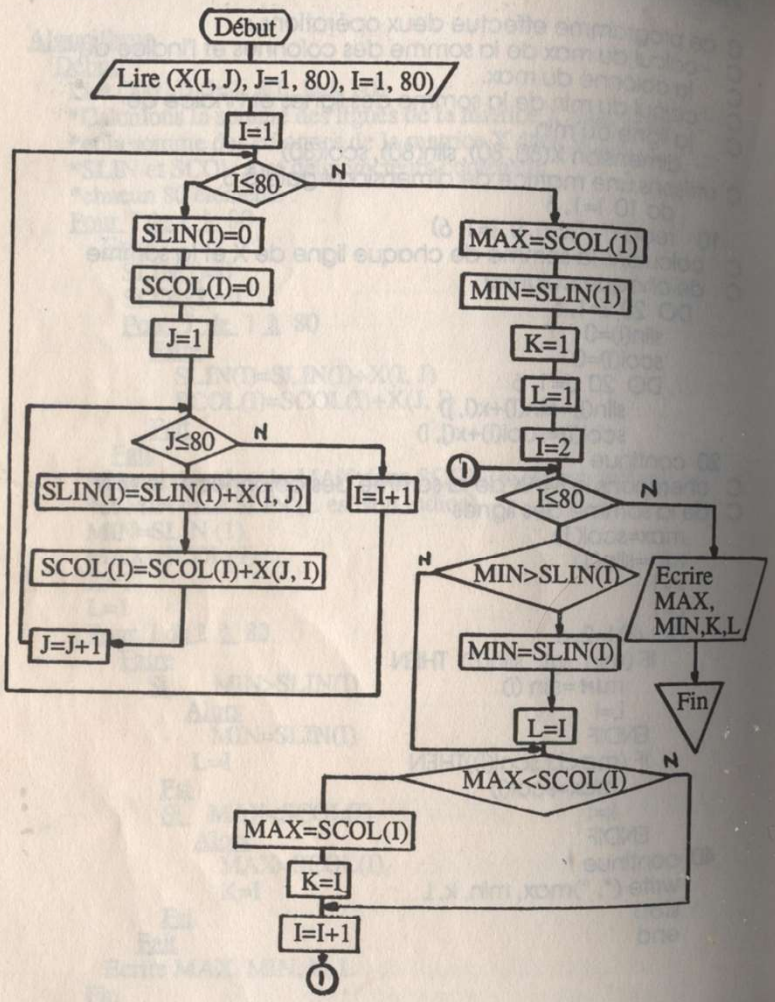
SOLUTION :

```
Algorithme
  Début
   Lire (X(I, J), J=1, 80), I=1, 80)
   *Calculons la somme des lignes de la matrice X dans SLIN
   *et la somme des colonnes de la matrice X dans SCOL .
   *SLIN et SCOL sont deux vecteurs contenant
   *chacun 80 éléments .
   Pour I de 1 à 80
      Faire
        SLIN(I)=0
        SCOL(I)=0
        Pour J de 1 à 80
           Faire
             SLIN(I)=SLIN(I)+X(I, J)
             SCOL(I)=SCOL(I)+X(J, I)
           Fait
      Fait
   *Cherchons alors le MAX dans SCOL (K est son indice)
   *et MIN dans SLIN (L est son indice).
   MIN=SLIN (1)
   MAX=SCOL(1)
   K=1
   L=1
   Pour I de 2 à 80
      Faire
        Si    MIN>SLIN(I)
           Alors
             MIN=SLIN(I)
           L=I
        Fsi
        Si  MAX<SCOL(I)
           Alors
             MAX=SCOL(I)
             K=I
        Fsi
      Fait
   Ecrire MAX, MIN, K, L
  Fin
```


PROGRAMME

```
C ce programme effectue deux opérations :
C - calcul du max de la somme des colonnes et l'indice de
C   la colonne du max.
C - calcul du min de la somme des lignes et l'indice de
C   la ligne du min.
      dimension X(80, 80), slin(80), scol(80)
C utilisons une matrice de dimension égale à 6
      do 10 i=1, 6
   10 read (*, *)(x(i, j), j=1, 6)
C  calculons la somme de chaque ligne de X et la somme
C  de chaque colonne
      DO 20 I=1, 6
        slin(i)=0
        scol(i)=0
        DO 20 J=1, 6
           slin(i)=slin(i)+x(i, j)
           scol(i)=scol(i)+x(j, i)
   20 continue
C  cherchons le MAX de la somme des colonnes et le MIN
C  de la somme des lignes
      max=scol(1)
      min=slin(1)
      k=1
      L=1
      DO 40 I=2, 6
        IF (min .gt. slin(i)) THEN
           min=slin (i) .
           L=i
        ENDIF
        IF (max.Lt.scol(i))THEN
           max=scol(i)
           k=i
        ENDIF
   40 continue
      write (*, *)max, min, k, L
      stop
      end
```

Organigramme



EXERCICE 49. CONSTRUCTION D'UNE MATRICE A PARTIR DES ELEMENTS D'UNE AUTRE

Construire une matrice A à partir d'une matrice T dont les éléments seront classés par ordre croissant suivant les lignes (ie : le dernier élément de chaque ligne est inférieur au premier de la ligne suivante) .
L'analyse du traitement doit être faite comme suit :

- rendre la matrice T en vecteur V .
- trier le vecteur V .
- remettre le vecteur V en matrice A .

SOLUTION :

```
Algorithme
    Début
        Lire m, n
        Lire ( T(I, J), J=1, M), I=1, N)
        *Transformation de la matrice T en vecteur V.
        K=0
        Pour I de 1 à N
            Faire
                Pour J de 1 à M
                    Faire
                        K=K+1
                        V(K)=T(I, J)
                    Fait
            Fait
        *Tri du vecteur V par la méthode du tri linéaire .
        Pour K de 1 à N*M-1
            Faire
                Pour J de K+1 à N*M
                    Faire
                        Si V(K)>V(J)
                            Alors
                                SAUV=V(K)
                                V(K)=V(J)
```

```
                V(J)=SAUV
              Fsi
            Fait
          Fait
        *Remise du vecteur V en matrice A .
        K=0
        Pour I de 1 à N
          Faire
            Pour J de 1 à M
              Faire
                K=K+1
                A(I, J)=V(K)
              Fait
          Fait
        Ecrire (A(I, J), J=1, M), I, N)
        Fin
```

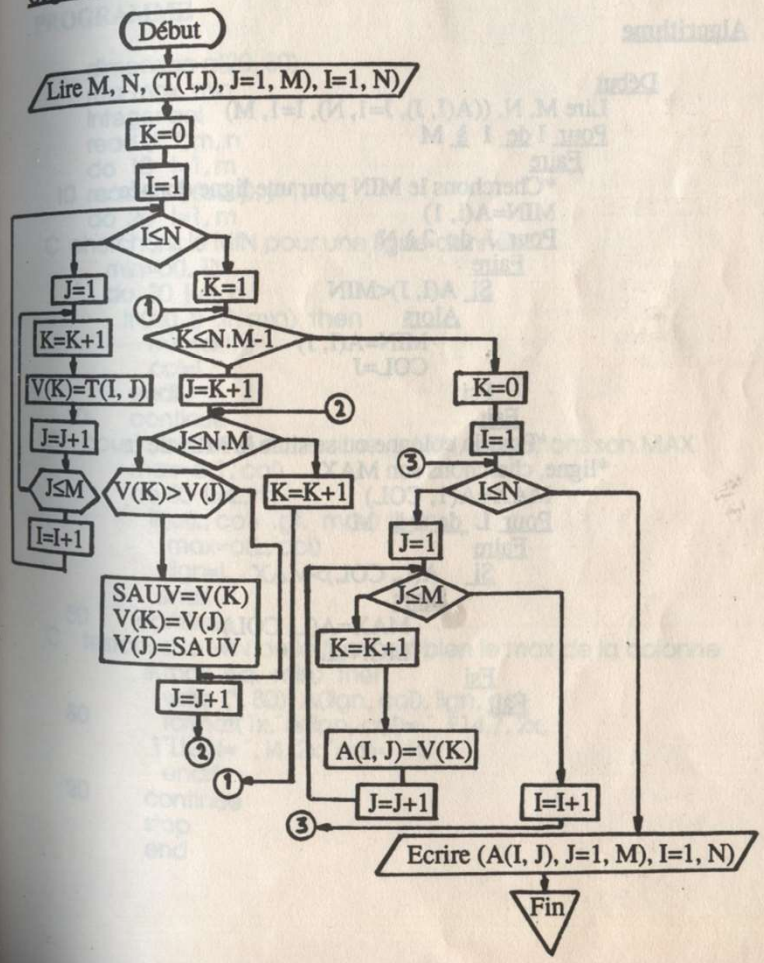
PROGRAMME

```
      dimension t (10, 20), a (10, 20), v (200)
      read (*, *)n, m
      do 10 i=1, n
10    read (*, *)(t(i, j), j=1, m)
C transformation de la matrice T en vecteur V .
      k=0
      do 20 i=1, n
        do 20 j=1, m
          k=k+1
          v(k)=t(i, j)
20    continue
C opérations de classement des éléments .
        do 50 k=1, n*m-1
          do 50 j=k+1, n*m
            if (v(k) .gt. v(j))  then
              sauv=v(k)
              v(k)=v(j)
              v(j)=sauv
            endif
50    continue
C transformation du vecteur V en matrice A .
      k=0
      do 70 i=1, n
        do 70 j=1, m
          k=k+1
          a(i, j)=v(k)
```



```
70    continue
C impression de la matrice A.
      do 90 i=1, n
90    write (*, 100) (a(i, j), j=1, m)
100   format (20(1x, f14.6))
      stop
      end
```

Organigramme



EXERCICE 50. RECHERCHE DU POINT DE SELLE

Nous appellons point de selle, l'élément de la matrice A qui est maximal dans sa colonne et minimal dans sa ligne.

Ecrire un programme qui fait la recherche du point de selle, en indiquant sa position dans la matrice A.

SOLUTION :

Algorithme

```
Début
    Lire M, N, ((A(I, J), J=1, N), I=1, M)
    Pour I de 1 à M
      Faire
        *Cherchons le MIN pour une ligne donnée .
        MIN=A(i, 1)
        Pour J de 2 à N
          Faire
            Si A(I, J)<MIN
              Alors
                MIN=A(I, J)
                COL=J
            Fsi
          Fait
        *Pour la colonne ou se situe le MIN de la
    *ligne, cherchons son MAX.
        MAX=A(1, COL)
        Pour L de 2 à M
          Faire
            Si  A(L, COL)>MAX
              Alors
                MAX=A(L, COL)
                LIGN=L
            Fsi
          Fait
```

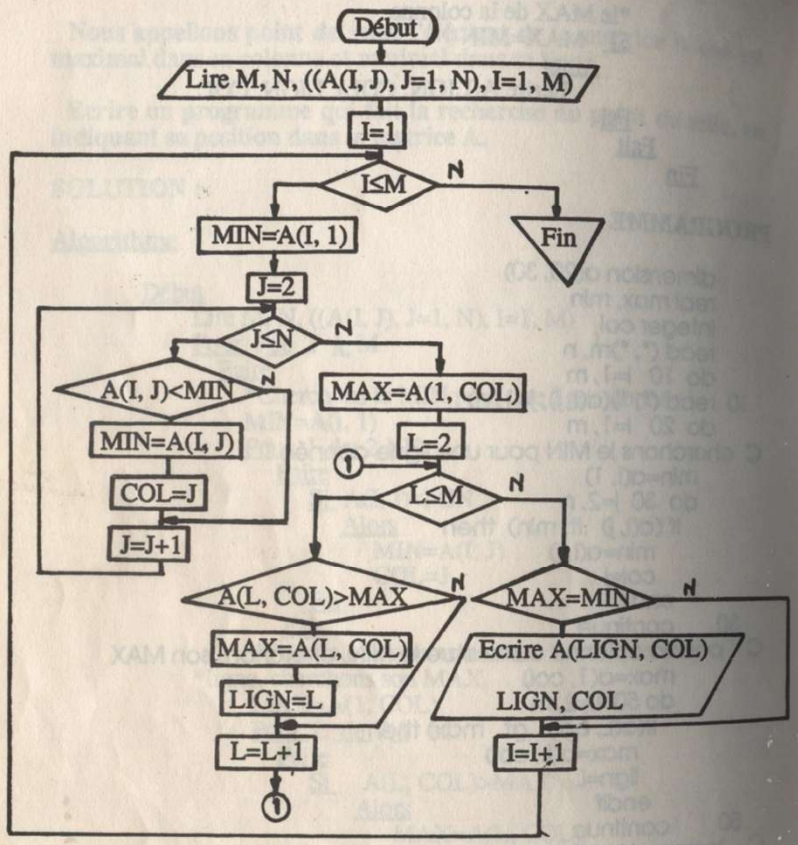


```
        *Testons si le MIN de la ligne est bien
        *le MAX de la colonne.
        Si  MAX=MIN
          Alors
            Ecrire A(LIGN, COL), LIGN, COL
        Fsi
      Fait
  Fin
```

PROGRAMME

```
      dimension a(20, 30)
      real max, min
      integer col
      read (*, *)m, n
      do  10  I=1, m
   10 read (*, *)(a(I, J), J=1, n)
      do  20  I=1, m
C  cherchons le MIN pour une ligne donnée
       min=a(I, 1)
       do  30 J=2, n
        If (a(I, J)  .lt. min)  then
          min=a(I, J)
          col=j
        endif
   30   continue
C    pour la colonne ou se situe le MIN, cherchons son MAX
        max=a(1, col)
        do 50  L=2, m
          If(a(L, col)  .gt.  max) then
            max=a(L, col)
            lign=L
          endif
   50   continue
C    testons si le MIN de la ligne est bien le max de la colonne
        If(max .eq.  min)  then
          write (*, 80)  A(lign, col), lign, col
   80     format(1x,'a(lign, col)= ', F14.7, 2x,
         1'LIGN= ', I4, 2x,'col=', I4)
          endif
   20   continue
        stop
        end
```

Organigramme



EXERCICE 51 CALCUL DES ELEMENTS D'UNE MATRICE

On lit une constante B entière. On calcule la matrice M(N, N) de la façon suivante :

$$\begin{bmatrix} B & B & B \ldots & B & B & B \\ B & B^2 & B^2 \ldots & B^2 & B^2 & B \\ B & B^2 & B^3 \ldots & B^3 & B^2 & B \\ \vdots & & & & & \\ B & B^2 & B^3 \ldots & B^3 & B^2 & B \\ B & B^2 & B^2 \ldots & B^2 & B^2 & B \\ B & B & B \ldots & B & B & B \end{bmatrix}$$

Ecrire un programme qui calcule cette matrice .

SOLUTION :

Algorithme

Début
*Lecture de l'entier B et de la dimension de la matrice.
Lire B; N
*La variable A est utilisée pour le calcul de la valeur à
*stocker dans un rang donné de la matrice M.
A=1
Pour I de 1 à N/2+1
Faire
*Calcul de la valeur à stocker.
A=A*B
*Stockage de A dans le rang adéquat de la matrice M

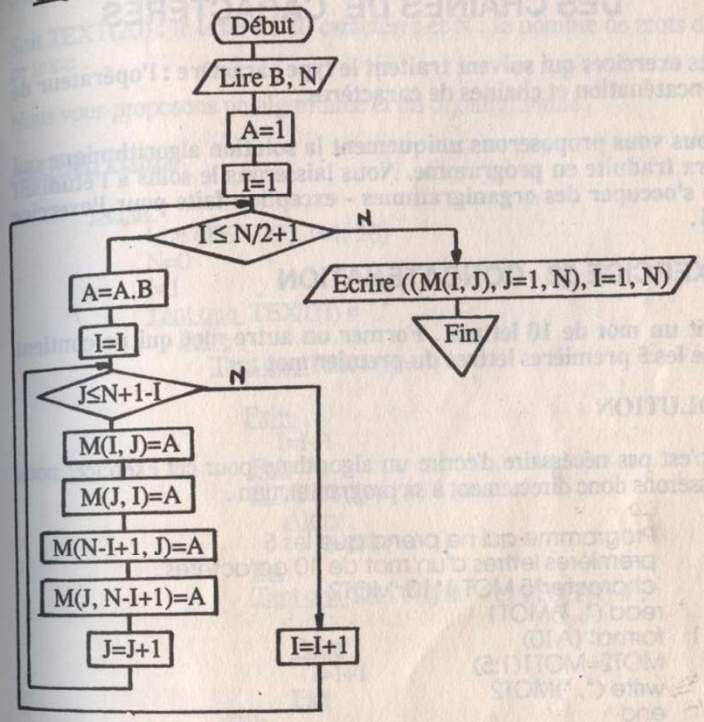
Compter le nombre de mots que comporte le texte .

```
Pour J de I à N+1-I
    Faire
        M(I, J)=A
        M(J, I)=A
        M(N-I+1, J)=A
        M(J, N-I+1)=A
    Fait
Fait
Ecrire ((M(I, J), J=1, N), I=1, N)
Fin
```

PROGRAMME

```
      dimension M(8, 8)
      integer b, a
      read (*, *)b, n
      a=1
      do 10 i=1, n / 2+1
C  calcul de la valeur à stocker en fonction de B
      a=a*b
      do  10  j=i, n+1-i
C  stockage de A dans le rang adéquat de M
      M(i, j)=a
      M(j, i)=a
      M(n-i+1, j)=a
      M(j, n-i+1)=a
 10   continue
      do  30  i=1, n
 30   write (*, 40)(M(i, j), j=1, n)
 40   format(8(1x, i4))
      stop
      end
```


Organigramme



TROISIEME CHAPITRE TRAITEMENT DES CHAINES DE CARACTERES

Les exercices qui suivent traitent le type caractère : l'opérateur de concaténation et chaines de caractères .

Nous vous proposerons uniquement la solution algorithmique qui sera traduite en programme. Nous laisserons le soins à l'étudiant de s'occuper des organigrammes - exception faite pour l'exercice 53.

EXERCICE 52 CONCATENATION

Soit un mot de 10 lettres . Former un autre mot qui ne contient que les 5 premières lettres du premier mot .

SOLUTION

Il n'est pas nécéssaire d'écrire un algorithme pour cet exercice; nous passerons donc directement à sa programmation .

```
C       Programme qui ne prend que les 5
C       premières lettres d'un mot de 10 caractères .
        character*5 MOT 1*10, MOT2
        read (*, 1)MOT1
1       format (A10)
        MOT2=MOT1(1:5)
        write (*, *)MOT2
        end
```

EXERCICE 53 NOMBRE DE MOTS DANS UN TEXTE

Soit un texte donné pouvant avoir N caractères (prenons N=20) . La fin du texte est spécifiée par un point, et le texte peut contenir des blancs au début et entre les mots .

SOLUTION

Soit TEXT(20) : le texte de 20 caractères et N : le nombre de mots de ce texte .

Nous vous proposons un algorithme et un organigramme.

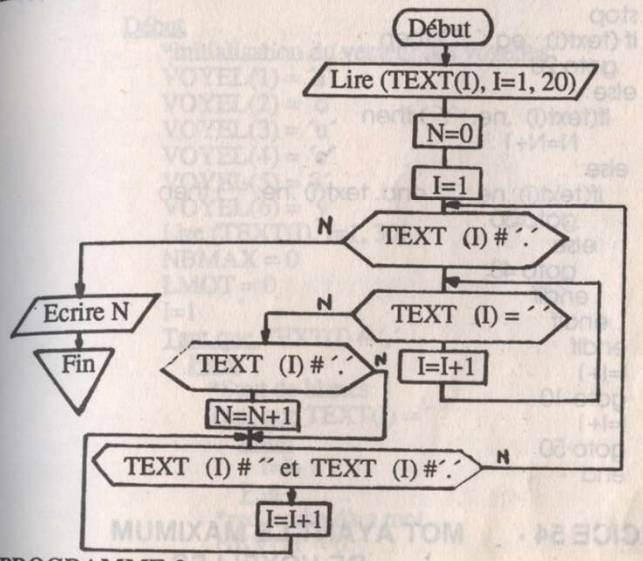
ALGORITHME

```
Début
    Lire (TEXT(I), I=1, 20)
    N=0
    I=1
    Tant que TEXT(I) # '.'
      Faire
        Tant que TEXT(I) = ' '
          Faire
            I=I+1
          Fait
        Si TEXT(I) # '.'
          Alors
            N=N+1
        Fsi
        Tant que TEXT(I) # ' ' et TEXT(I) #'.'
          Faire
            I=I+1
          Fait
      Fait
        Ecrire N
Fin
```

PROGRAMME-1

```
C       Programme qui compte le nombre de mot que contient
C       un texte dont la fin est spécifiée par un point.
        character*1 text(20)
C       Lecture du texte ...
        write (*, *) ´ Donnez le texte verticalement :´
        read (*, 1) (text(i), i=1, 20)
    1   format (A)
        write (*, *)(text(i), i=1, 20)
 C      Traitement ......
        N=0
        i=1
        DO while (text(i) .ne. ´.´)
          DO while (text(i) .eq. ´ ´)
            I=I+1
          Enddo
          if (text(i) .ne. ´.´) then
            N=N+1
          endif
          Do While (text(i) .ne. ´ ´ . and. text(i) .ne. ´.´)
            I=I+1
          Enddo
        Enddo
C       Impression du résultat ...
        write (*, *)´Nombre de mots contenus dans le texte:´,N
        stop
        end
```


ORGANIGRAMME



PROGRAMME-2

```
C       Programme qui compte le nombre de mots que contient
C       un texte dont la fin est spécifiée par un point .
        character *1  text(20)
C       Lecture du texte ...
        write (*, *) ´ Donnez le texte verticalement :
        read (*, 1) (text(i), i=1, 20)
    1   format (A)
        write (*, *) (text(i), i=1, 20)
 C      Traitement ....
        N=0
        i=1
   40   if (text(i) .ne. ´.´ ) then
          goto 10
        else
C       Impression du résultat ...
```

```
      write (*, *)´Nombre de mots contenus dans le texte :´, N
      stop
10    if (text(i) .eq. ´ ´) then
        goto 20
      else
        if(text(i) .ne. ´.´) then
          N=N+1
      else
50      if(text(i) .ne. ´ ´.and. text(i) .ne. ´.´) then
          goto 30
        else
          goto 40
        endif
      endif
    endif
20    I=I+1
      goto 10
30    I=I+1
      goto 50
      end
```

EXERCICE 54 · MOT AYANT LE MAXIMUM DE VOYELLES

Chercher le mot qui comporte le maximum de voyelles dans un texte pouvant contenir des blancs et dont la fin est donnée par un point .

(Nous supposerons que le texte contient N caractères, avec N = 30).

SOLUTION

Utilisons les vecteurs : TEXT(30) pour le texte, VOYEL(6) pour les voyelles , MOTC(25) qui contiendrait le mot courant dans la chaine et MAXVOY(25) le mot cherché avec LMOT sa longueur .
NBMAX compte le nombre maximal de voyelles trouvées et NBVOY le nombre de voyelles rencontrées dans un mot .

ALGORITHME-1

```
Début
  *initialisation du vecteur des voyelles,
  VOYEL(1) = ´a´
  VOYEL(2) = ´o´
  VOYEL(3) = ´u´
  VOYEL(4) = ´e´
  VOYEL(5) = ´i´
  VOYEL(6) = ´y´
  Lire (TEXT(I), I=1, 30)
  NBMAX = 0
  LMOT = 0
  I=1
  Tant que TEXT(I) # ´.´
    Faire
      *Saut de blancs
      Tant que TEXT(I) = ´ ´
        Faire
          I=I+1
        Fait
      *recherche d'un mot
      NBVOY = 0
      J=0
      Tant que TEXT(I)  # ´.´ et TEXT(I) # ´ ´
        Faire
          J=J+1
          MOTC(J) = TEXT(I)
          Pour K de 1 à 6
            Faire
              Si VOYEL(K)=MOTC(J)
                Alors
                  NBVOY=NBVOY+1
              Fsi
            Fait
          I=I+1
        Fait
        *tester si le mot courrant a le maximum
    *de voyelles par rapport aux mots
        *déjà rencontrés .
```

```
        Si NBVOY > NBMAX
          Alors
            NBMAX = NBVOY
            Pour K de 1 à J
              Faire
                MAXVOY(K) = MOTC(K)
              Fait
            LMOT = J
          Fsi
        Fait.
        Si LMOT # 0
          Alors
            Imprimer (MAXVOY(K), K=1, LMOT)
          Fsi
      Fin
```

PROGRAMME-1

```
C    Programme qui cherche le mot comportant le maximum
C    de voyelles dans un texte dont la fin est limité par un point
     character*1 text(30), voyel(6), motc(25), maxvoy(25),
    1message*35
C  Initialisons le vecteur des voyelles : VOYEL
     voyel(1)='a'
     voyel(2)='o'
     voyel(3)='u'
     voyel(4)='e'
     voyel(5)='i'
     voyel(6)='y'
C  Lecture du texte :
     message=' Donnez le texte verticalement :'
     write (*, 2)message
   2 format (A35)
     read (*, 1) (text(i), i=1, 30)
   1 format (A)
     write (*, *)(text(i), i=1, 30)
C  Traitement...
     nbmax=0
```



```
     i=1
     lmot=0
     DO WHILE (text(i) .ne. '.')
       DO WHILE (texte(i) .eq. ' ')
         i=i+1
       ENDDO
       nbvoy=0
       j=0
       DO WHILE ((text(i) .ne. '.') .AND. (text(i).ne.' '))
         j=j+1
         motc(j)=text(i)
         DO 50  k=1, 6
           IF (voyel(k) .eq. motc(j)) THEN
             nbvoy=nbvoy+1
           ENDIF
50       CONTINUE
         i=i+1
       ENDDO
       IF (nbvoy .gt. nbmax) THEN
         nbmax=nbvoy
         do  70  k=1, j
70       maxvoy(k)=motc(k)
         lmot=j
       ENDIF
     ENDDO
     IF (lmot .ne. 0)  THEN
       write (*, *)(maxvoy(k), k=1, lmot)
     ENDIF
     end
```

ALGORITHME-2

```
Début
   *initialisation du vecteur des voyelles
   VOYEL(1) = 'a'
   VOYEL(2) = 'o'
   VOYEL(3) = 'u'
   VOYEL(4) = 'e'
   VOYEL(5) = 'i'
   VOYEL(6) = 'y'
```

```
      Lire (TEXT(I), I=1, 30)
      NBMAX = 0
      LMOT = 0
      I=1
80    Si TEXT(I) # '.'
        Alors
          Aller à 10
        Sinon
          Si LMOT # 0
            Alors
              Ecrire (MAXVOY(K), K=1, LMOT)
              Aller à Fin
            Fsi
        Fsi
10      Si TEXT(I) = ' '
          Alors
            Aller à 20
          Sinon
            NBVOY = 0
            J=0
60          Si (TEXT(I) # '.' et TEXT(I)# ' ')
              Alors
                Aller à 30
              Sinon
                Aller à 80
            Fsi
          Fsi
20        I=I+1
          Aller à 10
30        J=J+1
          MOYC(J)=TEXT(I)
          Pour K de 1 à 6
            Faire
              Si VOYEL(K)=MOTC(J)
                Alors
                  NBVOY=NBVOY+1
              Fsi
            Fait
            I=I+1
```



```
          Aller à 60
40        NBMAX = NBVOY
          Pour K de 1 à J
            Faire
              MAXVOY(K) = MOTC(K)
            Fait
              LMOT = J
                Aller à 80
          Fin
```

PROGRAMME-2

```
C Programme qui cherche le mot comportant le maximun de
C voyelles dans un texte dont la fin est limité par un point
      character*1 text(30), voyel(6), motc(25), maxvoy(25),
     1 message*35
C Initialisons le vecteur des voyelles : VOYEL
      voyel(1)='a'
      voyel(2)='o'
      voyel(3)='u'
      voyel(4)='e'
      voyel(5)='i'
      voyel(6)='y'
C Lecture du texte :
      message=' Donnez le texte verticalement :
      write (*, 2)message
    2 format(A35)
      read (*, 1)(text(i), i=1, 30)
    1 format (A)
      write (*, *)(text(i), i=1, 30)
C Traitement ...
       nbmax=0
       i=1
       lmot=0
   80  IF (text(i) .ne. '.') THEN
         goto 10
       ELSE
         IF (lmot .ne. 0) THEN
```

```
      write (*, *)(maxvoy(k), k=1, lmot)
      STOP
    ENDIF
10  IF (text(i) .eq. ' ') THEN
      goto 20
    ELSE
      nbvoy=0
      j=0
      IF((text(i) .ne.' .') .AND. (text(i) .ne.' ')) THEN
        goto 30
      ELSE
        goto 80
      ENDIF
    ENDIF
20  i=i+1
    goto 10
30  j=j+1
    motc(j)=text(i)
    DO 50 k=1, 6
      IF (voyel(k) .eq. motc(j)) THEN
        nbvoy=nbvoy+1
      ENDIF
50  CONTINUE
    i=i+1
    goto 60
40  nbmax=nbvoy
    do 70 k=1, j
70  maxvoy(k)=motc(k)
    lmot = j
    goto 80
    end
```


QUATRIEME CHAPITRE . Les sous-programmes et les instructions d'efficacite de programmation

Nous avons repris certains exercices déjà proposés dans les chapitres précédents, pour les traiter sous forme de sous-programmes .

Deux types d'instructions de sous-programmes sont utilisés : SUB-ROUTINE et FUNCTION .

Vous y trouverez également des sous-programmes traitant des instructions relatives à l'allocation mémoire comme : COMMON , EQUIVALENCE et l'instruction sous forme de sous-programme BLOCK DATA .

EXERCICE-1 (voir exercice 13)

```
C  Ce programme calcule le pgcd de deux nombres.
C  Programme Principal
     integer repgcd, a, b
     read (*, *) a, b
C  appel du sous-programme pgcd
     repgcd=pgcd(a, b)
     write (*, *)repgcd
     stop
     end

C  Sous-Programme
     integer function pgcd(a, b)
     integer a, b, r
  10 r=mot(a, b)
     if(r .eq. 0) goto 20
     a=b
     b=r
     goto=10
  20 pgcd=b
C retour au programme principal
     return
     end
```

EXERCICE-2 (voir exercice 13)

```
C  Ce programme calcule le pgcd et le ppcm de deux
   nombres.
C  Il utilise deux sous-programmes l'un calculant le pgcd,
C  l'autre le ppcm.
C  Programme Principal
       integer repgcd, reppcm, a, b
       read(*, *) c, d
C  sauvegarde des valeurs de c et d dans L et m
       L=c
       m=d
C  appel du sous-programme pgcd
       repgcd=pgcd(c, d)
C  appel du sous-programme ppcm
       reppcm=ppcm(L, m)
       write (*, *) repgcd, reppcm
       stop
       end

C  Sous-Programme ppcm
       integer function ppcm(x, y)
       integer x, y, x1, y1
       x1=x
       y1=y
       ppcm=x1*y1 / pgcd(x, y)
C  retour au programme principal
       return
       end

C  Sous-Programme pgcd
       integer function pgcd(a, b)
       integer a, b, r
  10   r=mod(a, b)
       if (r .eq. 0)  goto  20
       a=b
       b=r
       goto  10
  20   pgcd=b
C  retour au programme principal
       return
       end
```


EXERCICE-3 (voir exercice 28)

```
C  Ce programme fait le produit des éléments
C  de la diagonale d'une matrice .

C  Programme  Principal
       dimension a(5, 5)
       read (*, *)n

       do 10 i=1, n
  10   read (*, *) (a(i, j), j=1, n)
C  appel du sous-programme
       result= prodia(n, a)
       write (*, *) result
       stop
       end

C  Sous-Programme
       function prodia(n, xx)
       dimension xx(5, 5)
       produi=1 .
       do  20  j=1, n
  20   produi=produi*xx(j, j)
       prodia=produi
C  retour au programme principal
       return
       end
```

EXERCICE-4 (voir exercice 34)

```
C  Ce programme fait le tri des températures
C  en utilisant un sous-programme SUBROUTINE.

C  programme principal
       dimension t(50), t1(50), t2(50)
       read (*, *)n
       do 5 i=1, n
   5   read (1, *) t(i)
       call temp (t, t1, t2, k, L, n)
       write (*, *) (t1(i), i=1, L)
       write (*, *) (t2(i), i=1, k)
       stop
       end
```

```
C  sous-programme
       subroutine temp(t, t1, t2, k, L, n)
       dimension t(50), t1(50), t2(50)
       k=0
       L=0
       do 40  i=1, n
       if(t(i) .le.20) goto 20
       if (t(i).ge.40)goto 30
       goto 40
  20   k=k+1
       t1(k)=t(i)
       goto 40
  30   L=L+1
       t2(L)=t(i)
  40   continue
       return
       end
```

EXERCICE-5 (voir exercice 17)

```
C  Ce programme cherche la valeur de l'exponentielle en
C  utilisant deux sous-programmes FUNCTION, l'un est
C  OUSS, l'autre FAC.

C  programme principal
       read (*, *)x, m
       rexpx = ouss(x, m)
       write(*, *)rexpx
       stop
       end

C  sous-programme-1
       function ouss(x, m)
       integer fac
       y=1
       do10  n=1, m
```



```
       y=y+x**n / fac(n)
   10  continue
       ouss = y
       return
       end

C  sous-programme-2
       function fac(n)
       integer fac
       fac = 1
       do  20  j=1,n
       fac=fac*j
   20  continue
       return
       end
```

EXERCICE-6 (voir exercice 37)

```
C  Ce programme classe les éléments
C  de troix vecteurs par ordre croissant.
C  Programme Principal
       dimension x(100), z(80), y(50)
       read (1, *) (x(i), i=1, 10)
       read (1, *) (z(i), i=1, 8)
       read (1, *) (y(i), i=1, 5)
       call vectri (x, 10)
       call vectri(z, 8)
       call vectri(y, 5)
       do 10  i=1, 5
  10   write (2, 3)x(i), z(i), y(i)
   3   format (2x, 3(f10.4, 2x))
       do 11 i=6, 8
  11   write (2, 4)x(i), z(i)
   4   format (2x, 2(f10.4, 2x))
       do 12 i=9, 10
  12   write (2, 5)x(i)
   5   format (2x, f10.4)
       stop
       end
```

```
C  Sous-Programme
       subroutine vectri(v, n)
       dimension v(10)
       do  10  i=1, n-1
       do  10  j=i, n
       if(v(i) .le. v(j))  goto  10
       r=v(i)
       v(i)=v(j)
       v(j)=r
 10    continue
       return
       end
```

EXERCICE-7 (voir exercice 18)

```
C  Ce programme calcule la valeur de cos(x) .
C  Pour cela , il utilise un S / P function qui calcule la
C  factorielle et un S / P Subroutine qui fait la somme des
C  termes de la série .

C  Programme principal
     external fact
     read (*, *)x
     call somme (x, recosx, fac)
     write (*, *) recosx
     stop
     end


C  Sous-Programme 1
     subroutine somme (x, some, fact)
     integer fact
     s=1
     n=1
     k=-1
300  t=k*x**(2*n) / fact(n)
     if (abs(t).lt.1.e-5)  goto  50
     s=s+t
     k=-k
     n=n+1
     goto  300
 50  some=s
     Return
     end
```



```
C  Sous-Programme 2
     function fact(N)
     integer  fact
     fact=1
     if (n .eq. 0) return
     do  10 j=1, n
     fact=fact*2*j*(2*j-1)
 10  continue
     return
     end
```

EXERCICE-8 (voir exercice 36)

```
C  Ce programme calcule la moyenne et l'écart type .
C  Il utilise un S / function qui calcule la moyenne
C  et un S / P subroutine qui calcule l'écart type .

C  Programme Principal
     EXTERNAL  xmoy
     dimension x(100)
     real moyen
     read (1, *)n
     read (1, *) (x(i), i=1, n)
     moyen=xmoy(n, x)
     call resul (n, x, result, xmoy
     write (*, *)moyen, result
     stop
     end
```

```
C  Sous-Programme 1
       function xmoy(n, x)
       dimension x(n)
       xmoy=0
       do 10 i=1, n
  10   xmoy=xmoy+x(i)
       xmoy=xmoy / n
       return
       end

C  Sous-Programme 2
       subroutine result (n, x, result, xmoy)
       dimension x(n)
       ecatyp=0
       do 20 i=1, n
  20   ecatyp=ecatyp+(x(i)-xmoy(n, x))**2
       result=sqrt(ecatyp / n)
       return
       end
```

EXERCICE 9 LONGUEUR D'UN MOT

```
C  Programme qui lit un texte de N caractères (n=30)
C  et qui détermine le nombre de mots du texte dont la
C  longueur est de sept (7) lettres .
C  Ce texte peut commencer par des blancs ou en contenir .

C  Programme principal
       character*1 car(30)
       common i
C  Lecture du texte ...
       write (*, *) ` Donnez le texte verticalement
       read (*, 3) (car(i), i=1, 30)
    3  format (A)
       write (*, *) (car(i), i=1, 30)
```



```
C  Traitement ...
       i=1
       nmot=0
    1  call cherch(car)
       if(i .gt. 30) goto 2
       L=long(car)
       if (L .eq. 7) nmot=nmot+1
       if (i .gt. 30) goto 2
       goto 1
C  Impression du résultat ...
    2  write(*, *)`Nombre de mots formés de 7 lettres:`, nmot
       stop
       end

C  Sous-Programme-1 , qui cherche le début d'un mot
       subroutine cherch(car)
       common i
       character *1 car (30)
       data blanc /` `/
    1  if(car(i) .ne. blanc) return
       i=i+1
       if (i .gt. 30) return
       goto 1
       end

C  Sous-Programme-2 , qui calcule la longueur d'un mot
       integer function long(car)
       common i
       character*1 car(30)
       long=0
    1  if(car(i) .eq.` `) return
       long=long+1
       i=i+1
       if(i .gt. 30) return
       goto 1
       end
```

EXERCICE-10 (voir exercice 54)

```
C  Programme qui cherche le mot comportant le maximun de
C  voyelles dans un texte dont la fin est limité par un point.

      character*1 text(30), voyel(6), motc(25),maxvoy(25),
     3 mvoy*6
      EQUIVALENCE (mvoy, voyel)
C   Initialisons le vecteur des voyelles : VOYEL
      data mvoy /aoueiy`/`Donnez le texte verticalement :`
C     Lecture du texte:
write (* , 2)' Donnez le texte verticalement :'
    2 format (A35)
      read (*, 1) (text(i), i=1, 30)
    1 format (A)
      write (*, *) (text(i), i=1, 30)
C   Traitement ...
      nbmax=0
      i=1
      lmot=0
   80 if (text(i) .ne. `.`) goto 10
      if (Lmot .ne. 0) then
C   Impression du résultat :
      write(*, *)`Le mot recherché est :`, (maxvoy(k),k=1,lmot)
      endif
      stop
   10 if (text(i) .eq. ` `) goto 20
      nbvoy=0
      j=0
   60 if((text(i) .ne. `.`) .AND. (text(i) .ne. ` `)) goto 30
      if(nbvoy .gt. nbmax) goto 40
      goto 80
   20 i=i+1
      goto 10
   30 j=j+1
      motc(j)=text(i)
      do 50 k=1, 6
      if(voyel(k) .eq. motc(j)) nbvoy=nbvoy+1
   50 continue
      i=i+1
      goto 60
   40 nbmax=nbvoy
      do 70 k=1, j
   70 maxvoy(k)=motc(k)
      lmot=j
      goto 80
      end
```


EXERCICE 11 CODIFICATION D'UN TEXTE

Soit un texte formé de lettres de l'alphabet et de blancs et dont la fin est limitée par un point .

Codifier ce texte , sachant que :

a→ d, b→ e, c→ f, ..., y → b, z→ c .

SOLUTION :

Utilisons : Text (70) pour le texte en clair .
Codtex(70) pour le texte codé .
Alpha (26) pour l'alphabet .
Code (26) pour l'alphabet codé .

PROGRAMME :

```
C  Programme qui codifie un texte contenant
C  des lettres de l'alphabet comme suit :
C  a→ d, b→ e, c→ f, ....., y → b, z→ c .
C . Le texte contient des blancs et la fin est limitée par un point.

      character*1 text(70), alpha(26), malpha*26,
     1car*70, codtex(70), cod(26), mcod*26
      EQUIVALENCE (malpha, alpha)
      EQUIVALENCE (mcod, cod)
      EQUIVALENCE (car, text)

C  Initialisons le vecteur de l'alphabet :
C  ALPHA et celui du code : COD
      data malpha /`abcdefghijklmnopqrstuvwxyz` /
      data mcod /`defghijklmnopqrstuvwxyzabc` /
C  Lecture du texte :
      write(*, 2)` Donnez le texte horizontalement :`
```

```
 2  format (A35)
    read(*, 1)car
 1  format(A30)
C  Traitement ....
    i=1
80  if(text(i).ne.`.`) goto 10
C  Impression du résultat :
    write(*; *)`Le texte codé est : `
    write(*, *)(codtex(i), i=1, 70)
    stop

10  if(text(i) .eq.` `) goto 20
60  if((text(i) .ne. `.`) .AND. (text(i).ne. ` `)) goto 30
    goto 80
20  codtex(i) = ` `
    i=i+1
    goto 10
30  do 50 j=1, 26
    if (text(i) .eq. alpha(j)) codtex(i)=cod(j)
50  continue
    i=i+1
    goto 60
    end
```

EXERCICE 12. (Voir exercice 47).

```
C  Dans ce programme nous utilisons les instructions
C  BLOCK DATA avec un COMMON étiqueté

C  Programme principal
    integer c(20), x(30), val
    common / sal / c, x
    common / nac / val, max
    call valg
    call valmax
    write (1, 15)
15  format (2x, 25HLes composantes de C(x) :)
    do 10 i=1, 20
10  write (1, *)max, C(i)
    write (1, 20)
```



```
20  format (12x, ` max `, 10x, `val`)
    write (1, *) max, val
    stop
    end

C  Sous-programme BLOCK DATA
    BLOCK DATA
    integer c(20), x(30), val
    common / sal / c, x
    common / nac / val, max
    data x / nac / val, max
    data x / 6*1, 2*2, 4*9,3*10, 11, 12, 3*13, 5*17, 3*18, 12*19/
    end

C  Sous-programme calculant le nombre de fois
C  qu'apparait chaque valeur du vecteur C
    subroutine VALG
    integer c(20) , x(30)
    common / sal / c, x
    do 25 i=1, 30
25  C(x(i)=C(x(i))+1
    return
    end

C  Sous-programme calculant la valeur maximale et son rang
    subroutine VALMAX
    integer c(20), x(30), val
    common / sal / c, x
    common / nac / val, max
    max=0
    do 30 i=1, 20
     if(max.lt.c(i)) then
       max=c(i)
       val=i
     endif
30  continue
    return
    end
```

CINQUIEME CHAPITRE. LES FICHIERS

Dans ce dernier chapitre , nous reprendrons certains exercices proposés dans les deux premiers chapitres pour les traiter sous forme de fichiers .

Néanmoins, nous vous proposons un problème de gestion qui n'a pas été traité auparavant .

EXERCICE 1 (voir exercie 37)

```
C  Programme de Tri Linéaire d'un vecteur A(10)

      dimension a(10)

     OPEN(1, status=`new`, file= F1.dat`)
     REWIND  1
     OPEN (2, status=`new`, file=`F2.res`)

     read(1, *)(a(i), i=1, 10)
     do 10  i=1, 9
     do 10  j=i+1, 10
     if (a(i).le.a(j))  goto  10
     u=a(i)
     a(i)=a(j)
     a(j)=u
 10  continue
     write (2, 20)(a(i), i=1, 10)
 20  format(5(1x, f10.4))
     stop
     end
```

EXERCICE 2 (voir exercice 50)

```
      dimension t(10, 20), a(10, 20), v(200)
     OPEN  (1, FILE = ` f1.dat` , status = `new` )
     rewind(1)
     OPEN (2, FILE = `f2.sol`, status =`old` )
```



```
      read(*, *)n, m
      do  10  i=1, n
 10   read(*, *)(t(i, j), j=1, m)
C   transformation de la matrice T en vecteur V.
      k=0
      do 20 i=1, n
        do 20 j=1, m
          k=k+1
          v(k)=t(i, j)
 20   continue
C   opérations de classement des éléments.
      do 50  k=1, n*m-1
        do  50  j=k+1, n*m
          if(v(k) .gt. v(j))  then
            sauv=v(k)
            v(k)=v(j)
            v(j)=sauv
          endif
 50   continue
C   transformation du vecteur V en matrice A .
      k=0
      do  70  i=1, n
        do  70  j=1, m
          k=k+1
          a(i, j)=v(k)
 70   continue
C    impression de la matrice A .
      write(2, *) ` Impression de la matrice A triée `
      write(2, *) `-------------------------------`
      do 90  i=1, n
 90   write (*, 100)(a(i, j), j=1, m)
100   format(20(1x, f14.6))
      stop
      end
```

EXERCICE 3 (voir exercice 52)

```
     dimension M(8, 8)
     integer b, a
     OPEN ( 7, file = `sortie.dat `, status = `new` )
     read(*, *)b, n
     a=1
     do 10 i=1, n / 2+1
C  calcul de la valeur à stocker en fonction de B
     a=a*b
```

```
        do 10 j=i,n+1-i
C stockage de A dans le rang adéquat de M
        M(i, j)=a
        M(j, i)=a
        M(n-i+1, j)=a
        M(j, n-i+1)=a
10      continue
        write(7, *) ` Matrice formée avec l'élément B `
        write(7, *) ` ----------------------------------`
        do 30 i=1, n
30      write (*, 40)(M(i, j), j=1, n)
40      format (8(1x, i4))
        stop
        end
```

EXERCICE 5 SALAIRES DES EMPLOYES

Une entreprise a fourni les renseignements suivants concernant ses employés :

MEMPj : matricule de l'employé j .

SALj : salaire mensuel de l'employé j (valable pour le mois passé) .

N : le nombre d'employés .

Elle demande de réaliser le programme qui calcule :

a) La somme des primes mensuelles perçues par chacun des employés de l'entreprise , en sachant qu'il y a une prime de rendement indviduel qui est de 20 % du salaire , et une prime de rendement collectif qui est de 5 % .

Pour le mois en cours, il y a une augmentation de salaire comme suit :

10 % , si le salaire est supérieur à 2500 DA .
15 %, s'il est inférieur .

b) La moyenne des salaires perçus par les employés .
c) Les 30 plus bas salaires de l'entreprise .

d) Le nombre d'employés dont le salaire est compris entre 3000 et 6000 DA.

Remarque : pour a, b et c le salaire est considéré avec primes comprises .

SOLUTION :

Soit :

PRIMj : utilisé pour le calcul des primes pour un employé j .
MOY : la moyenne des salaires des employés .
NCOMPT : le nombre d'employés dont le salaire est compris entre 3000 et 6000 DA .

avec :

PRIMj = SALj * 0.20+SALj *0.05

ALGORITHME :

```
Lire N
Lire (MEMP(j), j=1, N)
Lire (SAL(j), j=1, N)
MOY=0
NCOMPT=0
Pour J de 1 à N
   Faire
    *augmentation des salaires
    Si  SAL (j) > 2500
       Alors
          SAL(j) = SAL(j) + SAL(j) * 0.1
       Sinon
          SAL(j) = SAL(j) + SAL(J) * 0.15
    Fsi
    *primes
    PRIM(j) = sal(j) * 0.20+SAL(j) * 0.05
    *salaire avec prime
    SAL(j) = SAL(j) + PRIM(j)
```

```
      *moyenne
      MOY = MOY + SAL (j)
      *nombre d'employés dont le salaire est entre 3000
*et 6000 DA
      Si SAL (j) ≤ 6000 et SAL (j) ≥ 3000
        Alors
          NCOMPT = NCOMPT+1
      Fsi
    Fait
    MOY = MOY / N
    Imprimer (MEMP(j), j=1, N), (PRIM(j), j=1, N)
    Imprimer MOY, NCOMPT
    *calcul des 30 plus bas salaires
    Pour I de 1 à N-1
       Faire
         Pour J de I+1 à N
            Faire
              Si SAL (I) >SAL (J)
                Alors
                   SAUV = SAL(i)
                   SAL(I)=SAL(j)
                   SAL(j)=SAUV
                   NSAUV=MEMP(i)
                   MEMP(i)=MEMP(j)
                   MEMP(j)=NSAUV
                Fsi
         Fait
    Fait
  Imprimer (SAL(j), j=1, 30), (MEMP(j), j=1, 30)
  Fin
```

PROGRAMME

```
C Programme qui calcule les primes des employés d'une
C entreprise, la moyenne des salaires, le nbre d'employés qui
C perçoivent un salaire entre 3000 et 6000 DA, les 30 plus bas
C salaires .

      dimension memp(100), sal(100), prim(100)
      open (1, file=`F.dat`, status= `old`)
```



```
      rewind(1)
      open (3, file =` F.sor`, satus= `new`)

      read (1, *)n, (memp(j), j=1, n), (sal(j), j=1, n)
      moy=0
      ncompt=0
      do 10 j=1, n
      if (sal(j).GT.2500) then
        sal(j)=sal(j)+sal(j)*0.1
      else
        sal(j)=sal(j)+sal(j)*0.15
      endif
    prim(j)=sal(j)*0.20+sal (j) * 0.05
    sal(j)=sal(j)+prim(j)
    moy=moy+sal(j)
    if((sal(j).LE.6000).AND.(sal(j)).GE.3000))ncompt=ncompt+1
10  continue
    moy=moy /n
    write (3, *) ` Matricules et primes des employés`
    write (3, *) `-----------------------------------`
    do 20 J = 1, N
    write (3, *)memp(j), prim(j)
20  continue
    write (3, *)`Moyenne des salaires :`,moy
    write (3, *) `Nbre d'employés dont le salaire est com-
   1pris entre 3000 et 6000:`, ncompt
    do 30 i=1, n-1
    do 30 j=i+1, n
    if (sal(i) .gt. sal(j)) then
      sauv=sal(i)
      sal(i)=sal(j)
      sal(j)=sauv
      nsauv=memp(i)
      memp(i)=memp(j)
      memp(j)=nsauv
      endif
30  continue
    write (3, *)`Les 3 plus bas salaires sont : `
    write (3, *)`    Matricules   et   Salaires`
    write (3, *)`    ----------       ----------`
    do 40 j=1, 3
40    write (3, *)memp(j), sal(j)
      close (1)
       stop
       end
```

ANNEXE : FONCTIONS ARITHMETIQUES

1. FONCTIONS DE CONVERSIONS :

a- réels en entiers
INT(X) : troncature
IFIX(X) : flottant en fixe
DINT(X) : double précision en entier

b- entiers en réels
FLOAT(X) : entier en réel
DFLOAT(X) : entier en double précision

c- réels en complexes
CMPLX(X, Y) : résultat complexe

2. RESTE D'UNE DIVISION

MOD(X, Y) : X, Y entier ; le reste est entier
AMOD(X,Y) : X, Y réels ; le reste est réel
DMOD(X, Y) : X, Y double précision ; le reste est double précision

3. VALEUR ABSOLUE

ABS(X) : X réel
IABS(X) : X entier
DABS(X) : X double précision
CABS(X) : X complexe

4. RACINE CARREE

SQRT(X) : X réel ≥ 0
DSQRT(X) : X double précision ≥ 0
CSQRT(X) : X complexe

5. EXPONNENTIELLE

EXP(X) : X réel
DEXP(X) : X double précision
CEXP(X) : X complexe

6. LOGARITHMES

a- logarithmes Népérien	b- logarithmes décimaux	argument
ALOG(X)	ALGO10(X)	X réel
DLOG(X)	DLOG10(X)	X double précision
CLOG(X)		X complexe

7. TRIGONOMETRIE

a- sinus	b- cosinus	c- tangente	argument
SIN(X)	COS(X)	TAN(X)	X réel
DSIN(X)	DCOS(X)	DTAN(X)	X double précision
CSIN(X)	CCOS(X)		X complexe

8. TRIGONOMETRIE INVERSE

a- arc sinus
ARCSIN(X) X réel

b- arc cosinus
ARCCOS(X) X réel

c- arc tangente

ATAN(X)	$X \in [-\pi/2 , +\pi/2]$
DATAN(X)	$X \in [-\pi/2 , +\pi/2]$
ATAN2(X)	$X \in [-\pi , +\pi]$
DATAN2(X)	$X \in [-\pi , +\pi]$

9. MAXIMUM

AMAXO(X, Y, Z, ...)	X, Y, Z, ... : entiers, résultat réel
AMAX1(X, Y, Z, ...)	X, Y, Z, ... : réels, résultat réel
MAXO(X, Y, Z, ...)	X, Y, Z, ... : entiers, résultat entier
MAX1(X, Y, Z, ...)	X, Y, Z, ... : réels, résultat entier

10. MINIMUM

AMINO (X, Y, Z, ...)	X, Y, Z, ... : entiers, résultat réel
AMIN1 (X, Y, Z, ...)	X, Y, Z, ... : réels, résultat réel
MINO(X, Y, Z, ...)	X, Y, Z, ... : entiers, résultat entier
MIN1(X, Y, Z, ...)	X, Y, Z, ... : réels, résultat entier

11. PARTIES REELLE ET IMAGINAIRE D'UN COMPLEXE

REAL(X) : X complexe, résultat réel (partie réelle)
AIMAG(X) : X complexe, résultat réel (partie imaginaire)