OFPPT

ROYAUME DU MAROC

مكتب التكوين المهني وإنعاش الشغل

Office de la Formation Professionnelle et de la Promotion du Travail

DIRECTION RECHERCHE ET INGENIERIE DE FORMATION

RESUME THEORIQUE
&
GUIDE DES TRAVAUX PRATIQUES

MODULE N°3 :

Titre : Codification d'un algorithme et programmation procédurale

SECTEUR : **Tertiaire**

SPECIALITE : **Technicien Spécialisé en Développement Informatique**

NIVEAU : **TS**

Mars 2006

REMERCIEMENT

La DRIF remercie les personnes qui ont contribué à l'élaboration du présent document.

Pour la supervision :

MME.BENNANI WAFAE	DIRECTRICE CDC TERTIAIRE & TIC
M. ESSABKI NOURDDINE	CHEF DE DIVISION CCFF

Pour la conception :

- JELLAL ABDELILAH	Formateur animateur au CDC Tertiaire & TIC
- KHLIFA AIT TALEB	Formateur à l'ITA GUELMIM

Pour la validation :

Les utilisateurs de ce document sont invités à communiquer à la DRIF toutes les remarques et suggestions afin de les prendre en considération pour l'enrichissement et l'amélioration de ce programme.

Said Slaoui
DRIF

OBJECTIF OPERATIONNELS DE PREMIER NIVEAU
DE COMPORTEMENT

COMPORTEMENT ATTENDU

Pour démontrer sa compétence, le stagiaire doit **codifier un algorithme et utiliser un langage procédural** selon les conditions, les critères et les précisions qui suivent.

CONDITIONS D'EVALUATION

- Travail individuel effectué avec un PC équipé d'un environnement de développement :
- éditeur de texte
- Le langage de programmation Java
- une interface homme machine graphique(type Windows) n'est pas indispensable
- l'utilisation d'un formalisme de représentation des algorithmes est obligatoire.

CRITERES GENERAUX DE PERFORMANCE

- Utilisation des commandes appropriées.
- Respect du temps alloué.
- Respect des règles d'utilisation du matériel et logiciel Informatique.

PRECISIONS SUR LE COMPORTEMENT ATTENDU	**CRITERES PARTICULIERS DE PERFORMANCE**
A. Structurer le programme à codifier	• Analyse judicieuse des composants du programme. • Enumération des différentes instructions du programme • Structuration correcte de l'enchaînement des instructions dans un diagramme • Définition juste du format d'un algorithme • Définition juste des instructions d' E/S
D. Utiliser les instructions de base d'un algorithme	• Utilisation appropriée des instructions conditionnelles • Utilisation juste du syntaxe de la boucle • Utilisation judicieuse des notions fondamentales d'un tableau : - Notion de tableau une dimension - Notion de tableau multi - dimensions • Déclaration juste des tableaux • Affectation correcte des tableaux • Utilisation pertinente des tests booléens • Pratique approprié de la recherche dichotomique • Imbrication juste des structures répétitives et alternatives • Regroupement correct des instructions adéquates en fonctions et procédures cohérentes et réutilisables
C. Utiliser les fichiers	• Structuration correcte des données au sein d'un fichier texte • Définition judicieuse du type d'accès • accès séquentiel • accès direct (ou aléatoire) • Ouverture correcte d'un fichier texte pour : • Lecture • Ecriture
E. Traduire l'algorithme dans le langage de programmation JAVA	• Codification correcte de l'algorithme selon les instructions du langage JAVA • Utilisation judicieuse du compilateur (messages) et des outils de déboguage • Test de l'appel d'un sous programme • Correction éventuelle des erreurs

Partie 01 : Algorithme

1. INTRODUCTION

1.1. Notion de programme

Si l'on s'intéresse aux applications de l'ordinateur, on s'aperçoit qu'elles sont très nombreuses. En voici quelques exemples :

- Etablissement de feuille de payes, de factures
- Gestion de stocks
- Calcul de la trajectoire d'un satellite
- Suivi médical de patients dans un hôpital

...

Un ordinateur pour qu'il puisse effectuer des tâches aussi variées il suffit de le programmer. Effectivement l'ordinateur est capable de mettre en mémoire un programme qu'on lui fournit puis l'exécuter.

Plus précisément, l'ordinateur possède un ensemble limité d'opérations élémentaires qu'il sait exécuter. Un programme est constitué d'un ensemble de directives, nommées instructions, qui spécifient :

- les opérations élémentaires à exécuter
- la façon dont elles s'enchaînent.

Pour s'exécuter, un programme nécessite qu'on lui fournisse ce qu'on peut appelé << informations données » ou plus simplement << données ». En retour, le programme va fournir des << informations résultats » ou plus simplement résultats.

Par exemple un programme de paye nécessite des informations données : noms des employés, situations de famille, nombres d'heures supplémentaires, etc... Les résultats seront imprimés sur les différents bulletins de paye.

1.2. Le processus de la programmation

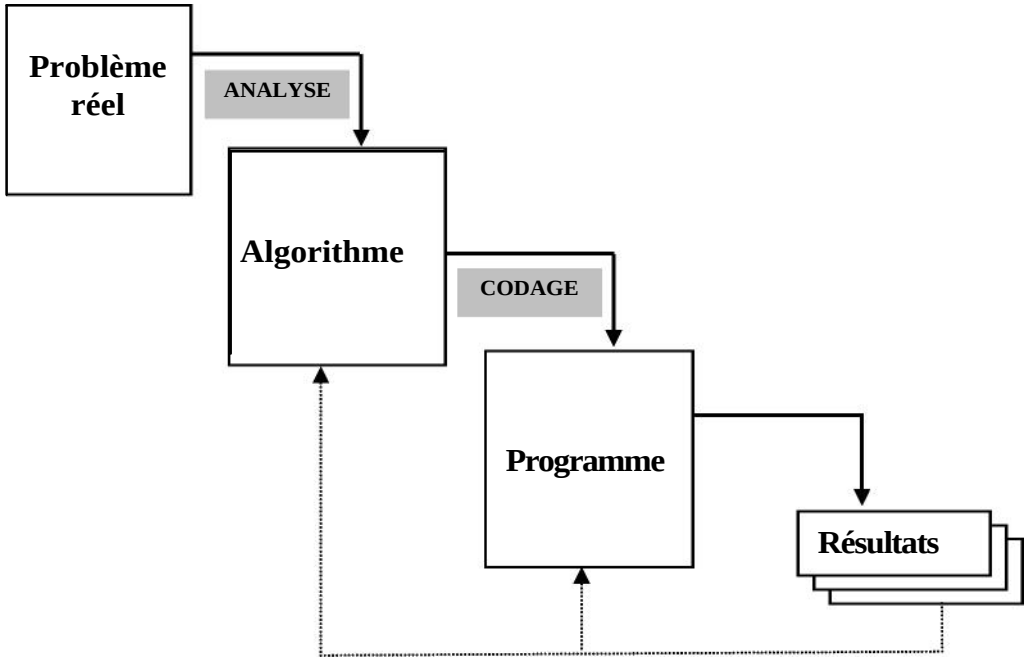
La programmation consiste, avant tout, à déterminer la démarche permettant d'obtenir, à l'aide d'un ordinateur, la solution d'un problème donné.

Le processus de la programmation se déroule en deux phases :

- dans un premier temps, on procède à ce qu'on appelle l'analyse du problème posé ou encore la recherche d'un algorithme[1] qui consiste à définir les différentes étapes de la résolution du problème. C'est la partie essentielle dans le processus de programmation. Elle permet de définir le contenu d'un programme en termes de données et d'actions.
- Dans un deuxième temps, on exprime dans un langage de programmation donné, le résultat de l'étape précédente. Ce travail, quoi qu'il soit facile, exige le respect strict de la syntaxe du langage de programmation.

Lors de l'étape d'exécution, il se peut que des erreurs syntaxiques sont signalées, ce qui entraîne des corrections en général simple ou des erreurs sémantiques plus difficiles à déceler. Dans ce dernier cas, le programme produit des résultats qui ne correspondent pas à ceux escomptés : le retour vers l'analyse sera alors inévitable.

1 Un algorithme est une suite d'actions que devra effectuer un ordinateur pour arriver à un résultat, à partir d'une situation donnée.



Les différentes étapes du processus de programmation

Donc, la résolution d'un problème passe tout d'abord par la recherche d'un algorithme.
L'objectif de ce cours est de vous fournir les éléments de base intervenant dans un algorithme : variable, type, instructions d'affectation, de lecture, d'écriture, structures.

2. LES VARIABLES

2.1. La notion de variable

Dans un programme informatique, on va avoir en permanence besoin de stocker provisoirement en mémoire des valeurs. Il peut s'agir de données issues du disque dur ou fournies par l'utilisateur (frappées au clavier). Il peut aussi s'agir de résultats obtenus par le programme, intermédiaires ou définitifs. Ces données peuvent être de plusieurs types (on en reparlera) : elles peuvent être des nombres, du texte, etc. Dès que l'on a besoin de stocker une information au cours d'un programme, on utilise une variable.
Une variable est un nom qui sert à repérer un emplacement donné de la mémoire, c'est à dire que la variable ce n'est qu'une adresse de mémoire.
Cette notion contribue considérablement à faciliter la réalisation des programmes. Elle permet de manipuler des données sans avoir à se préoccuper de l'emplacement qu'elles occupent effectivement en mémoire. Pour cela, il vous suffit tout simplement de leur choisir un nom. Bien entendu, la chose n'est possible que parce qu'il existe un programme de traduction (compilateur, interpréteur) de votre programme dans le langage machine ; c'est lui qui attribuera une adresse à chaque variable.
Le programmeur ne connaît que les noms A, MONTANT, RACINE... Il ne se préoccupe pas des adresses qui leur sont attribuées en mémoires.
Le nom (on dit aussi identificateur) d'une variable, dans tous les langages, est formé d'une ou plusieurs lettres ; les chiffres sont également autorisés à condition de ne pas apparaître au début du nom. La plupart des signes de ponctuation sont exclus en particulier les espaces.
Par contre, le nombre maximum de caractères autorisés varie avec les langages. Il va de deux dans certains langages jusqu'à quarante.
Dans ce cours, aucune contrainte de longueur ne vous sera imposée. De même nous admettrons que les lettres peuvent être indifférents des majuscules ou des minuscules.

Remarque : Pour les noms des variables choisissez des noms représentatifs des informations qu'ils désignent ; ainsi MONTANT est un meilleur choix que X pour désigner le montant d'une facture.

Une variable peut être caractérisé aussi par sa valeur. A un instant donné, une variable ne peut contenir qu'une seule valeur. Bien sûr, cette valeur pourra évoluer sous l'action de certaines instructions du programme.
Outre le nom et la valeur, une variable peut être caractérisée par son type. Le type d'une variable définit la nature des informations qui seront représentées dans les variables (numériques, caractères...).
Ce type implique des limitations concernant les valeurs qui peuvent être représentées. Il limite aussi les opérations réalisables avec les variables correspondantes. Ainsi, les opérations arithmétiques (addition, soustraction, multiplication, division) possibles des variables numériques, n'ont aucun sens pour des variables de type caractères. Par contre les comparaisons seront possibles pour les deux types.

2.2. Déclaration des variables

La première chose à faire tout au début de l'algorithme, avant de pouvoir utiliser des variables, c'est de faire la déclaration des variables.
Lorsqu'on déclare une variable, on lui attribue un nom et on lui réserve un emplacement mémoire. La taille de cet emplacement mémoire dépend du type de variable. C'est pour cette raison qu'on doit préciser lors de la déclaration le type du variable.
La syntaxe d'une déclaration de variable est la suivante :

VARIABLE nom : TYPE

ou

*VARIABLES nom1, nom2,... : T*YPE

2.3. Types de variables

2.3.1. Type numérique

Commençons par le cas très fréquent, celui d'une variable destinée à recevoir des nombres.
Généralement, les langages de programmation offrent les types suivants :

- ENTIER

Le type entier désigne l'ensemble des nombres entiers négatifs ou positifs dont les valeurs varient entre -32 768 à 32 767.
On écrit alors :

VARIABLES i, j, k : ENTIER

- REEL Le type réel comprend les variables numériques qui ont des valeurs réelles. La plage des valeurs du type réel est : $-3{,}40\text{x}10^{38}$ à $-1{,}40\text{x}10^{45}$ pour les valeurs négatives
$1{,}40\text{x}10^{-45}$ à $3{,}40\text{x}10^{38}$ pour les valeurs positives
On écrit alors :

VARIABLES x, y : REEL

Remarque : Le type de variable choisi pour un nombre va déterminer les valeurs maximales et minimales des nombres pouvant être stockés dans la variable. Elle détermine aussi la précision de ces nombres (dans le cas de nombres décimaux).

2.3.2. Type chaîne

En plus, du type numérique on dispose également du type chaîne (également appelé caractère ou alphanumérique).
Dans une variable de ce type, on stocke des caractères, qu'il s'agisse de lettres, de signes de ponctuation, d'espaces, ou même de chiffres. Le nombre maximal de caractères pouvant être stockés dans une seule variable chaîne dépend du langage utilisé.
On écrit alors :

VARIABLE nom, prenom, adresse : CHAINE

Une chaîne de caractères est notée toujours soit entre guillemets, soit entre des apostrophes.
Cette notation permet d'éviter les confusions suivantes :

- Confondre un chiffre et une suite de chiffres. Par exemple, 423 peut représenter le nombre 423 (quatre cent vingt-trois), ou la suite de caractères 4, 2, et 3.
- La confusion qui consiste à ne pas pouvoir faire la distinction entre le nom d'une variable et son contenu.

Remarque : Pour les valeurs des variables de type chaîne, il faut respecter la casse. Par exemple, la chaîne ‹Salut› est différente de la chaîne ‹salut›.

2.3.3. Type booléen

Dans ce type de variables on y stocke uniquement des valeurs logiques VRAI ou FAUX, TRUE ou FALSE, 0 ou 1.
On écrit alors :

VARIABLE etat : BOOLEEN

2.3.4. Opérateurs et expressions

2.3.4.1. Opérateurs

Un opérateur est un signe qui relie deux variables pour produire un résultat.
Les opérateurs dépendent des types de variables mis en jeu.
Pour le type numérique on a les opérateurs suivants :

+ : Addition
- : Soustraction
***** : Multiplication
/ : Division
A : Puissance

Tandis que pour le type chaîne, on a un seul opérateur qui permet de concaténer deux chaînes de caractères. Cet opérateur de concaténation est noté &.
Par exemple : la chaîne de caractères ″Salut″ concaténer à la chaîne ″tout le monde″ donne comme résultat la chaîne ″Salut tout le monde″.

2.3.4.2. Expressions

Une expression est un ensemble de variables (ou valeurs) reliées par des opérateurs et dont la valeur du résultat de cette combinaison est unique.
Par exemple :

7 5+4 x + 15 – y/2 nom & prenom

où x et y sont des variables numériques (réels ou entiers) et nom et prenom sont des variables chaîne.

Dans une expression où on y trouve des variables ou valeurs numériques, l'ordre de priorité des opérateurs est important. En effet, la multiplication et la division sont prioritaires par rapport à l'addition et la soustraction.
Par exemple, 12 * 3 + 5 donne comme résultat 41.
Si l'on veut modifier cette ordre de priorité on sera obligé d'utiliser les parenthèse.
Par exemple, 12 * (3 + 5) donne comme résultat 96.

2.3.5. L'instruction d'affectation

L'instruction d'affection est opération qui consiste à attribuer une valeur à une variable. On la notera avec le signe ←.
Cette instruction s'écrit :

VARIABLE ← valeur

Par exemple : MONTANT ← 3500.
On dit qu'on affecte (ou on attribue) la valeur 3500 à la variable numérique MONTANT.
Si dans une instruction d'affectation, la variable à laquelle on affecte la valeur et la valeur affectée ont des types différents, cela provoquera une erreur.
On peut aussi attribuer à une variable la valeur d'une variable ou d'une expression de façon générale.
On écrit :

VARIABLE ← EXPRESSION

Par exemple :

A ← B
A ← B * 2 + 5

Dans ce cas, l'instruction d'affectation sera exécutée en deux temps :
- D'abord, on calcule la valeur de l'expression
- On affecte la valeur obtenue à la variable à gauche.

On peut même avoir des cas où la variable de gauche qui figure dans l'expression à droite.
Par exemple :

A ← A + 5

Dans cette exemple, après l'exécution de l'instruction d'affectation la valeur de la variable A sera augmenter de 5.

Remarque :
Dans une instruction d'affection on a toujours :
- à gauche de la flèche d'affection un nom de variable
- à droite de la flèche d'affectation une valeur ou une expression
- l'expression à droite de la flèche doit être du même type que la variable située à gauche.

Si dans une instruction d'affectation une ces points n'est pas respecté, cela engendra une erreur.
Il est à noter que l'ordre dans lequel sont écrites les instructions est essentiel dans le résultat final.

Exemple :

CAS I	**CAS II**
A ← 15	A ← 30
A ← 30	A ← 15

Après exécution des deux instructions d'affection, la valeur de A sera :
- Cas I : 30
- Cas II : 15

Exercices
1. Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

```
Variables A, B : Entier
Début
    A ← 1
    B ← A + 3
    A ← 3
Fin
```

2. Quelles seront les valeurs des variables A, B et C après exécution des instructions suivantes ?

```
Variables A, B, C : Entier
Début
    A ← 5
    B ← 3
    C ← A + B
    A ← 2
    C ← B – A
Fin
```

3. Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

```
Variables A, B : Entier
Début
    A ← 5
    B ← A + 4
    A ← A + 1
    B ← A – 4
Fin
```

4. Quelles seront les valeurs des variables A, B et C après exécution des instructions suivantes ?

```
Variables A, B, C : Entier
Début
    A ← 3
    B ← 10
    C ← A + B
    B ← A + B
    A ← C
Fin
```

5. Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

```
Variables A, B : Entier
Début
    A ← 5
    B ← 2
    A ← B
    B ← A
Fin
```

Questions: les deux dernières instructions permettent-elles d'échanger les deux valeurs de B et A ? Si on inverse les deux dernières instructions, cela change-t-il quelque chose

?

6. Ecrire un algorithme permettant d'échanger les valeurs de deux variables A et B, et ce quel que soit leur contenu préalable.

7. On dispose de trois variables A, B et C. Ecrivez un algorithme transférant à B la valeur de A, à C la valeur de B et à A la valeur de C (toujours quels que soient les contenus préalables de ces variables).

8. Que produit l'algorithme suivant ?

```
Variables A, B, C : Caractères
    Début
        A ← "423"
        B ← "12"
        C ← A + B
    Fin
```

9. Que produit l'algorithme suivant ?

```
Variables A, B : Caractères
    Début
        A ← "423"
        B ← "12"
        C ← A & B
    Fin
```

Solutions

1.

Après exécution de l'instruction	**La valeur des variables est :**
A ← 1	A = 1 B = ?
B ← A + 3	A = 1 B = 4
A ← 3	**A = 3 B = 4**

2.

Après exécution de l'instruction	**La valeur des variables est :**
A ← 5	A = 5 B = ? C = ?
B ← 3	A = 5 B = 3 C = ?
C ← A + B	A = 5 B = 3 C = 8
A ← 2	A = 2 B = 3 C = 8
C ← B – A	**A = 2 B = 3 C = 1**

3.

Après exécution de l'instruction	**La valeur des variables est :**
A ← 5	A = 5 B = ?
B ← A + 4	A = 5 B = 9
A ← A + 1	A = 6 B = 9
B ← A – 4	**A = 6 B = 2**

Après exécution de l'instruction	**La valeur des variables est :**
A ← 3	A = 3 B = ? C = ?
B ← 10	A = 3 B = 10 C = ?

C ← A + B	A = 3 B = 10 C = 13
B ← A + B	A = 3 B = 13 C = 13
A ← C	**A = 13 B = 13 C = 13**

5.

Après exécution de l'instruction	**La valeur des variables est :**
A ← 5	A = 5 B = ?
B ← 2	A = 5 B = 2
A ← B	A = 2 B = 2
B ← A	**A = 2 B = 2**

Les deux dernières instructions ne permettent donc pas d'échanger les deux valeurs de B et A, puisque l'une des deux valeurs (celle de A) est ici écrasée.
Si l'on inverse les deux dernières instructions, cela ne changera rien du tout, hormis le fait que cette fois c'est la valeur de B qui sera écrasée.

6. L'algorithme est :

Début
C ← A
A ← B
B ← C
Fin

On est obligé de passer par une variable dite temporaire (la variable C).

7. L'algorithme est :

Début
D ← C
C ← B
B ← A
A ← D
Fin

En fait, quel que soit le nombre de variables, une seule variable temporaire suffit.

8. Il ne peut produire qu'une erreur d'exécution, puisqu'on ne peut pas additionner des caractères.

9. On peut concaténer ces variables. A la fin de l'algorithme, C vaudra donc "42312".

3. LES INSTRUCTIONS DE LECTURE ET ECRITURE

Considérons le programme suivant :

VARIABLE A : **ENTIER**
Début
A ← 12 ^ 2
Fin

Il permet de calculer le carré de 12.
Le problème de ce programme, c'est que, si l'on veut calculer le carré d'un autre nombre que 12, il faut réécrire le programme.

D'autre part, la machine calcule le résultat et l'utilisateur qui fait exécuter ce programme ne saura jamais que ce résultat correspond au carré de 12.
C'est pour cela qu'il faut introduire des instructions qui permettent le dialogue avec l'utilisateur.
En effet, il existe une instruction qui permet à l'utilisateur de faire entrer des valeurs au clavier pour qu'elles soient utilisées par le programme. La syntaxe de cette instruction de lecture est :

LIRE NomVariable

Lorsque le programme rencontre une instruction LIRE, l'exécution du programme s'interrompt,

attendant la saisie d'une valeur au clavier.

Dès que l'on frappe sur la touche ENTER, l'exécution reprend.
Une autre instruction permet au programme de communiquer des valeurs à l'utilisateur en les affichant à l'écran. La syntaxe de cette instruction d'écriture est :

ECRIRE NomVariable

ou de façon générale

ECRIRE Expression

Remarque : Avant de lire une variable, il est fortement conseillé d'écrire des libellés à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper. La même chose pour l'instruction d'écriture.

Exemple :

```
Variables A, CARRE : Réels
DEBUT
    Ecrire 'Entrez un nombre'
    Lire A
    CARRE ← A * A
    Ecrire 'Le carré de ce nombre est : '
    Ecrire CARRE
FIN
```

Exercices

1. Quel résultat produit le programme suivant ?

```
VARIABLES Val, Double : ENTIERS
Début
    Val ← 231
    Double ← Val * 2
    ECRIRE Val
    ECRIRE Double
Fin
```

2. Ecrire un programme qui demande deux nombres entiers à l'utilisateur, puis qui calcule et affiche le somme de ces nombres.

3. Ecrire un programme qui lit le prix HT d'un article, le nombre d'articles et le taux de TVA, et qui fournit le prix total TTC correspondant. Faire en sorte que des libellés apparaissent clairement.

4. Ecrire un programme qui lit une valeur et qui nous calcule l'inverse de cette valeur.

5. Le surveillant général d'un établissement scolaire souhaite qu'on lui écrit un programme qui calcule, pour chaque élève, la moyenne des notes des cinq matières. Ces matières sont avec leur coefficient :

MATIERE	COEFFICIENT
Math	5

Physique	5
Français	4
Anglais	2
Histoire - Géographie	2

Corrections

1. On verra apparaître à l'écran :

```
231
462
```

2. Le programme est :

```
VARIABLES A, B, SOMME : ENTIERS
Début
    ECRIRE 'Entrez le premier nombre'
    Lire A
    ECRIRE 'Entrez le deuxième nombre'
    Lire B
    SOMME ← A + B
    ECRIRE 'La somme de ces deux nombres est : '
    ECRIRE SOMME
Fin
```

Remarque : On peut remplacer les deux derniers lignes par :

```
    ECRIRE 'La somme de ces deux nombres est : ', SOMME
```

3. Le programme est :

```
VARIABLES pht, ttva, pttc : REELS
VARIABLE nb : ENTIER
    Début
        ECRIRE "Entrez le prix hors taxes :"
        LIRE pht
        ECRIRE "Entrez le nombre d'articles :"
        LIRE nb
        ECRIRE "Entrez le taux de TVA :"
        LIRE ttva
        Pttc ← nb * pht * (1 + ttva)
        ECRIRE "Le prix toutes taxes est : ", ttva
    Fin
```

4. Le programme est :

```
VARIABLES x, inverse : REELS
    Début
        ECRIRE "Entrez une valeur :"
        LIRE x
        inverse ← 1 / x
        ECRIRE "L'inverse est : ", inverse
    Fin
```

5. Le programme est :

```
VARIABLES mat, phy, ang, fra, hg, moyenne : REELS
    Début
        ECRIRE "Entrez la note de math :"
```

```
        LIRE mat
        ECRIRE "Entrez la note de physique :"
        LIRE phy
        ECRIRE "Entrez la note de français :"
        LIRE fra
        ECRIRE "Entrez la note d'anglais :"
        LIRE ang
        ECRIRE "Entrez la note d'histoire-Géo :"
        LIRE hg
        moyenne ← ((mat + phy) * 5 + fra * 4 + (ang
+ hg) * 2) / 18
        ECRIRE "La moyenne est : ", moyenne
Fin
```

4. LA STRUCTURE ALTERNATIVE

4.1. Les conditions simples

Une condition simple consiste en une comparaison entre deux expressions du même type.
Cette comparaison s'effectue avec des opérateurs de comparaison. Voici la liste de ces opérateurs accompagnés de leur signification dans le cas des types numérique ou chaîne :

Opérateur	Signification numérique	Signification chaîne
=	égal à	égal à
< >	différent	différent
<	inférieur	placé avant dans l'ordre alphabétique
>	supérieur	placé après dans l'ordre alphabétique
<=	inférieur ou égal	placé avant dans l'ordre alphabétique ou égal
>=	supérieur ou égal	placé après dans l'ordre alphabétique ou égal

Pour la comparaison du type chaîne c'est l'ordre alphabétique qu'est utilisé dans le cas où l'on compare deux lettres majuscules ou minuscules. Mais si l'on compare majuscules et minuscules, il faut savoir que les majuscules apparaissent avant les minuscules. Ainsi, par exemple : "M" < "m".

4.2. Les conditions complexes

Certains problèmes exigent parfois de formuler des conditions qui ne peuvent pas être exprimées sous la forme simple vu en dessus. A cet effet, la plupart des langages autorisent des conditions formées de plusieurs conditions simples reliées entre elles par ce qu'on appelle des opérateurs logiques. Ces opérateurs sont : **ET**, **OU** et **NON**.

- Pour que la condition complexe,

  condition] **ET** *condition2*

soit VRAI, il faut impérativement que la *condition]* soit VRAI et que la *condition2* soit VRAI.

- Pour que la condition

  condition] **OU** *condition2*

soit VRAI, il suffit que *condition]* soit VRAI ou *condition2* soit VRAI. Il est à noter que cette condition complexe sera VRAI si *condition]* et *condition2* sont VRAI.

- Le NON inverse une condition :

  NON(*condition*)

est VRAI si condition est FAUX, et il sera FAUX si condition est VRAI.

D'une manière générale, les opérateurs logiques peuvent porter, non seulement sur des conditions simples, mais aussi sur des conditions complexes. L'usage de parenthèses permet dans de tels cas de régler d'éventuels problèmes de priorité. Par exemple, la condition :

(a < 0 **ET** b > 1) **OU** (a > 0 **ET** b > 3)

est VRAI si l'une au moins des conditions entre parenthèses est VRAI.

4.3. La structure alternative

Supposons que nous avons besoin, dans un programme, d'afficher un message précisant que la valeur d'une variable est positive ou négative. Avec les instructions de base que nous avons vu (celles qui permettent la manipulation des variables : affectation, lecture, écriture), on ne peut pas. Il faut introduire une des instructions de structuration du programme (ces instructions servent à préciser comment doivent s'enchaîner chronologiquement les instructions de base) qui donne la possibilité d'effectuer des choix dans le traitement à réaliser. Ces instructions s'appellent structure alternative. Sa syntaxe est :

SI *condition* **ALORS**
bloc 1 d'instructions
SINON
bloc 2 d'instructions
FIN SI

Si la condition mentionnée après **SI** est VRAI, on exécute le *bloc1 d'instructions* (ce qui figure après le mot **ALORS**); si la condition est fausse, on exécute le *bloc2 d'instructions* (ce qui figure après le mot **SINON**).

Exemple :

SI a > 0 **ALORS**
ECRIRE "valeur positive"
SINON
ECRIRE "valeur négative"
FIN SI

Dans ce programme, on vérifie si la valeur de a est supérieure à 0, on affichera le message "valeur positive". Dans le cas contraire, il sera affiche le message "valeur négative".

La structure alternative peut prendre une autre forme possible où l'une des parties du choix est absente. Elle s'écrit dans ce cas :

SI condition ALORS
bloc d'instructions
FIN SI

Exemple : Dans un programme de calcul du montant d'une facture, on applique une remise de 1% si le montant dépasse 5000 Dhs. Nous écrirons :

SI montant > 5000 **ALORS**
montant ← montant * 0.99
FIN SI

4.4. Les structures alternatives imbriquées

Il peut arriver que l'une des parties d'une structure alternative contienne à son tour une structure alternative. Dans ce cas, on dit qu'on a des structures alternatives imbriquées les unes dans les autres.

Exemple : Ecrire un programme qui donne l'état de l'eau selon sa température.

Variable Temp : **Entier**
Début
Ecrire "Entrez la température de l'eau :"
Lire Temp
Si Temp =< 0 **Alors**

```
                    Ecrire "C'est de la glace"
            Sinon
            Si Temp < 100 Alors
                    Ecrire "C'est du liquide"
            Sinon
                    Ecrire "C'est de la vapeur"
            Finsi
            FinSi
Fin
```

On peut aussi écrire :

```
Variable Temp : Entier
Début
Ecrire "Entrez la température de l'eau :"
Lire Temp
Si Temp =< 0 Alors
   Ecrire "C'est de la glace"
Finsi
Si Temp > 0 Et Temp < 100 Alors
   Ecrire "C'est du liquide"
Finsi
Si Temp > 100 Alors
   Ecrire "C'est de la vapeur"
Finsi
Fin
```

La première version est plus simple à écrire et plus lisible. Elle est également plus performante à l'exécution. En effet, les conditions se ressemblent plus ou moins, et surtout on oblige la machine à examiner trois tests successifs alors que tous portent sur la même chose, la valeur de la variable Temp. Mais aussi, et surtout, nous avons fait des économies sur le temps d'exécution de l'ordinateur. Si la température est inférieure à zéro, celui-ci écrit « C'est de la glace » et passe directement à la fin, sans être ralenti par l'examen des autres possibilités.

4.5. Autre forme

Dans des langages de programmation, la structure alternative peut prendre une autre forme qui permet d'imbriquée plusieurs. Sa syntaxe est :

```
SELON expression
        valeur1 : action1
        valeur2 : action2

        ...
        valeurN : actionN
        SINON : action
FIN SELON
```

Si *expression* est égale à *valeuri*, on exécute actioni et on passe à la suite de l'algorithme. Sinon on exécute action et on passe à la suite de l'algorithme.

Exercices

1. Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si leur produit est négatif ou positif (on laisse de côté le cas où le produit est nul). Attention toutefois : on ne doit pas calculer le produit des deux nombres.

2. Ecrire un algorithme qui demande trois noms à l'utilisateur et l'informe ensuite s'ils sont rangés ou non dans l'ordre alphabétique.

3. Ecrire un algorithme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou négatif (on inclut cette fois le traitement du cas où le nombre vaut zéro).

4. Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si le produit est négatif ou positif (on inclut cette fois le traitement du cas où le produit peut être nul). Attention toutefois, on ne doit pas calculer le produit !

5. Ecrire un algorithme qui demande l'âge d'un enfant à l'utilisateur. Ensuite, il l'informe de sa catégorie :
- « Poussin >> de 6 à 7 ans
- « Pupille >> de 8 à 9 ans
- « Minime >> de 10 à 11 ans
- « Cadet >> après 12 ans

6. a partir d'un montant lu, on détermine un montant net par application d'une remise de :
- 1% si le montant est compris entre 2000 et 5000 Dhs (valeurs comprises) - 2 % si le montant est supérieur à 5000 Dhs.

Solutions

1. Le programme est :

```
Variables m, n : Entier
Début
    Ecrire "Entrez deux nombres : "
    Lire m, n
    Si m * n > 0 Alors
        Ecrire "Leur produit est positif"
    Sinon
        Ecrire "Leur produit est négatif"
    Finsi
Fin
```

2. Le programme est :

```
Variables a, b, c : Caractère
Début
    Ecrire "Entrez successivement trois noms : "
    Lire a, b, c
    Si a < b et b < c Alors
        Ecrire "Ces noms sont classés alphabétiquement"
    Sinon
        Ecrire "Ces noms ne sont pas classés"
    Finsi
Fin
```

3. Le programme est :

```
Variable n : Entier
Début
    Ecrire "Entrez un nombre : "
    Lire n
    Si n < 0 Alors
        Ecrire "Ce nombre est négatif"
    SinonSi n = 0 Alors
        Ecrire "Ce nombre est nul"
    Sinon
        Ecrire "Ce nombre est positif"
```

```
        Finsi
Fin
```

4. Le programme est :

```
Variables m, n : Entier
Début
    Ecrire "Entrez deux nombres : "
    Lire m, n
    Si m = 0 OU n = 0 Alors
        Ecrire "Le produit est nul"
    SinonSi (m < 0 ET n < 0) OU (m > 0 ET n > 0) Alors
        Ecrire "Le produit est positif"
    Sinon
        Ecrire "Le produit est négatif"
    Finsi
Fin
```

5. Le programme est :

```
Variable age : Entier
Début
    Ecrire "Entrez l'âge de l'enfant : "
    Lire age
    Si age >= 12 Alors
        Ecrire "Catégorie Cadet"
    SinonSi age >= 10 Alors
        Ecrire "Catégorie Minime"
    SinonSi age >= 8 Alors
        Ecrire "Catégorie Pupille"
    SinonSi age >= 6 Alors
        Ecrire "Catégorie Poussin"
    Finsi
Fin
```

6. Le programme est :

```
Variables montant , taux , remise : Réels
Début
    Ecrire "Entrez le montant : "
    Lire montant
      Si montant < 2000 Alors
            taux ← 0
    Sinon
      Si montant ≤ 5000 Alors
            taux ← 1
    Sinon
          taux ← 2
    Fin SI
    Fin Si
    Montant ← montant * (1 – taux / 100)
    Ecrire "Le montant net est : " , montant

Fin
```

5. LES STRUCTURES REPETITIVES

Reprenons le programme du surveillant général qui calcule la moyenne des notes. L'exécution de ce programme fournit la moyenne des notes uniquement pour un seul élève. S'il l'on veut les moyennes de 200 élèves, il faut ré exécuter ce programme 200 fois. Afin d'éviter cette tâche fastidieux d'avoir ré exécuter le programme 200 fois, on peut faire recourt à ce qu'on appelle les **structures répétitives**. On dit aussi les **structures itératives** ou **boucles**.
Une structure répétitive sert à répéter un ensemble d'instructions. Il existe trois formes de structures répétitives : **POUR**, **TANT QUE**, **REPETER**.

5.1. La structure POUR

Cette structure permet de répéter des instructions un nombre connu de fois. Sa syntaxe est :

POUR *compteur* = val_*initial* **A** *val_final* **PAS DE** *incrément*
Instructions à répéter
FIN POUR

compteur c'est ce qu'on appelle **compteur**. C'est une variable de type entier.
val_*initial* et *val_final* sont respectivement les valeur initiale et final prise par le compteur. Ce sont des valeurs entières.
incrément est la valeur d'augmentation progressive du compteur. La valeur par défaut du pas est de 1. Dans de telle on peut ne pas le préciser.

Remarques :
Pour un pas positif, la valeur négative doit être inférieure à la valeur finale. Pour un pas négatif, valeur négative doit être supérieure à la valeur finale.
Si la valeur initiale est égale à la valeur finale, la boucle sera exécutée une seule fois.

Exemple : Réécrivons le programme du surveillant général de façon qu'il puisse calculer les moyennes de 200 élèves.

```
VARIABLES mat, phy, ang, fra, hg, moyenne : REELS

VARIABLE i : ENTIER
    Début
        POUR i = 1 A 200
            ECRIRE "Entrez la note de math :"
            LIRE mat
            ECRIRE "Entrez la note de physique :"
            LIRE phy
            ECRIRE "Entrez la note de français :"
            LIRE fra
            ECRIRE "Entrez la note 'anglais :"
            LIRE ang
            ECRIRE "Entrez la note d'histoire-Géo :"
            LIRE hg
            moyenne ← ((mat + phy) * 5 + fra * 4 + (ang
            + hg) * 2) / 18
            ECRIRE "La moyenne est : ", moyenne
        FIN POUR
    Fin
```

Exercices
1. Ecrire un algorithme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit (cas où l'utilisateur entre le nombre 7) :

Table de 7 :

7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
...
7 x 10 = 70

2. Ecrire un algorithme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre. Par exemple, si l'on entre 5, le programme doit calculer :
1 + 2 + 3 + 4 + 5 = 15

3. Ecrire un algorithme qui demande un nombre de départ, et qui calcule sa factorielle.
NB : la factorielle de 8, notée 8 ! vaut 1 x 2 x 3 x 4 x 5 x 6 x 7 x 8

4. Ecrire un algorithme qui demande successivement 20 nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces 20 nombres :

Entrez le nombre numéro 1 : 12
Entrez le nombre numéro 2 : 14
...
Entrez le nombre numéro 20 : 6

Le plus grand de ces nombres est : 14

Modifiez ensuite l'algorithme pour que le programme affiche de surcroît en quelle position avait été saisie ce nombre :
C'était le nombre numéro 2

5. Ecrire un algorithme qui :
- lit d'abord une valeur
- ensuite il va lire successivement 20 nombres.
- enfin il va déterminer combien de fois la première valeur a été saisie (sans compter la première saisie).

Solutions

1. Le programme est :

```
Variables i , valeur : Entiers
DEBUT
        Lire valeur
        POUR i = 1 A valeur
                Ecrire valeur & " X " & i & " = " & valeur * i
        FIN POUR
FIN
```

2. Le programme est :

```
Variables i , valeur , somme : Entiers
DEBUT
        Lire  valeur  somme
        ← 0  POUR  i = 1  A
        valeur
                somme ← somme + i
        FIN POUR
        Ecrire "La somme des " & valeur & " premiers entiers est : " & somme
```

3. Le programme est :

```
Variables i , valeur , factoriel : Entiers
DEBUT
        Lire valeur
        factoriel ← 1
        POUR i = 1 A valeur
                factoriel ← factoriel * i
        FIN POUR
        Ecrire "Le factoriel de " & valeur & " est : " & factoriel
Fin
```

4. Le programme est :

```
Variables i , a , max , pmax : Entiers
DEBUT
        Ecrire << Entrez le nombre numéro 1 >>
        Lire a
        max ← a
        pmax ← 1
        POUR i = 2 A 20
                Ecrire << Entrez le nombre numéro >> , i
                Lire a
                SI a > max ALORS
                        max ← a
                        pmax ← i
                FIN SI
        FIN POUR
        Ecrire << Le plus grand nombre est : >> , max
        Ecrire << Sa position est : >> , pmax
FIN
```

5. Le programme est :

```
Variables i , a , b , S : Entiers
DEBUT
        Ecrire << Entrez un chiffre : >>
        Lire a
        S ← 0
        POUR i = 1 A 20
                Ecrire << Entrez un nombre : >>
                Lire b
                SI a = b ALORS
                        S ← S + 1
                FIN SI
        FIN POUR
        Ecrire << Le nombre de fois de saisie de >> , a , << est : >> , S
FIN
```

5.2. La structure TANT QUE

Cette structure permet de répéter les instructions **tant qu**'une condition est satisfaite. Sa syntaxe est :

```
TANT QUE condition
        Instructions à répéter
FIN TANT QUE
```

condition c'est une condition qu'on appelle parfois condition d'arrêt. Cette condition est testée avant la première exécution.
Cette structure diffère de la première par le fait qu'on va répéter des instructions pour un nombre de fois inconnu au préalable.

Exemple : Reprenant toujours le programme de notre surveillant. S'il ne sait pas combien de moyennes à calculer on ne pourra pas utiliser la structure **POUR**. Dans ce cas on est obligé d'utiliser la structure **TANT QUE**. Le programme sera alors :

```
Variables mat, phy, ang, fra, hg, moyenne : Réels
Variable reponse : Chaîne
DEBUT
        reponse ← "o"
        TANT QUE reponse = "o"
                Ecrire "Entrez la note de math :"
                Lire mat
                Ecrire "Entrez la note de physique :"
                Lire phy
                Ecrire "Entrez la note de français :"
                Lire fra
                Ecrire "Entrez la note d'anglais :"
                Lire ang
                Ecrire "Entrez la note d'histoire-Géo :"
                Lire hg
                moyenne ← ((mat + phy) * 5 + fra * 4 + (ang + hg) * 2) / 18
                Ecrire "La moyenne est : ", moyenne
                Ecrire "Voulez-vous continuer oui (o) /non (n) ?"
                Lire reponse
        FIN TANT QUE
FIN
```

Exercices

1. Ecrire un algorithme qui demande à l'utilisateur un nombre compris entre 1 et 3 jusqu'à ce que la réponse convienne.

2. Ecrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : << Plus petit ! », et inversement, << Plus grand ! » si le nombre est inférieur à 10.

3. Ecrire un algorithme qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

4. Ecrire un algorithme qui demande successivement des nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces nombres et quel était sa position. La saisie des nombres s'arrête lorsque l'utilisateur entre un zéro.

5. Lire la suite des prix (en dhs entiers et terminée par zéro) des achats d'un client. Calculer la somme qu'il doit, lire la somme qu'il paye, et déterminer le reste à rendre.

Solutions

1. Le programme est :

```
        Variable a : Réel
        Début
```

```
        Tant Que a < 1 OU a > 3
                Ecrire << Veuillez Saisir une valeur comprise entre 1 et 3 >>
                Lire a
        Fin Tant Que
Fin
```

2. Le programme est :

```
Variable a : Réel
Début
        Lire a
        Tant Que a < 10 OU a > 20
                Si a < 10 Alors
                        Ecrire << Plus grand ! >>
                Sinon
                        Ecrire << Plus petit ! >>
                                Fin Si
                                Lire a
        Fin Tant Que
Fin
```

3. Le programme est :

```
Variable a , i : Réel
Début
        Ecrire << Entrez un nombre >>
        Lire a
        i ← a + 1
        Tant Que i < a + 10
                Ecrire i
                i ← i + 1
        Fin Tant Que
Fin
```

4. Le programme est :

```
Variables i , a , max , pmax : Entiers
DEBUT
        Ecrire << Entrez le nombre numéro 1 >>
        Lire a
        max ← a
        pmax ← 1
        i ← 1
        TANT QUE a <> 0
                i ← i + 1
                Ecrire << Entrez le nombre numéro >> , i
                Lire a
                SI a > max ALORS
                        max ← a
                        pmax ← i
                FIN SI
        FIN TANT QUE
        Ecrire << Le plus grand nombre est : >> , max
        Ecrire << Sa position est : >> , pmax
FIN
```

5. Le programme est : **Variables** prixlu , mdu , mpaye , reste : **Entiers**

```
DEBUT
    Ecrire << Entrez le prix >>
    Lire prixlu
    mdu ← 0
    mdu ← mdu + prixlu
    TANT QUE prixlu <> 0
        Ecrire << Entrez le prix >>
        Lire prixlu
        mdu ← mdu + prixlu
    FIN TANT QUE
    Ecrire << Entrez le prix payé>>
    Lire mpaye
    reste ← mpaye - mdu
    Ecrire << Le reste est : >> , reste
FIN
```

5.3. La structure REPETER

Cette structure sert à répéter des instructions **jusqu'à** ce qu'une condition soit réalisée. Sa syntaxe est :

```
REPETER
    Instructions à répéter
JUSQU'A condition
```

Considérons le programme suivant :

```
Variables a , c : Entiers
DEBUT
    REPETER
        Lire a
        c ← c * c
        Ecrire c
    JUSQU'A a = 0
    Ecrire << Fin >>
FIN
```

Les mots **REPETER** et **JUSQU'A** encadrent les instructions à répéter. Cela signifie que ces instructions doivent être répéter autant de fois jusqu'à ce que la variable **a** prennent la valeur 0.
Notez bien que le nombre de répétition dans cette structure n'est indiqué explicitement comme c'est la cas de la structure TANT QUE. Il dépend des données que l'on fournit au programme.
Pour bien expliciter cela, voyons ce que produira ce programme si l'on lui fournit successivement les valeurs 2, 4, 0. Le résultat se présentera ainsi :

```
4
16
0
Fin
```

Exercices

1. Ecrire un algorithme qui demande successivement des nombres à l'utilisateur, et qui calcule le nombre de valeurs saisies. La saisie des nombres s'arrête lorsque l'utilisateur entre le caractère << n >> ou << N >>.

2. Ecrire un algorithme qui demande successivement des nombres à l'utilisateur, et qui calcule leur moyenne. La saisie des nombres s'arrête lorsque l'utilisateur entre un zéro.

3. Modifiez l'algorithme de l'exercice 1, de façon qu'il nous renseigne sur le nombre des valeurs positives et sur le nombre des valeurs négatives. Ne comptez pas les valeurs nuls.

4. Ecrire un algorithme qui lit les caractères saisies par l'utilisateur. A la fin ce programme nous affichera la phrase saisie. La saisie des caractères s'arrête lorsqu'on tape point << . >>. Pour l'utilisateur veut insérer un espace il lui suffit de tapez sur 0. Par exemple si l'utilisateur tape successivement les caractères << b >> , << o >>, << n >>, << j >>, << o >>, << u >>, << r >> , << t >>, << o >>, << u >>, << s >>, << . >> , il nous affichera la chaîne << bonjour tous >>.
Mais si il tape <<b>>,<<o>>,<<n>>,<<j>>,<<o>>,<<u>>,<<r>>,<<0>>,<<t>>,<<o>>,<<u>>,<<s>>,<<.>>, le programme affichera << bonjour tous >>.

Solutions

1. le programme est :

```
Variables a , compteur : Entiers
Variable reponse : Chaîne
DEBUT
        compteur ← 0
        REPETER
                Ecrire << Entrez un nombre : >>
                Lire a
                compteur ← compteur + 1
                Ecrire << Voulez-vous continuez Oui/Non ? >>
                Lire reponse
        JUSQU'A reponse = << N >> ou reponse = << n >>
        Ecrire << Le nombre de valeurs saisies est : >> , compteur
FIN
```

2. Le programme est :

```
Variables a , somme , moyenne , compteur : Entiers
DEBUT
        compteur ← 0
        somme ← 0
        REPETER
                Ecrire << Entrez un nombre : >>
                Lire a
                compteur ← compteur + 1
                somme ← somme + a
        JUSQU'A a = 0
        Moyenne ← somme / compteur
        Ecrire << La moyenne de valeurs saisies est : >> , moyenne
FIN
```

3. le programme est :

```
Variables a , npos , nneg : Entiers
Variable reponse : Chaîne
DEBUT
        npos ← 0
        nneg ← 0
        REPETER
                Ecrire << Entrez un nombre : >>
                Lire a
                SI a > 0 ALORS
                        npos ← npos + 1
                SINON
                        SI a < 0 ALORS
                                nneg ← nneg + 1
                        FIN SI
```

```
            FIN SI
            Ecrire << Voulez-vous continuez Oui/Non ? >>
            Lire reponse
        JUSQU'A reponse = << O >> ou reponse = << o >>
        Ecrire << Le nombre de valeurs positives saisies est : >> , npos
        Ecrire << Le nombre de valeurs négatives saisies est : >> , nneg
FIN
```

4. Le programme est :

```
Variables caractere , phrase : Chaînes
DEBUT
        phrase←<<>>
        REPETER
            Ecrire << Entrez une caractère : >>
            Lire caractère
            SI caractere = << 0 >> ALORS
                caractere ← <<>>
            FIN SI
            phrase ← phrase +caractere
        JUSQU'A caractere = << . >>
        Ecrire << La phrase résultante est : >> , phrase
FIN
```

6. LES TABLEAUX

6.1. Les tableaux à une seul dimension

Imaginez que l'on veuille calculer la moyenne des notes d'une classe d'élèves. Pour l'instant on pourrait l'algorithme suivant :

```
Variables somme, nbEleves, Note, i : Réels
DEBUT
        somme ← 0
        Ecrire " Nombre d'eleves :"
        Lire nbEleves
        POUR i = 1 A nbEleves
            Ecrire " Note de l'eleve numero ", i , " : "
            Lire Note
            somme ← somme + Note
        FIN POUR
        Ecrire " La moyenne est de :", somme / nbEleves
FIN
```

Si l'on veut toujours calculer la moyenne des notes d'une classe mais en gardant en mémoire toutes les notes des élèves pour d'éventuels calculs (par exemple calculer le nombre d'élèves qui ont des notes supérieurs à la moyenne). Dans ce cas il faudrait alors déclarer autant de variables qu'il y a d'étudiants. Donc, si l'on a 10 élèves il faut déclarer 10 variables et si l'on a N il faut déclarer N variables et c'est pas pratique. Ce qu'il faudrait c'est pouvoir par l'intermédiaire d'une seule variable stocker plusieurs valeurs de même type et c'est le rôle des tableaux.

Un **tableau** est un ensemble de valeurs de même type portant le même nom de variable. Chaque valeur du tableau est repérée par un nombre appelé **indice**.

Les tableaux c'est ce que l'on nomme un **type complexe** en opposition aux types de données simples vus précédemment. La déclaration d'un tableau sera via la syntaxe suivante dans la partie des déclarations :

Tableau *nomtableau* (*nombre*) : **Type**

nom_tableau : désigne le nom du tableau

nombre : désigne le nombre d'éléments du tableau. On dit aussi sa taille
Type : c'est le type du tableau autrement dit le type de tous ces éléments

Exemples :

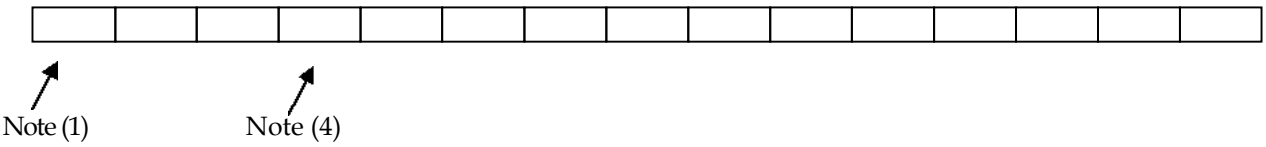
Tableau Note (20) : **Réel**

Note (20) est un tableau qui contient vingt valeurs réelles.

Tableau nom (10) , prenom (10) : **Chaîne**

Nom (10) et prenom (10) sont deux tableaux de 10 éléments de type chaîne.

Un tableau peut être représenté graphiquement par (exemple Note (15)) :



Si l'on veut accéder (en lecture ou en écriture) à la i ème valeur d'un tableau en utilisant la syntaxe suivante :

nom_tableau (*indice*)

Par exemple si X est un tableau de 10 entiers :

- X (2) ← - 5
  met la valeur -5 dans la 2 ème case du tableau
- En considérant le cas où *a* est une variable de type Entier, a ← X (2)
  met la valeur de la 2 ème case du tableau tab dans a, c'est- à- dire 5
- **Lire** X (1)
  met l'entier saisi par l'utilisateur dans la première case du tableau
- **Ecrire** X (1)
  affiche la valeur de la première case du tableau

Remarques :

- Un tableau possède un nombre maximal d'éléments défini lors de l'écriture de l'algorithme (les bornes sont des constantes explicites, par exemple 10, ou implicites, par exemple MAX). Ce nombre d'éléments ne peut être fonction d'une variable.
- La valeur d'un indice doit toujours :
  - être un nombre entier
  - être inférieure ou égale au nombre d'éléments du tableau

Exercices

1. Considérons les programmes suivants:

```
Tableau X (4) : Entier
DEBUT
        X (1)← 12
        X (2)← 5
        X (3)← 8
        X (4)←20
FIN

Tableau voyelle (6) : Chaîne
DEBUT
        voyelle (1) ← << a >>
        voyelle (2) ← << e >>
        voyelle (3) ← << i >>
        voyelle (4)← << o >>
```

```
        voyelle (5) ← « u »
        voyelle (6) ← « y »
FIN
```

Donner les représentations graphiques des tableaux X (4) et voyelle (6) après exécution de ces programmes.

2. Quel résultat fournira l'exécution de ce programme :

```
Variable i : Entier
Tableau C (6) : Entier
DEBUT
        POUR i = 1 A 6
                Lire C (i)
        FIN POUR
        POUR i = 1 A 6
                C (i) ← C (i) * C (i)
        FIN POUR
        POUR i = 1 A 6
                Ecrire C (i)
        FIN POUR
FIN
```

Si on saisit successivement les valeurs : 2 , 5 , 3 , 10 , 4 , 2.

3. Que fournira l'exécution de ce programme :

```
Tableau suite (8) : Entier
Variable i : Entier
DEBUT
        Suite (1) ← 1
        Suite (2) ← 1
        POUR i = 3 A 8
                suite (i) ← suite (i - 1) + suite (i - 2)
        FIN POUR
        POUR i = 1 A 8
                Ecrire suite (i)
        FIN POUR
FIN
```

4. Soit T un tableau de vingt éléments de types entiers. Ecrire le programme qui permet de calculer la somme des éléments de ce tableau.

5. Soit T un tableau de N entiers. Ecrire l'algorithme qui détermine le plus grand élément de ce tableau.

6. Ecrire un programme qui permet de lire 100 notes et de déterminer le nombre de celles qui sont supérieures à la moyenne.

7. Soit T un tableau de N entiers. Ecrire l'algorithme qui détermine simultanément la position du plus grand élément et la position du plus petit élément du tableau.

8. Soit T un tableau de N réels. Ecrire le programme qui permet de calculer le nombre des occurrences d'un nombre X (c'est-à-dire combien de fois ce nombre X figure dans le tableau T).

9. On dispose des notes de 25 élèves ; chaque élève peut avoir une ou plusieurs notes mais toujours au moins une. Ecrire un programme permettant d'obtenir la moyenne de chaque élève lorsqu'on lui fournit les notes. On veut que les données et les résultats se présentent ainsi :

Notes de l'élève numéro 1 *12*
12
-1
Notes de l'élève numéro 2

Notes de l'élève numéro 25 *15*
-1.....
Moyennes
Elève numéro 1 : 11

Elève numéro 25 : 15 Moyenne de la classe : 12.3
.....

Les parties italiques correspondent aux données tapées par l'utilisateur. La valeur -1 sert de critère de fin de notes pour chaque élève.

Solutions

1. La représentation graphique du tableau X (4) après exécution du premier programme est :

12	**5**	**8**	**20**

La représentation graphique du tableau voyelle (4) après exécution du deuxième programme est :

a	**e**	**i**	**o**	**u**	**y**

2. L'exécution du programme nous affichera successivement à l'écran :

4
25
9
100
16
4

3. L'exécution du programme nous affichera successivement à l'écran :

1
1
2
3
5 8
13
21

4. Le programme est :

Variables i , somme : **ENTIERS**

```
Tableau T (N) : ENTIER
DEBUT
        somme ← 0
        POUR i = 1 A N
                somme ← somme + T (i)
        FIN POUR
         Ecrire << La somme de tous les éléments du tableau est : » , somme
FIN
```

5. Le programme est :

```
Variables i , max : ENTIERS
Tableau T (N) : ENTIER
DEBUT
        max ← T (1)
        i ← 1
        REPETER
                i ← i + 1
                SI T (i) > max ALORS
                        max ← T (i)
                FIN SI
        JUSUQ'A i = N
         Ecrire << La somme de tous les éléments du tableau est : » , somme
FIN
```

6. Le programme est :

```
Variables i , somme , moyenne , nsup : Réels
Tableau Note (100) : Réel
DEBUT
        somme ← 0
        POUR i = 1 A 100
                Lire Note (i)
                somme ← somme + Note (i)
        FIN POUR
        Moyenne ← somme / 100
        nsup ← 0
        POUR i = 1 A 100
                SI Note (i) > moyenne ALORS
                        nsup ← nsup + 1
                FIN SI
        FIN POUR
         Ecrire << Le nombre de notes supérieures à la moyenne est : » , nsup
FIN
```

7. Le programme est :

```
Variables i , pmax , pmin : Entiers
Tableau T (N) : Réel
DEBUT
        max ← T (1)
        min ← T (1)
        pmax ← 1
        pmin ← 1
        i ← 1
        REPETER
                i ← i + 1
                SI T (i) > max ALORS
```

```
                max ← T (i)
                pmax ← i
            FIN SI
            SI T (i) < min ALORS
                min ← T (i)
                pmin ← i
            FIN SI JUSUQ'A i = N Ecrire << La position du plus
        grand élément du tableau est : » , pmax
        Ecrire << La position du plus petit élément du tableau est : » , pmin
FIN
```

8. Le programme est :

```
Variables X ,i,Compt : Réels
Variable Compt :ENTIER
Tableau T (N) : Réel
DEBUT
        Lire X

        POUR i=1 JUSQU'A i=N

            SI T (i) =X ALORS
                Compt← compt+1
            FIN SI
        FIN POUR
        Ecrire << Le nombre d'occurrences de cet éléments du tableau est : » , compt
FIN
```

9. Le programme est :

```
Variables i , note , nnote , snote , smoyenne , cmoyenne : Entiers
Tableau moy (25) : Réel
DEBUT
        POUR i = 1 A 25
            Ecrire << Notes de l'élève numéro » , i
            snote  ←  0
            nnote  ←  0
            REPETER
                Lire note
                SI note <> -1 ALORS
                    snote ← snote + note
                    nnote ←nnote + 1
                FIN SI
            JUSQU'A note = -1
            moy (i) = snote / nnote
            smoyenne = smoyenne + moy (i)
        FIN POUR
        Ecrire << Moyennes »
        POUR i = 1 A 25
            Ecrire << Elève numéro » , i , << : » , moy (i)
        FIN POUR
        cmoyenne = smoyenne / 25
        Ecrire << Moyenne de la classe : » , cmoyenne
```

6.2. Les tableaux dynamiques

Il arrive fréquemment que l'on ne connaisse pas à l'avance le nombre d'éléments que devra comporter un tableau. Bien sûr, une solution consisterait à déclarer un tableau avec une taille très grande. Cela pourra avoir comme conséquence soit que cette taille ne nous nous suffira pas ou qu'une place mémoire immense sera réservée sans être utilisée.
Afin de surmonter ce problème on a la possibilité de déclarer le tableau sans préciser au départ son nombre d'éléments. Ce n'est que dans un second temps, au cours du programme, que l'on va fixer ce nombre via une instruction de re-dimensionnement : **Redim**.

Exemple :
On veut saisir des notes pour un calcul de moyenne, mais on ne sait pas combien il y aura de notes à saisir. Le début de l'algorithme sera quelque chose du genre :

```
Tableau Notes () : Réel
Variable nb en Entier
        DEBUT
                Ecrire "Combien y a-t-il de notes à saisir ?"
                Lire nb
                Redim Notes(nb-1)

                ...
        FIN
```

Exercices

1. Insertion d'un élément dans un tableau

Soit T un tableau de N éléments. Ecrire un programme qui permet d'insérer un élément x à la position i du tableau T.

2. Suppression d'un élément du tableau

Soit T un tableau de N éléments. Ecrire un programme qui permet de supprimer un élément x du tableau.

Solutions

1. Le programme est :

```
Tableau T () : Entier
Variables i, x, j : Entier
DEBUT
        Ecrire « Donnez la dimension du tableau >>
        Lire N
        Redim T (N)
        POUR j = 1 A N
                Lire T (j)
        FIN POUR
        Ecrire « Entrez le nombre à insérer >>
        Lire x
        Ecrire « Entrez la position où insérer ce nombre >>
        Lire i
        Redim T (N +1)
        j = N
        TANT QUE j ≥ i
                T (j+1) = T (j)
                j = j - 1
        FIN TANT QUE
        T (i) = x
```

Dans ce programme on a travaillé avec un seul tableau dynamique. On peut aussi travailler avec le tableau T à dimension fixe et en définir un autre qui recevra tous les éléments de T plus l'élément à insérer. Le programme dans ce cas est :

```
Tableau T (N) : Entier
Tableau Tr (N+1) : Entier
Variables i , x , j , k : Entier
DEBUT
    POUR j = 1 A N
        Lire T (j)
    FIN POUR
    Ecrire « Entrez le nombre à insérer >>
    Lire x
    Ecrire « Entrez la position où insérer ce nombre >>
    Lire i
    j = 1
    k = 1
    TANT QUE k ≤ N + 1
        SI k ≠ i ALORS
            Tr (k) ← T (j)
            j ← j + 1
        SINON
            Tr (k) = x
        FIN SI
        k = k + 1
FIN TANT QUE 2. Le
programme est :
Tableau T (N) : Entier
Tableau Tr () : Entier
Variables i , x , j : Entier
DEBUT
    POUR j = 1 A N
        Lire T (j)
    FIN POUR
    Ecrire « Entrez le nombre à supprimer >>
    Lire x
    j ← 0
    POUR i = 1 A N
        SI T (i) ≠ x ALORS
            j ← j + 1
            ReDim Tr (j)
            Tr (j) = T (i)
        FIN SI
    FIN POUR
```

Dans ce programme on a considéré deux tableaux, le tableau T à dimension fixe et le tableau Tr dynamique. Il est aussi possible de travailler avec un seul tableau dynamique.

```
Tableau T () : Entier
Variables x, j , k , N : Entiers
DEBUT
    Ecrire « Donnez la dimension du tableau >>
    Lire N
    Redim T (N)
```

```
POUR j = 1 A N
        Lire T (j)
FIN POUR
Ecrire « Entrez le nombre à supprimer »
Lire x
j = 1
TANT QUE j ≤ N
        SI T (j) = x ALORS
                POUR k = j A N - 1
                        T (k) = T (k + 1)
                FIN POUR
                N ← N – 1
                ReDim T (N)
        SINON
                j ← j + 1
        FIN SI
FIN TANT QUE
```

6.3. Les tableaux multidimensionnels

Nous avons vu qu'un tableau à une dimension correspond à une liste ordonnée de valeurs, repérée chacune par un indice.

Dans tous les langages de programmation, il est possible de définir des tableaux à deux dimensions (permettant par exemple de représenter des matrices). Ainsi, on pourra placer des valeurs dans un tableau à deux dimensions et cela consiste comme dans le cas des tableaux à une dimension à donner un nom à l'ensemble de ces valeurs. Chaque valeur est alors repérée par deux indices qui précise la position.

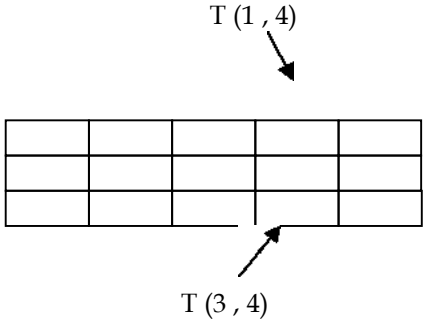
On déclare un tableau à deux dimensions de la façon suivante :

Tableau *nomjableau* (*i* , *j*) : **Type**

nomjableau : désigne le nom du tableau
i : désigne le nombre de lignes du tableau
j : désigne le nombre de colonnes du tableau
Type : représente le type des éléments du tableau

Exemple :

Soit T (3 , 5) un tableau d'entiers. On peut représenter graphiquement par :



T (1 , 4) et T(3 , 4) sont deux éléments du tableau. Entre parenthèse on trouve les valeurs des indices séparées par une virgule. Le premier sert à repérer le numéro de la ligne, le second le numéro de la colonne.

On accède en lecture ou en écriture à la valeur d'un élément d'un tableau à deux dimensions en utilisant la syntaxe suivante :

Nomtableau (*i* , *j*)

Par exemple si T est défini par : **Tableau** T (3 , 2) : Réel

- T (2 , 1) ← -1.2
  met la valeur -1.2 dans la case 2,1 du tableau
- En considérant le cas où a est une variable de type Réel, a ← T (2 , 1)
  met -1.2 dans a

Par extension, on peut aussi définir des tableaux à n dimensions. Leur déclaration sera à l'image de celle des tableaux à deux dimensions, c'est- à- dire :

Tableau *nom_tableau* (*i* , *j* , *k* , ...) : **Type**

Par exemple :

Tableau X (10 , 9 , 5) : **Entier**

Ainsi que leur utilisation :

- X (2 , 1 , 3) ← 10
- a ← X (2 , 1 , 3)

Exercices

1. Considérons le programme suivant :

```
Tableau X (2 , 3) : Entier
Variables i , j , val : Entiers
DEBUT
        val ← 1
        POUR i = 1 A 2
                POUR j = 1 A 3
                        X (i , j) ← val
                        val ← val + 1
                FIN POUR
        FIN POUR
        POUR i = 1 A 2
                POUR j = 1 A 3
                        Ecrire X (i , j)
                FIN POUR
        FIN POUR
```

a. Que produit l'exécution de ce programme.

b. que produira ce programme si l'on remplace les derniers lignes par :

```
        POUR j = 1 A 3
                POUR i = 1 A 2
                        Ecrire X (i , j)
                FIN POUR
        FIN POUR
```

2. Quel résultat fournira ce programme :

```
Tableau X (4 , 2) : Entier
Variables k , m : Entiers
DEBUT
        POUR k = 1 A 4
                POUR m = 1 A 2
                        X (k , m) ← k + m
                FIN POUR
        FIN POUR
        POUR k = 1 A 4
                POUR m = 1 A 2
                        Ecrire X (k , m)
                FIN POUR
        FIN POUR
```

3. Soit T un tableau à deux dimensions de vingt lignes et cinquante colonnes.
a. Ecrire un algorithme qui permet de calculer la somme de tous les éléments du tableau.
b. Ecrire l'algorithme qui permet de compter le nombre des éléments strictement positifs.
c. Ecrire l'algorithme permettant d'obtenir la somme des éléments positifs (spos) et la somme des éléments négatifs (sneg) de ce tableau.
d. Ecrire l'algorithme qui détermine la plus grande valeur des éléments du tableau.
e. Ecrire l'algorithme qui détermine simultanément l'élément le plus grand du tableau ainsi que sa position.

Solutions
1. L'exécution du programme donnera :
Les deux premières boucles du programme permettent de remplir le tableau. Ainsi la représentation graphique sera :

1	2	3
4	5	6

Les deux dernières boucles permettent d'afficher ces six éléments. Le résultat sera donc :
Question a

1
2
3
4
5
6

Question b

1
4
2
5
3
6

2. Les deux premières boucles de ce programme permettent de remplir le tableau. La représentation graphique sera ainsi :

2	3
3	4
4	5
5	6

Les dernières boucles permettent d'afficher les huit éléments du tableau. Le résultat est :

2
3
3
4
4
5
5
6

3. Soit T (20 , 50) un tableau de réels.
a. L'algorithme qui calcule la somme de tous les éléments du tableau est :

```
Tableau T (20 , 50) : Réel
Variables i , j : Entiers
Variable som : Réel
DEBUT
        som ← 0
        POUR i = 1 A 20
                POUR j = 1 A 50
                        som ← som + T (i , j)
                FIN POUR
        FIN POUR
        Ecrire << La somme de tous les éléments du tableau est : » , som
```

b. L'algorithme qui compte le nombre des éléments strictement positifs est :

```
Tableau T (20 , 50) : Réel
Variables i , j : Entiers
Variable npos : Réel
DEBUT
        npos ← 0
        POUR i = 1 A 20
                POUR j = 1 A 50
                        SI T (i , j) > 0 ALORS
                                npos ← npos + 1
                        FIN SI
                FIN POUR
        FIN POUR
        Ecrire << Le nombre des éléments strictement positifs du tableau est : » , npos
```

c. L'algorithme permettant d'obtenir la somme des éléments positifs (spos) et la somme des éléments négatifs (sneg) de ce tableau est :

```
Tableau T (20 , 50) : Réel
Variables i , j : Entiers
Variable spos , sneg : Réel
DEBUT
        spos ← 0 sneg ←
        0 POUR i = 1 A
        20
                POUR j = 1 A 50
                        SI T (i , j) > 0 ALORS
                                spos ← spos + T (i , j)
                        SINON
                                sneg ← sneg + T (i , j)
                        FIN SI
                FIN POUR
        FIN POUR
        Ecrire << La somme des éléments positifs du tableau est : » , spos
        Ecrire << La somme des éléments négatifs du tableau est : » , sneg
```

d. L'algorithme qui détermine la plus grande valeur des éléments du tableau est :

```
Tableau T (20 , 50) : Réel
Variables i , j : Entiers
Variable max : Réel
DEBUT
        max ← T (1 , 1)
        POUR i = 1 A 20




                POUR j = 1 A 50
                        SI T (i , j) > max ALORS
```

```
                    max ← T (i , j)
                FIN SI
            FIN POUR
        FIN POUR
        Ecrire << Le plus grand élément du tableau est : >> , max
        Ecrire << la position de l'élément i <<= >>,imax, <<et j=>> jmax
```

e. L'algorithme qui détermine simultanément l'élément le plus grand du tableau ainsi que sa position est :

```
Tableau T (20 , 50) : Réel
Variables i , j , imax , jmax : Entiers
Variable max : Réel
DEBUT
    max ← T (1 , 1)
    POUR i = 1 A 20
        POUR j = 1 A 50
            SI T (i , j) > max ALORS
                max ← T (i , j)
                imax ← i
                jmax ← j
            FIN SI
        FIN POUR
    FIN POUR
    Ecrire << Le plus grand élément du tableau est : >> , max
```

7. LES STRUCTURES

Imaginons que l'on veuille afficher les notes d'une classe d'élèves par ordre croissant avec les noms et prénoms de chaque élève. On va donc utiliser trois tableaux (pour stocker les noms, les prénoms et les notes). Lorsque l'on va trier le tableau des notes il faut aussi modifier l'ordre les tableaux qui contiennent les noms et prénoms. Mais cela multiplie le risque d'erreur. Il serait donc intéressant d'utiliser ce qu'on appelle **les structures**.

Les structures contrairement aux tableaux servent à rassembler au sein d'une seule entité un ensemble fini d'éléments de type éventuellement différents. C'est le deuxième type complexe disponible en algorithmique.

A la différence des tableaux, il n'existe pas par défaut de type structure c'est-à-dire qu'on ne peut pas déclarer une variable de type structure. Ce qu'on peut faire c'est de construire toujours un nouveau type basé sur une structure et après on déclare des variables sur ce nouveau type.

La syntaxe de construction d'un type basé sur une structure est :

```
TYPE NomDuType = STRUCTURE
    attribut1 : Type
    attribut2 : Type

    . . .

    attributn : Type
FIN STRUCTURE
```

Le type d'un attribut peut être :

- Un type simple
- Un type complexe
  - Un tableau
  - Un type basé sur une structure

Exemple :

```
TYPE Etudiant = STRUCTURE
    nom : chaîne
    prenom : chaîne
    note : Réel
```

FIN STRUCTURE

Dans cet exemple on a construit un type *Etudiant* basé sur une structure. Cette structure a trois attributs (on dit aussi champ) : nom, prenom et note.

TYPE Date = STRUCTURE
jour : **Entier**
mois : **Entier**
annee : **Entier**
FIN STRUCTURE

Dans ce deuxième exemple, on a construit un type *Date* basé sur une structure. Cette structure a aussi trois attributs : jour, mois et annee.

Après on peut déclarer des variables basé sur ce type. Par exemple :
Variable Etud : **Etudiant**
Donc Etud est une variable de type Etudiant.

Il est possible de déclarer un tableau d'éléments de ce type Etudiant par exemple. On pourra écrire donc :
Tableau Etud (20) : **Etudiant**
Etud (1) représente le premier étudiant.

Maintenant, pour accéder aux attributs d'une variable dont le type est basé sur une structure on suffixe le nom de la variable d'un point « . » suivi du nom de l'attribut. Par exemple, dans notre cas pour affecter le nom "Dinar" à notre premier étudiant, on utilisera le code suivant :
Etud (1).nom = « Dianr »

Exercices

1. Définissez la structure « Stagiaire » constituée des champs suivants :

Champ	**Type**
Nom	Chaîne
Prénom	Chaîne
Datenais	Structure

Le champ « Datenais » est aussi une structure dont les champs sont :

Champ	**Type**
Jour	Entier
Mois	Entier
Année	Entier

Ecrivez ensuite l'algorithme qui permet de lire et après afficher le nom, prénom et date de naissance d'un stagiaire.

2. On souhaite gérer les notes d'un étudiant. Pour cela on va définir la structure « Etudiant » dont les champs sont :

Champ	**Type**
Nom	Chaîne
Prénom	Chaîne
Note	Tableau de 3 éléments
Moyenne	Réel

Ecrire l'algorithme qui permet de lire les informations d'un étudiant (nom, prénom et notes), de calculer sa moyenne et d'afficher à la fin un message sous la forme suivante :

<< La moyenne de l'étudiant Dinar Youssef est : 12.45 >>

où << Dinar >> et << Youssef >> sont les noms et prénoms lus et 12.45 est la moyenne calculée.

3. Modifier l'algorithme de l'exercice précédent de façon que l'on puisse gérer les notes de 50 étudiants.

Solutions

1. L'algorithme est : **TYPE**
Date = STRUCTURE
    Jour : **Entier**
    Mois : **Entier**
    Annee : **Entier**
FIN STRUCTURE
TYPE Stagiaire = STRUCTURE
    Nom : **chaîne**
    Prenom : **chaîne**
    Datenais : **Date**
FIN STRUCTURE
Variable stag : **Stagiaire**
DEBUT
Ecrire << Entrez les information du stagiaire >>
Ecrire << Entrez le nom >>
Lire stag.Nom
Ecrire << Entrez le prénom >>
Lire stag.Prenom
Ecrire << Entrez le jour de naissance >>
Lire stag.Date.Jour
Ecrire << Entrez le mois de naissance >>
Lire stag.Date.Mois
Ecrire << Entrez l'année de naissance >>
Lire stag.Date.Annee
Ecrire << Le nom du stagiaire est : >> , stag.Nom
Ecrire << Son prénom est : >> , stag.Prenom
Ecrire <<Sa date de naissance est :>>, stag.Date.Jour , <</>>, stag.Date.Mois, <</>>, stag.Date.Annee

2. L'algorithme est : **TYPE**
Etudiant = STRUCTURE
    Nom : **Chaîne**
    Prenom : **Chaîne**
    Note (3) : **Réel**
    Moyenne : **Réel**
FIN STRUCTURE
Variable i : **Entier**
Variable som : **Réel**
Variable etud : **Etudiant**
DEBUT

Ecrire << Entrez les information de l'étudiant >>
Ecrire << Entrez le nom >>
Lire etud.Nom
Ecrire << Entrez le prénom >>
Lire etud.Prenom
Ecrire << Entrez la première note >>
Lire Etud.Note (1)
Ecrire << Entrez la deuxième note >>
Lire etud.Note (2)
Ecrire << Entrez la troisième note >>
Lire etud.Note (3)
som ← 0
POUR i = 1 **A** 3
som ← etud.Note (i)
FIN POUR
etud.Moyenne = som / 3
Ecrire <<La moyenne de l'étudiant >> , etud.Nom , << >> , etud.Prenom , << est : >> , etud.Moyenne

3. L'algorithme est :
TYPE Etudiant = STRUCTURE
Nom : **Chaîne**
Prenom : **Chaîne**
Note(3) : **Réel**
Moyenne : **Réel**
FIN STRUCTURE
Variable i , j : **Entier**
Variable som : **Réel**
Variable etud(50) : **Etudiant**
DEBUT
Ecrire << Entrez les information des étudiants >>
POUR j = 1 **A** 50
Ecrire << Entrez le nom >>
Lire etud(j).Nom
Ecrire << Entrez le prénom >>
Lire etud(j).Prenom
Ecrire << Entrez la première note >>
Lire etud(j).Note (1)
Ecrire << Entrez la deuxième note >>
Lire etud(j).Note (2)
Ecrire << Entrez la troisième note >>
Lire etud(j).Note (3)
som ← 0
POUR i = 1 **A** 3
som ← etud(j).Note (i)
FIN POUR
etud (j).Moyenne = som / 3
FIN POUR
POUR j = 1 **A** 50
Ecrire <<La moyenne de l'étudiant >> , etud(j).Nom , << >> , etud(j).Prenom , << est : >> ,

etud(j).Moyenne

FIN POUR

8. LES FONCTIONS ET PROCEDURES

En programmation, donc en algorithmique, il est possible de décomposer le programme qui résout un problème en des sous-programmes qui résolvent des sous parties du problème initial. Ceci permettra d'améliorer la conception du programme et ainsi sa lisibilité.
L'utilisation des sous-programmes s'avère utile aussi dans le cas où on constate qu'une suite d'actions se répète plusieurs fois.
Il existe deux types de sous-programmes les fonctions et les procédures. Un sous- programme est obligatoirement caractérisé par un nom (un identifiant) unique.
Le nom d'un sous-programme comme le nom d'une variable doit :

- Contenir que des lettres et des chiffres
- Commencer obligatoirement par une lettre

Le programme qui utilise un sous- programme est appelé **programme appelant**. Un sous-programme peut être invoqué n'importe où dans le programme appelant en faisant référence à son nom.
Un programme ainsi doit suivre la structure suivante :

Définition des constantes
Définition des types
Déclaration des variables
Définition des sous- programmes
DEBUT
Instructions du programme principal
FIN

8.1. Les fonctions

Une fonction est un sous-programme qui retourne un **seul** résultat. Pour définir une fonction on utilise la syntaxe suivante :

FONCTION nom_fonction (Argument1 : **Type** , Argument2 : **Type** ,....) : **Type**
Déclarations
DEBUT
Instructions de la fonction
nom_fonction ← Valeur renvoyée
FIN

On constate que la déclaration d'une fonction revient à lui préciser un nom, un type ainsi qu'une liste d'arguments.
Un argument (appelé **paramètre formel**) d'un sous- programme est une variable locale particulière qui est associée à une variable ou constante du programme appelant. Puisque qu'un argument est une variable locale, il admet un type.
Lorsque le programme appelant appelle le sous-programme il doit indiqué la variable ou la constante de même type, qui est associée au paramètre.
Par exemple, le sous-programme *sqr* permet de calculer la racine carrée d'un réel. Ce sous-programme admet un seul paramètre de type réel positif.
Le programme qui utilise *sqr* doit donner le réel positif dont il veut calculer la racine carrée, cela peut être :

- une variable, par exemple *a*
- une constante, par exemple 5.25

Les arguments d'une fonction sont en nombre fixe (≥ 0).
Une fonction possède un seul type, qui est le type de la valeur retournée qui est affecté au nom de la fonction.
Une fois la fonction définie, il est possible (en fonction des besoins) à tout endroit du programme appelant de faire appel à elle en utilisant simplement son nom suivi des arguments entre parenthèses.

Les parenthèses sont toujours présentes même lorsqu'il n'y a pas de paramètre.
Les arguments spécifiés lors de l'appel de la fonction doivent être en même nombre que dans la déclaration de la fonction et des types prévus. Dans le programme appelant ces arguments sont appelés **paramètres effectifs**.

La valeur ainsi renvoyée par la fonction peut être utilisée dans n'importante quelle expression compatible avec son type.

Exemple :

```
FONCTION Calcul (x : Réel , y : Réel , z : Réel) : Réel
        Variable a : Entier
DEBUT
        a ← 3
        Calcul ← (x + y + z) * a
FIN
```

Ça c'est un exemple de déclaration de fonction. Cette fonction appelée << Calcul >> est de type réel et elle admet trois arguments de type réel.
Maintenant voici un exemple de programme complet :

```
FONCTION Calcul (x : Réel , y : Réel , z : Réel) : Réel
        Variable a : Entier
DEBUT
        a ← 3
        Calcul ← (x + y + z) * a
FIN
Variables i , j , k , b : Réels
DEBUT
        Lire i
        Lire j
        Lire k
        b ← Calcul (i , j , k) + 2
        Ecrire b
FIN
```

Dans ce programme on fait appel a une fonction. Sa valeur est utilisée dans une expression.

Exercice

1. Définir la fonction << Somme >> qu'on lui passe deux valeurs de type entier et qui renvoie comme valeur la somme des valeurs reçues.

2. Définir la fonction << Absolue >> qui renvoie la valeur absolue d'une valeur qu'on lui passe comme paramètre.
3. Définir la fonction << Inverse >> qui renvoie l'inverse d'une valeur qu'on lui passe comme paramètre.

4. Définir la fonction << Max >> qui renvoie le maximum de deux valeurs.

5. Ecrivez un programme qui lit trois scores et qui utilise la fonction définie dans l'exercice précédent pour déterminer le meilleur score et l'afficher après.

Solutions

1. La définition de la fonction << Somme >> est :

```
FONCTION Somme (x : Réel , y : Réel ) : Réel
```

```
DEBUT
        Somme ← x + y
FIN
```

2. La définition de la fonction << Absolue >> est :

```
FONCTION Absolue (x : Réel) : Réel
DEBUT
        SI x > 0 ALORS
                Absolue ← x
        SINON
                Absolue ← -1 * x
        FIN SI
FIN
```

3. La définition de la fonction << Inverse >> est :

```
FONCTION Inverse (x : Réel) : Réel
DEBUT
        SI x ≠ 0 ALORS
                Inevrse ← 1 /x
        FIN SI
FIN
```

4. La définition de la fonction << Max >> est :

```
FONCTION Max (x : Réel , y Réel) : Réel
DEBUT
        SI x > y ALORS
                Max ← x
        SINON
                Max ← y
        FIN SI
FIN
```

5. Le prgramme est :

```
FONCTION Max (x : Réel , y Réel) : Réel
DEBUT
        SI x > y ALORS
                Max ← x
        SINON
                Max ← y
        FIN SI
FIN
Variables score1 , score2 , score3 , meil_score : Réels
DEBUT
        Ecrire << Entrez les troix scores : >>
        Lire score1 Lire score2 Lire score3 meil_score ←
        Max (Max (score1 , score2) , score3)
        Ecrire << Le meilleur score est : >> , meil_score
FIN
```

8.2. Les variables locales et globales

La **portée** d'une variable est l'ensemble des sous- programmes où cette variable est connue c'est-à-dire que les instructions de ces sous-programmes peuvent utiliser cette variable.
Une variable définie au niveau du programme principal (problème appelant) est appelée **variable globale**. Sa portée est totale : **tout** sous-programme du programme principal peut utiliser cette variable.
Cependant, une variable définie au sein d'un sous-programme est appelée **variable locale**. La portée d'une variable locale est uniquement le sous-programme qui la déclare.

Remarque :
Lorsque le nom d'une variable locale est identique à une variable globale, la variable globale est localement masquée. Dans ce sous-programme la variable globale devient inaccessible.

Exemple
Soit le programme suivant :

```
Fonction Surface (a : Réel) : Réel
Variables valeur , resultat : Réels
DEBUT
        valeur ← 3.14
        resulat ← valeur * a
        Surface ← resultat
FIN
Variable rayon : Réel
DEBUT
        Ecrire << Entrez le rayon du cercle : »
        Lire rayon
        Ecrire << La surface de cette cercle est : » , Surface (rayon)
FIN
```

Les variables *valeur* et *resultat* déclarées dans la fonction *Surface* sont locales.
Considérons presque le même programme sauf que la variable *valeur* est déclarée maintenant dans le programme appelant.

```
Fonction Surface (a : Réel) : Réel
Variables resultat : Réels
DEBUT
        resulat ← valeur * a
        Surface ← resultat
FIN
Variable valeur , rayon : Réel
DEBUT
        valeur ← 3.14
        Ecrire << Entrez le rayon du cercle : »
        Lire rayon
        Ecrire << La surface de cette cercle est : » , Surface (rayon)
FIN
```

Dans ce deuxième programme seule la variable *resultat* est locale. Tandis que la variable *valeur* est devenue globale. On peut par conséquent accéder à sa valeur dans la fonction *Surface*.

8.3. Les passage de paramètres

Il existe deux types d'association (que l'on nomme **passage de paramètre)** entre le(s) paramètre(s) du

sous-programme (fonction ou procédure) et variable(s) du programme appelant :

- **Passage par valeur**
- **Passage par adresse**

Dans le cas où l'on choisit pour un paramètre effectif un passage par valeur, la valeur de ce paramètre effectif ne change pas même si lors de l'appel du sous-programme la valeur du paramètre formel correspondant change. On peut dire que dans ce cas le paramètre effectif et le paramètre formel ont font deux variables différents qui ont seulement la même valeur. C'est la type de passage par défaut.
Dans le cas où l'on choisit pour un paramètre effectif un passage par adresse, la valeur de ce paramètre effectif change si lors de l'appel du sous-programme la valeur du paramètre formel correspondant change. On peut dire que dans ce cas le paramètre effectif et le paramètre formel ont font deux variables qui ont le même adresse (par conséquent valeur). Pour préciser qu'il s'agit d'un passage par adresse, il faut soulignés les paramètres concernés lors de la définition du sous-programme.

Exemple

Considérons les deux programmes suivants :

Programme 1

```
Fonction Calcul (a : Réel) : Réel

DEBUT
    Calcul ← a * 2
    a ← a – 1
FIN
Variable x : Réel
DEBUT
    x ← 3
    Ecrire Calcul (x)
    Ecrire x
FIN
```

Programme 2

```
Fonction Calcul (a : Réel) : Réel
DEBUT
    Calcul ← a * 2
    a ← a – 1
FIN
Variable x : Réel
DEBUT
    x ← 3
    Ecrire Calcul (x)
    Ecrire x
FIN
```

Dans le premier programme on a un passage de paramètre par valeur et dans le deuxième on a un passage de parametrespar adresse. Le premier programme affichera le résultat suivant :

6
3

car même si la valeur de *a* change celle de *x* non.
Tandis que le deuxième programme il affichera :

6
2

la valeur de *x* changera car celle de *a* a changée.

8.4. Les procédures

Les procédures sont des sous- programmes qui ne retournent **aucun** résultat. Elles admettent comme les fonctions des paramètres.
On déclare une procédure de la façon suivante :

PROCEDURE nom_procedure (Argument1 : **Type** , Argument2 : **Type** ,....)
Déclarations
DEBUT
Instructions de la procédure
FIN

Et on appelle une procédure comme une fonction, en indiquant son nom suivi des paramètres entre parenthèses**.**

Exercice

1. Ecrire une procédure qui reçoit la longueur et la largeur d'une surface et qui affiche la valeur de la surface. Donnez à cette procédure le nom « Surface ».
2. Ecrire une procédure qui permet d'échanger les valeurs de deux variables. Appelez cette procédure « Echanger ».
3. On dispose d'une phrase dont les mots sont séparer par des point virgules. Ecrivez une procédure qui permet de remplacer les points virgules par des espaces. On suppose qu'on dispose des fonctions suivantes :

- Longueur : permet de calculer la longueur d'une chaîne de caractères.
Utilisation : Longueur (chaîne)
- Extraire : permet d'extraire une partie (ou la totalité) d'une chaîne.
Utilisation : Extraire (chaîne , position_debut, longueur)
Paramètre : chaîne de laquelle on fait l'extraction
position_debut la position à partir de laquelle va commencer l'extraction
longueur désigne la longueur de la chaîne qui va être extraite.

Solutions

1. Le programme qui définit la procédure << Surface » est :

PROCEDURE Surface (longueur : **Réel** , largeur : **Réel**)
Variable s : **Réel**
DEBUT
s ← longueur * largeur
Ecrire << La surface obtenu est : » , s
FIN

2. Le programme qui définit la procédure << Echanger » est :

PROCEDURE Echanger (x : **Réel** , y : **Réel**)
Variable z : **Réel**
DEBUT
z ← x
x ← y
y ← z
FIN

3. Le programme de cette procédure est :

PROCEDURE Changer (chaine : **Chaîne**)

```
Variables i , l : Entier
Variables caract , schaine : Chaîne
DEBUT
        l ← Longueur (chaine)
        schaine = << << »
        POUR i = 1 A l
                caract ← Extraire (chaine , i , 1)
                SI caract = << ; » ALORS
                        caract = << »
                FIN SI
                schaine ← schaine & caract
        FIN POUR
        chaine ← schaine
FIN
Variable chaine : Chaîne
Variable i : Entier
DEBUT
        chaine ← << bonjour;tout;le;monde »
        changer (chaine)
        Ecrire chaine
```

9. LA RECHERCHE BINAIRE (RECHERCHE DICHOTOMIQUE)

Soit T un tableau de N éléments ordonnés et x un élément de même type que les éléments de T. Il s'agit d'examiner la présence de x dans T. Comme le tableau est ordonné, il satisfait la spécification suivante :

• i [, N-1] (i) ≤ T (i+1)

Au lieu de faire une recherche linéaire[2], on décompose le tableau en deux sous-tableaux T1 et T2 et trois cas peuvent se produire :

- x est trouvé la recherche est terminé.
- la recherche continue dans T1.
- la recherche continue dans T2.

Exemple

Soit le tableau suivant T :

1	2	3	4	5	6	7	8
3	7	8	8	11	15	20	24

Vous constatez que le tableau T est déjà ordonné. On va voir comment s'applique la méthode de recherche binaire pour rechercher si x = 20 existe dans le tableau T.

On divise l'intervalle des indices [1,8] en deux intervalles [1,4] et [5,8]. On obtient deux tableaux T1 et T2.

1	2	3	4	5	6	7	8
3	7	8	8	11	15	20	24

x ∉ T1 x ∈ T2

La recherche va continuer dans le tableau T2 puisque x (20) est plus supérieur que le plus grand élément de T1. Donc l'intervalle de recherche devient [5,8] et on va le diviser à son tour en deux intervalles [5,6] et [7,8].

5	6	7	8
11	15	20	24

x ∉ T1 x ∈ T2

De même, la recherche va continuer dans T2. L'intervalle de recherche devient [7,8]. On le divise en deux intervalles [7,7] et [8,8].

Finalement, x est trouvé.

7	8
20	24

x ∈ T x ∉ T2

Le programme de la recherche dichotomique est le suivant :

Tableau T(N) : **Entier**
Variables inf , sup , milieu , x : **Entier**
    Variable Trouve : **Booléen**
    DEBUT
        Trouve ← Faux
        inf = 1
        sup ← N
    TANT QUE inf ≤ sup **ET** Trouve = Faux

2 Parcourir tout le tableau T du premier élément au dernier.

```
        milieu ← (inf + sup) Div 2        ' Div est la division entière
        SI T(milieu) = x ALORS
                Trouve ← Vrai
        SINON
                SI T(milieu) < x alors
                        inf ← inf + 1
                SINON
                        sup ← milieu –1
                FIN SI
        FIN SI
FIN TANT QUE
SI Trouve = Vrai ALORS
        Ecrire « L'élément >> , x , « existe dans T >>
SINON
        Ecrire « L'élément >> , x , « n'existe pas dans T >>
FIN SI
FIN
```

10. LES ALGORITHMES DE TRI

Trier les éléments d'un tableau revient à ordonner tous ces éléments selon un ordre croissant ou décroissant.

Soit T un tableau de N éléments muni d'une relation d'ordre ≤. Trier ce tableau c'est construire un algorithme qui devra satisfaire à la spécification suivante :

 i [, N-1] (i) ≤ T (i+1)

Dans ce paragraphe on va traiter plusieurs algorithmes de tri : tri par sélection, tri par bulle, tri par comptage, tri par insertion, tri par shell.

10.1. Tri par bulle

Principe

Ce tri permet de faire remonter petit à petit un élément trop grand vers la fin du tableau en comparant les éléments deux à deux.

Si un élément d'indice *i* est supérieur à un élément d'indice i+1 on les échange et on continue avec le suivant. Lorsqu'on atteint le fin du tableau on repart du début. On s'arrête lorsque tous les éléments du tableau sont bien placés c'est-à-dire qu'on aura aucun changement d'éléments à effectuer.

Algorithme

```
        Tableau T(N) : Entiers
        Variables j , nc : Entiers
                DEBUT
                        REPETER
                                nc ← 0
                                POUR j = 1 A (N-1)
                                        SI T(j) > T(j+1) ALORS
                                                nc ← nc +1
                                                z ← T(j)
                                                T(j) ← T(j+1)
                                                T(j+1) ← z
                                        FIN SI
                                FIN POUR
```

JUSUQU'A nc = 0

FIN

Exemple

Soit le tableau suivant :

52	10	1	25

Boucle REPETER	**Etat du tableau**				**Valeur de nc**
Itération 1	10	52	1	25	3
	10	1	52	25	
	10	1	25	52	
Itération 2	1	10	25	52	1
	1	10	25	52	
	1	10	25	52	
Itération 3	1	10	25	52	0

10.2. Tri par sélection

Principe : Soit T un tableau de N éléments. On cherche le plus petit élément du tableau et on le place à la première position. Après, on cherche le plus petit dans les (N-1) qui reste et on le place en deuxième position et ainsi de suite.

52	10	1	25	62	3	8	55	3	23
1	52	10	25	62	3	8	55	3	23
1	3	52	10	25	62	8	55	3	23
1	3	3	52	10	25	62	8	55	23
1	3	3	8	52	10	25	62	55	23
1	3	3	8	10	52	25	62	55	23
1	3	3	8	10	23	52	25	62	55
1	3	3	8	10	23	25	52	62	55
1	3	3	8	10	23	25	52	62	55
1	3	3	8	10	23	25	52	55	62

Algorithme :

```
POUR i ALLANT DE 1 A 9
FAIRE
        Petit ← TAB (i)
POUR j ALLANT DE i A 10
FAIRE
        Si (TAB (j) < petit) ALORS petit ← TAB (j) ; position ← j FSI
FinPour
POUR j ALLANT DE position A i+1 PAS –1
FAIRE
TAB(j) ← TAB (j-1) ;
FinPour
TAB (i) ← petit ;
FinPour
```

5.4.2- Le tri bulle :

```
FAIRE
Inversion ← FAUX
POUR i ALLANT DE 1 A 9
FAIRE
```

```
Si (TAB (i) > TAB (i+1))
ALORS
        Tampon ← TAB (i) ;
        TAB (i) ← TAB (i+1) ;
```

```
                TAB (i+1) ← Tampon
                Inversion ← VRAI
        FSI
        FinPour
JUSQUA (inversion = FAUX) ;
```

5.4.3- Le tri par permutation :

```
POUR i ALLANT DE 1 A 9
FAIRE
        SI (TAB (i+1) < TAB (i))
        ALORS
                Abouger ← TAB (i+1)
                j ← 1 ;
                TanQue ((j < i) ET (TAB (j) < TAB (i+1)))
                Faire j ← j+1
                FTQ
                POUR k ALLANT DE i+1 A j+1 PAS –1
                Faire
                        TAB (k) ← TAB (k-1)
                FinPour
                TAB (j) ← abouger
        FSI
Fin Pour
```

5.4.4- Le tri par comptage :

```
POUR i ALLAN DE 1 A 10
FAIRE
        RES (i) ← 0
        NB (i) ← 0
        POUR j ALLANT DE 1 A 10
        FAIRE
                Si TAB (j) < TAB (i) ALORS NB (i) ←NB (i) + 1 FSI
        FinPour
FinPour
POUR i ALLANT de 1 A 10
FAIRE
        j ← NB (i)
        TantQue RES (j) <> 0
        Faire
                j ← j+1
        FTQ
        RES (j) ← TAB (i)
FinPour
```

5.4.5- Le tri alphabétique :

```
POUR nbmots ALLANT DE 1 A 10
Faire
        AFFICHER « Entrer le mot suivant »
        LIRE MOT
```

```
        Pluspetit ← VRAI
        j ← 1
        TANTQUE (pluspetit ET (j < nbmots))
        Faire
                k ← 1
                TantQue ((MOTS (j, k) = MOT (k)) ET k <= 20)
                Faire
                        k ← k+1
                FTQ
                Si (MOTS (j, k) < MOT (k))
                ALORS
                        j ← j+1
                SINON
                        Pluspetit ← FAUX
                FSI
        FTQ
        Si (j < nbmots)
        ALORS
                POUR i ALLANT DE nbmots A j+1 PAS –1
                Faire
                        POUR k ALLANT DE 1 A 20
                        Faire
                                MOTS (i, k) ← MOTS (i-1, k)
                        FinPour
                FinPour
        FSI
        POUR k ALLANT DE 1 A 20
        Faire
                MOTS (j, k) ← MOT (k)
        FinPour
FinPour
POUR i ALLANT DE 1 A nbmots
Faire
        AFFICHER MOTS (i)
FinPour
```