MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE TECHNOLOGIQUE
DIRECTION GENERALE DES ETUDES TECHNOLOGIQUES
ISET DJERBA
DEPARTEMENT INFORMATIQUE

COURS:
ALGORITHMIQUE ET STRUCTURES DE DONNEES I

PAR:
OLFA HAMROUNI, TECHNOLOGUE EN INFORMATIQUE

PUBLIC:
NIVEAU 1 TECHNICIEN SUPERIEUR EN INFORMATIQUE

VOLUME HORAIRE HEBDOMADAIRE:
4.5H CI

ECHEANCIER DES SEANCES DE COURS

BUT DU COURS:

Savoir analyser un problème et écrire l'algorithme correspondant.

DATE	COMPETENCES A ACQUERIR PAR L'ETUDIANT	MOYENS PEDAGOGIQUES
Semaine 1	- Définir un ordinateur - Citer les étapes de résolution d'un problème - Connaître et définir la notion d'algorithmique	Note de cours Tableau Support de TD
Semaine 2	- Définir et manipuler une variable et une constante - Savoir faire la différence entre une variable et une constante - Enumérer les types de base	Note de cours Tableau Support de TD
Semaine 3	- Connaître la structure générale d'un algorithme - Manipuler l'instruction d'affectation et les instructions d'entrée/sortie	Note de cours Tableau Support de TD
Semaine 4	- Connaître l'utilité des instructions conditionnelles - Acquérir les formes conditionnelles et savoir distinguer entre elles	Note de cours Tableau Support de TD
Semaine 5 + Semaine 6	- Connaître l'utilité des instructions itératives - Acquérir les formes itératives et savoir distinguer entre elles	Note de cours Tableau Support de TD
Semaine 7 + Semaine 8	- Définir la programmation modulaire - Connaître et manipuler les sous-programmes: procédures et fonctions	Note de cours Tableau Support de TD
Semaine 9 + Semaine 10 + Semaine 11	- Manipuler les tableaux unidimensionnels et bidimensionnels - Définir l'action de tri - Comprendre et écrire l'algorithme du tri par sélection - Comprendre et écrire l'algorithme du tri par insertion	Note de cours Tableau Support de TD

	- Comprendre et écrire l'algorithme du tri à bulle - Comprendre et écrire l'algorithme de la recherche séquentielle - Comprendre et écrire l'algorithme de la recherche dichotomique	
Semaine 12 + Semaine 13	- Manipuler les chaînes de caractères	Note de cours Tableau Support de TD
Semaine 14 + Semaine 15	- Connaître l'utilité des structures - Définir des structures simples - Savoir déclarer une variable de type structure simple et accéder à ses membres - Définir des structures complexes - Savoir déclarer une variable de type structure complexe et accéder à ses membres	Note de cours Tableau Support de TD

PLAN DU COURS

Chapitre 1: INTRODUCTION GENERALE

Objectif du chapitre: Présenter la notion d'algorithme et d'algorithmique

Plan du chapitre:

I. Définition d'un ordinateur
II. Les étapes de résolution d'un problème
III. Définition d'un algorithme
IV. De l'algorithme au programme

I. DEFINITION D'UN ORDINATEUR

Un ordinateur est un ensemble de circuits électroniques qui traite l'information grâce à un programme qu'il mémorise, communique et archive des informations.

Le traitement de l'information se fait automatiquement et vise à résoudre un problème bien défini.

Les différentes fonctions correspondent, en fait, à 3 constituants de l'ordinateur:

- la mémoire centrale
- l'unité centrale
- les périphériques

La mémoire centrale contient les programmes systèmes nécessaires au bon fonctionnement de l'ordinateur, et les programmes utilisateurs répondant à un besoin particulier et résolvant un problème rencontré par le dit utilisateur. Ces programmes auront besoin d'un ensemble de données afin d'être exécutés et fournir les résultats escomptés, ces données existent également dans la mémoire centrale.

L'unité centrale va s'occuper de l'exécution des programmes logés dans la mémoire centrale. Elle est constituée de:

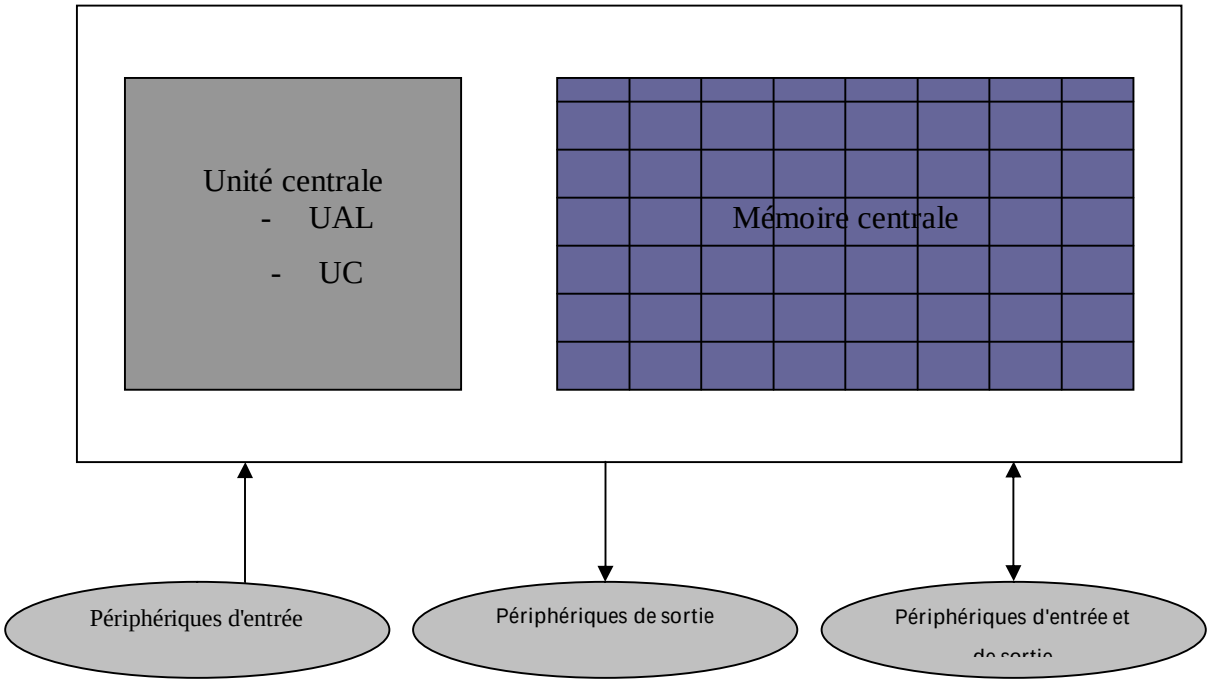
- l'unité arithmétique et logique (UAL) qui s'occupe de toutes les opérations arithmétiques et logiques (addition, soustraction, multiplication, division, comparaison, etc.)
- l'unité de commande (UC) qui exécute les programmes instructions par instructions en collaborant avec l'UAL.

Les périphériques sont les unités qui assurent la relation de l'ordinateur avec le monde extérieur. Ils se répartissent en 3 types:

- les périphériques d'entrée assurant l'entrée des données à partir de l'utilisateur (exemples: clavier, souris, microphone, etc.)
- les périphériques de sortie assurant la sortie des données vers l'utilisateur (exemples: imprimante, écran, haut-parleurs, etc.)

- Les périphériques d'entrée et de sortie assurant l'entrée et la sortie des données à partir et vers l'utilisateur (exemples: disque dur, bande magnétique, disquette, cd-rom groupés sous le nom mémoire auxiliaire ou mémoire de masse ou mémoire secondaire, modem, etc.)

Voici ci-dessous un schéma simplifié d'un ordinateur:



II. LES ETAPES DE RESOLUTION D'UN PROBLEME

Un programme logé dans la mémoire centrale avec l'ensemble de ses données permet de fournir, en s'exécutant, un résultat répondant à un besoin bien déterminé. Ce besoin est exprimé suite à l'existence d'un problème à résoudre. Ainsi, la résolution d'un problème passe par 3 étapes:

- la pré-analyse qui consiste à identifier et comprendre le problème
- l'analyse qui consiste à collecter les données nécessaires pour la résolution du problème
- l'élaboration de l'algorithme qui s'agit de l'ensemble d'étapes à suivre pour résoudre le problème

Exemple:

Problème: confection d'une robe par une couturière

- La pré-analyse: il s'agit de préparer une robe pour une cliente pour une date d
- L'analyse: il s'agit de collecter les données nécessaires pour commencer la confection de la robe:
    - Le modèle
    - Les mesures (taille, hanche, manche, tec.)
    - Le style de la robe (longue, courte, cintrée, ample, avec col ou non, etc.)
    - Le tissu
    - La date d
    - Etc.

- L'élaboration de l'algorithme: il s'agit de:
    - o préparer le patron
    - o couper le tissu suivant les mesures du patron
    - o confectionner la robe avec la machine à couture

III. DEFINITION D'UN ALGORITHME

Un algorithme est un ensemble d'étapes successives, finies et ordonnées et qui visent à résoudre un problème bien défini.

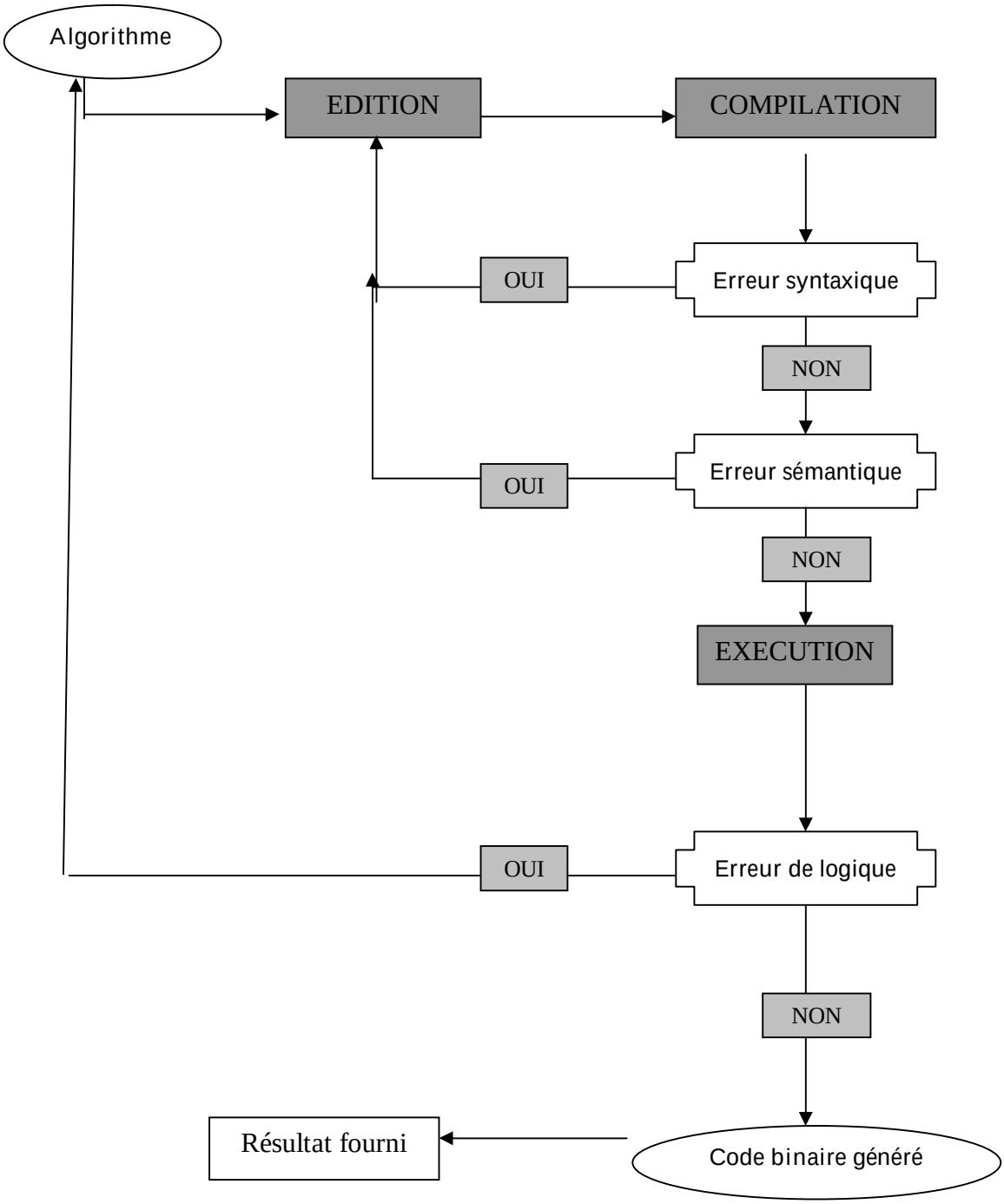
L'algorithmique est la logique d'écrire des algorithmes.

Exemples:

- Algorithme de démarrage d'une auto
- Algorithme de retrait d'argent auprès d'un guichet automatique
- Algorithme de calcul de la surface d'un rectangle
- Algorithme de calcul de la moyenne de 3 notes à coefficient égal à1

IV. DE L'ALGORITHME AU PROGRAMME

Pour passer d'un algorithme à un programme, on suit les étapes suivantes:



- EDITION : C'est la phase de saisie d'un programme écrit en un langage de programmation.
- COMPILATION : C'est la phase de détection des erreurs dans un programme. Ces erreurs sont de type syntaxique et sémantique.

  Les erreurs syntaxiques sont des erreurs générées par le compilateur s'il n'y a pas un respect de la syntaxe du langage de programmation avec lequel est écrit le programme compilé.

Les erreurs sémantiques sont générées si le programme effectue des opérations illégales ou interdites (Exp : division par zéro, affectation d'une valeur n'appartenant pas au domaine d'une variable, etc.).

- EXECUTION : C'est la phase d'exécution d'un programme. Ce dernier est syntaxiquement et sémantiquement correct. Au niveau de cette phase, il y a une vérification de la logique des résultats fournis par le programme exécuté. En effet, il faut vérifier si ce que fournit le programme correspond à ce qui est escompté.

Chapitre 2: PREMIERS ELEMENTS DE L'ALGORITHMIQUE

Objectif du chapitre: connaître les concepts de base de l'algorithmique

Plan du chapitre:

I. Notion de variable et de constante
II. Types de base
III. Structure générale d'un algorithme
IV. Instruction d'affectation
V. Instructions d'entrée/sortie

I. NOTION DE VARIABLE ET DE CONSTANTE

1. NOTION DE VARIABLE

Un programme s'exécute en manipulant des données se trouvant dans la mémoire centrale. Une variable est donc un nom d'un emplacement de la mémoire centrale qui contient des données. Ces données varient au cours de l'exécution du programme. Chaque variable possède un type et est rangée en mémoire à une adresse précise.

D'une manière générale, un nom d'une variable est formé d'une ou plusieurs lettres, les chiffres sont autorisés à condition de ne pas apparaître au début du nom.

En effet, le nom d'une variable doit respecter les 4 règles suivantes:

- il ne doit pas contenir d'espaces
- il ne doit pas contenir des caractères spéciaux (sauf le tiret de soulignement (_))
- il ne doit pas être long
- il ne doit pas commencer par un chiffre

Exemple:

- Montant : une variable désignant le montant d'une facture
- NumCarteEtud: une variable désignant le numéro de la carte d'un étudiant
- Note2Math : Une variable désignant la 2ème note de la matière mathématique

2. NOTION DE CONSTANTE

Une constante possède les mêmes caractéristiques qu'une variable sauf que la valeur ne change pas au cours de l'exécution du programme.

Exemple:

- PI est une constante.

Const PI=3.14 permet de donner la valeur 3.14 à la constante PI d'une manière absolue, on ne pourrait pas changer la valeur de PI au cours de l'exécution d'un programme.

- Taux est une constante

Const Taux=0.25 permet de donner la valeur 0.25 à la constante Taux d'une manière absolue, on ne pourrait pas changer la valeur de Taux au cours de l'exécution d'un programme.

II. TYPES DE BASE

Lors de l'utilisation d'une variable, nous devons spécifier si cette variable était destinée à contenir des valeurs telles que 123 ou 12.45 ou "bonjour". Donc, nous devons spécifier le type de la variable.

Un type impose des limites que ça soit:

- dans les valeurs que peut prendre cette variable
- dans l'ensemble des opérations réalisables sur les variables de ce type.

1. TYPE ENTIER

Le type entier comprend les valeurs numériques entières, positives ou négatives. Les opérations arithmétiques effectuées sur ces nombres sont:

L'addition (+), la soustraction (-), la division rélle (/), la division entière (Div), la multiplication (*), le reste de la division (mod) et l'opposé unaire (-).

Exemple:

X: entier

Age: entier

Annee, mois, jour: entier

2. TYPE REEL

Le type réel comprend les valeurs décimales positives ou négatives. Les opérations appliquées sur le type entier sont valables pour le type réel.

Exemple:

X: réel

3. TYPE CARACTERE

Une variable de type caractère accepte les 26 lettres latines majuscules et les 26 lettres minuscules, les 10 chiffres arabes et les caractères spéciaux. Ces valeurs sont délimitées par 2 apostrophes.

Exemple:

C: caractère

C1, C2: caractère

4. TYPE CHAINE DE CARACTERES

Les chaînes de caractères sont délimitées par des guillemets. Exemples: "ali","123","vrai".

Exemple:

Nom: chaîne de caractère

5. TYPE BOOLEEN

Il s'appelle aussi type logique et il accepte deux valeurs: vrai ou faux (true ou false).

Il y a 2 types d'opérateurs qui s'appliquent sur les variables de type booléen:

- Les opérateurs logiques
- Les opérateurs de comparaison

Les opérateurs logiques qui s'appliquent sur les variables de type booléen sont:

- NON pour la négation
- ET pour la conjonction
- OU pour la disjonction

Le tableau suivant résume le résultat de l'application de ces opérateurs sur deux variables var1 et var2.

var1	var2	NON var1	var1 ET var2	var1 ou var2
Vrai	Vrai	Faux	Vrai	Vrai
Vrai	Faux	Faux	Faux	Vrai
Faux	Vrai	Vrai	Faux	Vrai
Faux	Faux	Vrai	Faux	Faux

Les opérateurs de comparaison qui s'appliquent sur les variables de type booléen sont:

- <
- >
- <=
- >=
- =
- <>

III. STRUCTURE GENERALE D'UN ALGORITHME

Un algorithme est composé d'une suite de déclaration de types, de variables, de constantes et d'instructions.

Sa structure générale ressemble à la suivante:

```
ALGORITHME <Nom_Algorithme>
DEBUT
<Déclaration de types>
<Déclaration de constantes et de variables>
<instructions>
FIN
```

IV. INSTRUCTION D'AFFECTATION

Le rôle d'une affectation consiste à placer une valeur dans une variable. La notation a l'allure suivante:

<Variable> ß valeur

Ou

<Variable> ß <expression>

Exemple:

A ß 5
B ß A+2

Exercice1:

Donner, pour chaque instruction la valeur contenue dans chaque variable.

Instructions	Contenu de A	Contenu de B	Contenu de C
A ß 1			
B ß A+3			
B ß 5			
C ß A+B			
A ß 2			
C ß B-A			

1 Cas des expressions numériques:

Une expression numérique peut contenir des constantes, des variables, des opérateurs numériques (+, -, *, /, Div, mod, - unaire) et des parenthèses. L'ordre de priorité de ces opérateurs est résumé dans le tableau suivant:

Opérateurs	Ordre d'évaluation
NON - unaire	De droite à gauche
* / Div Mod	De gauche à droite
+ -	De gauche à droite

Il est à noter que les parenthèses sont plus prioritaires.

Exercice 2

Remplir le tableau suivant:

Instructions	Contenu de A	Contenu de B	Contenu de C	Contenu de X
A ß 1				
B ß 2				
C ß 3				
X ß A+B+C				
X ß (A+B)*C				
X ß A+B/C				
X ß (A+B)/C				

Exercice 3

Soient 2 variables var1 et var2 déclarées comme des entiers. Ecrire un algorithme pour placer dans var1 la valeur 30 et dans var2 la valeur 45, puis échanger les valeurs des 2 variables.

V. INSTRUCTIONS D'ENTREE/SORTIE

Pour communiquer avec un programme, nous serons amenés à utiliser un périphérique d'entrée pour transmettre des informations (données) et un périphérique de sortie pour afficher certaines informations (résultats).

1. INSTRUCTION D'ECRITURE

Elle est appelée aussi instructions de sortie. L'instruction d'écriture "écrire" a pour rôle d'afficher des informations sous une forme compréhensible sur un périphérique de sortie. En général, le périphérique de sortie utilisé est l'écran.

Ecrire (var) a affiche sur l'écran le contenu de la variable var

Ecrire (100) a affiche sur l'écran 100

Ecrire ("Bonjour") a affiche sur l'écran le texte Bonjour var

Ecrire ("Bonjour", var) a affiche sur l'écran le texte Bonjour puis le contenu de la variable var

Exercice 1

Ecrire un algorithme Affich_Double qui met dans la variable entière val la valeur 4 et met dans la variable double le double de val puis affiche val et double.

SOLUTION

```
ALGORITHME Affich_Double
DEBUT
val, double: entier
val ß 4
double ß val * 2
Ecrire (" Le double de: ", val, "est: ", double)
FIN
```

Exercice 2

Ecrire un algorithme qui met dans les variables entières X et Y les valeurs 3 et 15 et affiche leur somme et leur produit.

SOLUTION

```
ALGORITHME SommeProduit
DEBUT
X, Y: entier
X ß 3
Y ß 15
```

Ecrire (" la somme est:", X + Y)

Ecrire ("le produit est:", X * Y)

FIN

2. INSTRUCTION DE LECTURE

Elle est appelée aussi instruction d'entrée. L'instruction de lecture "lire" a pour rôle de permettre à l'utilisateur d'entrer des valeurs au programme à partir de l'entrée standard. En général, l'entrée standard est le clavier.

lire (var) a lire une valeur à partir du clavier et la mettre dans la variable var

Exercice 1

Ecrire un algorithme qui lit une valeur entière et qui affiche son carré.

SOLUTION:

ALGORITHME carre

DEBUT

Val: entier

Ecrire ("donner une valeur entière:")

Lire (val)

Ecrire ("le carré de", val, "est: ",val * val)

FIN

Exercice 2

Ecrire un algorithme qui lit

- le prix hors taxe d'un article
- le nombre d'articles achetés
- le taux de la TVA

Et qui affiche le montant total toute taxe comprise.

SOLUTION:

ALGORITHME montant

DEBUT

PHT, TAUX: réel

NBRE: entier

Ecrire ("donner le prix hors taxe d'un article, le nombre d'articles achetés et le taux de la TVA")

Lire (PHT)

Lire (NBRE)

Lire (TAUX)

Ecrire ("le montant total est:", (PHT*NBRE) * (1+TAUX))

FIN

Chapitre 3: INSTRUCTIONS CONDITIONNELLES

Objectif du chapitre: connaître et manipuler les instructions conditionnelles

Plan du chapitre:

I. INTRODUCTION

Les instructions d'affectation, d'entrée et de sortie sont insuffisantes pour confronter des situations traitant des conditions. On aura besoin alors de choisir entre 2 ou plusieurs traitements selon la réalisation ou non d'une certaine condition d'où la notion de traitement conditionnel.

II. INSTRUCTION CONDITIONNELLE A UN CHOIX

La forme générale de cette instruction est:

```
Si <expression_logique> alors
<Traitements>
Fin si
```

La partie Traitement est composée d'une ou de plusieurs instruction(s). Cette partie est exécutée si expression_logique est vraie.

Exemple:

Ecrire un algorithme Verif_moyenne permettant de lire la moyenne d'un étudiant et d'afficher réussite si la moyenne est supérieure ou égale à 10.

```
ALGORITHME Verif_moyenne
DEBUT
Moyenne: réel
Ecrire ("Saisir une moyenne:")
Lire (moyenne)
Si moyenne >= 10 alors
Ecrire ("réussite")
Fin si
FIN
```

III. INSTRUCTION CONDITIONNELLE A DEUX CHOIX

La forme générale de cette instruction est:

```
Si <expression_logique> alors
<Traitements_A>
Sinon
 <Traitements_B>
Fin si
```

La partie Traitements_A est composée d'une ou de plusieurs instruction(s). Cette partie est exécutée si expression_logique est vraie. Dans le cas contraire (expression_logique est fausse), c'est la partie Traitements_B qui sera traitée.

Exemple:

Ecrire un algorithme Verif_moyenne permettant de lire la moyenne d'un étudiant et d'afficher réussite si la moyenne est supérieure ou égale à 10 et redoublement sinon.

```
ALGORITHME Verif_moyenne
DEBUT
Moyenne: réel
Ecrire ("Saisir une moyenne:")
Lire (moyenne)
Si moyenne >= 10 alors écrire ("réussite")
Sinon
 écrire ("redoublement")
Fin si
FIN
```

IV. INSTRUCTION CONDITIONNELLE IMBRIQUEE

La forme générale de cette instruction est:

```
Si <expression_logique_1> alors
<Traitements_A>
sinon
 si <expression_logique_2> alors
<Traitements_B>
sinon si <expression_logique_3> alors
<Traitement_C>
sinon ……..
Fin si
```

La partie Traitements_A est composée d'une ou de plusieurs instruction(s). Cette partie est exécutée si expression_logique_1 est vraie. Dans le cas contraire (expression_logique_1 est fausse), un test sera fait pour expression_logique_2. Si elle est vraie alors la partie Traitements_B sera traitée, sinon (expression_logique_2 est fausse), un teste sera fait pour expression_logique_3. Si elle est vraie alors la partie Traitements_C sera traitée sinon on termine les test qui suivent et ainsi de suite.

Exemple:

Ecrire un algorithme Verif_moyenne permettant de lire la moyenne d'un étudiant et d'afficher:

- Redoublement si la moyenne est entre 0 et 10
- Mention passable si la moyenne est entre 10 et 12
- Mention assez bien si la moyenne est entre 12 et 14
- Mention bien si la moyenne est entre 14 et 16
- Mention très bien si la moyenne est entre 16 et 20
- Un message indiquant d'entrer une moyenne entre 0 et 20

```
ALGORITHME Verif_moyenne
DEBUT
Moyenne:réel
Ecrire ("Saisir une moyenne:")
Lire (moyenne)
Si (moyenne >=0) et (moyenne <10) alors
écrire ("redoublement")
Sinon
Si (moyenne >=10) et (moyenne <12) alors
écrire ("Mention passable")
sinon
si (moyenne >=12) et (moyenne <14) alors
écrire ("Mention Assez bien")
sinon
si (moyenne >=14) et (moyenne <16) alors
écrire ("Mention  Bien")
sinon
si (moyenne >=16) et (moyenne<=20) alors
écrire ("Mention  Très bien")
sinon
écrire ("la moyenne doit être entre 0 et 20")
Fin si
FIN
```

Exercice 1

Donner les écrans d'affichage si l'utilisateur entre les moyennes suivantes:

Moyenne	Ecran d'affichage
16.57	……
10.59	……
9.20	……
18	……
31	……
14.25	……
12.01	……
0	……
15.02	……
11.99	……
13.01	……

Exercice 2

En se basant sur le tableau suivant, écrire un algorithme qui lit le montant d'une facture et affiche le montant de la remise et le montant à payer:

Montant facture	Taux remise
< 2000	0
<= 2000 et > 5000	1%
>= 5000	2%

V. INSTRUCTION CONDITIONNELLE AUX CHOIX MULTIPLES

La forme générale de cette instruction est:

```
Selon (var)
Valeur 1:  <Traitement_1>
Valeur 2: <Traitement_2>
……
Valeur n:  <Traitement_n>
Autre: <Autre_Traitement>
Fin selon
```

Exercice

Ecrire un algorithme Inscription permettant de lire un jour et d'afficher le niveau à s'inscrire pendant ce jour.

Si le jour est 8 ou 10 du mois alors il s'agit du premier niveau qui va s'inscrire, si le jour est le 11 du mois alors, c'est le deuxième niveau, si le jour est 12, 13 ou 14 alors il s'agit du troisième et quatrième niveau, si le jour est le 15 du mois alors il s'agit du cinquième niveau, si un autre alors pas d'inscription.

SOLUTION:

```
ALGORITHME Inscription
DEBUT
Jour: entier
Ecrire ("Saisir un jour:")
Lire (jour)
Selon (jour)
8, 10: écrire ("Inscription du niveau 1")
11 : écrire ("Inscription du niveau 2")
12, 13, 14: écrire ("Inscription du niveau 3 et 4")
15: écrire ("Inscription du  niveau 5")
Autre: écrire ("pas d'inscription")
Fin selon
FIN
```

Chapitre 4: INSTRUCTIONS ITERATIVES OU REPETITIVES

Objectif du chapitre: connaître et manipuler les instructions itératives et savoir faire la comparaison entre elles.

Plan du chapitre:

I. INTRODUCTION

Les structures itératives sont utilisées pour décrire les répétitions d'une instruction ou d'une suite d'instructions. Toute répétition d'instructions, appelée aussi boucle d'instructions, doit être finie et celle-ci sera contrôlée à l'aide d'une expression logique ou condition dont le changement de valeur provoque l'arrêt de la répétition ou la poursuite de l'exécution de ces instructions.

On distingue 3 formes de boucles: boucle répéter, boucle pour et boucle tant que.

II. INSTRUCTION REPETER

La forme générale de cette instruction est:

```
Répéter
<Traitements>
jusqu'à <Condition_D_Arrêt>
```

Exemple

Ecrire un algorithme permettant de lire une valeur entière strictement inférieure à 100.

SOLUTION

```
ALGORITHME Lire_Valeur_Inf_100
DEBUT
Val: entier
Répéter
    Ecrire ("Donner une valeur inférieure à 100:")
    Lire (val)
```

Jusqu'à val < 100

FIN

Remarque: l'instruction répéter est exécutée avant d'évaluer la condition donc elle est exécutée au moins une fois.

Exercice

Ecrire un algorithme Somme permettant de faire la somme des 10 premières valeurs entières à partir de la valeur 1.

SOLUTION

```
ALGORITHME Somme
DEBUT
i, s: entier
i Ñ 1
s Ñ 0
répéter
   s Ñ s + i
   i Ñ i + 1
jusqu' à i > 10
écrire ("La somme est:", s)
FIN
```

III. INSTRUCTION POUR

La forme générale de cette instruction est:

```
Pour <var> de <vi> à <vf> pas= <p>
Faire
<Traitements>
Fin pour
```

Remarque:

- La condition d'entrée dépend de vf. Il s'agit d'une relation entre le compteur var et vf.
- Le passage à la valeur suivante du compteur dépend du pas. Il s'agit d'une relation entre le compteur var et le pas p.
- Si on n'attribue pas une valeur de p alors, par défaut c'est la valeur 1.

Exemple

Ecrire un algorithme permettant d'afficher le texte Bonjour 50 fois.

SOLUTION:

```
ALGORITHME Afficher_Bonjour
DEBUT
i: entier
pour i de 1 à 50   pas=1   faire
   écrire ("Bonjour")
Fin pour
FIN
```

Inutile

Remarque: L'instruction pour est conseillée si on connaît le nombre de répétition à effectuer.

Exercice

Réécrire l'algorithme Somme avec la forme pour.

SOLUTION:

```
ALGORITHME Somme
DEBUT
i, s: entier
s Ñ 0
pour i de 1 à 10 faire
   s Ñ s + i
Fin pour
écrire ("La somme est:", s)
FIN
```

IV. INSTRUCTION TANT QUE

La forme générale de cette instruction est:

```
Tant que <Condition_Entrée>  faire
<Traitements>
Fin tant que
```

Exemple:

Réécrire l'algorithme Afficher_Bonjour avec la version Tant que.

SOLUTION

```
ALGORITHME Afficher_Bonjour
DEBUT
i: entier
i Ñ 1
Tant que i <= 50 faire
```

```
   Ecrire ("Bonjour")
   i Ñ i + 1
Fin Tant que
FIN
```

Remarque:

- L'instruction Tant que est exécutée si est seulement si la condition d'entrée est vraie
- L'instruction Tant que ne peut pas exécuter la partie <Traitements> si la condition d'entrée est, dès le début, fausse.

Exercice

Réécrire l'algorithme Somme avec la version Tant que.

SOLUTION

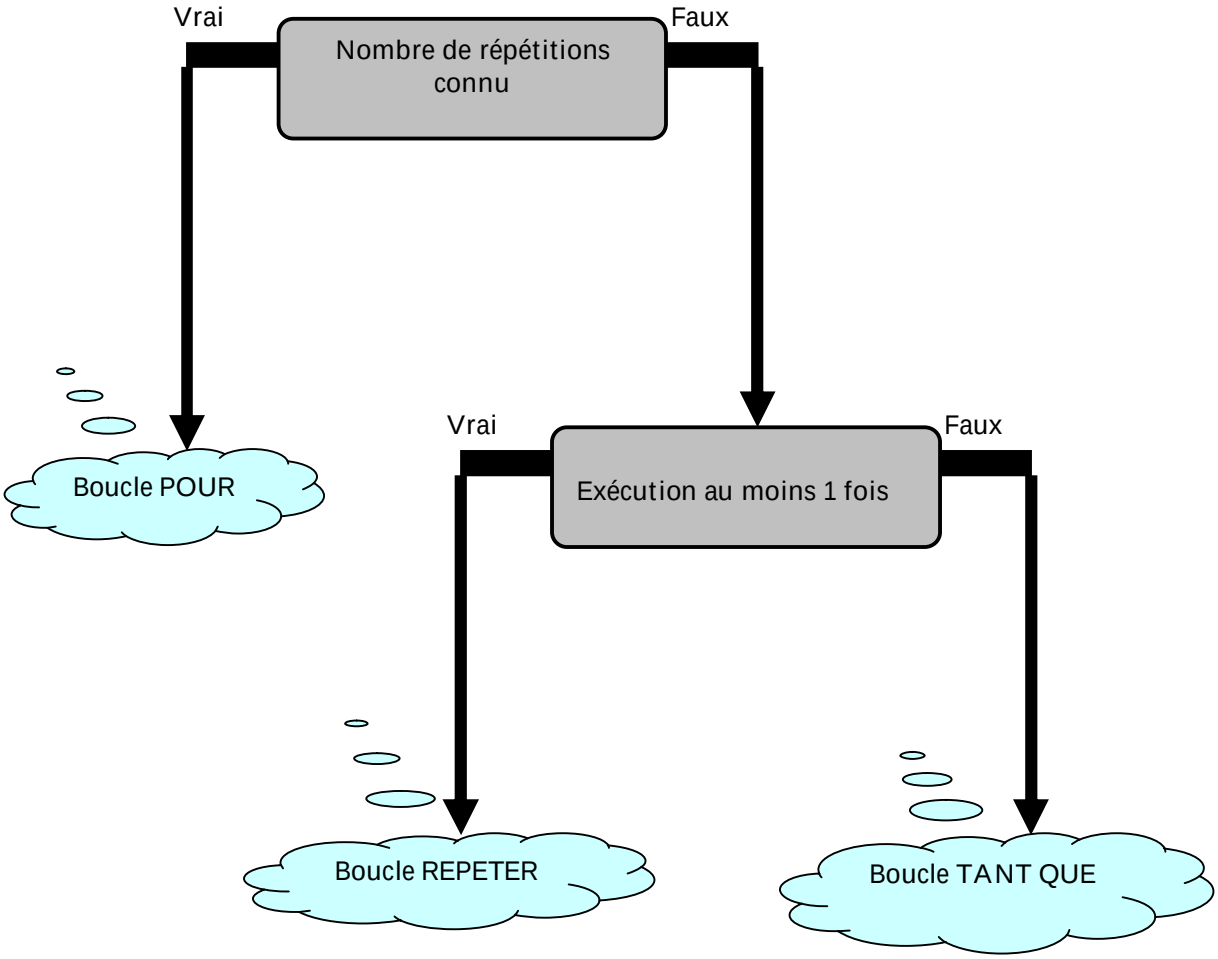
```
ALGORITHME Somme
DEBUT
i, s: entier
s Ñ 0
i Ñ 1
Tant que i <= 10 faire
   s Ñ s + i
   i Ñ i + 1
Fin tant que
écrire ("La somme est:", s)
FIN
```

V. COMPARAISON DES FORMES REPETITIVES

Pour choisir lesquelles des formes utiliser, il est conseillé de suivre la démarche suivante:

- Si on connaît le nombre de répétitions alors il est préférable d'utiliser POUR
- Sinon si on veut exécuter un traitement au moins une fois alors on utilise REPETER
- Sinon, on utilise tant que

La figure suivante résume la manière de choisir une forme répétitive:


Vrai
Nombre de répétitions connu
Faux
Boucle POUR
Vrai
Exécution au moins 1 fois
Faux
Boucle REPETER
Boucle TANT QUE

Chapitre 5: SOUS-PRIOGRAMMES

Objectif du chapitre: manipuler les procédures et les fonctions.

Plan du chapitre:

I. INTRODUCTION

La conception d'un algorithme procède en général par des affinements successifs. On décompose le problème à résoudre en sous-problèmes, puis ces derniers à leur tour, jusqu'à obtenir des problèmes faciles à résoudre. Pour chacun des sous-problèmes, on écrit un module appelé sous-programme.

Ainsi, la résolution du problème sera composée d'un algorithme principal et d'un certain nombre de sous-problèmes. L'algorithme principal a pour but d'organiser l'enchaînement des sous-programmes.

L'intérêt de l'analyse modulaire est:

- Répartir les difficultés entre les différents sous problèmes
- Faciliter la résolution d'un problème complexe
- Améliorer la qualité d'écriture du programme principal
- Minimiser l'écriture du code source dans la mesure où on utilise la technique de la réutilisation.

II. DEFINITION ET SYNTAXE

1. DEFINITION

Un sous-programme est une unité fonctionnelle formée d'un bloc d'instructions et éventuellement paramétré, que l'on déclare afin de pouvoir l'appeler par son nom en affectant des valeurs à ses paramètres (s'ils existent). Les données fournies au sous-programme et les résultats produits par ce dernier sont appelés des arguments ou des paramètres.

Un sous-programme peut être une procédure ou une fonction.

- Une procédure est un sous-programme ayant un nombre de paramètres, contenant un certain nombre d'instructions et admettant zéro ou plusieurs résultats.
- Une fonction est un sous-programme ayant un nombre de paramètres, contenant un certain nombre d'instructions et admettant au maximum un résultat unique affecté à son nom.

2. SYNTAXE

La syntaxe de définition d'une fonction est:

FONCTION <Nom_Fonction> (<Paramètres_Avec_Types_Et_Genres>) :<Type_Valeur_Retour>

Début Fonction

<Déclaration des variables>

<Instructions>

Retourner <Resultat>

Fin Fonction

La syntaxe de définition d'une procédure est:

PROCEDURE <Nom_Procédure> (<Paramètres_Avec_Types_Et_Genres>)

Début procédure

<Déclaration des variables>

<Instructions>

Fin procédure

III. GENRES DES PARAMETRES

Il existe trois genres de paramètres:

- Un paramètre donné: il s'agit d'un paramètre qui contient une valeur avant l'exécution du sous-programme. En cours d'exécution, la valeur ne change pas et reste inchangée jusqu'à la fin de l'exécution. On le symbolise par DON.
- Un paramètre résultat: il s'agit d'un paramètre qui ne contient pas de valeur avant l'exécution du sous-programme. En cours d'exécution, une valeur est affectée à ce paramètre afin d'être gardée pour la fin d'exécution. On le symbolise par RES.
- Un paramètre donné/résultat: il s'agit d'un paramètre qui contient une valeur avant l'exécution du sous-programme. Cette valeur change en cours d'exécution. Le paramètre aura une valeur à la fin qui n'est pas égale à celle de début d'exécution. On le symbolise par DONRES.

IV. APPEL DE SOUS-PROGRAMME

L'appel d'une fonction se fait comme suit:

<Var>™ <Nom_Fonction>(<Parametres_Effectifs>)

ou

écrire(<Nom_Fonction>(<Parametres_Effectifs>))

L'appel d'une procédure se fait comme suit:

<Nom_Procedure>(<Parametres_Effectifs>)

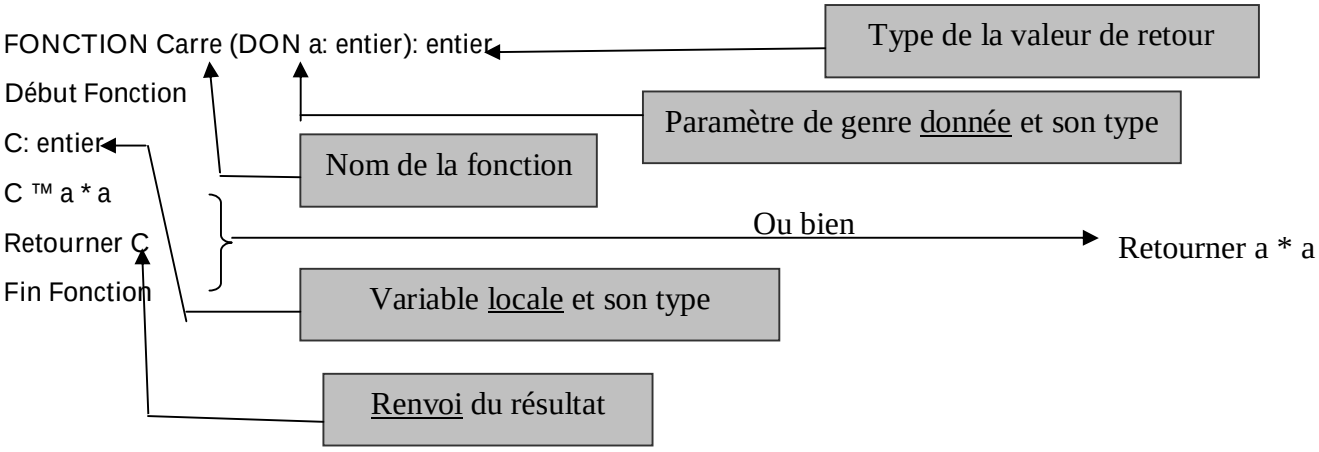
Généralement, c'est l'algorithme principal qui appelle ou invoque des sous-programmes. D'où, il est baptisé l'appelant et le sous-programme est nommé l'appelé. Cependant, un sous-programme peut appeler un autre sous-programme. Par la suite, le sous-programme qui appelle est l'appelant et celui qui a subit l'appel est l'appelé.

Lors de l'appel d'un sous-programme, deux formes de paramètres entrent en jeu : les paramètres formels et les paramètres effectifs.
Les paramètres spécifiés dans la définition de la fonction sont qualifiés de paramètres formels, par contre les paramètres qui seront transmis à la fonction lors de l'appel sont appelés des paramètres effectifs.
Remarque : Les paramètres formels et les paramètres effectifs doivent correspondre en nombre, en type et en ordre. Les noms peuvent se différer.

V. APPLICATIONS

Exemple1

Ecrire une fonction qui permet de calculer le carré d'un entier a donné



Exemple2

Ecrire une fonction qui calcule la somme de deux entiers a et b

FONCTION Somme (DON a entier, DON b entier):entier
Début Fonction
s: entier
s ™ a + b
Retourner s
Fin Fonction
Ou bien
Retourner a + b

Exemple3

Ecrire une procédure qui permet de lire 2 entiers a et b

PROCEDURE Lecture (RES a entier, RES b entier)

Début Procédure

Ecrire ("Donner deux entiers:")

Lire(a)
Lire(b)

Ou bien → Lire (a,b)

Fin Procédure

Exemple 4

Ecrire une procédure qui permute 2 variables entières a et b

PROCEDURE Permuter (DONRES a: entier, DONRES b: entier)

Début Procédure

Aux: entier

Aux f a

A f b

B f Aux

Fin Procédure

Exemple 5

Appelez les sous-programmes vues auparavant pour écrire un algorithme TEST permettant de:

- lire deux entiers x et y
- calculer la somme de x et y
- calculer le carré de x et le carré de y

ALGORITHME TEST

DEBUT

x, y, Som, CX, CY: entier ← Variables globales et leurs types

Lecture (x,y) ← Appel de la procédure Lecture avec des paramètres effectifs x et y

Som™ Somme(x, y) ← Appel de la fonction Somme avec des paramètres effectifs x et y
Le résultat calculé par la fonction Somme serait conservé dans la variable Som

Ecrire ("la somme des deux valeurs entrées est:", Som)

CX ™ Carre(x) ← Appel de la fonction Carre avec un paramètre effectif x
Le résultat calculé par la fonction Carre serait conservé dans la variable CX

Ecrire ("Le carré de x est:", CX)

CY ™ Carre(y) ← Appel de la fonction Carre avec un paramètre effectif y
Le résultat calculé par la fonction Carre serait conservé dans la variable CY

```
Ecrire ("Le carré de y est:", CY)
FIN
```

L'algorithme TEST a appelé les fonctions Somme et Carre et la procédure Lecture, donc il est nommé l'appelant. Par contre, les deux fonctions et la procédure ont subit l'action de l'invocation, donc elles sont nommées, les appelés.

Chapitre 6: TABLEAUX

Objectif du chapitre: manipuler les tableaux.

Plan du chapitre:

I. Introduction
II. Tableaux à une dimension
III. Tableaux à deux dimensions

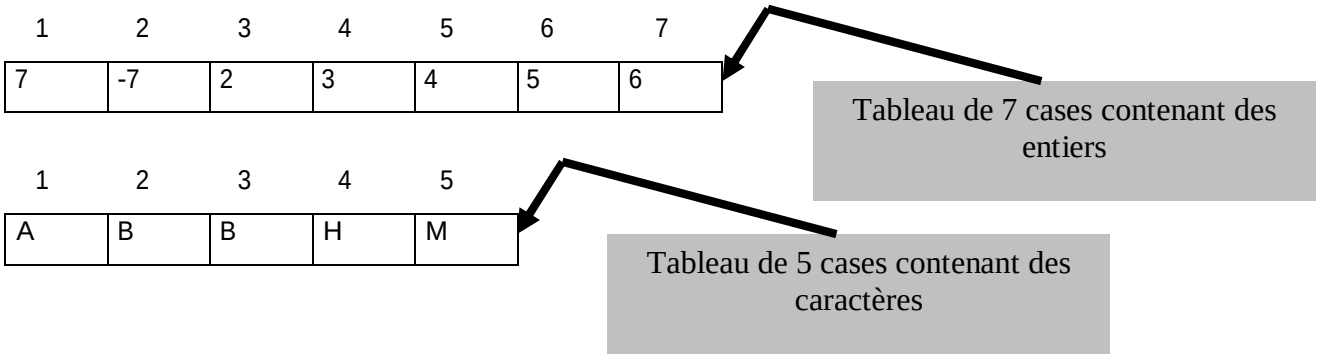
I. INTRODUCTION

1 Définition:

Un tableau est une structure de données formée d'éléments de même type, que nous pouvons accéder grâce à un indice.

En effet, un tableau n'est qu'une succession de boîtes ou cases ou zones mémoires assurant le rangement de plusieurs éléments du même domaine (même type).

Exemple:



1 Caractéristiques:

Un tableau est caractérisé par:

- un nom: un identificateur du tableau
- Le type de ses éléments qui peut être un entier, un réel, un caractère, une chaîne de caractères, un autre tableau ou un type composé.
- L'indice qui est l'indicateur assurant le parcours des cases du tableau une à une. L'indice peut être tout type dont les éléments possèdent un successeur (les types scalaires). En général, un indice est de type entier.

II. TABLEAUX A UNE DIMENSION

Les deux exemples ci-dessus sont des tableaux à une seule dimension ou unidimensionnels. Un tableau unidimensionnel est composé d'une seule ligne et de plusieurs colonnes. Le nombre de colonne indique la taille ou la longueur du tableau.

1. DECLARATION D'UN TABLEAU

La syntaxe générale de déclaration d'un tableau unidimensionnel est la suivante:

```
<Nom_Tableau> tableau [<Borne_Inf>..<Borne_Sup>] de <Type_Element>
```

Remarque: <Borne_Inf> et <Borne_Sup> est un intervalle de valeurs scalaires. En général, il s'agit de valeurs entières. Il est à ajouter qu'un tableau n'a de sens que si la borne Borne_Inf soit inférieure ou égale à la borne Borne_Sup.

Exemples:

```
Notes: tableau [1..30] de réel
TabCar: tableau [1..26] de caractères
TabEntier: tableau [1..20] de entier
```

2. ACCES A UN ELEMENT D'UN TABLEAU

La syntaxe générale pour accéder à un élément d'un tableau est:

<Nom_Tableau> [<Indice_De_L_Element>]

L'indice de l'élément est une valeur comprise entre Borne_Inf et Borne_Sup

Exemples

```
Accès au 20 ème élément du tableau Notes: Notes [20]
Accès au 24 ème élément du tableau TabCar: TabCar [24]
Accès au 5 ème élément du tableau TabEntier: TabEntier [5]
```

3. OPERATIONS ELEMENTAIRES SUR UN TABLEAU

a. REMPLISSAGE D'UN TABLEAU

Activité 1:

Ecrire une procédure qui permet de remplir un tableau T de 10 valeurs entières.

SOLUTION

```
procédure Remplissage (RES T: tableau [1..10] de entier)
Début Procédure
K: entier
pour k de 1 à 10 faire
écrire ("Donner la valeur de la case ", k)
lire (T[k])
Fin pour
```

```
Fin Procédure
```

b. AFFICHAGE DES ELEMENTS D'UN TABLEAU

Activité 2 :

Soit le tableau T vu à l'activité 1, écrire une procédure qui affiche tous ses éléments.

SOLUTION

```
procédure Affichage (DON T: tableau [1..10] de entier)
Début procédure
k: entier
pour k de 1 à 10 faire
écrire("La ", k, " ème case du tableau T est:", T[k])
Fin pour
Fin procédure
```

c. CALCUL DE LA SOMME DES ELEMENTS D'UN TABLEAU

Activité 3 :

Soit le tableau T vu à l'activité 1 et 2, écrire une fonction qui calcule la somme de se éléments.

SOLUTION

```
Fonction somme (DON T: tableau [1..10] de entier): entier
Début Fonction
K: entier
S ß 0
pour k de 1 à 10 faire
S ß S + T[k]
Fin pour
Retourner S
Fin Fonction
```

4. ALGORITHMES DE TRI

a. DEFINITION

On désigne par tri, l'opération qui consiste à ordonner une suite de valeurs suivant une relation d'ordre (ordre croissant et décroissant).

Il y a plusieurs façons ou techniques pour effectuer un tri. On va considérer 3 tris:

- tri par sélection
- tri par insertion
- tri à bulle

b. TRI PAR SELECTION

t u Principe de tri

Le principe de ce tri est défini comme suit:

- On recherche le plus grand des n éléments du tableau (dans le cas d'un tri décroissant ou le plus petit dans le cas d'un tri croissant)
- On échange cet élément avec le premier élément du tableau
- Le plus grand élément se trouve alors en première position. On peut appliquer alors les deux opérations précédentes aux n – 1 éléments restants, puis aux n – 2 éléments restants…et ceci jusqu'à ce qu'il ne reste plus qu'un seul élément (le dernier lequel est alors le plus petit).

t u Exemple

Soit le tableau suivant, trions-le par sélection suivant un ordre croissant.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
20	4	43	-1	5	2	0	10	5	12	14	9	8	-2

Etape 1

1	2	3	4	5	6	7	8	9	10	11	12	13	14
20	4	43	-1	5	2	0	10	5	12	14	9	8	-2

Ind =

Min =

Etape 2

1	2	3	4	5	6	7	8	9	10	11	12	13	14

Ind =

Min =

Etape 3

1	2	3	4	5	6	7	8	9	10	11	12	13	14

Ind =

Min =

Etape 4

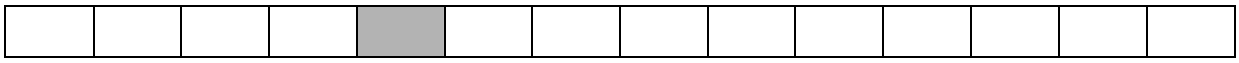
1	2	3	4	5	6	7	8	9	10	11	12	13	14

ind =

Min =

Etape 5

1	2	3	4	5	6	7	8	9	10	11	12	13	14

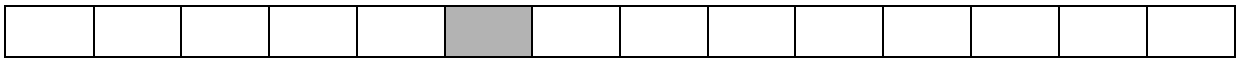


ind =

Min =

Etape 6

1 2 3 4 5 6 7 8 9 10 11 12 13 14



ind =

Min =

Etape 7

1 2 3 4 5 6 7 8 9 10 11 12 13 14

ind =

Min =

Etape 8

1 2 3 4 5 6 7 8 9 10 11 12 13 14

Ind =

Min =

Etape 9

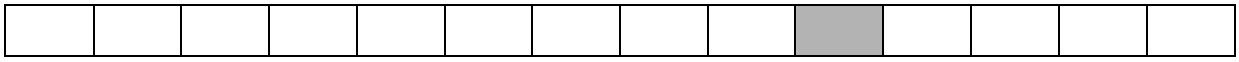
1 2 3 4 5 6 7 8 9 10 11 12 13 14

ind =

Min =

Etape 10

1 2 3 4 5 6 7 8 9 10 11 12 13 14



ind =

Min =

Etape 11

1 2 3 4 5 6 7 8 9 10 11 12 13 14

ind =

Min =

Etape 12

1 2 3 4 5 6 7 8 9 10 11 12 13 14

ind =

Min =

Etape 13

1	2	3	4	5	6	7	8	9	10	11	12	13	14

ind =

Min =

t u Algorithme

```
ALGORITHME Tri_Selection
DEBUT
T: tableau [1..14] de entier
i, min,ind, temp: entier
Pour ind de 1 à 13 faire
        min ß ind
        pour i de ind + 1 à 14 faire
                si T[i] < T[min] alors    min ß i    fin si
        fin pour
        temp ß T[ind]
        T[ind] ß T[min]
        T[min] ß temp
Fin pour
FIN
```

c. TRI PAR INSERTION

t u Principe de tri

Le principe du tri par insertion consiste à insérer un par un les éléments en les plaçant correctement: on insère le second élément du tableau dans sa place au niveau du sous tableau constitué du premier élément, on insère ensuite le troisième élément du tableau dans sa place au niveau du sous tableau constitué du premier et deuxième élément et ainsi de suite jusqu'au dernier élément.

t u Exemple

Soit le tableau suivant, trions-le par insertion suivant un ordre croissant.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
20	4	43	-1	5	2	0	10	5	12	14	9	8	-2

Etape 1

1	2	3	4	5	6	7	8	9	10	11	12	13	14
20	4	43	-1	5	2	0	10	5	12	14	9	8	-2

Ind =

j =

Etape 2

1	2	3	4	5	6	7	8	9	10	11	12	13	14

Ind =

j =

Etape 3

1	2	3	4	5	6	7	8	9	10	11	12	13	14

Ind =

j =

Etape 4

1	2	3	4	5	6	7	8	9	10	11	12	13	14

Ind =

j =

Etape 5

1	2	3	4	5	6	7	8	9	10	11	12	13	14

Ind =

j =

Etape 6

1	2	3	4	5	6	7	8	9	10	11	12	13	14

Ind =

j =

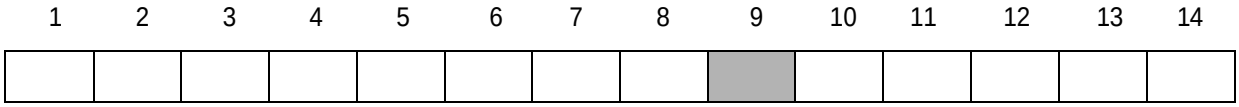
Etape 7

1	2	3	4	5	6	7	8	9	10	11	12	13	14

Ind =

j =

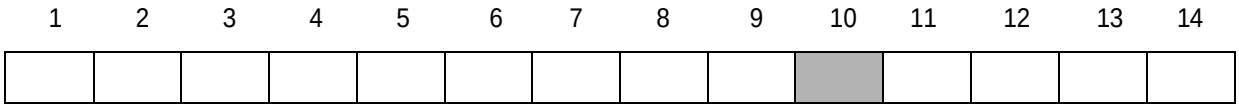
Etape 8



Ind =

j =

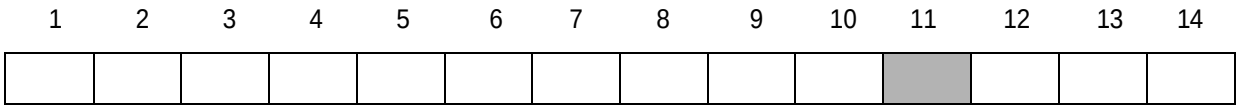
Etape 9



Ind =

j =

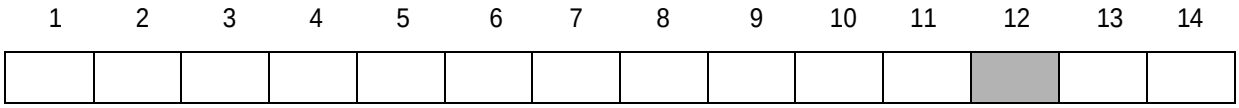
Etape 10



Ind =

j =

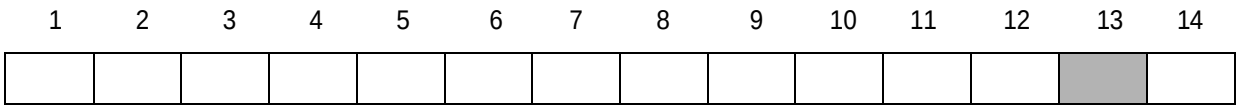
Etape 11



Ind =

j =

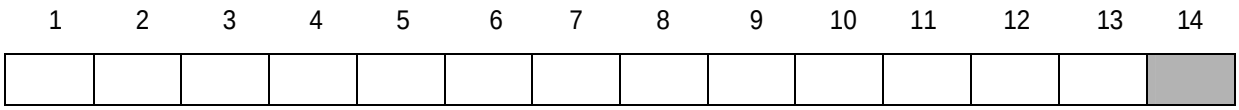
Etape 12



Ind =

j =

Etape 13



Ind =

j =

t u Algorithme

```
ALGORITHME Tri_Insertion
DEBUT
T: tableau [1..14] de entier
temp, ind, j: entier
pour ind de 2 à 14 faire
```

```
        temp ß T[ind]
        j ß ind – 1
        Tant que j > 0 et T[j] > temp faire
                T[j + 1] ß T[j]
                j ß j – 1
        Fin tant que
        T[j + 1] ß temp
Fin pour
FIN
```

d. TRI A BULLE

t u Principe de tri

Le principe du tri à bulle consiste à parcourir l'ensemble du tableau, depuis sa fin jusqu'à son début, en comparant deux éléments consécutifs et en les inversant s'ils sont mal classés. On se retrouve ainsi avec le plus petit élément placé en tête du tableau (cas de tri croissant et le plus grand en tête de tableau s'il s'agit d'un tri décroissant).

On renouvelle une telle opération (appelée passe) avec les n – 1éléments restants, puis les n – 2 éléments restants et ainsi de suite jusqu'à ce que:

- soit l'avant dernier élément ait été classé (le dernier étant alors obligatoirement à sa place)
- soit qu'aucune permutation n'ait eu lieu pendant la dernière passe (ce qui prouve alors que l'ensemble du tableau est convenablement ordonné)

t u Exemple

Soit le tableau suivant, trions-le par insertion suivant un ordre croissant.

1	2	3	4	5
20	4	43	-1	5

Etape 1

1	2	3	4	5
20	4	43	-1	5

i=

permute=

Etape 2

1	2	3	4	5

i=

permute=

Etape 3

1	2	3	4	5

i=

permute=

Etape 4

1	2	3	4	5

i=

permute=

t u Algorithme

```
ALGORITHME Tri_Bulle
DEBUT
T: tableau [1..5] de entier
i, temp: entier
permute: booléen
répéter
permute ß faux
pour i de 5 à 2 faire
si T[i - 1] > T[i] alors
        permute ß vrai
        temp ß T[i – 1]
        T[i – 1] ß T[i]
        T[i] ß temp
Fin si
Fin pour        Fin pour
jusqu'à non (permute)
FIN
```

5. ALGORITHMES DE RECHERCHE

a. DEFINITION

La recherche d'une information est une opération fréquemment rencontrée dans les traitements des suites de valeurs. Il y a deux types de recherche dans un tableau:

- recherche séquentielle
- recherche dichotomique

b. RECHERCHE SEQUENTIELLE

La recherche séquentielle a été vue dans ce cours sur un tableau non trié. Dans ce qui suit, elle va se faire sur un tableau trié. Ainsi le parcours du tableau est partiel, il ne sera total que s'i l'élément recherché figure à la fin du tableau.

Exercice

Soit un tableau T de 40 entiers et trié suivant un ordre croissant. Ecrire une fonction qui permet de rechercher un entier x donné. La fonction renvoie vrai si l'élément existe et faux sinon.

SOLUTION

```
Fonction Recherche_Seq(T: tableau[1..40] de entier, x: entier): booléen
DEBUT FONCTION
i: entier
i ß 1
Tant que T[i] < x et i <= 40 faire
        i ß i + 1
Fin tant que
Si T[i] > x ou i > 40 alors retourner faux
Sinon retourner vrai
Fin si
FIN FONCTION
```

c. RECHERCHE DICHOTOMIQUE

t u Principe de la recherche

Le principe de cette recherche consiste à comparer la valeur recherchée à l'élément central de la suite, si ce n'est pas la bonne, un test permet de trouver dans quelle moitié de la suite on trouvera la valeur. On continue jusqu'à ce que la suite soit de taille 1.

Exemple

Soit le tableau T suivant trié suivant un ordre décroissant. Rechercher la valeur 26.

Etape 1

1	2	3	4	5	6	7	8	9	10	11
1	3	4	7	9	13	26	43	52	100	101

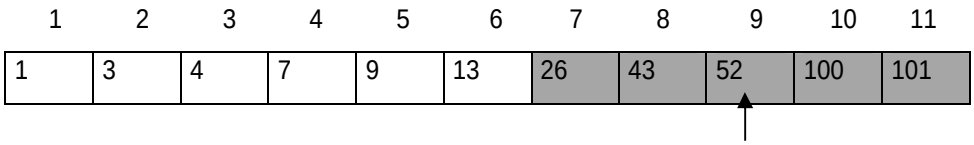
Binf = 1

Bsup = 11

Mil = 1 + 11 / 2 = 6

26 > T[6] donc 26 existera dans la partie du tableau allant de 7 à 11.

Etape 2



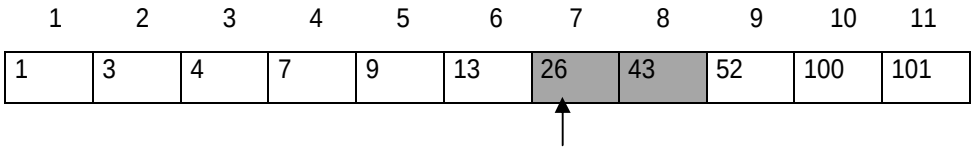
Binf = 7

Bsup = 11

Mil = 7+11 / 2 = 9

26 > T[9] donc 26 existera dans la partie du tableau allant de 7 à 8.

Etape 3



Binf = 7

Bsup = 8

Mil = 7 + 8 / 2 = 7

26 = T[7]

Trouvé ''''''

t u Algorithme

```
ALGORITHME Recherche_Dich
DEBUT
T: tableau [1..11] de entier
x, binf,bsup, mil: entier
écrire ("Donner la valeur à rechercher:")
lire (x)
binf ß 1
bsup ß 11
répéter
        mil ß (binf + bsup) / 2 / / il s'agit d'une division entière
        si T[mil] > x alors
                bsup ß mil – 1
        sinon     binf ß mil + 1
        fin si
jusqu' à T[mil] = x ou bsup < binf
si bsup < binf alors        écrire ("EXISTE PAS")
sinon   si T[mil] = x alors          écrire ("EXISTE")
fin si
FIN
```

III. TABLEAUX A DEUX DIMENSIONS

1. DECLARATION D'UN TABLEAU BIDIMENSIONNEL

Un tableau bidimensionnel est appelé aussi matrice. Elle est composée de lignes et de colonnes. La syntaxe de déclaration d'une matrice est:

```
<Nom_Matrice>: tableau [<Bligne1>..<Bligne2>] [<Bcolonne1>..<Bcolonne2>] de <Type_Elements>
```

Remarque: <Bligne1>..<Bligne2> et <Bcolonne1>..<Bcolonne2> sont des intervalles de valeurs scalaires. En général, il s'agit de valeurs entières. Il est à ajouter qu'une matrice n'a de sens que si la borne Bligne1 soit inférieure ou égale à la borne Bligne2 et la borne Bcolonne1 soit inférieure ou égale à la borne Bcolonne2.

Exemples:

```
Matrice1: tableau [1..30] [1..20] de réel
MatriceCar: tableau [1..26] [1..5] de caractères
MatriceEnt: tableau [1..4] [1..5] de entier
```

2. ACCES A UN ELEMENT D'UNE MATRICE

La syntaxe générale pour accéder à un élément d'une matrice est:

<Nom_Matrice> [indice1] [indice2]

Avec:

indice1 est compris entre Bligne1 et Bligne2

indice2 est compris entre Bcolonne1 et Bcolonne2

3. OPERATIONS ELEMENTAIRES SUR UNE MATRICE

a. REMPLISSAGE D'UNE MATRICE

Activité 1:

Ecrire une procédure permettant de remplir une matrice M d'entiers à 3 lignes et 2 colonnes.

SOLUTION

```
procédure Remplissage_Matrice (RES M: tableau [1..3][1..2] de entier)
Début Procédure
i, j: entier
pour i de 1 à 3 faire
pour j de 1 à 2 faire
écrire("Donner la valeur de la", i, " ème ligne et", j, " ème colonne ")
lire (M[i][j])
Fin pour
Fin pour
Fin  Procédure
```

Activité 2:

Ecrire une procédure permettant d'afficher tous les éléments de la matrice déjà définie dans l'activité1.

Activité 3:

Ecrire une fonction permettant de calculer la somme de tous les éléments de la matrice déjà définie dans l'activité1.

Chapitre 7: CHAINES DE CARACTERES

Objectif du chapitre: manipuler les chaines de caractères.

Plan du chapitre:

I. Introduction
II. Opérations élémentaires sur les chaines de caractères
III. Applications

I. INTRODUCTION

Une chaîne de caractères est considérée comme un tableau de caractères. Elle est délimitée par 2 guillemets. Une chaîne vide est représentée par 2 guillemets accolés.

La déclaration d'une chaîne de caractères se fait suivant la syntaxe suivante:

```
<Nom_Variable_Chaine>: chaine [longueur]
```

II. OPERATIONS ELEMENTAIRES SUR LES CHAINES DE CARACTERES

a. ENTREE ET SORTIE STANDARD DE CHAINES

- *Ecriture de chaînes*

Pour l'écriture de chaînes sur la sortie standard, on utilise la fonction écrire.

- *Lecture de chaînes*

Pour la lecture des chaînes à partir de l'entrée standard, on utilise la fonction lire.

Exemple:

```
Ch:chaine (30)
Ecrire ("Donner une chaine de caractères:")
Lire (ch)
Ecrire ("La chaine lue est:", ch)
```

b. COPIE D'UNE CHAINE DE CARACTERES DANS UNE AUTRE

La fonction Copier_chaine copie le contenu d'une chaîne de caractères (chaine_source) vers une autre chaîne (chaine_destination).

La sysntaxe est:

```
Copier_chaine (chaine_destination, chaine_source)
```

Exemple:

```
Ch: chaine (30)
Copier_Chaine (ch,"Bonjour")
```

Ecrire (ch)

¢ Ce qui va être affiché sur écran est: Bonjour

` Il est à signaler qu'il est interdit de remplacer Copier_Chaine (ch,"Bonjour") par ch ß "Bonjour"

c. RECHERCHE DE LA LONGUEUR D'UNE CHAINE

La fonction Longueur_chaine renvoie le nombre de caractères contenus dans une chaîne.

La syntaxe est:

Longueur_Chaine (chaine)

Exemple:

Ch: chaine (30)

L:entier

Copier_Chaine (ch,"Bonjour")

L ß Longueur_Chaine (ch)

Ecrire (L)

¢ Ce qui va être affiché sur écran est: 7

d. AJOUT DU CONTENU D'UNE CHAINE A UNE AUTRE

Il peut être nécessaire d'ajouter le contenu d'une chaîne de caractères à celui d'une autre chaîne. Si, par exemple, une chaîne contient un nom de fichier et une autre chaîne un nom de dossier, il est possible, en les rassemblant, de donner le chemin d'accès au fichier. Ce mécanisme d'addition de chaînes porte, en algorithmique et programmation, le nom de concaténation de chaînes de caractères. La fonction qui assure ce mécanisme est: Concaténer_Chaine. Sa syntaxe est:

Concaténer_Chaine (chaine_Destination, chaine_source)

Le résultat de la concaténation va se trouver au niveau de la chaîne destination.

Exemple:

Nom_Fichier: chaine (14)

Extension: chaine (5)

Copier_Chaine (Nom_Fichier,"Projet")

Copier_Chaine (extension,".doc")

Concaténer_Chaine (Nom_Fichier, extension);

Écrire ("Le nom complet du fichier est: ", Nom_Fichier);

¢ Ce qui va être affiché sur écran est: Projet.doc

e. COMPARAISON DE DEUX CHAINES DE CARACTERES

Pour tester l'égalité de deux chaînes, il y a la fonction Comparer_Chaine. Cette dernière détermine en plus, si une chaîne et plus grande qu'une autre, et ceci, par exemple, à des fins de tri. Cette fonction admet 2 paramètres chaine1 et chaine2.

Comparer_Chaine est une fonction qui retourne:

- 0 si chaine1=chaine2

- une valeur positive si chaine1 > chaine2
- une valeur négative si chaine1 < chaine2.

Exemple:

```
S1, S2: chaine(30)
Écrire ("Saisir 2 chaines")
Lire (S1, S2)
Si (non Comparer_Chaine (S1, S2)) alors
Ecrire ("Les 2 chaînes sont égales")
Sinon  écrire ("les 2 chaînes ne sont pas égales")
Fin si
```

f. VERIFICATION DU TYPE DE CARACTERE

Il existe des fonctions qui permettent de vérifier le type d'un caractère (majuscule, minuscule, chiffre, etc.). Le tableau ci-dessous résume quelques fonctions:

Fonction	Rôle	Valeur de retour
Estalnum	Vérifie si le caractère fourni en paramètre est un caractère alphanumérique (chiffre, lettre)	Estalnum (caractère) retourne: - 0 si le caractère n'est pas alphanumérique - <> 0 sinon
Estalpha	Vérifie si le caractère fourni en paramètre est un caractère alphabétique (lettre)	Estalpha (caractère) retourne: - 0 si le caractère n'est pas alphabétique - <> 0 sinon
EstChiffre	Vérifie si le caractère fourni en paramètre est un caractère numérique (chiffre)	EstChiffre (caractère) retourne: - 0 si le caractère n'est pas un chiffre - <> 0 sinon
EstMin	Vérifie si le caractère fourni en paramètre est une lettre minuscule	EstMin (caractère) retourne: - 0 si le caractère n'est pas une lettre minuscule - <> 0 sinon
EstMaj	Vérifie si le caractère fourni en paramètre est une lettre majuscule	EstMaj (caractère) retourne: - 0 si le caractère n'est pas une lettre majuscule - <> 0 sinon
EstEspace	Vérifie si le caractère fourni en paramètre est un espace	EstEspace (caractère) retourne: - 0 si le caractère n'est pas un espace - <> 0 sinon
VersMaj	Convertit un caractère, fourni en paramètre, en majuscule	VersMaj (caractère) convertit caractère en majuscule
VersMin	Convertit un caractère, fourni en paramètre, en minuscule	VersMin (caractère) convertit caractère en minuscule

III. APPLICATION

Ecrire un algorithme qui:

- lit une chaîne de caractère dont la longueur ne dépasse pas 50 caractères
- calcule et affiche le nombre de caractères alphanumériques, de lettres, de chiffres, de caractères majuscules, de caractères minuscules et d'espaces.

Chapitre 8: TYPE STRUCTURE

Objectif du chapitre: manipuler le type structure

Plan du chapitre:

I. Définition
II. Structures simples
III. Structures complexes
IV. Tableaux de structures
V. Application

I. DEFINITION

Une structure ou encore un enregistrement ets classé parmi les types avancés. Il contient un ou plusieurs champs groupés sous le même nom pour être traités comme une seule entité. Ces champs sont hétérogènes dans la mesure où leurs types sont différents.

Les champs d'une structure sont appelés membres et ils peuvent avoir l'un des types déjà définis y compris les tableaux et les structures.

II. STRUCTURES SIMPLES

1. DEFINITION

On définit le type structure comme suit:

<Nom_Structure>: structure

Membre 1: type

...

Membre n: type

Fin structure

Exemples:

z Définition de la structure Etudiant:

Etudiant: structure

Nom: chaine (50)

Prenom: chaine (25)

Adresse: chaine (50)

DateNaiss: chaine (10)

Fin structure

z Définition de la structure Date

Date: structure

Jour: entier

Mois: entier

Annee: entier

Fin structure

z Définition de la structure produit

Produit: structure

Code: chaine (6)

Designation: chaine (50)

Prix: réel

Fin structure

2. DECLARATION D'UNE VARIABLE DE TYPE STRUCTURE SIMPLE

P: Produit ØØØpermet de déclarer une varible P de type Produit.

E1, E2: Etudiant ØØØ permet de déclarer 2 varables E1 et E2 de type Etudiant.

3. ACCES AUX MEMBRES D'UNE STRUCTURE SIMPLE

Pour faire référence à un membre particulier, on sépare le nom de la variable de type la structure concernée de celui du membre visé, avec l'opérateur (.).

Exemples:

p.code ØØpermet d'accéder au champ code du produit p,

p.prix ØØpermet d'accéder au champ prix du produit p,

e1.nom ØØpermet d'accéder au champ nom de l'étudiant e1,

e2.adresse ØØpermet d'accéder au champ adresse de l'étudiant e2.

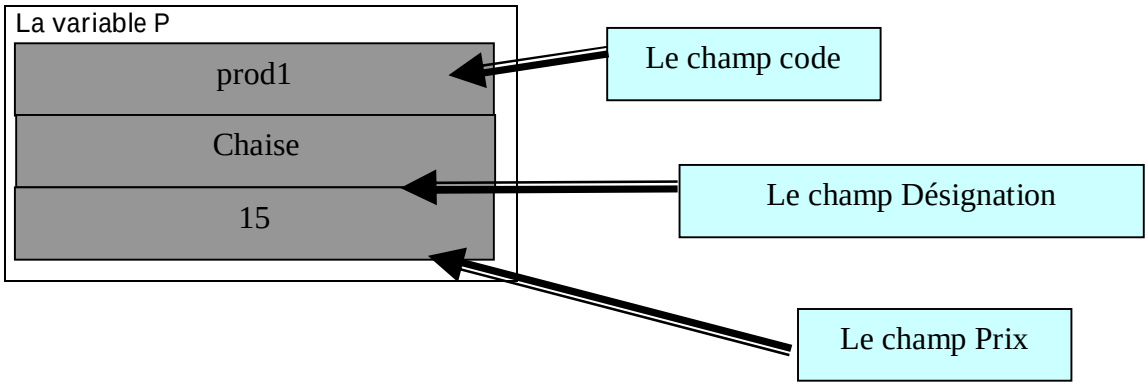
Exercice

Ecrire un algorithme permettant de créer une variable p de type la structure produit et de l'affecter les informations suivantes: (prod1, chaise, 15).

SOLUTION

```
ALGORITHME Essai
DEBUT
Produit: structure
        Code: chaine (6)
        Designation: chaine (50)
        Prix: réel
Fin structure
P: produit
Copier_Chaine (p.code, "prod1")
Copier_Chaine (p.designation, "chaise")
p.prix ß 15
FIN
```

Dans la mémoire centrale, la variable p de structure produit sera représentée comme suit:



III. STRUCTURES COMPLEXES

Revenons à l'exemple de la structure Etudiant et supposons que l'adresse n'est pas une chaîne de caractères mais elle est une structure qui est définie comme suit:

Adr: structure
    Rue: entier
    Cite chaine (20)
    Gouvernorat: chaine (50)
    CodePostal: entier
    Pays: chaine (50)
Fin structure

Supposons de même qu'on recourt à la structure Date déjà défini pour définir le champ DateNaiss. La structure Etudiant serait alors définie comme suit:

Etudiant: structure
    Nom: chaine (50)
    Prenom: chaine (25)
    Adresse: Adr
    DateNaiss: Date
Fin structure

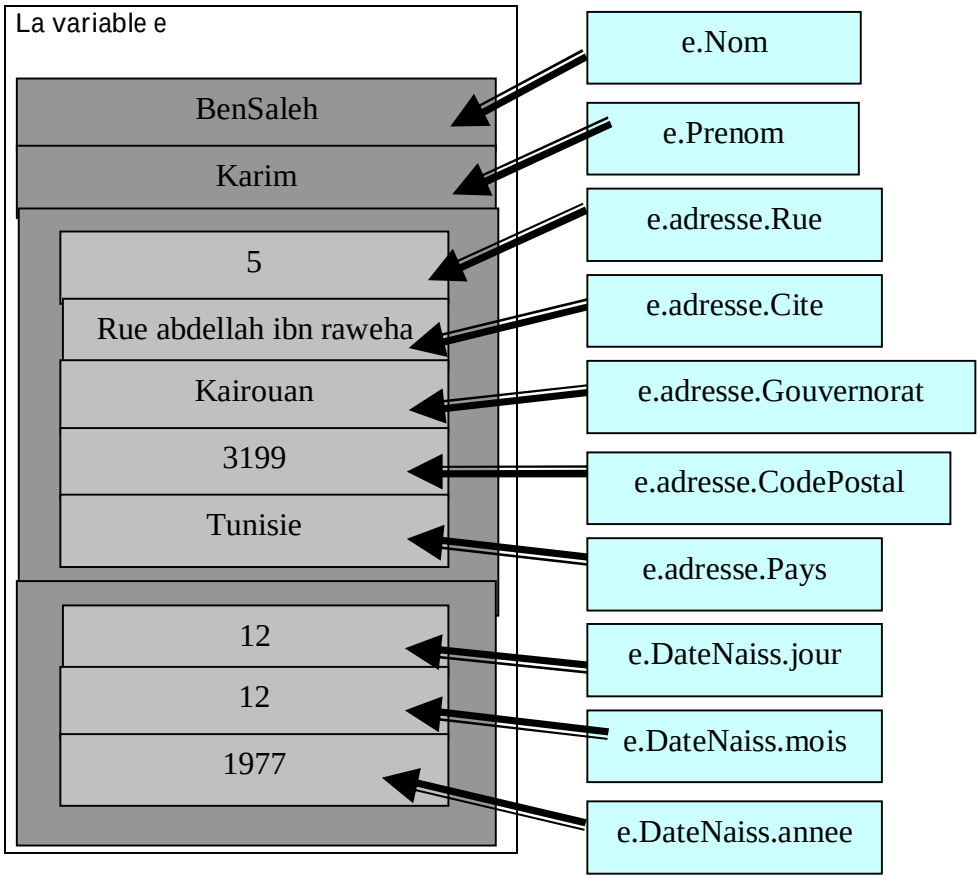
Exercice:

Ecrire un algorithme permettant de créer une variable de type Etudiant et de l'affecter les informations suivantes: (Bensaleh, Karim, (5, rue abdellah ibn raweha, kairouan, 3199, Tunisie), (12,12,1977))

ALGORITHME Essai2
DEBUT
Adr: structure
    Rue: entier
    Cite chaine (20)
    Gouvernorat: chaine (50)
    CodePostal: entier
    Pays: chaine (50)

```
Fin structure
Date: strucure
        Jour: entier
        Mois: entier
        Annee: entier
Fin structure
Etudiant: structure
        Nom: chaine (50)
        Prenom: chaine (25)
        Adresse: Adr
        DateNaiss: Date
Fin structure
E: Etudiant
Copier_Chaine (e.Nom, "Bensaleh")
Copier_Chaine (e.Prenom, "Karim")
e.adresse.Rue ß 5
Copier_Chaine (e.adresse.Cite, " rue abdellah ibn raweha")
Copier_Chaine (e.adresse.Gouvernorat, " Kairouan")
e.adresse.CodePotal ß 3199
Copier_Chaine (e.adresse.Pays, " Tunisie")
e.DateNaiss.jour ß 12
e.DateNaiss.mois ß 12
e.DateNaisse.annee ß 1977
Fin
```

Dans la mémoire centrale, la variable e de structure Etudiant sera représentée comme suit:



IV. TABLEAUX DE STRUCTURES

La syntaxe de définition d'un tableau de structures est la suivante:

<Nom_Tableau>: tableau [1..<Nombre_Elements>] de <Nom_structure>

Exemple

z Définir un tableau de 5 étudiants en se référant à la structure Etudiant déjà définie.

SOLUTION

ListeEtudiants: tableau [1..5] de Etudiant

La représentation, dans la mémoire centrale, de ce tableau est:

1	2	3	4	5
Nom	Nom	Nom	Nom	Nom
Prenom	Prenom	Prenom	Prenom	Prenom
Adresse Rue Cite Gouvernorat CodePostal Pays	Adresse Rue Cite Gouvernorat CodePostal Pays	Adresse Rue Cite Gouvernorat CodePostal Pays	Adresse Rue Cite Gouvernorat CodePostal Pays	Adresse Rue Cite Gouvernorat CodePostal Pays
DateNaiss Jour Mois Annee	DateNaiss Jour Mois Annee	DateNaiss Jour Mois Annee	DateNaiss Jour Mois Annee	DateNaiss Jour Mois Annee

z Accéder aux différents champs de l'étudiant existant dans la case 2 du tableau ListeEtudiants

ListeEtudiants[1].Nom ØØ permet d'accéder au nom de l'étudiant n° 1;
ListeEtudiants[1].Prenom ØØ permet d'accéder au prénom de l'étudiant n° 1;
ListeEtudiants[1].Adresse.Rue ØØ permet d'accéder à la rue de l'adresse de l'étudiant n° 1;
ListeEtudiants[1].Adresse.CiteØØ permet d'accéder à la cité de l'adresse de l'étudiant n° 1;
ListeEtudiants[1].Adresse.Gouvernorat ØØ permet d'accéder à la gouvernorat de l'adresse de l'étudiant n° 1;
ListeEtudiants[1].Adresse.CodePostal ØØ permet d'accéder au code postal de l'adresse de l'étudiant n° 1;
ListeEtudiants[1].Adresse.Pays ØØ permet d'accéder au pays de l'adresse de l'étudiant n° 1;
ListeEtudiants[1].DateNaiss.Jour ØØ permet d'accéder au jour de la date de naissance de l'étudiant n° 1;
ListeEtudiants[1].DateNaiss.Mois ØØ permet d'accéder au mois de la date de naissance de l'étudiant n° 1;
ListeEtudiants[1].DateNaiss.Annee ØØ permet d'accéder à l'année de la date de naissance de l'étudiant n° 1.

V. APPLICATION

Soit un tableau de structure produit. Le nombre de produits maximum est 100. Un produit est défini par sa référence, son libellé, sa couleur, son prix et sa quantité en stock.
Ecrire un algorithme qui permet de:

- demander de l'utilisateur de lire un nombre n<=100
- remplir le tableau avec n produits dont les caractéristiques sont entrées par l'utilisateur.
- Calculer et afficher la quantité moyenne stockée
- Calculer et afficher la référence du produit qui a le plus grand prix

BIBLIOGRAPHIE

1 Kyle LOUDON, traduction d'Eric JACOBONI, ALGORITHMES EN C, O'Reilly & Associates, Paris 2000,2001, 2002
ISBN: 2-84177-096-6

1 ClaudeDELANNOY, EXERCICES EN LANGAGE C, Eyrolles, Quatrième tirage 2000
ISBN: 2-212-08984-8

1 Peter AITKEN & Bradley L. JONES, LE LANGAGE C, Campus Press, France 2000
ISBN: 2-7440-0838-9

1 Claude DELANNOY, PROGRAMMER EN LANGAGE C, Eyrolles 1997
ISBN: 2-212-11072-3

1 Michael GRIFFITHS, ALGORITHMIQUE ET PROGRAMMATION, Hernes, 1992
ISBN: 2-86601-323-9

1 Jacques COURTIN & Irène KOWARSKI, INITIATION A L'ALGORITHMIQUE ET AUX STRUCTURES DE DONNEES, Dunod, 1994
ISBN: 2-10-004039-1

1 Guy PIERRA, LES BASES DE LA PROGRAMMATION ET DU GENIE LOGICIEL, Dunod, 1991
ISBN: 2-04-020722-8

1 Claude DELANNOY, LANGAGE C, Eyrolles, 1999, 2002
ISBN: 2-212-11123-1

1 Pc Poche, LANGAGE C, Micro Application, 2000, 20001
ISBN: 2-7429-2008-0