Chapitre 01 : Les éléments de base d'un algorithme

I. Introduction

I.1. Notion d'algorithme

Dans la vie courante, un algorithme peut prendre la forme :

- d'une recette de cuisine;
- d'un itinéraire routier;
- d'un mode d'emploi, etc.

Une recette de cuisine, par exemple, est un algorithme : à partir des ingrédients, elle explique comment parvenir au plat. De même, un itinéraire routier explique comment, à partir d'une position initiale, rejoindre une position finale en un certain nombre d'étapes … .

Exemple : Préparer la pâte à tarte

➢ **Les ingrédients :**

250 g de farine ;
50 g de beurre ;
1 verre de lait.

➢ **Les actions élémentaires à réaliser**

Début
  Incorporer le beurre dans la farine
  Pétrir le mélange jusqu'à ce qu'il soit homogène
  Ajouter du lait
  Mélanger
  Si la pâte est trop sèche, alors ajouter du lait
  Si la pâte à une bonne consistance, alors la laisser reposer une demi heure
  Faire cuire la pâte
Fin

Un algorithme sert à transmettre un savoir faire. Il décrit les étapes à suivre pour réaliser un travail. Tout comme le savoir-faire du cuisinier se transmet sous la forme d'une recette, celui d'un informaticien se transmet sous la forme d'un algorithme.

I.2. Définition d'un algorithme

Le mot « algorithme » provient de la forme latine (Algorismus) du nom du mathématicien arabe AL KHWARIZMI. Ce dernier formula une première définition : « Un algorithme est une séquence d'opérations visant à la résolution d'un problème en un temps fini.»
Nous pouvons adopter la définition suivante : un algorithme est la description de la méthode de résolution d'un problème quelconque en utilisant des instructions élémentaires. Ces instructions deviennent compréhensibles par l'ordinateur lors de la traduction de l'algorithme en un programme.

I.3. Algorithmique et programmation

Tout problème à programmer doit être résolu, d'abord sous forme d'algorithme, puis converti en programme dans le langage de votre choix. En effet, un algorithme est indépendant du langage de programmation utilisé.
Un programme est un enchaînement d'instructions, écrit dans un langage de programmation, exécutées par un ordinateur, permettant de

traiter un problème et de renvoyer des résultats. Il représente la traduction d'un algorithme à l'aide d'un langage de programmation.
Le cycle de développement d'un programme (ou d'une application) informatique peut se résumer ainsi (figure1) :

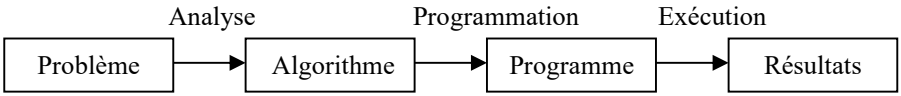


Figure 1 : Cycle de développement d'un programme

Exemple :

Parmi les langages de programmations, on peut citer : Pascal, C, C++, Visual Basic, Java, C#, J#, etc.

II. Structure générale d'un algorithme

Un algorithme est composé de trois parties principales (figure 2) :

- **l'en-tête** : cette partie sert à donner un nom à l'algorithme. Elle est précédée par le mot ***Algorithme*** ;
- **la partie déclarative** : dans cette partie, on déclare les différents objets que l'algorithme utilise (constantes, variables, etc.) ;
- **le corps de l'algorithme** : cette partie contient les instructions de l'algorithme. Elle est délimitée par les mots ***Début*** et ***Fin***.

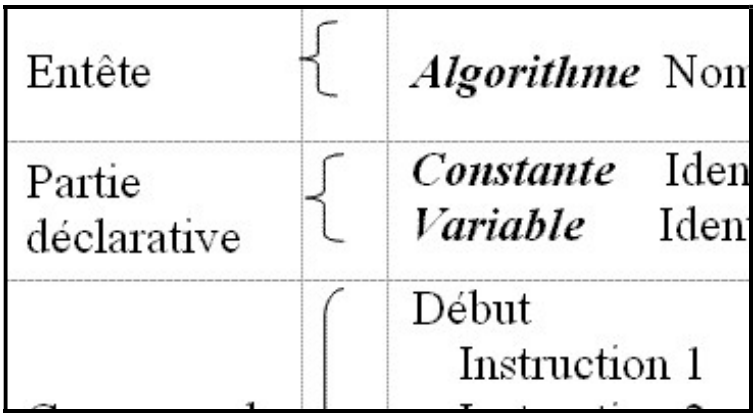


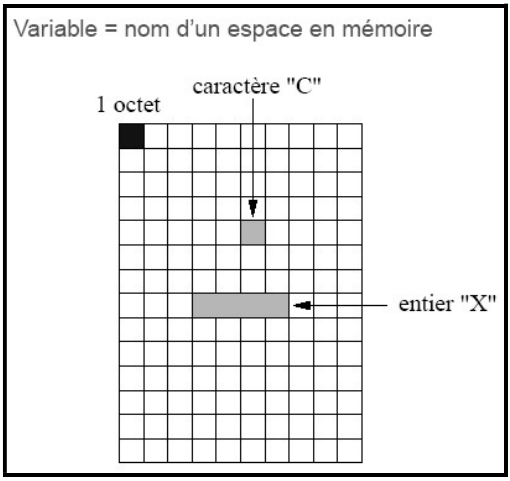
Figure 2 : Structure d'un algorithme

III. Les variables et les constantes

III.1. Notion de variable

Les données ainsi que les résultats des calculs intermédiaires ou finaux, sont rangés dans des ***cases mémoires*** qui correspondent à des ***variables***.

Ainsi, une variable (figure suivante) est rangée dans un emplacement mémoire nommé, de taille fixe (ou non) prenant au cours du déroulement de l'algorithme, un nombre indéfini de valeurs différentes.



III.2. Déclaration des variables

La partie déclaration consiste à énumérer toutes les variables dont on aura besoin au cours de l'algorithme.

Chaque déclaration doit comporter le nom de la variable (identificateur) et son type.

Syntaxe :

Variable identificateur : type

Exemples :

Variable surface : réel
Variable a : entier
Variable a, b, c, d : entiers
Variable Nom_Prenom : chaîne
Variable absent : logique

➢ Identificateur

Un identificateur est le nom donné à une variable, une fonction, etc. Ce nom doit obligatoirement **commencer** par **une lettre** suivie d'une suite de lettres et de chiffres et il **ne doit pas contenir d'espace**.

➢ Types de données

Le type d'une variable est l'ensemble des valeurs qu'elle peut prendre. Par exemple, une variable de type logique (booléen) peut prendre les valeurs ***Vrai*** ou ***Faux***.

Les différents types utilisés en algorithmique :

- **Type Entier** sert à manipuler les nombres entiers positifs ou négatifs. Par exemple : 5, -15, etc.
- **Type Réel** quant à lui sert à manipuler les nombres à virgule. Par exemple : 3.14, -15.5, etc.
- **Type Caractère** permet de manipuler des caractères alphabétiques et numériques. Par exemple : 'a', 'A', 'z', ' ?', '1', '2', etc.
- **Type Chaîne** sert à manipuler des chaînes de caractères permettant de représenter des mots ou des phrases. Par exemple : "bonjour", "Monsieur", etc.
- **Type Logique (Booléen) :** utilise les expressions logiques. Il n'y a que deux valeurs booléennes : ***Vrai*** et ***Faux***.

Exemple :

```
Variables n : entier
          r : réel
          a, b : logiques
          nom_etudiant : chaîne
```

A un type donné, correspond un ensemble d'opérations définies pour ce type :

Type	**Opérations possibles**	**Symbole ou mot clé correspondant**

Entier	Addition	+
	Soustraction	-
	Multiplication	*
	Division	/
	Division entière	(DIV en VB et % en C)
	Modulo (le reste de la division entière)	(MOD en VB)
	x exposant y	^ : en vb et pow(x,y) en C
	Comparaisons	<, =, >, <=, >=, ≠
En algorithmique nous symbolisons la division entière par DIV et le reste de la division entière par MOD		
Réel	Addition	+
	Soustraction	-
	Multiplication	*
	Division	/
	Exposant	^
	Comparaisons	<, =, >, <=, >=, ≠
Caractère	Comparaisons	<, =, >, <=, >=, ≠
Chaîne	Concaténation	(+ , & : en VB)
	Comparaison	<, =, >, <=, >=, ≠
Booléen	Logiques	ET, OU, NON et OU_{ex}

Exemple :

5 / 2 = 2.5
5 Div 2 = 2
5 Mod 2 = 1
5 ^ 2 = 25

"Bonjour" & " " & "Monsieur" donne "Bonjour Monsieur"
L'expression 5 > 2 est Vraie.
L'expression 7 < 4 est fausse.

Les opérations définies pour le type booléen sont :

- Le **ET** logique (and)
- Le **OU** logique (Or)
- Le **NON** logique (not)
- Le **OU_{ex} (**Ou exclusif appelée en VB Xor)

Nous résumons dans une table de vérité les résultats obtenus suivant les valeurs de deux opérandes :

P	Q	**Non** P	**Non** Q	P **et** Q	P **ou** Q	P **ou$_{ex}$** Q
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	1	0

Lois de De Morgan

Non (P et Q) ⇔ (non P) ou (non Q)
Non (P ou Q) ⇔ (non P) et (non Q)

III.3. Les constantes

Comme une variable, à une constante correspond un emplacement mémoire réservé auquel on accède par le nom qui lui a été attribué, mais dont la valeur stockée ne sera jamais modifiée au cours du programme.

Syntaxe :

Constante NOM_DE_LA_CONSTANTE = valeur

Exemple :

Constante PI = 3.14

IV. Les instructions de base

Une instruction est une action élémentaire commandant à la machine un calcul, ou une communication avec l'un de ses périphériques d'entrées ou de sorties. Les instructions de base sont :

IV.1. L'instruction d'affectation

L'affectation permet d'affecter une valeur à une variable. Elle est symbolisée en algorithmique par "←".
Le signe "←" précise le sens de l'affectation.

Syntaxe :

Variable ← Expression

Expression peut être soit :

- identificateur ;
- constante ;
- expression arithmétique ;
- expression logique.

Sémantique :

Une affectation peut être définie en deux étapes :

- évaluation de l'expression qui se trouve dans la partie droite de l'affectation ;
- placement de cette valeur dans la variable.

Exemple :

```
0  Algorithme      Calcul
1  Variables   A, B, C, D : entier
2  Début
3        A ← 10
4        B ← 30
5        C ← A+B
6        D ← C*A
7  Fin
```

Note : Les lignes sont numérotées pour faciliter l'explication.

Nous pouvons expliquer ce qui ce passe par le tableau suivant :

Variable	**N° de ligne**					
	1	**2**	**3**	**4**	**5**	**6**
A	?	?	10	10	10	10
B	?	?	?	30	30	30
C	?	?	?	?	40	40
D	?	?	?	?	?	400

Remarque :

Les variables numériques ne sont pas forcément initialisées à zéro.

Leur valeur peut être n'importe quoi. C'est la raison de la présence du point d'interrogation avant qu'une première valeur ne soit affectée à la variable.

Exemple :

```
0  Algorithme     Logique
1  Variables   A, B, C : Booléen
2  Début
3     A← Vrai
4     B←Faux
5     C←A et B
6  Fin
```

Variable	N° de ligne				
	1	2	3	4	5
A	?	?	Vrai	Vrai	Vrai
B	?	?	?	Faux	Faux
C	?	?	?	?	Faux

IV.2. L'instruction d'entrée

L'instruction d'entrée ou de lecture donne la main à l'utilisateur pour saisir une donnée au clavier. La valeur saisie sera affectée à une variable.

Syntaxe :

Lire (identificateur)

Exemples :

Lire(A)
Lire(A, B, C)

L'instruction ***Lire(A)*** permet à l'utilisateur de saisir une valeur au clavier. Cette valeur sera affectée à la variable A.

Remarque :

Lorsque le programme rencontre cette instruction, l'exécution

s'interrompt et attend que l'utilisateur tape une valeur. Cette valeur est rangée en mémoire dans la variable désignée.

IV.3. L'instruction de sortie

Avant de lire une variable, il est conseillé d'écrire des libellés à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper (sinon, l'utilisateur passe son temps à se demander ce que l'ordinateur attend de lui).
L'instruction de sortie (d'écriture) permet d'afficher des informations à l'écran.

Syntaxe :

Ecrire (expression)

Expression peut être une valeur, un résultat, un message, le contenu d'une variable, etc.

Exemple 1 :

Ecrire (A)

Cette instruction permet d'afficher à l'écran la valeur de la variable A.

Exemple 2 :

A ← 2
Ecrire ("La valeur de A est = ", A)

La dernière instruction affiche à l'écran :
La valeur de A est = 2

Exercice : Calcul PTTC

Ecrire un algorithme qui permet de saisir le prix HT (PHT) d'un article et de calculer son prix total TTC (PTTC). TVA = 20%.

Solution :

```
0   Algorithme     Calcul_PTTC
1   Variables   PHT, PTTC : réel
2   Constante TVA = 0.2
3   Début
4      Ecrire ("Entrez le prix hors taxes : ")
5      Lire (PHT)
6      PTTC ← PHT + (PHT * TVA)
7      Ecrire ("Le prix TTC est ", PTTC)
8   Fin
```

Explication de l'algorithme :

N° de ligne	Explication
0	Déclare un algorithme dont le nom est Calcul_PTTC
1	Déclare les différentes variables utilisées par l'algorithme
2	Déclare la constante TVA
3	Marque le début des traitements effectués par l'algorithme.
4	Affiche à l'écran le message : Entrez le prix hors taxes :
5	Permet à l'utilisateur de saisir une valeur au clavier qui sera affectée à la variable PHT.
6	Calcul le prix TTC et affecte le résultat à la variable PTTC
7	Affiche le résultat à l'écran
8	Marque la fin de l'algorithme

V. Les commentaires

Lorsqu'un algorithme devient long, il est conseillé d'ajouter des lignes de commentaires dans l'algorithme, c'est-à-dire des lignes qui ont pour but de donner des indications sur les instructions effectuées et d'expliquer le fonctionnement du programme (algorithme) sans que le compilateur ne les prenne en compte.
Dans la suite de ce livre, nous utiliserons trois types de commentaires :

```
// Commentaire sur une ligne
/* Commentaire */
' Commentaire
/* Commentaire
sur
plusieurs
lignes */
```

Remarque :

Parfois on utilise les commentaires pour annuler l'action de quelques instructions dans un algorithme ou un programme au lieu de les effacer. Voir l'exemple suivant :

```
Variable  i : entier
// Variable i : réel
i ← 4
/* i ← 4
i ← 4 * I */
```

Les instructions précédentes sont équivalentes à :

```
Variable  i : entier
i ← 4
```

Note :

Les annexes 01 et 02 de ce livre traduisent les instructions algorithmiques vues dans ce chapitre et les chapitres qui suivent en langage de programmation Visual Basic (VB) et en langage C.

Chapitre 02 : Les structures alternatives et répétitives

I. Les structures alternatives

I.1. Introduction

Contrairement au traitement séquentiel, La structure alternative ou conditionnelle permet d'exécuter ou non une série d'instructions selon la valeur d'une condition.

I.2. La structure Si ... Alors ... Sinon ... FinSi

Syntaxe :

Si condition **Alors**
    Instruction(s)1
Sinon
    Instruction(s)2
FinSi

Une condition est une expression logique ou une variable logique évaluée à ***Vrai*** ou ***Faux.***

La condition est évaluée. Si elle est vraie, la série d'instructions1 est

exécutée et l'ensemble d'instructions2 est ignoré, la machine sautera directement à la première instruction située après le ***FinSi***.

De même, au cas où la condition était fausse (avait comme valeur Faux), la machine saute directement à la première ligne située après le ***Sinon*** et exécute l'ensemble d'instructions2.

Exercice :

Ecrire un algorithme qui affiche si un nombre entier saisi au clavier est pair ou impair.

Solution :

On dit qu'un nombre entier n est pair si le reste r de la division entière de n par 2 est égale à 0. Sinon il est impair.

```
Algorithme Parité
// déclaration des variables
Variables n, r : entiers
Début
   Ecrire ("Entrez la valeur de n : ")
   Lire (n)
   // r est le reste de la division entière de n par 2
   r ← n Mod 2
   Si r = 1 Alors
      Ecrire (n, " est impair")
   Sinon
      Ecrire (n, " est pair ")
   FinSi
Fin
```

Remarque : Présentation de l'algorithme ou du programme

Certaines parties de l'algorithme ou du code sont en retrait par rapport à d'autres, c'est ce qu'on appelle ***l'indentation***. Celle-ci est très importante pour la lisibilité de l'algorithme ou du programme. Elle montre rapidement le début et la fin de chaque instruction ***alternative*** ou ***répétitive*** ainsi le début et la fin de l'algorithme ou du programme.

De plus, pour faciliter la compréhension, toutes vos instructions

doivent être commentées. Un développeur est souvent amené à modifier un code, par conséquent, des commentaires de qualité rendent cette tâche plus facile et plus rapide.

Note 1 :

Les deux blocs 1 et 2 d'instruction sont équivalents.

Bloc 1		**Bloc 2**

Si condition Alors
  Instructions(1)
Sinon
  Instructions(2)
FinSI
	Le bloc 1 est équivalent au bloc 2	Si ***Non*** (condition) Alors
		  Instructions(2)
		Sinon
		  Instructions(1)
		FinSI

Exemple :

Bloc 1		**Bloc 2**

Si note > = 10 Alors
  Ecrire("Note acceptable")
Sinon
  Ecrire("Mauvaise Note")
FinSI
	Le bloc 1 est équivalent au bloc 2	Si note < 10 Alors
		  Ecrire("Mauvaise Note")
		Sinon
		  Ecrire("Note acceptable")
		FinSI

Note 2 : Conditions composées

Certains problèmes exigent de formuler des conditions qui ne peuvent pas être exprimées sous la forme simple exposée ci-dessus.
Par exemple : la condition "x est inclus dans l'intervalle]10, 20 [" est composée de deux conditions simples qui sont : "x est supérieur à 10" et "x est inférieur à 20" reliées par l'opérateur logique ***Et***.

Exemple :

Ecrire un algorithme qui teste si une note saisie au clavier est comprise entre 0 et 20.

Solution :

```
Algorithme TestNote
Variables Note : réel
          Message : Chaîne
Début
  Ecrire ("Entrer la note : ")
  Lire (Note)
  Si Note >= 0 Et Note <= 20 Alors
    Message ← "La note " & Note & " est correcte"
  Sinon
    Message ← "La note " & Note & " est incorrecte"
  Fin Si
  Écrire (Message)
Fin
```

Exercice :

Ecrire un algorithme qui demande deux nombres m et n à l'utilisateur et l'informe ensuite si le produit est négatif ou positif. On inclut dans l'algorithme le cas où le produit peut être nul.

Solution :

```
Algorithme Signe_Produit
Variables m, n : entiers
Début
  Ecrire(Entrez deux nombres m et n : )"
  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
```

Remarque :

- Dans une condition composée employant à la fois des opérateurs ***Et***

et des opérateurs ***Ou***, la présence de parenthèses possède une influence sur le résultat, tout comme dans le cas d'une expression numérique comportant des opérateurs numériques (exemple : des multiplications et des additions).

- Toute structure de test requérant une condition composée faisant intervenir l'opérateur ***Et*** peut être exprimée de manière équivalente avec un opérateur ***Ou***, et réciproquement (Lois de Morgan).

I.3. La structure Si ... Alors ... FinSi

Cette structure est utilisée si on veut exécuter une instruction seulement si une condition est vraie et ne rien faire si la condition est fausse.

Syntaxe :

Si condition **Alors**
  Instruction(s)
FinSi

Exemple :

Une grande surface accorde à ses clients, une réduction de 2% pour les montants d'achat supérieurs à 1500,00dh.

Ecrire un algorithme permettant de saisir le prix total HT (PTHT) et de calculer le montant TTC (PTTC) en prenant en compte la remise et la TVA=20%.

Solution :

```
Algorithme     Calcul_PTTC
Constante TVA = 0.2
Variables   PTHT, PTTC : réel
Début
   Ecrire ("Entrez le prix total hors taxe : ")
   Lire (PTHT)
   Si PTHT > 1500 Alors
      PTHT ← PTHT * 0.98
   FinSi
   PTTC ← PTHT + (PTHT * TVA)
   Ecrire ("Le prix TTC est ", PTTC)
Fin
```

I.4. La Structure SI ... Alors ... SinonSI ... Sinon ... FinSi

Syntaxe :

Si condition1 **alors**
    Instructions1
SinonSi condition2 **alors**
    Instructions2
SinonSi condition3 **alors**
    Instructions3
    …..
Sinon
    Instructions
FinSi

L'instruction qui sera exécutée est l'instruction dont la condition est vraie. Si aucune condition n'a la valeur vraie, c'est l'instruction qui suit le ***Sinon*** qui sera exécutée.

Exemple :

Ecrire un algorithme qui permet d'afficher le maximum parmi deux nombres saisis au clavier.

Solution :

```
Algorithme Max
Variables A, B : réel
Début
   Ecrire ("Entrez la valeur de A : ")
   Lire (A)
   Ecrire ("Entrez la valeur de B : ")
   Lire (B)
   Si A > B Alors
      Ecrire ("Le maximum est ", A)
   SinonSI A = B alors
      Ecrire ("égalité")
   Sinon
      Ecrire ("Le maximum est ", B)
   FinSi
Fin
```

Remarque :

Une structure conditionnelle peut contenir à son tour une autre structure conditionnelle, c'est-à-dire l'une incluse dans l'autre. Dans ce cas, les instructions à exécuter après **Alors** et/ou **Sinon** sont, à leur tour, des instructions **Si...Alors...Sinon**.

Exemple :

Ecrire un algorithme permettant de calculer le montant des allocations familiales sachant que le montant dépend du nombre d'enfants :
- Si nombre d'enfants est inférieur ou égale à trois alors les allocations sont de 150 dh par enfant.
- Si le nombre d'enfants est strictement supérieur à trois et inférieur ou égale à six alors les allocations sont de :
    - 150 dh par enfant pour les trois premiers enfants
    - 38 dh par enfant pour les suivants
- Si le nombre d'enfants est strictement supérieur à six alors les allocations sont de :
    - 150dh par enfant pour les trois premiers enfants
    - 38 dh par enfant pour les 3 suivants

- 0 dh par enfant pour les suivants.

Solution :

```
Algorithme      Allocation_familiale
Variables   NE, M : entier
/* NE : le nombre d'enfants
M : montant des allocations familiales */
Début
   Ecrire ("Entrez le nombre d'enfants : ")
   Lire (NE)
   Si NE <= 3 Alors
      M ← NE * 150
   Sinon
      Si NE <=6 alors
         M ← 450 + (NE – 3) * 38
      Sinon
         M ← 564
      FinSi
   FinSi
   Ecrire ("Le montant d'allocation familiale est ", M)
Fin
```

I.5. Structure à choix multiples

Cette structure conditionnelle permet de choisir le traitement à effectuer en fonction de la valeur ou de l'intervalle de valeurs d'une variable ou d'une expression.

Syntaxe :

```
Selon sélecteur faire
   Valeur1 : action(s)1
   Valeur2 : action(s)2
   Valeur3 : action(s)3
   ....
   Valeurn : action(s)n
Sinon
action(s)
```

FinSelon

Lorsque l'ordinateur rencontre cette instruction, il vérifie la valeur de la variable de sélection (sélecteur) et il la compare aux différentes valeurs.

Les valeurs sont évaluées dans l'ordre, les unes après les autres, et dès qu'une de celle-ci est vérifiée l'action associée est exécutée. On peut utiliser une instruction **Sinon** (facultative), dont l'action sera exécutée si aucune des valeurs évaluées n'a été remplie.

Exemple :

Ecrire un algorithme permettant d'afficher le mois en toute lettre selon son numéro saisi au clavier.

Solution :

```
Algorithme Mois
Variables   N : Entier
Début
   Ecrire ("Donner le numéro du mois: ")
   Lire (N)
   Selon N faire
      1 : Ecrire ("Janvier")
      2 : Ecrire ("Février")
      3 : Ecrire ("Mars")
      4 : Ecrire ("Avril")
      5 : Ecrire ("Mai ")
      6 : Ecrire ("Juin")
      7 : Ecrire ("Juillet")
      8 : Ecrire ("Août")
      9 : Ecrire ("Septembre")
      10 : Ecrire ("Octobre")
      11 : Ecrire ("Novembre")
      12 : Ecrire ("Décembre")
   Sinon
      Ecrire ("Le numéro saisi est incorrecte: ")
   FinSelon
Fin
```

II. Les structures répétitives

II.1. Introduction

Prenons le cas d'une saisie au clavier, par exemple, on pose une question à laquelle l'utilisateur doit répondre par O (Oui) ou N (Non).

L'utilisateur risque de taper autre chose (une autre lettre), le programme peut soit planter par une erreur d'exécution soit se dérouler normalement jusqu'au bout, mais en produisant des résultats fantaisistes.

Pour remédier à ce problème, on peut mettre en place un contrôle de saisie pour vérifier que les données entrées au clavier correspondent

bien à celles attendues par l'algorithme.

On pourrait essayer avec un SI.

```
Algorithme  contrôle_de_saisie
Variable   Rep : caractère
Début
   Ecrire "Voulez vous une copie de ce cours ? (O/N)"
   Lire (Rep)
   Si Rep ≠ 'O' ET Rep ≠ 'N' Alors
      Ecrire ("Erreur de saisie. Recommencez")
      Lire (Rep)
   FinSi
Fin
```

L'algorithme ci-dessus résout le problème si on se trompe qu'une seule fois, et on fait entrer une valeur correcte à la deuxième demande. Sinon en cas de deuxième erreur, il faudrait rajouter un SI. Et ainsi de suite, on peut rajouter des centaines de SI.

La solution à ce problème consiste à utiliser une structure répétitive.

Une structure répétitive, encore appelée boucle, est utilisée quand une instruction ou une liste d'instructions, doit être répétée plusieurs fois. La répétition est soumise à une condition.

II.2. La boucle TantQue ... Faire

La boucle TantQue permet de répéter un traitement tant que la condition est vraie.

Syntaxe :

TantQue condition **faire**
    Instruction(s)
FinTantQue

L'exécution de la boucle dépend de la valeur de la condition. Si elle est vraie, le programme exécute les instructions qui suivent, jusqu'à ce qu'il rencontre la ligne FinTantQue. Il retourne ensuite sur la ligne du

TantQue, procède au même examen, et ainsi de suite.

La boucle ne s'arrête que lorsque la condition prend la valeur fausse, et dans ce cas le programme poursuit son exécution après FinTantQue.

Exemple :

```
Algorithme  contrôle_de_saisie
Variable   Rep : caractère
Début
   Ecrire ("Voulez vous une copie de ce cours ? (O/N)" )
   Lire (Rep)
   TantQue Rep ≠ 'O' ET Rep ≠ 'N' faire
      Ecrire (" Erreur de saisie")
      Ecrire ("Voulez vous une copie de ce cours ? (O/N)" )
      Lire (Rep)
   FinTantQue
Fin
```

Remarques :

- Etant donnée que la condition est évaluée avant la mise en oeuvre des instructions, ***ce qui est une sécurité***, il est possible que celles-ci ne soient jamais exécutées.
- Si une structure TantQue dans laquelle la condition ne devient jamais fausse. Le programme tourne dans une boucle infinie et n'en sort plus.

Exemple : Boucle infinie

Dans l'exemple ci-dessous nous avons une boucle infinie, l'ordinateur ne s'arrêtera jamais d'afficher le message **Bonjour** car la variable **I** qui est testée dans la condition n'est jamais incrémentée :

```
I ← 1
Tant que I < = 10 Faire
   Ecrire(" bonjour")
FinTantQue
```

Exercice :

Ecrire un algorithme qui calcule S = 1 + 2 + 3 + 4 + 5 + ... + N.

Solution :

```
Algorithme  Somme
Variables   S, I, N : Entier
Début
   Ecrire ("Entrer la valeur de N : ")
   Lire (N)
   S ← 0
   I ← 1
   TantQue  I <= N faire
      S ← S + I
      I  ← I +1
   FinTantQue
   Ecrire("La somme des ", N, " premiers entiers est : ",  S)
Fin
```

II.3. La boucle Pour ... Faire

La boucle **Pour … Faire** permet de répéter une liste d'instructions un nombre connu de fois.

Syntaxe :

Pour compteur ← valeur_initiale **jusqu'à** valeur_finale **Faire**
    Instruction(s)
FinPour

La variable compteur est de type entier. Elle est initialisée à la valeur initiale. Le compteur augmente sa valeur de **un** (1) **automatiquement** à chaque tour de boucle jusqu'à la valeur finale.

Pour chaque valeur prise par la variable compteur, la liste des instructions est exécutée.

Lorsque la variable compteur vaut la valeur finale, le traitement est exécuté une dernière fois puis le programme sort de la boucle.

Exemple :

Les instructions suivantes :

```
Pour k ← 0 jusqu'à 10 faire
   Ecrire( " 7 * ", k, " = ", 7*k)
   Ecrire("\")
FinPour
```

affichent à l'écran la table de multiplication de 7.

7 * 0 = 0	7 * 4 = 28	7 * 8 = 56
7 * 1 = 7	7 * 5 = 35	7 * 9 = 63
7 * 2 = 14	7 * 6 = 42	7 * 10 = 70
7 * 3 = 21	7 * 7 = 49	

Exercice :

Ecrire un algorithme permettant de calculer la somme des dix premiers nombres entiers.

Solution :

```
Algorithme  Somme
Variables   S, I : Entier
Début
   S ← 0
   Pour I ← 1 jusqu'à 10 Faire
      S ← S + I
   FinPour
   Ecrire("La somme des dix premiers entiers est : ", S)
Fin
```

Remarques :

- Par défaut la variable compteur est incrémentée de 1 à chaque tour de boucle
- Pour modifier la valeur d'incrémentation, il suffit de rajouter le mot **Pas** (en anglais **step**) et la valeur de ce pas à la boucle **Pour**.

La syntaxe générale de la structure Pour est :

Pour compteur ← VI **jusqu'à** VF **Pas** ValeurPas **Faire**
    Instruction(s)
FinPour

VI c'est la valeur initiale
VF c'est la valeur finale.

Exercice :

Ecrire un algorithme qui permet de saisir un nombre entier et qui calcule la somme des entiers pairs jusqu'à ce nombre. Par exemple, si l'on saisi 10, le programme doit calculer :
0 + 2 + 4 + 6 + 8 + 10 = 30

Solution :

```
Algorithme  Somme
Variables   S, I, N : Entier
Début
   Ecrire ("Entrer la valeur de N : ")
   Lire (N)
   S ← 0
   Pour I ← 0 jusqu'à N Pas 2 Faire
      S ← S + I
   FinPour
   Ecrire("La somme des nombres pairs est : ", S)
Fin
```

II.4. La boucle Répéter … Jusqu'à

Cette boucle sert à répéter une instruction jusqu'à ce qu'une condition soit vraie.

Remarque :

Cette boucle ne s'utilise en général que pour des menus, elle est dangereuse car il n'y a pas de vérification de la condition avant d'y entrer !

Syntaxe :

Répéter

Instruction(s)
Jusqu'à condition

La liste d'instructions est exécutée, puis la condition est évaluée. Si elle est fausse, le corps de la boucle est exécuté à nouveau puis la condition est réévaluée et si elle a la valeur vrai, le programme sort de la boucle et exécute l'instruction qui suit **Jusqu'à**.

Exemple :

En utilisant la boucle Répéter … Jusqu'à, écrire un algorithme qui calcule la somme des N premiers nombres entiers. On suppose que N est strictement positif.

Par exemple, si N = 6, le programme doit calculer : 1+2+3+4+5+6 =21

Solution :

```
Algorithme  Somme
Variables   S, I, N : Entier
Début
   Ecrire ("Entrer une valeur strictement positif : ")
   Lire (N)
   S ← 0
   I ← 1
   Répéter
      S ← S + I
      I ← I +1
   Jusqu'à  I >= N
   Ecrire("La somme des ",N," premiers entiers est : ", S)
Fin
```

Remarques :

- Les boucles « Répéter » et « TantQue » sont utilisées lorsqu'on ne sait pas au départ combien de fois il faudra exécuter ces boucles.
- A la différence de la boucle « TantQue », la boucle « Répéter » est exécutée au moins une fois.
- La condition d'arrêt de la boucle « Répéter » est la négation de

la condition de poursuite de la boucle « TantQue ».

- On utilise la boucle « Pour » quand l'on connaît le nombre d'itérations à l'avance.

Chapitre 03 : Les Tableaux

I. Introduction

Imaginons que dans un programme, nous avons besoin d'un grand nombre de variables, il devient difficile de donner un nom à chaque variable.

Exemple :

Ecrire un algorithme permettant de saisir cinq notes et de les afficher après avoir multiplié toutes les notes par trois.

Solution :

```
Algorithme  Note
Variables   N1, N2, N3, N4, N5 : Réel
Début
   Ecrire ("Entrer la valeur de la 1er note")
   Lire (N1)
   Ecrire ("Entrer la valeur de la 2ème note")
   Lire (N2)
   Ecrire ("Entrer la valeur de la 3ème note")
   Lire (N3)
   Ecrire ("Entrer la valeur de la 4ème note")
   Lire (N4)
```

```
   Ecrire ("Entrer la valeur de la 5ème note")
   Lire (N5)
   Ecrire ("La note 1 multipliée par 3 est : ", N1 * 3)
   Ecrire ("La note 2 multipliée par 3 est : ", N2 * 3)
   Ecrire ("La note 3 multipliée par 3 est : ", N3* 3)
   Ecrire ("La note 4 multipliée par 3 est : ", N4 * 3)
   Ecrire ("La note 5 multipliée par 3 est : ", N5 * 3)
Fin
```

La même instruction se répète cinq fois. Imaginons que si l'on voudrait réaliser cet algorithme avec 100 notes, cela deviendrait fastidieux.

Comme les variables ont des noms différents, on ne peut pas utiliser de boucle, ce qui allonge considérablement le code et le rend très répétitif.

Pour résoudre ce problème, il existe un type de données qui permet de définir plusieurs variables de même type.

II. Définition

Un tableau est une suite d'éléments de même type. Il utilise plusieurs cases mémoire à l'aide d'un seul nom. Comme toutes les cases portent le même nom, elles se différencient par un numéro ou un indice.

Nous pouvons représenter schématiquement un tableau nommé **Note** composé de cinq cases, dans la mémoire comme suit :

	Note[1]	Note[2]	Note[3]	Note[4]	Note[5]	
Note	12	14	11.5	13	12	← Contenu du tableau

Nous disposons alors de cinq variables. Pour les nommer, on indique le nom du tableau suivi de son indice entre crochets ou entre parenthèses : La première s'appelle Note[1], la deuxième Note[2], etc. jusqu'à la dernière Note[5].

Note[4] représente le 4ème élément du tableau **Note** et vaut 13.

III. Tableau à une dimension

III.1. Déclaration

La déclaration d'un tableau permet d'associer à un nom une zone mémoire composée d'un certain nombres de cases mémoires de même type.

Syntaxe :

Variable identificateur : tableau[indice_min .. indice_max] de type
Ou bien
Variable identificateur : tableau[taille] de type

Notes :

- Le premier élément d'un tableau porte l'indice zéro ou l'indice 1 selon les langages.
- La valeur d'un indice doit être un nombre entier.
- La valeur d'un indice doit être inférieure ou égale au nombre d'éléments du tableau. Par exemple, avec le tableau tab[1 .. 20], il est impossible d'écrire tab[0] et tab[21]. Ces expressions font référence à des éléments qui n'existent pas.

Exemple :

L'instruction suivante déclare un tableau de 30 éléments de type réel :

```
Variable Note : tableau[1 .. 30] de Réels
```

Note : c'est le nom du tableau (identificateur).
1 : c'est l'indice du premier élément du tableau.
30 : c'est l'indice du dernier élément du tableau (nombre d'éléments du tableau).

Exercice :

Déclarer deux tableaux nommés A et B composé chacun d'eux de 10 éléments de type chaîne

Solution :

Variables A, B : tableau[1 .. 10] de chaîne

Remarques :

- Lorsqu'on déclare un tableau, on déclare aussi de façon implicite toutes les variables indicées qui le constituent. Nous utiliserons souvent la valeur 1 comme premier indice mais on peut aussi utiliser un autre indice minimum, comme 0. Dans ce cas, l'indice maximum sera égal au nombre d'éléments – 1.
- Dans le langage C, les cases du tableau sont numérotées à partir de 0. En Visual Basic l'indice minimum est 1.

III.2. Utilisation

Les éléments d'un tableau sont des variables indicées qui s'utilisent exactement comme n'importe quelles autres variables classiques. Elles peuvent faire l'objet d'une affectation, elles peuvent figurer dans une expression arithmétique, dans une comparaison, elles peuvent être affichées et saisies etc.

L'utilisation de ces éléments se fait en suite, via le nom du tableau et son indice. Ce dernier peut être soit une valeur (exemple : Note[3]) , soit une variable (exemple : Note[i]) ou encore une expression (exemple : Note[i+1]).

Remarque :

Il ne faut pas confondre l'indice d'un élément d'un tableau avec son contenu. Par exemple, la 4eme maison de la rue n'a pas forcément 4 habitants.

Exemple 1 :

L'instruction suivante affecte à la variable X la valeur du premier élément du tableau Note :

X ← Note[1]

Exemple 2 :

L'instruction suivante affiche le contenu du quatrième élément du tableau Note :

```
Ecrire(Note[4])
```

Exemple 3 :

L'instruction suivante affecte une valeur introduite par l'utilisateur à l'élément trois du tableau Note

```
Lire (Note[3])
```

➢ Parcours complet d'un tableau :

Le fait que les éléments d'un tableau soient indicés permet de parcourir tous ces éléments avec une boucle, on utilisant une variable qui sert d'indice et s'incrémente à chaque tour de boucle.

Exercice 1 :

Ecrire un algorithme permettant de saisir 30 notes et de les afficher après avoir multiplié toutes ces notes par un coefficient fourni par l'utilisateur :

Solution :

```
Algorithme  tableau_note
Variable Note : tableau[1..30] de Réels
           Coef, i : entier
Début
   Ecrire ("Entrer le coefficient")
   Lire (Coef)
   // Remplissage du tableau Note
   Pour i ← 1 jusqu'à 30 Faire
      Ecrire ("Entrer la valeur de la note", i)
      Lire (Note[i])
   FinPour
   // Affichage des notes * Coef
   Pour i ← 1 jusqu'à 30 Faire
      Ecrire (Note[i] * Coef)
   FinPour
Fin
```

Exercice 2 :

Ecrire un algorithme qui calcul la somme des éléments d'un tableau.

Solution :

On suppose que le nombre d'éléments du tableau ne dépasse pas 100.

```
Algorithme  Somme_tableau
Constante M = 100
Variable T : tableau[1..M] de réels
          n, i : entier
          s : réel
Début
   // Demande de la taille du tableau
   Ecrire ("Entrer la taille du tableau")
   Lire (n)
   Si n > M alors
      N ← M
   FinSI
   // Remplissage du tableau
   Pour i ← 1 jusqu'à n Faire
      Ecrire ("Entrer la valeur de l'élément", i, "du tableau")
      Lire (T[i])
   FinPour
   // Calcul la somme  des éléments du tableau
   s ← 0
   Pour i ← 1 jusqu'à n Faire
      s ← s + T[i]
   FinPour
   Ecrire ("La somme des éléments du tableau est :", s)
Fin
```

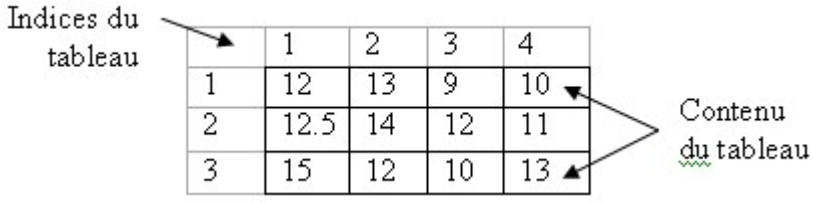
IV. Tableau à deux dimensions

Reprenons l'exemple des notes en considérant cette fois qu'un étudiant a plusieurs notes (une note pour chaque matière). Pour quatre étudiants, nous aurions le tableau de relevés des notes suivant :

	Etudiant 1	Etudiant 2	Etudiant 3	Etudiant 4
Informatique	12	13	9	10
Comptabilité	12.5	14	12	11
Mathématiques	15	12	10	13

Les tableaux à deux dimensions se représentent comme une matrice ayant un certain nombre de lignes (première dimension) et un certain nombre de colonnes (seconde dimension).

Nous pouvons représenter schématiquement un tableau de 3 lignes et de 4 colonnes comme suit :



IV.1. Déclaration

Syntaxe :

Variable identificateur : tableau[1..nb_lignes, 1..nb_colonnes] de type
Ou bien
Variable identificateur : tableau[nb_lignes, nb_colonnes] de type

Exemple :

L'instruction suivante déclare un tableau Note de type réel à deux dimensions composé de 3 lignes et de 4 colonnes :

```
Variable Note : tableau[1..3, 1..4] de réels
```

IV.2. Utilisation

Pour accéder à un élément de la matrice (tableau à deux dimensions), il suffit de préciser, entre crochets, les indices de la case contenant cet

élément.

Les éléments de la matrice peuvent être utilisés comme n'importe quelle variable.

Exemple :

L'instruction suivante affecte à la variable X la valeur du premier élément du tableau Note :

X ← Note[1,1]

➢ Parcours complet d'un tableau à deux dimensions:

Pour parcourir une matrice nous avons besoin de deux boucles, l'une au sein de l'autre, c'est ce qu'on appelle les boucles imbriquées. La première boucle par exemple est conçue pour parcourir les lignes tandis que la deuxième est utilisée pour parcourir les éléments de la ligne précisée par la boucle principale (la première boucle).

Exercice :

Ecrire un algorithme permettant la saisie des notes d'une classe de 30 étudiants en 5 matières.

Solution :

```
Algorithme  Notes
Constante N = 30
          M = 5
Variable note : tableau[1..N, 1..M] de Réels
        i, j : entier
Début
  // Remplissage du tableau note
  Pour i ← 1 jusqu'à N Faire
    Pour j ← 1 jusqu'à M Faire
      Ecrire ("Entrer la note de l'étudiant", i, "dans la matière", j)
      Lire (note[i,j])
    FinPour
  FinPour
Fin
```

V. Tableau dynamique

Les tableaux définis jusqu'ici sont dits statiques, car il faut qu'au moment de l'écriture du programme le programmeur décide de la taille maximale que pourra atteindre le tableau. Si le programmeur donne une taille très grande alors que lui il n'a pas besoin que d'une petite taille, dans ce cas le programme consomme trop de mémoire. Dans certaines situations, on ne peut pas savoir la taille du tableau dans la phase de programmation.

Les tableaux dynamiques sont des tableaux dont la taille n'est définie que lors de l'exécution. Pour créer un tableau dynamique, il suffit de lui affecter une taille vide.

Syntaxe :

Variable identificateur : **tableau** [] de type

Comme un tableau dynamique ne possède pas de taille prédéfinie, il convient de redimensionner le tableau avant de pouvoir s'en servir.

Syntaxe :

Redimensionner identificateur[N]

Exemple :

```
Variable t : tableau[] d'entiers
Redimensionner t[11]
```

La première instruction déclare un tableau dynamique. La deuxième redimensionne la taille du tableau t à 11 éléments.

Note : Tableau dynamique en VB

Pour créer un tableau dynamique d'entiers en VB on utilise la syntaxe suivante :

Dim T() As Integer

L'instruction suivante redimensionne le tableau T en 11 éléments:

Redim T(11)

Vous pouvez utiliser l'instruction ***ReDim*** pour changer la taille d'un tableau qui a déjà été déclaré. Si vous disposez d'un grand tableau et que certains de ses éléments ne sont plus nécessaires, ***ReDim*** peut libérer de la mémoire en réduisant la taille du tableau. En revanche, si votre code détermine qu'un tableau doit contenir plus d'éléments, ***ReDim*** peut les ajouter.

A chaque appel de ***ReDim***, le contenu du tableau précédent est effacé. Pour conserver les valeurs existantes, il faudra rajouter le mot clé ***Preserve***.

Exemple :

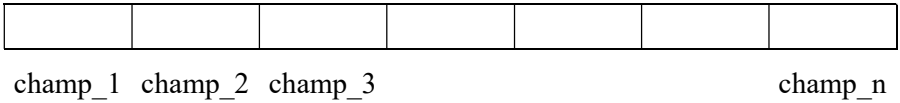
```
Redim Preserve T(11)
```

Chapitre 04 : Les structures

I. Introduction

Contrairement aux tableaux qui permettent de désigner sous un même nom un ensemble de valeurs de même type, chacune d'entre elles étant repéré ***par un indice***, les structures permettent de désigner sous un seul nom un ensemble de valeurs pouvant être de types différents. L'accès à chaque élément de la structure nommé champ se fera, cette fois, ***non plus par une indication de position***, mais par son nom au sein de la structure.

Nous pouvons représenter, schématiquement, une structure comme suit :



II. Déclaration d'une structure

La déclaration d'une structure ne définit aucune variable, elle permet de définir un modèle de structure. Déclarer une structure c'est définir un nouveau type.

La déclaration des structures se fait dans une section spéciale des algorithmes appelée Type, qui précède la section des variables.

Syntaxe:

Type Structure nom_structure
nom_champ1 : type_champ1
…
nom_champN : type_champN
FinStruct

Exemple : Déclaration d'une structure nommée étudiants

```
Type Structure étudiants
  Nom : chaîne
  Prénom : chaîne
  Age : entier
  Moyenne : réel
FinStruct
```

III. Déclaration d'une variable de type structure

Après avoir défini la structure (le type structuré), on peut l'utiliser comme un type normal tel que les types prédéfinis (réel, entier, etc) en déclarant une ou plusieurs variables de ce type.

Syntaxe :

Variable **nom_variable** : **nom_structure**

Une variable de type structure est une variable complexe composée de champs qui sont des variables simples ou complexes ou des tableaux.
Les variables de type structure sont appelées enregistrements.

Exemple :

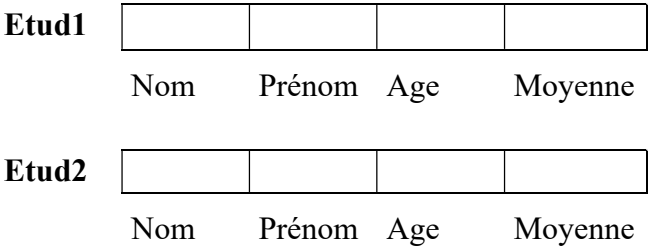
L'instruction suivante déclare deux variables Etud1 et Etud2 de type étudiants :

```
Variables Etud1, Etud2 : étudiants
```

Représentation :

Les enregistrements sont composés de plusieurs zones de

données, correspondant aux champs :



Exercice :

Soit la structure ***Date*** suivante et la variable ***t*** de type Date :

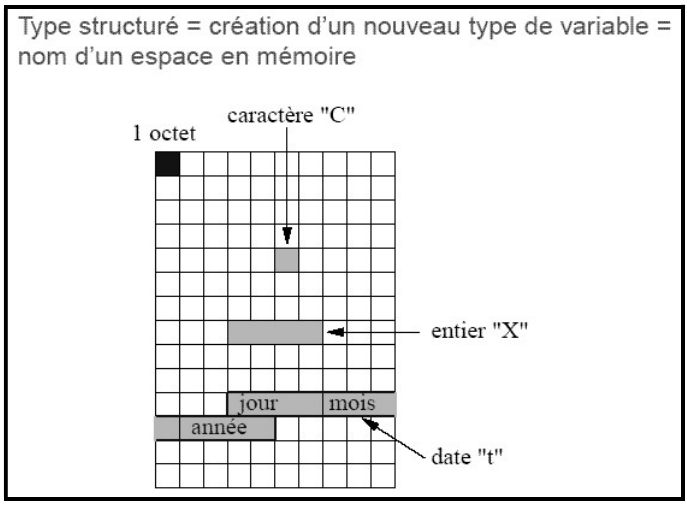
```
Type Structure Date
   jour : entier_long
   mois : entier_long
   annee : entier_long
FinStruct

Variable t : date
```

Donner la taille de la variable t ?

Solution :

La figure suivante montre que la taille de la variable ***t*** est 12 octets.



Remarque :

Si un entier est codé sur 2 octets (entier simple) alors la taille de la variable ***t*** est 6 octets.

IV. L'accès à un champ d'une structure

Chaque champ d'une structure peut être manipulé comme n'importe quelle variable du type correspondant. L'accès à un champ se fait en faisant suivre le nom de la variable de type structure du nom de champ séparé par ***un point***.

Syntaxe :

nom_var.nom_champ

Exemple :

Pour donner un nom à l'étudiant 1, nous emploierons l'instruction suivante :

Etud1.Nom ←"Sabir"

Exercice :

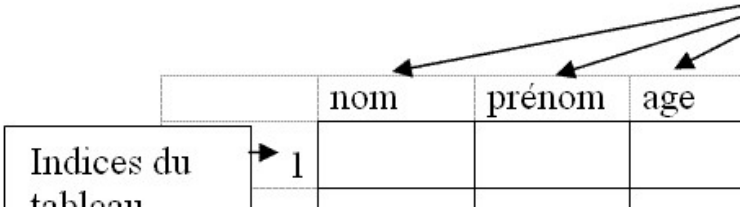
Ecrire un algorithme permettant de remplir la fiche de tous les

étudiants de la classe.

Solution :

```
Algorithme  fiche
Type Structure Classe
   nom : chaîne
   prénom : chaîne
   age : entier
   moyenne : réel
FinStruct
// M :  taille maximale du tableau
Constante M = 100
Variable étudiant : tableau[1.. M] de classe
            i, n : entier
Début
   Ecrire ("Entrer le nombre d'étudiants dans la classe")
   Lire (n)
   Pour i ← 1 jusqu'à n Faire
      Ecrire ("Entrer le nom de l'étudiant ", i)
      Lire (étudiant[i].nom)
      Ecrire ("Entrer le prénom de l'étudiant ", i)
      Lire (étudiant[i].prénom)
      Ecrire ("Entrer l'age de l'étudiant ", i)
      Lire (étudiant[i].age)
      Ecrire ("Entrer la moyenne de l'étudiant ", i)
      Lire (étudiant[i].moyenne)
   FinPour
Fin
```

Représentation :


nom
prénom
age
Indices du
1

Chapitre 05 : Les fonctions prédéfinies

I. Introduction

Certains traitements sont complexes à effectuer par un algorithme. Par exemple, le calcul du cosinus d'un angle nécessite une formule complexe. Tous les langages de programmation ont un certain nombre de fonctions qui permettent de connaître immédiatement ce genre de résultat. Certaines sont indispensables, car elles permettent d'effectuer des traitements qui seraient sans elles impossibles; d'autres servent à soulager le programmeur, en lui épargnant de longs algorithmes.

Tout langage de programmation dispose d'un ensemble de fonctions prédéfinies permettant de procéder à des conversion de type de données, de calcul mathématiques, de manipulation de date et d'heures, de manipulation de chaînes de caractères et bien d'autres fonctions utiles.

II. Les fonctions de chaînes de caractères

Une chaîne est une séquence de caractère dont la longueur correspond au nombre de caractères qu'elle contient. Si une chaîne est vide, sa longueur est égale à zéro. Dans tous les autres cas, elle est égale à un nombre entier positif.

➢ **Longueur d'une chaîne**

Syntaxe :

longueur(ch)

Cette fonction retourne (renvoie) le nombre de caractères d'une chaîne de caractères.

Exemple :

longueur("Bonjour")

Cette fonction retourne 7

➢ La fonction de concaténation de chaînes

La concaténation est la juxtaposition de deux chaînes (ou plus) afin d'en former une seule.

Syntaxe :

concat(ch1, ch2)

Cette fonction retourne une chaîne formée par la concaténation de ch1 et de ch2. La chaîne résultat est formée de ch1 suivi de ch2.

Exemple :

Concat("Bonjour", " Monsieur")
Concat("Bonjour", "Monsieur")
Concat("Bonjour", " ", "Monsieur")

Le résultat de la première instruction est : Bonjour Monsieur.
Le résultat de la deuxième instruction est : BonjourMonsieur.
Le résultat de la troisième instruction est : Bonjour Monsieur.

➢ La fonction de copie de chaînes

Syntaxe :

Copie(ch1, position, n)

Cette fonction recopie une partie de la chaîne ch1 à partir de la position précisée par l'expression entière ***position***, en limitant la recopie au nombre de caractères précisés par l'entier ***n***.

Exemple :

Copie("Bonjour",4,4)

Cette fonction retourne la chaîne ***jour***.

➢ La fonction de comparaison de chaînes

Syntaxe :

comp(ch1, ch2)

Cette fonction compare deux chaînes de caractères en utilisant l'ordre des caractères définis par le code ASCII. Elle fournit une valeur entière définie comme étant :
- positive si ch1 > ch2 ;
- nulle si ch1 = ch2. C'est-à-dire si ces deux chaînes contiennent exactement la même suite de caractères ;
- négative si ch1 < ch2.

Exemple :

Comp("bonjour","monsieur")

Cette fonction retourne une valeur négative.

➢ La fonction de recherche dans une chaîne

Il existe des fonctions de recherche dans une chaîne de caractères de l'occurrence d'un caractère ou d'une autre chaîne de caractères.

Syntaxes :

recherche(ch1, caractère)

Cette fonction recherche dans la chaîne ch1, la première position où apparaît le caractère mentionné.

recherche(ch1, ch2)

Cette fonction recherche dans la chaîne ch1, la première occurrence complète de la chaîne ch2. Elle renvoie un nombre correspondant à la position de la chaîne ch2 dans la chaîne ch1. Si la chaîne ch2 n'est pas comprise dans la chaîne ch1, la fonction renvoie zéro.

Exemple :

recherche("bonjour monsieur","jour")

Cette fonction retourne 4.

III. Les fonctions mathématiques

Les fonctions mathématiques prédéfinies permettent la réalisation d'un traitement mathématique sur des données numériques.

Fonction	Description	Exemple	Résultat
Abs(nombre)	Retourne la valeur absolue d'un nombre.	x←Abs(-12)	x =12
Ent(nombre)	Retourne la partie entière d'un nombre.	x←Ent(12.3)	x =12
Cos(angle)	Retourne une valeur spécifiant le cosinus d'un angle	x←Cos(0)	x =1
Sin(angle)	Retourne une valeur spécifiant le sinus d'un angle.	x←Sin(0)	x = 0
Tan(angle)	Retourne une valeur contenant la tangente d'un angle.	x←Tan(0)	x = 0
Sqrt(nombre)	Retourne une valeur spécifiant la racine carrée d'un nombre	x←Sqrt(4)	x = 2
Alea()	Retourne un nombre aléatoire compris entre 0 (inclus) et 1 (exclu)	x←alea()	0 =< x < 1

La fonction ***Alea*** renvoie un nombre réel aléatoire compris entre 0 et 1. Pour obtenir une valeur entière comprise entre min et max (avec min < max), on utilise la formule suivante :

Ent((max-min +1)*Alea() + min)

Exemple :

Générer un nombre aléatoire X compris entre 5 et 10.

Solution :

```
X ← Ent(6*Alea() + 5)
```

Chapitre 06 : Procédures et fonctions

I. Introduction

Lorsque l'on progresse dans la conception d'un algorithme, ce dernier peut prendre une taille et une complexité croissante. De même des séquences d'instructions peuvent se répéter à plusieurs endroits.

Un algorithme écrit d'un seul tenant devient difficile à comprendre et à gérer dès qu'il dépasse deux pages. La solution consiste alors à découper l'algorithme en plusieurs parties plus petites. Ces parties sont appellées des sous-algorithmes.

Le sous-algorithme est écrit séparément du corps de l'algorithme principal et sera appelé par celui-ci quand ceci sera nécessaire.

Il existe deux sortes de sous-algorithmes : les procédures et les fonctions.

II. Les procédures

Une procédure est une série d'instructions regroupées sous un nom, qui permet d'effectuer des actions par un simple appel de la procédure dans un algorithme ou dans un autre sous-algorithme.

Une procédure renvoie plusieurs valeurs (pas une) ou aucune valeur.

II.1. Déclaration d'une procédure

Syntaxe :

Procédure nom_proc(liste de paramètres)
Variables identificateurs : type
Début
Instruction(s)
FinProc

Après le nom de la procédure, il faut donner la liste des paramètres (s'il y en a) avec leur type respectif. Ces paramètres sont appelés **paramètres formels**. Leur valeur n'est pas connue lors de la création de la procédure.

Exemple :

Ecrire une procédure qui affiche à l'écran une ligne de 15 étoiles puis passe à la ligne suivante.

Solution :

```
Procédure Etoiles()
Variables i : entier
Début
   Pour i ← 1 jusqu'à 15 Faire
      Ecrire ("*")
   FinPour
   // \n : retour à la ligne
   Ecrire("\n")
FinProc
```

II.2. L'appel d'une procédure

Pour déclencher l'exécution d'une procédure dans un programme, il suffit de l'appeler.
L'appel de procédure s'écrit en mettant le nom de la procédure, puis la liste des paramètres, séparés par des virgules.

A l'appel d'une procédure, le programme interrompt son déroulement

normal, exécute les instructions de la procédure, puis retourne au programme appelant et exécute l'instruction suivante.

Syntaxe :

Nom_proc(liste de paramètres)

Les paramètres utilisées lors de l'appel d'une procédure sont appelés paramètres effectifs. Ces paramètres donneront leurs valeurs aux paramètres formels.

Exemple :

En utilisant la procédure ***Etoiles*** déclarée dans l'exemple précédent, écrire un algorithme permettant de dessiner un carré d'étoiles de 15 lignes et de 15 colonnes.

Solution :

```
Algorithme carré_étoiles
Variables j : entier

// Déclaration de la procédure Etoiles()
Procédure Etoiles()
Variables i : entier
Début
   Pour i ← 1 jusqu'à 15 Faire
      Ecrire ("*")
   FinPour
   Ecrire("\n")
FinProc

// Algorithme principal (Partie principale)
Début
   Pour j ← 1 jusqu'à 15 Faire
      // Appel de la procédure Etoiles
      Etoiles()
   FinPour
Fin
```

Remarque 1 :

Pour exécuter un algorithme qui contient des procédures et des fonctions, il faut commencer l'exécution à partir de la partie principale (algorithme principal).

Remarque 2 :

Lors de la conception d'un algorithme deux aspects apparaissent :

- la définition (déclaration) de la procédure ou fonction;
- l'appel de la procédure ou fonction au sein de l'algorithme principal.

II.3. Passage de paramètres

Les échanges d'informations entre une procédure et le sous algorithme appelant se font par l'intermédiaire de paramètres.

Il existe deux principaux types de passages de paramètres qui permettent des usages différents :

➢ Passage par valeur :

Dans ce type de passage, le paramètre formel reçoit uniquement ***une copie*** de la valeur du paramètre effectif. La valeur du paramètre effectif ne sera jamais modifiée.

Exemple :

Soit l'algorithme suivant :

```
Algorithme Passage_par_valeur
Variables N : entier

// Déclaration de la procédure P1
Procédure P1(A : entier)
Début
   A ← A * 2
   Ecrire (A)
FinProc

// Algorithme principal
Début
   N ← 5
   P1(N)
   Ecrire(N)
Fin
```

Cet algorithme définit une procédure P1 pour laquelle on utilise le passage de paramètres par valeur.

Lors de l'appel de la procédure, la valeur du paramètre effectif N est recopiée dans le paramètre formel A. La procédure effectue alors le traitement et affiche la valeur de la variable A, dans ce cas 10.

Après l'appel de la procédure, l'algorithme affiche la valeur de la variable N dans ce cas 5.

La procédure ne modifie pas le paramètre qui est passé par valeur.

➢ Passage par référence ou par adresse :

Dans ce type de passage, la procédure ***utilise l'adresse*** du paramètre effectif. Lorsqu'on utilise l'adresse du paramètre, ***on accède directement à son contenu***. La valeur de la variable effectif sera donc modifiée.

Les paramètres passés par adresse sont précédés du mot clé ***Var***.

Exemple :

Reprenons l'exemple précédent :

```
Algorithme Passage_par_référence
Variables N : entier

// Déclaration de la procédure P1
Procédure P1(Var A : entier)
Début
   A ← A * 2
   Ecrire (A)
FinProc

// Algorithme principal
Début
   N ← 5
   P1(N)
   Ecrire(N)
Fin
```

A l'exécution de la procédure, l'instruction ***Ecrire (A)*** permet d'afficher à l'écran 10. Au retour dans l'algorithme principal, l'instruction ***Ecrire (N)*** affiche également 10.

Dans cet algorithme le paramètre passé correspond à la référence (adresse) de la variable N. Elle est donc modifiée par l'instruction :

```
A ← A * 2
```

Remarque :

Lorsqu'il y a plusieurs paramètres dans la définition d'une procédure, il faut absolument qu'il y en ait le même nombre à l'appel et que l'ordre soit respecté.

III. Les fonctions

Les fonctions sont des sous algorithmes admettant des paramètres et retournant un seul résultat (une seule valeur) de type simple qui peut apparaître dans une expression, dans une comparaison, à la droite d'une affectation, etc.

III.1. Déclaration d'une fonction

Syntaxe :

Fonction nom_Fonct(liste de paramètres) : type
Variables identificateurs : type
Début
Instruction(s)
Retourner Expression
FinFonct

La syntaxe de la déclaration d'une fonction est assez proche de celle d'une procédure à laquelle on ajoute un type qui représente le type de la valeur retournée par la fonction et une instruction Retourner Expression. Cette dernière instruction renvoie au programme appelant le résultat de l'expression placée à la suite du mot clé Retourner.

Note :

Les paramètres sont facultatifs, mais s'il n'y a pas de paramètres, les parenthèses doivent rester présentes.

Exemple :

Définir une fonction qui renvoie le plus grand de deux nombres différents.

Solution :

```
// Déclaration de la fonction Max
Fonction Max( X : réel, Y : réel) : réel
Début
   Si X  > Y alors
      Retourner X
   Sinon
      Retourner Y
   FinSi
FinFonction
```

III.2. L'appel d'une fonction

Pour exécuter une fonction, il suffit de faire appel à elle en écrivant son nom suivie des paramètres effectifs. C'est la même syntaxe qu'une

procédure.

A la différence d'une procédure, la fonction retourne une valeur. L'appel d'une fonction pourra donc être utilisé dans une instruction (affichage, affectation,…) qui utilise sa valeur.

Syntaxe :

Nom_Fonc(liste de paramètres)

Exemple :

Ecrire un algorithme appelant, utilisant la fonction Max de l'exemple précédent.

Solution :

```
Algorithme Appel_fonction_Max
Variables A, B, M : réel

// Déclaration de la fonction Max
Fonction Max( X : réel, Y : réel) : réel
Début
   Si X  > Y alors
      Retourner X
   Sinon
      Retourner Y
   FinSi
FinFonction

// Algorithme principal
Début
   Ecrire ("Donnez la valeur de A")
   Lire(A)
   Ecrire ("Donnez la valeur de B")
   Lire(B)
   // Appel de la fonction Max
   M ← Max(A,B)
   Ecrire ("Le plus grand de ces deux nombres est : ", M)
Fin
```

IV. Portée des variables

La portée d'une variable désigne le domaine de visibilité de cette variable. Une variable peut être déclarée dans deux emplacements distincts.

Une variable déclarée dans la partie déclaration de l'algorithme principal est appelée variable globale. Elle est accessible de n'importe où dans l'algorithme, même depuis les procédures et les fonctions. Elle existe pendant toute la durée de vie du programme.

Une variable déclarée à l'intérieur d'une procédure (ou une fonction) est dite locale. Elle n'est accessible qu'à la procédure au sein de laquelle elle est définie, les autres procédures n'y ont pas accès. La durée de vie d'une variable locale est limitée à la durée d'exécution de la procédure.

Exemple :

```
Algorithme Portée
Variables X, Y : Entier

Procédure P1()
Variables A : entier
Début
   .....
FinProc

// Algorithme principal
Début
   .....
Fin
```

X et Y sont des variables globales, visibles dans tout l'algorithme.

A est une variables locale visible uniquement à l'intérieur de la procédure

Remarque :

Les variables globales sont à éviter pour la maintenance des programmes.

V. La récursivité

Une procédure (ou une fonction) est dite récursive si elle s'appelle elle-même.

Exemple :

Ecrire une fonction récursive permettant de calculer la factorielle d'un entier positif.

Solution :

n ! = n*(n-1) ! : la factorielle de n est n fois la factorielle de n-1 :

```
// Déclaration de la fonction Factorielle (Fact)
Fonction Fact(n : entier) : entier
Début
   Si n >1 alors
      retourner (fact(n-1) * n)
   Sinon
      retourner 1
   FinSI
FinFonction
```

Dans cet exemple, la fonction renvoie 1 si la valeur demandée est inférieure à 1, sinon elle fait appel à elle même avec un paramètre inférieur de 1 par rapport au précédent. Les valeurs de ces paramètres vont en décroissant et atteindront à un moment la valeur une (1). Dans ce cas, il n'y a pas d'appel récursif et donc nous sortons de la fonction.

Note :

Toute procédure ou fonction récursive comporte une instruction (ou un bloc d'instructions) nommée "point terminal" permettant de sortir de la procédure ou de la fonction.

Le "point terminal" dans la fonction récursive Fact est : retourner 1.

VI. Avantages des procédures et fonctions

- Les procédures ou fonctions permettent de ne pas répéter plusieurs fois une même séquence d'instructions au sein du programme (algorithme).
- La mise au point du programme est plus rapide en utilisant des procédures et des fonctions. En effet, elle peut être réalisée en dehors du contexte du programme.

- Une procédure peut être intégrée à un autre programme, ou elle pourra être rangée dans une bibliothèque d'outils ou encore utilisée par n'importe quel programme.

Chapitre 07 : Les fichiers

I. Introduction

Imaginons que l'on veuille écrire un programme permettant d'enregistrer des renseignements (Nom, Prénom, adresse, téléphone) concernant nos clients. Ce programme doit également permettre de consulter et de modifier ces informations. Ces données ne peuvent donc pas être inclues dans l'algorithme, ni entrées au clavier à chaque nouvelle exécution.

Le problème qui se pose réside au niveau de la sauvegarde des informations après la fermeture du programme. Les variables qui sont des cases mémoires ne répondent pas à notre besoin à cause de leur disparition à chaque fin d'exécution.

Les fichiers sont là pour résoudre ce problème. Ils servent à stocker des informations sur un support de stockage (disquette, disque dur, CD Rom, etc.) de manière permanente pour les réutiliser ultérieurement.

II. Types de fichiers

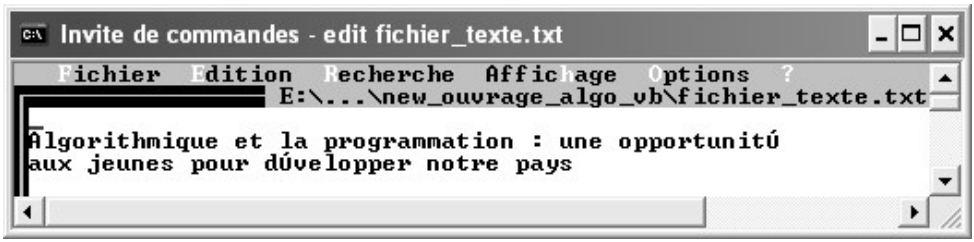
Le critère important qui différencie les fichiers est la façon dont les informations sont organisées sur ces derniers.

II.1. Les fichiers textes

Un fichier texte est formé de caractères ASCII, organisé en lignes, chacune se termine par un caractère de contrôle de fin de ligne.

Si chaque ligne contient le même genre d'informations, les lignes sont appelées des enregistrements. Par exemple, prenons le cas de départ, le fichier est destiné à stoker les coordonnées : nom, prénom, adresse et téléphone de chaque client. Dans ce cas, les informations concernant un client donné doivent être stockées sur une seule ligne.

Les fichiers texte peuvent être crées avec des éditeurs de texte et affichés de manière lisible à l'écran, voir la figure suivante :



II.2. Les fichiers binaires

Un fichier binaire contient des données non textuelles. Il n'est pas organisé sous forme d'enregistrement. Les fichiers binaires ne prennent sens que s'ils sont traités par un programme adapté. Par exemple un fichier son, une vidéo, une image, un programme exécutable, etc.

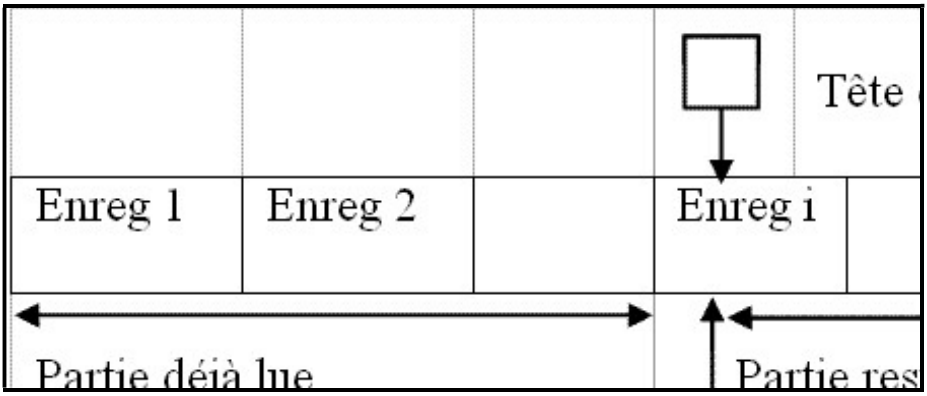
Dans les fichiers binaires, les données sont écrites à l'image exacte de leur codage en mémoire. Ceci facilite l'accès à ce type de fichier et le rend rapide, voir la figure suivante :

III. Types d'accès aux fichiers

Le type d'accès est la technique que la machine doit suivre pour aller chercher les informations contenues dans un fichier. On distingue trois types d'accès aux fichiers

➢ L'accès séquentiel :

Cet accès consiste à traiter les informations séquentiellement, c'est à dire dans l'ordre où elles apparaissent dans le fichier. On ne peut donc accéder à une information qu'en ayant au préalable examiné celle qui la précède. Le schéma suivant montre le principe de parcours d'un fichier à accès séquentiel.



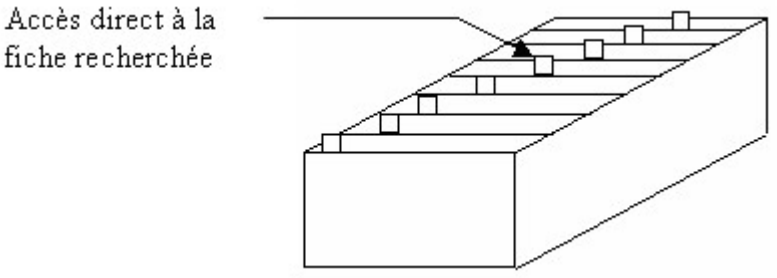
Remarques :

- La fin du fichier est repérée par un marqueur de fin du fichier.
- Pour ajouter une information, il faut que la tête de lecture /écriture se place en face du marqueur de fin de fichier.

➢ L'accès direct

Ce type d'accès consiste à se placer directement sur l'information souhaitée sans parcourir celles qui la précèdent, en précisant la position de l'élément recherché.

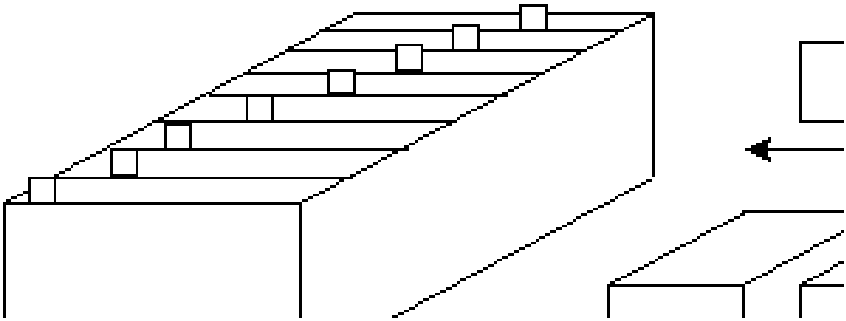
L'indication d'un numéro permet donc un accès direct et rapide à l'information ainsi référencée.



➢ L'accès indexé :

Ce type d'accès combine la rapidité de l'accès direct et la simplicité de l'accès séquentiel. Il est particulièrement adapté au traitement des gros fichiers, comme les bases de données.
Le principe est de créer des fichiers supplémentaires d'index (voir schéma ci-dessous).

Exemple : Fiche des livres



On parcourt un index pour rechercher une clef. On obtient ainsi l'adresse exacte de l'information recherchée. On peut utiliser des opérateurs de comparaisons sur les index (=, ≠, <, <=, >, >=). Il est alors possible, par exemple, de retrouver rapidement toutes les personnes de plus de 18 ans.
Dans l'exemple schématisé ci-dessus, on pourrait, grâce aux index, retrouver rapidement des livres à partir de leur auteur, de leur titre ou de leur thème.

IV. Traitement séquentiel des fichiers texte

IV.1. Ouvrir et fermer un fichier

➢ Ouvrir un fichier texte

Lorsqu'on désire accéder à un fichier, il est nécessaire avant tout accès, d'ouvrir le fichier.

Syntaxe :

Ouvrir Nom_du_Fichier en Num_canal en Mode

Nom_du_Fichier : c'est le nom physique du fichier.

Num_canal : c'est le nom logique du fichier. Pour ouvrir un fichier, il faut lui allouer un numéro du canal valide et disponible.

Mode : le mode d'ouverture du fichier conditionne le travail qui peut être effectué sur ses enregistrements. Il existe trois modes

d'ouverture du fichier texte :

- Lecture : permet d'ouvrir le fichier en lecture seul
- Ecriture : indique son accès en écriture. Dans ce mode, un nouveau fichier est toujours créé. Si le fichier existe déjà, il est réinitialisé à vide et son contenu précédent est perdu.
- Ajout : permet d'ajouter des données à un fichier séquentiel existant en conservant le contenu précédent.

Exemple :

```
// Num_canal égal à 1 si fiche.txt est le premier fichier à ouvrir.
// Il est égal à 2 si un fichier est déjà ouvert et fiche.txt est
// le deuxième fichier à ouvrir
Ouvrir "fiche.txt" en 1 en Lecture
```

Cette instruction ouvre le fichier ***fiche.txt*** en lecture seule. Le fichier à comme nom physique ***fiche.txt*** et comme nom logique un (1).

➢ Fermer un fichier texte

Une fois qu'on a terminé avec un fichier, il ne faut pas oublier de le fermer. On libère ainsi le canal qu'il occupait.

Syntaxe :

Fermer(Nom_du_Fichier)
Ou bien
Fermer (Num_canal)

Note :

Lorsqu'un fichier doit subir plusieurs interventions nécessitant plusieurs ouvertures, il sera nécessaire de fermer le fichier avant de le re-ouvrir.

IV.2. Lire et écrire dans un fichier

Soit un fichier texte contenant des données organisées dans des enregistrements de longueur fixe.

Champs	Nom	Prénom	Ville	Tél
Enreg 1	Mahboub	Khadija	Agadir	028435867
Enreg 2	Sabir	Ahmed	Marrakech	024251674
Enreg 3	Latif	Karima	Agadir	028457197

La lecture et l'écriture dans ce fichier exigent la déclaration de la structure suivante :

```
// Déclaration de la structure client
Type Structure client
   Nom : chaîne * 25
   Prénom : chaîne * 25
   Ville : chaîne * 15
   Tel : chaîne * 10
FinStruct
```

Après avoir défini la structure client, on peut l'utiliser pour créer une ou plusieurs variables correspondant à cette structure :

```
// Clt : variable de type client
Variables Clt : client
```

On peut maintenant remplir les différentes informations contenues au sein de la variable Clt de la manière suivante :

```
Clt.Nom ← "Mahboub"
Clt.Prénom ← "Khadija "
Clt.Ville ← "Agadir "
Clt.Tel ← "028435867"
```

Après avoir rempli les différents champs de la variable Clt, on peut utiliser cette variable pour écrire directement dans le fichier.

EcrireFichier 1, Clt

Pour une opération de lecture, il suffit de recopier un enregistrement dans une variable de type client et d'écrire la syntaxe suivante :

LireFichier 1, Clt

Exercice1 : Fichier Clients.txt

Ecrire un algorithme permettant de créer un fichier ***Clients.txt*** puis saisir les informations du 1er client et de les écrire dans ce fichier.

Solution :

```
Algorithme  Création_Fichier_Clients

// Déclaration de la structure client
Type Structure client
   Nom : chaîne * 25
   Prénom : chaîne * 25
   Ville : chaîne * 15
   Tel : chaîne * 9
FinStruct
// Clt : Variable de type client
Variable Clt : client

// Algorithme principal
Début
   // Création du fichier Clients.txt
   Ouvrir "Clients.txt" en 1 en Ecriture
   // Affectation de données aux champs de la structure
   Clt.Nom ← "Mahboub"
   Clt.Prénom ← "Khadija "
   Clt.Ville ← "Agadir "
   Clt.Tel ← "028435867"
   // Ecriture dans le fichier
   EcrireFichier 1, Clt
    // Fermeture du fichier
   Fermer (1)
Fin
```

Exercice 02 : Fichier clients.txt (suite)

En utilisant une procédure ***Affichage()***, écrire un algorithme permettant d'afficher à l'écran le contenu du fichier ***Clients.txt***. Cet algorithme complète l'exercice précédent.

Remarque :

La fonction booléenne ***EOF*** (acronyme pour End Of File) renvoie ***vrai*** si on a atteint la fin du fichier, sinon, elle renvoie ***faux***. Cette fonction est nommée parfois ***FinFichier***.

Solution :

```
Algorithme  Lecture_Fichier_Clients

// Déclaration de la structure client
Type Structure client
   Nom : chaîne * 25
   Prénom : chaîne * 25
   Ville : chaîne * 15
   Tel : chaîne * 10
FinStruct
Variable Clt : client

// Déclaration de la procédure Affichage
Procédure Affichage()
Début
   TanQue Non EOF(1)
      LireFichier, Clt
      Ecrire Clt
   FinTanQue
FinProc

// Algorithme principal
Début
   // Ouvrir le fichier Clients.txt
   Ouvrir "Clients.txt" en 1 en lecture
   // Appel de la procédure Affichage
   Affichage
   // Fermer le fichier
   Fermer (1)
Fin
```

On peut utiliser dans la procédure ***Affichage*** un tableau de structure dynamique dont le code est le suivant :

```
Variable Clts : Tableau[] de client

Procédure Affichage()
Début
   i ← -1
   Tantque Non EOF(1)
      i ← i + 1
      Redimensionner Clts[i]
      LireFichier 1, Clts[i]
      Ecrire Clts[i]
   FinTantQue
FinProc
```