LES ALGORITHMES

INTRODUCTION :

L'algorithme désigne la description d'une suite d'actions à effectuer dans un ordre donné pour parvenir à un résultat. La personne ou la machine qui exécute l'algorithme doit savoir effectuer les actions qui y sont énumérés. Les choix des actions élémentaire dépend donc de celui qui doit exécuter l'algorithme.

ALGORITHMES LANGAGES ET ORDINATEURS :

Les programmes informatiques utilisés dans les ordinateurs sont la tradition, dans un langage que ces machines peuvent exécuter, des algorithmes décrivant le travail que l'on attend d'eux. Les programmes utilisés partout. On les fabrique à l'aide d'autres programmes appelés compilateurs, interpréteurs en traduisant des textes rédigés dans des langages informatiques comme les langages PASCAL, C, etc.

ECRITURE DES ALGORITHMES :

Au lieu d'utiliser directement un langage de programmation pour exprimer les algorithmes que nous souhaitons voir exécuter, nous allons utiliser le langage que nous manipulons ceci donne la possibilité de s'exprimer sans contrainte de langage.
Les meilleurs algorithmes doivent respecter les trois qualités :

- **Justesse** : fait toujours exactement ce pour quoi il est fait.
- **Clarté** : un algorithme peu clair et peu lisible est difficile de le modifier.
- **Efficacité** : un algorithme efficace s'exécute rapidement et occupe seulement la quantité de mémoire nécessaire dans un ordinateur.

STRUCTURES DE CONTROLE :

Ecrire un algorithme c'est présenter la liste des actions à effectuer dans l'ordre pour réaliser un travail donné. Par exemple, pour aller au travail en voiture, on peut prévoir la suite des actions:
Ouvrir la porte de la voiture,

Monter dans la voiture,
Mettre le moteur en route,
…
Ces actions forment une séquence car ils doivent être exécutés l'un après l'autre dans l'ordre où ils sont fournis.
La séquence ne permet pas de décrire tous les algorithmes, par exemple lorsqu'on exige de tirer le starter avant de mettre le moteur en route si ce dernier est gelé et de actionner le démarreur jusqu'à ce que le moteur tourne, dans ce cas on peut prévoir la démarche suivante :
si le moteur est gelé **alors** tirer le starter
faire tourner le démarreur pendant 2 secondes
si le moteur ne démarre pas **alors** répéter les actions
    couper le contact
    faire tourner le démarreur pendant 2 secondes
La séquence et les structures de contrôle permettent de décrire l'algorithme précédent.

STRUCTURES DE DONNEES :
Il est souvent commode d'organiser l'ensemble des données, par exemple, le nom, le prénom, le matricule, la classe d'un élève dans un enregistrement pour disposer de données le caractérisant.
VARIABLES :
Une variable sert à stocker une valeur, on peut la représenter par un rectangle contenant sa valeur et à coté de ce rectangle on écrit le nom de la variable.
33 A CASABLANCA B
La variable de nom 'A' a pour valeur le nombre 33, la variable de nom «B» a pour valeur la chaîne de caractères 'CASABLANCA'.
On peut toujours changer la valeur d'une variable.
Une variable est caractérisée par son nom et son type qui sont fixés, ainsi que sa valeur qui peut change dans le temps.
Le nom d'une variable commence par une lettre suivie d'autres lettres et de chiffres, mais ne contient pas des caractères spéciaux (signe plus, espace, …).

TYPE DE VARIABLES :

Un type de variable définit l'ensemble des valeurs que peut prendre une variable et les opérations permises sur les valeurs de cette variable.

Quelques types de variables parmi les plus courants sont :

ENTIER : la valeur de la variable de ce type est un nombre entier compris entre deux valeur extrêmes et de signe quelconques. Les opérations permises sont celles de l'arithmétique habituelle sur les nombres entiers. Ainsi, il est possible de comparer des nombres entiers grâce aux symboles =, **!**=, <>, <=, >=.

REEL : la valeur de la variable de ce type est un nombre réel compris entre deux valeur extrêmes et de signe quelconques. Les opérations permises sont celles de l'arithmétique habituelle sur les nombres décimaux. Ainsi, il est possible de comparer des nombres entiers grâce aux symboles =, **!**=, <>, <=, >=.

CHAINE DE CARACTERES : dans ce cas la variable peut contenir une suite de caractères : lettres, chiffres, caractères spéciaux (',','+','#',…). Le nombre de caractères constituent une chaîne de caractères est généralement limité à 255 caractères. La seule opération permise pour ce type de variable est la concaténation.

La comparaison entre deux chaînes de caractères est possible selon l'ordre lexicographique (déterminé à partir d'un ordre défini sur l'ensemble de tous les caractères). Par exemple, 'auto' < 'avion' et '1996' < '43'.

BOOLEEN : Les deux seules valeurs que peut prendre une variable de type booléen sont **Vrai** et **Faux**. Les opérations permises sur ce type de variables sont les opérations logiques (**et**, **ou**, **non**).

CONSTANTES :

Lorsqu'on veut utiliser une valeur plusieurs fois dans un algorithme, il faut l'affecter à une variable au début de l'algorithme:

Constantes Pi=3.141592654 ;

AFFECTATION :

Cette opération fondamentale sur les variables consiste à changer la valeur de la variable. Nous la noterons par '**:=**' et nous écrivons :

- à gauche de ce signe d'affectation le nom de la variable.

- à droit de ce signe d'affectation une expression qui calcule la nouvelle valeur de la variable dont son nom situé à l'autre coté du signe.

L'affectation est symbolisée de diverses façons dans les langages de programmation, par exemple « := » en Pascal, « = » en langage C.
Le symbole d'affectation sera lus **« reçoit »** ou « **prend pour valeur** »

A :=3 ;
B :=A*5+5 ;

ENTREES / SORTIES :

Dans un algorithme, il est indispensable de communiquer avec les utilisateurs (afficher les résultats de calcule, demande la valeur d'une variable donnée). Pour cela nous utilisons les fonctions '**ECRIRE**' et '**LIRE'.**

Par exemple pour calculer le périmètre des rectangles, nous écrivons l'algorithme:

```
VARIABLES
Longueur   : REEL
Largeur    : REEL
Périmètre  : REEL
DEBUT
ECRIRE  "Veuillez entrer longueur du rectangle en mètres "
LIRE  Longueur
ECRIRE  "Veuillez entrer largeur du rectangle en mètres "
LIRE  Largeur
Périmètre :=(Longueur+Largeur)*2
ECRIRE "le périmètre du rectangle est : ", Périmètre," mètres."
FIN
```

TRACE D'UN ALGORITHME :

Pour comprendre le fonctionnement d'un algorithme, il est utile de l'exécuter instruction par instruction.

Exemple :
L'algorithme suivant échange les valeur des variables A et B :

```
VARIABLES
A     : ENTIER
B     : ENTIER
DEBUT
      A: =4
      (1)
      B: =5
      (2)
      A: =B-A   (3)
      B: =B-A   (4)
      A: =A*B   (5)
FIN
```

On note le changement des valeurs des différentes variables dans un tableau appelé « tableau trace de l'algorithme :

instruction	Valeur de A	Valeur de B
(1)	4	?
(2)	4	5
(3)	1	5
(4)	1	4
(5)	4	4

COMMENTAIRES : on commente une instruction ou une bloque d'instructions afin d'obtenir une meilleure lisibilité d'algorithme.

LES CONDITIONS ET LES INSTRUCTIONS DE CONTROLE :
Nous représentons les choix conditionnés dans un algorithme en utilisant les mot **si**, **alors** et **sinon.**
Exemple :

```
VARIABLES
Age :ENTIER
DEBUT
Ecrire "veuillez donner votre âge"
```

```
LIRE  AGE
SI (AGE < 13)
      ALORS ECRIRE"Trop jeune, allez voir un autre film !"
      SINON ECRIRE "Entrez, bonne soirée."
FIN
```

L'instruction **SI** (AGE < 13) **ALORS** écrire "Trop jeune, allez voir un autre film !" **SINON** écrire "Entrez, bonne soirée." est une instruction conditionnelle alternative: l'un ou l'autre des messages "Trop jeune, allez voir un autre film !" et "Entrez, bonne soirée." s'affiche selon la valeur de la condition AGE < 13.

Remarque :

- Pour une meilleure lisibilité de votre algorithme, les mots **ALORS** et **SINON** doivent être décalées vers la droite par rapport au mot **SI**.
- Lorsque plusieurs instructions doivent être effectuées après un même alors ou même sinon, nous indiquons qu'elles forment une séquence en les plaçant entre crochet ouvrant. Comparons par exemple les deux algorithmes :

```
VARIABLES
X, Y, Z : ENTIER
DEBUT
Y :=2
Ecrire "veuillez donner la valeur de X"
LIRE  X
SI (X > 10)
      ALORS
            [ X := X – Y
            [ Y := X
Z :=X*Y
ECRIRE Z
FIN
```

```
VARIABLES
X, Y, Z : ENTIER
DEBUT
Y :=2
Ecrire "veuillez donner la valeur de X"
LIRE  X
SI (X > 10)
      ALORS  X :=X-Y

Y :=X
Z :=X*Y
ECRIRE Z
FIN
```

Ces deux algorithmes sont corrects du point de vue syntaxique. Mais lorsque la condition "X>10" est réalisée, les deux actions "X :=X-Y" et "Y :=1" qui sont exécutées dans celui de la gauche, alors que celui de la droite exécute seulement l'instructions "X :=X-Y".

- La condition fournissant l'un des deux valeurs booléennes Vrai ou Faux selon que celle-ci s'est réalisée ou non.

Les comparaisons (conditions élémentaires) se présentent sous la forme d'égalités ou inégalités. Les opérateurs de comparaison utilisés pour exprimer ces conditions sont l'égalité (=), la non égalité (!= ou <>), les inégalités stricts (< et >) et les inégalités larges (<= ou >=). La table de vérité ci-dessous donne les valeurs prises par les différentes expressions de comparaisons possibles entre deux variables A et B.

A	B	A=B	A !=B	A<B	A>B	A<=B	A>=B
4	5	F	V	V	F	V	F
2	2	V	F	F	F	V	V
7	3	F	V	F	V	F	V

Les conditions complexes sont construites en combinaison entre plusieurs conditions élémentaires. Par exemple, le connecteur logique "**ET**" permet de construire une nouvelle condition "C1 **ET** C2" à partir de deux autres "C1" et "C2". Cette condition ne prend la valeur **Vrai** que si les deux conditions élémentaires "C1" et "C2" ayant la valeur **Vrai**. Elle prend la valeur **Faux** si l'une au mois des deux conditions élémentaires a la valeur **Faux**. Ce que l'on résume par la table de vérité :

C1	C2	C1 **ET** C2
V	V	V
V	F	F
F	V	F
F	F	F

Un autre connecteur **OU** permet de construire une nouvelle condition "C1 **OU** C2" à partir de deux autres "C1" et "C2". Cette condition

prend la valeur **Vrai** dès que l'une au moins des deux conditions a la valeur **Vrai**. Ce qui est exprimé dans la table de vérité :

C1	C2	C1 **OU** C2
V	V	V
V	F	V
F	V	V
F	F	F

Le dernier connecteur logique que nous verrons est le "**NON**" qui porte sur une seule condition dont il fournit la négation.

C	**NON** C
V	F
F	V

En utilisant les parenthèses, exactement comme dans les expressions arithmétiques, il est possible de construire des conditions aussi complexes qu'on le souhaite en combinons des conditions simples avec les connecteurs **ET**, **OU** et **NON.** Par exemple, la conditions ((X>7) **ET** (X<12)) **OU** (X>26) prend la valeur vrai si le nombre X est compris entre 7 et 12 ou bien dépasse 26. Il est simple d'exprimer la négation de cette condition en ajoutant le connecteur **NON** à cette condition en faisant attention aux parenthèses :
NON (((X>7) **ET** (X<12)) **OU** (X>26)**).**

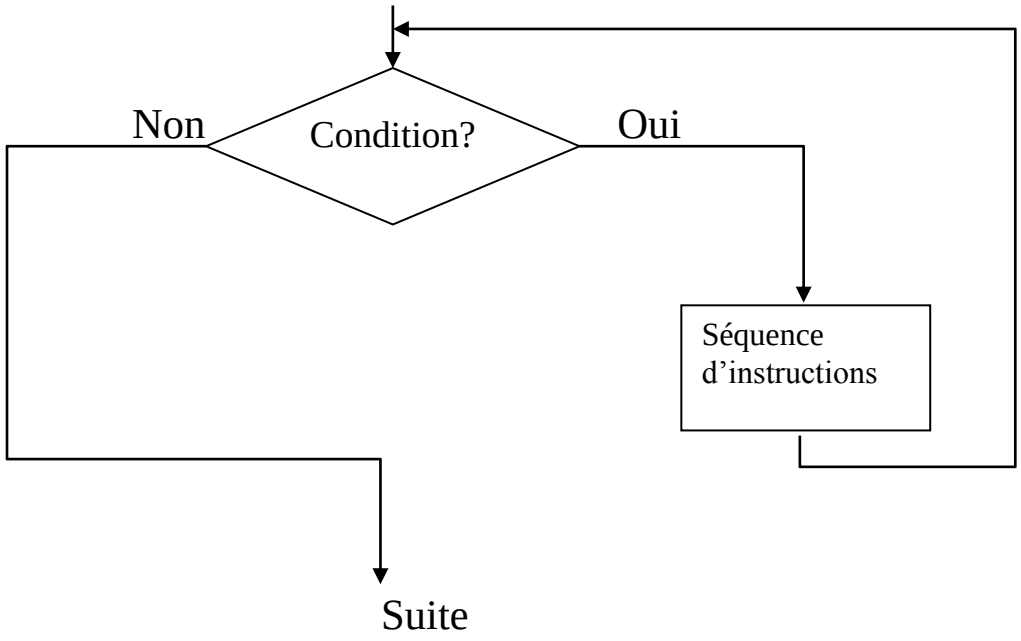
LA REPETITION :
LA BOUCLE **TANT QUE … FAIRE :**

« Voulez-vous enregistrer les modifications apportées au fichier (O/N) ?» est une question à laquelle l'utilisateur peux répondre par oui (O) ou non (N). Lorsque la réponse est incorrecte (ni O ni N), l'algorithme doit rejeter cette réponse et poser à nouveau la question. Dans ce cas, on peut envisager l'algorithme suivant:

```
Variables
R      : caractère
```

```
Début
Ecrire "Voulez-vous enregistrer les modifications apportées au fichier (O/N)?"
Lire R
Tant que (R != "O" et R != "N") faire
        Ecrire "Réponse incorrecte, répondez O ou N"
        Lire R
Fin
```

La boucle "**Tant que** condition **faire** séquence d'instructions" fait répéter une séquence d'instructions tant qu'une condition est satisfaite. Le fonctionnement de cette boucle est décrit par l'organigramme :



On peut décrire le fonctionnement de cette boucle par :

1. Evaluer la condition
2. Si sa valeur est **Oui**, alors exécute la séquence et retourne au 1.
3. Passe à la suite

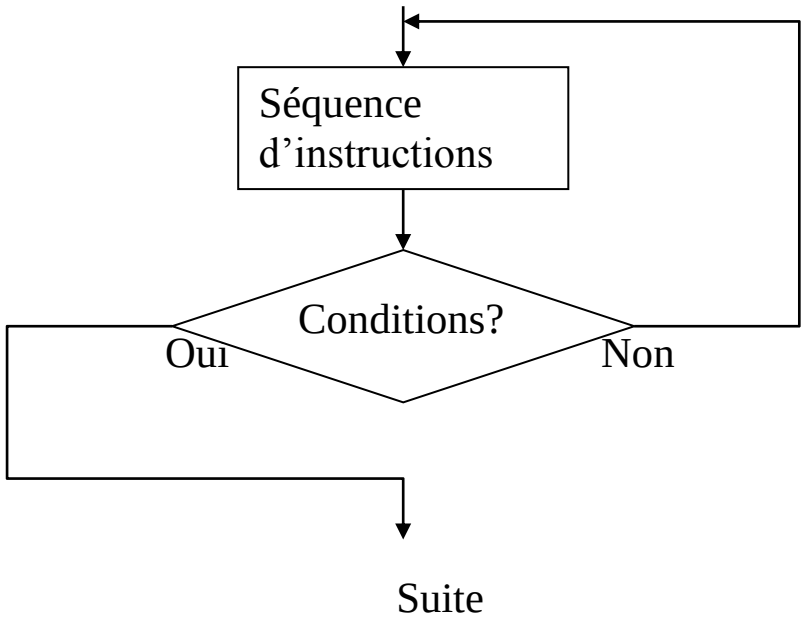
Remarques :

- Les variables intervenant dans la condition de la boucle doivent avoir reçu une valeur avant l'évaluation de cette condition.

- Si 1ère évaluation de la condition de la boucle fournit la valeur **Faux**, le corps de la boucle ne s'exécutera pas et on passe directement à l'instruction suivante.
- Si le corps de la boucle ne change pas la valeur de la condition et si celle-ci a la valeur Vrai, la séquence d'instructions sera répétée sans qu'on passe à la suite : une boucle infinie.

LA BOUCLE **REPETER … JUSQU'A** :

La boucle "**Répéter** séquence d'instructions **Jusqu'à** condition" exécute au moins une fois la séquence d'instructions avant toute évaluation de sa condition. Le fonctionnement de cette boucle est décrit par l'organigramme :



On peut décrire le fonctionnement de cette boucle par :

1. Exécuter toute la séquence
2. Evaluer la condition
3. Si sa valeur est **Non**, alors retourne au 1.
4. Passe à la suite

Remarques :

- Le corps de la boucle est systématiquement exécuté une fois avant le test d'arrêt.

- La condition de la boucle **Répéter** n'est examinée qu'après l'exécution du corps de la boucle.
- Après **Tant que** on écrit une condition pour continuer la répétition, après **Jusqu'à** on écrit une condition d'arrêt.

LA BOUCLE **POUR … FAIRE** :

La boucle "**Pour** variable :=départ **Jusqu'à** arrivée **Faire** séquence d'instructions" exécute un nombre fixe de fois sa séquence d'instructions. Par exemple, pour afficher la table de multiplication de 5 par les chiffres 0 jusqu'à 9.

```
Variables
N      : entier
Début
Pour  N :=0  Jusqu'à   9
Faire
Ecrire "7x ",N, "=",7*N
Fin
```

La variable qui est de type entier s'appelle la variable de contrôle de la boucle.

Remarques :

- Si départ > arrivée la séquence n'est jamais exécutée
- La variable de contrôle change automatiquement de valeur à chaque itération
- La variable de contrôle peut être de tout type sur lequel est défini un ordre par exemple le type caractère.

TABLEAUX

Lorsqu'on désire traités plusieurs variables de même type, par exemple, pour calculer la moyenne des consommations mensuelles d'électricité sur une année, nous avons besoin de stocker ces consommations mensuelles en mémoire. Pour cette raison, nous utilisons la structure de données appelée tableau qui permet de regrouper toute une famille de variables de même type sous un même nom et de résoudre certain type de problèmes. Par exemple, lorsqu'on veut compter pour chaque note de 0 à 20 le nombre des étudiants qui l'ont obtenu.

```
Constantes
Max =100
Variables
Note : tableau de [1..Max] nombres entiers
O    : tableau de [1..21] nombres entiers
N    : entier
I, J  : entiers
Début
Ecrire "Veuillez entrer le  nombre d'étudiants : "
Lire N
Pour I :=1 jusqu'à N faire
        Ecrire "Entrer la note obtenue par le ", I, "étudiant :"
        Lire Note[I]
Pour I :=1 jusqu'à 21 faire
        O[I] :=0
Pour J :=0 jusqu'à 20 faire
        Pour I := 1 jusqu'à N faire
                Si (Note[I] = J ) alors O[J +1]:= O[J +1] +1
        Ecrire "L'occurrence de la note", J,"est :", O[J +1]
Fin
```

On distingue les éléments de tableau par leurs numéros appelé index qui l'on écrit entre deux crochets à la suite du nom de tableau.
Certains langages de programmation exigent que les indices commencent soit à 0 soit à 1.

TABLEAUX A PLUSIEURS DIMENSIONS :

Pour stocker les notes d'une classe, par exemple, la note du 3ème module du 17ème élève, nous utilisons les tableaux à plusieurs indices. Ce type de tableaux se déclare en donnant les bornes de variation des deux indices, par exemple le tableau Notes déclaré par :

Variables Notes : tableau de [1..35] et [1..4] nombres réels.

PROCEDURES ET FONCTIONS :

PROCEDURES :

Si nous voulons poser, dans un algorithme, plusieurs questions auxquelles nous répondons par Oui ou Non. Ceci nécessite de recopier plusieurs fois les mêmes lignes qui constituent la question. Dans ce cas, nous pouvons créer une procédure qui prend en charge de poser les questions :

```
Variables
Question   : chaîne de caractères
Réponse    : caractère
Score              : entier
{Déclaration de la procédure}
Procédure nom_de_procédure(-> Q : chaine de cractères, <- R : caractère)
Début                                {début du corps de la procédure}
Ecrire Q
Lire R
Tant que ((R != "O") et (R != "N")) Faire
       Ecrire "Répondre par O ou N      "
       Lire R
Fin                                  {fin du corps de la procédure}
Début                                {début de l'exécution de l'algorithme}
Score :=0
Question := "Peut-on passer au feu rouge clignotant"
Nom_de_procédure(Question, &Réponse)
Si  Réponse="N" Alors Score :=Score+1
Question := "doit-on marquer un arrêt au STOP"
Nom_de_procédure(Question, &Réponse)
Si réponse= "O" Alors Score :=Score+1
Ecrire "Score marqués :", Score
Fin                                  {fin de l'algorithme}
```

Les procédures sont en général chargées d'exécuter une tache ou une partie spécifique d'un algorithme. Elles sont désignées par un nom et on peut les exécuter à volonté en les appelant dans l'algorithme.

Procédure nom_de_procédure corps_de_procédure défini la procédure de nom nom_de_procédure et son corps corps_de_procédure qui une partie séparée de l'algorithme. Cette définition permet d'attacher le nom de la procédure à la partie de l'algorithme désignée à exécuter par cette procédure.
On appelle la procédure dans le corps de l'algorithme pour exécuter sa tache.
Les procédures doivent être déclarées et définies en premier, mais l'exécution de l'algorithme commence par la première instruction qui suit le mot début de l'algorithme. Dans les langages de programmation, on emploie la dénomination de programme principale pour désigner la partie de l'algorithme qui contient la première instruction à exécuter.
Certains langages reconnaît le programme principal grâce à un en-tête spécifique : main () de la compilateur C.
L'intérêt des procédures est de séparer diverses parties d'un algorithme afin de mieux le structurer pour le rendre lisible :

```
Si condition Alors procédure1
             Sinon procédure2
```

3ème intérêt des procédures est la constitution d'une bibliothèque de procédures spécifique a un domaine comme calcule numérique ou le graphique.

PARAMETRES :

Dans la procédure précédente, il y a deux paramètres. Le premier Q permet à la procédure d'utiliser la valeur de la variable question qui est de même type que le paramètre Q. le deuxième paramètre a permit à la procédure de fournir une valeur à la variable réponse dans le programme principal.

Il est préférable de définir une procédure avec des variables qui lui appartiennent et que l'on fait correspondre à des valeurs ou à des variables du programme appelant pendant la durée d'exécution de procédure.

Les paramètres d'une procédure sont écrits entre parenthèses juste après le nom de la procédure.

PARAMETRE EN SORTIE :

Lors du premier appel à la procédure, la chaîne de caractères "Peut-on passer au feu rouge clignotant ?" qui était la valeur de la variable globale Question est automatiquement affecté au paramètre Q. La procédure seulement utilise la valeur de la variable question en tant que valeur de Q. dans ce cas, on parle de transmission par valeur (paramètre en entrée).

PARAMETRE EN ENTREE :

Nous voulons maintenant fournir les réponses au programme principal, pour cela nous avons ajouté un second paramètre R permettant la modification de la variable Réponse à l'intérieur de la procédure. Nous parlons dans ce cas de passage de paramètre par adresse (paramètre en sortie).

LES FONCTIONS :

Les langages de programmation offrent un vaste choix de fonctions prédéfinies, comme la partie entière d'un nombre réel, la racine carrée, ….

Les fonctions sont proches dont leurs définitions aux procédures à deux différences prés :

- Leur appel doit figurer dans des expressions.
- Leur exécution produit un résultat qui prend la place de la fonction lors de l'évaluation de l'expression.

Exemple :

Diamètre :=2*Racine (A/Pi)

Pour évaluer le membre de la droite de cette affectation, on doit d'abord calculer A/Pi puis fournir le quotient comme paramètre à la fonction racine qui calcule la racine et enfin multiplier le résultat par 2.

Pour définir notre propre fonction, il nous suffit :

- De les déclarer, comme les procédures avant le programme principal en le faisant précéder de mot fonction et suivi du type de résultat.
- De faire figurer dans le corps de la fonction le mot résultat suivi de la valeur ou de l'expression qui est le résultat que l'on veut fournir au programme qui utilise la fonction.

Voici un exemple d'algorithme qui calcule la somme des entiers de 1 à N en utilisant une fonction Somme :

```
Variables
N      : nombre entier
Fonction Somme : nombre entier (->Max : nombre entier)
Début
        Résultat Max*(Max+1)/2
Fin
Début
Ecrire "Somme des entiers de 1 jusqu'à ?"
Lire N
Ecrire Somme (N)
Fin
```

Le type du résultat de la fonction déclaré dans l'en tête de sa définition doit être compatible au type de la valeur fournis par la fonction.

VARIABLES LOCALES ET GLOBALES :

Une procédure ou une fonction peut avoir des variables pour faire son travail. Par exemple pour calculer la somme des entiers de 1 à N sans la formule connue :

```
Variables
N      : nombre entier
Fonction Somme : nombre entier (->Max : nombre entier)
Variables
S, I   : nombre entier
Début
        S :=0
        Pour I :=1 jusqu'à N faire S:=S+I
        Résultat S
Fin
Début
Ecrire "Somme des entiers de 1 jusqu'à ?"
Lire N
Ecrire Somme (N)
```

```
Fin
```

Les variables S et I déclarées à l'intérieur de la fonction sont appelées variables locales à la fonction car elles ne sont utilisables que dans le corps de la fonction.

Il est possible d'utiliser dans une procédure ou une fonction des variables du programme principal. Ces variables sont appelées des variables globales.

DECOMPOSITION D'UN ALGORITHME :

On définit les procédures et les fonctions dans le but de découper un problème complexe en problèmes plus simples et bien assemblés. Pour cela, on doit commencer par écrire le programme principal en y faisant apparaître des appels de procédures et de fonctions.

RECURSIVITE :

Une fonction récursive est une fonction qui utilise elle-même, par exemple pour calculer la somme de 1 à N en remarquant qu'elle peut s'obtenir en ajoutant N à la somme des entiers de 1 à N-1 et ainsi de suite. Pour arrêter ce procédé, remarquons que la somme des entiers de 1 à 1 est 1. Nous obtenons la fonction récursive suivante :

```
Fonction Somme : nombre entier (->N : nombre entier)
Début
Si   N=1        Alors          Résultat 1
                Sinon      Résultat N+Somme (N-1)
Fin
```

LES FICHIERS SEQUENTIELS :

Un ordinateur ne peut exécuter un programme que lorsque celui-ci est stocké dans le mémoire centrale où les données que le programme traite sont stockées. C'est-à-dire que les données s'effacent dès que l'alimentation est interrompue. De plus, les mémoires centrales ont une capacité limitée. Pour ces raisons les données sont stockées dans des mémoires non volatiles et de grande capacité.

Chaque programme est enregistré sur une mémoire de masse sous forme des fichiers programmes qui sont chargés en mémoire centrale pour les exécuter. Les données peuvent être stockées dans des fichiers de données. Lorsqu'un programme a besoin d'utiliser de données

provenant d'un fichier, il demande de les transférer vers des variables en mémoire centrale. Lorsqu'il désire faire conserver les données, il donne l'ordre de copier le contenu de variables dans un fichier de données.

ENREGISTREMENTS :

Le fichier informatique contient un enregistrement par étudiant. Chaque enregistrement contenant des informations (dans des zones appelées champs) sur un étudiant.

Voici un enregistrement correspondant un étudiant :

Nom	Prénom	Date de naissance			Année	Matricule	Groupe
		JJ	MM	AA			
Rida	Ahmed	24	11	67	06/07	342	3

Pour déclarer un enregistrement, nous écrivons :

Variables E : Enregistrement
- nom, prénom : chaîne de caractères
- jj, mm, aa : nombres entiers
- année : chaîne de caractères
- matricule : nombre entier
- groupe : caractère

On désigne un champ par le nom de l'enregistrement suivi d'un point, puis du nom du champ : E.nom, …., E.groupe.

OUVERTURE ET FERMETURE DES FICHIERS :

Pour accéder aux données d'un fichier, le programme doit ouvrir le fichier (c:\chemin\nom_fichier) et préciser s'il désire lire ou écrire dans le fichier. Dans les algorithmes, nous écrivons "ouvrir le fichier en lecture" ou "ouvrir le fichier en écriture". Le programme qui va se terminer ses opérations dans un fichier doit fermer ce dernier. Dans les algorithmes, nous écrivons "fermer le fichier". La fermeture des fichier est très important pour la sécurité des données car un fichier fermé risque moins d'être endommagé.

LES FICHIERS SEQUENTIELS :

Pour consulter un enregistrement, on est obligé de lire les enregistrements dans l'ordre de leur création jusqu'à atteindre celui qui nous intéresse.

Pour atteindre un enregistrement dans des fichiers séquentiels, nous utilisons l'instruction **LIRE** qui permet de lire l'enregistrement en cours et nous plaçons devant l'enregistrement suivant jusqu'à atteindre la fin de fichiers. On dispose d'une fonction booléenne **EOF** qui permet de déterminer s'il y a encore un enregistrement à lire.