SUPPORT DE COURS

ALGORITHMIQUE ET STRUCTURES DES DONNEES I

Elaboré par :

Faten Ben ARFIA BARAKET

(Maitre assistante en systèmes informatique)

SOMMAIRE

CHAPITRE I

INTRODUCTION

Objectif :

Découvrir la notion de l'algorithmique.

Eléments de contenu :

- Comment fonctionne l'ordinateur
- Comment parler à l'ordinateur
  - En langage de programmation
  - Les phases de la démarche de la programmation
- Définitions

I. COMMENT FONCTIONNE L'ORDINATEUR :

L'ordinateur est une machine qui :

- traite automatiquement des données selon un programme enregistré dans sa mémoire
- communique et archive des informations.

Ces différentes fonctions correspondent en fait à trois constituants différents :

1. ***La mémoire centrale (MC) :*** Elle permet de mémoriser les programmes pendant le temps nécessaire à leur exécution. On y trouve également les informations temporaires manipulées par ces programmes.
2. ***L'unité centrale de traitement (CPU ou UCT) :*** Elle est chargée de prélever une à une chaque instruction de programme situé en MC et de l'exécuter.
3. ***Les périphériques :*** Ils désignent tous les appareils susceptibles d'échanger des informations avec la mémoire centrale. Ils sont de deux sortes :
   - *Périphériques d'entrées-sorties :* Ils assurent la communication entre l'homme et l'ordinateur (clavier, écrans, imprimantes, souris,…)
   - *Périphériques d'archivage :* Ils assurent l'archivage de l'information (disques durs, bandes magnétique, disquettes…)

Voici un schéma fonctionnel d'un micro-ordinateur :

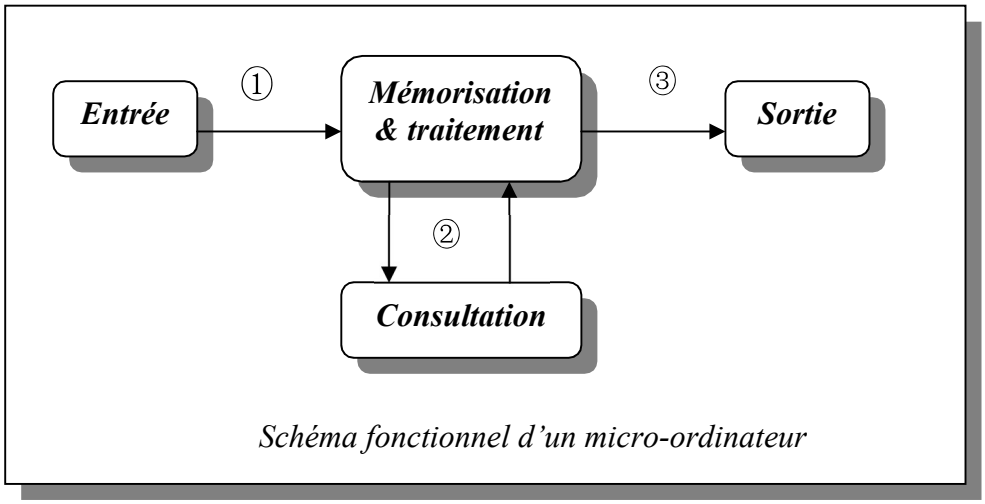


Schéma fonctionnel d'un micro-ordinateur

① Les informations qui viennent de l'extérieur, sont enregistrées en premier temps dans la MC.

② Ces informations seront cherchées à partir de la MC et traitées par l'UCT. Il peut, dans ce cas, y avoir consultation des supports externes d'informations.

③ Les résultats obtenus seront stockés dans la MC, pour être exportés par la suite vers l'extérieur.

II. COMMENT PARLER A L'ORDINATEUR :

II.1 En langage de programmation :

L'ordinateur ne sait exécuter qu'un nombre limité d'opérations élémentaires, dictées par des instructions de programme et codées en binaire. Nous disons alors que l'ordinateur ne peut comprendre que le ***langage machine***. En effet l'ordinateur est incapable de faire que ce soit s'il n'a pas reçu l'ordre et la manière de procéder. Le programme enregistré dans sa mémoire lui dicte instruction par instruction les opérations qu'il doit accomplir. Toutefois cela ne signifie pas que tout programme doit être réalisé en langage machine. Nous pouvons utiliser des langages de programmation tel que C, Pascal,… et ces programmes seront traduits en langage par des compilateurs ou des interpréteurs.

II.2 Les phases de la démarche de la programmation :

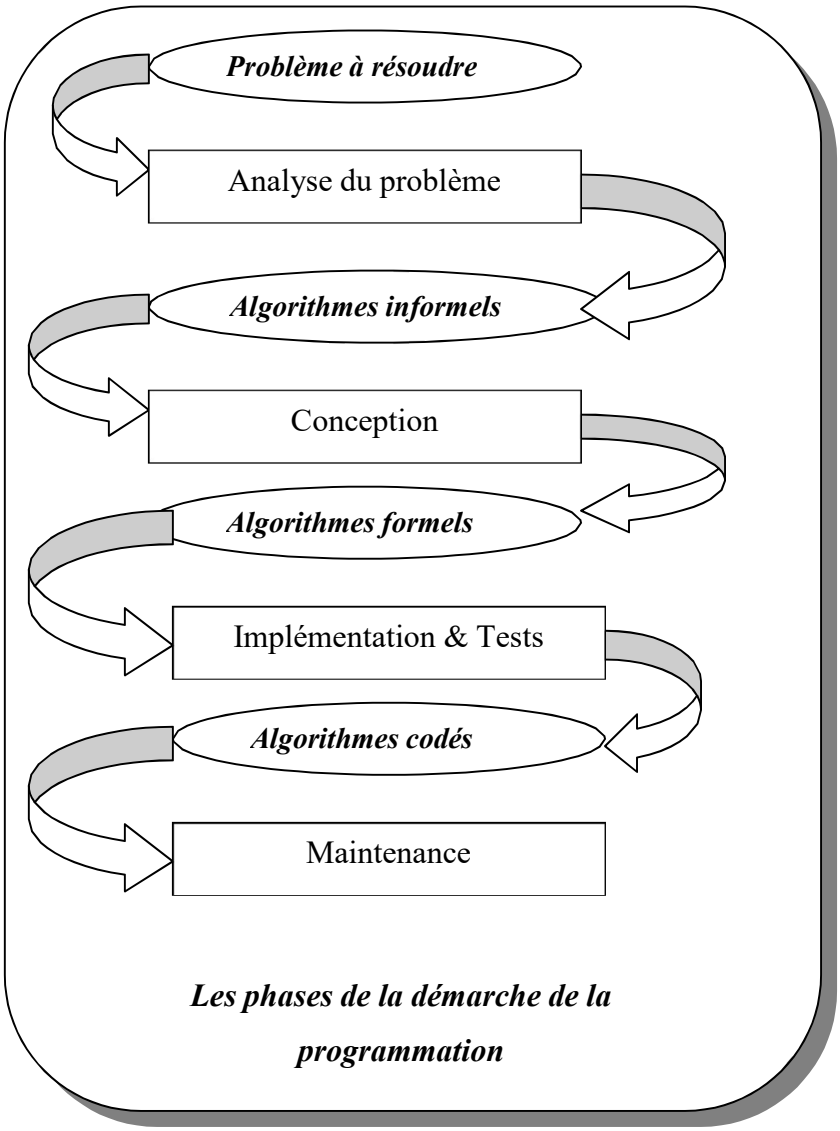
La démarche de la programmation se déroule en quatre phases :

1- ANALYSE DU PROBLEME *:* Dans cette première phase, on cherche par quel moyen, on pourra obtenir les résultats cherchés à partir des données dont on dispose *Cette phase aboutit à l'écriture des algorithmes informels.*

2- CONCEPTION : Elle consiste à exprimer les algorithmes informels dans le langage de programmation discret. Pendant cette phase, on pourra aussi décomposer le problème en entités élémentaires faciles à implémenter et à tester *Cette phase de conception aboutit à l'écriture des algorithmes formels.*

3- IMPLEMENTATION & TEST : On exprime dans un langage de programmation concret (évolué), le résultat de la phase précédente. Si l'analyse et la conception ont été convenablement menées, cette opération se résume en une simple transcription systématique. Pendant cette phase, on vérifie également le bon fonctionnement des unités de programmation *Cette phase aboutit à l'écriture des unités de programmation (algorithmes codés).*

4- MAINTENANCE : Pendant cette dernière phase, il pourrait être nécessaire de modifier le programme pour corriger les défauts, pour améliorer ses performances.



Les phases de la démarche de la programmation

Application :

Problème : En suivant les phases de la démarche de la programmation, écrire un algorithme qui permet de calculer le bénéfice réalisé sur la vente d'un produit donné.

III. DEFINITIONS :

- Un algorithme est la description d'un traitement automatisé de données destiné à être réalisé sur un ordinateur, après avoir traduit cette description dans un langage de programmation.
- Un algorithme est une suite d'opérations élémentaires devant être exécutées dans un ordre donné, pour accomplir une tâche donnée.

CHAPITRE II

LES STRUCTURES DE DONNEES

Objectif :

Comprendre les règles de définition des structures de données simples ainsi que les types simples et les opérations qui les manipulent.

Eléments de contenu :

- La notion de variable
- La notion de constante
- Les types de données :
    - Le type Entier
    - Le type réel
    - Le type caractère
    - Le type booléen
    - Le type énuméré
    - Le type intervalle

I. LA NOTION DE VARIABLE :

Une variable est un objet qui peut prendre différentes valeurs tout le long d'un algorithme.

Toute variable est caractérisée par :

- Son nom (un identificateur unique),
- Son type,
- Son contenu.

La lisibilité d'un algorithme dépendra étroitement de votre habilité à choisir des noms représentatifs des informations qu'ils désignent.

Exemples :

Montant : Pour une variable désignant le montant d'une facture.

Somme : Pour une variable désignant la somme des notes.

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

II. LA NOTION DE CONSTANTE :

Une constante est un objet ayant une valeur inchangée tout le long d'un algorithme.

Toute constante est caractérisée par :

- Son nom (un identificateur unique),
- Sa valeur.

Exemples :

PI = 3.14

e = 2.72

Fréquence = 50

III. LES TYPES DE DONNEES :

Le type permet :

- De déterminer l'ensemble des valeurs qui peuvent être affectées à une variable,
- De fixer l'ensemble des opérations qui peuvent être réalisées sur ces valeurs.

III.1 Le type entier :

Le type entier comprend un sous ensemble des nombres entiers (un sous ensemble de Z).

Les opérations réalisables sur ces nombres sont :

- L'addition : +
- La soustraction : -
- La multiplication : *
- La division réelle : /
- La division entière : DIV, tel que «P DIV Q» donne la partie entière du quotient de la division de P par Q.
- Le modulo : MOD, tel que «P MOD Q» donne le reste de la division entière de P par Q.
- La puissance : ** , tel que «P ** Q» donne P à la puissance Q.

Avec P et Q des valeurs entières positives.

Exemple :

11 DIV 2 vaut 5

11 MOD 2 vaut 1

III.2 Le type réel :

Le type réel comprend un sous ensemble des nombres réels (un sous ensemble de **R**).

Les opérations réalisables sur ces nombres sont :

- L'addition : +
- La soustraction : -
- La multiplication : *
- La division réelle : /
- La puissance : **

Exemples :

-2.5 3.14 0 -6

III.3 Le type caractère :

Le domaine des caractères est l'ensemble des caractères alphanumériques imprimables de l'alphabet latine : il s'agira des lettres (minuscules et majuscules), des chiffres, des symboles spéciaux (de ponctuation, signes et autres). Outre ces caractères imprimables, on admettra

tous les caractères spéciaux non imprimables ayant des significations particulières tel que le retour chariot, l'échappe (escape),etc. Un caractère sera toujours noté entre apostrophes.

Exemples :

'a' , 'b' , 'F' , '$' , '.'

Les opérations permises sur le type caractère sont les suivantes :

- Les opérateurs de comparaison (< , <= , > , >= , = , <>)

Exemples :

'A' < 'B' est une proposition vrai

'y' > 'd' est une proposition vrai

- **ORD('C') :** Renvoie le code ASCII du caractère C. Le résultat est un entier positif.

Exemples :

ORD('A') = 65

ORD('a') = 97

- **CHR(X) :** Renvoie le caractère dont le code ASCII est X.

Exemples :

CHR(65) = 'A'

CHR(97) = 'a'

- **SUCC(C) :** Renvoie le caractère successeur de C s'il existe.

Exemple :

SUCC('C') = 'D'

- **PRED (C) :** Renvoie le caractère prédécesseur de C s'il existe.

Exemple :

PRED('C') = 'B'

III.4 Le type booléen :

Ce type contient deux valeurs logiques **vrai** ou **faux**. Les opérateurs logiques qu'on peut appliquer sur ce type sont les suivants :

- **NON :** Négation
- **ET :** Conjonction
- **OU :** Disjonction

La table de vérité de ces opérateurs est la suivante :

X	Y	NON (X)	X ET Y	X OU Y
Vrai	Vrai	Faux	Vrai	Vrai
Vrai	Faux	Faux	Faux	Vrai
Faux	Vrai	Vrai	Faux	Vrai
Faux	Faux	Vrai	Faux	Faux

Exemples :

X booléen

Y booléen

X ← vrai

Y ← Non (X) : Y contient la valeur faux

- ***Opérateurs de comparaison arithmétique :***

< , > , <= , >= , = , !=

Exemples :

Note ← 9

Test ← Note > 10 : Test contient la valeur faux

III.5 Le type énuméré :

Il est défini par un ensemble fini de valeurs qui sont énumérées.

Syntaxe :

Type nom du type est { valeur 1, valeur 2,…,valeur n }

Exemples :

Type couleur est { bleu, blanc, rouge, jaune, vert}

Type saison est {printemps, Eté, Automne, Hiver }

Type forme est { rectangle, carré, triangle, ellipse}

Les opérations permises sur le type énuméré sont les suivantes :

- **SUCC (valeur) :** Appliquée à une valeur de ce type, elle retourne celle qui la suit si elle existe.
- **PRED (valeur) :** Appliquée à une valeur de ce type, elle retourne celle qui la précède si elle existe.
- **ORD (valeur) :** appliquée à une valeur de ce type, elle retourne son rang.

Exemples :

SUCC (Printemps) est Eté

PRED (Rouge) est blanc

ORD (rouge) est 3

III.6 Le type intervalle :

Le type intervalle possède les propriétés d'un type scalaire discret ordonné (entier, caractère, et scalaire énuméré)

La définition d'un intervalle est décrite par la donnée de deux constantes Borne Inférieure et borne supérieure appartenant à un type scalaire discret ordonné.

Syntaxe :

Type Type_intervalle = borne_inf .. borne_sup

Exemples :

Type lettre_maj = 'A' …'Z'

Type lettre_min = 'a' …'z'

Type jour_ouvrable = lundi …samedi

Type Mois = 1 .. 12

CHAPITRE III

LES STRUCTURES DE CONTROLE

Objectif :

Comprendre les structures de contrôle.

Eléments de contenu :

- Structure générale d'un algorithme
- Instruction d'affectation
- L'instruction d'écriture
- L'instruction de lecture
- Les structures conditionnelles
  - Structure conditionnelle à un seul choix
  - Structure conditionnelle à deux choix
  - Structure conditionnelle imbriquée
  - Structure conditionnelle à plusieurs choix

I. STRUCTURE GENERALE D'UN ALGORITHME :

Un algorithme est une suite structurée et finie d'actions ou d'instructions pour résoudre un problème

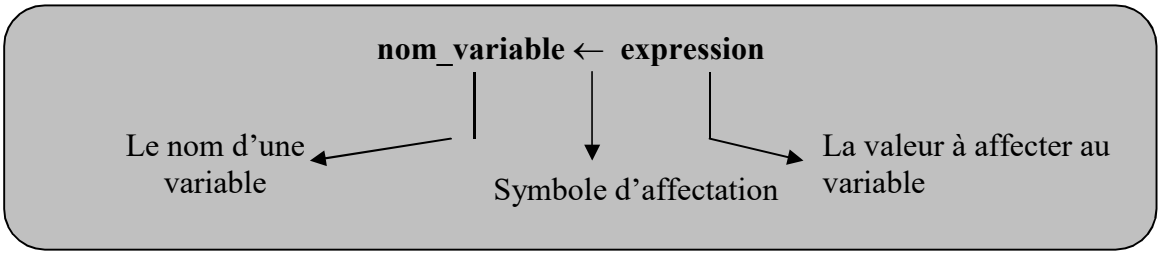
```
-- En-tête
ALGORITHME Nom_algorithme

-- Partie déclarative
CONST
    -- Liste des constantes avec leurs valeurs
TYPE
    -- Liste des types personnalisés
VAR
    -- Liste des variables avec leurs types
-- Corps de l'algorithme
DEBUT
    Instruction 1
    Instruction 2
    ...
    Instruction n
FIN
```

La structure générale d'un algorithme

II. INSTRUCTION D'AFFECTATION :

Cette action permet d'affecter une valeur à une variable, autrement dit, elle permet de modifier la valeur de la variable.

Syntaxe :



A gauche du symbole ← on trouve le nom d'une variable destinée à recevoir une valeur.

A droite du symbole ← on trouve une expression donnant une valeur.

Exemple 1 :

A ← 3
B ← A * 5
A ← B – 2
C ← A
C ← B + 6

Exemple 2 :

Pour chaque variable, on indique quelle est sa valeur après l'exécution de chacune des instructions mentionnées.

	X	**Y**
X ← 1	1	-
Y ← A + 3	1	4
X ← 3	3	4

Remarque : il est interdit d'écrire

A + B ← 8

Application 1 :

Donner pour chaque instruction la valeur contenue dans chaque variable :

	A	**B**	**C**
A ← 2			
B ← A + 4			
A ← 3			
B ← 6			
C ← A + B			
A ← 4			
C ← B - A			

Application 2 :

Ecrire l'algorithme de permutation de deux variables A et B en utilisant une variable ***AUX*** supplémentaire.

III. L'INSTRUCTION D'ECRITURE :

Elle permet de communiquer une donnée à partir d'un périphérique de sortie (écran).

Syntaxe :

Ecrire (expression_1, expression_2, ..., expression_n)

Ce sont des noms des données qui sont sous forme de variables, constantes ou d'expressions

Exemples :

- Ecrire (B) : afficher le contenu de la variable B.
- Ecrire ("ISSAT SOUSSE") : afficher la chaîne "ISSAT SOUSSE"
- Ecrire ("ISSAT SOUSSE", B) : afficher la chaîne "ISSAT SOUSSE" puis la valeur contenue dans la variable B.

Exemple :

```
Algorithme Ecriture
   Const
      a : Entier = 3
   Var
      x, y, z : Entier
Début
      x ← a
      y ← 15
      z ← x + y
      Ecrire (x, y)
      Ecrire ("la valeur de z est : ", z)
Fin
```

IV. L'INSTRUCTION DE LECTURE :

Elle permet d'introduire une donnée à partir d'un périphérique d'entrée (le clavier) et de l'affecter à une variable.

Syntaxe :

Lire (nom_variable_1, nom_variable_2, ..., nom_variable_n)

les noms de variables destinées à ranger les valeurs prises sur le périphérique

Exemple : écrire un algorithme carré qui permet de lire un nombre entier et d'afficher son carré.

```
Algorithme Carré
Var
   nombre, carré : Entier
Début
   Lire (nombre)
   carré ← nombre * nombre
   Ecrire ("Le carré de", nombre, "est : ", carré )
Fin
```

Application :

Ecrire un algorithme qui lit :

- Le prix hors taxe d'un article
- Le nombre d'articles
- Le taux de TVA

Et qui affiche le total tout taxe compris.

V. LES STRUCTURES CONDITIONNELLES :

Tous les langages de programmation possèdent des instructions permettant d'exécuter une séquence d'instructions si une condition est vérifiée. La séquence d'instructions peut être composée d'une ou plusieurs instructions. La condition s'exprime sous la forme d'une expression logique (booléenne) simple ou combinée (plusieurs conditions composées avec les opérateurs logiques ET, OU et NON).

On distingue quatre structures de traitement conditionnel :

- Une structure conditionnelle à un seul choix,
- Une structure conditionnelle à deux choix,
- Une structure conditionnelle imbriquée,
- Une structure conditionnelle à plusieurs choix.

V.1 Structure conditionnelle à un seul choix :

La structure conditionnelle simple se présente sous la forme suivante :

Structure conditionnelle à un seul choix

SI *(condition)* ALORS

Traitement

FIN SI

Si la condition est vérifiée (vraie) alors le traitement s'exécute. Dans le cas contraire, ne rien faire.

Exemple : Algorithme permettant d'afficher la valeur absolue d'un nombre saisi.

Algorithme Valeur_absolue

Var

Nombre : Réel

Début

Ecrire ("Donnez un nombre ")

Lire (Nombre)

Si (Nombre < 0) alors

Nombre ← - Nombre

Fin si

Ecrire ("La valeur absolue est", Nombre)

Fin

V.2 Structure conditionnelle à deux choix :

La structure conditionnelle à deux choix se présente sous la forme suivante :

Structure conditionnelle à deux choix

SI ***(condition)*** **ALORS**

Traitement 1

SINON

Traitement 2

FIN SI

Si la condition est vérifiée alors le traitement 1 s'exécute. Dans le cas contraire, c'est le traitement 2 qui va s'exécuter.

Exemple 1 : Algorithme permettant de comparer deux nombres réels.

Algorithme Comparaison

Var

X, Y : Réel

Début

Ecrire ("Donnez deux nombres")

```
  Lire (X,Y)
 Si (X > Y) alors
        Ecrire (" X est supérieur à Y")
            Sinon
        Ecrire (" Y est supérieur à X")
  Fin si
Fin
```

Exemple 2 : Cet algorithme permet de lire un nombre entier puis vérifier s'il est pair ou impair.

```
Algorithme Pair_Impair
Var
        nombre, reste : Entier
Début
        Ecrire ("Donnez un nombre entier")
        Lire (nombre)
        reste ← nombre  Mod  2
        Si reste = 0
           Alors
                Ecrire ("Le nombre ", nombre, " est pair")
           Sinon
                Ecrire ("Le nombre ", nombre, " est impair")
        Fin Si
Fin
```

V.3 Structure conditionnelle imbriquée :

On peut avoir besoin parfois d'imbriquer des schémas conditionnels et c'est selon le problème à résoudre. La structure conditionnelle imbriquée se présente sous la forme suivante :

Structure conditionnelle imbriquée

SI ***(condition1)*** **ALORS** traitement 1
SINON SI ***(condition2)*** **ALORS** traitement 2
SINON SI ***(condition3)*** **ALORS** traitement 3
...
...
SINON SI ***(condition N-1)*** **ALORS** traitement N-1
SINON traitement N
FIN SI

Exemple 1 : Calcul de remise. A partir d'un montant lu en données, on détermine un montant net par application d'une remise de :

- 5% si le montant est compris entre 2000 D (inclus) et 5000 D (inclus).
- 10% si le montant est supérieur ou égal à 5000 D.

```
Algorithme Calcul_remise
Var
   Montant, Remise : Réel
   Taux : Entier
Début
   Ecrire ("Donner le montant")
   Lire (Montant)
   Si  Montant < 2000
      Alors
        Taux ← 0
      Sinon
        Si  Montant <= 5000
           Alors
              Taux ← 5
           Sinon
              Taux ← 10
        Fin Si
   Fin Si
   Remise ← Montant * Taux / 100
   Montant ← Montant - Remise
Fin
```

Application1 :

A partir de la saisie du prix unitaire d'un produit (PU) et de la quantité commandée (QTCOM), afficher le prix à payer (PAP), en détaillant le port (PORT) et la remise (REM), sachant que :

- Le port est gratuit si le prix des produits (TOT = PU * QTCOM) est supérieur à 1000 dinars. Dans le cas contraire, le port est de 2% sur le total (TOT).
- La remise est de 5% si TOT est compris entre 500 et 2000 Dinars et de 10% au delà de 2000.

Application2 :

Facturation d'électricité : on désire calculer le montant de la facture d'électricité d'un abonné sachant que l'abonné :

1. paye 2500 millimes (frais d'abonnement) même s'il n'a rien consommé.
2. Paye sa consommation selon un tarif à tranches:
   - 100 millimes par Kwh pour les 100 premiers Kwh.
   - 75 millimes par Kwh pour les 150 Kwh suivants.
   - 50 millimes par Kwh pour la fraction de consommation qui excède 250 Kwh.

Les valeurs prélevées sur le compteur de l'abonné :

- *AI (Ancien_Index) :* variable numérique
- *NI (Nouvel_Index) :* variable numérique

Ecrire un algorithme qui nous permet d'obtenir la somme à payer dans une variable numérique *Montant.*

V.4 Structure conditionnelle à plusieurs choix :

Cette structure permet de faire un choix parmi plusieurs possibilités. Le choix du traitement à effectuer dépend de la valeur que prendra un sélecteur. Ce sélecteur est une variable, cette variable est comparée à une série de valeurs ou à un ou plusieurs intervalles. En cas d'égalité l'instruction est exécutée. Les autres ne seront pas exécutées. La structure conditionnelle à plusieurs choix se présente sous la forme suivante :

Structure conditionnelle à plusieurs choix

```
SELON Sélecteur FAIRE
   Liste_valeurs1 : Traitement 1
   Liste_valeurs2 : Traitement 2
   ...
   Liste_valeursN : Traitement N
   Autre : Traitement
FIN SELON
```

- ***Sélecteur :*** C'est un identificateur de variable ou une expression de type scalaire.
- ***Liste_valeurs1..Liste_valeursN :*** Ce sont des valeurs servant aux tests. Elles peuvent être données sous forme de constantes et/ou d'intervalles constantes detype compatible avec le sélecteur.

Exemple_1 : algorithme permettant à partir d'un menu affiché à l'écran, d'effectuer la somme ou le produit ou la moyenne de trois nombres.

```
Algorithme Menu
Var
   nb1, nb2, nb3 : Réel
   choix : Entier
Début
   Ecrire ("Donner trois nombres")
   Lire (nb1, nb2, nb3)
   -- Affichage du menu et saisie du choix
   Ecrire (" 1- pour la multiplication")
   Ecrire (" 2- pour la somme")
   Ecrire (" 3- pour la moyenne")
   Ecrire (" Votre choix :")
   Lire (choix)
   Selon (choix) faire
        1 : Ecrire ("le produit des trois nombres est : " , nb1*nb2*nb3)
        2 : Ecrire ("la somme des trois nombres est : " , nb1+nb2+nb3)
        3 : Ecrire ("la moyenne des trois nombres est : " , (nb1+nb2+nb3)/3)
        autre : Ecrire ("saisie de choix incorrecte")
   Fin Selon
Fin
```

Exemple 2 : Cet algorithme permet de saisir une date sous la forme JJ MM AA et d'afficher le nombre de jours du mois MM et le nombre de jours qui restent pour la fin de ce mois.

```
Algorithme Jours
   Var
        jj, mm, aa, jtot, jreste : Entier
Début
        Ecrire ("Donnez la date sous la forme JJ MM AA :")
        Lire (jj, mm, aa)
        -- On suppose que la date saisie par l'utilisateur est valide
       Selon  mm  faire
       1, 3, 5, 7, 8, 10, 12  : jtot ← 31   -- Janvier, Mars, Mai, Juillet, Août, Oct., Déc.
       4, 6, 9, 11            : jtot ← 30   -- Avril, Juin, Sép. et Nov.
       Autre                                -- Février
          Si (aa Mod 4 = 0)                 -- Année bissextile (Février a 29 jours)
             Alors
                jtot ← 29
             Sinon
                jtot ← 28
          Fin si
        Fin Selon
        jreste ← jtot - jj
        Ecrire ("Le mois ", mm, " a ", jtot, " jours et il reste encore ", jreste, " jours pour sa
   fin").
Fin
```

LES STRUCTURES ITERATIVES

Objectif :

Comprendre les structures itératives.

Eléments de contenu :

- Introduction
- La structure "répéter…jusqu'à"
- La structure "tant que…faire"
- La structure "pour…faire″

I. INTRODUCTION :

Dans un algorithme, il arrive souvent qu'une ou plusieurs instructions doivent être exécutées plusieurs fois dans une structure répétitive appelée boucle.

Dans une boucle, le nombre de répétitions peut être connu, fixé à l'avance, comme il peut dépendre d'une condition permettant l'arrêt et la sortie de cette boucle.

On distingue trois formes de boucles :

- La boucle ***Répéter…Jusqu'à***
- La boucle ***Tant que…Faire***
- La boucle ***Pour…Faire***

II. LA STRUCTURE "REPETER…JUSQU'A" :

Cette structure permet la répétition d'une ou plusieurs instructions jusqu'à ce qu'une condition soit vérifiée. Sa structure générale se présente ainsi :

La structure Répéter…Jusqu'à

REPETER
Instruction 1
Instruction 2
...
Instruction n
JUSQU'A (*Condition)*

- La *condition* est testée après chaque exécution du traitement. Si elle est vérifiée, la boucle s'arrête.
- Si la *condition* n'a pas été modifiée, on aura dans ce cas une boucle *infinie*.
- La séquence d'instructions est exécutée au moins une fois.
- Le nombre de répétitions dans la boucle est inconnu à l'avance.

Exemple : On veut lire, un nombre inférieur à 100. Soit l'algorithme suivant :

Algorithme Nbre_Inf_100
Var
Nombre : Réel
Début
Ecrire ("Donnez un nombre < 100")
Répéter
Lire (Nombre)
Jusqu'à (Nombre < 100)
Ecrire (Nombre)
Fin

Application :

Ecrire un algorithme qui permet de saisir un entier compris entre 5 et 10 en utilisant la structure ***répéter…jusqu'à***.

III. LA STRUCTURE "TANT QUE…FAIRE" :

Cette structure permet la répétition d'une ou plusieurs instructions tant qu'une condition est satisfaite. Sa forme générale est la suivante :

La structure Tant que…Faire

TANT QUE *(Condition)* **FAIRE**
Instruction 1
Instruction 2
...
Instruction n
FIN TANT QUE

- La *condition* est testée avant chaque exécution du traitement, si elle n'est pas vérifiée la boucle s'arrête, d'où le nombre de répétitions dans la boucle peut ne pasêtre connu à l'avance.
- Si le traitement de la boucle ***Tant Que*** est composé d'une seule instruction, il n'est pas utile de mentionner la fin de la boucle ***Fin Tant Que***.

Exemple : On désire réécrire l'algorithme précédent "Nbr_Inf_100" en utilisant le schéma répétitif Tant que…Faire

Algorithme Nbre_Inf_100
Var
Nombre : Réel
Début
Ecrire ("Donnez un nombre < 100")
Lire (Nombre)
Tant que (Nombre ≥ 100) **Faire**
Lire (Nombre)
Fin Tant que
Ecrire (Nombre)
Fin

Application :

Ecrire un algorithme qui permet d'afficher les nombres entiers de 1 à 10 en utilisant une boucle «Tant que…Faire».

IV. LA STRUCTURE "POUR…FAIRE" :

Cette structure permet de répéter un ensemble d'instructions un certain nombre de fois connu à l'avance. Sa forme générale est la suivante :

La structure Pour…Faire

POUR *Compteur* **De** *Début* **A** *Fin* [PAS *Incrément*] **FAIRE**
Instruction 1
Instruction 2
...
Instruction n
FIN POUR

- ***Compteur :*** variable de contrôle.
- ***Début et Fin :*** respectivement les valeurs initiale et finale de la variable de contrôle.
- ***Incrément :*** Valeur d'incrémentation de Compteur après chaque exécution dela boucle. Si aucune valeur n'est indiquée, l'argument **Pas** prend par défaut la valeur 1 comme valeur d'augmentation

- Dans cette structure répétitive ***Pour…Faire*** on n'a pas besoin d'augmenter (ou diminuer) la valeur du compteur à chaque exécution de la boucle puisque cette opération s'effectue automatiquement.
- Si la valeur initiale est égale à la valeur finale, la boucle est exécutée une seule fois.

Exemple : Supposons que l'on souhaite répéter 10 fois les instructions permettant de lire un nombre entier et d'en écrire le carré.

```
Algorithme Carré
Var
   Nombre, Carré, Compteur : Entier
Début
   Pour  Compteur De 1 A  10  Faire
      Lire (Nombre)
      Carré ← Nombre * Nombre
      Ecrire (Carré)
   Fin Pour
Fin
```

Application 1 :

Ecrire un algorithme qui permet de calculer la somme des 10 premiers entiers naturels positifs.

Application 2 :

Ecrire un algorithme qui permet d'afficher la table de multiplication par 7.

CHAPITRE V

LES TYPES COMPOSES

Objectif :

Comprendre les types composés et les opérations qui les manipulent.

Eléments de contenu :

- Les chaînes de caractères
  - Définition
  - Déclaration
  - Opérations de manipulation
- Les tableaux :
  - Les tableaux à une dimension
  - Les tableaux à deux dimensions
- Les enregistrements :
  - Définition
  - Déclaration
  - Opérations de manipulation

I. LES CHAINES DE CARACTERES :

I.1 Définition :

Une chaîne de caractères est une succession de n caractères. C'est un tableau dont les éléments sont des caractères.

I.2 Déclaration d'une variable chaîne :

- Une variable chaîne peut être d'une taille indéfinie (en fait d'une taille ne pouvant dépasser 255 caractères), elle est alors déclarée comme suit :

```
VAR
    Nom_Chaîne : Chaîne
```

- Si on veut spécifier une taille limite à une chaîne, on la déclare comme suit :

```
VAR
    Nom_Chaîne : Chaîne[TailleMax]
```

Exemple :

Prénom : Chaîne

Nom : Chaîne [20]

Adresse : Chaîne [30]

A la variable Nom, on peut affecter une chaîne de caractères ayant au maximum 20 caractères, à l'inverse de prénom pour laquelle le nombre de caractères est visiblementindéfini.

Exemple :

Nom ← "" -- Chaîne vide

Nom ← "Ben Amor"

Prénom ← "Amor"

Adresse ← "4, rue de Tunis"

Remarque :

On pourra accéder en lecture et en écriture au $i^{ème}$ caractère d'une chaîne CH en utilisant la notation **CH[i]** ou $1 \leq i \leq long(CH)$.

Exemple :

Nom ← "Ben Amor"

Nom [1] donne 'B'

Nom [6] donne 'M'

I.3 Les opérations de manipulation :

En plus des actions élémentaires de l'algorithmique (Lecture, Ecriture, Affectation), on trouve les opérations suivantes :

- **LONG (CH) :** Calcule la longueur d'une chaîne, en nombre de caractères.

Exemple :

L ← LONG("bonjour") → L a la valeur 7

L ← LONG("ISSAT SOUSSE") → L a la valeur 12

- **CONCAT (CH1, CH2, ……..,CHn) :** Retourne la concaténation des chaînes CH1, CH2,.. ,CHn.

Exemple :

CH ← CONCAT ("ISSAT", " ","SOUSSE") → CH contient la chaîne "ISSAT SOUSSE"

- **POS (CH1,CH2) :** Retourne la position à partir de laquelle CH2 est incluse dans CH1 sinon elle retourne la valeur 0.

Exemple :

Mot1 ← "Ben Amor"

Mot2 ← "Ben"

i ← POS (Mot1, Mot2) → i contient la valeur 1

j ← POS (Mot1, "Amor") → j contient la valeur 5

- **SOUS_CHAINE (CH, NBC, P) :** Retourne une sous chaîne d'une longueur ***NBC*** à partir de la position ***P*** dans ***CH.***

Exemple :

CH ← SOUS_CHAINE ("ISSAT SOUSSE", 4, 6) → CH contient la chaîne "SOUS"

- **MAJ (CH) :** Retourne la chaîne CH convertie en majuscule.

Exemple :

CH ← MAJ("nizar") → CH contient "NIZAR"

- **MIN(CH) :** Retourne la chaîne CH convertie en minuscule.

Exemple :

CH ← MIN("SOUSSE") → CH contient "sousse"

Application :

Ecrire un algorithme qui permet de :

- Lire une chaîne de caractères,
- Afficher la longueur de la chaîne,
- Afficher le caractère du milieu de la chaîne.

II. LES TABLEAUX :

II.1 Les tableaux à une dimension :

II.1.1 Définition :

Un tableau est un ensemble de données qui sont toutes de même type, qui possèdent un identificateur unique (le nom du tableau) et qui se différencient les unes des autres, dans ce tableau, par leur numéro d'indice.

II.1.2 Déclaration :

La déclaration se fait comme suit :

```
VAR
    Nom_Variable = Tableau [borne_inf...borne_sup] de type_élément
```

- ***borne_inf….borne_sup :*** Est un intervalle de valeurs scalaires : entier, caractère, type énuméré.
- ***borne_inf:*** Borne inférieure de l'intervalle des indices.
- ***borne_sup:*** Borne supérieure de l'intervalle des indices.
- ***type_élément*** peut être :
  - un type simple (logique, caractère, entier, etc…),
  - un type composé (chaîne de caractère, etc…).

Exemple :

NOTES tableau [1…30] de Réel

ALPHA tableau [0…26] de Caractère

Type CARTES est {trèfle, cœur, pique, carreau}

JEU tableau [trèfle…carreau] de entier

II.1.3 Accès à une valeur dans un tableau :

Pour désigner un élément d'un tableau on ajoute au nom de la variable ce qu'on appelle un indice qui doit être compris entre borne_inf et borne_sup.

Syntaxe :

```
Nom_Variable [indice]
```

Exemple :

NOTES [3] ←12.5

ALPHA [0] ← 'd'

JEU [cœur] ← 3

JEU [trèfle] ← 7

Application :

Ecrire un algorithme qui permet de charger un tableau de taille 10 par des entiers impairs. Ensuite faites la somme de ces éléments.

II.2 Les tableaux à deux dimensions :

II.2.1 Définition :

On peut aussi avoir des tableaux à deux dimensions (permettant ainsi de représenter par exemple des matrices à deux dimensions). Un tableau à deux dimensions est définie par un nombre de lignes et un nombre de colonnes.

II.2.2 Déclaration :

On déclare une matrice à deux dimensions de la façon suivante :

```
VAR
      Nom_Variable = Tableau [1...NL][1…NC] de type_élément
```

- **NL :** Nombre de lignes
- **NC :** Nombre de colonnes

II.2.3 Accès à une valeur:

Pour accéder aux éléments d'un tableau de type matrice on ajoute au nom de la variable ce qu'on appelle un indice pour les lignes et un autre pour les colonnes.

Syntaxe :

```
Nom_Variable [Nl] [Nc]
```

- **Nl :** Numéro de ligne,
- **Nc :** Numéro de colonne.

Application 1 :

Ecrire un algorithme qui permet de charger une matrice par des entiers positifs.

III. LES ENREGISTREMENTS :

III.1 Définition :

Un enregistrement est une structure composée d'un nombre fixe de composantes qu'on appelle champs.

Les champs peuvent être de types différents et chaque champ a un nom qui est l'identificateur du champ et qui servira pour le sélectionner.

III.2 Déclaration :

Syntaxe :

```
Type nom_enreg = Structure
        Nom_champs 1 : type 1
        Nom_champs 2 : type 2
        ……
        Nom_champs n : type n
Fin structure

Var non  var : nom  enreg
```

Exemple :

Un étudiant est décrit par son numéro de carte d'étudiant, nom, prénom, note et âge. La structure correspondante sera la suivante :

Type étudiant = **structure**

NCE : Entier
Nom : Chaîne
Prénom : Chaîne
Note : Réel
Age : Entier

Fin structure

Var Etud : étudiant

III.3 Opérations de manipulation :

L'accès à un champ donné fait par l'identificateur de la variable séparée par un point(**.**) de l'identificateur du champ

Syntaxe :

Nom_Var•nom_champ

Exemple :

Etud•Nom → Correspond au nom de l'étudiant Etud.

Etud•Note → Correspond a la note de l'étudiant Etud.

Si on veut gérer l'information date de naissance pour la structure étudiant. On doit alors ajouter un champ DateDeNaissance. Or la date contient 3 champs : numéro du jour, numérode mois et année. On aura besoin d'une structure pour définir le type de la date de naissance :

Type

Jour = 1..31
Mois = 1..12
Date = **structure**
JJ : Jour
MM : Mois
AA : entier
Fin Structure
Etudiant = **structure**
NCE : Entier
Nom : chaîne
Prénom : chaîne
Note : Réel
DateDeNaissance : Date
Fin Structure

Var Et : Etudiant

Pour accéder à l'information numéro du mois de la date de naissance de l'étudiant Et, on doit mettre :

Et•DateDeNaissance•MM

Si on veut enregistrer l'ensemble des notes de 6 matières pour chaque étudiant, l'information Note ne sera plus de type réel, mais plutôt de type tableau de taille 6. on aura alors la structure Etudiant comme suit :

```
Type    Etudiant = structure
                NCE : Entier
                Nom : chaîne
                Prénom : chaîne
                DateDeNaissance : Date
                Note : tableau [1..6 ] de réel
         Fin Structure
```

Pour accéder à la Note n°4 de l'étudiant Et, on doit mettre : Et•Note[4]

Si on veut gérer l'ensemble des informations de N étudiants d'une classe, on aura alors besoin d'un tableau de N éléments dont chaque case doit porter les informations d'un seul étudiant. On aura alors :

Var ***Tab_Etud : tableau [1..N] d'étudiant***

Pour accéder à l'information Prénom de l'étudiant n°5 :

Tab_Etud[5]•Prénom

Pour accéder à l'information numéro du jour de la date de naissance de l'étudiant n°5 :

Tab_Etud[5]•DateDeNaissance•JJ

Pour accéder à l'information Note n°5 de l'étudiant n°5 :

Tab_Etud[5]•Note[5]

Application :

Ecrire un algorithme qui permet de :

- Remplir un tableau de N étudiants,
- Chercher l'étudiant majeur du tableau.

Un étudiant est défini par : Matricule, nom, prénom, adresse, filière, niveau, âge et moyenne.

CHAPITRE VI

LA PROGRAMMATION PROCEDURALE

Objectif :

Comprendre les principes de la programmation procédurale.

Eléments de contenu :

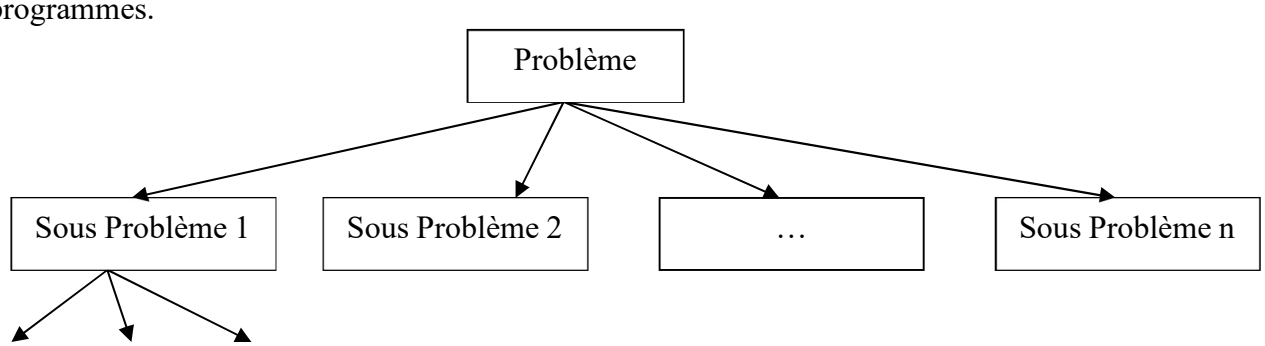
- L'analyse modulaire
- Les sous programmes
- Portée des variables
- Les paramètres formels
- Les paramètres effectifs
- Les fonctions
  - Définition
  - Déclaration
  - Appel de fonction
- Les procédures
  - Définition
  - Déclaration
  - Appel de procédure

I. L'ANALYSE MODULAIRE :

«Diviser les difficultés en autant de parcelles qu'il se peut afin de mieux résoudre» Descartes

La conception d'un algorithme procède en général par des affinements successifs : on décompose le problème à résoudre en sous problèmes, puis ces derniers à leur tour, jusqu'à obtenir des problèmes «facile à résoudre», pour chaque sous problème on écrira un sous programme.

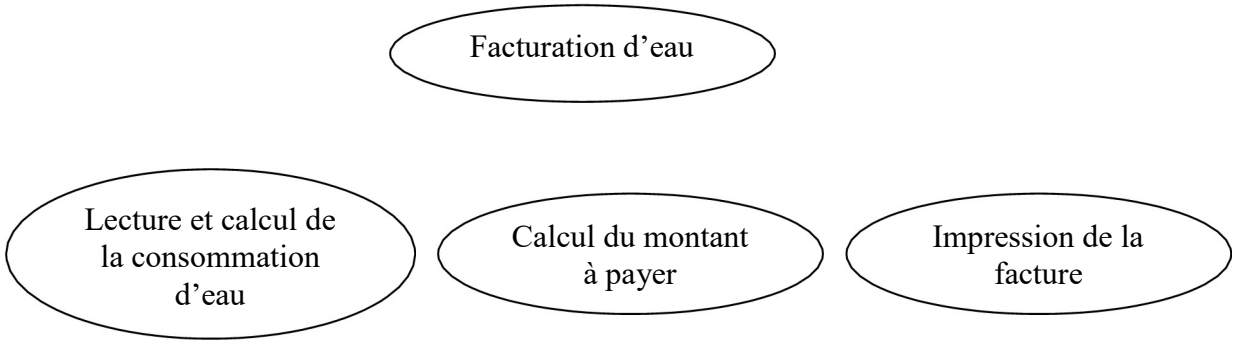
Ainsi la résolution du problème sera composée d'un algorithme principal et d'un certain nombre de sous programmes. L'algorithme principal a pour but d'organiser l'enchaînementdes sous programmes.



L'intérêt de l'analyse modulaire est :

- Répartir les difficultés
- Faciliter la résolution d'un problème complexe
- Pouvoir poursuivre l'analyse comme si les sous problèmes étaient résolus
- Faciliter l'écriture de l'algorithme en évitant les duplications

<u>Exemple</u> :



II. LES SOUS PROGRAMMES :

C'est une unité fonctionnelle formée d'un bloc d'instructions, nommée et éventuellement paramétrée. Que l'on déclare afin de pouvoir l'appeler par son nom en affectant s'il y a lieu des valeurs à ses paramètres. Les données fournies au sous programme et les résultats produits par ce dernier sont appelés des arguments ou des paramètres.

Un sous programme peut être une procédure ou une fonction.

III. PORTEE DES VARIABLES :

La *portée* d'une variable, c'est l'étendu du programme dans laquelle la variable est reconnu et peut être utilisée.

Une variable peut être définie de deux façons dans une structure de programme faisant appel à plus d'un sous programme. Elle peut être ***Locale*** ou ***Globale***.

- ***Variable locale :***

Une variable locale a une portée qui se limite dans le sous programme dans lequel elle est définie.

Exemple : Soit un sous-algorithme qui permet d'afficher la somme de deux entiers.

```
Sous-algorithme Affiche_Somme
Var  A, B, S : entier
Début
        Lire (A)
        Lire (B)
        S ← A + B
        Ecrire (S)
Fin
```

Les variables A, B, S dans le bloc Var sont des ***variables locales*** au sous-algorithme.

- **Variable globale :**

Une variable globale a une portée qui s'étend sur la structure générale des programmes. Elle est reconnue dans le programme et les sous programmes.

Une variable Globale est précédée par ***Global*** lors de sa déclaration.

Exemple : Dans l'exemple précédent, si on suppose que les deux entiers à traiter sont déclarés dans l'algorithme principal comme des variables globales, on aura l'algorithme et le sous-algorithme suivants :

```
Algorithme principal Somme
Global  X, Y : Entier
Début
        Lire (X)
        Lire (Y)
        Affiche_Somme
Fin
```

```
Sous-algorithme Affiche_Somme
Var  S : entier
Début
        S ← X + Y
        Ecrire (S)
Fin
```

IV. LES PARAMETRES FORMELS : (PARAMETRES FICTIFS)

Ce sont des variables déclarées lors de la définition d'un sous-algorithme. Ce type de variable est utilisé pour la communication entre le sous-algorithme appelant et sous-algorithme appelé. Lors de la définition du sous-algorithme, ils n'ont pas une valeur réelle.

Exemple : Dans cet exemple les variables X, Y peuvent être utilisées comme des paramètres formels.

```
Sous-algorithme Affiche_Somme ( X, Y : entier)
Var  S : entier
Début
        S ← X + Y
        Ecrire (S)
Fin
```

V. LES PARAMETRES EFFECTIFS : (PARAMETRES REELS)

Lors d'un appel, un sous-algorithme appelant doit transférer les informations nécessaires à l'exécution d'un sous-algorithme appelé. Ces informations peuvent différer de valeurs d'un appel à un autre.

↳ Les paramètres référencés lors d'un appel à un sous-algorithme sont des paramètres réels puisqu'ils correspondent à des variables ayant une existence réelle.

Exemple :

L'algorithme principal peut appeler le sous-algorithme avec des paramètres réels. Si on désire afficher la somme de deux entiers donnés par l'utilisateur, on aura l'algorithme suivant :

```
Algorithme principal Somme
Global  X, Y : Entier
Début
        Lire (X)
        Lire (Y)
        Affiche_Somme (X, Y)
Fin
```

VI. LES FONCTIONS :

VI.1 Définition :

- Une fonction est un sous programme admettant des paramètres et retournant un *seul* résultat.
  - Les paramètres sont en nombre fixe (≥0).
  - Une fonction possède un seul type, qui est le type de la valeur retournée.
  - Le passage des paramètres est uniquement en entrée : c'est pour cela qu'il n'est pas précisé.
- Généralement le nom d'une fonction est soit un nom (par exemple *minimum*), soit une question (par exemple *estVide*).

VI.2 Déclaration :

On déclare une fonction de la façon suivante :

FONCTION *Nom_Fonction* (Liste des paramètres formels) : *Type_Résultat*
Déclaration des variables locales
DEBUT
Instructions de la fonction
Nom_Fonction ← Résultat (a)
FIN

- *Type_Résultat :* c'est le type de la valeur retournée par la fonction.
- (a) : c'est une instruction d'affectation de l'identificateur de la fonction. Ce dernier (*Nom_Fonction*) doit impérativement prendre une valeur pour que la fonction puisse la retourner dans l'expression du module appelant.

Exemple 1 :

```
Fonction Factoriel ( n : entier) : entier
Var
   F,i : entier
Début
F ← 1
Pour i de 1 à n
        Faire
              F ← F * i
        Fin faire
Factoriel ← F
Fin
```

Exemple 2 :

```
Fonction valeur_absolue ( X : réel) : réel
Var
        A : réel
Début
Si (X>0)  Alors
        A ← X
        Sinon
        A ←   -X
Fin si
valeur_absolue ← A
Fin
```

Exemple 3 :

```
Fonction Surface (R : Réel) : Réel
Const
    Pi : Réel  = 3.14159
Début
    Surface ← Pi * R* R
Fin
```

VI. 3 Appel de fonction :

- L'appel de fonction est une expression, dont la valeur est le résultat retourné par cette fonction, son appel s'effectue donc comme si on va évaluer une expression.
- Les paramètres d'appel doivent correspondre en nombre, en type et en position aux paramètres formels de la fonction.
- La liste des paramètres peut être vide mais elle comporte généralement au moins un paramètre.

Trois façon d'appel :

- Var ← ***Nom_Fonction(paramètres effectifs)***
- Ecrire ("Le résultat est ", ***Nom_Fonction(paramètres effectifs)***)
- Expression opérateur ***Nom_Fonction(paramètres effectifs)***

Exemple 1 :

```
A ← valeur_absolue(-31)
D ← -7.5
C ← valeur_absolue(D) – D * A
```

Exemple 2 : Fonction de calcul de la surface d'un disque.

```
Algorithme  Calcul_Surface
Var
    Rayon : Réel
Début
    Ecrire ("Saisir le rayon du disque : ")
    Lire (Rayon)
    Ecrire ("Surface = ", Surface(Rayon))
Fin

Fonction  Surface (R : Réel) : Réel
Const
    Pi : Réel  = 3.14159
Début
    Surface ← Pi * R* R
Fin
```

Application :

Ecrire un algorithme qui permet de calculer la somme suivante :

$$\sum (i !) / N \quad \text{pour i allant de 0 à N}$$

En utilisant les fonctions suivantes :

- Une fonction FACT pour calculer le factoriel d'un entier,
- Une fonction TERME pour calculer le terme ((i !) / N).

VII. LES PROCEDURES :

VII.1 Définition :

- Une procédure est un sous programme qui ne retourne aucun résultat
- Par contre elle admet des paramètres avec des passages :
  - En entrée, préfixés par ***Entrée*** (ou **E**),
  - En sortie, préfixés par ***Sortie*** (ou **S**),
  - En entrée/sortie, préfixés par ***Entrée/Sortie*** (ou **E/S**).
- Généralement le nom d'une procédure est un verbe.

VII.2 Déclaration :

On déclare une procédure de la façon suivante :

PROCEDURE ***Nom_****Procédure* (**Entrée** Liste de paramètres formels en entrée ;
Sortie Liste de paramètres formels en sortie ;
Entrée/Sortie Liste de paramètres formels en entrée/sortie***)***
Déclarations des variables locales
DEBUT
Instruction 1
Instruction 2
…
Instruction N
FIN

Avec :

- Paramètre fourni en ***Entrée*** : Ce paramètre est fourni par le programme ou le sous programme appelant, sa valeur reste inchangée après l'exécution du sous programme appelé.
- Paramètre fourni en ***Sortie*** : Ce paramètre est retourné par la procédure appelée, l'exécution de la procédure appelée calcule une valeur qui lui sera affectée en fin de traitement.
- Paramètre fourni en ***Entrée/Sortie*** *:* Ce paramètre est fourni par le sous programme appelant, est modifié par la procédure appelée.

Exemple :

Procédure Affiche_Somme (**Entrée** a : entier ; **Entrée** b : entier)
VAR
S : entier
Début
S ← a + b
Ecrire (S)
Fin

VII.3 Appel de procédure:

L'appel d'une procédure avec des paramètres en entrée se fait par le nom de la procédure suivi de la liste des paramètres effectifs transmis à la procédure.

Syntaxe :

Nom_Procédure **(paramètres effectifs)**

Exemple :

```
Algorithme Affiche
VAR
      X, Y : entier
Début
      Ecrire ( "Donnez un entier")
      Lire (X)
      Ecrire ( "Donnez un autre entier")
      Lire (Y)
      Affiche_Somme (X, Y)
Fin
```

Application :

Ecrire

- Une fonction **SAISIE** qui permet de saisir une chaîne de caractères non vide.
- Une procédure **POSITION** qui permet de chercher un caractère C donne dans une chaîne CH donnée. Si le caractère existe ; l a procédure doit faire retourner sa première position dans la chaîne ;0 sinon. Cette procédure suppose que la fonction POS n'existe pas.
- Une procédure **OCCRRENCE** qui permet de chercher le nombre d'occurrences d'un caractère dans une chaîne en utilisant la procédure POSITION.
- Un algorithme **COMPTER** qui permet de saisir un caractère, une chaîne de caractères et d'afficher le nombre d'occurrences d'un caractère C dans une chaîne CH.

Travaux dirigés

SERIE TD N°: 1

AFFECTATION, LECTURE & ECRITURE

Exercice 1 :
Soit un entier A>0 et exprimé en secondes, convertir A en heures, minutes et secondes.
Exemple : A=6400 secondes donne 1 heure 46 mn 4 secondes.

Exercice 2:
Ecrire un algorithme qui permet de convertir une vitesse donnée en ***Km/h*** en ***m/mn*** et ***m/s***.

Exercice 3 :
Ecrire un algorithme qui permet de calculer le prix d'une livraison de fuel en tenant compte d'une remise de 5 % et la TVA de 17 %.

Exercice 4:

On considère X, Y, Z trois variables numériques, on désire effectuer la permutation circulaire entre X, Y et Z en utilisant une variable supplémentaire ou nom. Ecrire la séquence d'actions dans les deux cas.

Exercice 5:
Ecrire un algorithme qui permet de calculer la somme de deux réels et d'afficher le résultat de cette somme.

Exercice 6:
Donner le contenu des variables **a**, **b** et **c** après l'exécution de chacune des instructions suivantes :

1-
a ← 5
b ← 3
c ← a + b
a ← 2
c ← b - a

2-
a ← 5
b ← a + 4
a ← a + 1
b ← a - 4

3-
a ← 5
b ← 10
c ← a + b
b ← a + b
a ← c

4-
a ← 5
b ← 7
a ← b
b ← a

SERIE TD N°: 2

LES STRUCTURES CONDITIONNELLES

Exercice 1 :

A, B et C étant des variables numériques, on considère les deux séquences algorithmiques S1 et S2 :

```
Début
 Lire (A,B,C)
 Si ( A > B ) et ( A< C ou B < C)
    Alors
      A ← A + B
       Si A < C
            Alors
                  C ← C – A
            Sinon
                  C ← C + A
                  B ← C - B
       Finsi
    Finsi
        Ecrire A, B, C
Fin
```

```
but
re (A,B,C)
 ( A < C et  B > 5 ) ou C > 8
 Alors
    A ← B + C
    B ← A
 Sinon
    Si ( A > C ) et ( B > A )
          Alors
                B ← B + A
                C ← C + B - A
          Sinon
                B ← C + A
                A ← A + B
    Finsi
 Fin i
      Ecrire A, B, C
Fin
```

Pour chacune de ces deux séquences, donner les valeurs après exécution de A, B et C dans les 3 cas suivants :

A	B	C
8	5	7
5	12	8
15	16	17

Exercice 2 :

Etant donné un entier NB positif ou nul, donner l'algorithme qui permet de déterminer la parité de NB.

Exercice 3 :

Soit A et B deux variables numériques. Ecrire un algorithme qui détermine le plus petit entre A et B.

Même exercice avec 3 variables A, B et C.

Exercice 4 :

Résolution d'une équation du 1er degré de type ax+b=0 avec a et b quelconques.

Exercice 5 :

Ecrire un algorithme de résolution d'une équation second degré où les paramètres de l'équation $\in$ R.

Exercice 6 :

Ecrire un algorithme permettant de déterminer l'admission d'un candidat à l'entrée dans une faculté de médecine en fonction des résultats obtenus au baccalauréat :

Bac math et moyenne $\geq$ 12
Ou Bac Sciences et moyenne $\geq$ 14
Ou Bac Lettres et moyenne $\geq$ 15

Exercice 7:

Ecrire un algorithme qui détermine la date du lendemain (on suppose que la date introduite est contrôlée)

Exemple :

Date introduite : 12/05/2010
Date de lendemain : 13/05/2010

Exercice 8 :

Ecrire un algorithme qui lit un nombre de 3 chiffres et vérifie s'il est égal à la somme des cubes des chiffres qui le composent.

Exemple : **$153 = (1)^3 + (5)^3 + (3)^3$**

SERIE TD N°: 3

LES STRUCTURES ITERATIVES

Exercice 1 :

Ecrire un algorithme qui permet de lire un entier positif compris entre 1 et 12 et de calculer son factoriel.

Exercice 2 :

Ecrire un algorithme qui permet de vérifier si un nombre positif est premier.

Exercice 3 :

Ecrire un algorithme qui permet de déterminer tous les nombres premiers entre 5 et 100.

Exercice 4 :

Ecrire un algorithme qui calcule le prix d'un appartement de n pièces (**n<=4**) connaissant les dimensions de chacune des pièces et le prix au mètre carré.

Exercice 5:

Un nombre est dit parfait s'il est égal à la somme de ses diviseurs sauf par lui-même

Exemple : 6 = 1+2+3

Ecrire un algorithme qui lit un nombre et vérifie s'il est parfait.

Exercice 6 :

Ecrire un algorithme qui permet de lire un entier strictement positif et d'afficher ses diviseurs.

Exercice 7 :

Soit la suite U_n définie par :

$U_0 = 1/5$

$U_n = U_{n-1} + 1/U_{n-1}$

Ecrire un algorithme qui permet de calculer le $23^{ème}$ élément de la suite et la somme des 23 premiers éléments de cette suite.

Exercice 8 :

Ecrire un algorithme qui permet de lire un nombre NB positif et de déterminer tous ses facteurs premiers.
Exemple : NB = 30 = 2 x 3 x 5

Exercice 9 :

Ecrire un algorithme qui permet de compter le nombre de zéro qui se trouvent dans les nombres entre 1 et 101.

SERIE TD N°: 4

LES CHAINES DE CARACTERES

Exercice 1 :

Ecrire un algorithme qui permet d'inverser une chaîne de caractères quelconque.

Exemple : tunisia donne aisinut

Exercice 2 :

Ecrire un algorithme qui vérifie si une chaîne de caractères est palindrome. Une chaîne est dite palindrome si elle se lit de gauche à droite comme elle se lit de droite a gauche(elle, radar,…)

Exercice 3 :

Ecrire un algorithme qui permet de vérifier l'existence d'un caractère donné dans une chaîne de caractères.

Exercice 4 :

Ecrire un algorithme qui permet de lire une chaîne de caractères **CH** et un caractère **C** et de calculer le nombre d'occurrence de **C** dans **CH**.

Exercice 5 :

Ecrire un algorithme qui permet de :

- Saisir une chaîne de caractère CH (caractère par caractère) ne contenant que des caractères alphabétiques majuscules se terminant par le caractère '#'.
- Eliminer l'apparition du caractère 'A' dans la chaîne saisie et afficher le résultat.
- Eclater la chaîne résultante en deux sous-chaînes CH1 et CH2 : la première contient uniquement les lettres de 'B' et 'M', et la deuxième contient uniquement les lettres de 'N' à 'Z'.

Exemple : CH = 'BZACMDXAA#' après élimination du caractère 'A', on aura CH ='BZCMDX#'
Après éclatement on aura CH1 = 'BCMD' et CH2 = 'ZX'

SERIE TD N°: 5

LES TABLEAUX

Exercice 1 :

Soit T un tableau de dimension N :

Ecrire un algorithme qui permet le chargement de T avec des nombres quelconques, de déterminer le nombre des éléments positifs, négatifs et nuls.

Exercice 2 :

Ecrire un algorithme qui range par ordre décroissant dans un tableau B les carrés des 10 premiers entiers naturels.

Exercice 3 :

Ecrire un algorithme qui détermine le nombre d'occurrences d'une valeur V dans un tableau de N éléments.

Exercice 4 :

Soit un tableau VT de dimension N. Ecrire un algorithme qui vérifie si VT est symétrique.

Exercice 5 :

On considère une classe de 30 élèves. Chaque élève suit 7 cours qui n'ont pas les mêmes coefficients.

Ecrire un algorithme qui permet de saisir toutes les notes (obtenues par les élèves dans les cours) dans un tableau **NOTE** et de calculer la moyenne de chaque élève dans un tableau **MOY**. Déterminer la liste des admis (un élève est admis si sa moyenne est ≥ 10).

Exercice 6 :

Soit un tableau TAB possédant N éléments. Ranger dans le même tableau les éléments dans l'ordre inverse.

Exemple :

TAB	1	2	3	0	4	-1

Sera transformé in TAB	-1	4	0	3	2	1

Exercice 7 :

Transférer les N éléments d'un tableau **T1** dans un tableau **T2** selon le principe suivant :

- Les éléments du rang impaire de T1 sont rangés dans T2 en ordre inverse en commençant par la fin de T2.
- Les éléments de rang paires de T1 sont rangés dans T2 dans le même ordre.

Exemple :

T1	-1	2	7	1	3	1	-2

T2	2	1	1	-2	3	7	-1

Exercice 8:

Ecrire un algorithme **Matrice_ordonnée** qui permet de charger une matrice de **L** lignes et **C** colonnes avec des entiers ordonnés dans l'ordre croissant.

Exemple :

4	15	30	30
55	61	70	82
82	90	110	120

Exercice 9 :

Soit **MAT** une matrice carrée d'ordre N. Ecrire un algorithme qui vérifie si MAT est symétrique.

Exercice 10 :

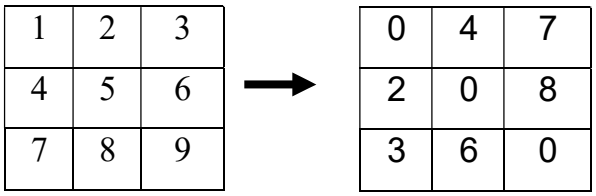
Soit une matrice **MAT** ayant **N** lignes et **N** colonnes

1) Mettre à zéro la diagonale principale de **MAT**

NB : la diagonale principale d'une matrice carrée est l'ensemble des éléments du tableau dont les indices de lignes et de colonnes sont égaux.

2) Faire une inversion des lignes et des colonnes de MAT et ce en effectuant une symétrie par rapport à la diagonale principale.

Exemple :



SERIE TD N°: 6

LES ENREGISTREMENTS

Exercice 1 :

Une date se compose de trois entités bien connues : le jour, le mois, et l'année (Exemple : 15/01/2010). Définissez les types Jour, mois, Année et Date.

Exercice 2:

Un étudiant est décrit par son numéro de carte d'étudiant, son nom, son prénom, sa date et son lieu de naissance, son adresse (à décomposer : Rue, Ville, Code postal, Pays) et son groupe. Définissez le type Etudiant.

Exercice 3 :

Décrire la structure de données relatives à une personne. Cette structure doit contenir les informations suivantes :

- Le nom, et le prénom
- La nature du sexe : (masculin, féminin)
- La date de naissance,
- Les dates de naissances des parents,
- L'état civil de la personne : (marié, célibataire)
- Si la personne est mariée nous devons savoir :
  - Le nom et prénom de son conjoint.
  - Le nombre d'enfants (on peut avoir au maximum 15 enfants)
  - Pour chaque enfant, nous devons reconnaître la date de naissance, le sexe et le prénom.
- Si la personne est célibataire nous devons savoir la situation militaire.

Exercice 4:

Ecrire un algorithme «Remplir» qui remplit un tableau T par des étudiants inscrit à l'ISSAT de SOUSSE.

Ces caractéristiques sont le numéro d'inscription, le nom, le prénom, l'adresse, la date de naissance, l'état civil et le niveau.
La capacité maximale de T étant de 1200 étudiants.

Exercice 5 :

Ecrire un algorithme «Recherche» qui recherche, dans le tableau T, un étudiant connu par son nom et son prénom (deux variables lues).
Si cet étudiant existe, l'algorithme doit afficher toutes ses caractéristiques, sinon, il affiche le message «Cet étudiant n'est pas inscrit»

SERIE TD N°: 7

LES SOUS-ALGORITHMES

Exercice 1 :

Ecrire une fonction **Existe** permettant de vérifier l'existence d'un caractère donné dans un chaîne de caractère.

Exercice 2 :

Ecrire une fonction **Pos** qui permet de chercher la première occurrence d'un caractère donné **C** dans une chaîne de caractère **CH**.

Exercice 3 :

On suppose que la comparaison de deux chaînes est interdite, alors que la comparaison de deux caractères est permise. Ecrire une procédure **Comp** permettant d'afficher le résultat de la comparaison de deux chaînes de caractère CH1 et CH2.

Exercice 4 :

Ecrire une procédure **Eclater** qui permet d'éclater une chaîne de caractère en deux sous chaînes :

- Une contenant les voyelles.
- Une contenant les consonnes.

Exercice 5 :

Ecrire :

- La fonction **NB_ZERO(NB)** qui compte le nombre de zéros dans un nombre entier. ***Exemple :*** 10001 contient 3 zéros.
- La procédure **TOT_ZERO(V,N)** qui détermine le total des zéros existant dans les N nombres du vecteur V.
- La procédure **SAISI_TAB(V,N)** qui permet la saisie de N nombres entiers distincts dans un vecteur V.

Exercice 6 :

Transférer les éléments nuls d'un tableau **T1** à la fin de **T2** tout en effectuant le décalage des éléments non nuls.

Exemple :

T1	4	0	0	2	0	1	-40
T2	4	2	1	-40	0	0	0

1. Ecrire une procédure **SAISIE** permettant de saisir une valeur N comprise entre 2 et 100.
2. Ecrire une procédure **LECT** permettant de remplir le tableau V avec N éléments entiers.
3. Ecrire une procédure **Décalage** qui fait le traitement demandé.
4. Ecrire une procédure **AFFICHE** qui affiche le contenu d'un tableau donné.
5. Ecrire une procédure **AFFICHE_TOUT** permettant d'afficher T1 et T2.

En utilisant les procédures précédentes. Ecrire l'algorithme principal correspondant.

Exercice 7 :

On désire faire la saisie d'une chaîne de caractères ne contenant que des caractères alphabétiques majuscules et se terminant par un caractère donné par l'utilisateur, remplacer toutes les occurrences d'un caractère X par un autre caractère Y donné par l'utilisateur et indiquer si les caractères figurent dans l'ordre croissant ou non dans la chaîne résultante.

Exemple : CH = 'ABC#' $\Rightarrow$ ses élément figurent dans l'ordre croissant.

CH = 'ABDC#' $\Rightarrow$ ses élément ne figurent pas dans l'ordre croissant.

Avec : '#' est un exemple de caractère de terminaison de la chaîne saisie, donné par l'utilisateur.

Pour ce faire, vous êtes demandés d'écrire :

- Une fonction qui permet de lire une chaîne de caractère CH ne contenant que des caractères alphabétiques majuscules et se terminant par un caractère donné par l'utilisateur.
- Une procédure qui permet de remplacer toutes les occurrences d'un caractère X donné, par un autre caractère Y donné, dans une chaîne CH.
- Une fonction qui permet de vérifier si les caractères d'une chaîne CH figurent dans l'ordre croissant ou non. Cette fonction doit faire retourner la valeur vrai dans le cas où les caractères figurent dans l'ordre croissant et faux dans le cas contraire.
- L'algorithme principal correspondant.

Exercice 8 :

Soit **MAT** une matrice de M lignes et N colonnes contenant des valeurs numériques quelconques.

On appelle le point colonne MAT(I,J) l'élément de la matrice qui est maximum sur sa ligne et minimum sur sa colonne.

Exemple :

MAT :

5	18	6	10
5	**17**	4	15
4	20	18	18
9	19	6	6

Cette matrice admet un point colonne : MAT (2,2).

On désire écrire un algorithme qui détermine les points de la matrice MAT.

Le principe de la solution consiste à parcourir la matrice ligne par ligne et utiliser deux procédures. L'une renvoie le numéro de colonne du maximum de la ligne d'une matrice et l'autre qui cherche le numéro de ligne du minimum de la colonne renvoyée par la première procédure.

On vous demande de :

- Ecrire une procédure **MAXLIG** qui renvoie le numéro de colonne du maximum de la ligne d'une matrice ;
- Ecrire une procédure **MINCOL** qui cherche le numéro de ligne du minimum de la colonne renvoyée par la première procédure ;
- L'algorithme principal **COL** permettant de chercher les points colonnes d'une matrice.

BIBLIOGRAPHIE

📖 **Titre** : Initiation à l'algorithmique et aux structures de données 1
Maison d'édition : BORDAS
Auteur : Jacques Courtin, Irène Kowarski, Jacques Arsac
Année : 1987

📖 **Titre** : Initiation à l'algorithmique et aux structures de données 2
Maison d'édition : BORDAS
Auteur : Jacques Courtin, Irène Kowarski, Jacques Arsac
Année : 1987

📖 **Titre** : Algorithmes : cours et exercices
Maison d'édition : BREAL
Auteur : Sylvie Tormento et Sophie Boutin
Année : 1994

📖 **Titre** : Algorithmique et structures de données
Maison d'édition : Abeille
Auteur : S. Graïne
Année : 2001