ELEMENTS
D'
ALGORITHMIQUE

R. Florentin

A - GENERALITES

1°) Introduction

Exemple 1.
Considérons le problème suivant à résoudre :

Le pneu d'une voiture est crevé en route. Elle dispose d'une roue de secours dans son coffre, et de tout ce qu'il faut pour effectuer un changement de roue. Aidez le conducteur de cette voiture à effectuer ce remplacement en lui indiquant la démarche à suivre.

Une démarche proposée:

- Enlever la roue crevée,
- sortir du coffre la roue de secours,
- placer la roue de secours à la place de la roue crevée,
- remettre la roue crevée dans le coffre,
- Continuer sa route en pensant s'arrêter à la prochaine station pour réparation.

Cette démarche est constituée d'une succession de tâches, d'actions à réaliser.
Elle est faite utilisant le langage usuel. Les mots sont les mots du vocabulaire français, compris par tout le monde. Elle décrit les étapes à suivre, et l'ordre d'exécution de ces taches.
Mais ces étapes sont décrites sans grande précision. Par exemple, l'étape « enlever la roue crevée » est vague, et n'indique pas comment il faut procéder.
Pour faire exécuter ces taches par quelqu'un à qui on doit tout dire, il faut être plus explicite, précis. Il faut utiliser des mots sans ambiguïté, sans possibilité d'interprétation. Il faut une description de taches qui puisse être comprise par n'importe se trouvant dans cette situation, avec n'importe quelle voiture de tourisme.
Il faut écrire un algorithme.

Exemple 2.
Considérons une liste de 10 entiers quelconques que l'on désire ranger.
Par exemple : { 14 ; 2 ; 5 ; -7 ; 10 ; 7 ; -7 ; 3 ; 1 ; 15 }
Trouver une démarche qui permette de ranger cette liste.

Il ne s'agit pas de réussir à ranger cette liste particulière, il s'agit de proposer une démarche générale, qui range cette liste , mais surtout qui permette de ranger une liste quelconque composée d'un nombre fini d'éléments de « même nature ».

2°) Définitions

Algorithme : Un algorithme est une description d'une succession d'actions (instructions) qui, une fois exécutées correctement, conduit à un résultat donné.

Remarques :

- ✓ Un algorithme est **correct** si le résultat obtenu après son application est toujours celui escompté.
- ✓ Un algorithme est écrit pour être traduit dans un langage de programmation donné. Le code ainsi obtenu sera exécuté par un ordinateur.
- ✓ Un algorithme :
  - fait passer d'un état initial d'un problème à un état final de celui-ci, de façon déterministe.
  - doit utiliser des données connues de l'utilisateur,
  - doit contenir un nombre fini d'actions exécutables,
  - doit être défini sans ambiguïté, et les objets qu'il manipule doivent être définis de manière très précise.

- doit avoir toutes ses opérations qui peuvent être exécutées par un homme avec des moyens manuels.
- doit être indépendant de tout langage de programmation.

A tout cela s'ajoutent des qualités qui sont généralement demandées à tout programme informatique :

- une rapidité d'exécution (le temps de calcul est souvent facturé et très cher),
- une occupation de l'espace mémoire satisfaisante.

Langage :

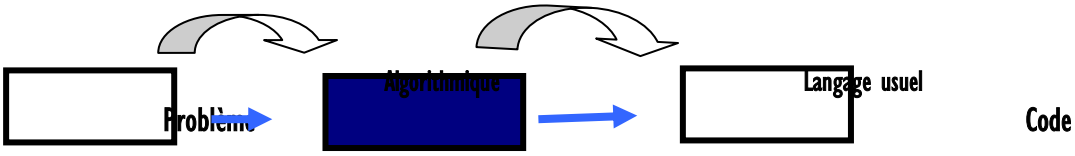
Quand deux entités veulent communiquer, pour se comprendre, elles doivent adopter un langage commun de communication. Un algorithme (on dit aussi programme) décrit les étapes adoptées pour résoudre un problème donné en utilisant une machine automatique virtuelle qu'est l'homme. Il précise des instructions qui peuvent être exécutées pour la résolution de ce problème. Un algorithme donne des instructions.

Un algorithme utilise des mots d'un vocabulaire, celui du langage appelé **algorithmique**. Ce vocabulaire est issu de celui du langage commun sur lequel on a imposé des restrictions.

Pseudo-code :

Un **code** est un programme écrit dans un langage de programmation donné. Il est destiné à être exécuté par un ordinateur. On peut citer Pascal, Fortran, C, C++, Java etc…..

Un **pseudo code** est un programme écrit en algorithmique. Il est plus souple et n'est pas soumis aux rigueurs strictes des langages de programmation (rigueur du vocabulaire , de la syntaxe, de la grammaire).



Objet :

En algorithmique on manipule des objets.

Exemples : Dans l'exemple 1, on manipule des roues, des clés, un cric, une cale. Ce sont des objets.
Dans l'exemple 2, la liste à ranger est un objet, de même que la taille de cette liste (un entier).

Un objet peut être :

- un objet nécessaire avant l'exécution du programme: ce sont des **données**. La roue de secours, la roue crevée, le cric, etc… sont des données. La liste de l'exemple 2 est une donnée.
- un objet **résultat**. Par exemple, la liste, rangée, de l'exemple 2, est un résultat.

Exercice1

Donner une succession d'actions à mener pour résoudre le problème posé dans l'exemple 2.

3°) Structure d'un algorithme

Un algorithme utilise des données généralement pour le traitement qu'il propose, et sort un ou des résultats.

Un algorithme commence toujours par un mot clé **PROGRAM**, pour qu'on puisse facilement le reconnaître, suivi d'un nom pour que l'on sache à peu près ce qu'il est censé faire.
Puis vient la partie **DECLARATION**, où on spécifie les caractéristiques de tous les objets manipulés selon des règles bien précises. La description elle même des étapes à suivre pour la résolution du problème vient après, et constitue ce qu'on appelle le **CORPS de l'algorithme.** Il commence toujours par le mot clé **DEBUT** et se termine par le mot clé **FIN.**

Illustration :

```
PROGRAM <nom de l'algo >
    (* Début de la partie déclaration : cette ligne elle-même est une ligne de
 commentaires *)
     .........
    (* Fin de la partie déclaration *)
    (* Début du corps de l'algorithme *)
DEBUT
.........
.........
FIN
```

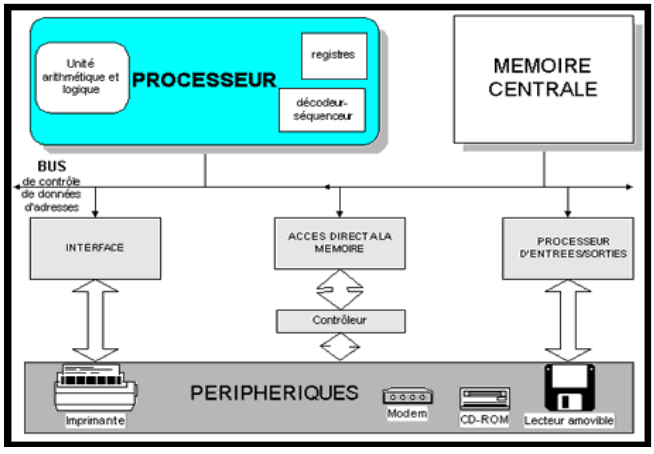
Remarque : Convention d'écriture : Par **PROGRAM** <nom de l'algo > , il faut comprendre qu'il faut mettre à la place de <.....> le nom du programme . Exemple : **PROGRAM** calc_moyenne

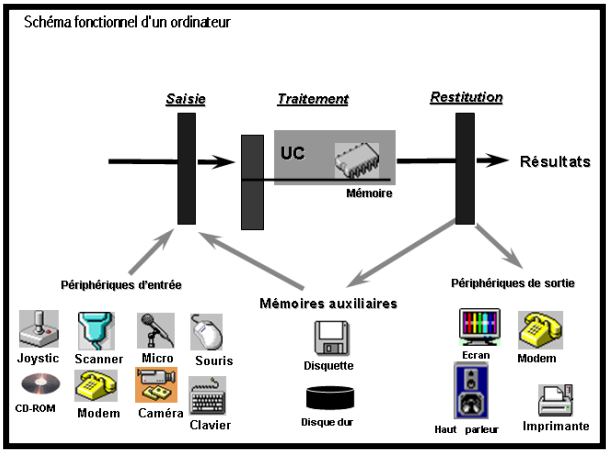
4°) Architecture

Un algorithme est destiné à être traduit dans un langage donné, puis exécuté par un ordinateur. Il travaille avec des données et sort des résultats. Ces données auront à être stockées en mémoire pour qu'on puisse les (ré)utiliser.
D'où la nécessité de bien comprendre l'architecture d'un ordinateur.

Un ordinateur est une machine qui **saisit** (périphériques d'entrée) , **stocke** (mémoire) , **traite** (programme) des informations (données) et **restitue** (périphériques de sortie) des informations (résultats).

Architecture de base actuelle et Schéma fonctionnel d'un ordinateur





que l'ordinateur utilise pour coder les informations). Un ordinateur [illegible] Les plus courants actuellement sont des ordinateurs de 32 bits.
La mémoire (RAM) d'un ordinateur peut être vue comme un tableau à une dimension (vecteur). Chaque case de ce tableau est un mot. Chaque mot est repéré par son adresse qui indique l'emplacement physique de l'information.

Pour un algo, le mot (ou le groupe de mots nécessaires) est connu par son nom. Alors qu'avec un langage usuel, un mot est caractérisé par son nom et son adresse indiquant son emplacement exact dans la mémoire.
Les données peuvent être sauvegardées dans des mémoires de masse (ou secondaires) (Disque dur , Clé USB, CD , etc …). Ces mémoires de masse sont découpées en fichiers, eux-mêmes regroupés au sein de répertoires selon une structure arborescente. Il s'agit du **système de fichiers** dont la gestion revient au **système d'exploitation**
A un système d'exploitation correspond un système de fichiers bien déterminé.

Les informations sont codées. Plusieurs codages sont utilisés dont, par exemple :

- le codage binaire qui est une représentation correcte de l'environnement réel de travail du processeur.
- Le codage octal (en base 8)
- Le codage hexadécimal (en base 16)
- Le codage ASCII
- ……..

5°) Les objets manipulés par l'algorithme

a) Caractéristiques d'un objet

Un objet manipulé par un algorithme doit être parfaitement défini avant son utilisation. Il est parfaitement défini si on connaît :

- son identificateur, qui est le nom qu'on lui donne : une suite de caractères alphanumériques sans espace et commençant obligatoirement par une lettre ou _ (underscore). De préférence ce nom est choisi en rapport avec le contenu de l'objet.

  Exemples : moyenne ; _prix ; m876 sont des identificateurs valables.
  7prix n'est pas un identificateur valable.

- La nature de son contenu : elle est soit **CONSTANTE** soit **VARIABLE**.
  VARIABLE si on permet au contenu de changer au cours du programme.
  CONSTANTE si non.

- son type :
  Le type définit la nature du contenu d'une variable. Il est défini par l'ensemble de ses constantes et par l'ensemble des opérations que l'on peut appliquer à ces constantes.

  Précision : Tous les mots clés du vocabulaire du langage algorithmique seront écrits dans ce **FORMAT .**
  Il est interdit de prendre un mot clé comme identificateur d'une variable ou d'une constante.

b) Les types usuels

BOOLEEN :
Ensemble des constantes : { .**VRAI**. ; .**FAUX**. }
Opérateurs : **ET** ; **OU** ; **NON**

Numérique (**ENTIER** ou **REEL**)
Ensemble des constantes : Z pour **ENTIER** ou D (abusivement appelé IR) pour **REEL**
Ensemble des opérateurs : + (addition) ; * (multiplication) ; - (soustraction) ; / (division réelle) ; ^ (exponentiation); **DIV** (division entière) ; **MOD** (reste d'une division);
ENT (partie entière)
On peut distinguer les entiers simples (précision) , les entiers longs , les réels simples (précision) , les réels double précision . Le choix dépend de l'ordre de grandeur de la valeur de la variable ou de la constante.

Exemple: une constante entière : 90876 ;
une constante réelle : 6.9876 ; 7.08 E+4

Remarque :
Quand on programme dans un véritable langage de programmation, il faut aussi se soucier des ordres de grandeurs des variables numériques utilisées, à l'intervalle d'appartenance des valeurs de ces variables.
Ces ordres de grandeurs se mesurent en octets.
Par exemple, le type **ENTIER** (**integer** en pascal) utilise 2 octets, donc concerne les entiers relatifs appartenant à l'intervalle [-32768 ; 32767]. Le type **REEL** (**real** en pascal) occupe 6 octets.
Etc…

CARACTERE ou CHAINE DE CARACTERE

CARACTERE :
Ensemble des constantes : l'ensemble des lettres de l'alphabet (majuscule , minuscule) , des différents codes d'opérations, de ponctuations et d'autres codes tels $, & ; …. . En fait tout ce qui est répertorié dans le tableau des codes ASCII.

CHAINE DE CARACTERE :
Ensemble des constantes : Une chaîne de caractères est soit une chaîne vide soit un caractère suivi éventuellement d'autres caractères.
Ensemble des opérateurs:
SSCHAINE (<chaîne>, <pos> , <nb de caractères >) : extrait une partie d'une chaîne de caractères;
Concaténation : <chaine1>//<chaine2> donne une nouvelle chaîne contenant les deux chaînes juxtaposées.
LONGUEUR(<chaîne>) : renvoie la longueur d'une chaîne de caractère ;
On peut aussi citer : **RANG** (); **CODE**() ; **CAR**(); **CVCHAINE**() ; **CVNOMBRE**()….

Exemples :
Ecriture d'une constante **CARACTERE** : 'm' , ' ?' , '*', ''.
Ecriture d'une constante **CHAINE DE CARACTERE** : "voiture"
LONGUEUR("voiture") =7 ;
"ma "// "maman" = "ma maman" ;
SSCHAINE ("maman", 2,3)="ama"

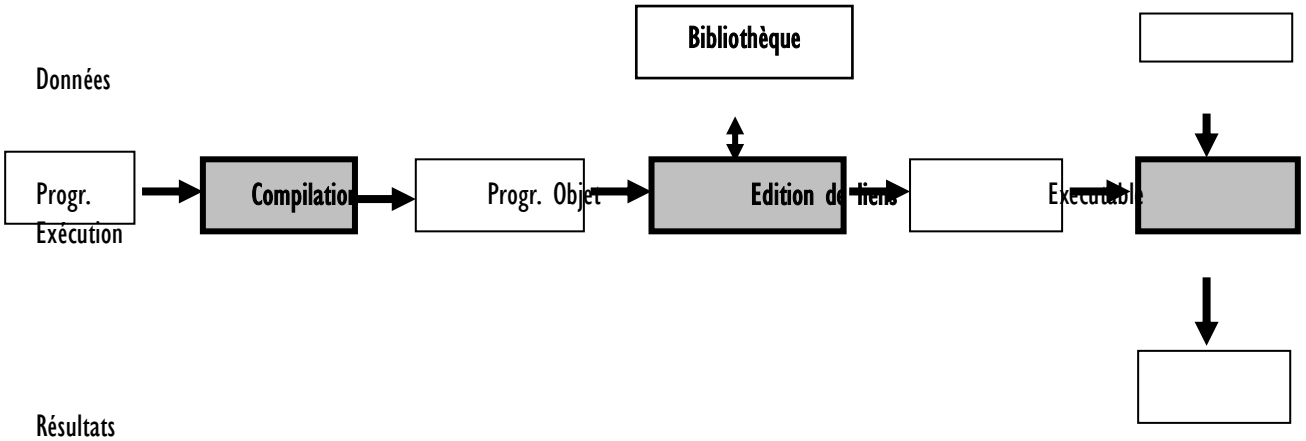
Attention : 'a' est de type **CARACTERE** alors que "a" est de type **CHAINE DE CARACTERE** .
'a'//'b' n'a pas de sens alors que "a"//"b" donne "ab".

6°) Traitement subi par un programme

L'algorithmique est un véritable langage de programmation, avec ceci en moins : il utilise un langage qui est très souple dans la syntaxe. Et il ne se soucie pas de la manière avec laquelle il sera traité : compilé ou interprété, par exemple.
La conception d'un algorithme doit donc être faite toujours dans le souci de le voir tout juste après traduit dans un langage donné.

Un programme, une fois écrit dans un langage compilé par exemple, subit :

- la compilation, pour la vérification du vocabulaire et de la syntaxe utilisés. Il en sort un programme objet (***.obj)
- puis l'édition des liens, qui génère un programme exécutable (***.exe)
- l'exécution, pour la résolution effective du problème.



7°) Retour sur la structure d'un algorithme

On a vu qu'un algorithme doit commencer par le mot clé **PROGRAM** suivi d'un nom . Puis après vient éventuellement la partie déclarative, pour se terminer par le corps du programme encadré par un **DEBUT** et un **FIN**.

La partie déclarative est l'endroit de l'algorithme où on précise tous les objets qui sont utilisés dans l'algorithme :

- identificateur et type pour les objets de nature **VARIABLE**,
- identificateur , type et valeur pour les objets de nature **CONSTANTE**.

Règle : On déclare les **CONSTANTE**s d'abord, s'il y en a , puis toutes les **VARIABLE**s après.
Une variable non déclarée provoque une erreur de compilation.

Remarque : Pour les **CONSTANTE**s, le type est défini par la valeur elle-même, comme précisé dans l'exemple ci-dessous.

Exemple

```
PROGRAM essai01   (* en-tête du programme *)
(* Début déclaration *)
    CONSTANTE n=25 , pi=3.14 ( * n est de type ENTIER ;  pi de type REEL *)
  CONSTANTE z=9.8767654356678765  ( * z est de type REEL DBLE PRECISION *)
 VARIABLE : i, j, k : ENTIER   (* i est un compteur ,  a contient la moyenne ……. *)
            factoriel:  ENTIER LONG
                    a, b, c : REEL
            f, u :  REEL DBLE PRECISION
            reponse : BOOLEEN
```

```
(* fin de la déclaration *)
(* début du corps du programme *)
DEBUT
        ……..
        ……..
FIN
```

Remarque : Les commentaires sont indispensables pour une bonne lecture d'un programme. Il faut utiliser des commentaires quand on écrit un algorithme : pour soi même d'abord, puis pour les autres qui auront à lire, et à comprendre, votre programme.

B – LE CORPS DE L'ALGORITHME

B-1 : LES INSTRUCTIONS DE BASE

C'est dans le corps de l'algorithme que l'on décrit la succession des taches à réaliser pour résoudre le problème donné, à l'aide d'instructions.

Ces instructions peuvent être regroupées dans quatre (4) grandes familles, les seules qu'un ordinateur est capable de traiter :

- **l'affectation des variables**
- **la lecture / l'écriture**
- **les tests**
- **les boucles.**

Ces 4 grandes familles constituent les instructions de base d'un algorithme.

1°) **Ordre d'exécution des instructions**
L'ordre d'exécution des instructions est essentiel. Elles sont exécutées **de manière séquentielle**.

2°) **Instruction d'affectation**
Déclarer une variable (ou une constante) c'est réserver une boite en mémoire destinée à contenir une valeur de la variable (ou la valeur de cette constante).

En algorithmique, pour une variable, la place est juste réservée lors de la déclaration.

Affecter une valeur à cette variable c'est décider de lui donner cette valeur. Omettre de déclarer une variable pose donc problème car lors d'une affectation, son emplacement physique est introuvable car non réservé.

En algorithmique, l'instruction d'affectation se note ← :

Exemple : moyenne ← 12.5 (le contenu de la variable moyenne est 12.5 : cette variable a été déclarée **REEL**)
Une variable peut recevoir :

- une valeur d'une constante, nom ← "marie" ou note ← 2
- une valeur d'une expression, moyenne ← (x+4+z)/3 : ***on calcule d'abord l'expression*** (x+4+z)/3 ***, le résultat est affecté à moyenne.***

- une valeur d'une autre variable moyenne ← aux : *le contenu de aux est mis dans moyenne. A la fin de l'instruction, moyenne et aux ont le même contenu.*

Remarque :

- ✓ Une instruction du type **4←aux** où aux est une variable, n'est pas correcte.
- ✓ Lors d'une affectation, à gauche on trouve toujours une variable.
- ✓ Il faut que la variable qui reçoit une valeur soit du même type que la valeur qu'elle reçoit. Sinon, en général, il y a erreur de compilation, ou alors c'est celle qui reçoit qui impose son type.

Exercice 2

Que contient a à la fin des algorithmes suivants ?

```
PROGRAM valeur1
VARIABLE a : ENTIER
DEBUT
      a ← 34
      a ← 12
FIN
```

```
PROGRAM valeur2
VARIABLE a : ENTIER
DEBUT
      a ← 12
      a ← 34
FIN
```

```
PROGRAM valeur3
VARIABLE a : REEL
DEBUT
a←2
a←a+a*3+2*a^3
FIN
```

Réponse

a=12	a=34	a=24

Exercice 3

Ecrire un algorithme permettant de calculer le nombre de caractères d'un mot donné.

Précision :

La présentation d'un algorithme, destiné à résoudre un problème donné, comprend toujours trois parties :

- ✓ l'analyse du problème, où on précise comment le problème a été compris et de quelle manière globalement il sera résolu
- ✓ la spécification des variables et constantes utilisées dans le programme, pour une meilleure compréhension de la résolution
- ✓ l'algorithme lui même

Une solution

Analyse :

L'opérateur **LONGUEUR ( )** a comme paramètre une chaine de caractères, et renvoie la longueur de cette chaîne de caractères, qui n'est autre que le nombre de caractères qui forment cette chaîne. Il s'agit donc ici d'utiliser directement cet opérateur.

Une variable de type **CHAINE DE CARACTERE** sera utilisée pour recevoir la chaine dont on veut calculer la longueur (objet donnée) et une variable de type **ENTIER** est utilisée pour recevoir le résultat (objet résultat),

Variables :

mot : variable chaine de caractères (dont on veut déterminer le nombre de caractères). C'est une donnée.

nbcaract : variable de type ENTIER contenant le nombre de caractères de la chaîne. C'est un résultat.

Algorithme

```
PROGRAM longmot0
VARIABLE  nbcaract : ENTIER
          mot : CHAINE DE CARACTERE
DEBUT
    mot← "INFORMATIQUE"
    nbcaract←LONGUEUR(mot)
FIN
```

Remarques

- ✓ mot pouvait être déclaré comme constante chaîne de caractères
- ✓ le résultat de l'algorithme est inaccessible à l'utilisateur. Le travail est effectué mais le résultat n'est pas restitué.

3°) **Instructions d'Entrée/Sortie (E/S)**

Ce qui manque au programme précédent ce sont les instructions d'entrée/sortie.

Une **instruction d'entrée** est une instruction permettant de saisir des données, utilisant une périphérique d'entrée comme le clavier.

Syntaxe : **SAISIR** (< identificateur >) ou **LIRE** (< identificateur >)

Une **instruction de sortie** est une instruction permettant de restituer un résultat, utilisant une périphérique de sortie : l'écran par exemple, ou l'imprimante, ou une disquette etc…. ;

Syntaxe : **AFFICHER** (< identificateur >) ou **ECRIRE** (< identificateur >)

Reprenons l'exemple ci-dessus :

```
PROGRAM longmot1
VARIABLE  nbcaract : ENTIER
          mot : CHAINE DE CARACTERE
DEBUT
    AFFICHER ( "donner le mot : " ) (* instruction d'entrée demandant une chaîne à
    l'utilisateur *)
    LIRE ( mot )   (* là on saisira INFORMATIQUE *)
    nbcaract←LONGUEUR(mot)
    AFFICHER ( "longueur du mot  ", mot, " est :  ", nbcaract ) (* instruction de
    sortie affichant le résultat *)
FIN
```

Si on lance ce programme, voici ce qui serait affiché à l'écran :

donner le mot :
INFORMATIQUE ↵ (↵ désigne la touche « entrée » ou « valider » ou « return » du clavier)

longueur du mot INFORMATIQUE est : 12

Remarque :
L'on pourrait s'intéresser à la présentation du résultat une fois calculé. C'est ce qu'on appelle le format de sortie. On peut aussi s'intéresser au format de saisie des données (format d'entrée). Mais en algorithmique, on ne se soucie pas trop des formats (en entrée ou en sortie), pour la simple raison que les formatages sont propres à chaque langage.

Exercice 4
Ecrire un algorithme qui saisit deux variables de type **CHAINE DE CARACTERE**, qui affiche le contenu de chaque variable, qui échange les contenus, et affiche à nouveau les contenus.

Solution proposée :
Analyse : tout est dit dans l'énoncé
Variables utilisées :

- deux variables nom1 et nom2 de type **CHAINE DE CARACTERE** . Ce sont des données mais aussi des résultats.
- une variable aux de type **CHAINE DE CARACTERE** : variable de travail qui servira lors des échanges des contenus.

```
PROGRAM echange
VARIABLE nom1, nom2 , aux: CHAINE DE CARACTERE
DEBUT
        (* saisie des données *)
        AFFICHER("Donner le mot a mettre dans la premiere boite")
        LIRE(nom1)
        AFFICHER("Donner le mot a mettre dans la seconde boite")
        LIRE(nom2)
        AFFICHER("dans boite N°1 il y a le mot : ", nom1, " et dans la seconde, le mot: ", nom2)
        (* procédé d'échange des contenus *)
        aux←nom1
        nom1←nom2
        nom2←aux
        (* affichage des nouveaux contenus *)
        AFFICHER("dans boite N°1 il y a le mot : ", nom1, " et dans la seconde, le mot: ", nom2)
FIN
```

Remarque :
Lorsqu'on connaît le nombre maximal de caractères d'une variable de type **CHAINE DE CARACTERE**, on pourra le préciser entre crochets lors de la déclaration.

Exemple
VARIABLE nom1, nom2 , aux: **CHAINE DE CARACTERE[10]** déclare comme des variables chaîne de 10 caractères au maximum.

4°) Les tests : Structures conditionnelles.

On distingue :

a) **La structure conditionnelle.**

Syntaxe :

```
< instruction 0 >
SI < condition > ALORS  < instruction1 >
FSI
<instruction2>
```

Si la condition est vérifiée, instruction1 est exécutée, et après on exécute instruction2. Dans le cas contraire, c'est instruction2 qui est traitée, instruction1 n'est pas traitée.

Remarque :

Si au lieu de instruction1 (unique) on a à réaliser un groupe d'instructions (instruction multiple), on doit encadrer ce groupe d'instructions avec un **DEBUT ... FIN.**

Exemple :

Reprenons l'exemple longmot1.

On procédera à l'affichage de nbcaract seulement dans le cas où ce nombre est supérieur à 1. L'algorithme devient :

```
PROGRAM longmot2
VARIABLE  nbcaract : ENTIER  (* nombre de caractères du mot *)
                   mot : CHAINE DE CARACTERE[20]
      DEBUT
      AFFICHER ( "donner le mot : " )
      LIRE ( mot )   (* là on saisira « INFORMATIQUE », par exemple  *)
      nbcaract←LONGUEUR(mot)
      SI ( nbcaract > 1 ) ALORS  AFFICHER ("longueur du mot  ", mot, " est :  ", nbcaract )
      FSI
      FIN
```

Ou encore

```
PROGRAM longmot3
VARIABLE  nbcaract : ENTIER  (* nombre de caractères du mot *)
                   mot : CHAINE DE CARACTERE[20]
DEBUT
AFFICHER ( "donner le mot : " )
LIRE  (mot)  (* là on saisira « INFORMATIQUE » *)
nbcaract←LONGUEUR(mot)
SI  ( nbcaract > 1 ) ALORS DEBUT
                           AFFICHER ( "la longueur est supérieure à 1 : on affiche")
                           AFFICHER ( "longueur du mot  ", mot, " est :  ", nbcaract )
                           FIN
FSI
FIN
```

a) **La structure alternative**.

Syntaxe :

```
< instruction0>
SI < condition > ALORS < instruction1 >
                 SINON < instruction2 >
FSI
<instruction3>
```

Si la condition est vérifiée, instruction1 est exécutée, puis instruction3. Dans le cas contraire, instruction2 est exécutée à la place de instruction1, puis instruction3

Exemple

Ecrire un algorithme nommé **division** qui :

- saisit deux réels a et b,
- calcule le quotient de a par b dans le cas où b est non nul, et affiche ce quotient et arrête le programme,
- avertit l'utilisateur de l'impossibilité de l'opération dans le cas où b est nul puis arrête le programme.

Analyse: à faire en exercice

Variable : deux variables a et b de type **REEL**

Algorithme

```
PROGRAM division
VARIABLE a, b : REEL
DEBUT
    ( * saisie des valeurs de a et b *)
    AFFICHER (" donner a= " )
    LIRE(a)
    AFFICHER (" donner b= " )
    LIRE(b)
    SI (b=0) ALORS AFFICHER ("diviseur nul : calcul impossible")
             SINON AFFICHER( "le quotient de ", a, "par ",b," est :", a/b )
    FSI
FIN
```

c) **Les SI imbriqués**

Syntaxe :

```
SI < condition1> ALORS < instruction1>
            SINON SI <condition2> ALORS <instruction2>
                                  SINON SI…….
                                ……..
SINON < instruction finale> ( *si aucune des conditions précédentes n'est vérifiée*)
FSI
```

Exercice 5

Ecrire un algorithme qui saisit 2 réels a et b, résout l'équation $ax+b = 0$ et affiche la solution si elle existe.

Analyse : a, b et c sont quelconques. a peut être nul, b peut être non nul lorsque a est nul etc…
Variables : a et b : des réels

Algorithme :

```
PROGRAM exo5
VARIABLE        a, b  :  REEL
DEBUT
(* saisie des réels a et b *)
   AFFICHER ("donner a :")
   LIRE (a)
   AFFICHER ("donner b :")
   LIRE (b)
(* resolution *)
   SI (a=0) ALORS
            SI (b=0) ALORS  (* cas a=0 et b=0 *)
                        AFFICHER("IR est solution")
                    SINON  (* cas a=0 et b nul *)
                        AFFICHER("aucune solution")
            FSI
          SINON  (* cas a non nul *)
            AFFICHER("la solution est x=  ",-b/a)
   FSI
FIN
```

d) **Structure de choix**

choix est une variable pouvant prendre plusieurs valeurs val1, val2, …..., valN. A chacune de ces valeurs est associée l'exécution d'instructions instruction1 (pour val1), instruction2 (pour val2) , ……, instructionN.

L'instruction instruction_def (instruction par défaut) est exécutée si la valeur prise par choix ne correspond à aucune de des valeurs val1, val2, ….., valN.

Syntaxe :

```
SUIVANT ( choix ) FAIRE
      < val1 > : < instruction1 >
      < val2 > : < instruction2 >
      ……..
      ……..
      < valN > : < instructionN >
SINON       < instruction_par_defaut >
FINSUIVANT
```

Exercice 6

Ecrire un algorithme qui, à partir d'un menu affiché à l'écran permettant de faire un choix, effectue ou la somme, ou le produit ou le quotient de 2 nombres réels a et b saisis en début de programme. Le quotient demandé est b/a, si a est non nul.

Analyse :

On conçoit le menu à partir duquel l'utilisateur fera son choix d'opérations. Prévoir pour le quotient le cas où a peut être nul (utilisation d'une structure conditionnelle).

Variables : a, b : des réels
choix : variable de type **CARACTERE** désignant le choix effectué parmi les trois opérations possibles.

Algorithme :

```
PROGRAM exo6
VARIABLE        a, b : REEL
                choix : CARACTERE
DEBUT
(* saisie des réels a et b *)
   AFFICHER ("donner a")
   LIRE (a)
   AFFICHER ("donner b")
   LIRE (b)
(* creation du menu *)
   AFFICHER ("operations possibles : addition – produit – quotient ")
   AFFICHER ("taper s pour somme ")
   AFFICHER ("taper p pour produit ")
   AFFICHER ("taper q pour quotient de b par a ")
(* choix de l'operation *)
   AFFICHER (" faites votre choix")
   LIRE (choix)
   SUIVANT (choix) FAIRE
         's' : AFFICHER ("la somme est :",a+b)  (* traitement somme de a et b *)
         'p' : AFFICHER ("le produit est :",a*b)  (* traitement produit de a et b *)
         'q' : DEBUT   (* traitement du quotient de b par a : prévoir a nul *)
             SI (a=0) ALORS AFFICHER ("operation impossible : division par 0")
                      SINON AFFICHER (" le quotient de",b,"par ",a," est :",b/a)
             FSI
           FIN
   SINON AFFICHER (" choix non prevu")
   FINSUIVANT
FIN
```

5°) **Les boucles**

On parle de boucle quand il s'agit de répéter un certain nombre (fini) de fois une instruction ou un groupe d'instructions, le nombre de répétitions étant, selon le cas , connu à l'avance ou pas.

Remarque :

- ✓ L'exécution des successions d'instructions «étant séquentielle, les boucles permettent un « retour » en arrière (remontée) lors de l'exécution d'un programme.
- ✓ Toute boucle doit comporter un test d'arrêt.
- ✓ On dit d'un programme qu'il « **boucle** » si le corps de cette boucle (le groupe d'instructions qu'elle est censée répéter) est répétée indéfiniment.

a) Structure itérative (ou boucle TANTQUE)

Cette structure permet la répétition d'un groupe d'instructions tant qu'une condition donnée est satisfaite.

Syntaxe :
```
TANTQUE < condition > FAIRE
                < instruction 1>
                ..................
                < instruction k>
FINTQUE
```

Remarque : Pour cette boucle :

- ✓ le test est réalisé avant d'entrer dans la boucle. On peut donc ne jamais y entrer !
- ✓ le nombre de répétitions n'est pas connu à l'avance.

Exercice 7:

Ecrire un algorithme qui saisit vingt (20) nombres strictement positifs, et calcule la somme de ces 20 nombres en utilisant **TANTQUE**.

On suppose que l'opération ne s'arrête que lorsque ces 20 nombres ont été saisis et qu'on y arrive toujours.

Analyse : On fait ici le choix de saisir jusqu'à obtenir les 20 nombres strictement positifs.
On met en place un compteur qui compte les nombres strictement positifs, et la somme est calculée au fur et à mesure. (On a ainsi la somme dès qu'on a saisi le vingtième nombre positif). Utilisation d'une structure conditionnelle et d'une boucle **TANTQUE**

Variables : j (compteur des nombres positifs), som (somme des nombres positifs) et saisie (nombre qui est en train d'être saisi)

Algorithme :

```
PROGRAM exo7  (* à tracer après *)
VARIABLE        j : ENTIER
            som, saisie : REEL
DEBUT
    som←0   (* initialisation des compteurs *)
    j←0
    TANTQUE j<20 FAIRE
        AFFICHER ("donner un reel ")
        LIRE (saisie)
        SI (saisie>0) ALORS
                DEBUT
                  j←j+1 (* incrémentation du compteur j *)
                  som ← som+saisie  (* mise à jour de la somme *)
                FIN
        FSI
```

```
    FINTQUE
    AFFICHER (”la somme est :”,som)
FIN
```

b) Structure répétitive (ou boucle REPETER)

Cette structure permet la répétition d'un groupe d'instructions jusqu'à ce qu'une condition donnée soit satisfaite.

Syntaxe :

```
REPETER
        < instruction 1>
        ……….
        < instruction k>
    JUSQUA<condition>
```

Elle ressemble à une structure itérative. La seule différence est que la boucle est exécutée au moins une fois, la condition étant testée seulement à la fin de celle-ci.
Ici encore, le nombre de répétitions n'est pas connu à l'avance.

Exercice 8: (même énoncé que l'exercice 7 mais avec **REPETER**)
Ecrire un algorithme qui saisit des nombres au clavier et calcule la somme des 20 premiers nombres positifs de cette saisie.

On suppose que l'opération ne s'arrête que lorsque ces 20 nombres ont été saisis et qu'on y arrive toujours.

```
PROGRAM exo8
VARIABLE        j : ENTIER
                som, saisie : REEL
DEBUT
    som←0  (* initialisation *)
    j←0
    REPETER
        AFFICHER (”donner un reel “)
        LIRE(saisie)
        SI (saisie>0) ALORS
                DEBUT
                   j←j+1
                  som ← som+1
                FIN
        FSI
    JUSQUA (j=20)
    AFFICHER(”la somme est :”,som)
FIN
```

Exercice 9:
Ecrire un algorithme qui saisit au clavier une série de nombres réels qui doivent être compris entre 0 et 20, calcule et affiche leur moyenne une fois la saisie terminée. La saisie est considérée comme terminée si le nombre saisi est en dehors de l'intervalle [0 ; 20].
Deux situations à prévoir :

Première situation :

On suppose que l'utilisateur ne se trompe jamais dans sa saisie. La saisie d'un nombre situé en dehors de l'intervalle [0 ; 20] marque alors réellement la fin de la saisie. Attention au cas où le premier nombre saisi est situé en dehors de l'intervalle.

Seconde situation :

On suppose qu'il peut se tromper en saisissant les nombres.

Proposition de solution

Première situation :

```
PROGRAM exo9_1
VARIABLE        nb2saisie : ENTIER
                somme, saisie : REEL
                arret : BOOLEEN
DEBUT
    somme←0  (* initialisation *)
      nb2saisie←0
    arret←.FAUX.
    REPETER
        AFFICHER ("saisir un reel ")
        LIRE(saisie)
        SI ((saisie>=0) ET (saisie<=20)) ALORS
                                          DEBUT
                                            somme←somme+saisie
                                             nb2saisie←nb2saisie+1
                                          FIN
                                       SINON arret←.FAUX.
        FSI
    JUSQUA (arret)
      SI (nb2saisie=0)      ALORS AFFICHER ("aucune saisie, pas de moyenne calculée")
                            SINON AFFICHER ("la moyenne est : ", somme/nb2saisie)
    FSI
FIN
```

Seconde situation

```
PROGRAM exo9_2
VARIABLE        nb2saisie : ENTIER
                somme, saisie : REEL
                arret : BOOLEEN
                voulu : CARACTERE
DEBUT
    somme←0
      nb2saisie←0
    arret←.FAUX.
    REPETER
        AFFICHER ("saisir un reel ")
```

```
        LIRE(saisie)
        SI  ((saisie>=0) ET (saisie<=20)) ALORS
                                        DEBUT
                                           somme←somme+saisie
                                            nb2saisie←nb2saisie+1
                                        FIN
                                SINON
                                    DEBUT
                                      AFFICHER ("arret o/n ?")
                                      LIRE  (voulu)
                                      SI  (voulu='o') ALORS arret←.VRAI.
                                            SINON AFFICHER  ("saisie nulle")
                                       FSI
                                    FIN
          FSI
        JUSQUA  (arret)
          SI (nb2saisie=0)      ALORS AFFICHER ("aucune saisie, pas de moyenne calculée")
                                SINON AFFICHER ("la moyenne est : ", somme/nb2saisie)
        FSI
    FIN
```

c) Structure POUR (ou boucle POUR)

On est ici dans le cas où le nombre de répétitions est connu à l'avance.
On introduit alors un compteur de répétitions (compteur de boucle).

Syntaxe :

```
POUR < compteur> ←<val_debut> JUSQUA <val_fin>  PASDE < incrément > FAIRE
          < instruction1>
          ……..
          < instruction k>
 FPOUR
```

Remarques :

- ✓ compteur , val_init , val_fin sont des variables ou constantes de type **ENTIER**,
- ✓ Si val_init est égale à val_fin, la boucle est exécutée une seule fois.
- ✓ **PASDE** est l'incrément, le pas d'avancement, constant durant la boucle. La valeur par défaut est 1, et dans ce cas, on peut ne pas préciser le pas.
- ✓ Si la valeur de **PASDE** est positive, val_fin doit être supérieure à val_init, sinon la boucle n'est pas exécutée. C'est le contraire si le pas est négatif.

Exemple:

[1] Calcul de la moyenne de 20 nombres saisis au clavier et affichage de cette moyenne.
[2] Liste des 50 premiers entiers impairs.

Exercice 10

Lire et interpréter l'algorithme ci-dessous :

```
PROGRAM bidon

        (* déclaration*)
        CONSTANTE       nbessai =10
        VARIABLE a, b, c  : REEL
                 sortie   : CARACTERE
                 ok       : BOOLEEN
                 i,j      : ENTIER

DEBUT   (* Corps du programme *)
    j←0
    REPETER  (* on désire effectuer plusieurs saisies*)
        i←0
        REPETER  (* le reel  a  doit être non nul *)
            AFFICHER("donner a")
            LIRE(a)
            i←i+1
        JUSQUA ((a≠0) OU (i=nbessai))
        SI((i=nbessai) ET (a=0)) ALORS
                                DEBUT
                                    AFFICHER("vous avez depasse le nombre d_essais
                                              autorises")
                                 AFFICHER("sasisie incomplete")
                                  FIN
                                SINON
                                   DEBUT
                                 AFFICHER("donner b, c dans cet ordre")
                                 LIRE(b, c)
                                 AFFICHER("Saisie N°", j+1, " : ")
                                 AFFICHER("a= ", a, "   b= ",b, "   c=", c)
                                 j←j+1
                                FIN
          FSI
          AFFICHER("voulez-vous arrêter o/n ?")
          LIRE(sortie)
          ok ← (sortie='o')
    JUSQUA(ok)   (* arrêt du programme *)

FIN
```

Exercice 11 : Exemple d'algorithme

Ecrire un algorithme qui saisit une série de n réels, n étant un entier non nul demandé en début de programme, et affiche à la fin de la saisie le plus petit des réels saisis ainsi que le numéro de saisie de celui-ci, et le nombre de réels strictement positifs saisis.

Traitement

Analyse : Il s'agit ici de saisir n réels, de rechercher le plus petit d'entre eux, de préciser son numéro de saisie, et de compter le nombre de réels strictement positifs. On n'a aucun moyen de garder en mémoire toutes les valeurs saisies. Tout doit donc être fait au fur et à mesure que l'on saisit ces n réels : on détermine en même temps le plus petit de tous ceux qui ont été saisis ainsi que son numéro, et on compte le nombre de positifs.

Variables : n :le nombre de réels à saisir
saisie : le réel de la saisie courante
min : le plus petit des réels saisis
imin : le numéro de saisie de min
nbrepos : le nombre de réels saisis strictement positifs
i : un compteur courant de saisie.

L'algorithme : *(* qui doit comporter des commentaires pour la compréhension et la lisibilité de l'algorithme *)*

```
PROGRAM exemple01
VARIABLE
        i, imin, n, nbrepos :      ENTIER
        min, saisie :      REEL
DEBUT
        AFFICHER("donner le nombre de reels a saisir ")
        LIRE(n)
(* contrôle de la saisie de n qui doit être strictement positif *)
        TANTQUE( n≤ 0 ) FAIRE
                AFFICHER(" redonner n")
                LIRE(n)
        FINTQUE
        AFFICHER("donner le premier nombre")
        LIRE(saisie)
        ( * initialisation de imin à 1, min à la première valeur saisie, et nbrepos à 0 ( si saisie <
        0 ) ou 1 ( si saisie >0 )*)
        imin←1
        min←saisie
        nbrepos←0
        SI ( saisie>0) ALORS nbrepos←1
        FSI
        ( * boucle des saisies successives. A chaque fois on actualise min, imin ( si saisie < min
        ) et nbrepos si saisie >0 *)
        POUR i←2 JQUA n FAIRE
                AFFICHER("donner le", i, "eme nombre")
                LIRE(saisie)
                SI ( min >saisie ) ALORS
                        DEBUT
                                min←saisie
                                imin←i
                        FIN
                FSI
                SI  ( saisie >0 ) ALORS nbrepos←nbrepos+1
```

```
            FSI
        FPOUR
        AFFICHER("la ", imin, "eme saisie correspond au plus petit element qui est", min)
        AFFICHER("et on a saisi ", nbrepos, " nombres strictement positifs")
FIN.
```

SERIE N°1 : Instructions de base

SERIE1_1

Tracer les algorithmes suivants :

```
PROGRAM exo11
VARIABLE
        a, b, c : ENTIER

DEBUT
        a←5
        b←-24
        c←a+b
        a←2
        c←c+b-2*a
FIN
```

```
PROGRAM exo12
VARIABLE
        a, b, c : REEL

    DEBUT
    a←-1
    a←a+2
b← a*2+a
    c←7
    c←b-c
    c←c+a-b
    a←b-c*a
    a←(b-a)*c
    b←(a+c)*b
FIN
```

SERIE1_2

Ecrire l'instruction qui permet d'afficher le message BONJOUR à l'écran.
Ecrire l'instruction qui permet de saisir une valeur réelle qui est ensuite stockée dans la variable x.
Ecrire l'instruction qui affiche le contenu de la variable y, précédé du message « la valeur de l'exemple est : ».
Ecrire l'instruction qui permet de permuter le contenu des variables x et y.

SERIE1_3

Soit l'algorithme suivant : (les mots clés sont en gras et en majuscule)

```
ALGO adeviner
VARIABLE a, b, c : ENTIER
DEBUT
    AFFICHER(" donner a= ")
    LIRE(a)
    AFFICHER(" donner b= ")
    LIRE(b)
    AFFICHER("les valeurs saisies sont :  a=",a," et  b=",b)
    SI ( a<=b ) ALORS AFFICHER("on n'a rien a faire")
```

```
            SINON
              DEBUT
                    c←a
                    a←b
                    b←c
                    AFFICHER("voici le resultat :  a=",a," et  b=",b)
              FIN
    FSI
  AFFICHER("au revoir")
FIN
```

Que fait cet algorithme ? Préciser ce qui est affiché à l'écran dans les cas où on a saisi :

a. a=12 , b=34
b. a=12 , b=12
c. a=12 , b=7.

SERIE1_4

1. On achète une certaine quantité d'un produit.
Ecrire un algorithme (Algo_1) saisissant le prix Hors Taxe (PHT) d'une unité de ce produit, la quantité achetée, et affiche le montant total de l'achat en PTT (Prix Toutes Taxes), sachant que la TVA appliquée est de 18,6%.
2. Ecrire un algorithme (Algo_2) qui saisit un entier naturel N, et calcule la somme des entiers consécutifs inférieurs ou égaux à ce nombre, sans utiliser aucune formule.
3. Ecrire un algorithme (Algo_3) qui demande un entier N au départ, et un entier positif S, affiche N et les S prochains nombres consécutifs suivant N ainsi que la somme de tous les nombres affichés.

SERIE1_5

1. Ecrire un algorithme permettant d'effectuer la somme, le produit et la moyenne de trois nombres saisis au clavier.
2. Ecrire un algorithme permettant d'effectuer le produit ou la moyenne de réels strictement positifs (dont on ne connaît pas à l'avance le nombre) saisis au clavier. La saisie s'arrêtera si on tape un nombre négatif ou nul, et on menu sera affiché au départ.
3. Ecrire un algorithme qui saisit N (un entier naturel non nul) et N réels strictement positifs (N est demandé en début de l'algo), détermine et affiche le plus petit réel saisi ainsi que le rang de saisie de celui-ci.

SERIE1_6

Ecrire un algo permettant de résoudre dans l'ensemble des réels ou des nombres complexes une équation du type $ax^2+bx+c=0$, où a, b et c sont des réels.
L'algo demandera au départ l'ensemble de travail, puis à la sortie, affichera la (ou les) solution(s), s'il en existe, ou alors un message indiquant que l'équation n'a aucune solution.

SERIE1_7

1. Ecrire un algo qui :
   - saisit une note,
   - vérifie si elle est correcte ou pas (note correcte : note comprise entre 0 et 20)
   - affiche cette note ainsi que le message « admis » si elle est supérieure ou égale à 10.
3. Ecrire un algo qui :
   - saisit N notes qui doivent être correctes (N est inconnu au départ),
   - calcule la moyenne des notes saisies,

- affiche cette moyenne ainsi que le nombre de notes supérieures ou égales à 10.

SERIE1_8

1. Ecrire un algo qui :
   - saisit un entier N, puis saisit N réels qui seront stockés dans un tableau A,
   - détermine le plus petit élément de A, et place cet élément en dernière position,
2. Utiliser cet algo pour réaliser le tri de A par ordre décroissant.

SERIE1_9

Ecrire un algo qui permet de convertir en base 2 un nombre écrit en base 10.
Ecrire un algorithme qui compte les occurrences d'une lettre dans un mot.

SERIE1_10

Proposer un algorithme permettant de faire, disposant de deux matrices ,

- la somme, si possible, de ces deux matrices,
- le produit, si possible, de ces deux matrices.

Si l'opération a pu être effectuée, afficher le résultat. Sinon, afficher : » Calcul impossible car……. ».

SERIE1_11

Ecrire un algorithme qui réalise la transposition d'une matrice carrée de taille N en n'utilisant que la seule matrice des données.

B-2 LES TABLEAUX

Les données sont rangées en mémoire. En algorithmique, on se contente de donner des noms et valeurs aux variables et constantes. C'est le S.E qui s'occupera de leur affecter les adresses précisant leur emplacement physique en mémoire.
Les données utilisées jusque là sont des données de types simples, des données de base.
Certaines données, pour être exploitées convenablement et utilisées dans un algorithme, ont parfois besoin d'être structurées.
Considérons par exemple la situation suivante: Xavier est professeur principal d'une classe d'un collège. Il doit disposer des notes de l'année de ses élèves et préparer ces données, les exploiter pour les présenter en conseil de classe à la fin de l'année (calcul de moyennes annuelles, classement des élèves, …….).
Pour un traitement informatique plus efficace, il est plus indiqué de présenter, par exemple, les noms des élèves d'une classe, regroupés pour constituer une nouvelle donnée qu'on appellera **nom**. C'est une structure nouvelle, linéaire, constituée de données de même type, ici CHAINE DE CARACTERE : un **tableau de dimension un.**
On pourra faire de même pour les notes de ces élèves.

1°) Les tableaux à une dimension

a) Définition d'un tableau :

Un tableau à une dimension est une structure de donnée linéaire qui permet de stocker des données **de même type**.
Chacun des éléments d'un tableau est repéré à l'aide d'un indice qui indique sa position dans ce tableau.
L'indice minimal, par défaut, est 1.
Un tableau est un type de donnée, que l'on doit déclarer avant d'être utilisé.
Exemple : La liste à ranger de l'exemple 2 de la page 2 peut être mise dans un tableau.

b) Syntaxe de déclaration

Un tableau est caractérisé par un nom (l'identificateur), l'intervalle d'évolution de ses indices et le type (commun) de ses éléments. La syntaxe est:

```
<nom>: TABLEAU [<indice_minimal> : <indice_maximal>] DE <type des données>
```

Exemple : la classe seconde1 compte 8 élèves.
La déclaration du tableau des noms nommé seconde1, est :

```
seconde1 : TABLEAU [1 :8] DE CHAINE DE CARACTERE[10]
```

Exemple : seconde1

indice	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
seconde1	**alain**	**jean**	**koffi**	**jacques**	**kouakou**	**fatou**	**ballo**	**armel**

seconde1[5]= "kouakou". Remarquer le crochet [ ].

Le plus petit indice est 1 ; le plus grand est 8.

Exemples : notes : **TABLEAU[1 : 8] DE REEL**
passage: **TABLEAU[1 : 8] DE BOOLEEN** (** il s'agit de répertorier les états de chaque élève : admis (OUI), recalé (NON) **)

c) Opérations sur les tableaux

c-1 Création d'un tableau :

Créer un tableau c'est remplir ses différentes cases. Cette opération peut se faire séquentiellement ou dans un ordre quelconque.

Exemple :

```
PROGRAM creation
VARIABLE
seconde1: TABLEAU[1 : 8] DE CHAINE DE CARACTERE[10]
i : ENTIER
DEBUT
        POUR  i ← 1 JUSQUA 8 FAIRE
        AFFICHER ("donner le ",i," eme nom : ")
        LIRE(seconde1[i])
        FPOUR
FIN
```

Remarque :
On pouvait déclarer une dimension maximale de seconde1 , puis demander à l'utilisateur le nombre exact de ses saisies, en « conversationnel ». Cela aurait donné l'algorithme suivant :

```
PROGRAM creation
CONSTANTE n =35 (* effectif maximal d'une classe : 35  *)
VARIABLE
seconde1: TABLEAU [1 : n]DE CHAINE DE CARACTERE[10]
i , nbeleve: ENTIER
DEBUT
        AFFICHER ("donner le nombre d_eleves de la classe : ")
        LIRE (nbeleve)
        TANTQUE ( nbeleve > n ) FAIRE  (* ne pas dépasser n* )
        AFFICHER ("redonner le nombre d""eleves de la classe : ")
        LIRE (nbeleve)
        FINTQUE
        POUR  i ← 1 JUSQUA nbeleve FAIRE
        AFFICHER ("donner le nom du ",i," ème élève: ")
```

```
            LIRE(seconde1[i])
            FPOUR
FIN
```

c-2 Edition d'un tableau

Editer un tableau c'est parcourir ses différentes cases et afficher leurs contenus.
Pour la suite, on peut supposer que le tableau seconde1 est déjà en mémoire. Nous reproduisons ici quand même à chaque fois la partie création pour avoir un programme cohérent, qui ne nécessite pas cette mise en mémoire que l'on ne maîtrise pas encore à ce niveau du cours.

```
PROGRAM edition
CONSTANTE n =35 (*effectif maximal d'une classe : 35  *)
VARIABLE
seconde1: TABLEAU [1 : n]DE CHAINE DE CARACTERE[10]
i , nbeleve: ENTIER
DEBUT
            (* On crée le tableau seconde1  *)
      AFFICHER ("donner le nombre d""eleves de la  classe : ")
      LIRE (nbeleve)
      TANTQUE ( nbeleve > n ) FAIRE  (* ne pas dépasser n* )
            AFFICHER ("redonner le nombre d""eleves de la classe : ")
            LIRE (nbeleve)
      FINTQUE
      POUR  i ← 1 JUSQUA nbeleve FAIRE
            AFFICHER ("donner le nom du ",i," ème élève: ")
            LIRE(seconde1[i])
      FPOUR
      (* Partie édition du tableau seconde1  *)
      POUR  i ← 1 JUSQUA nbeleve FAIRE
      AFFICHER (i," ème élève: ", seconde1[i])
      FPOUR
FIN
```

c-3 Rechercher un élément dans un tableau

Faire attention : l'élément cherché peut ne pas être dans le tableau, mais peut aussi s'y trouver en plusieurs endroits. Dans cet exemple, la recherche consiste à savoir si le nom est présent sur la liste. Le programme s'arrête dès qu'on a trouvé une occurrence du nom.

```
PROGRAM recherche
CONSTANTE n =35 (*effectif maximal d'une classe : 35  *)
VARIABLE
seconde1: TABLEAU [1 : n]DE CHAINE DE CARACTERE[10]
i , nbeleve: ENTIER
nom : CHAINE DE CARACTERE[10]
trouve : BOOLEEN
DEBUT
```

```
                                    (* On crée le tableau seconde1  *)
        AFFICHER ("donner le nombre d""eleves de la classe : ")
        LIRE (nbeleve)
        POUR  i ← 1 JUSQUA nbeleve FAIRE
              AFFICHER ("donner le nom du ",i," ème élève: ")
              LIRE(seconde1[i])
        FPOUR
        (* Partie recherche d'un nom dans le tableau seconde1  *)
        AFFICHER ("donner le nom a chercher : ")
        LIRE (nom)
        i←1
        trouve←.FAUX.
        REPETER
              SI ( nom=seconde1[i] ALORS trouve←.VRAI.
              FSI
        i←i+1
        JUSQUA( trouve OU i=nbeleve )
        SI (NON trouve)   ALORS AFFICHER("ce nom n_est pas sur la liste ")
                          SINON AFFICHER(nom," est sur la liste")
FIN
```

c-4 Exercice 12 Supprimer ou ajouter un élément d'un tableau

Faire attention :

- ✓ pour un ajout, il faut avoir prévu une dimension suffisante pour contenir tous les éléments à ajouter. Les déclarations des tableaux sont faites en général en début du programme (déclaration statique)
- ✓ pour une suppression, il faut prévoir un « marquage » pour signifier que l'élément a été supprimé. Un élément donné est toujours présent dans la mémoire sauf s'il a été écrasé.

c-5 Exercice 13 Trier une tableau

Chercher sur Internet des algorithmes de tri.

Ecrire un algorithme du :

- ✓ tri par sélection,
- ✓ tri par insertion,
- ✓ tri bulles.

Exercice 14 :

Ecrire un algorithme du tri bulles pour un tableau de n notes (n au plus égal à 35).

```
PROGRAM  tribulles
CONSTANTE n =35 (*effectif maximal d'une classe : 35  *)
VARIABLE  note_math: TABLEAU [1 : n] DE REEL
            i , nbeleve: ENTIER
          aux :  REEL
          trie :  BOOLEEN
DEBUT
```

```
                                        (* On crée le tableau note_math *)
        AFFICHER ("donner le nombre d""eleves de la classe : ")
        LIRE (nbeleve)
        POUR  i ← 1 JUSQUA nbeleve FAIRE
              AFFICHER ("donner la note du ",i," eme élève: ")
              LIRE(note_math[i])
        FPOUR
        (* Partie tri du tableau note_math *)
        REPETER
        trie←.VRAI.
        POUR i←1 JQUA nbeleve -1 FAIRE
        SI note_math[i]>note_math[i+1] ALORS
                                        DEBUT
                                            trie←.FAUX.
                                            aux←note_math[i]
                                            note_math[i]←note_math[i+1]
                                            note_math[i+1]←aux
                                        FIN
        FSI
        FPOUR
        JUSQUA(trie)
        POUR  i←1 JQUA nbeleve FAIRE
        AFFICHER(note_math[i],"  ")
        FPOUR
FIN
```

c-6 Exercice 15 :Rechercher un élément dans un tableau trié

Tenir compte du caractère trié du tableau.

c-7 Exercice 16 Supprimer ou ajouter un élément d'un tableau trié

Tenir compte du caractère trié du tableau et faire attention aux ajouts aux extrémités, aux suppressions aux extrémités du tableau.

c-8 Exercice 17 : Fusionner deux tableaux de dimensions n et m, déjà triés

Le résultat est un tableau de taille n+m.

2°) Les tableaux à deux dimensions

Xavier, au lieu d'avoir à manipuler plusieurs tableaux de notes (un par matières)) désire regrouper les notes (des éléments du même type), dans un seul tableau. Ainsi, il n'aura finalement à gérer, par classe, que deux tableaux : le tableau des noms (à une dimension), et celui des notes, qui est d'un type nouveau : de dimension 2.

La syntaxe pour ce nouveau type est la suivante :

```
<nom> : TABLEAU[<indice_ligne_min> : <indice_ligne_max>,  <indice_colonne_min> :
<indice_colonne_max>]  DE <type donnée>
```

Exemple :

notes: **TABLEAU[**1:35 , 1: 6**] DE REEL** (**effectif maximal d'une classe : 35, nombre maximal de notes par élève : 6 **)

Exercice 18

Tracer l'algorithme ci-dessous et préciser ce qui est affiché à l'écran pour salaire=478900

```
PROGRAM utile
VARIABLE        i , salaire: ENTIER
                A : TABLEAU[1:5,1:2] DE ENTIER
DEBUT
        A[1,1]← 10000
        A[2,1]← 5000
        A[3,1]← 1000
        A[4,1]← 500
        A[5,1]← 100
        POUR i ← 1 JQUA 5 FAIRE
                        A[i,2]← salaire DIV A[i,1]
                        salaire ← salaire – A[i,1]*A[i,2]
        FINPOUR
        POUR i ← 1 JQUA 5 FAIRE
                        AFFICHER ( A[i,1], ' ⟶ ', A[i,2])
        FINPOUR
FIN
```

Exercice 19 :

Ecrire un algorithme qui recherche le plus grand élément en valeur absolue d'un tableau 4x5 de réels, puis calcule la moyenne de tous ses éléments, et calcule la moyenne des carrés des écarts de tous les éléments à la moyenne (l'écart type)

Exercice 20:

Ecrire un algorithme qui

- ✓ commence par présenter un menu d'opérations parmi celles décrites ci-dessous
- ✓ puis demande le choix de l'opération à effectuer.

Opérations possibles :

- ✓ calcul de la somme S de deux matrices rectangulaires A(n, m) et B(p, q),
- ✓ calcul du produit H de deux matrices rectangulaires A(n, m) et B(p, q)
- ✓ transposition d'une matrice carrée D et calcul de sa trace. On n'utilisera que la seule matrice D pour le calcul de la transposée.
- ✓ calcul de la transposée E d'une matrice rectangulaire A(n,m).

```
PROGRAM exo20
CONSTANTE k=100
VARIABLE
        n,m,p,q,v,i,j :ENTIER
        A,B,S,H,D,E :TABLEAU [1 :k,1 :k] DE REEL
        trace,aux:REEL
        choix: CARACTERE
```

```
DEBUT
      (* presentation du menu*)
      AFFICHER(" operations possibles: somme-produit-transposee-transposee carree")
      AFFICHER(" taper la lettre precedant l_operation choisie ")
      AFFICHER("          s :operation somme ")
      AFFICHER("          p :operation produit ")
      AFFICHER("          t :operation transposee et trace d'un tableau carre ")
      AFFICHER("          r :operation transposee d'un tableau rectangulaire ")
      AFFICHER(" faites votre choix")

      (* traitement des différentes opérations*)
      SUIVANT choix FAIRE
            's' :   DEBUT
                    AFFICHER("donner nbre lignes et nbre colonnes de A")
                    LIRE(n,m)
                    AFFICHER("donner nbre lignes et nbre colonnes de B")
                    LIRE(p,q)
                    SI ((n<>p) ET (m<>q)) ALORS  AFFICHER("somme impossible")
                                SINON
                                      DEBUT
                                      POUR i←1 JQUA n FAIRE
                                          POUR j← 1 JQUA m FAIRE
                                            AFFICHER("donner A(",i,",",j,")")
                                            LIRE (A[i,j])
                                            AFFICHER("donner B(",i,",",j,")")
                                            LIRE (B[i,j])
                                            S[i,j] :=A[i,j]+B[i,j]
                                          FPOUR
                                      FPOUR
                                      (*affichage du resultat*)
                                      POUR i←1 JQUA n FAIRE
                                          POUR j← 1 JQUA m FAIRE
                                            AFFICHER(" S(",i,",",j,")",S[i,j])
                                          FPOUR
                                      FPOUR
                                      FIN
                    FSI
                    FIN

            'p':    DEBUT
                    AFFICHER("donner nbre lignes et nbre colonnes de A")
                    LIRE(n,m)
                    AFFICHER("donner nbre lignes et nbre colonnes de B")
                    LIRE(p,q)
                    SI (m<>p) ALORS  AFFICHER("produit impossible")
                          SINON
```

```
                    DEBUT
                        POUR  i←1 JQUA n FAIRE
                            POUR j← 1 JQUA m FAIRE
                              AFFICHER("donner A(",i,",",j,")")
                              LIRE (A[i,j])
                            FPOUR
                        FPOUR
                        POUR i←1 JQUA m FAIRE
                            POUR j← 1 JQUA q FAIRE
                              AFFICHER("donner B(",i,",",j,")")
                              LIRE (B[i,j])
                            FPOUR
                        FPOUR
                    (* calcul du produit*)
                    POUR i ← 1  JQUA n FAIRE
                        POUR j ←1 JQUA q FAIRE
                          H[i,j] ←0
                          POUR v← 1 JQUA m FAIRE
                                H[i,j] ← H[i,j]+A[i,v]*B[v,j]
                          FPOUR
                        FPOUR
                    FPOUR
                    (* affichage du resultat*)
                    POUR i←1 JQUA n FAIRE
                        POUR j← 1 JQUA q FAIRE
                          AFFICHER(" H(",i,",",j,")=",H[i,j])
                        FPOUR
                    FPOUR
                    FIN
            FIN
't':    DEBUT
        AFFICHER("donner le nbre de lignes de D")
        LIRE(n)
        POUR i←1 JQUA n FAIRE
            POUR j← 1 JQUA n FAIRE
              AFFICHER("donner D(",i,",",j,")")
              LIRE (D[i,j])
            FPOUR
        FPOUR
        (*calcul de la transposee et de la trace*)
        trace←0
        POUR i ← 1 JQUA n FAIRE
                trace← trace+D[i,i]
               POUR j ← 1 JQUA i-1 FAIRE
                        aux ← D[i,j]
                 D[i,j] ← D[j,i]
```

```
                    D[j,i] ← aux
                FPOUR
            FPOUR
            (*affichage du resultat*)
            POUR i←1 JQUA n FAIRE
                POUR j← 1 JQUA n FAIRE
                  AFFICHER(" D(",i,",",j,")=",D[i,j])
                FPOUR
            FPOUR
            AFFICHER(" la trace est:",trace)
            FIN

        'r':  DEBUT
            AFFICHER("donner nbre lignes et nbre colonnes de A")
            LIRE(n,m)
            POUR i←1 JQUA n FAIRE
                POUR j← 1 JQUA m FAIRE
                  AFFICHER("donner A(",i,",",j,")")
                  LIRE (A[i,j])
                FPOUR
            FPOUR
            (*calcul de la transposee*)
            POUR i←1 JQUA m FAIRE
                POUR j← 1 JQUA n FAIRE
                  E[i,j] ←A[j,i]
                FPOUR
            FPOUR
            (*affichage du resultat*)
            POUR i←1 JQUA m FAIRE
                POUR j← 1 JQUA n FAIRE
                  AFFICHER(" E(",i,",",j,")=",E[i,j])
                FPOUR
            FPOUR
            FIN
        SINON
            AFFICHER(" choix non prevu donc non traite")
        FINSUIVANT
FIN
```

SERIE N°2 : Les Tableaux

SERIE2_1

Ecrire un algorithme qui convertit en base 2 un nombre écrit en base 10.
Ecrire un algorithme qui compte les occurrences d'une lettre dans un mot.

SERIE2_2

T est un tableau de réels de taille m (avec m < 100). Ecrire un algorithme pour chacune des taches suivantes :

a) Edition
b) Recherche d'un élément saisi en début d'algorithme
c) Suppression ou ajout (un choix est proposé sous forme de menu)
d) Tri :
   Chercher sur Internet des algorithmes de tri.
   Ecrire un algorithme du :
   - tri par sélection,
   - tri par insertion,
   - tri bulles.
e) Recherche d'un élément dans un tableau trié.
f) Suppression ou ajout d'un élément dans un tableau trié (un choix est proposé sous forme de menu)
g) Fusion de deux tableaux de dimensions n et m, déjà triés.

SERIE2_3

1°) Saisir 20 réels au clavier, donner le plus grand et le plus petit de ces 20 réels (pas de tableau).
2°) Créer un tableau de 20 réels saisis au clavier et trier dans l'ordre croissant ces réels. On n'utilisera qu'un seul tableau .
3°) Créer un tableau de 20 titres (de longueur limitée à 20 caractères) de chanson d'un « HIT PARADE » saisis au clavier et rechercher la présence dans ce tableau d'un titre donné, avec dans l'affirmative, la position occupée par ce titre.

4°) Créer un tableau de 20 noms des artistes du « TOP20 » de la semaine. La longueur des noms est limitée à 20 caractères . Rechercher la présence dans ce tableau d'un artiste donné et le (ou les) rang(s) occupé(s) par cet artiste.

SERIE2_4

1°) Un produit vient d'être interdit à la vente, et vous voulez le supprimer de votre liste de produits disponibles en rayon de votre magasin. Un produit est caractérisé dans vos fichiers par son nom (chaîne de 20 caractères au plus) , et son code (ensemble de 15 caractères). Votre liste comporte 100 produits.
Proposer un premier algorithme permettant d'effectuer la mise à jour de votre liste par la suppression de ce produit de la liste. On suppose que la liste est déjà en mémoire, triée par ordre alphabétique des noms, dans un tableau appelé LISTPROD et que le code produit sert de clé primaire
2°) Proposer un second algorithme permettant d'enregistrer un nouveau produit sur la liste. On suppose que le tableau est de taille suffisante pour l'ajout du produit.
3°) Vous disposez de deux listes de 50 et 70 produits que vous voulez fusionner pour ne constituer qu'une seule liste. Les deux listes sont triées par ordre alphabétique des noms chacune. Proposer un algorithme réalisant cette fusion.

SERIE2_5

Ecrire un algorithme permettant, à partir de 2 tableaux T1 et T2 contenant chacun N entiers non triés, de modifier les tableaux de telle sorte que T1 contienne tous les entiers pairs contenus à l'origine dans T1 et dans T2, et que T2 contienne les entiers impairs contenus à l'origine dans T1 et T2.
L'hypothèse de départ est que T1 et T2 réunis contiennent N entiers pairs et N entiers impairs.
En résultat, les tableaux initiaux T1 et T2 seront affichés sous la forme:

T1	T2
*	*
*	*
…..	….
*	*

Puis, à la suite, après une ligne séparatrice sur laquelle on peut lire

" Résultats de la séparation ",

on affichera les nouveaux tableaux T1 et T2, qu'on aura triés au préalable, sous la forme précisée ci-dessous.

Nombres pairs	Nombres impairs
*	*
*	*
…..	….
*	*

&&&&&&&&&&&&&&&&&

C – LES SOUS PROGRAMMES

1°) Introduction

Un algorithme n'utilise que l'ensemble des mots qui forment son vocabulaire. Les mots écrits jusque là sous ce **FORMAT** sont les seuls mots que l'on a vus de ce vocabulaire. Ils servent à mettre en place le programme et à décrire les différentes actions élémentaires utilisées pour la résolution d'un problème.
Toute action qui ne peut être décrite à l'aide d'un mot du vocabulaire est une action non élémentaire, une action composée. Elle est donc une succession d'actions élémentaires, une succession d'instructions de base.
Le vocabulaire d'un langage donné est toujours en nombre très limité. Il en est de même de l'algorithmique.

Prenons un exemple :
Dans un programme, à chaque fois qu'on affiche un nouveau résultat, on doit sauter 4 lignes.
Pour sauter 4 lignes on écrit les 3 lignes de codes suivantes :

```
POUR i←1 JQUA 4 FAIRE
  AFFICHER(" ")
FINPOUR
```

Si on devait afficher 5 fois dans le programme, on aurait à écrire 15 lignes : 5 groupes de codes pratiquement identiques .
D'où l'idée d'écrire à part ce bout de code, de le nommer (***sauter4ligne*** par exemple) et d'appeler ***sauter4ligne*** à chaque fois qu'on en a besoin. En somme de créer un sous-programme.
On vient d'enrichir ainsi le vocabulaire de notre algorithmique avec le nouveau mot ***sauter4ligne.***
Ce sous programme est écrit en même temps que le programme qui l'utilise (programme appelant), n'est pas dans le corps de ce programme. Il est écrit « à côté » et est appelé en cas de besoin.

2°) Définition et types de sous programme.

a) Définition

Un sous programme est un programme particulier, qui décrit une action précise, qui est utilisé dans le contexte d'un autre programme (le programme appelant).

Pour fonctionner, selon l'action qu'il a à réaliser, il peut avoir besoin de paramètres : des données à fournir dès le départ (paramètre_donnée), ou des variables qui vont contenir des résultats (paramètre_resultat), ou même des paramètre_donnée_résultat (la variable qui contient une donnée va aussi contenir un résultat) .
Les sous-programmes qui ont besoin de paramètres pour travailler sont appelés des sous-programmes paramétrés, les autres sont dits non paramétrés.

Exemples :

- Le sous programme qui saute 4 lignes n'a pas besoin de paramètre.
- Imaginons maintenant que le nombre de lignes à sauter soit variable. Le bout de codes pour les sauts de lignes dépendra alors du nombre ***k*** de lignes à sauter. ***k*** est un paramètre_donnée.
  On aura alors le sous-programme nommé ***sauterligne(k)*** :

```
POUR i←1 JQUA k FAIRE
  AFFICHER(" ")
FINPOUR
```

- Un sous programme qui calcule une somme de 2 matrices aura besoin de paramètre_données (les 2 matrices, la taille commune) et de paramètre_résultat (une matrice et sa taille).

b) Types :

Un sous-programme est soit :

- soit une **FONCTION**:
- soit une **PROCEDURE**.

Une fonction est un sous-programme qui doit renvoyer un et un seul résultat.
Tout sous-programme qui n'est pas une fonction est une procedure.

3°) Syntaxe et structure d'un sous programme

Un sous programme est d'abord un programme. On retrouve donc pratiquement les mêmes syntaxes et structures.

a) Fonction

Syntaxe et structure

```
FONCTION <nom> ( données : liste paramètres données : <type> ) :< type résultat>
      (*Partie déclaration des variables*)
            ……….

       (*Corps de la procédure*)
            DEBUT
                  ……….
            <nom> ←
            FIN <nom>
```

Appel d'une fonction :

L'appel doit être contenu dans une instruction d'affectation. Exemple a← nom_de_la_fonction(paramètres)

Attention :

- Dans le corps d'une fonction doit figurer l'affectation <nom> ← , qui permet de donner le résultat au programme appelant.
- En algorithmique, nous retiendrons que les seuls paramètres d'une fonction sont des paramètres données

Exemple : Calcul de n! (factoriel de n)

```
FONCTION factoriel(données : n : ENTIER ) : ENTIER
VARIABLE fact : ENTIER
DEBUT
        fact←1
        TANTQUE n>1
                fact←fact*n
                n←n-1
        FINTQUE
        factoriel←fact
FIN  factoriel
```

Pour cet exemple, l'appel se fera, par exemple : a←factoriel(5) et retournera 5 !=120.

Exemple de programme utilisant la fonction factoriel ci-dessus : Calcul du nombre de combinaisons de p éléments parmi n ($0 \leq p \leq n$).

La formule que l'on utilisera est : $C_n^p = \frac{n!}{p!(n-p)!}$.

La fonction factoriel sera alors appelée 3 fois.

```
PROGRAM combinaison
VARIABLE
        n, p : ENTIER
        combi : ENTIER    (* contiendra la valeur de la combinaison *)
DEBUT
        AFFICHER(" donner n")
        LIRE(n)
        TANTQUE(n<0) FAIRE    (* contrôle de la positivité de n *)
                AFFICHER(" redonner n >0")
        FINTQUE
        AFFICHER("donner p")
        LIRE(p)
        TANTQUE ((p>n) OU  (p<0) ) FAIRE    (* il faut que l'on ait  0 ≤ p ≤ n *)
                AFFICHER(" redonner p")
        FINTQUE
        combi ← factoriel(n)
```

```
        combi ← combi DIV ( factoriel(p) * factoriel(n-p))
        AFFICHER("combinaison de ",p," elements parmi ",n," =", combi)
    FIN.
```

```
(* écriture de la fonction factoriel *)
    FONCTION factoriel(données : m : ENTIER ) : ENTIER
    VARIABLE fact : ENTIER
    DEBUT
        fact←1
        TANTQUE m>1
            fact←fact*m
            m←m-1
        FINTQUE
        factoriel←fact (* permet de renvoyer le résultat au programme appelant *)
    FIN factoriel
```

Exercice 16.

Ecrire une fonction ***coder*** qui convertit en base b (b<=10) un nombre en base 10.

Exercice 17.

Ecrire une fonction ***decoder*** qui convertit en base 10 un nombre donné en base b.

Paramètres : nbracoder : le nombre à coder en base 10 de type entier
base : la base dans laquelle le nombre est donné (base <=10)

Variables locales
nbre : variable qui contient le nombre en base 10.
a : variable de travail

```
FONCTION decoder ( donnee : nbracoder, base : ENTIER) :ENTIER
    VARIABLE
        pui, nbre10, a : ENTIER
    DEBUT
        nbre10 ← 0
        pui ←1
        REPETER
            a ←nbracoder-( nbracoder DIV 10 ) * 10
            nbre10 ←nbre10+a*pui
            pui ←pui*base
            nbracoder←←nbracoder DIV 10
        JUSQUA ( nbracoder = 0 )
        decoder ← nbre10
    FIN decoder
```

b) Procédure

Une procédure peut ne renvoyer aucun résultat au programme appelant ou lui renvoyer un ou plusieurs résultats.

Parmi les paramètres à préciser pour une procédure, il y a les paramètres données, les paramètres résultats. Un paramètre peut même être à la fois donnée et résultat.

La syntaxe est:

```
PROCEDURE <nom> ( données : liste param_données : <type>, param_resultats : liste des
param_données_résultats :< type >, résultats : liste param_resultats :<type>)
(*Partie déclaration des variables*)
                ..................
(*Corps de la procédure*)
                DEBUT
                        ..................
                FIN procédure
```

Appel d'une procédure:

L'appel se fait de la manière suivante : nom_de_la_procédure (liste paramètres)

Exemple : Saut de m lignes

```
        PROCEDURE sauterligne (données : m : ENTIER )
(* pas de variables à déclarer )
        DEBUT
                POUR i allant de 1 JQUA m FAIRE
                AFFICHER (" ")
                FINPOUR
        FIN sauterligne
```

L'appel se fera, pour sauter 5 lignes : sauterligne(5)

4°) Variables locales – globales.

Les variables déclarées au sein d'un sous programme sont appelées variables locales. Une variable locale n'existe que le temps de l'exécution du sous programme et n'est pas visible (ne peut être utilisée) à l'extérieur de ce sous programme.
Une variable déclarée à l'extérieur d'un module donné sera en général visible par tout module écrit après cette déclaration. C'est une variable globale pour ces modules.

5°) Passage des paramètres.

Lorsqu'un programme appelle un sous programme donné, il lui donne les paramètres nécessaires pour son exécution.
Ces paramètres sont des variables.
Le passage se fait « **par valeurs** » si ce sont les copies de ces variables qui sont utilisées par le sous programme. Les paramètres ne sont donc pas affectés par un quelconque changement de valeur opéré à l'intérieur des sous programmes.
Le passage se fait « **par adresses** » si ce sont les adresses de ces paramètres qui sont données au sous programme pour travailler. Dans ce cas, toute modification effectuée sur un paramètre durant l'exécution du sous programme est effective et persiste même en dehors du sous programme.

Attention : Lors de ces passages, les paramètres fournis doivent correspondre, en TYPES, en NOMBRE, aux types et au nombre des paramètres du sous programme, dans l'ORDRE d'apparition de ceux-ci.

5°) Récursivité

Définition

Un sous programme qui peut s'appeler lui-même est un sous programme récursif.

Particularité : il faut à chaque fois un **critère d'arrêt**.

Certains langages permettent la récursivité, comme le C, Pascal,..... Ce n'est pas le cas de Fortran77, par exemple.

Exemple

Programmer en utilisant une fonction récursive le $n^{ième}$ terme d'une suite de Fibonacci.

Rappel : une suite (U_n) de Fibonacci a comme définition par récurrence :

$$U_0 = U_1 = 1 \quad \text{et pour tout } n > 1,\ U_n = U_{n-2} + U_{n-1}$$

Une réponse

```
FONCTION fibonacci(donnée : n : ENTIER) : ENTIER

VARIABLE fibo : ENTIER
DEBUT
SI ( n=0 OU n=1) ALORS fibo←1 (* critère d'arrêt *)
                 SINON fibo←fibonacci(n-2)+fibonacci(n-1)
FINSI
fibonacci←fibo
FIN
```

Exercice 18

Tracer la fonction ci-dessus pour n=5.

6°) Création de types « propriétaires »

Pour des besoins particuliers, on peut être amené à créer de nouveaux types de données, de variables.

La déclaration d'un nouveau type doit être effectuée juste après la déclaration des constantes, et avant celle des variables.

Exemple 1

On peut « rebaptiser » le type booleen en un nouveau type VraiouFaux :

La déclaration serait alors :

```
TYPE VraiouFaux=BOOLEEN
```

L'exploitation en serait :

```
PROGRAM toto
CONSTANTE M=70
TYPE  VraiouFaux=BOOLEEN
VARIABLE
      i,j :ENTIER
      reponse :VraiouFaux
      .......
DEBUT
.......
FIN
```

Exemple 2

Déclaration d'un type tableau de 120 réels

```
PROGRAM titi
CONSTANTE M=120
TYPE tab=TABLEAU[1:M] DE REEL
……..
VARIABLE
      A :tab
```

Le nouveau type tab peut être utilisé dans une procédure pour la déclaration faite à un de ses paramètres qui est un tableau d'au plus 120 éléments de type réel.

7°) Programme principal – Sous-programmes

Lorsque pour résoudre un problème donné, on a besoin de sous-programmes, le programme qui gère les appels à tous ces sous-programmes en vue de résoudre le problème est appelé programme principal. C'est un programme qui est uniquement appelant.
Alors qu'un sous programme est appelé mais peut éventuellement appeler un autre sous- programme : il est alors appelant.

Exercice 19

a) Ecrire une procédure nommée ***saisie*** qui a comme paramètres un tableau B de taille m et sa taille m, et qui saisit les éléments de ce tableau.
b) Ecrire une procédure nommée ***affichage*** qui a comme paramètres un tableau B de taille m et sa taille m, et qui affiche les éléments de ce tableau.
c) Ecrire une procédure nommée ***valmin*** qui a comme :

- paramètres données un tableau B de taille m, sa taille m, un indice ind_1 (inférieur ou égal à m)
- paramètres résultats un réel min et un entier $indmin$,

et qui renvoie min, le plus petit des éléments de B allant de l'indice ind_1 à l'indice m, ainsi que son indice $indmin$.
d) Ecrire une procédure nommée ***tri***, qui a comme paramètres données un entier m, et comme paramètre donnée réultat un tableau B de taille m, qui trie ce tableau en utilisant la procédure $valmin$, ainsi que le procédé de tri par sélection.
e) Ecrire un programme principal nommé ***triselect*** qui saisit un tableau A de réels de taille n, affiche ce tableau, le trie en utilisant la procédure ***tri***, et qui affiche à nouveau le tableau A.

Analyse :
La procédure ***valmin*** renvoie le plus petit des éléments dont les indices sont compris entre ind_1 et la taille m du tableau.
Dans la procédure tri, il s'agit de deux boucles imbriquées :

la première parcourt le tableau B de 1 à m
la seconde détermine, en utilisant ***valmin***, le plus petit élément de B, de $B[i]$ à $B[m]$, place en $B[i]$ ce plus petit élément. Tout ceci, pour i allant de 1 à m.

Le programme principal consiste à déclarer le type tab utilisé dans les procédures, et à faire des appels.
Un type ***tab*** est déclaré dans le programme principal : tableau de 100 éléments au maximum.

Algorithmes
a) Procédure saisie :

paramètres : ceux indiqués dans l'énoncé

variables locales: i : compteur de boucle

```
PROCEDURE saisie (donnee : m : ENTIER, resultat : B : tab )
        VARIABLE i : ENTIER
        DEBUT
                POUR i ←1 JQUA m FAIRE
                AFFICHER (" donner le ",i, "eme element :")
                LIRE (B[i])
                FPOUR
        FIN saisie
```

b) Procedure affichage
paramètres : ceux indiqués dans l'énoncé
variables locales: i : compteur de boucle

```
PROCEDURE affichage (donnee : m : ENTIER, B : tab )
        VARIABLE i : ENTIER
        DEBUT
                POUR i ←1 JQUA m FAIRE
                AFFICHER (" ",B[i])
                FPOUR
        FIN affichage
```

d) Procedure ***valmin***

paramètres : ceux indiqués dans l'énoncé :
variables locales: i : compteur de boucle

```
        PROCEDURE valmin (donnee : B :tab; m, ind_1 : ENTIER; resultat : min: REEL; indice: ENTIER
        )
        VARIABLE i : ENTIER
        DEBUT
                min ← B[ind_1]
                indice ←ind_1
                POUR i ←ind_1 JQUA m FAIRE
                SI ( B[i] < min ) ALORS
                                DEBUT
                                        min ← B[i]
                                        indice ← i
                                FIN
                FSI
                FPOUR
        FIN valmin
```

d) Procedure ***tri***
paramètres : ceux indiqués dans l'énoncé :
variables locales: i : compteur de boucle

indmin : indice du minimum du sous tableau considéré
min : la valeur minimale du sous tableau
aux : variable d'échange

```
PROCEDURE tri (donnee : m: ENTIER; donnee-resultat : B: tab )
VARIABLE        i, indmin : ENTIER
                min, aux : REEL
DEBUT
        POUR i ←1 JQUA m FAIRE
        valmin( B, m, i, min, indmin )
        aux ← B[i]
        B[i] ←min
        B[indmin] ← aux
        FPOUR
FIN tri
```

e) Programme principal de triselect

paramètres : ceux indiqués dans l'énoncé :
variables locales: A : tableau de réels
n : nombre d'éléments de A

```
PROGRAM triselect
TYPE tab=TABLEAU[1..100] DE REEL
VARIABLE n : ENTIER
         A : tab
DEBUT
        AFFICHER (" donner le noimbre d elements  n  0 < n < 101 )
        LIRE(n)
        TANTQUE ((0>=n) OU (100<n)) FAIRE
        AFFICHER (" redonner  n")
        FINTQUE
        saisie ( n, A )
        affichage (n, A )
        tri (n, A )
        affichage (n, A )
FIN.
```

SERIE N°3 : Les Sous Programmes

Serie3_1

1°) Pour pouvoir utiliser des tableaux comme paramètres d'un sous-programme, il faut avoir déclaré un type nouveau : un type tab, par exemple. Donner la déclaration à faire pour un type tableau de n éléments

2°) a) Ecrire une procédure SAISIEMAT permettant de saisir un tableau X de taille connue n.

b) Ecrire une procédure AFFICHEMAT permettant d'afficher le contenu d'un tableau X de taille connue n.

c) Ecrire une procédure SAUTERLIGNE qui permet de sauter k lignes à l'affichage.

Serie3_2

A et B sont deux tableaux d'entiers de tailles respectives N et M. Le PM des tableaux A et B est l'entier obtenu en multipliant chaque élément de A par chaque élément de B et en faisant la somme de tous ces produits.

a) Ecrire une fonction PM qui calcule le PM de deux tableaux X et Y.

b) Ecrire un programme principal qui saisit deux tableaux A et B, les affiche à l'écran pour vérification après la saisie, et qui calcule le PM des deux tableaux A et B. On sautera 4 lignes après l'affichage de A, 5 lignes après l'affichage de B.

On pourra utiliser les sous programmes de l'exercice 1.

Serie3_3

Ecrire un sous programme SPROG qui fait le produit de deux matrices rectangulaires A(n,p) et B(p,q) et qui met le résultat dans la matrice P. Ecrire ensuite un programme principal qui utilise ce sous programme.

Serie3_4

Soit T un tableau de n caractères formant un mot donné. Ecrire une fonction qui permet de reconnaître que ce mot est un palindrome ou pas. En paramètres données on aura le tableau T et sa taille n.

Serie3_5

1°) a) Ecrire un sous programme MOYENNE qui calcule la moyenne des éléments d'un tableau T de réels.
b) Ecrire un sous programme ECTYPE qui calcule l'écart type des valeurs du tableau T ci-dessus.
2°) Ecrire un programme principal qui calcule la moyenne et l'écart type des éléments d'un tableau A donné de m réels. On saisira A à l'aide du sous programme SAISIEMAT de l'exercice 1.

E – ENREGISTREMENTS - FICHIERS

E-1 : ENREGISTEMENT

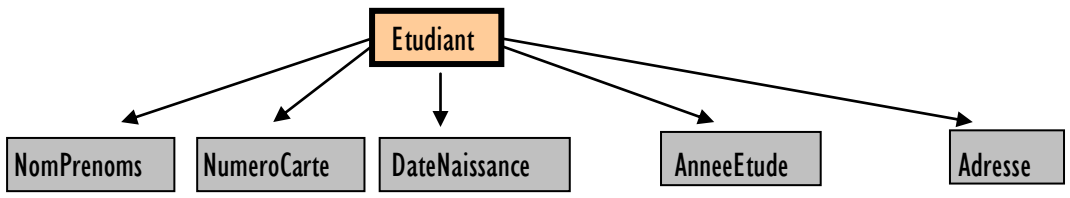
1°) Introduction

Les tableaux nous ont permis de grouper des données mais celles-ci DEVAIENT ETRE DU MEME TYPE.
Si des données sont de types distincts (par exemple une qui est **REEL** et une autre **CARACTERE** , ou même une de type **ENTIER** et une de type **REEL**), elles ne pourront pas être éléments d'un même tableau.

Or considérons par exemple le cas d'étudiants inscrits à l'université. Un étudiant est pour l'université une donnée regroupant plusieurs informations : le nom et les prénoms , la date de naissance , le n° de carte d'étudiant , l'adresse , l'année d'étude , ses notes , etc ….

2°) Structure en arborescence

Un étudiant peut être représenté à l'aide du schéma suivant :



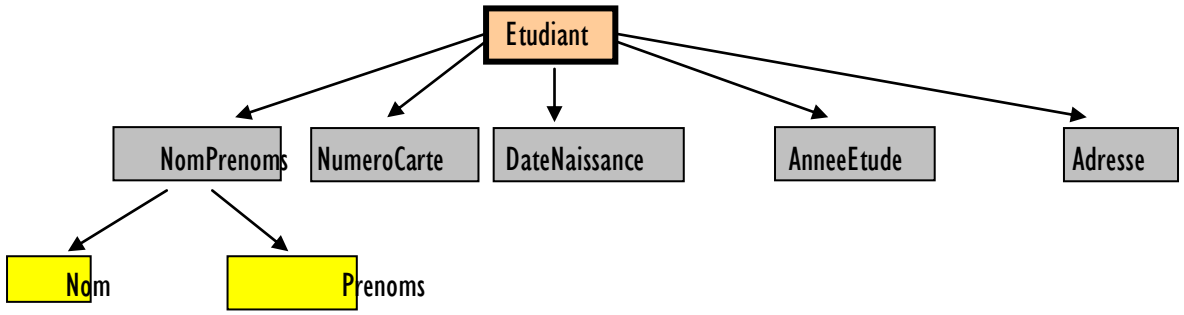
Types ------► Ch. Caract Ch. Caract Ch. Caract Entier Ch. Caract

Problème : Comment peut-on représenter cette donnée Etudiant ?

Cette nouvelle structure qui représente un nouveau type de donnée est une **structure en arborescence**.
Elle est caractérisée par :

- **la racine**, qui représente la donnée elle-même,
- les différentes données de type simple que constituent les branches, qui caractérisent la donnée Etudiant, et qu'on appelle **les champs** de la structure.

Remarque :
Il peut arriver qu'un champ soit lui aussi une structure interne en arborescence. Par exemple le champ NomPrenoms peut être pris comme une structure dont les champs sont Nom et Prénoms. On aurait alors la structure globale suivante :



3°) Clé primaire d'une structure

Définition
Une clé primaire d'une structure est un champ ou un groupe de champs de cette structure qui permet de la caractériser de manière unique.
Par exemple, ici le champ NumeroCarte peut être pris comme clé primaire car le N° de carte d'un étudiant est unique au sein de l'université.
Mais si l'étude porte sur les étudiants de deux universités, par exemple, ce seul champ ne pourra plus être pris comme clé. Il faudrait dans ce cas, par ex, considérer le groupe de champs (NumeroCarte et NomUniversite) pour constituer une clé primaire.

4°) Taille d'une structure

La taille d'une structure est le nombre d'octets occupés par l'ensemble des champs de cette structure.

5°) Description d'une structure

Prenons l'exemple de la structure Etudiant. Elle peut être décrite selon la syntaxe ci-dessous:

```
STRUCTURE Etudiant
        NomPrenom :     CHAINE DE CARACTERE [15 ]
        NumeroCarte :   CHAINE DE CARACTERE [10 ]
        DateNaissance : CHAINE DE CARACTERE [10 ]
        AnneeEtude :    CHAINE DE CARACTERE [5 ]
        Adresse :       CHAINE DE CARACTERE [20 ]
```

```
FIN STRUCTURE
```

6°) Utilisation

Un nouveau type de variable peut alors être déclaré : le type **STRUCTURE** etudiant.
La déclaration peut être alors la suivante :

```
VARIABLE
        individu1 : STRUCTURE etudiant
        individu2 : STRUCTURE etudiant
        i,j : ENTIER
        reponse : BOOLEEN
```

7°) "Valeur" d'une structure.

Pour donner une "valeur" à une structure, il faut donner des valeurs à chacun de ses champs.
Pour avoir accès à un champ donné, on utilise la notation pointée.

Syntaxe : structure.champ

Exemple : pour donner une valeur au champ N°carte (NumeroCarte) de la variable individu1 , on écrit, par exemple :

individu1.NumeroCarte ← "1254DE "

individu1.NumeroCarte est une donnée de type simple.

8°) **Opérations permises**

Les seules opérations possibles avec les structures sont :

- **l'affectation :**
  Si x et y sont deux variables de même type et de type structure , l'opération x← y signifie que les valeurs des champs de x sont égales à celles des champs correspondant de y.
- **la comparaison :**
  x et y sont égaux s'ils sont de même type et si les champs de x ont tous la même valeur que les champs correspondants de y.

9°) Enregistrement

a) Définition

En algorithmique , pour pouvoir utiliser une structure de données en arborescence, on doit déclarer un nouveau type de données appelé: **ENREGISTREMENT**.

b) Syntaxe :

```
TYPE Etudiant=ENREGISTREMENT
        NomPrenoms :   CHAINE DE CARACTERE [15 ]
        NumeroCarte :  CHAINE DE CARACTERE [10 ]
        DateNaissance : CHAINE DE CARACTERE [10 ]
        AnneeEtude :   CHAINE DE CARACTERE [5 ]
        Adresse : CHAINE DE CARACTERE [20 ]
FIN ENREGISTREMENT
```

c) Utilisation

Exercice 20 : Créer un programme qui saisit deux étudiants etudiant1 et etudiant2, et qui renseigne tous les champs de ces deux enregistrements.

En algorithmique

```
PROGRAM saisie
TYPE Etudiant=ENREGISTREMENT
        NomPrenoms :    CHAINE DE CARACTERE [15 ]
        NumeroCarte :   CHAINE DE CARACTERE [10 ]
        DateNaissance : CHAINE DE CARACTERE [10 ]
        AnneeEtude :    CHAINE DE CARACTERE [5 ]
        Adresse :       CHAINE DE CARACTERE [20 ]
FIN ENREGISTREMENT
VARIABLE
        personne : Etudiant
        individu1 : Etudiant
        individu2 : Etudiant
        i : ENTIER

DEBUT
        AFFICHER("donner le nom de l_individu1")
        LIRE(personne.NomPrenoms)
        AFFICHER("donner le numero de sa carte")
        LIRE(personne.NumeroCarte)
        AFFICHER("donner sa date de naissance")
        LIRE(personne.DateNaissance)
        AFFICHER("donner son année d_etude")
        LIRE(personne.AnneeEtude)
        AFFICHER("donner son adresse")
        LIRE(personne.Adresse)
        individu1← personne
        AFFICHER("donner le nom de l_individu2")
        LIRE(personne.NomPrenoms)
        AFFICHER("donner le numero de sa carte")
        LIRE(personne.NumeroCarte)
        AFFICHER("donner sa date de naissance")
        LIRE(personne.DateNaissance)
        AFFICHER("donner son année d_etude")
        LIRE(personne.AnneeEtude)
        AFFICHER("donner son adresse")
        LIRE(personne.Adresse)
        individu2← personne
FIN.
```

Remarque : l'instruction **avec**

Le groupe d'instructions :

```
AFFICHER ( 'Nom et Prenoms :',personne.NomPrenoms )
AFFICHER ( 'Nunerode Carte :',personne.Numerocarte )
AFFICHER ( 'DatedeNaissance :',personne.DateNaissance )
AFFICHER ( 'Annee d etude : ',personne.AnneeEtude )
AFFICHER ( 'Adresse: ',personne.Adresse )
```

peut être remplacé par :

```
AVEC personne FAIRE
DEBUT
   AFFICHER ('Nom et Prenoms :',NomPrenoms )
   AFFICHER ('Nunerode Carte :',Numerocarte )
   AFFICHER ('DatedeNaissance :',DateNaissance )
   AFFICHER  ('Annee d etude : ',AnneeEtude )
   AFFICHER  ( 'Adresse: ',Adresse )
FIN avec
```

Exercice 21

Reprendre le programme ci-dessus en utilisant une procédure appelée **e_saisie**

E-2 : Enregistrement des ENREGISTREMENTS : les FICHIERS

Comment stocker des enregistrements ?

1°) Tableau d'enregistrements

On peut penser à un tableau d'enregistrements. Cela est possible car un tableau peut recevoir des données toutes de même type. Dans ce cas, si T est un tableau d'enregistrements, T[i] est un enregistrement.
Pour prendre connaissance des enregistrements contenus dans T, on peut afficher les valeurs prises par les champs de l'enregistrement T[i].

Exemple_Exercice 22:

Un athlète est considéré comme un enregistrement nommé **athlete** dont les champs sont :

- le numéro d'inscription (champ ***num***) de type chaine de 8 caractères
- la taille en cm (champ ***taille***) de type entier
- le poids en kg (champ ***poids***) de type entier

1. Effectuer la déclaration de ce nouveau type de donnée appelé **athlete**.
2. T est un tableau de taille maximale N=120 dont les éléments sont de type **athlete**.

Ecrire un programme nommé ***gestion*** qui effectue les taches suivantes:

- saisie d'un nombre m ($2 \leq m \leq 120$),

- saisie des m éléments du tableau T,
- saisie d'une taille t donnée,
- recherche dans le tableau T **du premier** athlète dont la taille est égale à t :
    - si la recherche est fructueuse, affichage de tous les champs de l'athlète trouvé,
    - sinon, affichage du message « aucun athlete ne correspond a cette taille ».

Solution :

Question 1 :

Algorithmique

```
TYPE athlete = ENREGISTREMENT
        num : CHAINE DE CARACTERE [8]
        taille : ENTIER
        poids : ENTIER
FIN ENREGISTREMENT
```

Question 2 : Algorithme gestion

Analyse

La succession des tâches à réaliser est clairement précisée dans l'énoncé.

Saisie de m : un contrôle de saisie est mis en place, utilisant une boucle **TANTQUE**

Saisie de T : T est un tableau d'enregistrements. T[i] est de type athlete.

Pour la création et le remplissage du tableau T, une boucle de saisie T[i] est nécessaire. Pour chaque valeur de i, il faut saisir les trois champs de T[i].

Recherche : t désigne la valeur de la taille que l'on recherche. Pour effectuer cette recherche, on compare le champ taille de l'enregistrement T[i] à t. En cas d'égalité, une variable booléenne trouve, initialisée à **FAUX**, est mise à **VRAI**. On sort de la boucle et on affiche tous les champs de cet enregistrement T[i]. Sinon, la recherche continue jusqu'à ce qu'on trouve le premier enregistrement qui convient, ou la fin du tableau T.

La boucle de recherche adoptée est la boucle **TANTQUE**.

Variables :

i : compteur de boucle ,

T : tableau dont les éléments sont de type athlete,

m : nombre d'éléments de T, de type **ENTIER**,

ta : la taille à chercher, de type **ENTIER**.

trouve : variable de type **BOOLEEN** servant au test de boucle.

```
PROGRAM gestion
TYPE athlete = ENREGISTREMENT
        num : CARACTERE * 8
        taille : ENTIER
        poids : ENTIER
FIN ENREGISTREMENT
```

```
VARIABLE
        i, m : ENTIER
        T : TABLEAU [1 :120] DE athlete
        ta : ENTIER
        trouve : BOOLEEN
DEBUT
        (* saisie du nombre d'athlètes *)
        AFFICHER( " donner le nombre d_athletes a saisir ")
        LIRE(m)
        TANTQUE ((m< 2) OU (m>120)) FAIRE    (* contrôle de la valeur de m *)
              AFFICHER(" mauvaise valeur de m, redonner m ")
        FINTQUE
        (* creation du tableau T *)
        POUR i ← 1 JQUA m FAIRE
                AFFICHER ( " donner le numero de l_athlete")
                LIRE  ( T[i].num )
                AFFICHER ( " donner la taille de l_athlete")
                LIRE ( T[i].taille )
                AFFICHER  ( " donner le poids de l_athlete")
                LIRE ( T[i].poids )
        FPOUR
        AFFICHER ( " donner la taille a chercher ")  (* saisie de la taille à chercher *)
        LIRE ( ta )
        (* Recherche *)
        trouve  ← .FAUX.
        i← 0
        TANTQUE ( ( NON trouve ) ET ( i < m)) FAIRE
        i←i+1
        SI ( ta=T[i].taille) ALORS trouve ← .VRAI.
        FSI
        FINTQUE
        (* résultat de la recherche *)
        SI ( trouve ) ALORS      (* le premier enregistrement tel que taille=t est trouvé *)
                      DEBUT
                        AFFICHER (" numero=", T[i].num )
                        AFFICHER (" taille=", T[i].taille )
                        AFFICHER (" poids=", T[i].poids )
                      FIN
                     SINON   AFFICHER ( " aucun athlete ne correspond a cette taille ")
        FSI
FIN
```

Exercice 23

Traduire en pascal l'algorithme ci-dessus de l'exercice 22

2°) Les Fichiers

Un autre type de stockage des enregistrements existe : le fichier (d'enregistrements), qui constitue aussi une structure de données.

a) Définition

Un fichier est une structure de données dont les éléments sont des « enregistrements » **gardés en mémoire de masse**.

b) Remarques

[1] Le mot « enregistrement » signifie ici une donnée qui est enregistrée et non nécessairement une donnée de type **ENREGISTREMENT** que l'on a enregistrée.
La donnée enregistrée peut être une donnée de type texte, ou d'un autre type.
[2] Pour toute la suite nous envisageons de ne traiter que des données de type **ENREGISTREMENT** que l'on aura enregistrées dans un fichier.
[3] Pour toute la suite donc, un **FICHIER** est une structure de données dont les éléments sont des **ENREGISTREMENTS**.
[4] Le fait de déclarer un type nouveau qui est un **ENREGISTREMENT** entraine automatiquement au niveau du langage la création d'un type **FICHIER** accompagnant le type **ENREGISTREMENT.**

c) Exemple

La déclaration

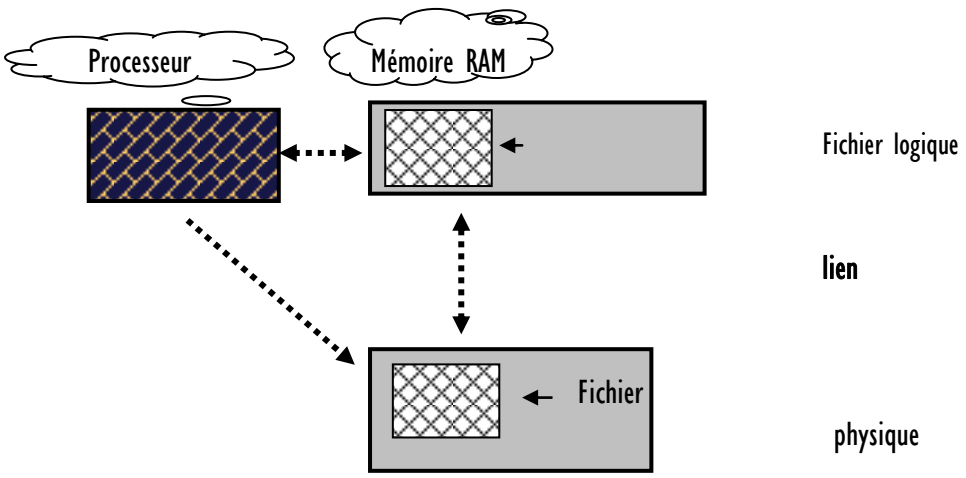
```
TYPE athlete = ENREGISTREMENT
        num : CARACTERE * 8
        taille : ENTIER
        poids : ENTIER
FIN ENREGISTREMENT
```

entraine la création du type **FICHIER DE** athlete

d) Fichier et mémoire de masse : principe de fonctionnement.

Un fichier est prévu pour être gardé en mémoire de masse. Il est donc prévu pour résider en permanence sur un support tel qu'un disque dur ou une clé USB, par exemple.

Qu'est ce qui se passe quand on désire travailler sur un fichier donné ?



Un fichier doit être ouvert avant qu'on puisse travailler là-dessus.
L'ordinateur travaille non pas directement avec le fichier physique (qui est en mémoire de masse) mais avec un copie de celui-ci (un fichier logique) mis en mémoire RAM dès son ouverture.
A l'ouverture, un lien logique est créé et maintenu durant toute la durée de la session de travail.
Tout travail non abouti est effectué sur le fichier logique. Et dès qu'un résultat devient définitif, on enregistre réellement celui-ci sur le fichier physique.
A la fin de la session, on ferme le fichier. Cela consiste à couper le lien logique entre fichier physique et fichier logique.

3°) Organisations d'un fichier

L'organisation d'un fichier est la manière avec laquelle sont rangées et manipulées les données enregistrées dans ce fichier.
Il y a trois types d'organisation, qui donnent leur nom au fichier concerné:

- L'organisation séquentielle (fichier à accès séquentiel)
- L'organisation directe (fichier à accès direct)
- L'organisation séquentielle indexée (fichier à accès séquentiel indexé)

a) Organisation séquentielle

Avec ce type d'organisation, les enregistrements sont rangés les uns après les autres et sont accessibles selon la méthode FIFO (Premier entré Premier sorti)

b) Organisation directe

Les données sont rangées de manière aléatoire et repérées par leurs positions

c) Organisation séquentielle indexée

On utilise deux types de fichiers

- le fichier où sont enregistrées de manière aléatoire les données (fichier des données)
- un ou plusieurs fichiers d'index permettant de retrouver les enregistrements

d) Remarque

Pour toute la suite, on s'intéressera aux seuls fichiers d'ENREGISTREMENTS à accès séquentiel.

e) Méthode d'accès aux fichiers séquentiels

Pour pouvoir travailler sur un enregistrement donné, il faut avoir parcouru tous les autres enregistrements précédents.
L'avancement est déterminé par un index qui est en position initiale (sur le premier enregistrement, s'il existe) à l'ouverture d'un tel fichier.

f) Opérations sur les fichiers

[1] Ouverture d'un fichier

Pour ouvrir un fichier, il faut donner son nom physique, le nom logique qu'on lui donne, et le type de « travail » que l'on veut effectuer.
Il y a trois modes de travail :

- le mode ECRITURE (ou CREATION) : "**W**" ou "**E**". Il s'agit de créer un fichier.

- Le mode LECTURE : "**R**" ou "**L**"
- Le mode MIS A JOUR : "**W/R**" ou "**E/L**"

Syntaxe :

OUVRIR (<nom logique ou nom interne>, <nom physique ou nom externe>, <mode>)

Exemple : **OUVRIR**(fichclient, "H:\donnees\stock2010.dat","**L**")
pour ouvrir en mode lecture le fichier de nom logique **fichclient**, de nom physique :

H:\donnees\stock2010.dat.

[2] **Fermeture**

A la fin d'une session, on doit toujours fermer le fichier.

Syntaxe : **FERMER**(<nom logique>)

[3] **Suppression**

Il s'agit ici d'une suppression du fichier physique (suppression définitive), qui est possible seulement pour des fichiers qui sont fermés.

Syntaxe : **SUPPRIMER**(<nom physique>)

g) **Notion de fin de fichier**

A l'ouverture ou après une série de recherche, l'index peut être situé en foin de fichier. Dans ce cas, on ne peut plus poursuivre toute recherche. Et il faut le savoir.
Une fonction existe, à valeur booléenne, qui permet de gérer cette situation : la fonction **FDF** ou **EOF** .

Syntaxe **FDF**(<nom logique>) qui est .**VRAI**. si on est en fin de fichier, et .**FAUX**. sinon.

h) **Opérations à l'intérieur d'un fichier**

[1] **Lecture d'un enregistrement**

Syntaxe : **LIRE** (<nom logique>, <variable enregistrement>)

[2] **Insertion d'un enregistrement.**

Syntaxe : **ECRIRE** (<nom logique>, <variable enregistrement>)

[3] **Modification d'un champ d'un enregistrement.**

Syntaxe : **REECRIRE** (<nom logique>, <variable enregistrement>)

Remarque : On ne modifie jamais la clé primaire

[4] Suppression d'un enregistrement.

Syntaxe : **SUPPRIMER** (<nom logique>, <variable enregistrement>)

[5] Parcours séquentiel

Principe : Il faut placer l'index en première position et tester si on est en face d'un enregistrement ou en fin de fichier.

Syntaxe

```
LIRE (<nom logique>, <variable enregistrement>)
TANTQUE(NON FDF(<nom logique>) FAIRE

      LIRE(<nom logique>, <variable enregistrement>)
FINTQUE
```

Remarque : ce bout de code peut être utilisé quand on parcourt le fichier et qu'on veuille y faire des opérations.
On aurait alors :

```
LIRE (<nom logique>, <variable enregistrement>)
TANTQUE(NON FDF(<nom logique>) FAIRE
      ..................
      ..................  (* opérations*)
      ..................
      LIRE(<nom logique>, <variable enregistrement>)
FINTQUE
```

Exemples

[1] CREATION D'UN FICHIER

Créer un fichier (fich_stock.dat situé à la racine d'une disquette) de produits considérés comme enregistrements dont les champs sont :

- son nom (nom : chaine de 10 caractères)
- son code produit (clé primaire) (**codeprod** : chaine de 8 caractères)
- le prix (**prix** : entier)
- la quantité en stock (**qstock** : entier)

En algorithmique

```
PROGRAM   creation
   TYPE produit=ENREGISTREMENT
     nom : CHAINE DE CARACTERE[10]
     codeprod : CHAINE DE CARACTERE[8]
     prix :ENTIER
     qstock : ENTIER
   FIN ENREGISTREMENT

VARIABLE
     i : ENTIER
     e_produit : produit
     f_produit : FICHIER DE produit
```

```
      encore : CARACTERE
DEBUT
    OUVRIR(f_produit,"A:\fich_stock.dat","W")
     i←0
    REPETER
          i←i+1
       AFFICHER("donner le nom du ",i,"eme produit")
       LIRE(e_produit.nom)
       AFFICHER("donner le code du ",i,"eme produit")
       LIRE(e_produit.codeprod)
       AFFICHER("donner le prix du ",i,"eme produit")
       LIRE(e_produit.prix)
       AFFICHER("donner la quantite en stock du ",i,"eme produit")
       LIRE(e_produit.qstock)
       ECRIRE(f_produit, e_produit)
       AFFICHER("encore un enregistrement ?o/n")
       LIRE(encore)
       JUSQUA(encore='n')
       AFFICHER("vous avez saisi ",i,"enregistrements")
       FERMER(f_produit)
FIN.
```

[2] EDITION D'UN FICHIER

Le fichier fich_stock.dat est créé. Editer ce fichier.

En algorithmique

```
PROGRAM   edition
    TYPE produit=ENREGISTREMENT
      nom : CHAINE DE CARACTERE[10]
      codeprod : CHAINE DE CARACTERE[8]
      prix :ENTIER
      qstock : ENTIER
    FIN ENREGISTREMENT

VARIABLE
     e_produit : produit
     f_produit : FICHIER DE produit

DEBUT
    OUVRIR(f_produit,"A:\fich_stock.dat","L")
    LIRE(f_produit,e_produit)
    AFFICHER("Nom : "," Code "," Prix :"," Qutite")

    TANQUE(NON FDF(f_produit)) FAIRE
    AFFICHER(e_produit.nom, e_produit.codeprod,e_produit.prix,e_produit.qstock)
```

```
    FERMER(f_produit)
FIN.
```

[3] MISE A JOUR D'UN FICHIER

On se propose de modifier un des champs d'un enregistrement dans un fichier.
Il s'agit donc de d'abord rechercher l'enregistrement concerné et de modifier un de ses champs (qui ne soit pas la clé primaire). Prenons par exemple le champ qstock.

```
PROGRAM   mise_a_jour
TYPE produit=ENREGISTREMENT
      nom : CHAINE DE CARACTERE[10]
      codeprod : CHAINE DE CARACTERE[8]
      prix :ENTIER
      origine : CHAINE DE CARACTERE[15]
      qstock : ENTIER
FIN ENREGISTREMENT
VARIABLE
      e_produit : produit
      f_produit : FICHIER DE produit
      code :CHAINE DE CARACTERE[8]

DEBUT
      OUVRIR(f_produit,"A:\fich_stock.dat","E/L")
      AFFICHER("donner le code du produit a chercher")
      trouve←.FAUX.
      LIRE(code)
      LIRE(f_produit,e_produit)
      TANQUE(NON FDF(f_produit) ET NON trouve) FAIRE
            SI (e_produit.code=code) ALORS
                  DEBUT
                        AFFICHER("enregistrement trouve")
                        AFFICHER("donner la nouvelle quantite:")
                        LIRE(e_produit.qstock)
                        trouve←.VRAI.
                        REECRIRE(f_produit, e_produit)
                        AFFICHER("le champ stock du produit", e_produit.nom,
                        "a été mis a jour")
                  FIN
            FSI
            LIRE(f_produit,e_produit)
      FINTQUE
      SI(NON trouve ) ALORS AFFICHER("produit inexistant, aucune mise a jour effectuee")
      FERMER(f_produit)
FIN.
```

SERIE N°4 : Enregistrements et Fichiers

Serie4_1

1°) On veut créer un nouveau type de données relatives aux clients d'un magasin donné qui est un enregistrement appelé client dont les champs sont:

- le nom (nom du champ : ***nom***) : une chaîne de 20 caractères au maximum
- le prénom (***prenoms***) : une chaîne de 30 caractères au maximum
- le numéro client (***numcli***): un entier
- l'adresse (***addr***) : une chaîne de 20 caractères au maximum

a) Préciser la clé qu'on peut prendre pour cet enregistrement.

b) Donner la déclaration à faire pour ce nouveau type.

2°) On veut ranger les 80 clients fidèles d'un magasin dans un tableau TABCLI.

a) Ecrire un algorithme qui saisit ces clients et les place dans le tableau TABCLI.

b) Ecrire un algorithme qui restitue tous les renseignements disponibles concernant le client N°237 et qui donne après tous les numéros des clients qui portent le même nom que le client N°237.

Serie4_2

On désire créer un fichier de points matériels du plan. Un point est caractérisé par son nom, son abscisse et son ordonnée. Un point sera considéré comme un enregistrement dont les champs sont :

Nom : chaine de 2 caractères,
Abs : réel,
Ord : réel.

1°) Donner la clé primaire de cet enregistrement. Justifier.

2°) Déclarer un type nommé **point** pour cet enregistrement.

3°) On veut enregistrer des points dans un fichier nommé **base_points.dat**, localisé dans le dossier **donnee** situé à la racine d'une disquette.

a) Ecrire un algorithme **CreeFich** qui crée ce fichier en enregistrant les points ci-dessous:
M1(2, -4) ; M2(-3 , 7) , M3(5, -5) , M4(2, 6) , M5(5, 5)

b) Écrire un algorithme qui lit le fichier **base_points.dat** et qui:

- recherche le point M2, change son ordonnée (Ord=15) s'il a été trouvé,
- calcule l'équation de la droite des extrêmes ajustant le nuage formé par ces points.
- ajoute un autre point M6(-4, 7)
- liste tous les points enregistrés sous la forme nom (Abs, Ord)

Serie4_3 (Examen MIAGE)

On veut créer un nouveau type de données relatives aux candidats à un concours donné. Un candidat est un enregistrement appelé candidat dont les champs sont: le nom et un prénom (nom du champ: ***nomprenom***) : une chaîne d'au plus 20 caractères , le numéro d'inscription (***numero***) : un entier, la note finale (***note***) : un réel

1°) Préciser la clé qu'on peut prendre pour cet enregistrement et donner la déclaration à faire pour ce nouveau type.

2°) Ecrire un programme (PROG1) qui crée un fichier nommé ***concours_2009,*** stocké dans le dossier EXAM à la racine d'une disquette, et qui contient les enregistrements des 280 candidats à ce concours.

3°) Ce fichier étant déjà crée, écrire un algorithme (PROG2) qui vérifie si le numéro 165 est bien celui du candidat Kouakou Olivier puis affiche sa note dans l'affirmative, et enfin, à la suite de cette opération, corrige son nom (Kakou Olivier au lieu de Kouakou Olivier).

Serie4_4

Le cabinet Picsou est chargé de faire l'état des salaires des agents de la SCESA à la fin de chaque mois. Il utilise pour cela deux fichiers :

- IdAgent, fichier contenant les informations utiles concernant chaque agent : Matricule, Nom, Adresse, Service, Fonction.
- SalAgent , fichier des salaires dont les champs sont : Matricule, SalBrut, SalNet, DatePaie. La retenue sur salaire (retenue 1) est de 3%, appliquée sur le salaire brut pour avoir le salaire net.

1°) On s'intéresse au fichier IdAgent. On veut créer ce fichier et l'enregistrer dans le répertoire SCESA à la racine du disque dur E : sous le nom Fich_IdAgent.

a) Ecrire la déclaration du type d'enregistrement à utiliser pour créer ce fichier.

b) Quel(s) champ(s) peut-on prendre comme clé primaire pour ce fichier ? Justifier.

b) Ecrire un programme creeAgent qui crée le fichier IdAgent et l'enregistre dans le dossier SCESA du disque E.

2°) Ecrire un programme creeSalaire qui crée le fichier Fich_SalAgent et l'enregistre dans le même répertoire SCESA du disque E :

3°) a) Ecrire une procédure editAgent qui permet d'éditer le fichier IdAgent et le fichier Fich_SalAgent.

b) Ecrire une procédure majAgent qui permet de faire une mise à jour du fichier IdAgent et du fichier Fich_SalAgent.

c) Ecrire une procédure supprAgent qui permet de supprimer un agent des fichiers.

4°) Les fichiers IdAgent et Fich_SalAgent sont déjà crées. Ecrire un programme principal qui

- édite le fichier IdAgent,
- liste tous les agents embauchés après 2004,
- liste tous les agents dont les salaires nets sont au-dessus de 400000F,
- supprime dans les deux fichiers l'agent de N°matricule 123ER45 dont le nom est Kouakou,
- modifie les salaires de ceux dont le salaire net est supérieur ou égal à 600000F en leur faisant une retenue supplémentaire de 0,5% (retenue 2) sur leur salaire brut.

5°) Ecrire un programme principal appelé EDITPAIE qui édite la fiche de paie des agents de cette entreprise chaque fin du mois :

Cette fiche de paie doit comprendre :

- les nom et adresse et N° matricule de l'agent
- le salaire brut
- le nombre de retenues (1 ou 2)
- le montant des retenues
- le salaire net
- la date de paie

Format de sortie de la fiche de paie:

ENTREPRISE SCSA
Fiche de paie

Mois:…….. Année…….....
Service :……..
Nom :…….
Matricule :……
Adresse :……
Année d'embauche :……..
Fonction :……
Nombre de retenues :……
Montant total des retenues :……...
Salaire brut :……

Salaire net du mois :……