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 proposition de démarche :

- 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 indique une succession de tâches à réaliser. C'est déjà un algorithme.

Essayons d'en dégager les premières caractéristiques :

- Il est écrit en utilisant le langage usuel : les mots sont ici ceux du vocabulaire français (il est adressé à des personnes comprenant le français).
- Il décrit **une succession de taches à exécuter,** des étapes à suivre donc, et précise en même temps **un** ordre d'exécution de ces taches.
- Le nombre de taches à exécuter est en nombre fini.
- Les taches à exécuter doivent être assez explicite (les mots utilisés doivent ainsi être assez précis) pour que celles-ci puissent être exécutées par « n'importe qui ». Ce qui n'est pas le cas, par exemple, de la tache « enlever la roue crevée », qui reste après tout compréhensible par celui qui sait déjà enlever une roue de voiture.

Ecrire correctement un algorithme c'est se conformer au moins à ces obligations.

Un algorithme est une description d'une succession de taches (d'instructions) à exécuter pour une résolution d'un problème donné

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 }

Problème posé :

Proposer une description d'une succession de taches à réaliser pour ranger cette liste par ordre croissant.

Il s'agit donc d'écrire un algorithme de rangement (on dit aussi tri) d'une liste de nombres.

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 », qui ne sont pas nécessairement des nombres.
(à faire en exercice)

Un algorithme doit être conçu pour une résolution d'une famille de problèmes, et non pour résoudre un problème très particulier.

2°) Définitions

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

Algorithme correct : Un algorithme est **correct** si le résultat obtenu après son application est toujours celui escompté.

Objets : Un algorithme manipule **des objets**.

Exemples :
Pour l'exemple N°1 : une roue crevée, un crick, une roue de secours (du même type que la roue crevée) etc..
Pour l'exemple N°2 : une liste d'objets de même nature, le nombre n de ces objets.

3°) Propriétés

- ✓ 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 :

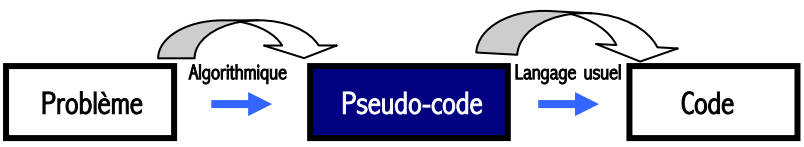
- une rapidité d'exécution (le temps de calcul est souvent facturé et très cher),
- une occupation de l'espace mémoire satisfaisante.

4°) Langage

Quand deux entités veulent communiquer, pour se comprendre, elles doivent adopter un langage de communication.
Un algorithme décrit les étapes adoptées pour résoudre un problème donné en utilisant un ordinateur.
Un algorithme donne des instructions.
Pour tout cela, il utilise des mots d'un vocabulaire, celui du langage appelé **algorithmique**.
Le vocabulaire de ce langage est issu du vocabulaire du langage commun sur lequel on a imposé des restrictions.

5°) 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 algorithme est un programme écrit en algorithmique. C'est un **pseudo code** car il est en réalité destiné à une machine virtuelle qui est l'homme, et non à un ordinateur. Il n'est pas soumis aux rigueurs strictes des langages de programmation (rigueur du vocabulaire , de la syntaxe, de la grammaire).



Remarque : Des logiciels d'apprentissage de l'algorithmique existent, proposés surtout aux lycéens (Algobox – Xcas etc...)
L'algorithmique devient alors dans ce cas un véritable langage.

Ces logiciels sont gratuits, téléchargeables sur Internet.

Objet :
Nous avons dit plus haut qu'un algorithme manipule des objets.
Un objet peut être :

- un objet demandé en début de processus : ce sont des **données**. La roue de secours, la roue crevée, le cric, etc… sont des données. La liste de l'exemple 2 , ainsi que sa taille n, sont des données.
- 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.

6°) 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 **déclarative**, où on spécifie les caractéristiques de tous les objets manipulés.
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.**

Remarque importante :
L'algorithmique est un langage de communication, différent d'un langage de programmation.
Un langage de programmation est très rigide et très strict. Il faut être capable de respecter scrupuleusement le vocabulaire, la syntaxe pour pouvoir écrire correctement un programme dans un langage donné.
L'algorithmique est un peu plus souple. Et on peut même dire qu'on peut avoir autant d'algorithmique que de groupes de personnes décidant de travailler et programmer ensemble.

L'algorithmique présenté ici est une présentation parmi tant d'autres. Mais à partir du moment où on l'a adopté, il s'agit de s'y conformer strictement.

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
```

Convention d'écriture : **PROGRAM** <nom de l'algo > : on met à la place de <.....> le nom du programme .

Exemple : **PROGRAM** calc_moyenne

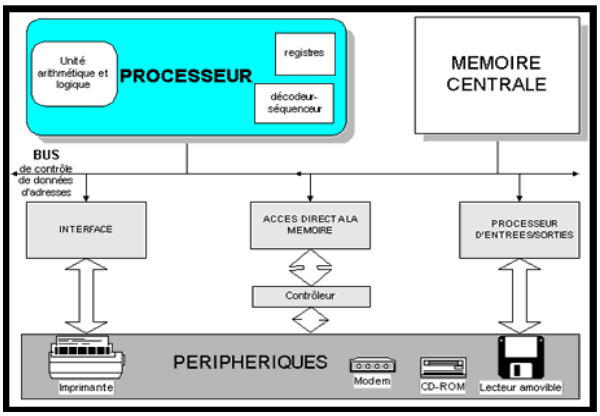
7°) Architecture

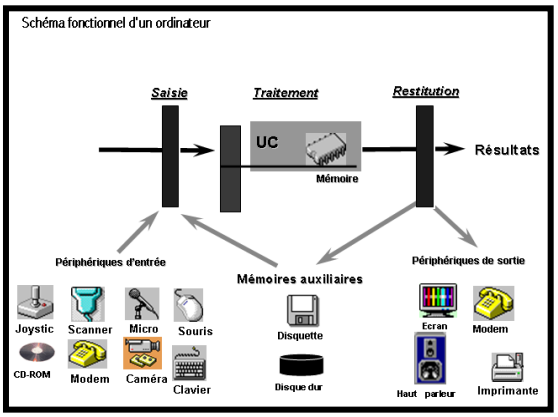
Un algorithme est destiné à être traduit dans un langage donné, puis exécuté par un ordinateur. Il travaille avec des données et sort des résultats. Ces données auront à être stockées en mémoire pour qu'on puisse les (ré)utiliser.
D'où la nécessité de bien comprendre l'architecture d'un ordinateur.

Architecture d'un ordinateur

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

Architecture de base actuelle et schéma fonctionnel d'un ordinateur





Un ordinateur travaille avec des mots de longueur fixe (mot de n bits = bloc de n bits, n etant un nombre predefini, que l'ordinateur utilise pour coder les informations). Un ordinateur 16 bits utilise des mots de 16 bits, c'est-à-dire 2 octets. 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, seul le **nom** donné au **mot** (ou au groupe de mots nécessaires) est connu et a son importance. Alors qu'avec un langage de programmation usuel, un **mot** est caractérisé par son **nom** et son **adresse** indiquant son emplacement exact dans la mémoire.
Les données peuvent être sauvegardés 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 incombe au **système d'exploitation** .
A un système d'exploitation correspond un système de fichiers bien déterminé.
Les informations doivent être codées.
Plusieurs codages sont utilisés dont :

- 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
- ……

8°) Objets manipulés par une algorithme

a) Caractéristiques d'un objet

Un objet manipulé par un algorithme doit être, pour un algorithme, parfaitement défini avant son utilisation. Et 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 caractéristique de ce qu'il doit contenir :(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 : **NON** ; **OU** ; **ET**

Table de vérité : A et B sont des propriétés.

A	B	**NON** A	A **OU** B	A **ET** B
.**VRAI**.	.**VRAI**.	.**FAUX**.	.**VRAI**.	.**VRAI**.
.**VRAI**.	.**FAUX**.	.**FAUX**.	.**VRAI**.	.**FAUX**.
.**FAUX**.	.**VRAI**.	.**VRAI**.	.**VRAI**.	.**FAUX**.
.**FAUX**.	.**FAUX**.	.**VRAI**.	.**FAUX**.	.**FAUX**.

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 (en simple précision) , les entiers longs (en double précision) , les réels simples (en simple précision) , les réels en double précision . Le choix dépend de l'ordre de grandeur de la valeur de la variable ou de la constante.

Exemple: une constante entière : 2876 ;
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) , les différents codes d'opérations, de ponctuations et d'autres codes tels $, & ; …. . En fait tout ce qui est répertorié dans le tableau des codes ASCII.
Ensemble des opérateurs: **CODE**() ; **CAR**() ; **PREC**() ; **SUCC**() ;

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**()….

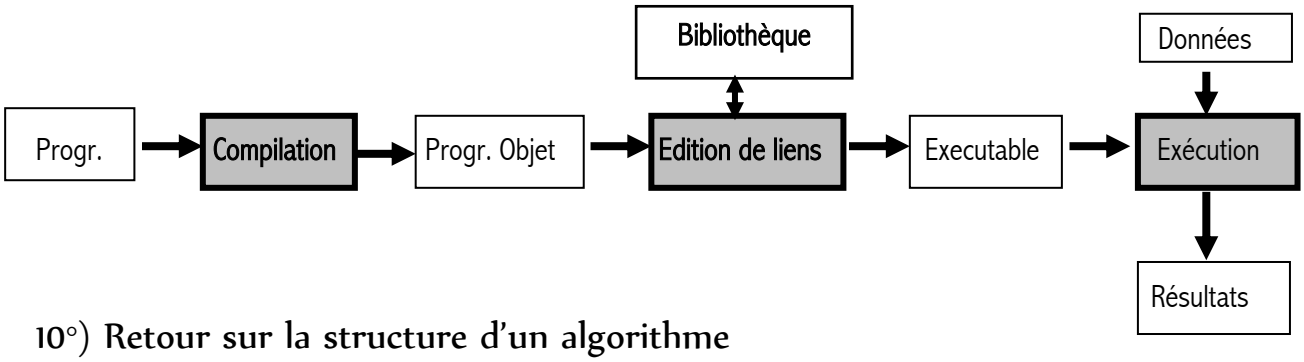
Exemples :
Ecriture d'une constante **CARACTERE** : 'm' , ' ?' , '*', '"'.
Ecriture d'une constante **CHAINE DE CARACTERE** : ''voiture'', ''c''.
LONGUEUR(''voiture'') =7 ;
''ma ''// ''maman'' = ''ma maman'' ;
SSCHAINE (''maman'', 2,3)=''ama''

Attention : 'a' est de type **CARACTERE** alors que ''a'' est de type **CHAINE DE CARACTERE** .

9°) Traitement subi par un algorithme

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.
Un programme, une fois écrit dans un langage compilé par exemple, subit, dans l'ordre :

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



10°) 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, il se termine 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 cet algorithme :

- identificateur et type pour les variables,
- identificateur et valeur pour les constantes.

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

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=9.87676543456678765  ( * 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 et une bonne compréhension 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 : 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 4 grandes familles, les seules qu'un ordinateur est capable de traiter :

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

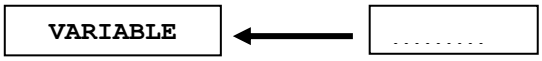
Ces 4 grandes familles constituent les instructions de base d'un algorithme.

Remarque : Ordre d'exécution des instructions
Dans un algorithme, les instructions sont exécutées de manière séquentielle. L'ordre de ces instructions est donc essentiel.

1°) Instruction d'affectation
Déclarer une variable (ou une constante) c'est réserver un emplacement en mémoire, destiné 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 puisque 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 donc 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 vaut 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*3+2*a^3
FIN
```

Réponse

a=12	a=34	a=22

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 (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 de type **CHAINE DE CARACTERE** (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.

2°) 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. Cela se fait avec 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 par 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
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.

3°) Les tests : Structures conditionnelles.

a) Opérateurs relationnels

=	égalité
<	inférieur strictement
<=	inférieur ou égal
>	Supérieur strictement
>=	supérieur ou égal
<>	différent (on peut aussi mettre ≠)

On distingue les structures suivantes:

b) La structure conditionnelle.

Syntaxe :

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

Si la condition est vérifiée, instruction1 est réalisée, 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 encadre 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
```

c) 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 : impossible!!!'')
                  SINON AFFICHER ( ''le quotient de '', a, ''par '',b,'' est :'', a/b )
        FSI
FIN
```

d) 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  '',-b/a)
   FSI
FIN
```

e) 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, 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

p, s et q : des variables caractères représentant 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
```

4°) 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 est 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 des nombres au clavier jusqu'à ce qu'il y ait vingt (20) nombres strictement positifs, et calcule la somme de ces 20 nombres en utilisant **TANTQUE**.

Analyse : 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
VARIABLE        j : ENTIER
                som, saisie : REEL
DEBUT
   som←0
   j←0
   TANTQUE j<20 FAIRE
      AFFICHER ("donner un reel ")
      LIRE(saisie)
      SI (saisie>0) ALORS
                    DEBUT
                        j←j+1
                        som ← som+saisie
                    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
   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].

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.

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, constant durant la boucle. La valeur par défaut est 1.
- ✓ Si la valeur de **PASDE** est positive, val_fin doit être supérieure à val_init, sinon la boucle n'est pas exécutée.

Exemples :

[1] Ecriture d'un algorithme calculant la moyenne de 20 nombres saisis au clavier et affichant cette moyenne.
[2] Programme qui donne la 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.

SERIE N°1

Les Instructions de base : Affectation – Entrée/Sortie – Tests – Boucles

SERIE_1_1

Tracer les algorithmes suivants :

```
PROGRAM exo11
VARIABLE
        a, b, c : ENTIER
DEBUT
        a←1
        b←4
        c←a+b
        a←2
        c←(c+b-2*a) DIV a
        b←c MOD a
FIN
```

```
PROGRAM exo12
VARIABLE
        a, b, c : REEL
DEBUT
        a←2
        a←a+2
        b← a*2+a
        c←4
        c←b-c
                c←c+a-b
        a←b-c*a
        a←(b-a)*c
        b←(a+c)*b/a*c^2
FIN
```

SERIE_1_2

Ecrire un algorithme qui saisit deux réels a et b, les affiche avec le format « le contenu de a est …….. » et « le contenu de b est……. », qui échange les deux contenus lorsque ceux-ci sont différents, et affiche dans ce cas les nouveaux contenus selon le même format.

SERIE_1_3

1°) On achète une certaine quantité d'un produit. Ecrire un algorithme

- qui saisit le prix Hors Taxe (PHT) d'une unité de ce produit ainsi que la quantité achetée,
- qui calcule le montant total de l'achat en PTT (Prix Toutes Taxes), sachant que la TVA appliquée est de 18,6%,
- qui calcule et affiche le prix à payer (PAP) tenant compte du fait qu'un montant TTC supérieur à 20000 F fait bénéficier d'une remise de 5%.

2°) Ecrire un algorithme 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 qui demande un entier N et un entier positif S, affiche N et les S prochains nombres consécutifs suivant N ainsi que leur somme.

SERIE_1_4

Ecrire un algorithme permettant de résoudre dans l'ensemble des nombres complexes, l'équation $ax^2+bx+c=0$, où a, b et c sont des réels.

SERIE_1_5

1. a, b, c et d sont des entiers. Quelle est l'affirmation contraire de l'affirmation
   « a=b ou a=c ou a=d ou b=c ou b=d ou c=d » ?
2. On admet que l'opérateur **ALEA**(n) donne de manière aléatoire un entier compris entre 1 et n (au sens large).
   On considère l'algorithme suivant : (; *est ici un séparateur d'instructions lorsque celles-ci sont écrites sur une même ligne*)

```
PROGRAM aleatoire
VARIABLE
        n, i, a, b, c, d : ENTIER
DEBUT
        AFFICHER(" Donner un entier n strictement positif, avec n > 4")
        LIRE(n)
        i←0
        a←0 ; b←0 ; c←0 ; d←0
```

```
        TANTQUE(a=b OU a=c OU a=d OU b=c OU b=d OU c=d ) FAIRE
        a←ALEA(n) ; b←ALEA(n) ; c←ALEA(n) ; d←ALEA(n)
        i←i+1
        FINTQUE
        AFFICHER (a,'' '',b,'' '',c,'' '',d)
        AFFICHER ('' au bout de '',i,'' fois'')
FIN.
```

a. Les affichages suivants peuvent-ils être issus de cet algorithme ?
   Affichage1 : 1 5 12 7 10 ***Affichage2***: 6 4 9 4 ***Affichage3***: 4 6 17 9
b. Que fait exactement cet algorithme ? Justifier.
c. Vous êtes responsable du contrôle anti-dopage d'une course cycliste célèbre et on vous propose cet algorithme. A quoi peut-il vous servir ?

SERIE_1_6

1°) Ecrire un algorithme permettant la saisie d'une note, son affichage à l'écran, et l'affichage d'un message si la note est en dessous de la moyenne.
2°) Ajouter à l'algorithme de la question 1°) une vérification de la note (elle doit être comprise entre 0 et 20). Une note incorrecte ne sera pas affichée à l'écran, le message « erreur de saisie » sera affiché à la place.
3°) Ecrire un algorithme demandant le nombre N de notes à saisir, saisit ces N notes en effectuant une vérification de celles-ci, et affiche leur moyenne. Prévoir le cas accidentel où N a été mis à 0.

SERIE_1_7

Soit f une fonction définie continue sur l'intervalle [a ; b]. On suppose que si α et β sont deux solutions de l'équation f(x)=0, alors $|\alpha-\beta| > 0{,}1$.
Ecrire un algorithme qui saisit deux réels a et b, avec $a \leq b$, qui compte le nombre de solutions de l'équation f(x)=0 dans [a, b] et qui donne une valeur approchée à 10^{-3} près de ces solutions.

SERIE_1_8

Soient les deux suites : (U_n) et (V_n) définies par : $\begin{cases} V_0 = 0 \\ V_{n+1} = \sqrt{\dfrac{1+V_n}{2}} \end{cases}$ et $\begin{cases} U_0 = 2 \\ U_{n+1} = \dfrac{U_n}{V_{n+1}} \end{cases}$. La théorie affirme que la suite (U_n) converge vers π. Ecrire un algorithme qui permet de vérifier cette assertion. La convergence est-elle rapide ?

SERIE_1_9 :

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.

B-2 : Les tableaux

Les données sont rangées en mémoire et c'est le système d'exploitation (S.E.) qui s'en charge.
Nous nous contentons de donner des noms à nos variables et constantes. C'est le S.E qui s'occupera de leur donner les adresses.
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 de deux classes 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, …….).
Intéressons-nous par exemple à la liste des élèves : il est plus commode pour lui de présenter les élèves d'une classe en les regroupant et constituer ainsi une structure de données nouvelle : les **tableaux,** ici à une dimension.

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 particulier de donnée, que l'on doit déclarer avant d'être utilisé.

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 donc :

<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 :15**] 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. On acceptera au plus 15 élèves dans cette classe.

Exemples : notes : **TABLEAU[**1 : 15**] DE REEL**
passage: **TABLEAU[**1 : 15**] DE BOOLEEN** (* il s'agit de répertorier les états de chaque élève : admis (OUI) ou 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 (* au maximum dans une classe , il y a 35 élèves *)
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)
        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 (* au maximum dans une classe , il y a 35 élèves *)
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)
                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 (* au maximum dans une classe , il y a 35 élèves *)
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 Supprimer ou ajouter un élément d'un tableau (Exercice 13)

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 Trier une tableau (Exercice 14)

Chercher sur Internet des algorithmes de tri.

Ecrire un algorithme du :

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

Exercice 12 :

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

```
PROGRAM tribulles
CONSTANTE n =35 (* au maximum dans une classe , il y a 35 élèves *)
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 Rechercher un élément dans un tableau trié (Exercice 15)

Tenir compte du caractère trié du tableau.

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

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

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

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** (** une classe de 35 élèves au maximum avec au plus 6 notes par élèves **)

Exercice 13

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 14 :

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 15 :

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

SERIE_2_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.

SERIE_2_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.

SERIE_2_3

1°) Saisir 20 réels au clavier et donner le plus grand et le plus petit de ces 20 réels (on n'utilisera 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.

SERIE_2_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.

SERIE_2_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.
Le vocabulaire d'un langage donné est toujours en nombre très limité. Il en est de même de l'algorithmique, même si celui-ci est quand même plus tolérant.

Prenons un exemple :
Soit le cas d'un programme où à chaque fois qu'on affiche un nouveau résultat, on saute des lignes.
Pour sauter k lignes on écrit le bout de codes suivant :

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

Soit 3 lignes de code à écrire à chaque fois.
Si on devait le faire 5 fois dans le programme, on aurait écrit 15 lignes (5 groupes de codes pratiquement identiques).
D'où l'idée d'écrire à côté ce bout de code, de le nommer (***sauterligne(k)*** par exemple, ***k*** étant le nombre de lignes à sauter) et d'appeler ***sauterligne(k)*** 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 ***sauterligne(k).***
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 (ou pas) d'arguments ou paramètres : des données à fournir dès le départ (argument_donnée), ou des variables qui vont contenir des résultats (argument_resultat), ou même des arguments_donnée_résultat (la variable qui contient la donnée va aussi contenir un résultat) .

Exemples :
- Le sous programme qui saute k lignes a besoin qu'on lui fournisse le nombre k de lignes à sauter. C'est un argument_donnée. Il n'a pas besoin d'argument_résultat.
- Un sous programme qui calcule une somme de 2 matrices aura besoin d'arguments_données (les 2 matrices, la taille commune) et d'arguments_résultats (une matrice et sa taille).

b) Types :
On compte deux types de sous programme :
- un sous programme appelé **FONCTION** qui est un sous programme qui fournit obligatoirement un résultat et un seul.
- un sous programme qui n'est pas une fonction, qui est appelé **PROCEDURE**.

3°) Syntaxe d'un sous programme

Un sous programme est d'abord un programme. Sa syntaxe est pratiquement la même que celle d'un programme.

a) Fonction

Puisqu'on est sûr d'obtenir un résultat pour une fonction, la syntaxe et la structure sont:

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

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

Appel d'une fonction :

L'appel doit être contenu dans une instruction d'affectation. Par exemple a← nom_de_la_fonction(liste des arguments fournis)

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 arguments d'une fonction sont des arguments_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.
Une procédure peut aussi ne pas avoir besoin d'arguments.

SYNTAXES :

b-1 : **Procédure sans argument** :

La syntaxe est:

```
PROCEDURE <nom>
(*Partie déclaration des variables*)
                ……..
(*Corps de la procédure*)
  DEBUT
                        ……..
  FIN <nom>
```

Appel d'une procédure:

L'appel se fait de la manière suivante : nom_de_la_procédure

Exemple : Saut de 4 lignes

```
PROCEDURE sauterligne
(* pas de variables à déclarer )
DEBUT
        POUR i allant de 1 JQUA 4 FAIRE
        AFFICHER ('' '')
        FINPOUR
FIN sauterligne
```

L'appel se fera comme suit :

```
……..
……..
 sauterligne
……..
```

b-2 : **Procédure avec arguments** :

Parmi les arguments à préciser pour une telle procédure, il y a les arguments_données, les arguments_résultats. Un argument peut même être à la fois donnée et résultat.

La syntaxe est:

```
PROCEDURE <nom> ( données : liste arguments_données : <type>, arguments_resultats : liste des
arguments_données_résultats :< type >, résultats : liste arguments_resultats :<type>)
(*Partie déclaration des variables*)
                ……..
(*Corps de la procédure*)
  DEBUT
                        ……..
  FIN <nom>
```

Appel d'une procédure:

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

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.
On pourra alors les utiliser pour renvoyer des résultats lors d'un appel de sous-programmes. Cela permettrait ainsi à une fonction de renvoyer plus d'un résultat, à une procédure même sans argument de renvoyer des résultats.

5°) Passage des arguments.

Lorsqu'un programme appelle un sous programme donné, il lui donne les valeurs des arguments nécessaires pour son exécution. Ces arguments 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 variables fournies, qui sont des variables du programme appelant, ne sont donc pas affectées 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 variables qui sont données au sous programme pour travailler. Dans ce cas, toute modification effectuée sur une de ces variables durant l'exécution du sous programme est effective et persiste même en dehors du sous programme.

Attention : Lors de ces passages, les variables fournissant les valeurs requises pour les différents arguments doivent correspondre, en TYPES, en NOMBRE, aux types et au nombre des arguments du sous programme, dans l'ORDRE d'apparition de ceux-ci.

6°) 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 :

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

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 dont un des paramètres est un tableau de type tab120.

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 :

arguments: 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

arguments : 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***

arguments: 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***

arguments : 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

arguments: 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

Serie_3_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.

Serie_3_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.

Serie_3_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.

Serie_3_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.

Serie_3_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.