- PARTIE N°1 -

LES STRUCTURES DE DONNÉES ET LES STRUCTURES SIMPLES

I. Introduction :

Un algorithme appelé aussi pseudo-code est une suite structurée et finie d'actions ou d'instructions pour résoudre un problème.

Structure **:**

ALGORITHME *Nom*

DEBUT

Traitements

FIN

Déclaration des objets

Objet	Type / Nature

II. Les déclarations :

1. Les constantes:

Une constante est un **objet** dont la valeur est **fixe** au cours de l'exécution du programme.

Une constante est caractérisée par :

- Son **nom**
- Sa **valeur**

Exemples :

II=3,14

Déclaration:

Objets	Type
Pi	Constante de valeur 3.14

2. Les variables :

Une variable est un objet dont la valeur est susceptible de **changer** au cours de l'exécution d'un programme.

Une variable est caractérisée par :

- Son **nom**
- Son **type**
- Son **contenu**

III. Les types de données simples:

- ***Le type entier :*** peut être un **nombre** appartenant à un **sous** ensemble de **Z**

  Opérateur : +, -, *, /, MOD et DIV

- ***Le type réel :*** peut être un **nombre** appartenant à un **sous** ensemble de **R**

  Opérateur : +, -, *, /

- ***Le type booléen :*** peut être l'une des valeurs logique **Vrai ou Faux**

  Opérateur : NON, OU, ET

- ***Le type caractère :*** peut être un **alphabet**, un **chiffre** ou un **symbole**.

- ***Le type chaine caractère :*** c'est une suite de **caractères**.

IV. Les Entrées/Sorties:

Pour assurer la communication entre l'utilisateur et la machine, il existe des instructions appelées instructions d'entrées et de sorties ou tout simplement Operations d'entrées/sorties.

1. L'opération d'entrée:

Dans un premier sens, on distingue l'instruction qui permet à l'utilisateur de rentrer ou de saisir des valeurs au clavier pour qu'elles soient utilisées par le programme. Cette opération s'appelle la lecture.

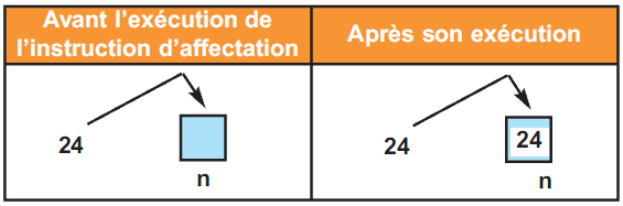
Tout simplement, pour que l'utilisateur entre à l'aide du clavier la nouvelle valeur d'un objet n, on appliquera la syntaxe suivante :

Syntaxe
Lire(NomObjet)

2. L'affectation:

L'action d'affectation est le fait d'attribuer une valeur a un objet donné. L'instruction d'affectation se note avec le signe ←

Ainsi, l'instruction n ← 24 permet d'attribuer la valeur 24 à l'objet n.



La syntaxe de l'instruction d'affectation se présente comme suit :

Syntaxe
NomObjet←valeur

3. L'opération de sortie:

Dans un deuxième sens, on distingue l'instruction qui permet au programme de communiquer des valeurs à l'utilisateur en les affichant à l'écran. Cette opération s'appelle écriture.

Tout simplement, pour que la machine affiche la valeur d'un objet n, vous appliquerez la syntaxe suivante :

Syntaxe
Ecrire("Message",NomObjet,expression)

V. Les fonctions prédéfinies :

a. Les fonctions sur les types numériques

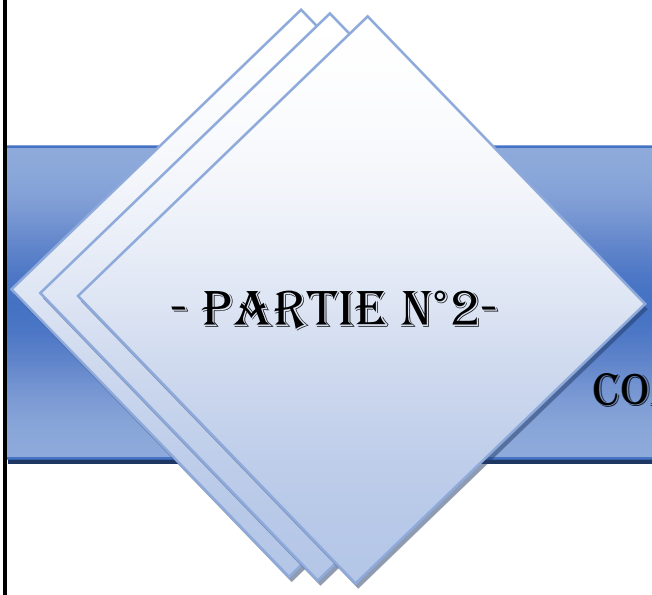
Algorithmique	Rôle
Arrondi (x)	Retourne l'entier le plus proche de la valeur de x.
RacineCarré (x)	Retourne la racine carrée d'un nombre x positif.
Aléa (vi, vf)	Retourne un entier aléatoire de l'intervalle [vi, vf].
Ent (x)	Retourne la partie entière de x.

b. Les fonctions sur le type caractère

Algorithmique	Rôle
Ord (c)	Retourne le code ASCII du caractère **c**.
Chr (d)	Retourne le caractère dont le code ASCII est **d**.

c. *Les fonctions sur le type chaînes de caractères*

Algorithmique	Rôle
Long (ch)	Retourne le nombre de caractères de la chaîne **ch**.
Pos (ch1, ch2)	Retourne la première position de la chaîne **ch1** dans la chaîne **ch2**.
Convch (x)	Retourne la conversion d'un nombre **x** en une chaîne de caractères.
Estnum (ch)	Retourne **Vrai** si la chaîne **ch** est convertible en une valeur numérique, elle retourne **Faux** sinon.
Valeur (ch)	Retourne la conversion d'une chaîne **ch** en une valeur numérique, si c'est possible.
Sous_chaine (ch, d, f)	Retourne une partie de la chaîne **ch** à partir de la position **d** jusqu'à la position **f** (f exclue).
Effacer (ch, d, f)	Efface des caractères de la chaîne **ch** à partir de la position **d** jusqu'à la position **f** (f exclue).
Majus (ch)	Convertit la chaîne **ch** en majuscules.



- PARTIE N°2-

LES STRUCTURES DE CONTRÔLE CONDITIONNELLES

I. Introduction :

Les structures conditionnelles s'appliquent sur des traitements qui ne peuvent être exécutés qu'après l'évaluation d'un ensemble de conditions qui s'effectuent généralement depuis une comparaison afin de retourner une variable booléenne

II. La structure conditionnelle simple:

On distingue 2 structures conditionnelles simples:

- Structure conditionnelle simple réduite si on a un seul traitement.

Syntaxe
Si Condition **Alors** Traitement **FinSi**

- Structure conditionnelle simple complète si on a un deux traitements.

Syntaxe
Si Condition **Alors** Traitement1 **Sinon** Traitement2 **FinSi**

Activité 1:

Ecrire un algorithme permettant de vérifier si un entier n saisie est paire.

Solution :

```
Algorithme parité
Début
        Ecrire("Entrez un entier :")
        Lire(n)
        Si n MOD 2=0 Alors
                Ecrire(n,"est paire")
        Sinon
                Ecrire(n,"n'est pas paire")
        Fin SI
Fin
```

Tableau de Déclaration des objets:

Objets	Type
n	entier

III. La structure conditionnelle généralisée ou alternative:

Syntaxe

```
Si Condition1 Alors
        Traitement1
Sinon Si Condition2 Alors
        Traitement2
Sinon Si Condition3 Alors
        Traitement3
Sinon .....
        .....
Sinon Si conditionN-1 Alors
        TraitementN-1
[Sinon
        TraitementN]
FinSi
```

Activité 2:

Résoudre un problème qui demande un nombre afin de déterminer s'il est négatif, positif ou nul.

Solution :

```
Algorithme parité
Début
        Ecrire("Entrez un entier :")
        Lire(n)
        Si n <0 Alors
                Ecrire(n,"est négatif")
        Sinon Si n >0 Alors
                Ecrire(n,"est positif")
        Sinon
                Ecrire(n,"est nul")
        Fin SI
Fin
```

- PARTIE N°3-

LES STRUCTURES DE CONTRÔLE ITÉRATIVES

I. Introduction :

Les structures itératives s'appliquent sur des traitements qui se répètent que se soient un nombre de fois connus ou non connu.

II. La structure itérative complète :

Boucle Pour ou aussi structure de contrôle itérative complète car elle est utilisée dans le cas où le traitement à faire nécessite une répétition avec le nombre des itérations connu à l'avance.

Syntaxe

```
Pour Compteur de Début à Fin [Pas = valeur_pas] Faire
        Traitement
Fin Pour
```

Interprétations:

1) compteur : variable **compteur**

Début : valeur **initiale**

Fin : valeur **finale**

2) Le nombre de répétition est **connu**

3) Le compteur doit être de type **scalaire**

5) La valeur du compteur sera incrémentée (**+pas**) ou décrémentée (**–pas**) automatiquement.

7) La boucle s'arrête si le compteur atteint la valeur **finale**

Activité 1:

1) Ecrire un algorithme qui permet de déterminer si un entier saisi est parfait ou non.

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

2) Exécutez manuellement la boucle utilisée pour les entiers 5 et 6.

Solution :

```
Algorithme parfait
Début
        Ecrire("Entrez un entier :")
        Lire(n)
        sd←0
        Pour i de 1 à n-1 faire
                Si n mod i=0 Alors
                        sd←s+i
                Fin si
        Fin Pour
        Si n =sd Alors
                Ecrire(n,"est Parfait")
        Sinon
                Ecrire(n,"n'est pas  parfait")
        Fin SI
Fin
```

Tableau de Déclaration des objets:

Objets	Type
n,sd,i	entier

III. *La structure itérative à condition d'arrêt :*

- ***La boucle Répéter……jusqu'à :*** appelée aussi structure itérative à condition d'arrêt car elle est utilisée quand il s'agit de répéter un traitement un nombre de fois **non connu** et qu'on est sûr que le traitement itératif s'exécutera au **moins une fois**

Syntaxe :
Répéter Traitement **Jusqu'à** condition

Activité 2:

Ecrire un algorithme qui permet d'introduire un nombre positif.

Solution :

```
Algorithme Positif
Début
        Répéter
                Ecrire("Entrez un entier positif:")
                Lire(n)
        Jusqu'à n>0
Fin
```

Tableau de Déclaration des objets:

Objets	Type
n	entier

- ***La boucle Tant que……faire:*** appelée aussi structure itérative à condition d'arrêt car elle est utilisée quand il s'agit de répéter un traitement un nombre de fois **non connu** et qu'on est sûr que le traitement itératif s'exécutera **au moins zéro fois**

Syntaxe :
Tant que condition **Faire** Traitement **Fin Tant que**

Activité 3:

Ecrire un algorithme qui permet d'introduire une chaine de caractères afin d'éffacer les espaces superflus.

Solution :

```
Algorithme SuperFlux
Début
                Ecrire("Entrez une chaine:")
                Lire(ch)
                Tant que pos("  ",ch) ≠-1 Faire
                        Effacer(ch, pos("  ",ch), pos("  ",ch)+1)
                Fin tant que
                Ecrire("La nouvelle chaine est :",ch)
Fin
```

Tableau de Déclaration des objets:

Objets	Type
ch	Chaine de caractères

- PARTIE N°4-

LES TABLEAUX

I. Introduction :

Un tableau est une structure de données qui permet de ranger un ensemble de valeurs de même type.

II. Les tableaux à une dimension:

Déclaration 1:

Tableau de Déclaration des objets:

Objet	Type
Nom_tableau	Tableau de N Type _élément

Déclaration 2:

Tableau de création des nouveaux types:

Nouveau type
Nom_type = Tableau de N Type _élément

Tableau de Déclaration des objets:

Objet	Type
Nom_tableau	Nom_type

Activité 1 :

Ecrire un algorithme permettant de remplir un tableau par 7 entiers représentant des prix d'un produit vendu le long de toute la semaine afin de calculer et d'afficher la somme de ces prix.

Exemple :

Produit :	**12500**	**5410**	**3000**	**4320**	**10235**	**44500**	**2000**

Solution :

```
Algorithme Somme
Début
        Pour i de 0 à 6 faire
                Ecrire("Remplir la case d'indice",i," :")
                Lire(T[i])
        Fin pour
        s←0
        Pour i de 0 à 6 faire
                s←s+T[i]
        Fin pour
        Ecrire("La somme des prix est :",s)
Fin
```

Tableau de Déclaration des objets:

Objets	Type
T	Tableau de 7 entiers
i,s	entier

III. *Les tableaux à deux dimensions :*

Un tableau à deux dimensions a la forme d'une matrice composée de lignes et de colonnes. Toutes les cases de ce tableau auront le même type. L'accès à une case de ce tableau s'effectue par conséquent grâce à deux indices comme suit

M [i,j] où i représente l'indice de la ligne et j celui de la colonne.

Déclaration 1:

Tableau de Déclaration des objets:

Objet	Type
Nom_tableau	Tableau de N lignes * M colonnes Type _élément

Déclaration 2:

Tableau de création des nouveaux types:

Nouveau type
Nom_type = Tableau de N lignes * M colonnes Type _élément

Tableau de Déclaration des objets:

Objet	Type
Nom_tableau	**Nom_type**

Activité 2 :

Ecrire un algorithme permettant de remplir une matrice de 5 lignes et 7 colonnes par des entiers représentant des prix de 5 produits vendu le long de toute la semaine afin de calculer et afficher la somme de ces prix.

Solution :

```
Algorithme Somme
Début
        Pour i de 0 à 4 faire
                Pour i de 0 à 6 faire
                        Ecrire("T[",i,"][",j,"]=")
                        Lire(T[i][j])
                Fin Pour(j)
        Fin pour(i)
        s←0
        Pour i de 0 à 4 faire
                Pour i de 0 à 6 faire
                        s←s+ T[i][j]
                Fin Pour(j)
        Fin pour(i)
        Ecrire("La somme des prix est :",s)
Fin
```

Tableau de création des nouveaux types:

Type	
TAB	Tableau de 5 lignes et 7 colonnes d'entiers

Tableau de Déclaration des objets:

Objets	Type
T	Tableau de 7 entiers
i,j,s	entier

- PARTIE N°5-

LES SOUS-PROGRAMMES

I. Introduction :

1. Définition :

Un sous-algorithme est un moyen simple et efficace pour structurer les algorithmes. Elle consiste à associer un identifiant à un groupe d'instructions qui peut être activé par l'appel de cet identifiant.

2. Intérêts :

- Présenter des solutions claires en évitant la **redondance** des codes dans un programme.
- Simplifier la résolution du problème initial en supposant que les différents modules prévus sont déjà **implémentés**
- Se concentrer sur la résolution d'un sous problème à la fois.
- Détecter facilement les parties à consulter ou à modifier.

On distingue deux sortes de modules : ***Les fonctions et Les procédures***

II. Les fonctions :

Une fonction est un sous-algorithme qui renvoie une valeur d'un seul type. Ce type sera celui du **résultat**

Syntaxe :

```
Fonction Nom_fonction (pf1: type1, pf2: type2, … , pfn : typen) : Type_résultat
DEBUT
   Traitement
   Retourner Résultat
FIN
```

III. Les procédures :

Une procédure est un sous-algorithme qui renvoie **zéro, un seul résultat non simple ou plusieurs résultats retournés dans les paramètres en ajoutant le symbôle @ avant chaque objet résultat.**

Syntaxe :

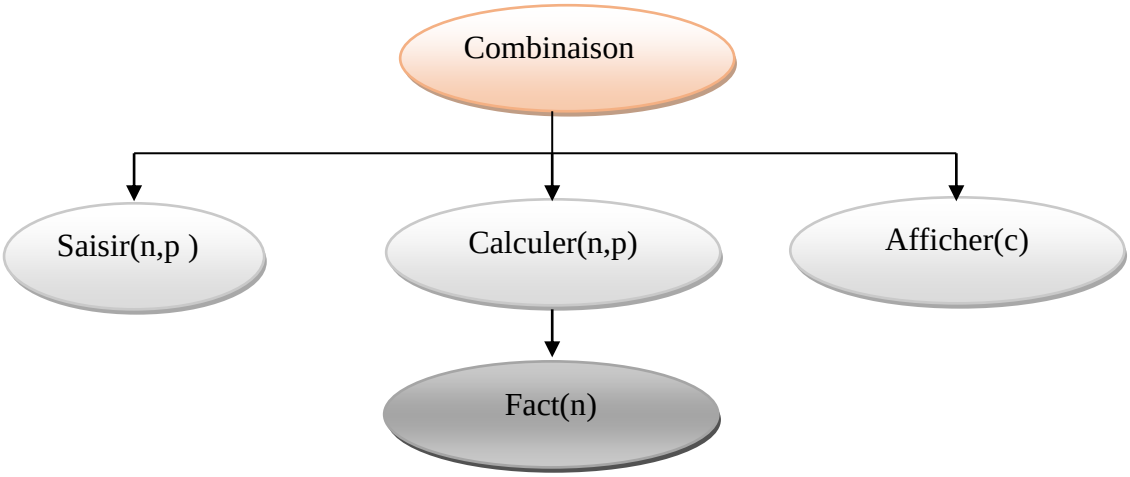
```
Procédure Nom_procédure (pf1: type1, pf2: type2, … , pfn: typen)
DEBUT
   Traitement
FIN
```

Activité:

En utilisant l'approche de l'analyse modulaire, On se propose de chercher puis d'afficher le C(n,p) de deux entiers positifs saisis. Sachant que :

$$C_n^p = \frac{n!}{p!(n-p)!}$$

Décomposition modulaire :



Algorithme du programme principal :

```
Algorithme Combinaison
Début
        Saisir(n )
        Saisir(p )
        Calculer(n,p)
        Afficher(c)
Fin
```

Tableau de Déclaration des objets:

Objets	Type
n,p	entier
c	réel
Saisir	procédure
Calculer	Fonction
Afficher	procédure

Algorithme de la procédure saisir:

```
Procédure saisir(@n,:entier)
Début
        Répéter
                Ecrire("Entrez un entier positif:")
                Lire(n)
        Jusqu'à n>0
Fin
```

Algorithme de la fonction calculer:

```
Fonction calculer(n,p:entier):réel
Début
        c←fact(n)/(fact(p)*fact(n-p))
        Retourner c
Fin
```

Tableau de Déclaration des objets:

Objets	Type
c	réel
fact	fonction

Algorithme de la fonction fact:

```
Fonction fact(n:entier):entier
Début
        f←1
        Pour i de 1 à n faire
                f←f*i
        Fin Pour
        Retourner f
Fin
```

Tableau de Déclaration des objets:

Objets	Type
f	entier

Algorithme de la procédure Afficher:

```
Procédure Afficher(c:réel)
Début
        Ecrire("C(",n,",",p,")=",c)
Fin
```

Interprétations:

- Les paramètres effectifs et les paramètres formels doivent se correspondre de point de vue **nombre**, **ordre** et **type**
- Une fonction possède un type qui est celui de son **résultat** Par exemple la fonction fact est de type entier.
- Tous les paramètres formels d'une fonction ont un mode de passage par **valeur**.
- Les paramètres formels d'une procédure ont un mode de passage par **valeur** ou par **variable**.

- PARTIE N°6-

LES ALGORITHMES DE TRI

I. Introduction :

Le tri consiste à repartir en plusieurs classes selon certains critères un ensemble d'éléments.
Il existe plusieurs algorithmes de tri parmi les quel on cite :

- **Le tri à bulle**
- **Le tri par selection**

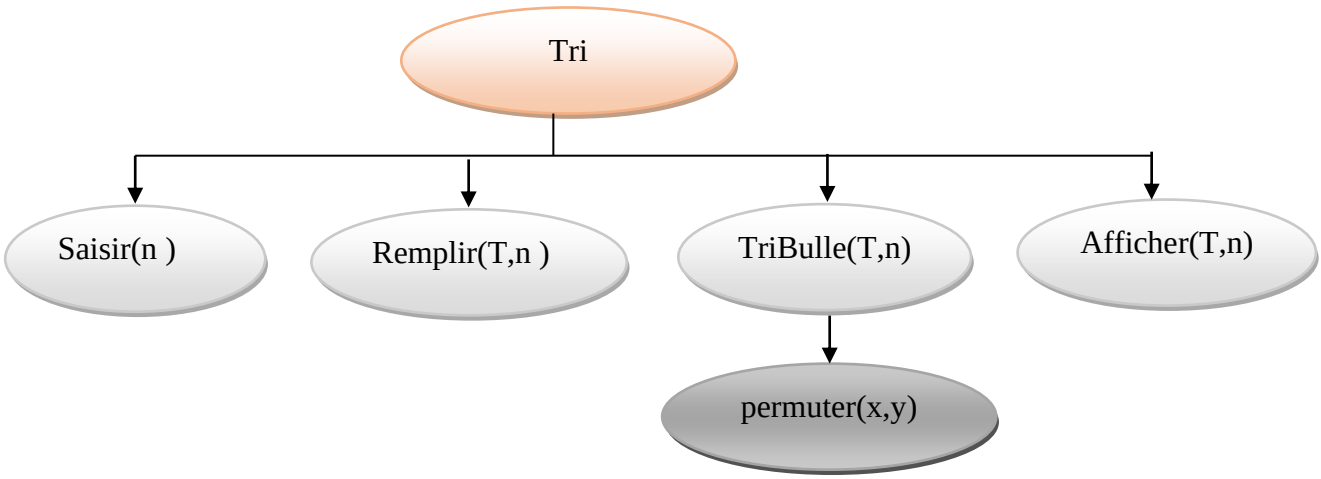
II. Le tri à bulle :

Principe :

Le principe du tri par sélection consiste à :

- Parcourir le tableau en comparant chaque 2 éléments consécutifs et les échanger (ou permuter) s'ils sont désordonnés
- Refaire l'action précédente jusqu'à le tableau soit trié.

Décomposition modulaire :



Algorithme du programme principal

```
Algorithme Tri
Début
        Saisir(n )
        Remplir(T,n)
        TriBulle(T,n)
        Afficher(T,n)
Fin
```

Tableau de création des types :

Types	
TAB	Tableau de 100 entiers

Tableau de déclaration des objets globaux :

Objets	Type
T	TAB
n	entier
Saisir	procédure
Remplir	procédure
TriBulle	procédure
Afficher	procédure

Algorithme du module saisir :

```
Procédure saisir(@n:entier)
Début
        Répéter
                Ecrire("Entrez le nombre de cases du tableau:")
                Lire(n)
        Jusqu'à 5<=n<=100
Fin
```

Algorithme du module remplir :

```
Procédure Remplir(@T :TAB ;n:entier)
Début
        Pour i de 0 à n-1 faire
                Ecrire("Remplir la case d'indice",i," :")
                Lire(T[i])
        Fin pour
Fin
```

Tableau de déclaration des objets locaux :

Objets	Type
i	entier

Algorithme de la procédure TriBulle (par ordre croissant):

```
Procédure TriBulle(@T :TAB ;n:entier)
Début
        Répéter
                b←vrai {on suppose que le tableau est trié}
                Pour i de 0 à n-2 faire
                        Si T[i]<T[i+1] Alors
                                Permuter(T[i],T[i+1])
                                b←faux
                        Fin si
                Fin pour
        Jusqu'à b=vrai
Fin
```

Tableau de déclaration des objets locaux :

Objets	Type
i	entier
b	booléen
permuter	procédure

Algorithme de la procédure permuter :

```
Procédure permuter(@x,y:entier)
Début
        aux←x
        x←y
        y←aux
Fin
```

Tableau de déclaration des objets locaux :

Objets	Type
aux	entier

Algorithme de la procédure Afficher :

```
Procédure Afficher(T :TAB ;n:entier)
Début
        Pour i de 0 à n-1 faire
                Ecrire(T[i],"|")
        Fin pour
Fin
```

Tableau de déclaration des objets locaux :

Objets	Type
i	entier

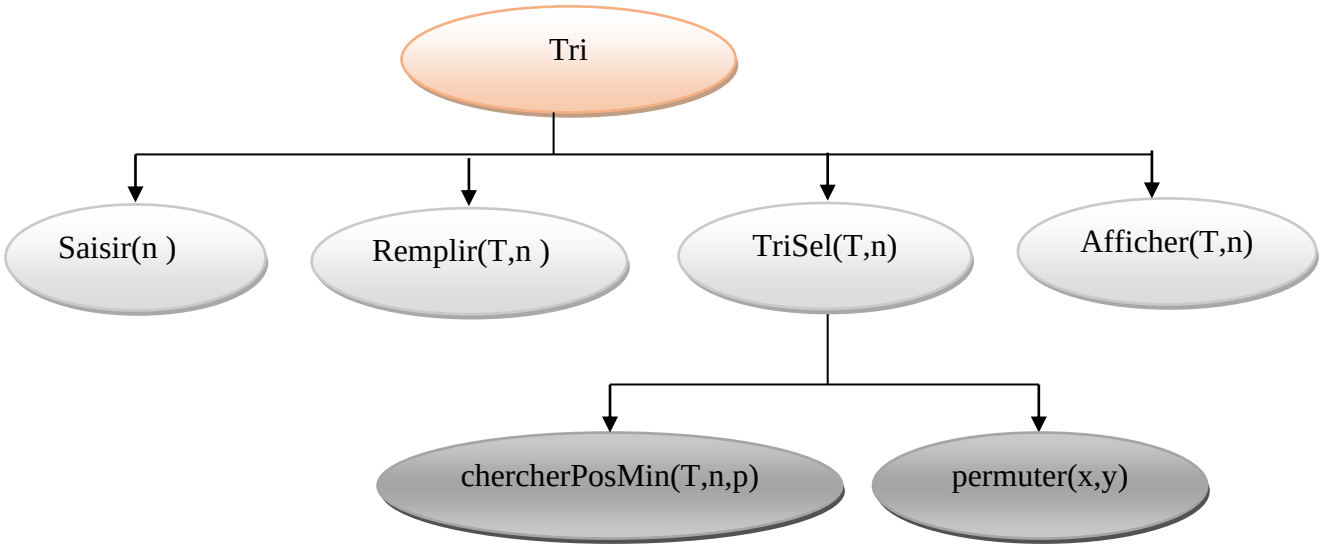
III. Le tri par sélection :

Principe :

Le principe du tri par sélection consiste à :

- Rechercher le plus petit élément de l'élément et le permuter avec le premier élément
- Rechercher le second plus petit élément et le permuter avec le deuxième élément
- Répéter ce traitement jusqu'à ce que le tableau soit trié.

Décomposition modulaire :



Algorithme de la procédure TriSelection

```
Procédure TriSel(@T :TAB ;n:entier)
Début
        Pour i de 0 à n-1 faire
                Posmin ← chercherPosMin(T,n,i)
                Si posmin ≠ i Alors
                        Permuter(T[i],T[posmin])
                Fin si
        Fin pour
Fin
```

Tableau de déclaration des objets locaux :

Objets	Type
i	entier
posmin	entier
chercherPosMin	Fonction
Permuter	Procédure

Algorithme de la fonction chercherPosMin

```
Fonction chercherPosMin(T :TAB ;n,p:entier):entier
Début
        posmin ←p
        Pour i de p+1 à n faire
                Si t[i]<T[posmin] Alors
                        posmin ←i
                Fin si
        Fin Pour
        Retourner posmin
Fin
```

Tableau de déclaration des objets locaux :

Objets	Type
i,posmin	entier

Algorithme de la procédure permuter

Procédure permuter(@x,y:entier)

Début

aux ← x

x ← y

y ← aux

Fin

Tableau de déclaration des objets locaux :

Objets	Type
aux	entier

- PARTIE N°7-

LES ALGORITHMES DE RECHERCHES

I. Introduction :

Il existe plusieurs méthodes de recherche parmi les quels on cite :

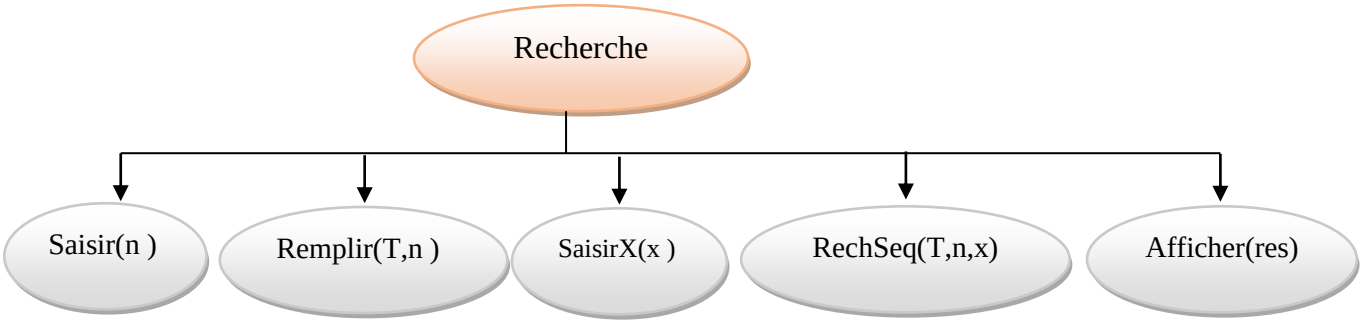
- **Recherche séquentielle**
- **Recherche dichotomique**

II. La recherche séquentielle:

Principe :

La recherche séquentielle qui consiste à parcourir une liste de valeurs jusqu'à trouver la valeur cherchée ou atteindre la fin de la liste.

Décomposition modulaire :



Algorithme de la fonction RechSeq

```
Fonction RechSeq (T :TAB ;n,x:entier):booléen
Début
        res ←Faux , i ←0
        Répéter
                i ←i+1
                Si t[i]=x Alors
                        res ←Vrai
                Fin si
        Jusqu'à (res) ou (i=n-1)
        Retourner res
Fin
```

Tableau de déclaration des objets locaux :

Objets	Type
i	entier
res	booléen

III. La recherche dichotomique:

Principe :

- Déterminer l'indice de l'élément du milieu.
- Si l'élément du milieu égal à celui recherché l'élément est trouvé

  Si l'élément du milieu est supérieur à celui recherché on recherche dans le sous tableau gauche.

  Si l'élément du milieu est inférieur à celui recherché on recherche dans le sous tableau droit.
- Répéter le traitement précédent jusqu'à trouver l'élément ou atteindre la fin du recherche.

Algorithme du module RechDich

```
Fonction RechDich (T :TAB ;n,x:entier):booléen
Début
        res ←Faux , d ←0,f ←n-1
        Répéter
                mil ←(d+f)div 2
                Si t[mil]=x Alors
                        res ←Vrai
                sinon si T[Mil]>x Alors
                        f ←mil-1
                sinon
                        d ←mil+1
        Jusqu'à (res) ou (d>f)
        Retourner res
Fin
```

Tableau de déclaration des objets locaux :

Objets	Type
mil,d,f	entier
res	booléen

- PARTIE N°8-

CALCUL ARITHMÉTIQUE

I. Introduction :

Cette partie est conçue pour les calculs, on va donc automatiser des règles mathématiques afin qu'on puisse les utiliser plus simplement sur machine.

II. Le PGCD :

Le plus grand commun diviseur de deux entiers a et b appelé aussi PGCD est le plus grand entier permettant de diviser a et b. On le note PGCD (a,b).

Activité 1 :

1) Compléter les valeurs en pointillé :

PGCD (15,27)=PGCD(27,15)= PGCD(**15**,**12**)=PGCD(**12**,**3**)=PGCD(**3**,**0**) =**3**

2) Déduire le principe pour déterminer le PGCD.

3) Ecrire l'algorithme correspondant

Principe :

En appliquant la méthode d'Euclide :

- le PGCD (a,b) égale au PGCD (b,a mod b)
- Répéter la modification de a et b jusqu'à b égale à 0 et on dit que a est le pgcd.

Algorithme de la fonction PGCD : { on suppose que a et b strictement positif }

```
Fonction PGCD (a,b:entier):entier
Début
        Répéter
                aux ←a
                a ←b
                b ←aux mod b
        Jusqu'à b=0
        Retourner a
Fin
```

Tableau de déclaration des objets locaux :

Objets	Type
aux	entier

III. Le PPCM :

Le plus petit commun multiple de deux entiers a et b appelé aussi PPCM est le plus petit entier multiple a la fois de a et de b. On le note PPCM (a,b).

Principe :

- Déterminer Max(m,n)
- Chercher les multiples successifs de Max(m,n)
- Répéter l'étape 2 jusqu'à avoir un multiple de min(m,n)

Algorithme du module PPCM

```
Fonction PPCM (a,b:entier):entier
Début
        Si m > n alors
                max ←m , min ← n
        sinon
                max  ←n , min ← m
        Fin si
        aux ←max
         Tant que (max Mod  min ≠ 0) Faire
                max ←max + aux
        Fin Tant que
        Retourner max
Fin
```

Tableau de déclaration des objets locaux :

Objets	Type
min,max,aux	entier

IV. Les nombres premiers:

Un entier supérieur a 1 est dit premier s'il n'est divisible que par 1 et par lui même.

Principe1 :

En appliquant la règle mathématique, on peut calculer le nombre de diviseur de n, s'il est égale à 2 on dit que n est premier.

Principe2 :

Parcourir les diviseurs possible et on arrête dés qu'on trouve un diviseur ou atteindre la fin de la liste.

Algorithme du module Premier

Méthode1

```
Fonction premier (n:entier):booleén
Début
        nb ←0
        Pour i de 1 à n faire
                Si n mod i=0 Alors
                        nb ←nb+1
                Fin si
        Fin Pour
        Si nb=2 Alors
                res ←vrai
        Sinon
                res ←faux
        Fin si
        Retourner res
Fin
```

Méthode2

```
Fonction premier (n:entier):booleén
Début
        res ←vrai , i ←1
        répéter
                i ←i+1
                res ← n mod i=0
        jusqu'à (res=faux) ou (i=n-1)
        Retourner res
Fin
```

Tableau de déclaration des objets locaux :

Objets	Type
i,nb	entier
res	booléen

V. La factorielle :

La factorielle d'un entier n donné est le produit de tous les entiers de l'intervalle [1,n]. La factorielle de n est noté n !

Principe :

En appliquant la règle mathématique n !=n*(n-1)*(n-2)*…..*2*1

Algorithme du module Factorielle

```
Fonction Factorielle (n:entier):entier
Début
        f←1
        Pour i de 1 à n faire
                f←f*i
        Fin Pour
        Retourner f
Fin
```

Tableau de déclaration des objets locaux :

Objets	Type
i,f	entier

VI. La décomposition en facteurs premiers :

La décomposition d'un entier en produit de facteurs premiers consiste à écrire cet entier sous la forme d'un produit de ces diviseurs premiers.

Principe :

La méthode consiste à chercher et ranger les facteurs premiers de n dans un tableau

Algorithme du module FactPemier

```
Procédure FactPremier (Var Fact :Tab ; Var f : entier ; n : entier)
Début
        i ←2 , f ←0
        Répeter
                Si n mod i = 0 alors
                        n ←n div i
                        f←f + 1
                        Fact[f] ←i
                sinon
                        i ←i + 1
                Fin si
        Jusqu'à (n = 1)
fin
```

Tableau de déclaration des objets locaux :

Objets	Type
i	entier

- PARTIE N°9-

LES ENREGISTREMENTS

I. Introduction :

Un enregistrement est un type correspondant à un ensemble d'éléments de types éventuellement différent aux quels on accède grâce à un nom.

Un enregistrement, comme un tableau, regroupe plusieurs informations. Alors que dans un tableau tous les éléments doivent être du même type, dans un enregistrement les différents composantes (ou champs) peuvent être de type différent.

II. Déclaration :

Tableau de déclaration des nouveaux types

Type
Nom_type = Enregistrement Champ 1 : Type 1 …… : …… Champ n : Type n Fin Nom_type

Tableau de déclaration des objets

Objet	Type / Nature
Identificateur_objet	Nom_type

Activité 1:

Déclarer le type note comme étant un enregistrement composé des champs : valeur (nombre), un coefficient (nombre) et une matière (chaîne de caractères).

Solution :

Tableau de déclaration des nouveaux types

Type
Note = Enregistrement valeur : entier Coefficient : entier matière: chaine Fin Note

III. Manipulation des enregistrements :

Pour un tableau on accède à un élément en donnant le nom du tableau et la valeur de l'indice, pour un enregistrement on repère une composante par le nom de l'enregistrement et le nom du champ auquel on veut accéder séparés par un point : **Variable.Champ**

- Pour affecter une valeur à un champ on utilise la syntaxe suivante :

Syntaxe :
Variable.Champ ⟵ *Valeur*

- Pour lire un champ on utilise la syntaxe suivante :

Syntaxe :
Lire(Variable.Champ)

- Pour Afficher un champ on utilise la syntaxe suivante :

Syntaxe :
Ecrie(Variable.Champ)

Activité 2:

a. Déclarez une variable enregistrement pour représenter la fiche d'un étudiant sachant qu'elle contient les informations suivantes : Nom, Prénom, sexe (F ou G), date de naissance et la moyenne au baccalauréat.

b. Affectez respectivement les valeurs suivantes à cette variable : "Kéfi", "Nour", "F", "27/11/1983" et 13.25

Solution :

a. Tableau de déclaration des nouveaux types

Type
Fiche = Enregistrement nom, prénom : Chaîne sexe : Caractère date_nais : Chaîne moy : Réel Fin Note

Tableau de déclaration des objets

Objets	Type
etudiant	Fiche

b. etudiant.nom ← "Kéfi" etudiant.prénom ← "Nour"

etudiant.sexe ← "F" etudiant.date_nais ← "27/11/1983"

etudiant.moy ← 13.25

Activité 3:

Nous supposons que le nombre d'étudiants dans une classe est égal à N (4 <=N<=30).

Question :

Proposez une structure de données utilisant un vecteur d'enregistrements pour représenter ces N étudiants sachant que Date est un enregistrement contenant comme valeur jour, mois et année?

Solution :

Tableau de déclaration des nouveaux types

Type
Date = Enregistrement jour : Entier mois : Chaîne année : Entier **Fin Date** **Fiches** = Enregistrement nom, prénom : Chaîne sexe : Caractère Date_nais : Date moy : Réel **Fin Fiches** **Tab** = Tableau de 30 Fiches

Tableau de déclaration des objets

Objets	*Type*
T	*Tab*

Activité 4:

Une société veut informatiser la gestion de ses employés. Elle détient pour chacun les informations suivantes :

- Le nom et le prénom (chaîne de caractères)
- Le grade : uniquement G1, G2, G3 ou G4
- Le code fiscal (un entier)
- L'assurance maladie (O pour oui et N pour non)

Le nombre d'employés est N avec $4 \leq N \leq 120$.

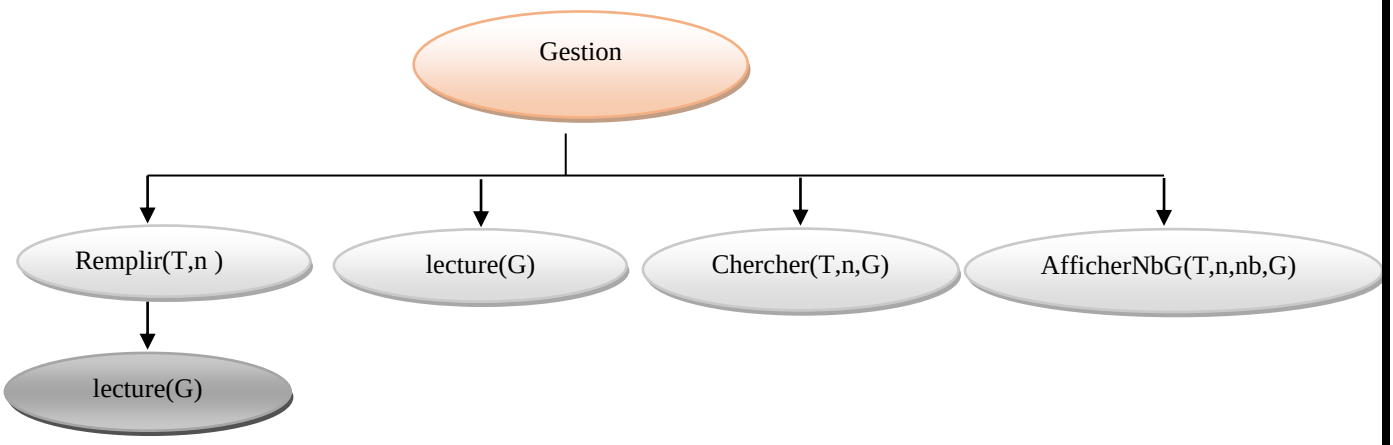
Questions :

En utilisant l'approche de l'analyse modulaire écrire un algorithme qui permet la saisie de toutes les fiches de renseignements puis d'afficher :

1- Toutes les fiches.

2- Le nombre d'employés ayant un grade donné et leur pourcentage par rapport au nombre total des employés.

Décomposition modulaire:



Programme principal :

Algorithme Gestion

Début

Remplir(T,n)

Lecture(G)

Chercher(T,n,G)

Afficher(T,n,nb,G)

Fin

Tableau de création des nouveaux types :

Types
Fiche=Enregistrement nom : Chaîne de 40 caractères grade : Chaîne de 2 caractères code : Entier non signé am : Caractère Fin Fiche
Tab = Tableau de 100 fiches
Gr = Chaîne de 2 caractères

Tableau de déclaration des objets globaux :

Objets	Type
T	TAB
N, nb	entier
G	Gr
Remplir	Procédure
Lecture	procédure
chercher	Fonction
Afficher	Procédure

Algorithme de la procédure Remplir :

```
Procédure Remplir(@T :Tab ;@n :entier)
Début
        Répéter
                Ecrire ("Entrer le nombre d'employés : ")
                Lire (n)
        Jusqu'à  (n>4) et (n<100)
         Pour i de 1 à N Faire
                Ecrire ("Entrer le nom et le prénom de l'employé : ") ;
                Lire (T[i].nom)
                Lecture (T[i].grade)
                Ecrire ("Entrer son code fiscal : ") ;
                Lire (code)
                Répéter
                        Ecrire ("Assuré (Oui / Non ) ? : ")
                        Lire (T[i].am)
                Jusqu'à (Majus(T[i].am) ="O") ou (Majus(T[i].am) ="N")
        Fin Pour
Fin
```

Tableau de déclaration des objets locaux :

Objets	Type
i	entier

Algorithme de la procédure Lecture :

```
Procédure Lecture(@G : Gr)
Début
        Répéter
                Ecrire ("Entrer le grade : ")
                Lire (G)
        Jusqu'à (G="G1") ou (G="G2") ou (G="G3") ou (G="G4")
 Fin
```

Algorithme de la fonction chercher :

```
Fonction  Chercher( T :Tab ; n :entier ;G :Gr) :entier
Début
        nb ← 0
        Pour i de 1 à N Faire
                Si (T[i].grade = G) Alors
                        nb ← nb + 1
                Fin Si
        Fin Pour
        Retourner  nb
Fin
```

Tableau de déclaration des objets locaux :

Objets	Type
i,nb	entier

Algorithme de la procédure Afficher :

```
Procédure Afficher(T :Tab ;n,nb :entier ;G :Gr)
Début
        Pour i de 1 à n Faire
                Ecrire ("Nom & prénom : ", nom)
                Ecrire ("Grade : ", grade)
                Ecrire ("Code fiscal : ", code)
                Ecrire ("Assurance maladie :", am)
        Fin Pour
        Ecrire ("Le nombre d'occurrences de ', G, " Est égal à : ", nb)
        P ← (Nb_G * 100 ) / N
        Ecrire ("Le pourcentage de ", G, " Est de : ", P, " %")
Fin
```

- PARTIE N°10-

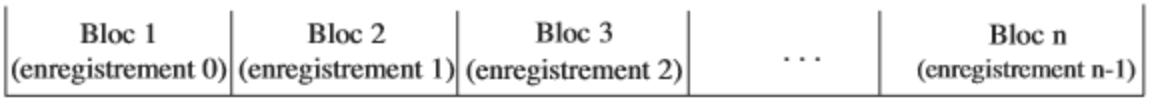
LES FICHIERS

I. Introduction :

Un fichier est un ensemble structuré de données de même type, nommé et enregistré sur un support lisible par l'ordinateur (disque dur, flash disque, CD Rom, ..etc). Un fichier peut contenir des caractères (fichier textes), des programmes, des valeurs (fichier de données).

II. Organisation des fichiers :

Les blocs de données formant un fichier sont enregistrés les uns à la suite des autres, de façon linéaire, comme indiqué dans le schéma suivant :



Pour accéder (lire ou modifier) à un bloc (un enregistrement), nous devons tout d'abord le localiser, c'est-à-dire le **pointer**.

Le pointeur de fichier est un **entier** qui indique à partir de quelle position (ou adresse) du fichier la prochaine fonction **d'entrée/sortie** doit s'effectuer.

III. Types d'accès :

On distingue deux types d'accès aux données d'un fichier :

- **Accès séquentiel:** Pour lire une information particulière, il faut **lire toutes les informations situées avant.**
- **Accès direct :** Nous pouvons accéder directement à l'information désirée, en précisant **le numéro d'emplacement (le numéro d'ordre) de cette information.**
- Tout fichier peut être utilisé avec l'un ou l'autre des types d'accès. Le choix de ce type n'est pas un choix qui concerne le fichier lui-même, mais uniquement la manière dont il va être traité par la machine. C'est donc dans le programme, et seulement dans le programme, que l'on choisit le type d'accès souhaité.

IV. Les fichiers texte :

1. Lecture dans un fichier :

Activité 1 :

Le fichier notes.txt contient les notes obtenues par des étudiants pour le cours de l'algorithme. Chaque ligne du fichier ne contient qu'une note.

Ecrire un algorithme d'un programme qui lit chaque ligne de ce fichier, extrait les notes sous forme de ***réel*** et les stocke dans un tableau.

Solution guidée :

Activité 1.1 :

Déclarer les structures adéquates au problème.

Solution :

Tableau de déclaration des objets :

Objets	Type
Fiche	Fichier texte
T	Tableau de réels
i,cp	entier
ch	chaine

Activité 1.2 :

Ouvrir le fichier notes en mode lecture.

Solution :

Ouvrir ("note.txt", Fiche, "r")

Interprétation :

- **ouvrir ("nom_Physique",nom_Logique, "r"): ouvre le fichier en mode lecture seule**

Activité 1.3 :

Parcourir le fichier jusqu'à la dernière ligne en ajoutant chaque ligne comme une valeur dans un tableau.

Solution :

```
Algorithme Récupérer
Début
        Ouvrir ("note.txt", Fiche, "r")
        i←0
        Tant que NON (Fin_fichier (Fiche)) faire
                Lire_ligne ( fiche, ch)
                T[i] ←valeur(ch)
                i←i+1
         Fin Tant que
        Pour cp de 0 à i faire
                Ecrire (T[cp], '|')
        Fin Pour
        Fermer (fiche)
Fin
```

Interprétations :

- **Lire_ligne (nom_Logique, variable):** lit une ligne d'un fichier et la renvoie sous forme de chaîne de caractères.

➡ À chaque nouvel appel de **Lire_ligne( )**, la ligne suivante est **renvoyée**

Associée à la boucle while, cette méthode permet de lire un fichier **ligne par ligne.**

- **Fin_fichier (nom_Logique) : Retourne Vrai si le pointeur est à la fin du fichier sinon elle retourne Faux.**
- **fermer (nom_Logique): Fermeture d'un fichier**

Activité 1.4 :

Modifiez le code précédent afin de récupérer les lignes sans utilisé une boucle

Solution :

```
Algorithme Récupérer
Début
        Ouvrir ("note.txt", Fiche, "r")
        Lire (fiche,ch)
Fin
```

Interprétation :

- **lire (nom_Logique,variable) : lit tout le contenu d'un fichier et renvoie une chaîne de caractères unique (le retour à la ligne est compté comme un caractère).**

2. *Ecriture dans un fichier :*

Activité 2 :

On veut consulter le fichier notes.txt afin de calculer la moyenne de classe.

Ecrire un algorithme d'un programme qui affiche la moyenne dans un nouveau fichier texte appelé moyenne.txt.

Solution :

```
Algorithme Récupérer
Début
        Ouvrir ("note.txt", Fiche, "r")
        i ←0
        Tant que NON (Fin_fichier (Fiche)) faire
                Lire_ligne ( fiche, ch)
                T[i] ←valeur(ch)
                i ←i+1
         Fin Tant que
        s ←0
        Pour cp de 0 à i faire
                s ←s+ T[i]
        Fin Pour
        moy ←s/i
        Ouvrir ("moyenne.txt", FicheMoy, "w")
        Ecrire (FicheMoy,str(moy))
Fin
```

Interprétations :

- **ouvrir ("nom_Physique",nom-logique,"w"): ouvre le fichier en mode lecture seule**
- **Ecrire (nom_Logique,variable): Ecriture dans un fichier texte**
- **Ouvrir ("nom_Physique", nom-logique,"a") : ouvre le fichier en mode écriture et ajoute à la fin du fichier.**

V. Les fichiers de données :

Activité 3 :

Au lieu de sauvegarder seulement les notes des élèves, nous proposons d'enregistrer dans un fichier appelé CARNET, la fiche de renseignements et les notes de chaque élève. Une fiche de renseignements comporte les informations suivantes :

- Nom et prénom,
- Sexe (F ou G),
- Date de naissance,
- Les notes sont : note1, note2 et note3.

Consulter le fichier crée afin de déterminer et afficher le nombre de garçons existant.

Solution guidée :

Activité 3.1 :

Déclarer les structures adéquates au problème.

Solution :

Tableau de création des nouveaux types :

Types
Fiche=Enregistrement nom_prenom : Chaîne sexe : caractères date_naissance : Chaîne note1, note2, note3 : Réel Fin Fiche
Carnet = Fichier de fiches

Tableau de déclaration des objets :

Objets	Type
info	Fiche
eleve	carnet

Activité 3.2 :

1. Créer un fichier intitulé ***«donnee_Eleve.dat»*** dans la racine C :
2. Saisir l'enregistrement d'un élève afin de l'afficher dans le fichier.

Solution :

```
Algorithme datElv
Début
        Ouvrir ("c:/ donnee_Eleve.dat", eleve, "wb")
        Ecrire ("Donner le nom de l'élève : ")        Lire (info.nom_prénom)
        Ecrire ("Donner son sexe ")                   Lire (info.sexe)
        Ecrire ("Donner sa date de naissance : ")     Lire (info.date_naissance)
        Ecrire ("Donner sa première note : ")         Lire (info.note1)
        Ecrire ("Donner sa deuxième note : ")         Lire (info.note2)
        Ecrire ("Donner sa troisième note : ")        Lire (info.note3)
        Ecrire (eleve, info)
```

Activité 3.3 :

Consulter le fichier ***«donnée_Eleve.dat»*** et afficher les renseignements associés à chaque élève.

Solution :

```
        Lire (eleve, info)
        Ecrire ("Nom : ", info.nom_prénom)
        Ecrire ("Sexe ", info.sexe)
        Ecrire ("Date de naissance : ", info.date_naissance)
        Ecrire ("Première note : ", info.note1)
        Ecrire ("Deuxième note : ", info.note2)
        Ecrire ("Troisième note : ", info.note3)
        Fermer (eleve)
Fin
```

Activité 3.4 :

Consulter le fichier ***«donnée_Eleve.dat»*** afin de déterminer et afficher le nombre de garçons existant.

Solution :

```
Algorithme Calcul
Début
        Ouvrir ("c:/ donnee_Eleve.dat", eleve, "rb")
        nbg← 0
        Tant que Non (Fin_fichier (eleve) ) Faire
                Lire (eleve, info)
                Si  info.sexe = "G" Alors
                        nbg := nbg +1
                Fin Si
        Fin Tant Que
        Ecrire ("Nombre de garçons : ", nbg)
Fin
```

Interprétations :

- **lire (nom_Logique, objet): Lecture d'un enregistrement d'un fichier**

Remarque :

- Les fichiers de données (data) ont une extension **.dat** (pour les enregistrements)

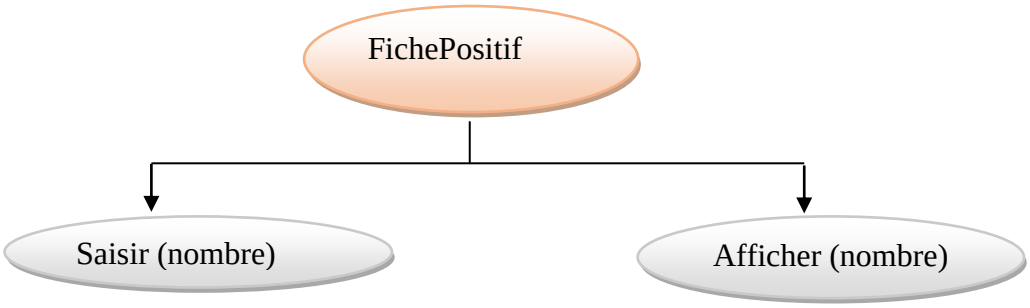
Activité 4 :

Nous nous proposons d'écrire un programme intitulé FichePositif, qui :

- Enregistrer sur la racine C : un fichier ayant pour nom Nombres.FCH comportant des entiers positifs saisis au clavier. Le nombre de ces entiers est inconnu à l'avance. La condition d'arrêt est la saisie d'un nombre négatif. Ce dernier ne sera pas enregistré dans le fichier.
- Affiche tous les éléments du fichier.

En appliquant l'approche de l'analyse modulaire, résoudre le problème déjà décrit.

Décomposition modulaire :



Algorithme du programme principal :

```
Algorithme FichePositif
Début
        Ouvrir ("c:/ Nombres.FCH", nombre, "wb")
        Saisir (nombre)
        Afficher (nombre)
        Fermer (nombre)
Fin
```

Tableau de création des nouveaux types :

Types
F_Entiers = Fichier d'entiers

Tableau de déclaration des objets globaux:

Objets	Type
nombre	F_Entiers
Saisir	procédure
Afficher	procédure

Algorithme de la procédure Saisir:

```
Procédure Saisir(@nombre :F_Entiers)
Début
        Ecrire ("Taper un entier : ")
        Lire ( V)
        Tant que V ≥ 0 Faire
                Ecrire (nombre , V)
                Ecrire ("Entrer un entier : ")
                Lire ( V)
        Fin Tant que
Fin
```

Tableau de déclaration des objets locaux:

Objets	Type
v	entier

Algorithme de la procédure Afficher:

```
Procédure Afficher(@nombre :F_Entiers)
Début
        Tant que Non (Fin_fichier (nombre)) Faire
                Lire (nombre,  V)
                Ecrire (V)
        Fin Tant que
Fin
```

Tableau de déclaration des objets locaux:

Objets	Type
v	entier

Remarque :

- Les fichiers de données (data) ont une extension .FCH (pour les **nombres**)
- Les modes d'ouvertures sont :
  - **"rb" : Lecture**
  - **"wb" : Ecriture**
  - **"ab" : Ecriture à la fin du fichier**
- **Pointer (Nom_logique, p) : Se pointer à une position p du fichier**
- **Taille_fichier (Nom_logique) : Retourne la taille du fichier**