ENSIAS

ALGORITHMIQUE

A. ETTALBI

A.U. : 2018-2019

ettalbi1000@gmail.com

Module M1.1

Intitulé : Algorithmique et Programmation

Responsable : A. ETTALBI

Volume horaire : 54 Heures

Période : Semestre S1 (1er Semestre de la 1ère Année)

Composition de M1.1

2 Eléments de Module :

- M1.1.1 : **Algorithmique** (28H)
- M1.1.2 : **Programmation** (26H)

Caractéristiques des Eléments de Module du Module M1.1

	Nom	**Responsable**	**Volume horaire**	**Coeff**
M1.1.1	Algorithmique	ETTALBI	Cours : 14H TD : 14H	1
M1.1.2	Programmation	NASSAR	Cours : 12H TP : 14H	1

ENSIAS

ALGORITHMIQUE

A. ETTALBI

A.U. : 2018-2019

ettalbi1000@gmail.com

Plan Général

I- Définitions

II- Objectifs de la programmation

III- Les langages de programmation

IV- Algorithme et Organigramme

V- Structure d'un Algorithme

VI- Structures de données

VII- Fonctions (Sous-Programmes)

Exercices-Corrections

I- DEFINITIONS

- **Informatique :**

Définition 1 :

Traitement automatique de ***l'information***

Définition 2 :

Science de ***l'ordinateur***

I- DEFINITIONS

- **Information :**

  Toute donnée ***brute*** qui peut être ***quantifiée***, ***stockée*** pour être ***traitée*** afin de donner un ***résultat***.

I- DEFINITIONS

- **Ordinateur :**

Machine qui permet de ***mémoriser*** une grande quantité d'information et faire des ***calculs*** de base sur ces informations très ***rapidement***.

Composants de l'ordinateur

- **Composants internes :**
  - Unité Arithmétique et Logique
  - Mémoires (RAM, ROM, …)
  - Entrées/Sorties

Composants de l'ordinateur

- **Composants externes ou Périphériques :**

Eléments externes à l'ordinateur

qui communiquent avec lui

Composants de l'ordinateur

- **3 types de Périphériques :**

- d'entrée (clavier, souris, scanner)

- de sortie (écran, imprimante)

- de stockage (disques, CD-ROM)

II- OBJECTIF DE LA PROGRAMMATION

- Résolution automatique (par l'ordinateur) des problèmes.
- En profitant de la rapidité et de la capacité de stockage de l'ordinateur.

II- OBJECTIF DE LA PROGRAMMATION

- **<u>Etapes en général à suivre :</u>**
  - Position du Problème
  - Modèle de résolution
  - Algorithme de résolution
  - Programme informatique

EXEMPLE

Exemple :

Calcul de la consommation d'un véhicule

Etape 1 : *Position du Problème*

- Consommation = Nombre de litres consommés en 100 km
- Données :
  - K1 : Kilométrage au départ
  - K2 : Kilométrage à l'arrivée
  - L1 : Nombre de litres au départ
  - L2 : Nombre de litres à l'arrivée

Etape 2 : *Modèle de Résolution*

K2-K1 ⟶ L1-L2

100 ⟶ X ?

- Données d'entrée : K1, K2, L1, L2
- Résultat à calculer (à chercher) : X
- Donc :

X=100 * (L1-L2) / (K2-K1)

Etape 3 : *Algorithme de Résolution*

Début :

Afficher("Donner K1,K2,L1,L2 : ")

Lire(K1,K2,L1,L2)

Si (K1=K2) Alors Afficher("Erreur ")

Sinon Calculer X←100*(L1-L2)/(K2-K1)

Afficher("la consommation est : ",X, "%")

FinSi

Fin.

Etape 4 : *Programme en langage C*

```
#include<stdio.h>
 main() {
  int K1,K2,L1,L2; float X;
  printf("Donner K1,K2,L1,L2 :");
  scanf("%d%d%d%d",&K1,&K2,&L1,&L2);
  if(K1==K2) printf("Erreur");
  else { X = 100 * (L1-L2) / (K2-K1);
          printf("La consommation est %f %",X);
        }
}
```

III- LES LANGAGES DE PROGRAMMATION

- **Définition :**

– Moyen de communication entre l'homme et la machine

– Ensembles de conventions pour assurer un dialogue bidirectionnel entre le développeur et l'ordinateur.

III- LES LANGAGES DE PROGRAMMATION

- **3 Niveaux :**
  - Langage machine (binaire)
  - Langages d'assemblage
  - Langages évolués

LANGAGE MACHINE

- Lié à la structure **électronique** interne de l'ordinateur,
- Composé de **bits** (0 et 1),
- **Jamais** utilisé par les programmeurs,
- **Seul** langage compris par la machine,
- Tout programme informatique doit être **traduit** vers ce langage.

LANGAGES D'ASSEMBLAGE (Langages de bas niveau)

- Propres à chaque machine (càd à chaque **processeur**),
- **Proches** du langage machine,
- **Loin** du langage naturel,
- Nécessitent bien-sûr un **traducteur**.

LANGAGES EVOLUES
(Langages de haut niveau)

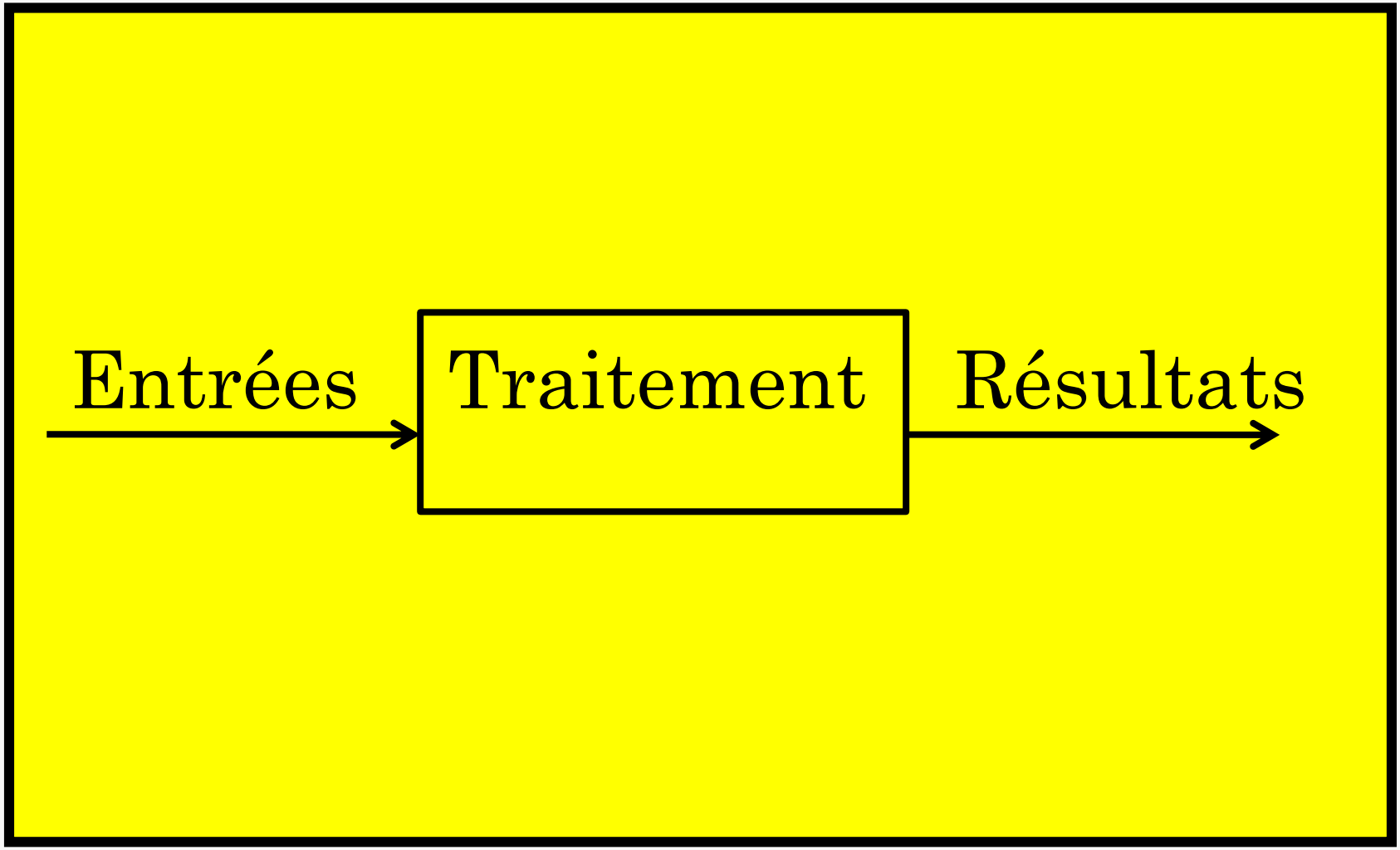
- **Proches** du langage naturel
- **Loin** du langage machine
- Nécessitent une traduction vers le langage machine par un ***compilateur*** ou un ***interpréteur***

Exemples : Pascal, C, JAVA, SQL

PROGRAMME INFORMATIQUE

- Ensemble d'instructions agissant sur des données en entrées pour donner des résultats en sortie.
- Les informations manipulées sont codifiées sous forme de variables, constantes, …

PROGRAMME INFORMATIQUE



PROGRAMME INFORMATIQUE

- **Variable :**

Objet informatique identifié par un "***identificateur***", d'un ***"type"*** donné, possédant une ***case mémoire*** et une ***valeur*** qui peut ***changer*** au cours de l'exécution du programme.

PROGRAMME INFORMATIQUE

- **Constante :**

Objet informatique identifié par un ***"identificateur"***, possédant une ***case mémoire*** et une ***valeur qui ne peut pas changer*** au cours de l'exécution du programme.

PROGRAMME INFORMATIQUE

- **Identificateur :**

Chaîne de caractères ***alphanumériques*** qui ***commence*** par un caractère alphabétique, qui ***ne contient pas*** d'espaces et pas de ***caractères spéciaux*** (é, è, à, ê, î, ...)

PROGRAMME INFORMATIQUE

Exemple d'identificateurs *corrects*:

A, a, age, etudiant, UnEtudiant

Exemple d'identificateurs *incorrects*:

à, âge, étudiant, 1Etudiant

PROGRAMME INFORMATIQUE

- **Type (Domaine) :**

Représente l'ensemble des ***valeurs possibles*** pour une variable et spécifie l'ensemble des ***opérations possibles*** sur cette variable.

PROGRAMME INFORMATIQUE

- **Types prédéfinis :** Entiers, Réels, Chaînes de caractères, Dates, …
- **Types définis par l'utilisateur :** Etudiants, Vecteurs à 3 dimensions, Matrices carrées d'ordre N, …

ETAPES D'UN PROGRAMME INFORMATIQUE

- **Edition** : écriture du programme,
- **Compilation** : détection des erreurs et traduction vers le langage machine,
- **Exécution** : qui ***"doit"*** donner les résultats attendus du programme.

IV-ALGORITHME ET ORGANIGRAMME

Algorithme :

- Description des ***étapes*** de résolution d'un problème.
- Constitué d'un ensemble d'***opérations*** élémentaires (afficher, lire, calculer, …)

IV-ALGORITHME ET ORGANIGRAMME

Organigramme :

- Description des étapes de résolution d'un problème,
- Constitué d'un ensemble de ***figures géométriques*** permettant de schématiser les opérations.

EXEMPLE
Calcul du Périmètre d'un Cercle

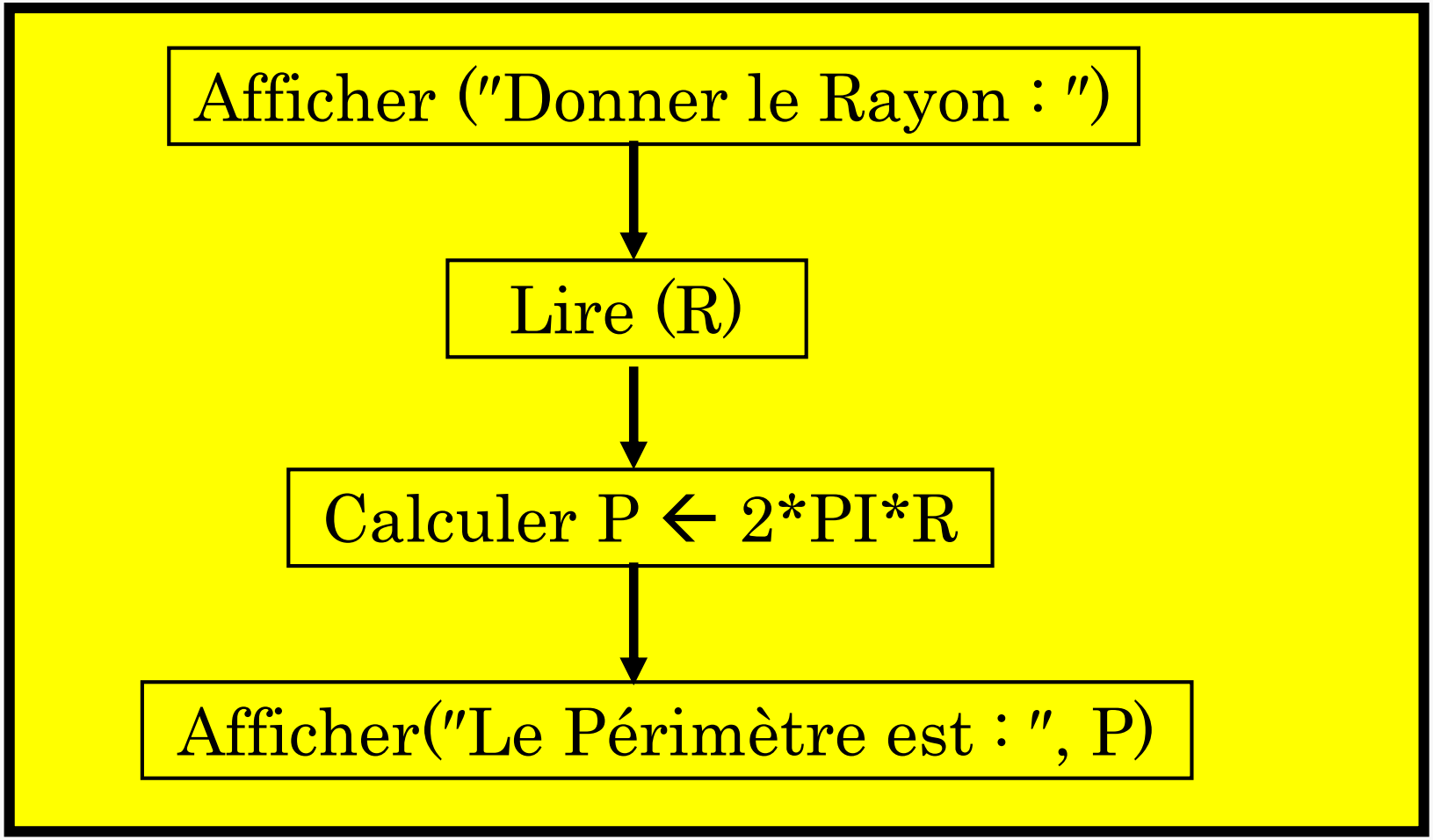
- **Données en entrées :**
  - R (Rayon)
  - PI (constante = 22/7)
- **Résultat à calculer :**
  - P (Périmètre)
- **Modèle de résolution :**

$$P = 2 * PI * R$$

EXEMPLE
Calcul du Périmètre d'un Cercle
Algorithme

```
Objets :
   PI : Constante=22/7
   P, R : variables réelles
Début :
    Afficher("Donner le Rayon : ")
    Lire(R)
    Calculer P ← 2*PI*R
    Afficher("Le Périmètre est : ", P)
Fin.
```

EXEMPLE
Calcul du Périmètre d'un Cercle
Organigramme



EXEMPLE
Calcul du Périmètre d'un Cercle
Programme en C

```
#include<stdio.h>
#define PI 3.14  /* constante */
 main()
{ float P, R;
  printf("Donner le Rayon :");
  scanf("%f",&R);
  P=2*PI*R;
  printf("Le Perimetre est %f", P);
}
```

EXERCICE D'APPLICATION

Calcul de la moyenne de deux nombres

1) Donner les objets en entrée et en sortie.

2) Donner le modèle de résolution.

3) Dresser l'algorithme.

4) Ecrire le programme en langage C.

V- STRUCTURE D'UN ALGORITHME

Types d'instructions :

- Instructions **séquentielles.**
- Instructions **alternatives.**
- Instructions **itératives.**

INSTRUCTIONS SEQUENTIELLES

- S'exécutent ***toutes***.
- S'exécutent l'une après l'autre

dans un ordre ***séquentiel***.

Exemples :

Lire, Afficher, Calculer

EXEMPLE

Calcul du périmètre et de la surface d'un rectangle (Algorithme)

```
Objets :
  Long , Larg, P, S  : variables réelles,
Début :
   Afficher("Donner la longueur et la largeur : ")
   Lire(Long, Larg)
   Calculer P← 2*(Long + Larg)
   Calculer S← Long * Larg
   Afficher("Le Périmètre est : ", P)
   Afficher("La Surface est : ", S)
Fin.
```

EXEMPLE

Calcul du périmètre et de la surface d'un rectangle (Programme)

```
#include<stdio.h>
main()  { float Long, Larg, P, S;
  printf("Donner la longueur et la largeur : ");
  scanf("%f%f", &Long, &Larg);
  P = 2*(Long + Larg);
  S = Long * Larg;
  printf("Le Périmètre est : %f", P);
  printf("\nLa Surface est : %f", S);
}
```

INSTRUCTIONS ALTERNATIVES

2 types :

- A deux alternatives :

  1 Choix (Chemin) parmi 2.

- A plusieurs alternatives :

  1 Choix parmi plusieurs.

INSTRUCTIONS A 2 ALTERNATIVES

- Se basent sur une ***condition***
- Contiennent 2 blocs d'instructions.
- Si la condition est vraie, on exécute le 1er bloc, si elle est fausse, on exécute le 2ème bloc.
- Le 2ème bloc est facultatif

INSTRUCTIONS A 2 ALTERNATIVES

- **Syntaxe générale**

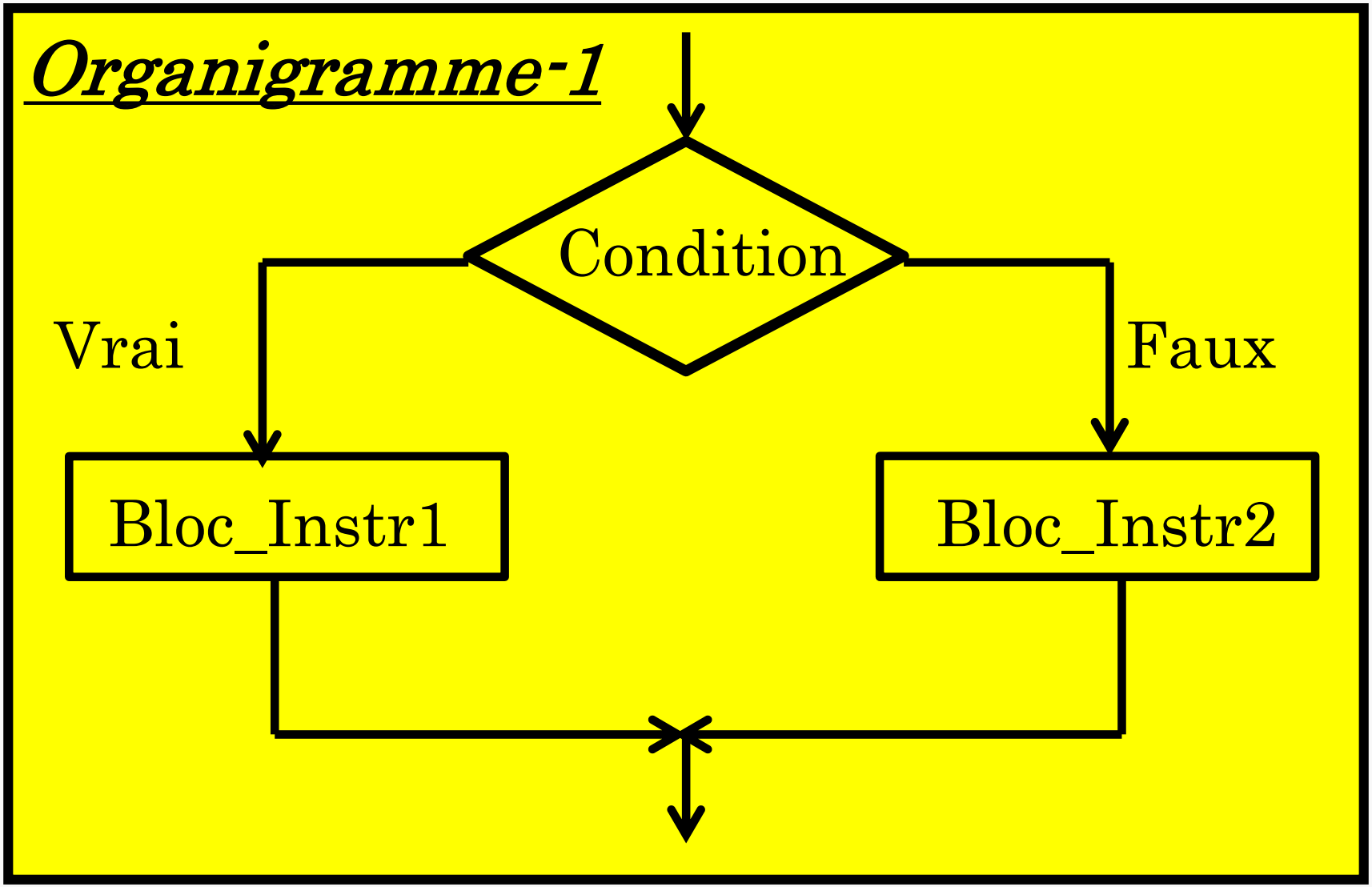
  Si Condition **alors** Bloc_Instructions1
  [Sinon Bloc_Instructions2**]**
  FinSi

Avec :

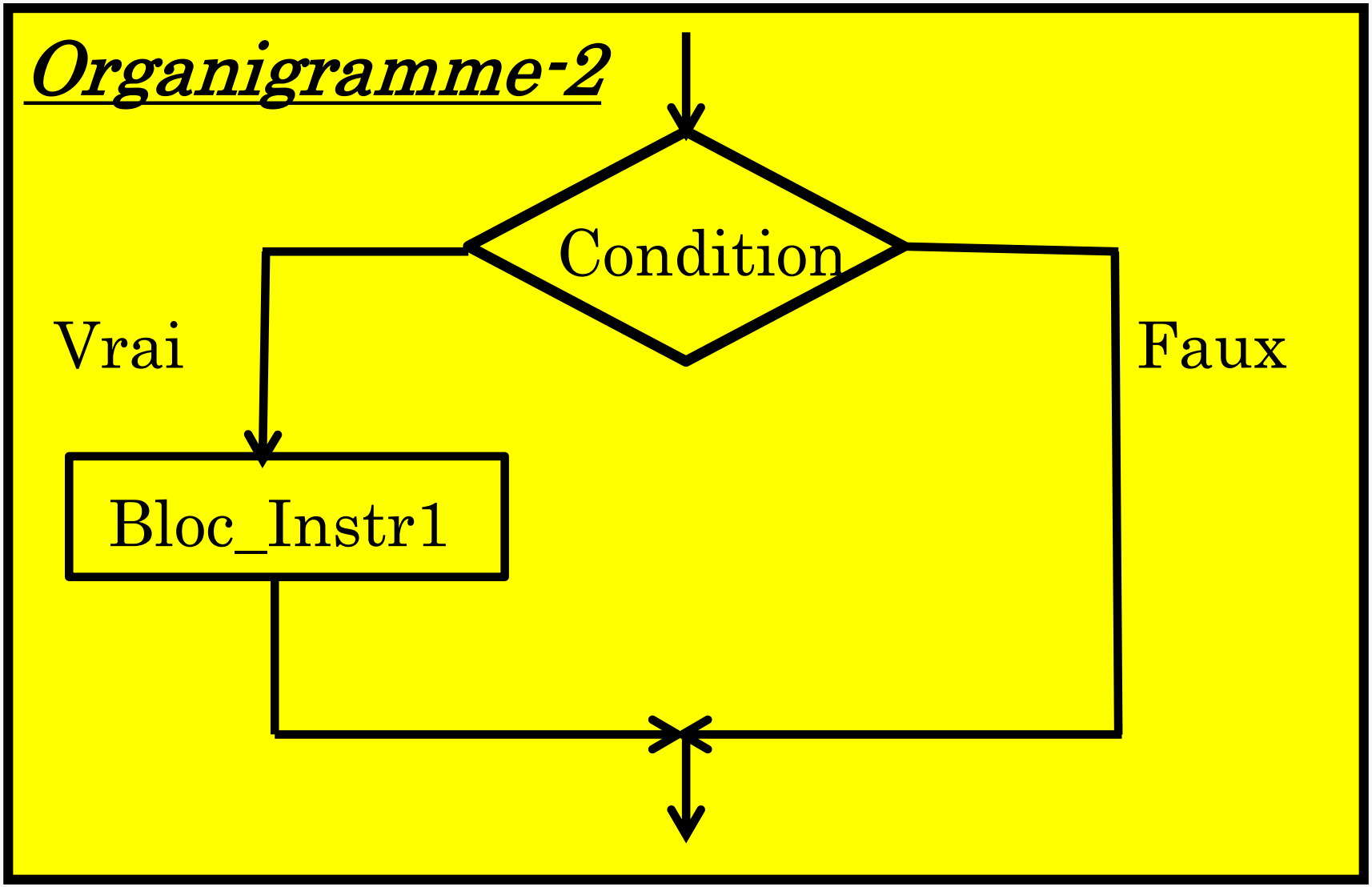
-**Condition** : expression booléenne (**V** ou **F**)

-**Bloc_Instructions1** et **Bloc_Instructions2** : peuvent être une instruction simple, des instructions séquentielles et/ou alternatives et/ou itératives

INSTRUCTIONS A 2 ALTERNATIVES



INSTRUCTIONS A 2 ALTERNATIVES



INSTRUCTIONS A 2 ALTERNATIVES

Exemple 1 :

Valeur absolue d'un entier a :

```
Si a>0 alors Abs←a
Sinon Abs←-a
FinSi
```

INSTRUCTIONS A 2 ALTERNATIVES

Exemple 2 :

Min, Max de 2 entiers a et b :

```
Si a>b alors Max←a
             Min←b
Sinon  Max←b
       Min←a
FinSi
```

Instructions alternatives imbriquées

Exemple 1 :

Lire 2 entiers a et b et afficher un message indiquant si a est supérieur strictement à b, b est supérieur strictement à a ou s'ils sont égaux.

Instructions alternatives imbriquées

Algorithme :

```
Objets : a, b : variables entières
Début
Afficher ("Donner 2 entiers : ")
Lire(a, b)
Si a>b alors Afficher(a, " > " , b)
Sinon  Si b>a  alors Afficher(a, " < " , b)
            Sinon Afficher(a, " = " , b)
            FinSi
FinSi
Fin.
```

Instructions alternatives imbriquées

Exemple 2 :

Lire 3 entiers a, b et c et afficher le maximum et le minimum entre eux en supposant qu'ils sont disjoints deux à deux.

Instructions alternatives imbriquées

Algorithme :

Objets : a, b, c : variables entières

Début :

```
Afficher("Donner 3 entiers disjoints : ")
Lire(a,b,c)
Si a>b alors  Si a>c alors Afficher("Max = " , a)
                Sinon Afficher("Max = " , c)
                FinSi
              Si b>c alors Afficher("Min = " , c)
              Sinon Afficher("Min = " , b)
              FinSi
```

Instructions alternatives imbriquées

Algorithme (Suite) :

```
Sinon  Si b>c alors Afficher("Max = " , b)
        Sinon Afficher("Max = " , c)
        FinSi
       Si a>c alors Afficher("Min = " , c)
       Sinon Afficher("Min = " , a)
       FinSi
FinSi
Fin.
```

EXERCICE D'APPLICATION
Affichage de la mention en fonction de la note

Ecrire un algorithme qui lit une note et affiche le message "admis avec mention" si cette note est supérieure à 14, "admis" si cette note est entre 12 et 14, "ajourné" si cette note est entre 10 et 12 et "exclus" si la note est inférieure à 10.

INSTRUCTIONS A PLUSIEURS ALTERNATIVES

- Se basent sur N conditions.
- Contiennent N Blocs d'instructions.
- Si la $i^{ème}$ condition est vraie, on exécute le $i^{ème}$ bloc d'instructions.

INSTRUCTIONS A PLUSIEURS ALTERNATIVES

- Syntaxe générale

```
Selon (variable)
    Si Val_1 : Bloc_Instructions1, Sortir
    Si Val_2 : Bloc_Instructions2, Sortir
    .........
    Si Val_N : Bloc_InstructionsN, Sortir
    [Si Autres : Bloc_Instructions]
FinSelon
```

INSTRUCTIONS A PLUSIEURS ALTERNATIVES

Exemple :

Tarifs d'un Zoo :

Code Visiteur	Type Visiteur	Tarif en DH
0	Enfant	10
1	Etudiant	12
2	Agé	15
3	Autres	25

INSTRUCTIONS A PLUSIEURS ALTERNATIVES

```
Algorithme :
Objets : Code, Tarif : entiers
Début :
Afficher ("Donner le code visiteur 0, 1, 2 ou 3 : ")
Lire(Code)
Selon (Code)
    Si 0 : Tarif←10, Sortir
    Si 1 : Tarif←12, Sortir
    Si 2 : Tarif←15, Sortir
    Si 3 : Tarif←25, Sortir
FinSelon
Afficher("Vous devez payer : ", Tarif, " DH")
Fin.
```

INSTRUCTIONS A PLUSIEURS ALTERNATIVES

Programme en C :

```
#include<stdio.h>
main()
{ int Code, Tarif;
  printf("Donner le type 0, 1, 2 ou 3 : ");
  scanf("%d",&Code);
  switch(Code)
   { case 0 : Tarif=10; break;
     case 1 : Tarif=12; break;
     case 2 : Tarif=15; break;
     case 3 : Tarif=25; break;
   }
printf("Vous devez payer %d DH", Tarif); }
```

Correction de l'exercice
Calcul de la moyenne de deux nombres

1) Donner les objets en entrée et en sortie.

2) Donner le modèle de résolution.

3) Dresser l'algorithme.

4) Ecrire le programme en langage C.

Correction de l'exercice
Calcul de la moyenne de deux nombres

1) Les objets en entrée :

a, b : variables entières

Les objets en sortie :

M : variable réelle.

2) Modèle de résolution :

M=(a+b)/2.0

Correction de l'exercice
Calcul de la moyenne de deux nombres

3) Algorithme:

Objets : a, b : variables entières

M : variable réelle

Début:

Afficher("Donner 2 entiers : ")

Lire(a, b)

Calculer M ← (a+b)/2.0

Afficher("Leur moyenne est : ", M)

Fin.

Correction de l'exercice
Calcul de la moyenne de deux nombres

4) Programme en langage C:

```
#include<stdio.h>
main()
{ int a, b; float M;
  printf("Donner 2 entiers :");
  scanf("%d%d", &a, &b);
  M=(a+b)/2.0;
  printf("Leur moyenne est %f", M);
}
```

EXERCICE D'APPLICATION
Affichage de la mention en fonction de la note

Ecrire un algorithme qui lit une note et affiche le message "admis avec mention" si cette note est supérieure à 14, "admis" si cette note est entre 12 et 14, "ajourné" si cette note est entre 10 et 12 et "exclus" si la note est inférieure à 10.

Correction de l'exercice *Affichage de la mention en Fonction de la note*

Algorithme:

Objets : N : variable réelle
Début :
Afficher("Donner la note :")
Lire(N)
Si N>=14 alors Afficher("Admis avec mention")
Sinon Si N>=12 alors Afficher("Admis")
Sinon Si N>=10 alors Afficher("Ajourné")
Sinon Afficher("exclus")
FinSi
FinSi
FinSi
Fin.

Correction de l'exercice *Affichage de la mention en Fonction de la note*

Programme en C :

```
#include<stdio.h>
main()
{ float N;
  printf("Donner la note :");
  scanf("%f", &N);
  if (N>=14) printf("Admis avec mention");
  else if (N>=12) printf("Admis");
        else if (N>=10) printf("Ajourné");
              else printf("exclus");
}
```

INSTRUCTIONS ITERATIVES (Boucles)

- Ecrites une seule fois.
- S'exécutent plusieurs fois.
- Sont caractérisées par :
  - La **condition de sortie** qui doit être valide (sinon boucle infinie),
  - Le **corps** de la boucle qui contient les instructions qui doivent se répéter.

INSTRUCTIONS ITERATIVES (Boucles)

- On distingue deux types :

– se basant sur le **nombre d'itérations** (On connait au départ le nombre de répétitions du corps de la boucle),

– se basant sur une **condition** (On ne connait pas le nombre de répétitions).

INSTRUCTIONS ITERATIVES
(se basant sur le nombre d'itérations)

- Utilisent une variable appelée compteur de boucle.
- Le compteur de boucle est une variable entière parcourant un intervalle [Min, Max] avec un pas donné.

INSTRUCTIONS ITERATIVES
(se basant sur le nombre d'itérations)

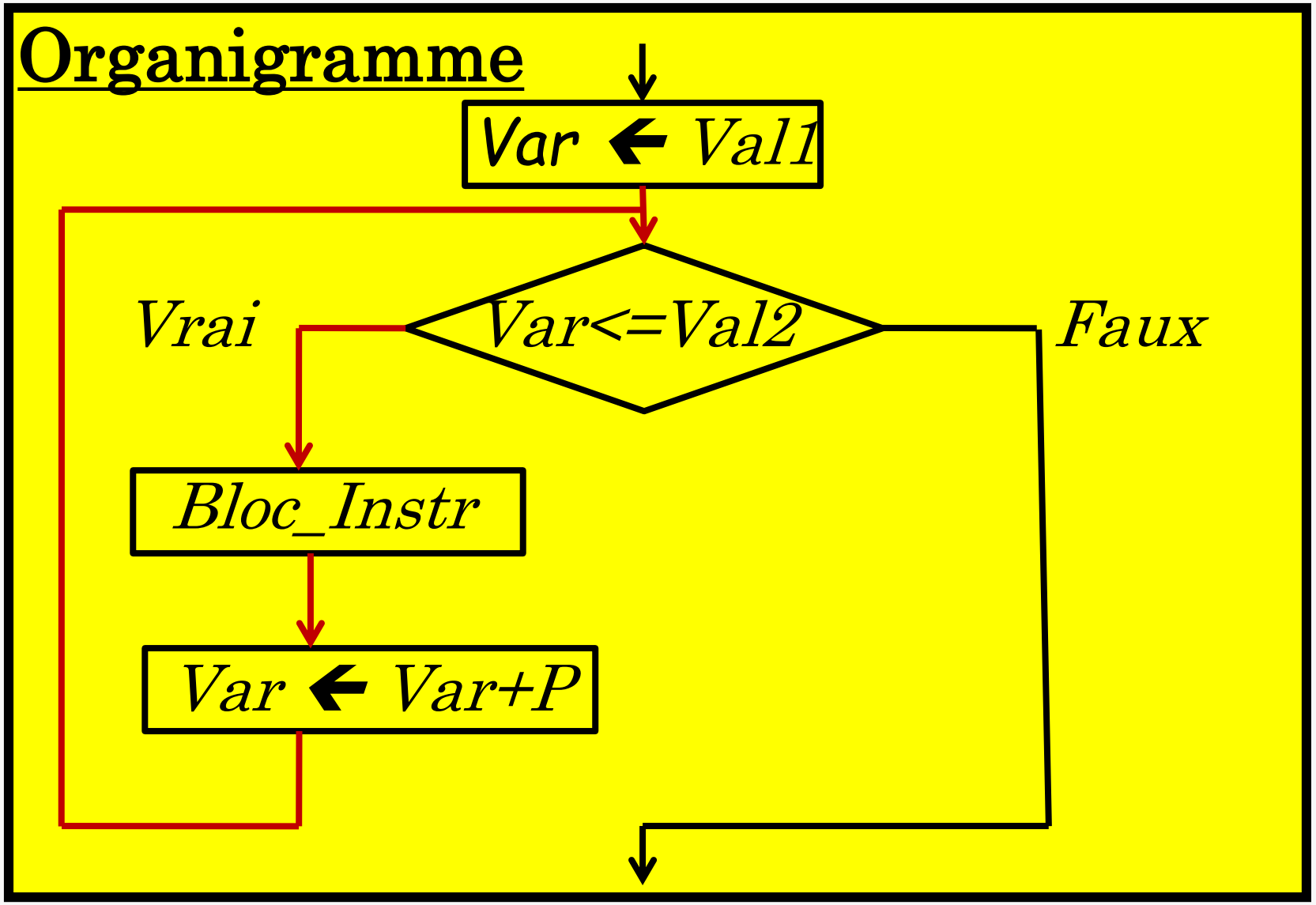
Syntaxe générale

Pour *Var ← Val1 jusqu'à Val2 [Pas=P] faire*
Bloc_Instructions
Fin-Pour

Avec :

- ***Val1***, ***Val2*** et ***P*** : des valeurs entières
- P : a par défaut la valeur 1
- ***Bloc_Instructions*** : peut être une instruction simple, des instructions séquentielles et/ou alternatives et/ou itératives.

INSTRUCTIONS ITERATIVES
(se basant sur le nombre d'itérations)



INSTRUCTIONS ITERATIVES
(se basant sur le nombre d'itérations)

Exemple 1:

Lire 10 notes et calculer leur moyenne.

Modèle :

Moyenne = la Somme des Notes divisée par le nombre de Notes

Exemple 1(Suite)

Variables à utiliser :

N : Note (réel),

MN : Moyenne des Notes (réel)

I : Compteur de la Boucle (entier)

Constante à utiliser :

NN=10 : Nombre de Notes

Exemple1 (Suite) : Algorithme

```
Objets : N, MN : variables réelles,
         I : variable entière,
         NN : constante entière=10,
Début :
MN←0
Pour I←1 jusqu'à NN faire
        Afficher("Donner une note :")
        Lire(N)
        MN←MN+N
Fin-Pour
MN←MN/NN
Afficher("La moyenne de ces notes est :", MN)
Fin.
```

Exemple 1(Suite) : Programme

```
#include<stdio.h>
#define NN 10  /* Constante */
main()
{ float N, MN; int I;
  MN=0;
  for (I=1 ; I<=NN ; I++)
        { printf("Donner une note :");
          scanf("%f",&N);  MN=MN+N;
        }
MN=MN/NN;
printf("La moyenne de ces notes est :%f", MN);
}
```

INSTRUCTIONS ITERATIVES
(se basant sur le nombre d'itérations)

Exemple 2:

Lire 2 entiers A et B et afficher les nombres compris strictement entre A et B ainsi que leurs carrés et leurs cubes.

Exemple 2(Suite)

Variables à utiliser :

A, B : Entiers à lire (**Entrées**)

I : Compteur de la boucle représentant aussi le $I^{ème}$ entier entre A et B

CA : Carré du $I^{ème}$ entier entre A et B

CU : Cube du $I^{ème}$ entier entre A et B

CA, CU, I : des **Sorties**

Exemple2 (Suite) : Algorithme

```
Objets : A, B, I, CA, CU : variables entières,
Début :
Afficher("Donner 2 entiers A et B avec A < B :")
Lire(A, B)
Pour I←A+1 jusqu'à B-1 faire
  Calculer CA ← I*I
  Calculer CU ← CA*I
  Afficher("L'entier est : ", I)
  Afficher("Son carré est : ", CA)
  Afficher("Son cube est : ", CU)
Fin-Pour
Fin.
```

Exemple 1(Suite) : Programme

```
#include<stdio.h>
main()
{ int A, B, I, CA, CU;
  printf("Donner 2 entiers A et B avec A < B :");
  scanf("%d%d", &A, &B);
  for (I=A+1 ; I<B ; I++)
        {CA = I*I; CU = CA*I;
         printf("L'entier est :%d", I)
         printf("\nSon carré est : %d", CA)
         printf("\nSon cube est : %d", CU)
        }
}
```

INSTRUCTIONS ITERATIVES
(se basant sur une condition)

- Se basent sur une condition (expression booléenne)
- N'utilisent pas de compteur mais l'utilisateur peut en définir en cas de besoin.

INSTRUCTIONS ITERATIVES
(se basant sur une condition)

2 types :

- On teste la condition **avant** d'entrer dans la boucle

  ⇒ Nombre d'itérations **>=** 0

- On exécute la boucle, **ensuite** on teste la condition

  ⇒ Nombre d'itérations > 0

INSTRUCTIONS ITERATIVES
se basant sur une condition

Syntaxe-1 :

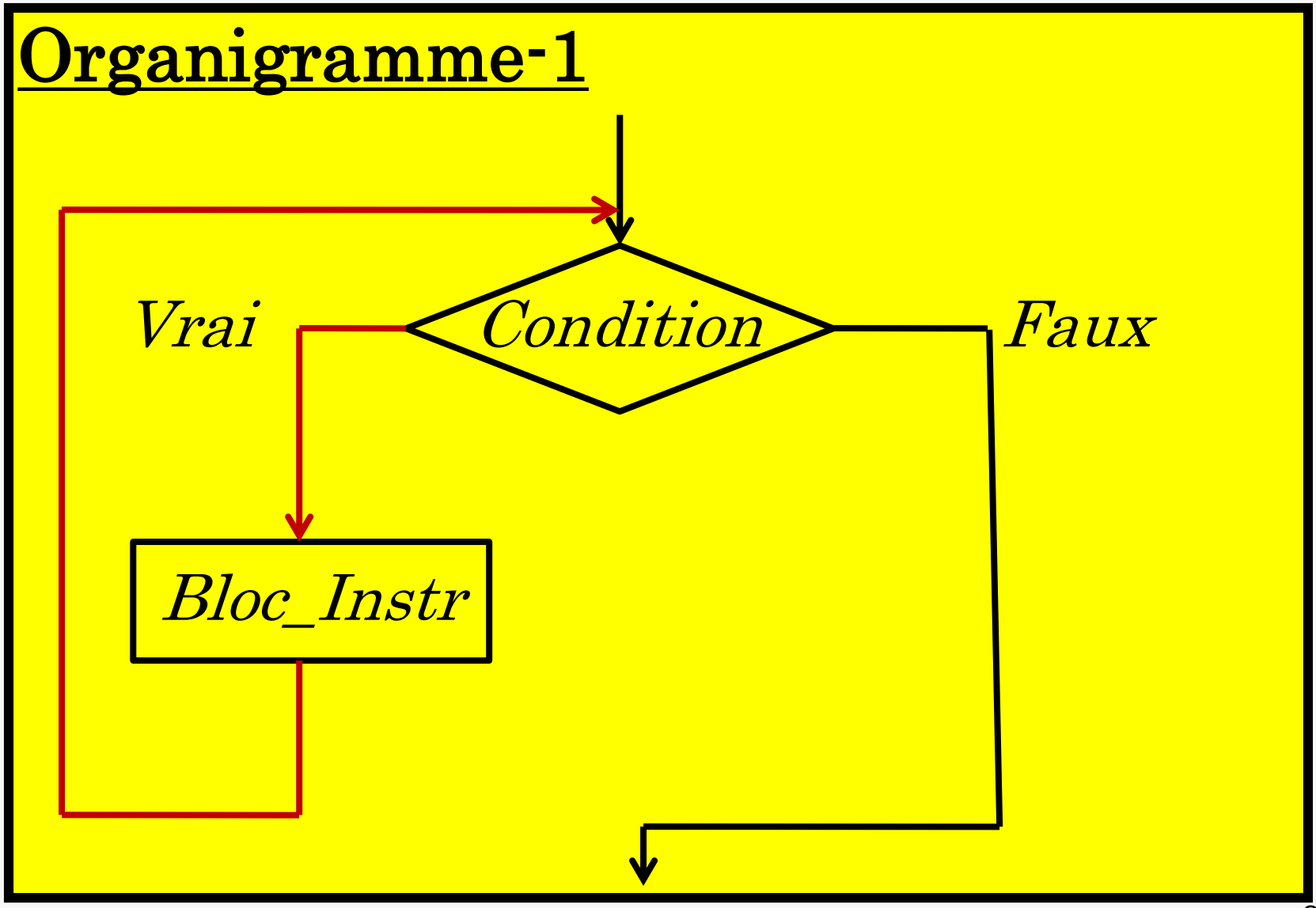
```
Tant que (Condition)
        Bloc_Instructions
Fin-Tant-que
```

Avec :

- **Condition** : expression booléenne
- **Bloc_Instructions** : peut être une instruction simple, des instructions séquentielles et/ou alternatives et/ou itératives.

INSTRUCTIONS ITERATIVES

se basant sur une condition



INSTRUCTIONS ITERATIVES
se basant sur une condition

Syntaxe-2 :

Répéter

Bloc_Instructions

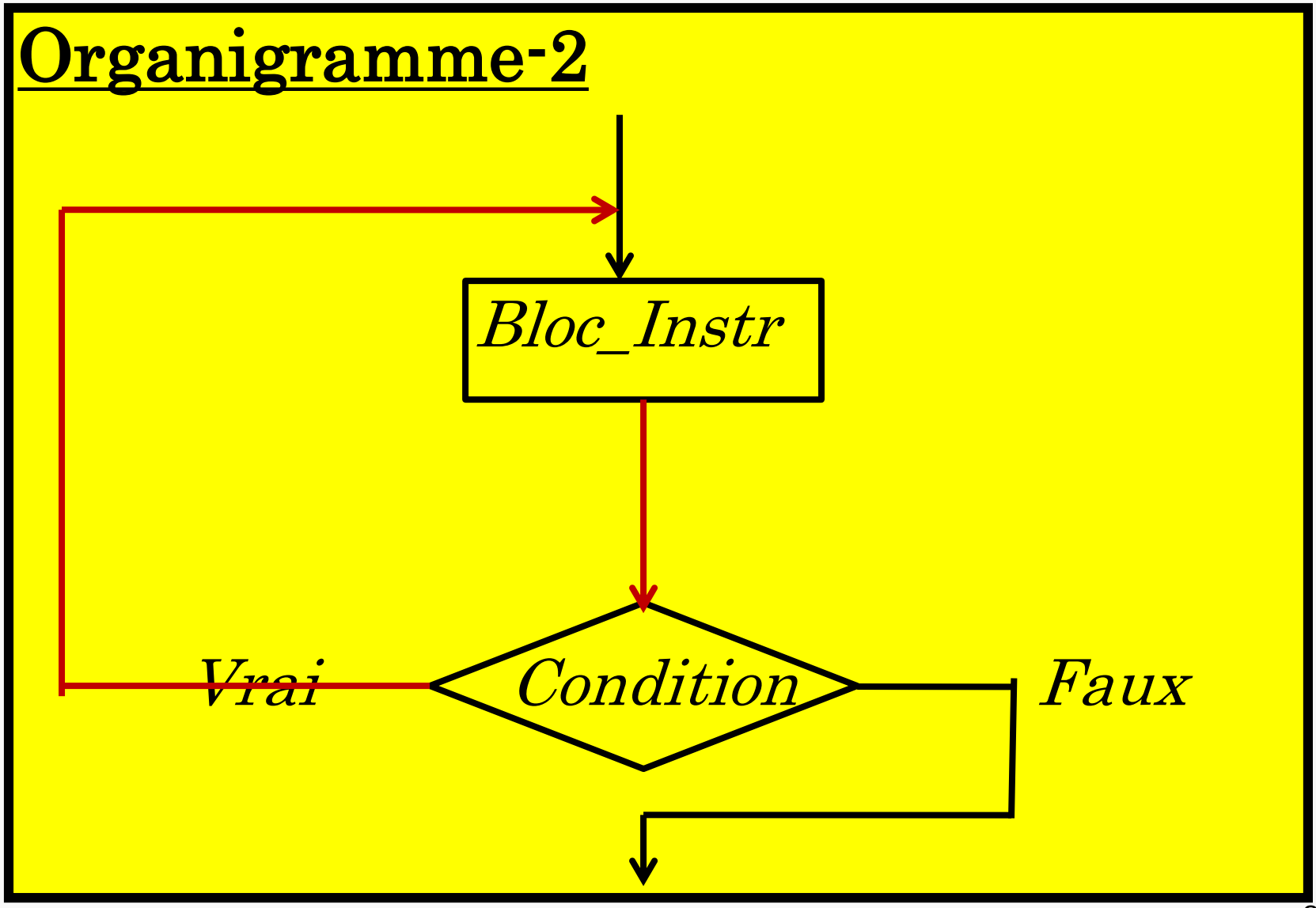
Tant que (Condition)

Avec :

- **Condition** : expression booléenne
- **Bloc_Instructions** : peut être une instruction simple, des instructions séquentielles et/ou alternatives et/ou itératives.

INSTRUCTIONS ITERATIVES

se basant sur une condition



INSTRUCTIONS ITERATIVES
se basant sur une condition

Exemple 1:

- Lire des entiers jusqu'à ce qu'on donne 0 puis afficher le nombre d'entiers lus ainsi que leur somme.
- Besoin d'un compteur : entier initialisé à 0 et incrémenté au besoin.

*Exemple1 (Suite) : Algorithme Avec **Tant-Que***

```
Objets : E, NE, SE : variables entières
Début :
        SE←0; NE←0; E←1
Tant que (E≠0) faire
   Afficher("Donner un entier et 0 pour sortir :")
   Lire(E)
   Calculer SE←SE+E
   Si E≠0 alors Calculer NE←NE+1
   FinSi
Fin-Tant-que
  Afficher("Le nombre d'entiers lus est :", NE)
  Afficher("Leur somme est :", SE)
Fin.
```

Exemple1 (Suite) : Programme en C

```
#include<stdio.h>
main()
{ int E, SE, NE;  SE=0; NE=0; E=1;
  while (E!=0)
   {printf("Donner un entier et 0 pour sortir :");
    scanf("%d", &E);
    SE=SE+E;
    if (E!=0) NE=NE+1;
    }
  printf("Le nombre d'entiers lus est : %d", NE);
  printf("\nLeur somme est :%d", SE);
}
```

*Exemple1 : Algorithme Avec **Répéter-Tant-Que***

```
Objets : E, NE, SE : variables entières,
Début :
SE←0; NE←0   /* Pas besoin de E←1; */
Répéter
   Afficher("Donner un entier et 0 pour sortir :")
   Lire(E)
   Calculer SE←SE+E
   Si E≠0 alors Calculer NE←NE+1 FinSi
Tant que (E≠0)
Afficher("Le nombre d'entiers lus est :", NE)
Afficher("Leur somme est :", SE)
Fin.
```

Exemple1 : Programme en C

```
#include<stdio.h>
main()
{ int E, SE, NE;  SE=0; NE=0;
  do {
        printf("Donner un entier et 0 pour sortir :");
        scanf("%d", &E);
        SE=SE+E;
        if (E!=0) NE=NE+1;
     }
  while (E!=0);
  printf("Le nombre d'entiers lus est : %d", NE);
  printf("\nLeur somme est :%d", SE);
}
```

INSTRUCTIONS ITERATIVES
se basant sur une condition

Exemple 2:

- Lire les Prix Unitaires et les Quantités de produits achetés et afficher le total à payer.
- On boucle tant que l'utilisateur le désire.

*Exemple2 (Suite) : Algorithme Avec **Tant-Que***

```
Objets : PU, PT : variables réelles
         Qte, Choix : variables entières
Début :
PT←0; Choix←1
Tant que (Choix≠0) faire
  Afficher("Donner Le PU et la Qte achetée:")
  Lire(PU, Qte)
  Calculer PT←PT+(PU*Qte)
 Afficher("Taper 1 pour Continuer ou 0 pour sortir :")
  Lire(Choix)
Fin-Tant-que
Afficher("Le prix total à payer est :", PT, "DH")
Fin.
```

Exemple2 (Suite) : Programme en C

```
#include<stdio.h>
main()
{ float PU, PT; int Qte, Choix;
 PT=0; Choix=1;
 while (Choix!=0) {
  printf("Donner Le PU et la Qte achetée:");
  scanf("%f%d", &PU, &Qte);
  PT=PT+(PU*Qte);
 printf("Taper 1 pour Continuer ou 0 pour sortir :")
 scanf("%d", &Choix);
}
printf("Le prix total à payer est :%f DH", PT);
}
```

*Exemple2 : Algorithme Avec **Répéter-Tant-Que***

```
Objets : PU, PT : variables réelles;
         Qte, Choix : variables entières,
Début :
PT←0;
Répéter
 Afficher("Donner Le PU et la Qte achetée:")
  Lire(PU, Qte)
  Calculer PT←PT+(PU*Qte)
 Afficher("Taper 1 pour Continuer ou 0 pour sortir :")
  Lire(Choix)
Tant que (Choix≠0)
Afficher("Le prix total à payer est :", PT, "DH")
Fin.
```

Exemple2 : Programme en C

```
#include<stdio.h>
main()
{ float PU, PT; int Qte, Choix;
 PT=0;
 do {
  printf("Donner Le PU et la Qte achetée:");
  scanf("%f%d", &PU, &Qte);
  PT=PT+(PU*Qte);
 printf("Taper 1 pour Continuer ou 0 pour sortir :")
 scanf("%d", &Choix);
}
while (Choix!=0);
printf("Le prix total à payer est :%f DH", PT);
}
```

Exercice d'application N°1

Lister tous les nombres entiers pairs inférieurs ou égaux à un nombre N lu à partir du clavier.

- Utiliser les 3 boucles séparément
- Donner l'algorithme et le programme en C pour chaque cas.

Exercice d'application N°2

Lire un entier N et afficher sa table de multiplication.

- Utiliser les 3 boucles séparément.
- Donner l'algorithme et le programme en C pour chaque cas.

Attention aux erreurs logiques dans les boucles

- **Condition de sortie non valide** : on risque d'avoir une boucle infinie.
- **Corps de la boucle incorrect** : on risque de répéter des instructions qui ne doivent pas être répétées ou l'inverse.

Exemple 1 : Min et Max d'entiers

Algorithme

Objet : N, Min, Max : variables entières

Début:

Afficher("Donner un entier et 0 pour sortir :")

Lire(N)

Min←N; Max←N

Tant Que (N ≠ 0) faire ... Boucle Infinie Si !!...

Afficher("Donner un entier et 0 pour sortir :")

Si (Min > N) Min←N FinSi

Si (Max < N) Max←N FinSi

Fin-Tant-que

Afficher("Le Min est :", Min, " Le max est : ", Max)

Fin.

Exemple 2 : Norme d'un Vecteur

Algorithme

Objet : I, C : variable entière
X, N : variables réelles

Début:

Afficher("Donner la taille du Vecteur :")
Lire(C); N←0
Pour I←1 jusqu'à C faire
  Afficher("Donner la coordonnée :", I, " : ")
  Lire(X)
Fin-Pour
Calculer N←N+(X*X) ... N = ... !! ...
Afficher("La Norme est : ", RacineCarrée(N))
Fin.

Remarque

- Les 3 boucles sont équivalentes avec quelques modifications.
- On choisit la boucle au nombre d'itérations si on connait au départ combien de fois on va boucler.
- On utilise la boucle se basant sur une condition lorsque la sortie de la boucle est conditionnée par une condition et on ne connait pas au départ le nombre de répétitions.

Exemple : Somme des carrés des entiers inférieurs ou égaux à N

Algorithme avec la boucle Pour

Objet : N, S, I : variables entières
Début :
Afficher("Donner un entier :")
Lire(N); S←0
Pour I←1 jusqu'à N faire
    Calculer S←S+I*I
Fin-Pour
Afficher("La somme des carrés des entiers
            inférieurs ou égaux à ", N, " est : ", S)
Fin.

Exemple : Somme des carrés des entiers inférieurs ou égaux à N

Algorithme avec la boucle Tant-Que

Objet : N, S, I : variables entières
Début :
Afficher("Donner un entier :")
Lire(N); S←0; I←1
Tant Que (I<=N) faire
    Calculer S←S+I*I
    I←I+1
Fin-Tant-Que
Afficher("La somme des carrés des entiers
            inférieurs ou égaux à ", N, " est : ", S)
Fin.

Exemple : Somme des carrés des entiers inférieurs ou égaux à N

Algorithme avec la boucle Répéter

Objet : N, S, I : variables entières
Début :
Afficher("Donner un entier :")
Lire(N); S←0; I←1
Répéter
    Calculer S←S+I*I
    I←I+1
Tant Que (I<=N)
Afficher("La somme des carrés des entiers
                inférieurs ou égaux à ", N, " est : ", S)
Fin.

Les boucles imbriquées

- Le corps d'une boucle peut contenir une autre boucle
- On parle alors de boucles imbriquées
- Le système ne passe à l'itération suivante d'une boucle que lorsqu'il a terminé l'exécution de toute la boucle interne

Exemple 1: Affichage des nombres premiers inférieurs à 100

- Un entier N est premier si tous les entiers compris entre 2 et Racine(N) ne sont pas des diviseurs de N
- On a besoin d'une boucle pour parcourir l'intervalle [2, 100]
- Dans cette boucle, on aura besoin d'une autre boucle pour parcourir l'intervalle [2, Racine(N)]

Exemple 1: Affichage des nombres premiers inférieurs à 100

```
Objet : N, I, R : Variables entières
Début :
Pour N←2 jusqu'à 100 faire
        I←2;
        Calculer R←RacineCarrée(N)
        Tant que(N mod I ≠ 0 et I <= R) faire
                Calculer I←I+1
        Fin-Tant-que
        Si (I > R) alors
                Afficher(N, " est premier :")
        Fin-Si
Fin-Pour
Fin.
```

Exemple 2: Moyenne générale des étudiants

- On saisit les notes et les coefficients d'un étudiant et on calcule sa moyenne générale
- On répète cela tant que l'utilisateur le désire
- On aura donc besoin de 2 boucles : une pour parcourir les étudiants et l'autre pour saisir les notes et les coefficients.

Exemple 2: Moyenne générale des étudiants

```
Objet : N, I, R, C, SC : Variables entières
        No, SNC : variables réelles
Début :
Répéter
   Afficher("Donner le nombre d'examens :")
   Lire(N);    SC←0;    SNC←0
   Pour I←1 jusqu'à N faire
         Afficher("Donner la note et le coefficient:")
         Lire(No, C)
         Calculer SNC←SNC+(No*C)
         Calculer SC←SC+C
   Fin-Pour
   Afficher("La moyenne générale est :", (SNC/SC))
   Afficher("Taper 1 pour continuer:")
   Lire(R)
Tant que (R=1)
Fin.
```

VI- STRUCTURES DE DONNEES

- **Variable simple :**

  ne peut contenir qu'une seule information (car elle a une seule case mémoire)

- **Variable structurée :**

  peut contenir plusieurs informations (car elle possède plusieurs cases mémoires)

VI- STRUCTURES DE DONNEES

2 Types de Structures de Données :

- Structures de données ***homogènes*** :

  Tous les éléments sont de même type

  Exemple : tableaux

- Structures de données ***hétérogènes*** :

  Les éléments peuvent être de types différents

  Exemple : structures (enregistrements)

TABLEAUX

- Structure de données homogènes
- Caractérisé par un **identificateur** (**nom**) , le **type** des éléments et la **taille** (Nombre d'éléments) qui doit être une **constante** entière positive (dû à la réservation d'espace mémoire)

TABLEAUX

- Chaque élément est repéré dans le tableau par un **indice.**
- Un indice est un entier variant de 0 à la taille moins un (Taille-1) (En Pascal de 1 à Taille)

TABLEAUX (Exemple)

Déclaration Algorithmique:

N : Constante entière égale à 10

T : Tableau d'entiers de taille N

Déclaration En C :

```
#define N 10
int T[N];
```

OPERATIONS SUR LES TABLEAUX

- Les tableaux s'utilisent en général avec la **boucle au nombre d'itérations**
- **Opérations :**

Lecture, Affichage, Somme des éléments, Min/Max des éléments, Tri des éléments, Somme de deux tableaux, …

LECTURE DES ELEMENTS D'UN TABLEAU

Algorithme :

```
i : variable entière
Pour i←0 jusqu'à (N-1) faire
  Afficher("Donner l'élément d'indice:",i)
  Lire(T[i])
Fin-Pour
```

LECTURE DES ELEMENTS D'UN TABLEAU

Programme en C :

```
int i;
for (i=0; i<N; i++)
  {printf("Donner l'élément d'indice %d:",i);
   scanf("%d", &T[i]);
   }
```

AFFICHAGE DES ELEMENTS D'UN TABLEAU

Algorithme :

```
i : variable entière
Afficher("les éléments du tableau sont:")
Pour i←0 jusqu'à (N-1) faire
        Afficher(T[i]," ")
Fin-Pour
```

AFFICHAGE DES ELEMENTS D'UN TABLEAU

Programme en C :

```
int i;
printf("les éléments du tableau sont:");
for (i=0; i<N; i++)
   printf("%d ",T[i]);
```

SOMME DES ELEMENTS D'UN TABLEAU

Algorithme :

Objet : S, i : variables entières

S←0

Pour i←0 jusqu'à (N-1) faire

    Calculer S←S+T[i]

Fin-Pour

Afficher("la somme est : ", S)

SOMME DES ELEMENTS D'UN TABLEAU

Programme en C :

```
int i;
int S=0;
for (i=0; i<N; i++)
        S=S+T[i];
printf("La somme est : %d",S);
```

Min/Max DES ELEMENTS D'UN TABLEAU

Algorithme :

Objets : i, Min, Max : variables entières
Min←T[0]
Max←T[0]
Pour i←1 jusqu'à (N-1) faire
    Si (Min>T[i]) alors Min←T[i] FinSi
    Si (Max<T[i]) alors Max←T[i] FinSi
Fin-Pour
Afficher("le Minimum est : ", Min)
Afficher("le Maximum est : ", Max)

Min/Max DES ELEMENTS D'UN TABLEAU

Programme en C :

```
int i, Min, Max;
Min=T[0]; Max=T[0];
for (i=1; i<N; i++)
        { if (Min>T[i]) Min=T[i];
          if (Max<T[i]) Max=T[i];
        }
printf("\nLe minimum est : %d",Min);
printf("\nLe maximum est : %d",Max);
```

SOMME DE DEUX TABLEAUX

Algorithme :

T1,T2,T3: tableaux d'entiers de taille N

//Lecture de T1 et T2

Pour i←0 jusqu'à (N-1) faire

    Calculer T3[i]←T1[i]+T2[i]

Fin-Pour

Afficher("le tableau somme est : ")

//Affichage de T3

SOMME DE DEUX TABLEAUX

Programme en C :

```
#define N 10
int i, T1[N], T2[N], T3[N];
/* Lecture de T1 et T2 */
for (i=0; i<N; i++)
        T3[i]=T1[i]+T2[i];
/* Affichage de T3 */
```

RECHERCHE D'UN ENTIER DANS UN TABLEAU

- L'entier existe ou non ?
- 1ère occurrence de l'entier s'il existe
- Le nombre d'occurrences de l'entier
- Les indices des occurrences de l'entier
- Recherche dans un tableau trié

L'entier **A** existe ou non ?

```
Algorithme
i←0
Tant-Que(i<N et T[i]≠A) faire
        Calculer i←i+1
Fin-Tant-que
Si (i<N) alors
      Afficher(A," existe dans le tableau")
Sinon Afficher(A," n'existe pas")
FinSi
```

L'entier **A** existe ou non

Programme en C

```
i=0;
while(i<N &&T[i]!=A)
        { i=i+1; }
if (i<N)
   printf("%d existe dans le tableau", A);
else printf("%d n'existe pas", A);
```

1ère occurrence d'un entier A

Algorithme

i←0

Tant-Que(i<N et T[i]≠A) faire

    Calculer i←i+1

Fin-Tant-que

Si (i<N) alors

Afficher("1ère occurrence de:",A," est:",i)

Sinon Afficher(A," n'existe pas")

FinSi

1ère occurrence d'un entier A

Programme en C

```
i=0;
while(i<N &&T[i]!=A)
        { i=i+1; }
if (i<N)
 printf("1ère occurrence de:%d est:%d",A,i);
else printf("%d n'existe pas",A);
```

Nombre d'occurrences de **A**

Algorithme

i, C : variables entières

C←0

Pour i←0 jusqu'à (N-1) faire

    Si (T[i]=A) alors Calculer C←C+1

    FinSi

Fin-Pour

Afficher("Nombre d'occurrences de:"

    ,A," est:",C)

Nombre d'occurrences de A

Programme en C

```
int i, C;
C=0;
for(i=0; i<N; i++)
        { if (T[i]==A) C=C+1; }
printf("Nombre d'occurrences de:%d
          est:%d",A,C)
```

Indices des occurrences de A

Algorithme

```
i : variables entières
Afficher("L'entier:",A," existe dans les
                    indices suivants :")
Pour i←0 jusqu'à (N-1) faire
        Si (T[i]=A) alors Afficher(i," ")
        FinSi
Fin-Pour
```

Indices des occurrences de A

Programme en C

```
int i;
printf("L'entier:%d existe dans les
                indices suivants :", A)
for(i=0; i<N; i++)
      { if (T[i]==A) printf("%d ", i); }
```

Recherche dans un tableau trié

- Soit T un tableau d'entiers contenant N éléments ordonnés par ordre croissant,
- L'objectif est de rechercher dans ce tableau un entier A saisi au clavier,
- L'algorithme s'appelle Dichotomie
- 2 versions : itérative et récursive

Principe de l'Algorithme

-On compare A avec l'entier du milieu du tableau (d'indice MI),

-Si cet entier est égal à A c'est terminé

-S'il est supérieur strictement à A alors on cherche A dans le 1er sous-tableau (Indices 0 à (MI-1))

-S'il est inférieur strictement à A alors on cherche A dans le 2ème sous-tableau (Indices (MI+1) à (N-1))

-Si les bornes du tableau ne sont plus valides on sort (A n'existe pas dans le tableau)

Algorithme Dichotomie (itératif)

```
Objets : Bi, Bs, Mi, A : variables entières
         Trouvé : variable booléenne
Début:
Bi←0   Bs←(N-1)   Trouvé←faux
Tant Que ((Bi<=Bs) et (non Trouvé)) faire
        Calculer Mi←(Bs+Bi)/2
        Si (T[Mi]=A) alors Trouvé←vrai
        Sinon Si (T[Mi]>A) alors Calculer Bs←(Mi-1)
                Sinon Calculer Bi←(Mi+1)
                FinSi
        FinSi
Fin-Tant-Que
Si (Trouvé) alors Afficher(A," existe à l'indice:",Mi)
Sinon Afficher(A," n'existe pas")
FinSi
Fin.
```

Exercice d'application N°1

Ecrire un algorithme qui lit deux tableaux d'entiers T1 et T2 de taille 10 chacun et affiche les éléments existants dans T1 sans exister dans T2 (càd T1-T2) et les éléments existants dans T2 sans exister dans T1 (càd T2-T1).

Exercice d'application N°2

Ecrire un algorithme qui lit deux tableaux de réels P1 et P2 de taille 10 chacun comportant les coefficients de 2 polynômes de degré ?? et stocke la somme de ces 2 polynômes dans un 3ème polynôme P3 de degré ?? puis affiche P3.

Tri d'un tableau

Différents algorithmes de tri :

– Tri à Bulles

– Tri par sélection

– Tri par insertion

– Tri rapide (Quicksort)

– Tri par tas (Heapsort)

– Tri fusion

– …

Tri à Bulles

Principe de l'algorithme :

– consiste à faire remonter progressivement les plus grands éléments du tableau

– on parcourt le tableau et on compare les couples d'éléments successifs

– lorsque 2 éléments successifs ne sont pas ordonnés, on les permute

–Si on fait au moins une permutation dans un cycle, on doit refaire le cycle

–L'algorithme s'arrête lorsqu'on fait un cycle sans permutation.

Algorithme Tri à Bulles

```
Objets : I, P : variables entières
         R : variable booléenne
Début:
Répéter
R←faux
Pour I←0 jusqu'à (N-2) faire
        Si (T[I]>T[I+1]) alors
                P←T[I]
                T[I]←T[I+1]
                T[I+1]←P
                R←vrai
        FinSi
Fin-Pour
Tant Que (R=vrai)
Fin.
```

Exemple: Algorithme Tri à Bulles

5	2	8	6	10	1	4	3	9	7
2	5	6	8	1	4	3	9	7	10
2	5	6	1	4	3	8	7	9	10
2	5	1	4	3	6	7	8	9	10
2	1	4	3	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10

Exercice d'application N°3

Soit T un tableau d'entiers contenant les éléments suivants: 8 6 0 4 2 7 9 1 3 5

1) Donner le contenu du tableau après chaque itération de l'algorithme Tri à Bulles

2) Déduire le nombre d'itérations nécessaires pour que le tableau soit trié

Tri par sélection

Principe de l'algorithme :

–Rechercher le plus petit élément du tableau et l'échanger avec l'élément d'indice 0

–Rechercher le second plus petit élément et l'échanger avec l'élément d'indice 1

–Continuer de cette façon jusqu'à ce que le tableau soit entièrement trié

–On peut s'intéresser à rechercher le plus petit élément ou son indice

–Donc, pour chaque i, T[i]=…??

Algorithme Tri par sélection

```
Objets : I, J, P : variables entières
Début:
Pour I←0 jusqu'à (N-2) faire
        Pour J←I+1 jusqu'à (N-1) faire
                Si (T[I]>T[J]) alors
                        P←T[I]
                        T[I]←T[J]
                        T[J]←P
                FinSi
        Fin-Pour
Fin-Pour
Fin.
```

Exemple: Algorithme Tri par sélection

5	2	8	6	10	1	4	3	9	7
1	5	8	6	10	2	4	3	9	7
1	2	8	6	10	5	4	3	9	7
1	2	3	8	10	6	5	4	9	7
1	2	3	4	10	8	6	5	9	7
1	2	3	4	5	10	8	6	9	7
1	2	3	4	5	6	10	8	9	7
1	2	3	4	5	6	7	10	9	8
1	2	3	4	5	6	7	8	10	9
1	2	3	4	5	6	7	8	9	10

Exercice d'application N°4

Soit T un tableau d'entiers contenant les éléments suivants: 8 6 0 4 2 7 9 1 3 5

1) Donner le contenu du tableau après chaque itération de l'algorithme Tri par sélection

2) Déduire le nombre d'itérations nécessaires pour que le tableau soit trié

Tri par insertion

Principe de l'algorithme :

–Trier les 2 premiers éléments du tableau

–Insérer le 3ème élément à sa bonne place de telle manière à ce que les 3 premiers éléments soient triés

–Et ainsi de suite : insérer le i^{ème} élément à sa bonne place pour que les i premiers éléments soient triés

Algorithme Tri par insertion

```
Objets : I, J, X : variables entières
Début:
Pour I←1 jusqu'à (N-1)  faire
        X←T[I]
        J←I
        Tant que ((J>0) et (T[J-1] >X)) faire
                T[J]←T[J-1]
                Calculer J←J-1
        Fin-Tant-que
        T[J]←X
Fin-Pour
Fin.
```

Exemple: Algorithme Tri par insertion

5	2	8	6	10	1	4	3	9	7
2	5	8	6	10	1	4	3	9	7
2	5	8	6	10	1	4	3	9	7
2	5	6	8	10	1	4	3	9	7
2	5	6	8	10	1	4	3	9	7
1	2	5	6	8	10	4	3	9	7
1	2	4	5	6	8	10	3	9	7
1	2	3	4	5	6	8	10	9	7
1	2	3	4	5	6	8	9	10	7
1	2	3	4	5	6	7	8	9	10

Exercice d'application N°5

Soit T un tableau d'entiers contenant les éléments suivants: 8 6 0 4 2 7 9 1 3 5

1) Donner le contenu du tableau après chaque itération de l'algorithme Tri par insertion

2) Déduire le nombre d'itérations nécessaires pour que le tableau soit trié

Remarque sur les tableaux

Problème :

Taille constante ⇒ sur-utilisation ou sous-utilisation du tableau

Exemple :

N : constante entière = 10

T tableau d'entiers de taille N

- ***Si on a besoin de plus de 10 cases ?***
- ***Si on veut stocker juste 3 entiers ?***

Remarque sur les tableaux(suite)

Solution :

-Utiliser des tableaux dynamiques utilisant les **pointeurs** : réserver au besoin et libérer les espaces non utilisés (Cas des langages C et C++)

-Utiliser un type prédéfini dont la gestion de la mémoire est faite par le système (Cas des types prédéfinis du langage JAVA: Vector, ArrayList,…)

Les tableaux à 2 dimensions

-Sont aussi appelés **Matrices**

-Chaque élément est repéré par **2 indices**

-Ils sont manipulés en utilisant **2 boucles imbriquées**

-Une boucle pour parcourir les **lignes**

-L'autre boucle pour parcourir les **colonnes**

Exemple de tableaux à 2 dimensions

Déclaration:

N : constante entière égale à 10

M : constante entière égale à 20

T : tableau d'entiers de taille N×M

⇒ T est un tableau à 2 dimensions contenant 200 (10×20) éléments, càd une matrice formée de 10 lignes et 20 colonnes

Exemple de tableaux à 2 dimensions

Lecture :

```
I, J : variables entières
Pour I←0 jusqu'à (N-1) faire
        Pour J←0 jusqu'à (M-1) faire
                Afficher("Donner un entier:")
                Lire(T[I][J])
        Fin-Pour
Fin-Pour
```

Exemple de tableaux à 2 dimensions

Affichage : (Matricielle)

```
I, J : variables entières
Afficher("La matrice est:")
Pour I←0 jusqu'à (N-1) faire
        Pour J←0 jusqu'à (M-1) faire
                Afficher(T[I][J]," ")
        Fin-Pour
        Retourner à la ligne
Fin-Pour
```

Les structures

Définition:

Une structure est un ensemble d'informations **homogènes** (relatives à la même entité) qui peuvent être de types différents.

Exemples:

- **Etudiant**(Code, Nom, Prénom, Classe, …)
- **Matière**(Code, Nom, Coefficient)
- **Enseignant**(Nom, Prénom, Spécialité, …)
- **Date**(Jour, Mois, Année)
- **Examen**(Date, Matière, Etudiant, Note)

Les structures

Déclaration Algorithmique

Structure **NomStructure**

Champ1 : Type1

Champ2 : Type2

.....

ChampN : TypeN

Fin-Structure

Rmq : Cette déclaration ne réserve ***aucun espace mémoire***

Exemple

Structure représentant les Vecteurs dans l'espace à 3 dimensions

```
Structure Vecteur3d
    X : réel
    Y : réel
    Z : réel
Fin-Structure
```

Les variables structurées

Déclaration Algorithmique

Structure NomStructure **NomVariablestructurée**

Remarque :

Cette déclaration réserve l'espace mémoire nécessaire pour tous les champs de cette variable structurée

Exemple

Déclaration d'un Vecteur V1 comme étant une variable structurée de la structure Vecteur3d précédente :

Structure Vecteur3d V1

⇒ Le système réserve l'espace mémoire nécessaire pour les champs X, Y et Z de la variable structurée V1

Accès aux champs

Pour accéder aux champs d'une variable structurée, on utilise l'opérateur "."

Exemple :

V1.X représente le champ X de V1

V1.Y représente le champ Y de V1

V1.Z représente le champ Z de V1

Exemple : Lecture et affichage d'une variable structurée

```
Structure Vecteur3d
        X, Y, Z : réels
Fin-Structure
Objet : V1 : variable de type Vecteur3d
Début :
   Afficher("Donner les coordonnées du vecteur :")
   Lire(V1.X, V1.Y, V1.Z)
   Afficher("Les coordonnées du vecteur sont :")
   Afficher("(",V1.X, ",", V1.Y, ",", V1.Z, ")")
Fin.
```

Exercice d'application N°1

Ecrire un algorithme dans lequel :

- On définit la structure Complexe comportant 2 champs R (Réel) et I (Imaginaire) de type réel
- On déclare 2 Complexes C1 et C2
- On lit leurs champs respectifs
- On affiche C1, C2, leur somme et leur produit

Structures de données particulières

On peut manipuler des structures de données particulières telles que :

- Des tableaux de structures
- Des structures comportant des tableaux
- Des structures comportant d'autres structures
- Des Piles et des Files

Tableaux de structures

Exemple

Structure Vecteur3d

X, Y, Z : réels

Fin-Structure

Objet : N : constante entière égale à 10

T : tableau de Vecteur3d de taille N

i : variable entière

/* Les éléments de T sont des variables structurées de type la structure Vecteur3d */

Tableaux de structures

```
Début :
Pour i←0 jusqu'à (N-1) faire
 Afficher("Donner les coordonnées du vecteur :")
  Lire(T[i].X, T[i].Y, T[i].Z)
Fin-Pour
Pour i←0 jusqu'à (N-1) faire
    Afficher("Les coordonnées du vecteur sont :")
    Afficher("(",T[i].X, ",", T[i].Y, ",", T[i].Z, ")")
    Retourner à la ligne
Fin.
```

Tableaux de structures

Remarque :

Dans cet exemple, T[i] représente le $i^{ème}$ élément du tableau T, c'est une variable structurée de type la structure Vecteur3d, donc il possède 3 champs : X, Y et Z, et ceci pour tout i allant de 0 jusqu'à (N-1)

Structures comportant des tableaux

Exemple

```
Structure Etudiant
    Nom, Prénom : chaines de caractères
    TN : tableau de réels de taille 10
Fin-Structure
//TN: comporte les notes de l'Etudiant
Objet : E : variable de type Etudiant
        i : variable entière

/*TN est un champ de la structure E, il est
    déclaré comme étant un tableau de
    réels*/
```

Structures comportant des tableaux

```
Début :
Afficher("Donner le nom et le prénom de l'Etudiant:")
Lire(E.Nom, E.Prenom)
Pour i←0 jusqu'à 9 faire
        Afficher("Donner la note :")
        Lire(E.TN[i])
Fin-Pour
Afficher("Le nom et le prénom de l'Etudiant sont:")
Afficher(E.Nom, "   ", E.Prenom)
Afficher("Ses notes sont:")
Pour i←0 jusqu'à 9 faire
        Afficher(E.TN[i] , "   ")
Fin-Pour
Fin.
```

Structures comportant des tableaux

Remarque :

Dans cet exemple, TN a été déclaré comme étant un champ de la structure Etudiant. Donc, et puisque E a été déclaré comme variable structurée de type la structure Etudiant, E.TN est un tableau.

Structures comportant des structures

Exemple

```
Structure Date
    Jour, Mois, Année : entiers
Fin-Structure
Structure Personne
    Nom, Prénom : chaines de caractères
    DN : variable de type Date //Date Naissance
Fin-Structure
Objet : P : variable de type Personne

/*P est une variable structurée de type la
    structure Personne, elle comporte entre
    autres un champ DN qui est une variable
    structurée de type la structure Date*/
```

Structures comportant des structures

```
Début :
  Afficher("Donner le nom et le prénom :")
  Lire(P.Nom, P.Prenom)
  Afficher("Donner le jour de naissance :")
  Lire(P.DN.Jour)
  Afficher("Donner le mois de naissance :")
  Lire(P.DN.Mois)
  Afficher("Donner l'année de naissance :")
  Lire(P.DN.Année)
  Afficher("Nom et Prénom :", P.Nom, " ", P.Prénom)
  Afficher("Date naissance est :")
  Afficher(P.DN.Jour,"/", P.DN.Mois ,"/", P.DN.Année)
Fin.
```

Structures comportant des structures

Remarque :

Dans cet exemple, P a été déclaré comme étant une variable structurée de type la structure Personne. Donc, elle comporte entre autres un champ DN qui est lui-même une variable structurée de type la structure Date.

Donc, P.DN.Jour représente le Jour de la variable structurée DN de la variable structurée P

Piles et Files

Pile : Structure de données fonctionnant avec l'algorithme LIFO (Last In First Out)

Exemple : Pile de livres, d'assiettes, …

File : Structure de données fonctionnant avec l'algorithme FIFO (First In First Out)

Exemple : File d'attente de personnes, de processus, …

Caractéristiques d'une Pile

-**Taille** : Nombre maximum d'éléments

-**NC** : Nombre courant d'éléments

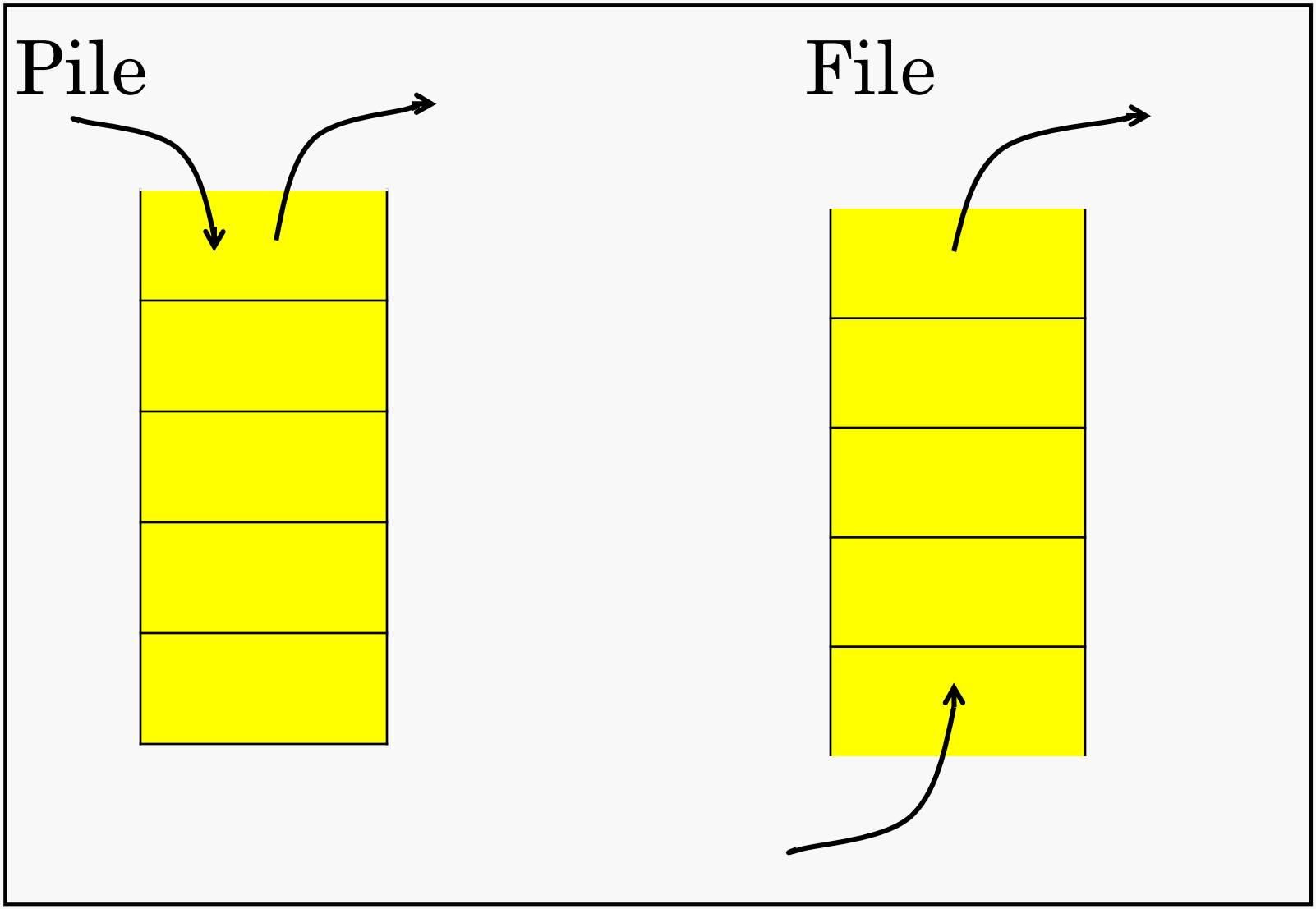
-**Tête** : par où entrer et sortir les éléments

-Une **SD** pour stocker les éléments de la Pile

Caractéristiques d'une File

- **Taille** : Nombre maximum d'éléments
- **NC** : Nombre courant d'éléments
- **Tête** : par où entrer les éléments
- **Queue** : par où sortir les éléments
- Une **SD** pour stocker les éléments de la File

Pile et File



Opérations sur les Piles

- **Empiler** : entrer un élément dans la Pile
- **Dépiler** : sortir un élément de la Pile (Le dernier empilé)
- **Savoir l'état de la Pile** : Pleine ou non, Vide ou non
- **Vider la Pile** : dépiler tous les éléments de la Pile
- **Afficher les éléments de la Pile !!!**

Opérations sur les Files

-**Insérer**: entrer un élément dans la File

-**Retirer**: sortir un élément de la File(Le premier inséré)

-**Savoir l'état de la File** : Pleine ou non, Vide ou non

-**Vider la File** : retirer tous les éléments de la File

-**Afficher les éléments de la File !!!**

Implémentation des Piles et Files

On peut implémenter une Pile (une File) avec :

-Un **Tableau statique** (mauvaise exploitation de la mémoire !!)

-Une **SDD** : tableau dynamique, liste chainée, arbre, ...(Les données ne sont pas permanentes !!)

-Un **Fichier** (La gestion est faite par l'utilisateur !!)

Pile avec Tableau statique

Structure **Pile**

NM : constante entière égale à ***Taille***

NC : variable entière initialisée à 0

Tab : tableau d'***Eléments*** de taille NM

Fin-Structure

//**Elément**: Type des éléments de la Pile

//**NM**: Nombre Maximum d'Eléments

//**NC**: Nombre Courant d'Eléments

Pile avec Tableau statique

```
Fonction PilePleine(P: Pile) : booléen
Si (P.NM=P.NC) alors
        retourner (Vrai)
Sinon
        retourner (Faux)
FinSi
Fin-Fonction
```

Pile avec Tableau statique

```
Fonction PileVide(P: Pile) : booléen
Si (P.NC=0) alors
        retourner (Vrai)
Sinon
        retourner (Faux)
FinSi
Fin-Fonction
```

Pile avec Tableau statique

```
Fonction Empiler(P: Pile, E : Elément)
Si (PilePleine(P)) //P.NC=P.NM
alors
        Afficher("Impossible d'empiler")
Sinon P.Tab[P.NC]←E
        Calculer P.NC←P.NC+1
FinSi
Fin-Fonction
```

Pile avec Tableau statique

```
Fonction Dépiler(P: Pile) : Elément
Si (PileVide(P)) //P.NC=0
alors
        Afficher("Impossible de dépiler")
Sinon Calculer P.NC←P.NC-1
        retourner (P.Tab[P.NC])
FinSi
Fin-Fonction
```

Pile avec Tableau statique

```
Fonction Vider(P: Pile)
Tant que (non PileVide(P)) faire
            //P.NC≠0 ou P.NC>0
      Dépiler(P)
Fin-Tant-que
Fin-Fonction
```

Pile avec Tableau statique

```
Fonction Afficher(P: Pile) !!!
Objet : i : variable entière
Si (PileVide(P)) alors
   Afficher("La pile est vide")
Sinon
  Afficher("Les éléments de la pile :")
  Pour i←0 jusqu'à (P.NC-1) faire
        Afficher(P.Tab[i]," ")
 Fin-Pour
FinSi
Fin-Fonction
```

File avec Tableau statique

Structure **File**

NM : constante entière égale à ***Taille***

NC : variable entière initialisée à 0

Tête : variable entière initialisée à 0

Tab : tableau d'***Eléments*** de taille NM

Fin-Structure

//**Elément**: Type des éléments de la Pile

//**NM**: Nombre Maximum d'Eléments

//**NC**: Nombre Courant d'Eléments

//**Tête** : Indice du 1er Elément

//**Queue** = ???

File avec Tableau statique

```
Fonction FilePleine(F: File) : booléen
Si (P.NM=P.NC) alors
        retourner (Vrai)
Sinon
        retourner (Faux)
FinSi
Fin-Fonction
```

File avec Tableau statique

```
Fonction FileVide(F: File) : booléen
Si (F.NC=0) alors
        retourner (Vrai)
Sinon
        retourner (Faux)
FinSi
Fin-Fonction
```

File avec Tableau statique

```
Fonction Insérer(F: File, E : Elément)
Si (FilePleine(F)) //P.NC=P.NM
 alors Afficher("Impossible d'insérer")
Sinon F.Tab[(F.Tête+F.NC) mod F.NM]←E
        Calculer F.NC←F.NC+1
FinSi
Fin-Fonction
```

File avec Tableau statique

```
Fonction Retirer(F: File) : Elément
Objet : E : variable de type Elément
Si (FileVide(F)) //F.NC=0
alors Afficher("Impossible de retirer")
Sinon E←F.Tab[F.Tête]
        Calculer F.Tête←(F.Tête+1) mod NM
        Calculer F.NC←F.NC-1
        Retourner (E)
FinSi
Fin-Fonction
```

File avec Tableau statique

```
Fonction Vider(F: File)
Tant que (non FileVide(F)) faire
            //P.NC≠0
      Retirer(F)
Fin-Tant-que
Fin-Fonction
```

File avec Tableau statique

```
Fonction Afficher(F: File)
Objet : i :variable entière
Afficher("Les éléments de la file :")
Si (F.Tête+F.NC<F.NM) alors
   Pour i←F.Tête jusqu'à (F.Tête+F.NC-1)
        faire  Afficher(F.Tab[i]," ")
   Fin-Pour
```

File avec Tableau statique

```
Sinon
  Pour i←F.Tête jusqu'à (F.NM-1) faire
        Afficher(F.Tab[i]," ")
  Fin-Pour
  Pour i←0 jusqu'à ((F.Tête+F.NC) mod F.NM)
        faire  Afficher(F.Tab[i]," ")
  Fin-Pour
FinSi
Fin-Fonction
```