SOMMAIRE

I. Introduction

Quand une personne débute dans le domaine de l'informatique, il commence généralement à travailler avec des logiciels qui existent ou qui sont crées préalablement par quelqu'un d'autre, On l'appelle cependant Utilisateur. Mais un jour ces logiciels ne satisferont plus tous les besoins de cet utilisateur. Ce n'est qu'à ce moment là qu'il va penser à créer ses propres applications ou programmes et passer du stade d'utilisateur au stade du Programmeur.

Programmer est facile à dire, mais comment ? . La réponse est simple, il suffit de suivre les étapes suivantes :

Réaliser un cahier de charges ;
Construire un organigramme ou schéma représentant l'enchaîne-ment des instructions d'un programme et l'acheminement des traitements ;
En déduire l'Algorithme ou pseudo-langage (langage compréhen-sible) en suivant des règles bien déterminées ;
Traduire l'Algorithme en un langage de programmation afin qu'il devient compréhensible pour la machine ;
Compiler le programme pour pouvoir déterminer les erreurs de syntaxes et les corriger ;
Exécuter le programme ;

Ceci dit, établir Un cahier de charges est l'étape la plus difficile et délicate. C'est en quelque sorte l'inventaire de ce qu'on a, ce qu'on veut obtenir, par quel moyen, et dans combien de temps. En d'autre manière c'est le cahier dans lequel on va :

Expliquer ce qu'on veut réaliser exactement ;
Déterminer les ***données en sortie***, c'est à dire les données qu'on veut obtenir après traitement ;
En déduire les ***données en entrées***, c'est à dire les données dont la machine a besoin pour réaliser les traitements voulus ;
Déterminer les ***traitements*** nécessaires pour obtenir les données en sortie à partir des données en entrées ;

Le cahier ainsi établit, il reste à l'informatiser, mais comment ? . C'est ce que nous proposons de voir dans le présent rapport.

II. Les structures de base d'un langage de programmation

Dans ce qui suit, nous allons voir comment établir un organigramme puis en déduire le pseudo-langage correspondant et delà le programme. Nous commencerons tout d'abord par définir qu'est ce qu'un programme.

Un programme est une suite d'instructions exécutées par la machine. Une instruction est un ordre qu'on demande à la machine d'exécuter.

Ces instructions peuvent soit s'enchaîner les unes après les autres, on parle alors de ***SEQUENCE D 'INSTRUCTIONS*** ; ou bien s'exécuter dans certains cas et pas dans d'autres, on parle alors de ***STRUCTURE ALTERNATIVE*** ; ou se répéter plusieurs fois, on parle alors de ***STRUCTURE REPETITIVE***.

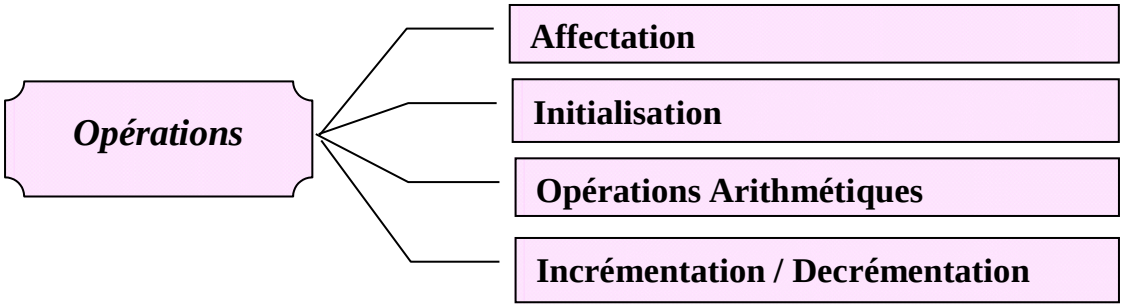
A. La séquence d'instructions

Une instruction est une action que l'ordinateur est capable d'exécuter. Chaque langage de programmation fournit une liste d'instructions qui sont implémentées et que l'on peut donc utiliser sans les réécrire.

Dans notre pseudo-langage, nous n'aurons que la liste minimal d'instructions, nécessaire et suffisante pour les programmes que nous aurons à écrire.

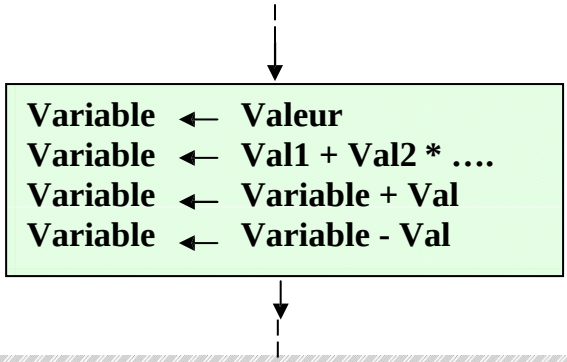
1. Les Opérations sur les variables

Tout au long d'un programme, une variable peut subir plusieurs changements issu de certaines opérations (Affectation, Incrémentation, Decrémentation, Arithmétique) et delà un changement de sa valeur.



Dans un organigramme, elles sont représentées ainsi :

Formalisme d'une Séquence d'instructions d'Opérations



Dans le pseudo-langage, elles s'écrivent ainsi :

Variable ⟵ Valeur
Variable ⟵ Val1 + Val2 * ….
Variable ⟵ Variable + Val
Variable ⟵ Variable - Val

Par exemple **:**

A ⟵ 5
S ⟵ 3 + (2 * 3) / 6
B ⟵ A + (20 * S)
A ⟵ A - 1

a) L'affectation

Une affectation c'est le fait de mémoriser une valeur à un endroit dans la mémoire nommé variable ou dont l'adresse est le nom de la variable.

Variable ⟵ Valeur

Ce qui se lit « Variable reçoit valeur » .

Par exemple **:**

I ⟵ 5 ; Termine ⟵ Vrai

b) L'Incrémentation et la Decrémentation

Incrémenter une variable c'est le fait de lui ajouter une valeur à sa valeur initiale. Decrémenter une variable c'est le fait de soustraire une valeur de sa valeur initiale. Cette notion est souvent utilisée dans le cas ou on a besoin de créer un compteur.

Variable ⟵ Variable + Val

Par exemple **:**

I ⟵ I + 1
J ⟵ j – 2

c) L'Initialisation

C'est le fait de donner une valeur initiale à une variable avant de l'incrémenter ou de la décrémenter. Cette notion est souvent employée lorsqu'on utilise un compteur.

Var ← Val_I

Par exemple :

I ← 0
I ← I + 1

d) Les opérations arithmétiques

Les opérations arithmétiques courantes sont l'addition, soustraction, multiplication et division. Ces opérations s'écrivent de la manière suivante :

Variable ← Val1 + Val2
Variable ← Val1 - Val2
Variable ← Val1 * Val2
Variable ← Val1 / Val2

On doit toujours affecter le résultat d'une opération dans une variable

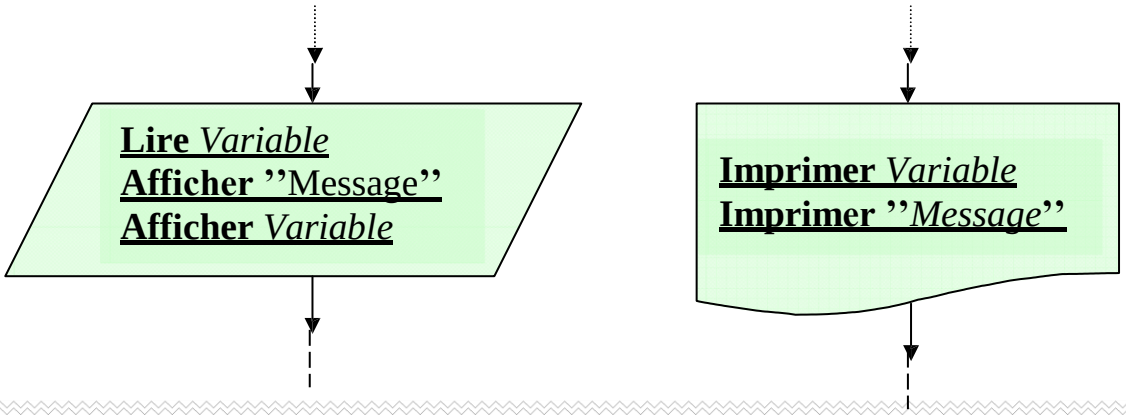
Par exemple :

A ← 5 + 2 ; B ← A – 1 ; C ← (B / 2) – 1 + A

2. Le dialogue avec l'utilisateur

Pour permettre au programme de dialoguer avec l'utilisateur, c'est à dire d'afficher un résultat à l'écran et de lire une entrée au clavier, il faut au moins deux instructions une pour lire et l'autre pour afficher à l'écran ou pour imprimer. Dans un organigramme, elles sont représentées ainsi :

Formalisme d'une Séquence d'instructions de dialogue avec l'utilisateur



Dans le pseudo-langage, elles s'écrivent ainsi :

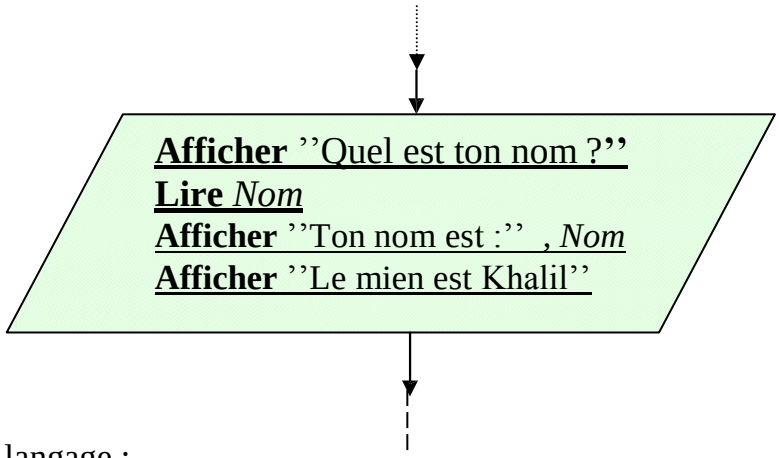
Lire Variable
Lire Variable1, Variable2,….

Afficher ''Texte''
Afficher Variable
Afficher ''Texte'' , Variable

Imprimer ''Texte''
Imprimer Variable
Imprimer ''Texte'' , Variable

La première lit tous les caractères qui sont saisis au clavier, jusqu'à ce que l'utilisateur appuie sur la touche entrée, et stocke le résultat dans la variable. La seconde affiche sur l'écran le ou les textes et la valeur des variables. La troisième imprime le ou les textes et la valeur des variables.

Par exemple : Cette séquence d'instructions va permettre de dialoguer avec l'utilisateur. En organigramme elle sera représentée comme suit :



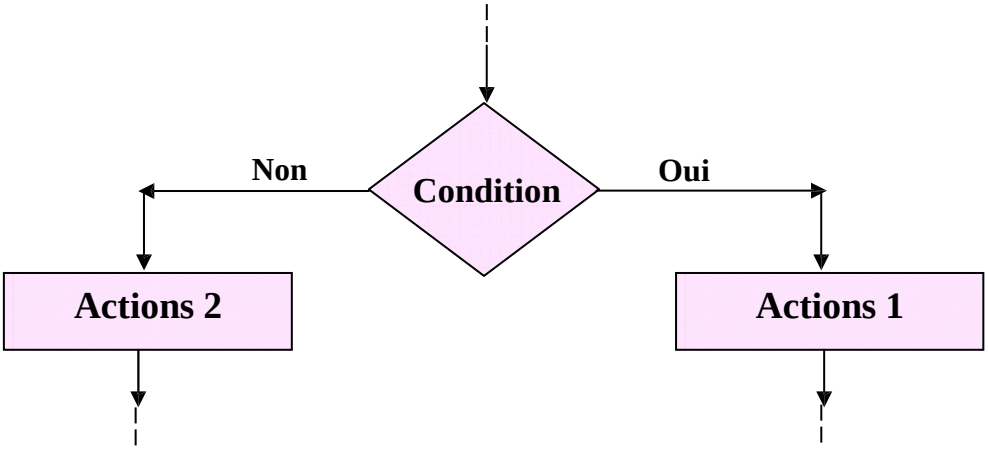
En pseudo-langage :

Afficher ''Quel est ton nom ? : ''
Lire Nom
Afficher ''Ton nom est :'' , Nom
Afficher ''Le mien est Khalil''

B. La structure alternative

Il est souvent nécessaire lorsque l'on écrit un programme de distinguer entre plusieurs cas conditionnant l'exécution de telles ou telles instructions. Pour ce faire, on utilise une structure alternative (Conditionnelle) : Si on est dans tel cas on fait cela sinon on fait ceci. Le formalisme de cette structure dans un organigramme sera comme suit :

Formalisme d'une Structure alternative



La syntaxe de cette structure en pseudo-langage est la suivante :

SI Condition **ALORS** Actions1 [**SINON** Actions2] **FINSI**
NB : Les crochets signifient que la partie sinon est facultative.

Les actions

Les actions qui suivent le *Sinon* ou le *Alors* peuvent être :

Une simple instruction
Une suite d'instructions
Une autre structure alternative
Une autre structure répétitive

Les conditions

Pour exprimer les conditions on utilise les opérateurs conditionnels suivants :
= égal ; < Inférieur ; > Supérieur ; <= Inférieur ou égal ; >= Supérieur ou égal ;
<> différents ;

Par exemple :

(A<1) (A<=B) ……..

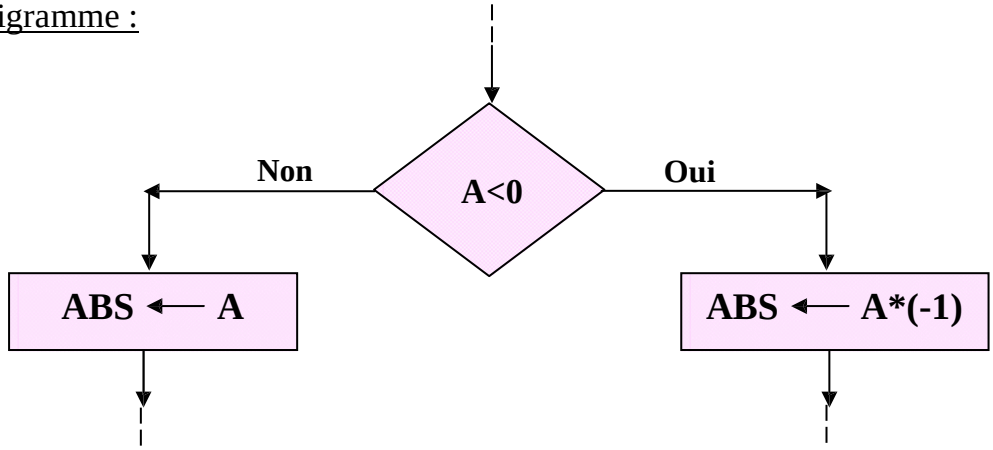
On peut combiner des conditions à l'aide des opérateurs logiques suivant : **Ou** ; **Et** ; **XOR**;

Par exemple :

(A<>B) **ET** (A>=5)
((A<0) **ET** ((B=0) **OU** (C<>A)) **XOR** (D=1))

Lorsque l'on écrit de telles conditions, il est recommandé de mettre toutes les parenthèses afin d'éviter les erreurs.

Par exemple : Programme permettant de calculer la valeur absolue d'un nombre :
Organigramme :



Ce qui donnera en pseudo-langage :

Si (A<0) **Alors** ABS ← A* (-1) **Sinon** ABS ← A **FINSI**

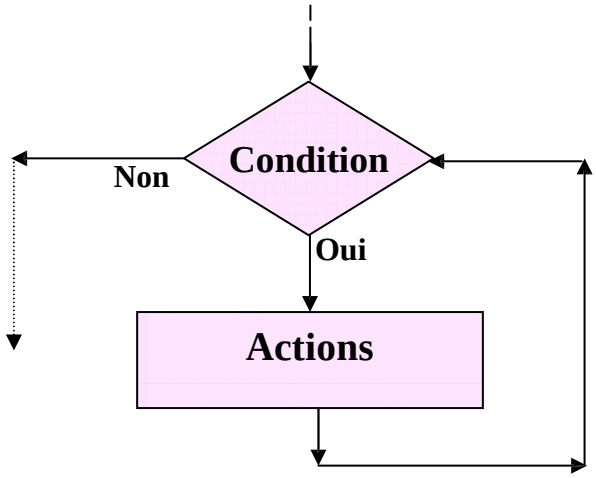
C. La Structure répétitive

Un programme a presque toujours pour rôle de répéter la même action un certain nombre de fois. Pour ce faire, on utilise une structure permettant de dire «Exécuter telles actions jusqu'à ce que telle condition soit remplie » .
Bien qu'une seule soit nécessaire, la plupart des langages de programmation proposent trois types de structures répétitives. Voici celles de notre pseudo-langage.

1. La boucle TantQue

Le formalisme de cette structure dans un organigramme sera comme suit :

Formalisme d'une Boucle TantQue



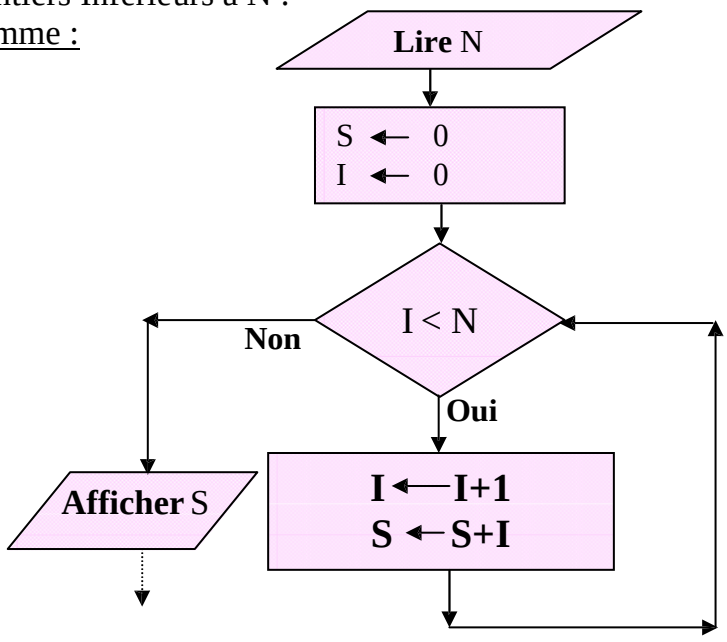
La syntaxe de cette structure en pseudo-langage est la suivante :

TantQue Condition **Faire**
    Actions
Fin TantQue

Ce qui signifie tant que la condition est vraie, on exécute les actions.

Par exemple : Ceci est un programme qui va permettre de calculer la somme des nombres entiers Inférieurs à N :

Organigramme :



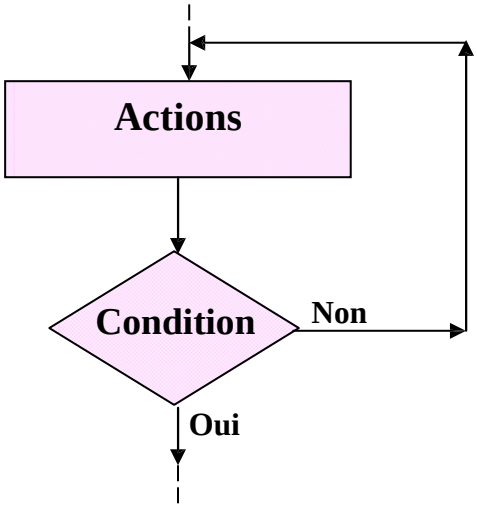
Algorithme (Pseudo-langage) :

```
Lire n
I ⟵ 0
S ⟵ 0
TantQue ( I< n ) Faire
        I ⟵ I+1
        S ⟵ S+ I
FinTQ
Afficher S
```

2. La Boucle Répéter Jusqu'à

Cette boucle va pouvoir exécuter les actions au moins une fois la vérification de la condition se fait à la fin de la boucle contrairement à la boucle tant que.

Formalisme d'une Boucle Répéter jusqu'à



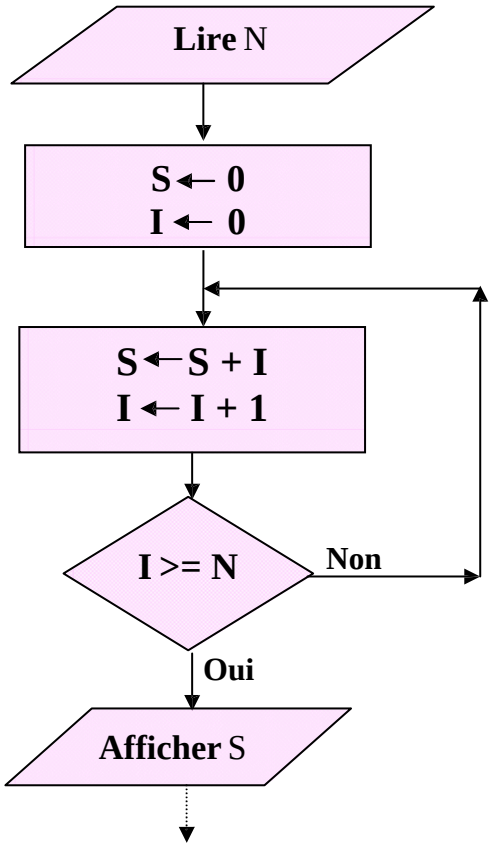
La syntaxe de cette structure en pseudo-langage est la suivante

```
Répéter
    Actions
Jusqu'à Condition
```

Ce qui signifie que l'on exécute les actions jusqu'à ce que la condition soit vraie.

Par exemple : Le même exemple avec la boucle répéter jusqu'à :

Organigramme :



Algorithme ou pseudo-langage :

```
Lire n
I ⟵ 0
S ⟵ 0
Répéter
        S ← S + I
        I ← I+1
Jusqu'à ( I >= n )
Afficher S
```

3. La Boucle Pour

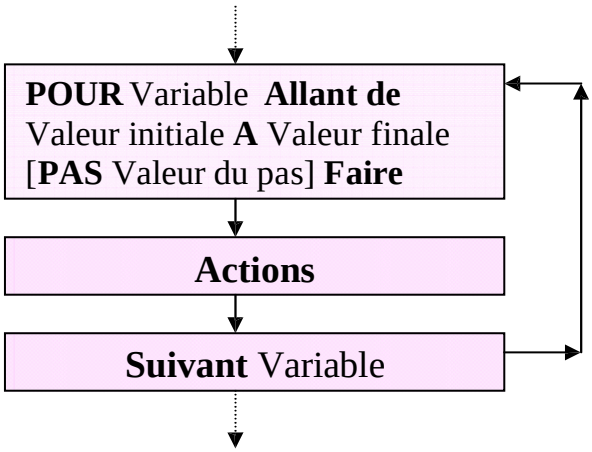
Très souvent, on utilise une structure répétitive avec un compteur et on s'arrête lorsque le compteur a atteint sa valeur finale.

Par exemple :

```
Lire n
I ← 0
S ← 0
Répéter
        S ← S + I
        I ← I+1
Jusqu'à ( I >= n )
Afficher S
```

C'est pourquoi la plupart des langages de programmation offrent une structure permettant d'écrire cette répétitive plus simplement. Dans le pseudo-langage c'est la structure POUR. L'organigramme de cette structure peut être représenter comme suit :

Formalisme d'une Boucle Pour



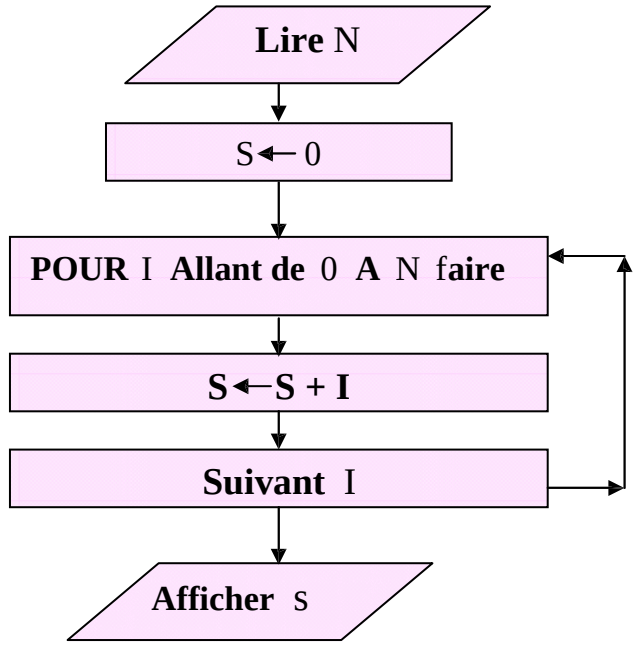
La syntaxe de cette structure en pseudo-langage est la suivante

```
POUR Variable  Allant de Valeur initiale A Valeur finale [PAS Valeur du pas] Faire
    Actions
Suivant Variable
```

Lorsque le pas est Omis, il est supposé égale à (+1).

Par exemple : Ceci est un programme qui va permettre de calculer la somme des nombres entiers Inférieurs à N :

Organigramme :

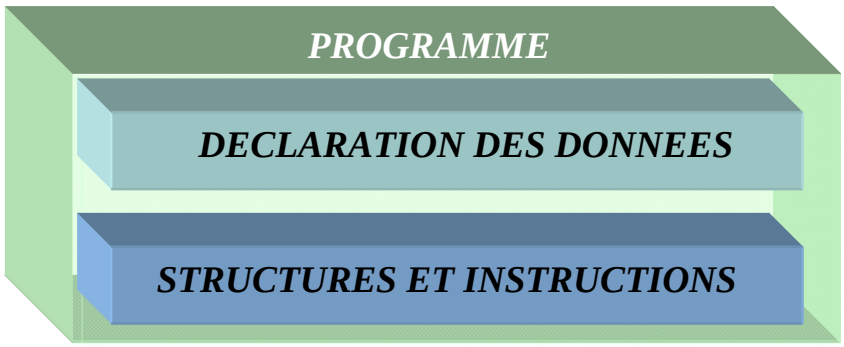


Algorithme ou pseudo-langage :

```
Lire n
S ← 0
Pour I Allant de 0 A N Faire
        S ← S + I
Suivant I
Afficher S
```

D. La déclaration des variables

Un programme est composé de deux parties :



La partie instruction contient les instructions à exécuter.
La partie données contient toutes les variables utilisées par le programme.

Un programme exécutable est chargé dans la mémoire centrale de l'ordinateur, les valeurs que l'on a affectées aux variables doivent être conservées tout le temps du déroulement du programme. Par conséquent il faut que le programme soit capable de réserver la place nécessaire aux variables.
Pour ce faire, les variables doivent être déclarées afin que le compilateur sache quelle place elles vont occuper.

1. Les types de Variables

Les variables que l'on utilise dans le programme ne sont pas toutes de même nature, il y a des nombres, des caractères, … on dit que les variables sont typées.

Il est nécessaire de donner un type aux variables, car cela permet d'une part de contrôler leur utilisation (on ne peut pas deviser un caractère par un nombre) et d'autre part cela indique quelle place il faut réserver pour la variable.
Généralement les variables peuvent être des types suivants :

Entier

Il s'agit des variables destinées à contenir un nombre entier positif ou négatif. Dans notre Pseudo-langage, on écrira la déclaration des variables de type entier comme suit :

Variable1, Variable2… : **Entier**

Généralement un entier occupe deux octets, ce qui limite les valeurs de –32768 à +32768. Cependant cela dépend des machines, des compilateurs et des langages. Certains langages distinguent les entiers courts (1 octets), les entiers longs (4 octets) et les entiers simples (2 octets).

Variable1, Variable2… : **Entier Simple**
Variable1, Variable2… : **Entier Long**

REEL

Il s'agit des variables numériques qui ne sont pas des entiers, c'est à dire qui comporte des décimales. Généralement un nombre réel est codé sur 4 octets. Il existe deux types de réel : simple et double. Dans notre pseudo-langage, la déclaration des variables de ce type sera comme suit :

Variable1, Variable2,… : **REEL**
Variable1, Variable2,… : **Réel Double**

CARACTERE

Les variables de type caractère contiennent des caractères alphabétiques ou numériques (de 0 à 9), mais dans ce cas ils ne sont pas considérés comme étant des nombres et on ne peut pas faire d'opérations dessus. Un caractère occupe 1 octet.
Dans notre pseudo-langage, une variable de type caractère de déclare ainsi :

Variable1, Variable2,… : **CAR**

Chaîne de caractères

Ce type de variable peut contenir plusieurs caractères. Dans le pseudo-langage, on le déclarera ainsi :

Variable1, Variable2,… : **Chaîne**

BOOLEEN

Il est souvent nécessaire lorsque l'on écrit un programme d'introduire des variables qui prennent les valeurs Vrai ou Faux ou les valeurs Oui ou Non.
Pour cela Il Existe un type particulier dont les variables ne peuvent prendre que 2 valeurs : **Vrai** ou **Faux.**
Dans notre pseudo-langage Ce type de variable se déclare de la manière suivante.

Variable1, Variable2,… : **BOOLEEN**

Les TABLEAUX

On peut regrouper plusieurs variables sous un même nom, chacune étant alors repérée par un numéro. C'est ce que l'on appelle un tableau. On peut faire un tableau avec des variables de n'importe quel type.
Dans tous les cas le $i^{ème}$ élément d'un tableau appelé TAB sera adressé par TAB(i).
Généralement on fait des tableaux à une dimension, mais il existe également des tableaux à deux dimensions, dans ce cas TAB(i,j) représente la $j^{ème}$ colonne et la $i^{ème}$ ligne. Dans notre pseudo-langage un tableau est déclarer de la manière suivante :

Variable : **TABLEAU [Longeur] Type**
Variable : **TABLEAU [nb_ligne , nb_col] Type**

Par exemple :

mot : **TABLEAU [10] CAR**
Liste_nbr : **TABLEAU [25]**
Mots : **TABLEAU [10,20] CAR**

E. Les fonctions et Procédures

Lorsque l'algorithme devient trop compliqué, on aura envie de le découper, de manière à ce que chaque partie soit plus simple et donc plus lisible.
De même lorsqu'une partie de code doit être exécutée plusieurs fois, à des endroits différents, ou réutilisée ultérieurement on pourra ne l'écrire qu'une fois et lui donner un nom en faisant une Fonction ou une Procédure.

1. Procedure

Une procédure est une suite d'instructions servant à réaliser une tâche précise en fonction d'un certain nombre de paramètres. Les paramètres sont de deux types.

Les paramètres de type Val : il contienne une valeur qui sera utilisée dans la procédure.

Les paramètres de type Var : ils représentent une variable du programme appelant qui pourra être lue et modifiée si nécessaire.

Dans notre pseudo-langage une procédure se déclare de la manière suivante :

```
Procedure Nom_Procedure [ (Val  Variable : Type , ……, Var Variable : Type ) ]
Déclaration des Variables Locales
    Début Procedure
        Actions
    Fin Procedure
```

Les variables que l'on déclare localement dans la procédure ne sont connues que dans cette procédure.
Pour utiliser ou appeler cette procédure à partir d'un programme, on écrit :

```
Nom_Procedure [ ( Variable1, Valeur1, ……) ]
```

A chaque paramètre de type VAR on associe un nom de variable connue dans le programme appelant, à chaque paramètre de type Val, on associe une variable ou une valeur.

Par exemple :

```
Procedure Remplir_Chaine ( Val Longueur : Entier , Val Caractere : CAR , Var
Chaine : TABLEAU [MAXCAR] CAR )

I, long  : Entier

Début Procédure
    Si Longueur > MAXCAR Alors
       Long =MAXCAR
```

```
        Sinon
           Long = Longueur
        FinSI

        Pour I allant de 1 à long Faire
           Chaîne(i) =Caractère
        FinPour

Fin Procédure
```

On l'utilise ainsi :

```
Remplir_Chaine (12, ''*'', Ligne)
        Ou
Remplir_Chaine(longmot, rep, message)
```

2. Fonction

Une fonction est une procédure dont le but est de déterminer une valeur et de la retourner au programme appelant.
Dans notre pseudo-langage, elle se déclare de la manière suivante :

```
Fonction Nom_Fonction (Val Variable : Type , ……, Var Variable : Type ) : Type
Déclaration des variables locales
Début Fonction
        Actions
        Nom_Fonction = Valeur
Fin Fonction
```

Les même remarques pour la procédure s'applique pour une fonction. De plus il faut noter que la fonction retourne une valeur (ou le contenu d'une variable), donc on doit indiquer son type.

L'appel d'une fonction s'écrit :

```
Variable = Nom_Fonction (Variable, ……, Valeur )
```

Une fonction ne peut donc pas retourner un tableau.

Par exemple :

```
Fonction Valeur_Absolue ( Val nombre : Entier ) : Entier
Début Fonction
        Si (nombre<0) Alors
           Valeur_Absolue = (nombre*(-1))
```

```
        Sinon
           Valeur_Absolue = nombre
        FinSi
Fin Fonction
```

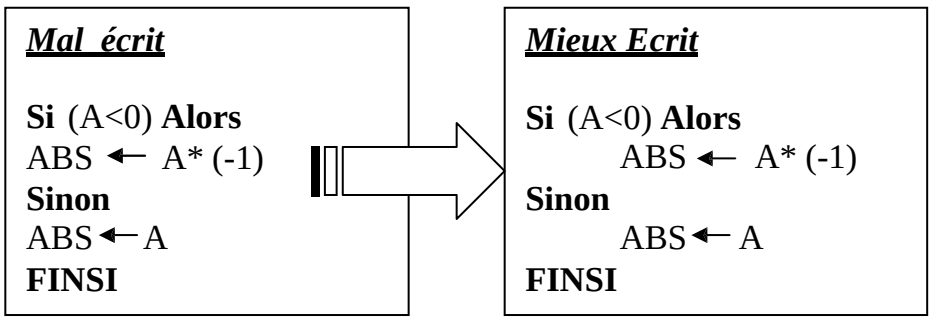
On l'utilise ainsi :

```
I = Valeur_Absolue(I)
L= Valeur_Absolue(-120)
```

III. Les Règles de programmation

Un programme doit être le plus lisible possible, de manière à ce qu'en le lisant n'importe qui d'autre que l'auteur soit capable de comprendre ce qu'il fait rien qu'en le lisant. Pour cela il faut suivre les quelques règles suivantes :

Le nom des variables doit être significatif, c'est à dire indiquer clairement à quoi elles servent.
Un algorithme ne doit pas être trop long (une page d'écran). S'il est trop long il faut le découper en fonctions et procédures.
Les structures de contrôles doivent être indenter, c'est à dire, par exemple, que les instructions qui suivent le Alors doivent toutes être alignées et décalées d'une tabulation par rapport au SI. Il en est de même pour les répétitives.
A chaque imbrication d'une structure de contrôle, on décale d'une tabulation.

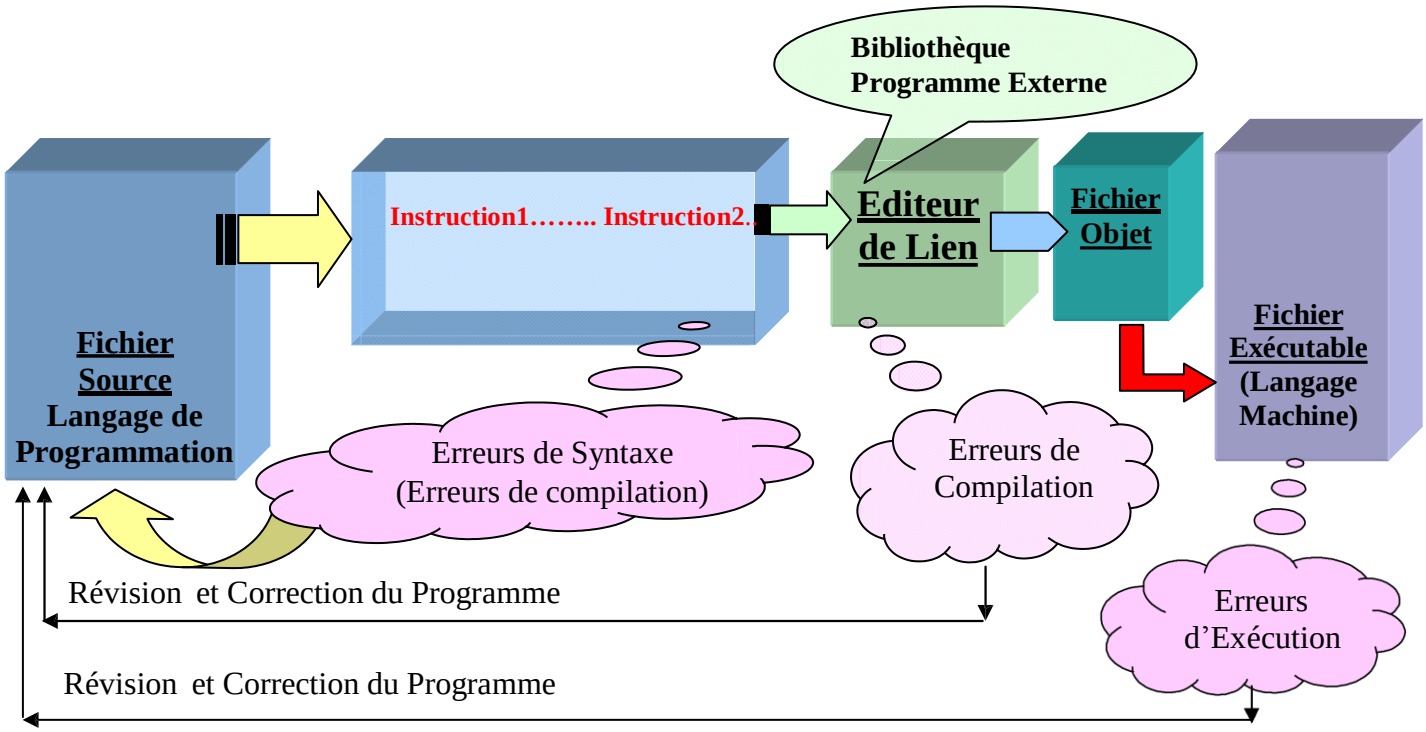


En ce qui concerne *les fonctions et procédures*, il y' a aussi des règles à respecter :

On doit toujours être capable de donner un nom significatif à une procédure ou à une fonction.
Le nombre de paramètres ne doit pas être trop grand (en général inférieur à 5) car cela nuit à la lisibilité du programme.
Une procédure ou une fonction doit être la plus générale possible de manière à pouvoir être réutilisée dans d'autres circonstances.
Si le but d'une procédure et de calculer une valeur simple, il est préférable d'en faire une fonction.
Il est souvent plus claire d'écrire une fonction booléenne plutôt qu'une condition complexe.

IV. La compilation

Un langage de programmation sert à écrire des programmes de manière à les exécuter. Des outils, appelés Compilateurs, permettant de traduire le langage écrit par le programmeur en langage machine. Il fonctionne de la manière suivante :



Le compilateur analyse le langage source afin de vérifier la syntaxe et de générer un fichier objet en langage intermédiaire assez proche du langage machine. Tant qu'il y a des erreurs de syntaxe, le compilateur est incapable de générer le fichier objet.

Souvent on utilise dans un programme des fonctions qui soit, on été écrite par quelqu'un d'autre soit sont fournies dans une bibliothèque (Sous forme de Fichier). Dans ce cas, le compilateur ne les connaît pas et ne peut donc pas générer le langage intermédiaire correspondant.

C'est le travail de l'éditeur de liens que d'aller résoudre les références non résolues. C'est à dire que lorsqu'il y a appel dans le fichier objet à des fonctions, procédures ou variables externes, l'éditeur de liens recherche les objets ou bibliothèque concernés et génère l'exécutable. Lorsque l'éditeur de liens ne trouve pas ces références il se produit une erreur.

V. Résumé

Structure Générale d'un Algorithme

Programme Nom_Programme

Déclarations

Déclarations des Variables Globales :
Nom_Var : Type
Nom_Tab : Tableau [i ,J] Type

(Type = Entier, Réel, Car, Chaîne ou Booléen)

Déclaration Des Fonctions :
Fonction Nom_Fonction (Val nom : Type , Var nom : Type) : Type
Déclaration de variables Locales
Début Fonction
Actions
Fin Fonction

Déclarations des Procédures :
Procédure Nom_Fonction (Val nom : Type , Var nom : Type)
Déclaration de variables Locales
Début Procédure
Actions
Fin Procédure

Début_Programme

Actions

Suite d'Instructions :
Opérations (Arithmétique , Incémentation,…..)
Instructions d'Entrée / Sortie (**Lire** , **Afficher** ..)
Appel de procédure ou de Fonction
- **Nom_Procedure**
- Var ← **Nom_Fonction** (Vars ou Vals,…)

Structure :
Structure Alternative
Si Condition **Alors** Action1 **Sinon** Action2 **FinSi**
Structure répétitive
Répéter Actions **Jusqu'à** Condition
TantQue Condition **Faire** Actions **FinTQ**
Pour Var **Allant de** Val_i **à** Val_f [Pas de Val]
Faire Actions **Suivant** Var
Structure Complexe (Combine les deux)

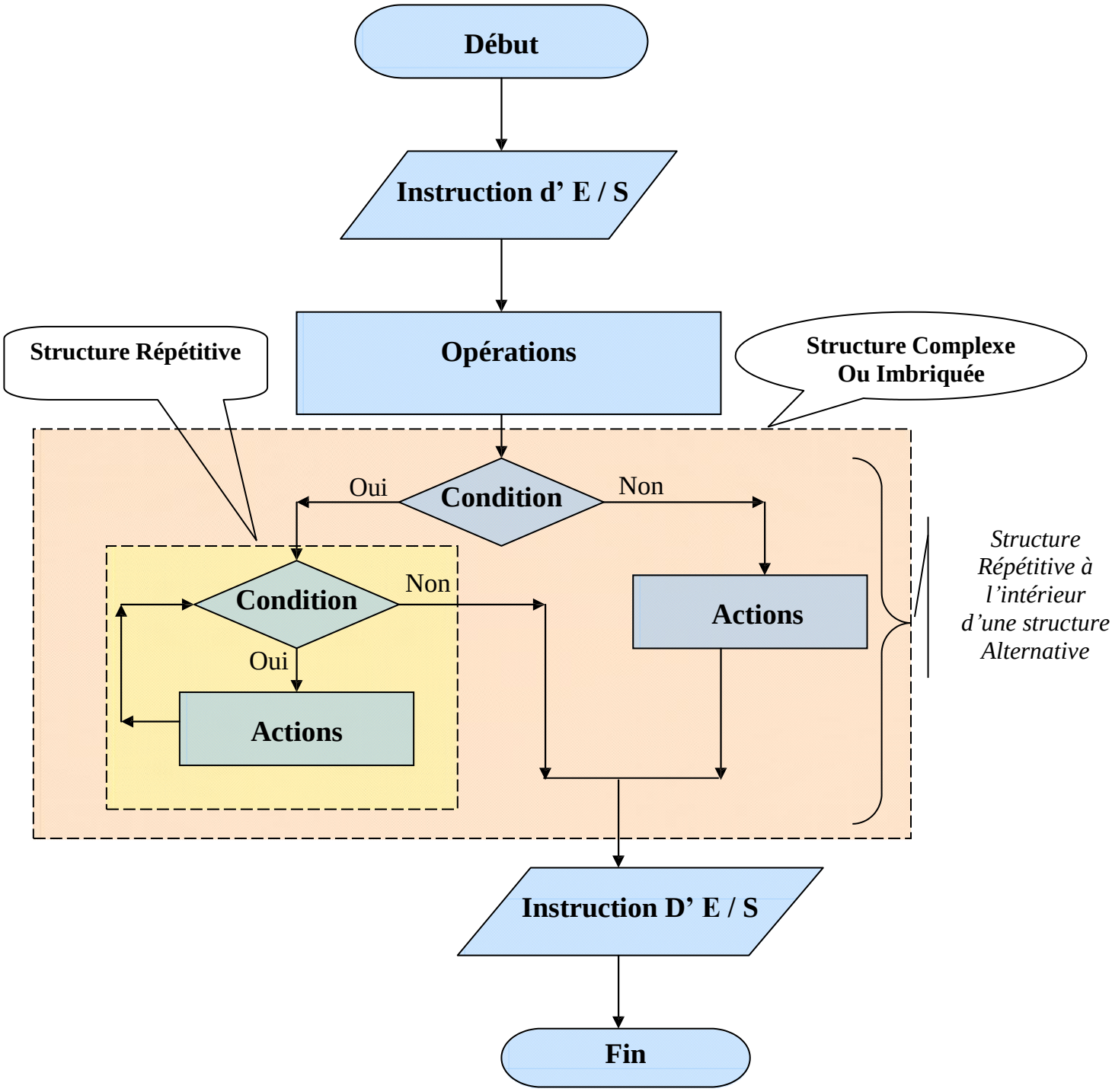
Fin_Programme

Condition :
Condition Operateur_Logique Condition
Variable Opérateur_Conditionnel Valeur
Variable Opérateur_Conditionnel Variable

Opérateurs Logiques : Et , Ou , Non , Xor

Opérateurs Conditionnels : =, < , > , >= , >= , <>

Structure générale d'un organigramme



Site Web :

1. **http://www.labo-info.co.cc**
2. **http://www.algofree.co.cc**