Notions d'algorithme et de programme

1. Concept d'un algorithme

Le mot « Algorithme » est inventé par le mathématicien « ALKHAWARISMI».

Définition1 : Un Algorithme est une séquence d'actions primitives réalisant un traitement. Il décrit le plan ou les séquences d'actions pour la résolution d'un problème donné.

Définition 2 : Un algorithme est une suite d'actions (ou d'*instructions*) séquentielles et ordonnées, permettant de résoudre un problème donnée

Une fois l'algorithme est écrit (avec n'importe quelle langues), il sera transformé, après avoir choisi un langage de programmation, en un programme (code source) qui sera ensuite compilé (traduit) et exécuté par l'ordinateur.

Donc un programme est une suite d'instructions écrit dans un langage de programmation et exécuté par un ordinateur permettant de résoudre un problème donné.

2. Structure d'un algorithme

Un algorithme est composé de trois parties principales :

1. **La partie *Entête*** dans cette partie on déclare le nom de l'algorithme à travers un identificateur 2.
2. **La partie déclarations** permet de déclarer les données qui sont utilisées par le programme (les variables et les constantes)
3. **La partie instructions** constitue le programme principal.

Exemple :

Algourithme pour calculer le périmètre d'un cercle

Algorithme périmètre_cercle

Partie entête

Constante

PI = 3.1415

Partie déclaration

Variable

Rayon: **Réel**

périmètre: **Réel**

Début

Écrire(" saisir le ayon)

Lire(rayon)

périmètre <- 2 * PI * rayon

Partie instruction

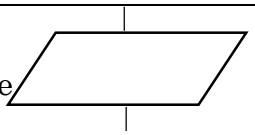
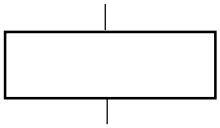
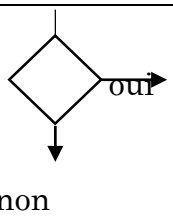
Écrire("Le périmètre est : ", périmètre)

Fin

3. La Représentation En Organigramme

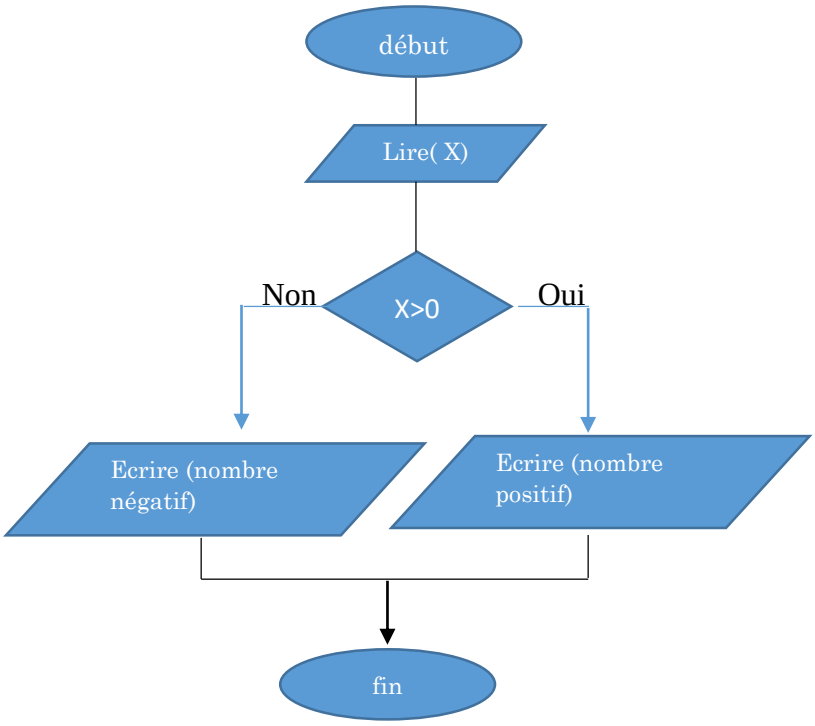
Un organigramme est une représentation graphique d'un algorithme, il permet de schématiser graphiquement la solution d'un problème. La représentation graphique permet une lecture aisée des algorithmes mais présente toutefois l'inconvénient de consommer une place importante. Elle utilise les symboles de la norme NF Z 67-010 dont les principaux sont les suivants :

symbole	désignation
L'ovale	Exprime le début ou la fin de l'organigramme

Le parallélogramme 	utilisé pour la lecture des données et l'affichage (entrée/sortie)
Le rectangle 	utilisé pour les opérations d'affectation
Le losange 	utilisé pour Instruction conditionnelle.

Exemple

Soit l'organigramme d'un algorithme qui lit la valeur d'une variable X puis affiche si elle est positive ou négative.



Algorithme calcul
Variable X:entier
Début
Lire X
Si X>0 alors écrire (nombre positif
Sinon (Ecrire nombre négatif)
fin

4. Démarche pour résoudre un problème

Un algorithme s'écrit le plus souvent en pseudo langage de programmation afin de faciliter ultérieurement sa traduction dans un langage de programmation. Un algorithme n'est donc exécutable directement par aucune machine. Mais il a l'avantage d'être traduit facilement dans tous les langages de programmation.
La rédaction d'un algorithme est un exercice de réflexion qui se fait sur papier et qui précède l'étape de la programmation.
L'élaboration d'un programme passe généralement par les étapes suivantes :

1. Analyse et comprendre du problème posé.
2. Décomposer si c'est possible le problème en sous-problèmes plus simple à résoudre.
3. Elaboration de l'algorithme
4. la traduction de l'algorithme en programme, en choisissant un langage de programmation (C, JAVA, PYTHON…).

En fait L'algorithme est la résolution brute d'un problème informatique, il est indépendant du langage de programmation, Par exemple, on utilisera le même algorithme pour une implantation en Java, ou bien en C ou c++…etc.

5. LE LANGAGE C

Le langage 'C' est un langage extrêmement performant et il est parmi les langages de programmation les plus répondu et les plus utilisé en informatique.
Ce langage a été conçu en 1972 par Dennis Richie et Ken Thompson. Il est devenu de plus en plus populaire dans les années 80. En 1983, l'ANSI (American National Standards Institute) décida de normaliser le langage, ce travail s'acheva en 1989 par la définition de la norme ANSI C.
Le langage C a été conçu pour l'écriture de systèmes d'exploitation et du logiciel de base (90% du noyau du système UNIX est écrit en C, le compilateur C lui-même est écrit en grande partie en langage C, de nombreux logiciels tels que Microsoft Word ou Excel sont écrits en C ou en son successeur orienté objet C++.

5.1. La Compilation

Pour être capable de programmer sur ordinateur avec un langage de programmation compilé, il faut d'abord installer le compilateur de ce langage sur cet ordinateur.

Le C est un langage compilé (par opposition aux langages interprétés). Cela signifie que le programme (le code source) n'est pas directement exécutable par le microprocesseur, il faut d'abord le traduire en langage machine à l'aide d'un programme appelé compilateur.

Le compilateur C transforme un programme C (fichier suffixé par .c) en un programme objet (fichier suffixé par .o)

Chaque compilateur possède sa propre commande pour créer le code objet.

Sous Windows par exemple on peut citer : C Microsoft, **Turbo C**, Turbo C++, Borland C, Borland C++.

5.2. Pour quoi utiliser le langage C ?

- Souple et puissant : Le langage C est utilisé pour des projets variés comme les systèmes d'exploitation (unix), les traitements de textes (Microsoft Word ou Excel), les graphiques, les tableurs, les compilateurs pour d'autres langages.
- Il existe un large choix de compilateurs et d'utilitaires. Avec la norme ANSI, le C est devenu un langage portable : un programme C écrit pour un type d'ordinateur(IBM) peut être compilé pour tourner sur un autre système avec très peu ou aucune modification.
- Le langage C est modulaire et structuré.

6. Structure d'un programme en C

La structure générale d'un programme écrit en C est la suivante :

```
 # include<stdio.h>
main()
{
   < Déclaration des variables locales>
    < Partie Instructions ;>
}
```

 # include <stdio.h> : Directive à destination du préprocesseur. Son rôle est de demander de charger des fonctions situées dans le fichier <stdio.h> (fichiers en-tête).

main(): se nomme un « en-tête ». Elle précise le début du programme principal.
Le programme (principal) proprement dit vient à la suite de cet en-tête. Il est délimité par les accolades « { » et « } ». On dit que les instructions situées entre ces accolades forment un « bloc ». Ainsi on peut dire que la fonction main est constituée d'un en-tête et d'un bloc.
<Déclaration des variables locales> : tout programme en C manipule des données sous forme des variables et des constantes. Ces variables doivent être déclarés avant leur utilisation.
< Partie instruction> : c'est le corps du programme, là où on doit préciser les instructions à suivre pour résoudre le problème posé.

7. Les éléments du base du C

Un programme en langage C est constitué des éléments suivants :
– les identificateurs,
– les mots-clefs,
– les constantes,
– les variables
– les opérateurs,
-Les commentaires.

7.1. les identificateurs

Un identificateur est une suite de caractères composée de lettres (’a’- ’z’ et ’A’ `-’Z’), de chiffre (0 à 9) ou du signe (-). Son rôle est de donner un nom à une entité du programme (variable, constante ou une fonction).

Un identificateur doit vérifier les conditions suivantes :

- Il ne doit pas commencer par un chiffre : 1x n’est pas valide,
- les lettres accentuées sont interdites : xé n’est pas valide
- les identificateurs sont case-sensitive : x1 et X1 sont deux identificateurs différents,

Exemple :

var1, S, i, sont des identificateurs valides, tandis que i:j ou 1i ne le sont pas.

7.2. Les mots-clefs

Les *mots-clefs*, sont des mots réservés pour le langage lui-même et ne peuvent pas être utilisés comme identificateurs. L’ANSI C compte 32 mots clefs :

Auto- const- double- float- int- short- struct- unsigned-break- continue -else for- long- signed- switch- void-case- default- enum- goto- register- sizeof- typedef- volatile-char- do- extern- if- return- static –union- while.

7.3. Les Constantes

Une constante est un objet contenant une valeur qui ne peut jamais être modifiée. Le type de la constante étant déterminé par la façon dont la constante est écrite. Les constantes peuvent être des entiers, des réels ou des caractères.

Sa déclaration en C :

 # define nom-de-constante sa-valeur

***Exemple* :**

define X 100

7.4. Les Variables

Une variable est un objet contenant une valeur pouvant être modifiée. Elle peut être considérée comme une boîte dans laquelle on met des données. Il est possible de lire le contenu de cette boîte ou d'écrire des données dans celle-ci.
Sa déclaration est de la forme suivante :

Type nom-variable ;

Exemple

int x;

float y=23,5 ;

En C une variable doit être déclarée avant son utilisation.
Une variable ne peut contenir qu'une seule valeur à la fois. Affecter une valeur à une variable ayant déjà une valeur revient à la modifier.

▶ Les types prédéfinis

Toute variable, constante ou fonction en C possède un type précis. Le type d'un objet définit la façon dont il est représenté en mémoire.
Les types de base en C sont les suivant :

- **Char** : désigne un caractère tel que char x= 'a'
- **Int, short, long** : désigne un entier. Le bit de poids fort d'un entier est son signe.
- **Unsigned** : signifie que l'entier n'a pas de signe (un nombre naturel)
- **Float, double, long double** : servent à représenter des nombres en virgule flottante (des réels). Ils correspondent aux différentes précisions possibles.

Remarque :
Les valeurs maximales et minimales des différents types entiers sont définies dans la librairie standard **limits.h.**
la fonction **sizeof (type)** permet de renvoyer le nombre d'octets nécessaires pour stocker le type cité.

7.5. Les commentaires

Les commentaires dans un programme sont des messages qui explique des parties du votre code source. Ils peuvent être placés à n'importe quel endroit du programme. Un commentaire débute par /* et se termine par */.

Par exemple,

/* Ceci est un commentaire */

8. Les opérateurs

8.1. L'affectation

En C, l'affectation est un opérateur à part entière. Elle est symbolisée par le signe =. Sa syntaxe est la suivante :

var = expression

Cela signifie que la valeur de « **expression** » est affecté à la variable var.

Le terme de gauche de l'affectation doit être une variable mais pas une constante.

Exemple:

X=3 signifie que la valeur de x est 3

X=x-2 signifie que la valeur de x devient (3-2)

donc x=1 (l'ancienne valeur est écrasée).

8.2. Les opérateurs d'affectation composée

Les opérateurs d'affectation composée sont :

+= -= *= /= %=

Pour tout opérateur ***op***, l'expression

$$exp1\ \mathbf{op=}\ exp2 \Leftrightarrow exp1 = exp1\ \mathbf{op}\ exp\text{-}2$$

autrement dit:

x+=y ⇔ x=x+y

x*=y ⇔ x=x*y

x/=y ⇔ x=x/y

x%=y ⇔ x=x%y

8.3. Les opérateurs relationnels

Les opérateurs relationnels sont :

> strictement supérieur

>= supérieur ou égal

< strictement inférieur

<= inférieur ou égal

== : (égal) compare deux valeurs et vérifie leur égalité

!= différent

Leur syntaxe est la suivante:

expression-1 opérateur expression-2

Les deux expressions sont évaluées puis comparées. Ainsi La valeur rendue égale à 1 si la condition est vraie, et 0 sinon.

8.4. Les opérateurs logiques

Les opérateurs logiques sont :

&& (et logique): Vérifie que toutes les conditions sont réalisées

|| (ou logique): Vérifie qu'une des conditions est réalisée

! (NON logique): Inverse l'état d'une variable booléenne (retourne la valeur 1 si la variable vaut 0, et 0 si elle vaut 1)

la valeur retournée par ces opérateurs = 1 si la condition est vraie et 0 sinon.

Dans une expression de type:

expression-1 ***op-1*** *expression-2* ***op-2 ...****expression-n*

l'évaluation se fait de gauche à droite et s'arrête dès que le résultat final est déterminé.

Exemple:

```
int x=1 , y=3 , z
if ((x >= 0) && (y <= 0) &&  !(z == 0))
```

La dernière clause ne sera pas évalué car y>0

8.5. Les opérateurs arithmétiques

Les opérateurs arithmétiques classiques sont:

- l'opérateur unaire (- changement de signe)

et les opérateurs binaires :

- l' addition (+)
- la soustraction (-)
- La multiplication (*)
- La division (/)
- Le modulo : donne le reste de la division (%) { il s'applique qu'`a des opérandes de type entier. }

Remarque: en C il n'y a pas d'opérateur effectuant l'élévation à la puissance . il faut utiliser la fonction pow(x,y) de la librairie math.h pour calculer X^y

8.6. Les opérateurs d'incrémentation et de décrémentation

Les opérateurs d'incrémentation (++) et de décrémentation (--) s'utilisent aussi bien en suffixe (X++) qu'en préfixe (++X).

Dans les deux cas la variable X sera incrémentée, toutefois dans :

X++: la valeur retournée sera la valeur de X avant l'incrémentation (l'ancienne valeur)

++X: la valeur retournée sera la valeur de X après l'incrémentation (la nouvelle valeur)

Exemple:

```
int a = 3, b, c;
b = ++a; /*  b valent 4 */
c = b++; /* c = 4 et b = 5 */
```

- **Règles de priorité des opérateurs**

Le tableau suivant classe les opérateurs par ordres de priorité décroissants. Les opérateurs placés sur une même ligne ont la même priorité.

Ordre de priorité des opérateurs
() []
! ++ --
* / %
+ -
< <= >= >
== !=
&&
||
%= /= *= -= += =

Table2. **Ordre de priorité des opérateurs**

9. Les fonctions d'entrées/sorties

Il s'agit des fonctions de la librairie standard **stdio.h** utilisées avec les unités classiques d'entrées-sorties, qui sont respectivement le clavier et l'´ecran.

9.1. La fonction d'affichage « printf »

La fonction printf est une fonction d'affichage qui permet d'imprimer ses arguments sur écran: c'est est une fonction d'impression formatée, ce qui signifie que les données sont converties selon le format particulier choisi.

la fonction **printf** permet d'afficher un texte entre double apostrophe tel que:

Printf (" bonjour tout le monde ")

cela va afficher sur écran le texte : bonjour tout le monde

Elle permet aussi d'afficher une liste des valeurs dans un format spécifié. Sa syntaxe est la suivante:

printf ("chaine de format" , arg1, arg2,...)

La *chaine de format* contient le texte à afficher et les spécifications de format correspondant à chaque expression de la liste. Les spécifications de format ont pour but d'annoncer le format des données à visualiser.

Chaque spécification de format commence par un % et se termine par un caractère de conversion désignant le format d'impression (d, f, c, s,,,) (voir table 3).

format	**Conversion en**	**Représentation en**
%d	int	Décimale signé
%f	float	Décimale virgule fixe
%c	char	caractère
%s	Char*	Chaine de caractères
%o	Unsigned int	Octale non signé
%x	Unsigned int	Hexadécimale non signé

Table 3. quelque Formats d'impression pour la fonction printf

Exemple 1:

```
#include <stdio.h>
main()
{ float x= 3.5 ;
 int y = 2 ;
 printf(" Le résultat de la division de %f par %d est %f \n", x, y, x/y) ;
}
```

Ce programme imprime à l'écran :

le résultat de la division de 3,5 par 2 est 1,75

remarque :

\n permet d'envoyer le curseur à la ligne

\t permet de faire une tabulation

Exemple 2 :

```
#include <stdio.h>
main()
{
int i =23;
char c = 'A';
Printf (" impression de i: \n");
printf("%d \t  %o \t %x", i, i, i);
Printf (" impression de C: \n");
printf("%c \t  %d ", c, c);
}
```

Ce programme imprime à l'écran :

```
 23  27     17
 impression de C:
A     65
```

9.2. La fonction de saisie scanf

La fonction scanf lit les données saisis au clavier, les interprète selon les spécifications incluses dans le champs « **format** » et stocke les résultats dans les arguments ***arg1, arg2*** … qui précisent les adresses où il faut stocker les entrées correspondantes.

Sa syntaxe est la suivante :

scanf(" *format*", &*arg1*,...,&*argn*)

format indique le format dans lequel les données lues sont converties. Elle ne contient pas d'autres caractères (notamment pas de \n).

Comme pour la fonction printf , les format sont spécifiées par un caractère (d, f, c , s,,,) précédé du signe %.

Exemple1 **:** Saisie d'une valeur entière, et stockage de cette valeur dans la variable x

```
 #include <stdio.h>
 main()
{   int x ;
scanf("%d", &x) ;
}
```

Exemple 2

```
  #include <stdio.h>
  main()
{  float x;
  printf("entrez un nombre réel x = ");
  scanf("%f", &x);
  printf(« x = %f\n",x);
}
```

9.3. Affichage et lecture des caractères

La fonction **getchar()** permet de lire un caractère. Elle retourne un *int* correspondant au caractère lu. Sa syntaxe est la suivante :

caractere = getchar();

La fonction **putchar()** permet d'imprimer (afficher) un caractère sur la sortie standard (écran ou un fichier).Sa syntaxe est la suivante:

putchar (caractere);

Les fonctions getchar() et putchar() sont fonctions d'entrées-sorties non formatées.

Exercice:

Ecrire un programme qui permet d'afficher les lettre A M G avec leur code ascii et leur code en hexadécimale

Solution :

```
#include <stdio.h>
Main()
{ char x='A', y='M', z='G';
Printf (' caractère = %c \t  code=%d \t code hexadécimale= %x \n  », x, x, x);
Printf (' caractère = %c \t  code=%d \t code hexadécimale= %x \n  », y, y, y);
```

Printf (‘ caractère = %c \t code=%d \t code hexadécimale= %x \n », z, z, z); }

10. Les Structures de contrôle

Les instructions d'un programme sont exécutées généralement d'une manière séquentielle. Cependant, dans plusieurs cas, on est amené soit à choisir entre deux ou plusieurs choix d'exécution ou bien à répéter l'exécution d'un ensemble d'instructions, pour cela nous avons besoins de structures de contrôle qui nous permettent de contrôler le fonctionnement de notre programme. Parmi les structure de contrôle nous citons : les structure de contrôle conditionnel et les structure de contrôle répétitives (boucles).

10.1. Structures de contrôle conditionnel

Ces structures sont utilisées lorsqu'on a des instructions ou des blocs d'instructions qui ne doivent s'exécuter que sous certaines contraintes ou si certaines conditions sont vérifiées. Ces instructions permettent de déterminer quelles instructions seront exécutées et dans quel ordre.

10.1.1. Le Branchement conditionnel if….else

Dans ce type de test un des deux blocs d'instructions est exécuté suivant la satisfaction ou non d'une condition (expression logique qui peut être vraie ou fausse). Sa forme générale est la suivante :

if (*condition*)
bloc instruction-1
else
bloc instruction-2

Si la condition est vraie, c'est ***bloc instruction-1*** qui est exécuté, sinon c'est ***bloc instruction-2.***

Exemple1 : Maximum de deux valeurs réelles

```
# include <stdio.h>
Main()
{ float x,y;
```

```
Printf(" saisir deux valeurs réelles");
Scanf ( %f%f , &x,&y);
  if (x<y)
      printf ("le maximum est : %f", y)
 else
      printf ("le maximum est: %f ", x);
}
```

Exemple 2: résolution d'une équation ax2+bx+c = 0 avec a ≠ 0

```
#include <stdio.h>
#include <math.h>
 main()
{   float a,b,c,x1,x2,delta;
    printf("Entrer les coefficients a,b et c\n");
    scanf("%f%f%f",&a,&b,&c);
     delta = pow(b,2)-4*a*c;
       if (delta > 0 )
      { x1 = (-b-sqrt (delta)/(2*a));
        x2 = (-b+sqrt (delta)/(2*a));
       printf("l'equation admet deux solution : x1 %f et x2 %f", x1,x2);
       }
       else
         if (delta == 0)
         {   x1 = -b/(2*a);
             printf("l'equation admet une solution double %f", x1);
         }
       else
              printf("l'equation n'admet pas de  solution");
 }
```

Remarque :

La structure ***if (condition) expression-1***
else expression-2

est équivalente à

condition ? expression-1 : expression-2

Exemple:

x >= 0 ? x++ : x--

Cette instruction est équivalente à

```
if (x>=0)
        x++ ;
else
        x-- ;
```

10.1.2. Branchement multiple « switch »

Quand on dispose de plusieurs choix, il est préférable d'utiliser l'instruction de branchement multiple 'switch '.

Sa forme générale est celle-ci :

```
  switch ( expression )
    {    case valeur-1:
                    liste d'instructions 1
                    break;
         case valeur-2:
                    liste d'instructions 2
                    break;
                    ...
         default:
                  liste d'instructions x
         break;
    }
```

Si la valeur de « *expression* » est égale à l'une des v*aleurs*, la ***liste d'instructions*** correspondant est exécutée, sinon la ***liste d'instructions x*** correspondant à ***default*** est exécutée.

Exemple:

```
#include <stdio.h>
 int main()
    {  char lettre;
       printf ("Entrer un caractère\n");
       scanf ("%c", &lettre);
       switch(lettre)
           {  case 'r' :  printf ("Rouge");
                          break;
              case 'b' : printf ("Bleu");
                          break;
              case 'n': printf ("Noir");
                         break;
              default : printf ("Erreur");
                          break;
            }
         }
```

10.2. Les structures de contrôle répétitives (les boucles)

Les *boucles* permettent de répéter une série d'instructions un nombre fini de fois.

10.2.1. La Boucle while

La syntaxe de la boucle while est la suivante :

```
while ( condition )
      {
        liste d' instructions
       }
```

Tant que ***condition*** est vérifiée, *liste d'instruction* est exécutée. Si *condition* n'est pas verifée au départ, *liste d'*instruction ne sera jamais exécutée.

Exemple:

```
#include <stdio.h>
Main()
{ int i;
  i = 1;
  while (i < 10)
    {
       printf(" i = %d",i);
       i++;
     }
}
```

10.2.2. La Boucle do…while

Sa syntaxe est la suivante :

Do {
liste d'instructions
}
while (*condition*)

*La boucl do…while s'*exécute au moins une fois même si la condition n'est pas vérifiée.

Exemple:

```
#include <stdio.h>
Main()
{ int i;
  i = 1;
  do
    {  printf(" i = %d",i);
       i++;
     } while (i < 10)
}
```

10.2.3. La Boucle for

Sa syntaxe est la suivante :

for (*initialisation* ; *condition* ; *incrémentation*)

liste d'instruction

Une version équivalente en utilisant la boucle while est :

***initialisation*;**
while **(*condition*)**
{
Liste d'instruction
***incrémentation*;**
}

Exemple: afficher tous les entiers de 0 à 10,

```
#include <stdio.h>
Main()
{  int i;
 for (i=0; i<=10; i++)
 printf(" i = %d",i);
}
```

A la fin de cette boucle, i vaudra 11.