Chapitre I

Notions de bases de la programmation en C++

I.1. Introduction

Le langage C++ a été conçu par Bajarne Stroustup[1] en 1983, il fait partie de langages de programmation les plus utilisés dans le monde en raison de sa rapidité d'exécution, richesse de sa bibliothèque, variété de ses techniques de programmation...etc. Il est entièrement basé sur le langage C mais avec plusieurs nouveautés comme la possibilité de la programmation orientée objet [1]. Ce chapitre sera focalisé sur les notions de base de la programmation en C++. Dans la première partie, nous allons présenter les logiciels nécessaires pour la programmation en C++. Ensuite, nous présenterons en détail les règlements de la déclaration et d'utilisation des constates, les variables et les références. Dans la dernière partie de ce chapitre nous allons étudier les différentes structures de contrôle conditionnel et itératif en C++. Plusieurs exemples d'application et des exercices avec solutions seront présentés pour faciliter la compréhension des notions étudiées.

I.2. Logiciels nécessaires pour programmer en C++

Pour écrire et exécuter un programme C++, nous allons besoin au trois logiciels installés sur notre ordinateur :

I.2.1 Editeur de texte (en anglais : Editor)

Editeur de texte est un logiciel permet d'écrire le code source du programme. Le Bloc-Notes de Windows ou ***vi*** du Linux peuvent remplir cette tâche. Néanmoins, il est très recommandé d'utiliser un éditeur de texte intelligent qui peut colorer automatiquement les mots clés, ouvrir et fermer les parenthèses…etc., c'est qui rendre le code du programme lisible et compréhensible.

I.2.2 Compilateur (en anglais : compiler)

Le compilateur est un logiciel permet de traduire le code des instructions (programme) C++ en

[1] Informaticien Danois né le 30 décembre 1950.

langage machine (binaire), ce dernier sera ensuite exécuté par le système d'exploitation de l'ordinateur, voir figure (I.1).

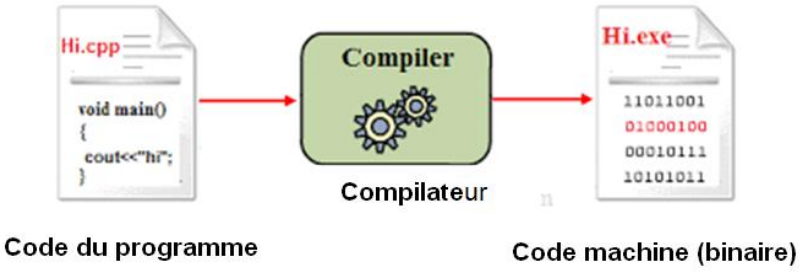

Figure (I.1). Fonction du compilateur : il permet de faire une transformation du code C++ en code machine.

I.2.3 Débogueur (en anglais : debugger)

Le débogueur est un logiciel permet de signaler au programmeur les erreurs du programme, par exemple erreur à l'écriture d'une instruction, syntaxe non respectée, manque d'une parenthèse ou point-virgule…etc.

Remarque : Dans le marcher on peut trouver les trois programme précédents (Editeur de texte, Compilateur et Débogueur) combinés dans un seul logiciel appelé IDE (Integrated Development Environment), on citons par exemple '*Code::Blocks'* et '*Visual C++'*.

I.3. Notions de base de la programmation en C++

I.3.1. Utilisation des commentaire en C++

Les commentaires sont des phrases ou des paragraphes ajoutés au code du programme pour le but d'expliquer son fonctionnement ou d'ajouter des explications à l'attention du lecteur. Ils n'ont aucun impact sur le fonctionnement du programme (c.à.d. le compilateur ne les exécute pas). En C++, il y a deux types de commentaire :

1- **Commentaires courts** : ils peuvent être insérés sur une seule ligne en utilisant les double barre oblique (//).

Exemple I.1

```
Cout <<"Bonjour les étudiants" endl ; //cette instruction affiche une phrase sur l'écran de l'ordinateur.
```

2- **Commentaires longs :** ils peuvent s'étendre sur plusieurs lignes. Le texte du commentaire doit être placés entre les symboles : /* et */.

Exemple I.2

```
Cout <<"Bonjour les étudiants" endl ;
/* L'instruction précédente affiche la chaîne de caractère (phrase) Bonjour les étudiants sur l'écran de l'ordinateur ; */
```

I.3.2. Variables en C++, déclaration et utilisation

Une variable est une partie de la mémoire dont sa contenue (nombre, adresse…) peut être modifié ou initialisé pendant l'exécution du programme.

I.3.2.1 Type de variable

En C++, il existe plusieurs types de variables, voici quelque type de variables les plus utilisées :

bool: mot clé permet de déclarer une variable boolienne, elle peut contenir seulement deux valeurs, true (1), ou false (0). Cette variable réserve un 1bit dans la mémoire.

char: mot clé permet de déclarer une variable de type caractère qui peut contenir un caractère (a, z, ..@,....). Cette variable réserve un 01 octet dans la mémoire.

int: mot clé permet de déclarer une variable entière qui peut contenir un nombre entier dont sa valeur comprise entre -32768 et 32767. Cette variable réserve généralement 02 octets dans la mémoire.

unsigned int: variable de type entier positif, peut contenir un nombre entier positif dont sa valeur est comprise entre 0 et 65535(02 octets).

double, float: variables de type réel.

String: une chaîne de caractère peut contenir un mot ou une phrase.

I.3.2.2 Syntaxe de la déclaration des variables en C++

La déclaration des variables consiste à définir le nom, le type, et la valeur du variable initiale de la variable comme suit :

Type nom(valeur initiale), ou Type nom = valeur initiale ;

Exemple I.3

unsigned int age(30); // variable entière positive nommée age et contient une valeur égale 30.

int mesure = (-1500); // variable entière nommée mesure et contient une valeur égale -1500.

int x ; // variable entière nommée x et non initialisée.

double ksi = 0,707 ; // variable réelle nommée ksi et contient une valeur de 0,707.

char lettre('z'); // variable caractère, nommée lettre et contient le caractère z.

string specialite("Electronique") ; / variable string nommée specialite et contient la chaîne de caractère :"Electronique " */.* [2]

Remarques :

1- Pour la déclaration des variables de type **caractère**, il faut mettre la valeur initial entre deux guillemets (' ') ;

[2] Pour ce niveau, on peut considérer le string comme variable.

2- Pour la déclaration des variables de type **string**, il faut mettre la valeur initial entre guillemet doublé (" ");

I.3.3. Références en C++, déclaration et utilisation

La référence est une étiquette accrochée à une case mémoire (variable par exemple). La notion de référence est propre au langage C++, elle n'existe pas en langage C ou Pascal.

Syntaxe de la déclaration : **type& nom** (nom de la variable à accrocher).

Où le type de référence doit être le même de celui de variable à accrocher, par exemple une référence de type 'int' ne peut pas accrocher avec une variable de type double ou float.

Exemple I.4

```
5   int main()
6   {
7       int age ("22");
8       int &ref_age (age);
9       cout << " Votre age via variable est:"<< age << endl;
10      cout <<"votre age via reférence est: " << ref_age<< endl;
11  return 0;
12  }
```

Dans le programme ci-dessous, on a déclaré une variable 'age' de type entier initialisée par 22 (ligne 7). Dans la ligne 8, on a déclaré une référence 'ref_age' accrochée au variable 'age'. Dans la ligne 9, on a affiché le contenu de la variable 'age' par la méthode habituellement utilisée. Par contre, dans la ligne 10, on a affiché le contenu de la variable 'age' en utilisant cette fois ci la référence 'ref_age'. **Donc on a pu accéder au contenue de la variable 'age' via la référence 'ref_age'.**

I.3.4. Constantes, déclaration et utilisation

Un constant est une partie de la mémoire dont son contenu ne peut pas être changé durant l'exécution du programme. Les constants sont déclarés en écrivant le mot-clé '**const'** suivi par le type, le nom et la valeur de chaque constante.

Exemple I.5

```
const float PI (3.1415) ; / /Déclaration d'une constante nommée PI de type float et de valeur 3.1415
const char BEEP ('\b') ; / * Déclaration d'une constante nommée BEEP de type char, et contient
le caractère \b (bip sonore) * /.
```

I.3.5. Opérateurs standards en C++

I.3.5.1. Opérateurs arithmétiques

Les opérations arithmétiques sont utilisées pour les calculs numériques. Il y a cinq opérateurs arithmétiques sont : + (Addition), - (Soustraction), ***** (Multiplication), / (Division), **%** (Modulo ou reste d'une division entière, par exemple 11%3 donne 2, 15%3 donne 0).

I.3.5.2. Opérateurs de comparaison

Les opérateurs logiques sont utilisés généralement pour faire une condition sur l'exécution ou non d'une ou multiple instruction.

&& (et logique), *exemple*, 1 && 0 donne 0, 1 && 1 donne 1.

'| |' (ou logique), *exemple*, 1| | 0 donne 1.

'!' (négation logique), *exemple* !1 donne 0, !2021 donne 0.

== (est-il égal à ?, *exemple* x==10, y+1== 04.

!= (différent de ?), *exemple* x !=10.

<, <=, >,>= (inférieur de, inférieur ou égal de, supérieur de….).

I.3.5.3. Opérateurs d'affectation composée

+= (ajouter à)

-= (diminuer de)

*= (multiplier par)

/= (diviser par)

%= (modulo)

Exemple I.6

```
#include <iostream>
#include <string>  // Bibliothèque string
using namespace std;
int main()
{
    int a(4),b(2);// Déclaration de deux variables (a et b)de type
                  // entier initialisée respectivement à 4 et 2

    a+=2;   // a vaut 6
    b-=2;   // b vaut 0
    a *=3;  // a vaut 18
    a /=3;  // a vaut 6
    a %=5;  // a vaut 1
    return 0;
}
```

I.3.5.4. Opérateurs d'incrémentation et de décrémentation

Evidemment, pour incrémenter une variable x on utilise x=x+1, et pour le décrémenter on utilise x=x-1. Cependant, on peut effectuer ces opérations de manière plus simple et plus avantageuse grâce aux opérateurs ++ (incrémentation par 1) et -- (décrémentation par 1).

Exemple I.7

Le programme montre l'utilisation des opérateurs d'incrémentation et de décrémentation.

```
#include <iostream>
using namespace std;
int main()
{
    int x(10),y(5);// Déclaration de deux variables(x et y) de type
                   // entier initialisée respectivement à 10 et 5
```

```
 7       x++;// x vaut 11
 8       y--;// y vaut 4
 9       ++x;// x vaut 12
10       ++y;// y vaut 5
11       return 0;
12   }
```

Remarque importante

Si on a une incrémentation/décrémentation et une affectation en même temps, il y une différence importante entre les écritures (x++, x--) et (++x, --x). La différence entre les deux écritures est illustrée dans l'exemple suivant:

z = x++; // **Passe d'abord** la valeur de x a z, puis incrémenter.

z = ++x ; // **Incrémenter d'abord** la valeur de x et **passe après** la valeur incrémentée à z.

Supposant que les valeurs initiales de z et x étaient respectivement 0 et 1, le contenu de x et z après l'exécution de chaque instruction sont montrés dans les commentaires ci-dessous :

z=0, x=1; // z= vaut 0 et x= vaut 1.

z = x++; // z vaut 1 et x vaut 2 (**Passe d'abord** la valeur de x a z, puis incrémenter).

z = ++x; // z vaut 3 et x vaut 3 (Incrémenter d'abord x et **passe après** la valeur incrémentée à z).

I.3.5.5. Opérateur d'entrée/sortie

Pour d'afficher une valeur numérique ou chaine de caractère sur l'écran de l'ordinateur, on utilise le mot clé '**cout'** suivi par l'opérateur de sortie '**>>**' et la valeur à afficher (nombre, mot, phrase,...).

Exemple I.8

```
cout << 2021; // Affichage des valeurs numériques
cout <<"Bonjour les étudiants"; // Affichage d'une chaine de caractère (mot ou phrase).
```

Exemple I.9

Exemple:

```
int age(0) ; string nom ("0") ; // Déclaration de deux variables
cout << "Quel est votre nom ?" <<endl; //Demande de l'utilisateur d'introduire son nom
getline (cin, nom ) ; /* lecture de la chaine de caractère (peut contenir un espace) introduit et la stocker dans la variable 'nom'*/
cout<< "Quel est votre age ?"<<endl; //Demande de l'utilisateur d'introduire son age
cin>> age; // lecture du nombre introduit par l'utilisateur et le stocker dans la variable 'nom'.
```

Pour lire au clavier une valeur numérique ou une chaine de caractère, on utilise le mot clé '**cin'** suivi l'opérateur d'entrée **'>>'** et le nom de la variable à utiliser pour stocker la valeur entrée.

Exercice I.1

Ecriviez un programme qui demande à l'utilisateur d'introduire son nom, son prénom, son âge et sa spécialité, puis afficher toutes ces informations dans une seule phrase.

Solution d'Exercice I.1

```
#include <iostream>
#include <string>  // Bibliothèque string
using namespace std;
int main()
{
    string nom("0"),prenom("0"),specialite("0");// Trois variables de type
                                                // string
    int age (0); // une variable de type entier
    cout<< "Quel est votre nom ?"<<endl;//Demander l'âge de l'utilisateur
    cin>> nom ; // lire et stocker le nom de l'utilisateur
    cout<< "Quel est votre prenom ?"<<endl;//Demander le prénom d'utilisateur
    cin>> prenom;// lire et stocker le pénonom de l'utilisateur
    cout<< "Quel est votre specialite ?"<<endl;
    cin>> specialite;
    cout<< "Quel est votre age ?"<<endl;//
    cin>> age;
    /* l'instruction c-dessous affiche le nom le prénom, l'âge et la spécialité
    de l'utilisateur*/
    cout<< "Bonjour "<< nom <<" "<<prenom <<" vous avez "<<" "<<age<<
     " ans" <<" et votre specialite est " <<specialite <<endl;
    return 0;
}
```

Remarque

Dans le programme précédent le nom et le prénom introduit ne doivent présenter des espaces (ex Ben Djamil) sinon cin va lire seulement le mot avent l'espace. Pour lire une chaine de caractère avec espaces, vous pouvez utiliser la fonction *geline* par exemple.

I.3.6. Structure de contrôle alternatifs

Les structures de contrôles alternatifs permettent de définir la suite dans laquelle les instructions sont exécutées, c.à.d. elles permettent d'imposer des conditions pour l'exécution ou non d'un bloc d'instruction.

I.3.7.1. Structure de contrôle alternative if-else

La syntaxe d'utilisation de la structure de contrôle if-else est la suivante :

```
if (Conditions)
   <bloc d'instructions 1>
else
   <bloc d'instructions 2>
```

Si les conditions (une ou multiples) sont vérifiées, alors le *<bloc d'instructions 1>* sera exécuté et le *<bloc d'instructions 2>* est ignoré.

Si les conditions ne sont pas vérifiées, alors le *<bloc d'instructions 2>* sera exécuté et le *<bloc d'instructions 1>* sera ignoré.

Remarque : le <bloc d'instructions> peut-être :

- Multiple d'instruction placé impérativement entre deux accolades ;
- une seule instruction placée facultativement entre deux accolades.

Exemple I.10

Le programme suivant lit deux nombres différents (a, b) entrés au clavier et affiche le plus grand des deux nombres.

```
#include <iostream>
using namespace std;

int main()
{
    double a(0), b(0);// Déclaration des variables a et b

    cout << "Introduisez trois nombres différent:" << endl;
    cin >> a >> b; // lire det stocker les deux nombres
                   // dans les varaibels a,b

    if (a>b) // si la condition 'a>b' est vérifée l'instruction suivante
             // sera exécutée
    {
     cout << " le premier nombre " << a << " est le plus grand";
    }
    else // si la condition 'a>b' n'est pas vérifée l'instruction suivante
         // sera exécutée
    {
     cout << " le deuxième nombre " << b << " est le plus grand";
    }
return 0;
}
```

I.3.7.2. Structure de contrôle alternative if-else if- …-else

Dans le cas de plusieurs alternatives (conditions), il est possible de combiner plusieurs structures if-else, la syntaxe dans ce cas est comme suit :

```
if (Condition 1)
   <bloc d'instructions 1>
else if (Condition 2)
   <bloc d'instructions 2>
        .....
        .....
else if (Condition n)
<bloc d'instructions n>
else // Facultative, elle est exécuté lorsqu'aucune condition n'a été vérifiée
<bloc d'instructions n+1>
```

Les conditions < Conditions 1 >, < Conditions 2 >... < Conditions n > sont évaluées l'une après l'autre jusqu'à ce que l'une de celles soit vérifiée, le bloc d'instructions lié à la condition vérifiée sera exécuté et le traitement de la commande sera ensuite terminé. Le <bloc d'instructions n+1> sera exécuté si seulement si toutes les conditions ne sont pas vérifiées.

Exemple I.11

Le programme suivant demande à l'étudiant d'introduire sa note d'examen en module 'Programmation orientée objet' puis affiche la mention correspondante à la note introduit comme suit selon la grille suivante:

Si 18≤ note <20, la mention est : Excellent ;

Si 16≤ note <18, la mention est : Très bien ;

Si 14≤ note <16, la mention est : Bien ;

Si 10≤ note <14, la mention est : Passable ;

Si 0≤ note <10, la mention est : Insuffisant ;

Si note <0, ou note >0, Demander de l'étudiant d'introduire à nouveau sa note ;

```
#include <iostream>
using namespace std;
int main()
{
 double note (0);
 cout << "Introduisez votre note d'examen en module Progra
 cin >> note;
 if (note <20 && note >=18)
    {
    cout << "Excellent" << endl;
    }
    else if (note < 18 && note >= 16)
    {
    cout << "Très bien" << endl;
    }
    else if (note < 16 && note >=14)
    {
    cout << "bien" << endl;
    }
    else if (note < 14 && note >=10)
    {
    cout << "passable" << endl;
    }
    else if (note < 10 && note >=0)
    {
    cout << "insuffisant" << endl;
    }
    else
    {
    cout << "Votre note doit etre entre 0 et 20" << endl;
    }
return 0;
}
```

Exercice I.2

Ecrivez un programme qui calcule les solutions réelles d'une équation du second degré $ax^2+bx+c=0$, tel que ($a\neq 0, b\neq 0, c\neq 0$).

Solution d'Exercice I.2 :

```
#include <iostream>
#include<cmath> // pour qu'on puissent utilser les fonctions 'pow' et sqrt'
using namespace std;
int main()
{
    double a(0),b(0),c(0),delta(0);
    cout << "Introduisez les valeurs de a, b et c" << endl;
    cin >>a >> b >> c;//lecture des valeur a,b et c
    delta= pow(b,2)-4*a*c;//Calcul de delta
    if (delta>0)
    {   // delta >0==> il y a deux solutions
```

```
12          double x1= (-b-sqrt(delta))/(2*a); // première raçine
13          double x2= (-b+sqrt(delta))/(2*a); // deuxième raçine
14          cout << " Il y a deux solutions: " << x1 <<"\t"<< x2<<endl;
15      }
16      else if(delta==0)
17      {   // il y a une solution double
18          double x1= (-b-sqrt(delta))/(2*a);
19          cout << "Il y a une solution: " << x1<<endl;
20      }
21      else
22      {   // pas de solution dans l'ensemble R
23          cout << "il n'y a pas de solutions dans |R: "<<endl;
24      }
25      return 0;
26  }
```

I.3.7.3. Structure de contrôle alternative Switch

La structure 'switch' permet de tester l'égalité du contenu d'une variable (de type int ou char) pour plusieurs valeurs constantes. En plus, elle offre une écriture alternative plus organisée et plus simple que celle de l'instruction 'if'. Néanmoins, cette structure présente les limitations suivante :

- Elle ne permet de tester que l'égalité (on ne peut pas tester la supériorité ou l'infériorité) ;
- Elle ne peut être utilisé qu'avec des nombres entiers (type int) ou des caractères (type char), on ne peut pas l'utiliser avec des nombres de type double ou float par exemple;

Syntaxe de la structure switch :

switch (*expression*)
{
case constante 1 : // *si la valeur de l'expression* == *constante* 1
< *blocd'instructions*1 >
break ; // *retour* (*fin de test*)
. . .
case constante n : // *si la valeur de l'expression* == *constante n*
< *bloc d'instructions n*+1 >
break ; // *retour* (*fin de test*)
default : // *si la valeur de l'expression est différente de toutes les constantes*
< *bloc d'instructions n*+1 >
}

La syntaxe consiste à évaluer l'expression (placée devant switch) puis chercher la valeur de l'expression dans la liste des constantes placées devant chaque 'case'. Si la valeur est trouvée, le bloc d'instructions correspondant est exécuté et le processus est terminé. Sinon, le programme bascule vers le default (qu'est facultatif) et exécute le bloc d'instrucion n+1.

Exercice I.3

En utilisant la structure 'switch', écrire un programme qui demande de l'étudiant d'introduire son rang de classement au Master (A, B, C, D, ou E) et affiche ensuite le coefficient qui va multiplier par sa moyenne générale, on donne :

rang= A, le coefficient est 1 ;

rang= B, le coefficient est 0.8 ;

rang= C, le coefficient est 0.7 ;

rang= D, le coefficient est 0.6 ;

rang= E, le coefficient est 0.5 ;

Solution Exercice I.3 :

```
#include <iostream>
using namespace std;
int main()
{
    char rang('0');
    cout << "Vous etes dans quel rang (A,B,C, D Ou E)?" << endl;
    cin >>rang;
    switch(rang)
    {
    case 'A':
        cout << "Votre moyenne sera multipliée par: "<< 1 << endl;
        break;
    case 'B':
        cout << "Votre moyenne sera multipliée par: "<< 0.8 << endl;
        break;
    case 'C':
        cout << "Votre moyenne sera multipliée par: "<< 0.7 << endl;
        break;
    case 'D':
        cout << "Votre moyenne sera multipliée par: "<< 0.6 << endl;
        break;
    case 'E':
        cout << "Votre moyenne sera multipliée par: "<< 0.5 << endl;
        break;
    default:
        cout << "Les rangs possibles sont: A,B,C,D ou E "<< endl;
    }
    return 0;}
```

I.3.7. Structure de contrôle répétitif (les boucles)

Les boucles permettent de répéter l'exécution d'un bloc d'instructions plusieurs fois. Le nombre de répétition soit dépend d'une validation d'une condition logique ou défini par le programmeur. En C++, il existe trois types de boucles qui sont : boucle 'while', la boucle 'do-while' et la boucle 'For'.

I.3.7.1. Boucle while

La syntaxe de la boucle while est la suivant :

while (*condition*)
{
  < *bloc d'instructions* >
}

Tant que la <*condition*> est vérifiée, le <*bloc d'instructions*> est exécuté. Sinon l'exécution continue avec l'instruction qui suit le <*bloc d'instructions*>.

Exemple I.12

Le programme suivant permet d'afficher les nombre entier de 0 à 9.

```
#include <iostream>

using namespace std;

int main()
{
    int i(0);
    while(i<10)
    {
    cout << i << endl;
    i++;
    }

    return 0;
}
```

I.3.7.2. Boucle do-while

La boucle 'do-while' est similaire à la boucle 'while' sauf que la boucle 'do-while' exécute le <*bloc d'instructions*> et évalue après la condition. La syntaxe de cette boucle est :

do
{
< bloc d'instructions >
}
while(*condition*) ;

Remarque :

Avec la boucle do-while, le <*bloc d'instructions*> est exécuté au moins une fois.

Exemple I.13

Le programme suivant redemande de l'utilisateur d'introduire son âge tant que celui-ci est inférieur à 0, égal à 0 ou supérieur à 110.

```
#include <iostream>

using namespace std;

int main()
{
    int age(0);
    do
    {
    cout << "Introduisez votre age " << endl;
    cin >> age;
    }
    while (age <=0 || (age >110));
    cout << "Merci d'avoir entré votre age " << endl;
    return 0;
}
```

I.3.7.3. Boucle for

La syntaxe de cette la boucle 'for' est la suivante :

for (*initialisation*; *condition*; *miseà jour*)
{
< bloc d'instructions >
}

Où :

- L'initialisation permet de déclarer et/ou initialiser la (les) variable(s) de contrôle de la boucle ;
- La condition permet de décider si la boucle est répétée ou non ;
- La mise à jour permet d'actualiser la (les) variable (s) de contrôle de la boucle.

Exemple I.13

Le programme suivant permet de calculer la somme de N premiers nombres entiers (s=1+2+…….N).

```
1   #include <iostream>
2   using namespace std;
3
4   int main ()
5   {
6       int N(0);double somme(0);
7       cout << "Entrer un entier positif"<<endl;
8       cin >> N;
9       for(int i=1;i<=N;i++)
10      {
11          somme+=i;
12      }
13  cout << "la somme de N premiers entirs égale: "<<somme<< endl;
14  return 0;
15  }
```

I.4. Résumé

Dans ce chapitre, nous avons étudié les notions de base de la programmation en C++. En premier temps, nous avons étudié comment déclarer et utiliser les variables, les constantes, et les références. Nous avons étudié également les opérateurs d'affectation simple et composée, les opérateurs d'incrémentation et de décrémentation, les opérateurs d'entrée et de sortie et les opérateurs de comparaison. Ensuite, nous avons étudié en détail les structures de contrôle alternatif conditionnel (if, if-else, switch) et les structure de control réplétive (while, do-while et for). Plusieurs exemples et des exercices avec des solutions sont présentés pour faciliter la compréhension et la maitrise des notions étudiées. Dans le prochain chapitre nous allons étudier les fonctions, les tableaux, les pointeurs et les structures.