Chapitre1

Introduction au langage C++

Plan

1- Définition
2- Historique
3- Notions de base
4- Instructions de base
5- Fonctions
6- Fonction main
7- Fonctions d’entrée/sortie de base
8- Structures de contrôle

1. Définition

Le langage C++ est un langage évolué et structuré. C'est en ce sens une évolution du langage C. Il possède en outre les fonctionnalités de la programmation orientée objet.

2. Historique

Dans les années 80, Bjarne Stroustrup développa le C++ afin d'améliorer le C, en lui ajoutant des « des classes ». Le premier nom de ce langage fut d'ailleurs « C with classes ». Ce fut en 1998 que le C++ fut normalisé pour la première fois. Une autre norme corrigée fut adoptée en 2003. Une mise à jour importante fu C++11, suivie de C++14, ajoutant de nombreuses fonctionnalités.

3. Notions de base

Commentaires:

Il existe deux types de commentaires en C++:

- Commentaire sur une seule ligne:

```
// Commentaire
```

- Commentaire sur plusieurs lignes:

```
/* Commentaire */
```

3. Notions de base

Types prédéfinis:

Il existe plusieurs types prédéfinis:

- Les booléens: bool (true ou false)
- Les caractères: char
- Les entiers: int, long int, short int, unsigned int
- Les réels: float, double, long double

3. Notions de base

Notation des valeurs:

Les entiers se notent comme suit:

- Base 10: les chiffres {0,1,…, 9}, les signes (+) et (-). (Exemple: 1234, -1234)
- Base 16: les chiffres {0,1,…,9,A,B,…,F}. (Exemple: 0x1AE,0x76B)
- Base 8: les chiffres {0,1,…,7}. (Exemple: 0145)

3. Notions de base

Les flottants se notent comme suit:

[signe] chiffres [.[chiffres]][e|E [signe] exposant][f]

- Signe: le signe du flottant
- Exposant: L'exposant du flottant (e ou E)
- Suffixe: le suffixe (f) précise si le flottant est de type float ou non.

(Exemple: -123.56f, 12e-8)

3. Notions de base

Les caractères se notent entre guillemets simples:

'A' 'b' '('

Les chaines de caractères se notent entre double guillemets:

"Chaine de caractères"

3. Notions de base

Définition des variables:

- Les variables peuvent être définies comme suit: type identificateur;

  (Exemple: double poids;)

- Il est possible de créer et d'initialiser une variable: type identificateur=valeur;

  (Exemple: int i=0;)

3. Notions de base

Remarques:

- Les variables peuvent être définies quasiment n'importe où dans le programme.
- Les variables non initialisées contiennent des valeurs aléatoires, il faut éviter de les utiliser avant une initialisation correcte.

3. Notions de base

Définition d'un tableau:

- La définition d'un tableau s'effectue en faisant suivre l'identificateur du tableau d'une paire de crochets, contenant le nombre d'éléments:

  type identificateurTab[taille]([taille]…);

  (Exemple: float Montab[100];)

- L'accès à un élément du tableau s'effectue par son indice: type identificateurTab[indice];
- (Exemple: Montab[0]=5;)

4. Instructions de base

Instruction:

- En général, une instruction se termine par un point virgule (;), ce caractère marque la fin.
- Les instructions contiennent des expressions (Combinaison entre les opérandes et les opérateurs).

4. Instructions de base

Les principales opérations:

- Affectation: variable = valeur;
- Opérations de base: valeur op valeur

  où op est l'une des opérateurs suivants:

  +,-,*,/,%,&,|,^,~,<<,>>,++,--,?:

4. Instructions de base

- Affectation composée: variable opaff valeur

où opaff est l'une des opérateurs suivants:

+=,-=,*=

Cette syntaxe est équivalente à:

variable = variable op valeur

(Exemple: A+= 5 est équivalent à A=A+5)

4. Instructions de base

- Opérateur ternaire d'évaluation conditionnelle:

Test ? Expression1 : Expression2

(Exemple: Min= (i<j) ? i : j;)

5. Fonctions

Définition:

- La définition de la fonction se fait comme suit: type identificateur (paramètres)

  { … /*Instructions*/ }

- La syntaxe du paramètre est la suivante:

  type param [=valeur]

5. Fonctions

- Le mot return permet de retourner le résultat d'une fonction.
- La procédure est une fonction qui ne retourne rien, ce qui nécessite l'utilisation du mot void:

```
void  identificateur (paramètres)
    { … /*Instructions*/   }
```

5. Fonctions

- L’appel de la fonction s’effectue en utilisant son identificateur et en déterminant les valeurs de ses paramètres.
- La déclaration d’une fonction signale son existence au compilateur:

  type identificateur (paramètres);

5. Fonctions

- Le passage de paramètres à une fonction s'effectue en trois modes:
  - Passage par valeur:

    type identificateur (type param);
  - Passage par adresse:

    type identificateur (type *param);
  - Passage par référence:

    type identificateur (type & param);

5. Fonctions

- Exemple:

```
int min(int,int);  //Déclaration de la fonction

int main()         //La fonction principale
{
    int i=min(2,3);
    return 0;
}

int min(int i,int j)  //Définition de la fonction
{
    if(i<j) return i;
    else return j;
}
```

5. Fonctions

Surcharge des fonctions:

- La surcharge est la possibilité de définir plusieurs fonctions qui portent le même nom, mais avec une différence au niveau des paramètres.

5. Fonctions

- Exemple:

```
//Définition des fonctions
float test(int i,int j)
{
    float r;
    r=(float) i+j;
    return r;
}

float test(float i,float j)
{
    float r;
    r=i*j;
    return r;
}
```

Appel la 1ère fonction

```
//Appel des fonctions
float a,b;

a=test(2,3);

b=test(2.5,3.2);
```

Appel la 2ème fonction

5. Fonctions

Fonction inline:

- Une fonction inline est une fonction déclarée avec le mot clé inline:

inline type identificateur (paramètres)

{ … /*Instructions*/ }

- Exemple:

```
inline int max(int i,int j)
{
     if(i>j) return i;
     else return j;
}
```

5. Fonctions

- C’est une demande au compilateur de remplacer l’appel de la fonction par le code
- Ce type de fonction augmente la performance du programme (la rapidité) .
- Les restrictions des fonctions inline sont:
  - Elles ne peuvent pas être récursives
  - Elles ne sont pas instanciées

5. Fonctions

Fonction statiques:

- Une fonction statique est une fonction déclarée avec le mot clé static:

  static type identificateur (paramètres)

  { … /*Instructions*/ }

- Exemple:

```
static float local(float i,float j)
{
    return i*j;
}
```

5. Fonctions

- Lorsque on définit une fonction dans un fichier, elle peut être utilisée dans un autre fichier (Fonction externe).
- Cependant, on peur avoir des fonctions utilisables localement (uniquement dans le fichier où elle a été déclarée), ce type fonction est appelé « Fonction statique ».

5. Fonctions

Fonction prenant un nombre variable de paramètres:

- Dans ce type de fonction, le nombre de paramètres peut apparaître variable à l'appel de la fonction:

```
type  identificateur (paramètres,…)
    { … /*Instructions*/  }
```

5. Fonctions

- On doit disposer d'un critère pour savoir le dernier paramètre: le nombre de paramètre qui peut être fourni en premier, une valeur de paramètre particulière qui détermine la fin de la liste ou bien une chaine descriptive.
- L'accès à la liste de paramètres s'effectue à l'aide de : #include<stdarg.h>

5. Fonctions

- On doit disposer d'un critère pour savoir le dernier paramètre: le nombre de paramètre qui peut être fourni en premier, une valeur de paramètre particulière qui détermine la fin de la liste, …
- L'accès à la liste de paramètres s'effectue à l'aide de : #include<stdarg.h> hhghg
- Outils à utiliser: le type (va_list) et les expressions (va_start, va_arg et va_end)

5. Fonctions

Principe de récupération(liste des paramètres):

- Déclaration d'une variable de type:

  va_list variable

- Initialisation de la variable déclarée:

  va_start(variable,paramètre)

- Récupération des paramètres :

  va_arg(variable,type)

- Destruction de la variable: va_end(variable)

5. Fonctions

- Exemple:

```c
#include <stdarg.h>
/* Fonction effectuant la somme de "compte" paramètres : */
double somme(int compte, ...)
{
    double resultat=0; /* Variable stockant la somme. */
    va_list varg;            /* Variable identifiant le prochain paramètre. */
    va_start(varg, compte); /* Initialisation de la liste. */
    do /* Parcours de la liste. */
    {
        resultat=resultat+va_arg(varg, double);
        compte=compte-1;
    } while (compte!=0);
    va_end(varg); /* Terminaison. */
    return resultat;
}
```

6. Fonction main

Définition:

- Lorsqu'un programme est chargé, son exécution commence par l'appel d'une fonction spéciale du programme appelée « main » (principal en anglais)
- Cette fonction marque le début du programme.
- Exemple:

```
int main()
{
    /* le code du programme*/
    return 0;
}
```

7. Fonctions d'E/S

- Un flux est une notion qui représente un flot de données séquentielles en provenance d'une source de données ou à destination d'une autre partie du système.
- Les programmes disposent (dès leur lancement) de trois flux d'entrée/sortie standards.

7. Fonctions d'E/S

- Généralement, le **flux d'entrée standard** (stdin) est associé au flux de données provenant d'un terminal, et **le flux de sortie standard** (stdout) à la console de ce terminal. Le troisième flux standard (stderr) est **le flux d'erreur standard** qui, par défaut, est également associé à l'écran, et sur lequel le programme peut écrire tous les messages d'erreur.

7. Fonctions d'E/S

Fonction printf:

printf(chaîne de format[, valeur [, valeur [...]]])

- La chaîne de format peut contenir du texte, mais surtout elle doit contenir autant de formateurs (%indicateur) que de variables à afficher.

7. Fonctions d'E/S

	Type de donnée à afficher	Caractère de formatage
Numériques	Entier décimal signé	d
	Entier décimal non signé	u ou i
	Entier octal non signé	o
	Entier hexadécimal non signé	x (avec les caractères 'a' à 'f') ou X (avec les caractères 'A' à 'F')
	Flottants de type double	f, e, g, E ou G
Caractères	Caractère isolé	c
	Chaîne de caractères	s
Pointeurs	Pointeur	p

7. Fonctions d'E/S

- Exemple:

```c
#include <stdio.h>
int main(void)
{
int i = 2;
printf("Voici la valeur de i : %d.\n", i);
return 0;
}
```

7. Fonctions d'E/S

Fonction scanf:

scanf(chaîne de format,&variable [, &variable [...]])

- La syntaxe du formateur : %type.

7. Fonctions d'E/S

- (cout): le symbole associé au flot de sortie standard (terminal)
- (cin) : le symbole associé au flot d'entrée standard (clavier)
- (>>): la lecture dans un flot
- (<<): l'écriture dans un flot
- (iostream): la librairie à inclure

7. Fonctions d'E/S

cout:

cout<<expression;

avec expression peut être de type:

- de base (int, long, bool, float, double,...)
- Pointeur sur type de base
- Chaîne de caractères (char*)

7. Fonctions d'E/S

- Exemple:

```
#include<iostream>
using namespace std;
int main()
{   //Ecriture
    cout << "La somme est : " ;
    cout << (12.5 + 3.0) << endl;
    return 0;
}
```

7. Fonctions d'E/S

cout:

cin>>expression;

avec expression peut être de type:

- de base (int, long, bool, float, double,...)
- Pointeur sur type de base
- Chaîne de caractères (char*)

7. Fonctions d'E/S

- Exemple:

```
#include<iostream>
using namespace std;
int main()
{    int x;
     double y;
     //Lecture
     cout << "Entrer deux valeurs: "<<endl;
     cin >> x >> y ;
     cout << "La somme est: ";
     cout << (x + +y) << endl;
     return 0;
}
```

7. Fonctions d'E/S

- Exemple complet:

```c
#include <stdio.h>
long double x, y;
int main(void)
{
printf("Calcul de moyenne\n");          /* Affiche le titre. */
printf("Entrez le premier nombre : ");
scanf("%Lf", &x);                    /* Entre le premier nombre. */
printf("\nEntrez le deuxième nombre : ");
scanf("%Lf", &y);                    /* Entre le deuxième nombre. */
printf("\nLa valeur moyenne de %Lf et de %Lf est %Lf.\n",x, y, (x+y)/2);
return 0;
}
```

8. Structures de contrôle

Structure conditionnelle if:

if(test) {instructions;}

- Exemple:

```
if(N%2==0)
   {printf("%d est un nombre pair \n",N);}
if(n%2!=0)
   {printf("%d est un nombre impair \n",N);}
```

8. Structures de contrôle

Structure conditionnelle if else:

```
if(test) {instructions1;}
else {instructions2};
```

- Exemple:

```
if(N%2==0)
   {printf("%d est un nombre pair \n",N);}
else
   {printf("%d est un nombre impair \n",N);}
```

8. Structures de contrôle

Opérateurs de comparaison		Opérateurs logiques	
==	égalité	&&	et logique
!=	inégalité	||	ou logique
<	infériorité	!	négation logique
>	supériorité		
<=	infériorité ou égalité		
>=	supériorité ou égalité		

8. Structures de contrôle

Structure conditionnelle à choix multiple:

switch(variable/expression)
{ case cas1: [instructions;[break;]]
  …
  case casN: [instructions;[break]]
  [default: instructions;[break];]
}

- Exemple:

```
switch(valeur)
{case 1: printf("Un \n");break;
 case 2: printf("Deux \n");break;
 case 3: printf("Trois \n");break;
 case 4: printf("Quatre \n");break;
}
```

8. Structures de contrôle

Boucle for:

for(initialisation; test; itération)
{instructions;}

- Exemple:

```
somme = 0;
for (i=0; i<=10; i=i+1)
{somme = somme + i;}
```

8. Structures de contrôle

Boucle while:

while(test)

{instructions;}

- Exemple:

```
somme =i= 0;
while ( i<=10 )
{somme = somme + i;
 i=i+1;}
```

8. Structures de contrôle

Boucle do while:

do{instructions;}
while(test);

- Exemple:

```
somme =i= 0;
do{somme = somme + i;
   i=i+1;
  }while ( i<=10 );
```