Université de Monastir

Cours: Programmation C++

Chapitre 1: Apports syntaxique de C++ par apport au C

Réalisé par:
Dr. Sakka Rouis
https://github.com/srtaoufik/coursCpp

1

I. Commentaire

En C	En C++
/* commentaire sur plusieurs lignes */	/* commentaire sur plusieurs lignes */ // commentaire sur une seule ligne

Exemple :

```
void main(){
        //déclaration
        int i ;
        …
}
```

2

II. Déclaration de variables

On peut déclarer des variables locales à un bloque.
Cette facilité autorise une gestion plus précise de la mémoire.

Exemple :

```
for (int i=0 ; i<10 ;i++){
        int j ;
        …
}
```


III. Qualificateur de constantes

Il n'est pas possible de déclarer en C des constantes autres que littérales. Le préprocesseur va nous permettre de définir des constantes symboliques, qui sont en fait vues comme des macros:

Exemple :

```
#define PI 3.14
#define PI2 3.14 * 3.14
…
float x = PI;
```

RQ. En utilisant le mot clé **const** on peut définir des variables constantes

Exemple :

```
#define M 10     //ok C/C++
const int maxarray = 255; //ok C/C++
int T [maxarray]; // Ok C++; not Ok in C
```

IV. Les nouvelles possibilités d'E/S

En C++ on peut toujours utiliser les fonctions standards d'E/S (printf et scanf) offertes par C en utilisant la bibliothèque <stdio.h>.
Cependant il existe une 2ème possibilité fournie par C++ qui donne 2 opérateurs basés sur la notion de flot :
(**un flot est un ensemble de donnée**).

La définition de ces 2 opérateurs d'E/S est disponible dans le fichier d'entré <iostream.h>.

En C/C++	En C++
# include <stdio.h>	# include <iostream.h>
printf (''bonjour'') ;	cout << ''bonjour'' ;
scanf (''%d'', &x) ;	cin >> x ;

5

IV. Les nouvelles possibilités d'E/S

RQ.
Selon le compilateur, par fois, avant d'utiliser cin et cout, il faut écrire:

```
# include <iostream>
using namespace std ;
…
int x;
cout<<''bonjour'' ;
cin>>x;
```

```
# include <iostream>
 …
int x;
std::cout<<''bonjour'' ;
std::cin>>x;
```

Exercice:
Écrire un programme C++ qui permet la saisie de deux entiers A et B. Calculer puis afficher leur somme et leur produit.

6

V. Conversion de types

Le C++ autorise les conversions de type entre variable de type :

char <--> int <--> float <--> double

Exemples :

conversion simple	lors de l'appel d'une méthode
int x=2 ; float y=x ;	void f (int, int) ; float a=2.2 ; int b=2 ; f (a, b) ;

VI. Les arguments par défaut

En C++ on peut préciser la valeur prise par défaut par un argument d'une fonction.

Lors de l'appel à cette fonction, si on ne met pas l'argument, il prendra la valeur par défaut, dont le cas contraire, la valeur par défaut est ignorée.

VI. Les arguments par défaut

Exemple :

```
void f1(int n=3){…}
void f2 (int n, float x=2.3) {…}
void f3(char a, int b=21, float c=5){...}
void main(){
        char a='x'; int i=2; float r=3.2;
        f1(i)  ;  //appel de f1 avec n=2
        f1()  ;  //appel de f1 avec n=3
        f2(i,r)  ;  //appel de f2 avec n=2 et x=3.2
        f2(i)  ;  //appel de f2 avec n=2 et x=2.3
        f3(a,i,r)  ;  //appel de f3 avec …
        f3(a)  ;  //appel de f3 avec …
        f3() ;  //erreur
```


VI. Les arguments par défaut

RQ :

Les arguments dont les valeurs sont définit par défaut doivent obligatoirement situés à la fin de la liste des arguments.

Exemple :

```
void f1 (int x, int n=3) {…}   //ok
void f2 (int n=2, float x) {…}   //erreur
void f3 (char a, int b=2, float c) {...}  //erreur

//correction de f2 et f3
void f2 (float x, int n=2) {…}  //ok
void f3 (char a, float c, int b=2) {...}  //ok
```

VI. Les arguments par défaut

Exercice :

En utilisant la possibilité d'initialisation des arguments offerte par le langage C++, créer la fonction **prod** qui permet de calculer le produit de 2, 3, ou 4 entiers. Valider cette fonction sur quelques exemples.

VII. La sur définition d'une fonction

Le C++ autorise la surcharge de fonctions: la définition de fonctions différentes et portant le même nom à condition de les différencies par le type des arguments.

VII. La sur définition d'une fonction

Exemple:

```
void test (int n=0, float x=2.3) {
        cout<<"fonction 1 avec  n = "<<n<<"et  x= "<<x<<endl ;
}
void test (float x=3.4, int n=5){
        cout<<"fonction 2 avec  n = "<<n<<"et  x= "<<x <<endl;
}
void main(){
        int i=3 ; float n=3.3 ;
        test(i,n) ; //appel f1
        test(n,i) ; //appel f2
        test(i) ; //appel f1
        test(n) ; //appel f2
        test(); // erreur dans ce cas
}
```


VIII. Les opérateurs new et delete

Les 2 opérateurs **new** et **delete** remplacent les fonctions de gestion dynamique de la mémoire malloc et free. Ils permettent donc de réserver et de libérer une place mémoire.

Exemple en C/C++ :

```
int *p ;
long  nb=12;
p=(int *) malloc (nb*sizeof(int));
…
frre (p) ;
```

VIII. Les opérateurs new et delete

Exemple en C++ :

```
int *pi; float *pr ;
pi=new int;  //allocation d'une seule valeur
pr= new float [50];
…
delete pi;
delete pr;
```

RQ: Il ne faut pas utiliser conjoignaient malloc et delete ou new et free

IX. Notion de référence

En C l'operateur **&** désigne l'adresse, en C++ il peut désigner soit l'adresse soit une référence selon le contexte. Seul le contexte de programme permet de déterminer s'il s'agit d'une référence ou d'une adresse.

Exemple :

```
int n=3 ;
int &p=n ; //p et n ont la même @ mémoire
cout<< p ; // affiche 3
```

IX. Notion de référence

IX.1. Passage de paramètres par référence

En C un sous-programme ne peut modifier la valeur d'une variable local passée en argument d'une fonction que si on passe l'adresse de cette variable.

Exemple en C/C++ :

```
//passage par valeur
void permutation( int a, int b){
        int c ;
        c=a ; a=b ; b=c ;}
void main(){
        int x=2 ;int y=3 ;
        permutation(x,y) ;
//après l'exécution le changement ne se fait pas : x=2 et y=3
}
```

17

IX. Notion de référence

IX.1. Passage de paramètres par référence

Exemple en C/C++ :

```
//passage par adresse
void permutation ( int *a, int *b) {
        //*a représente le contenu de a
        int c ;
        c=*a ; *a=*b ; *b=c ;}

void main(){
        int x=2 ;int y=3 ;
        permutation(&x,&y) ;
        // après l'exécution x=3 et y=2
}
```

18

IX. Notion de référence

IX.1. Passage de paramètres par référence

RQ : En C++ on préfère d'utiliser le passage par référence que le passsage par adresse.

Exemple en C++ :

```
void permutation ( int &a, int &b){
        int c ;
        c=a ; a=b ; b=c ;}

void main(){
        int x=2 ;int y=3 ;
        permutation(x,y) ;
        // après l'exécution x=3 et y=2
}
```

19

IX. Notion de référence

IX.1. Passage de paramètres par référence

RQ :

Une référence doit être toujours initialisée :

```
int &p ;//incorrecte
```

On ne peut pas référencer une constante

```
int &p=3 ; //incorrecte
```

On ne peut pas référencer une expression

```
int &p=n+2;// incorrecte
```

Une référence ne doit pas modifier

```
int &p=n ;
p=q ;//correcte ⇔n=q
&p=q ;//incorrecte
```

20

IX. Notion de référence

IX.1. Passage de paramètres par référence

RQ :

Il est possible de définir des références sur une constante en utilisant le mot-clé **const**.

```
const int &p=3 ; //correcte
```

Ici le compilateur génère une variable temporelle qui contient la valeur 3 et affecte à p une référence de cette zone mémoire temporelle. La durée de vie de cette variable temporelle est très courte.

IX. Notion de référence

IX.1. Passage de paramètres par référence

Exercice :

- Créer la fonction void duplicate (int* x) qui permet la duplication de l'argument x.
- Redéfinir cette fonction en utilisant le passage par référence des arguments.
- Écrire la fonction main qui teste les deux fonctions décrites précédemment.

IX. Notion de référence

IX.2. Utilisation d'une référence comme valeur de retour d'une fonction

Le mécanisme de passage par référence peut être appliqué à la valeur de retour d'une fonction

Exemple :

```
int &f()  ; // f retourne une référence sur entier
int *f() ; // f retourne un pointeur sur entier
```


IX. Notion de référence

IX.2. Utilisation d'une référence comme valeur de retour d'une fonction

Exemple 1:

```
int &f ( ) {
        int n ;… retourn n ;}

void main() {
        int p…  p=f ( ) ;
/* cette appelle affecte à p la valeur située à l'emplacement
mémoire référencé par f ( ) */
int x=2 ;
f ( )=2*x+5 ;
/*dons cette instruction, à la référence retourner par f ( ) on va
stocker une valeur.*/
```

IX. Notion de référence

IX.2. Utilisation d'une référence comme valeur de retour d'une fonction

Exemple 2 :

```
int paire= 0 ;
int impaire=1 ;
int &f (int a) {
        return ((a%2==0) ? Paire : impaire);
}
void main() {
f(5)=3;
f(8)=22;
cout<<"paire"<<paire<<endl;
cout<<"impaire"<<impaire<<endl ;
}
```

25

X. Fonction en ligne

Exemple en C/C++ :

```
#include <iostream>
using namespace std;
#define max(x,y) (x<y) ?y: x  //c'est une macro fonction en C/C++
#define CARRE(x) x*x          //c'est une macro fonction en C/C++

void main() {
        int a, b, c;
        a=2; b=3;
        a = max(a, b) ; //ok
        c=max (a++, b++); //effet de bord
        cout<< c ;  // affiche 4, normalement elle affiche 3
        cout<<CARRE(2+4);  //affiche : 14 car calcul de 2+4*2+4
}
```

26

X. Fonction en ligne

Exemple en C/C++ :

```
#include <iostream>
using namespace std;
#define max(x,y) (x<y) ?y: x  //c'est une macro fonction en C/C++
#define CARRE(x) x*x          //c'est une macro fonction en C/C++

void main() {
        int a, b, c;
        a=2; b=3;
        a = max(a, b) ; //ok
        c=max (a++, b++); //effet de bord
        cout<< c ;  // affiche 4, normalement elle affiche 3
        cout<<CARRE(2+4);  //affiche : 14 car calcul de 2+4*2+4
}
```


X. Fonction en ligne

Exemple en C++
//on évite l'effet de bord en ajoutant

```
inline int max( int x, int y ) {
        return (x > y) ? x : y;
}
inline long carre(long ) ;
…
long carre (long t)  {
        return t*t;
}
```

La notion de fonction en ligne a été introduite en C++ pour contrôler les défauts des macros fonctions en C.