Course: C++ Programming

Chapter 1: Syntactic contributions of C++ compared to C

Directed by:
Dr. Sakka Rouis

https://github.com/srtaoufik/coursCpp

1

Chapter 1: Syntactic contributions of C++ compared to C

I.Comment

In C	In C++
/* multi-line comment */	/* comment on several lines */ // single line comment

Example :

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

2

II.Declaring variables

You can declare variables local to a block.
This facility allows for more precise memory management.

Example :

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


III. Lnew I/O possibilities

In C++ you can still use the standard I/O functions (printf and scanf) offered by C by using the <stdio.h> library.
However, there is a second possibility provided by C++ which gives two operators based on the notion of flow.

The definition of these 2 I/O operators is available in the input file <iostream>.

In C/C++	In C++
# include <stdio.h>	# include <iostream>
printf("hello");	cout << "hello" ;
scanf("%d", &x);	cin >> x ;

III. Lnew I/O possibilities

REMARK: Depending on the compiler, sometimes before using cin and cout, you have to write:

```
# include <iostream>
using namespace std ;
…
int x;
cout<<"hello" ;
cin>>x;
```

```
# include <iostream>
…
int x;
std::cout<<"hello" ;
std::cin>>x;
```

Exercise:
Write a C++ program that allows the entry of two integers A and B. Calculate and then display their sum and their product.

IV.Type conversion

C++ allows type conversions between type variables:

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

Examples:

simple conversion	when calling a method
int x=2 ; float y=x ;	void f(int, int); float a=2.2 ; int b=2 ; f (a, b);

V.Default arguments

In C++ you can specify the default value taken by an argument of a function.

When calling this function, if the argument is not set, it will take the default value, otherwise the default value is ignored.

V.Default arguments

Example :

```
void f1(int n=3){…}
void f2 (int n, float m=2.3) {…}
void f3(char a, int b=21, float c=5){...}
void main(){
        char i='x'; int j=2; float k=3.2;
        f1(j) ; //call f1 with n=2
        f1() ; //call f1 with n=3
        f2(j,k) ; //call f2 with n=2 and m=3.2
        f2(j) ; //call f2 with n=2 and m=2.3
        f3(i,j,k) ; //call f3 with …
        f3(i); //call f3 with …
        f3() ; //error
}
```

V.Default arguments

RQ:

Arguments whose values are defined by default must be located at the end of the argument list.

Example :

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

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


V.Default arguments

Exercise :

Using the argument initialization facility offered by the C++ language, create the function **prod** which allows you to calculate the product of 2, 3, or 4 integers.

Validate this function on some examples.

VI.The overdefinition of a function

C++ allows function overloading: defining different functions with the same name as long as they are differentiated by the type of arguments.

VI.The overdefinition of a function

Example:

```
void test (int n=0, float x=2.3) {
cout<<"function 1 with n = "<<n<<"and x= "<<x<<endl ;
}
void test (float x=3.4, int n=5){
cout<<"function 2 with n = "<<n<<"and x= "<<x <<endl;
}
void main(){
int i=3 ; float n=3.3 ;
test(i,n); //call f1
test(n,i); //call f2
test(i); //call f1
test(n); //call f2
test(); // error in this case
}
```

VII. The operators new and delete

The 2 operators **new** and **delete** replace dynamic memory management functions **malloc** and **free** they therefore allow you to reserve and free up memory space.

Example in C/C++:

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


VII. The operators new and delete

Example in C++:

```
int *pi;
float *pr;
pi=new int; //allocation of a single value
pr= new float[50];
…
delete pi;
delete pr;
```

REMARK**:**Do not use malloc and delete or new and free together

VIII. Concept of reference

In C the operator **&** denotes the address, in C++ it can denote either the address or a reference depending on the context.

Only the program context can determine whether it is a reference or an address.

Example :

```
int n=3 ;
int &p=n ; //p and n have the same @ memory
cout<< p ; // displays 3
```


VIII. Concept of reference

VIII.1. Passing parameters by reference

In C, a subroutine can only modify the value of a local variable passed as an argument to a function if the address of this variable is passed.

Example in C/C++:

```
//pass by value
void permutation(int a, int b){
int c ;
c=a ; a=b ; b=c ;
}
void main(){
int x=2 , y=3 ;
permutation(x, y);
cout <<"after permutation x= "<<x<<" and y="<<y ;
//after execution the change is not made: x=2 and y=3
}
```

VIII. Concept ofreference

VIII.1. Passing parameters by reference

Example in C/C++:

```
//pass by address
void permutation (int *a, int *b) {
  // *a represents the contents of the variable pointed to by a
        int c ;
        c=*a ; *a=*b ; *b=c ;
}
void main(){
        int x=2, y=3 ;
        permutation (&x, &y);
        cout <<"after permutation x= "<<x<<" and y="<<y ;
        // after execution x=3 and y=2
}
```


VIII. Concept ofreference

VIII.1. Passing parameters by reference

R:In C++ we prefer to use pass by reference rather than pass by address.

Example in C++:

```
void permutation (int &a, int &b) {
        int c ;
        c=a ; a=b ; b=c ;
}
void main() {
        int x=2, y=3 ;
        permutation(x,y);
        cout <<"after permutation x= "<<x<<" and y="<<y ;
        // after execution x=3 and y=2
}
```

VIII. Concept ofreference

VIII.1. Passing parameters by reference

R:

A reference must always be initialized:

```
int &p ;   //incorrect
```

You cannot reference a constant

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

You cannot reference an expression

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

A reference must not modify

```
int q=3, n=7;
int &p=n ;
p=q ;  //correct⇔n=q=3
&p=q ;//incorrect
```


VIII. Concept ofreference

VIII.1. Passing parameters by reference

RQ:

It is possible to define references to a constant using the keyword **const**.

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

Here the compiler generates a time variable that contains the value 3 and assigns to p a reference to this time memory area.

VIII. Concept ofreference

VIII.1. Passing parameters by reference

Exercise :

•Create the function void duplicate(int *x) which allows the duplication of the argument x.
•Redefine this function using pass-by-reference arguments.
•Write the main function that tests the two functions described previously.

VIII. Concept ofreference

VIII.2. Using a reference as a function's return value

The pass-by-reference mechanism can be applied to the return value of a function

Example :

```
int &f() ; // f returns an integer reference
int *f() ; // f returns a pointer to an integer
```

VIII. Concept ofreference

VIII.2. Using a reference as a function's return value

Example 1:

```
int even = 0;
int odd = 1;
// Function returning a reference
int& f(int a) {
    return (a % 2 == 0) ? even : odd;
}

void main() {
    f(5) = 3;   // modifies 'odd' because 5 is odd
    f(8) = 22;  // modifies 'even' because 8 is even

    cout << "even = " << even << endl;
    cout << "odd = " << odd << endl;
}
```


VIII. Concept ofreference

VIII.2. Using a reference as a function's return value

Example 2:

```
// Function returning a reference to an array element
int& getElement(int arr[], int index) {
    return arr[index];
}
int main() {
    int A[5] = {10, 20, 30, 40, 50};
    cout << "Before modification:" << endl;
    for (int i = 0; i < 5; i++)
        cout << A[i] << " ";
    cout << endl;
    // Direct modification through the returned reference
    getElement(A, 2) = 100;   // modifies A[2]
    getElement(A, 4) = 500;   // modifies A[4]
    cout << "After modification:" << endl;
    for (int i = 0; i < 5; i++)
        cout << A[i] << " ";
    cout << endl;
}
```

IX. Online Function

Example in C/C++:

```
#include <iostream>
using namespace std ;
#define max(x, y) (x>y) ? x: y //this is a macro function in C/C++
#define carre(x) x*x //this is a macro function in C/C++

void main() {
int a, b, c, d;
a=2; b=3;
c = max(a, b); //ok
cout << "c = " << c << endl;// displays c=3
d=max(a++,b++); //side effect
cout<<"d=" <<d ; // displays d=4, normally it displays 3
cout<< "a="<<a = <<"and b= "<< b ; // displays a=3 and b=5
cout<<carre(2+4); //displays: 14 because calculation of 2+4*2+4
}
```


IX. Online Function

//we avoid the side effect by adding the inline keyword

Example in C++

```
inlineint max(int x, int y) { // Method 1
    return (x > y) ? x : y;
}
inlineint square(int ); // Method 2
int square(int x) {
    return x * x;
}
void main() {
    int a, b, c, d;
    a = 2;
    b = 3;
    c = max(a, b); // ok
    cout << "c = " << c << endl; // Displays c = 3
    d = max(a++, b++);
    cout << "d = " << d << endl; // Displays d = 3
    cout << "a = " << a << " and b = " << b << endl; // Displays a = 3 and b = 4
    cout << carre(2 + 4); // Displays: 36
}
```

The concept of inline function was introduced in C++ to control the defects of macro functions in C.