Object Oriented Programming (1)

Prepare

By

Prof Dr. Ahmed Younes

2022 - 2023

Chapter:1

Review on
The Functions

Introduction

The purpose of using functions

- Organization of the program.
- Easy to track the program.
- Easy detection of errors if they occur.
- Reduce writing code sentences

The Types of Functions

There are two types of functions in C++:

- **Built In Functions**
- **User Defined Functions**

In this chapter, we show the following:

1. **How to prototype a user defined functions.**
2. **How to use these functions**

Function Prototype

Int function name (int)

- **The type of int In parentheses tells that the argument to be passed to the function will be of the type int. the int preceding the function name indicating the type of value returned by the function is int.**

Function Definition

```
return-value-type function-name (parameter list)
{
    declarations and statements
}
```

return-value-type: The type of value returned by the function which can be any type of C++ data

function-name: The name of the function, which follows the rules for naming identifiers

Parameter List: is the list of arguments passing to the function and it can be void:

Function Call

- **The function is called by its name with passing the arguments(if the function receive arguments).**
- **The function can return values to the statement that called it.**
- **It must precede the name of the function its identifier, and if the function returns nothing, the void keyword must be used to indicate that**

Example

```
#include<iostream.h>
int square(int);  //function prototype
main()
{
  for(int x=1;x<=10;x++)
  cout<<square(x)<<" ";
  cout<<endl;
}
```

Example

```
//now function definition
int square(int y)
{
    return y*y;
}
```

The Output:

```
1 4 9 16 25 36 49 64 81 100
```

Example

- The square(x) function is called inside the main function by writing square.

- The result is returned to the main function, where the square function is called, where the result is displayed, and this process is repeated ten times using the for loop

Arguments

- **Arguments are the mechanism used to pass information from a function call to the function itselfSince data is copied and values are stored in separate variables in the function, these variables are named in the function definition**

Example:2

```
#include <iostream.h>
int maximum (int, int, int);
main( )
{
  int a, b, c;
  cout << "Enter three integers: " ;
  cin >> a >> b >> c ;
  cout << " maximum is : " << maximum (a, b, c) <<
  endl;
  return 0;
}
```

Example:2

```
int maximum (int x, int y, int z)
{
  int max = x;
  if (y > x)
  max = y;
  if (z > max)
  max = z;
  return max;
}
```

Example:2

Outputs of the program:

Assuming the user has entered the numbers 22, 85, 17

```
Enter three integers: 22 85 17

Maximum is: 85
```

Notice

- In the function declaration, it is not necessary to define the name of arguments as:

  Void fffff (int, int).

- But it the best that declare the name of tat arguments:

  Void fffff (int x, int y).

- If the programmer not define the type of function return in the part of function declaration, the default type of this function is int

Functions with Empty arguments

- In C++, the function of empty arguments is written as:

```
Void ffff ( );
OR
Void ffff (void);
```

Example:

```
// Functions that take no arguments
#include <iostream.h>
void f1 ( );
void f2 (void);
main( )
{
f1 ( );
f2 ( );
return 0;
}
```

Example:

```
void f1 ( )
{
   cout << "Function f1 takes no arguments" << endl;
}
void f2 (void)
{
   cout << "Function f2 also takes no arguments" << endl;
}
```

The output:

```
Function f1 takes no arguments
Function f2 also takes no arguments
```

Inline Functions

The function is inline by using the keyword inline

```
inline void func1( )
{
   statements
}
```

The inline function is used when the operations of it function is short.

Example:1

```
#include<iostream.h>
inline float cube(float s){return s*s*s;}
main()
{
   cout<<"\nEnter the side length of your cube : ";
   float side;
   cin>>side;
   cout<<"volume of cube is "
   <<cube(side)
   <<endl;
}
```

Exapmple:1

- **The output:**

Enter the side length of your cube : 5

volume of cube is 125

Example:2

```
#include <iostream.h>
inline int mult( int a, int b){return (a*b);}
main( )
{
   int x, y, z;
   cin >> x >> y >> z;
   cout << "x = " << x << " y = " << y << " z = " << z <<
   endl;
   cout << "product1" << mult (x ,y) << endl;
   cout << "product2" << mult (x +2, y) << endl;
   return 0;
}
```

Example:2

- The Output: if input x=3, y=4, z=5

  x = 3 y =4 z = 5

  product1 12

  product2 32

Default Arguments

Default arguments allow one or more arguments to be ignored when the function is called, and when present Missing Argument The function declaration supplies constant values for those missing arguments.

Example

```
#include <iostream.h>
inline box(int length=1,int width=1,int height=1)
{return length*width*height;}
main()
{
  cout<<"The box volume is "<<box()<<endl
  <<"width 1 and height 1 is"<<box(10)<<endl;
  return 0;
}
```

Example

The output

The box volume is 1

Width 1 and height1 is 10

Overloading Functions

- **Function Overloading is defined as the process of having two or more function with the same name, but different in parameters is known as function overloading in C++. In function overloading, the function is redefined by using either different types of arguments or a different number of arguments. It is only through these differences compiler can differentiate between the functions.**

Overloading Functions

- **Let's see the simple example of function overloading where we are changing number of arguments of square() method.**

Example:1

```
#include <iostream.h>
int square(int x){return x*x;}
double square(double y){return y*y;}
main ()
{
  cout<< " The square of integer 7 is"
  <<" "<<square(7)<< endl
  <<"The square of double 7.5 is"
  <<" "<<square(7.5)<< endl;    return 0;
}
```

Overloading Functions

The Outputs:

The square of integer 7 is 49

The square of double 7.5 is 56.25

Example:2

```
#include<iostream.h>
int mul(int,int);
float mul(float,int);
int main()
{
    int r1 = mul(6,7);
    float r2 = mul(0.2,3);
    cout << "r1 is : " <<r1<< endl;
    cout <<"r2 is : "  <<r2<< endl;
    return 0;
}
```

Example:2

```
int mul(int a,int b)
{
    return a*b;
}
float mul(double x, int y)
{
    return x*y;
}
```

Example:2

The Output:

r1 is : 42

r2 is : 0.6

Chapter:2

Review on Arrays

Array

- Array
  - Consecutive group of memory locations
  - Same name and type
- To refer to an element, specify
  - Array name and position number
- Format: *arrayname[position number]*
  - First element at position 0
  - n element array c:

    c[0], c[1]…c[n - 1]

Array

- **Array elements are like normal variables**

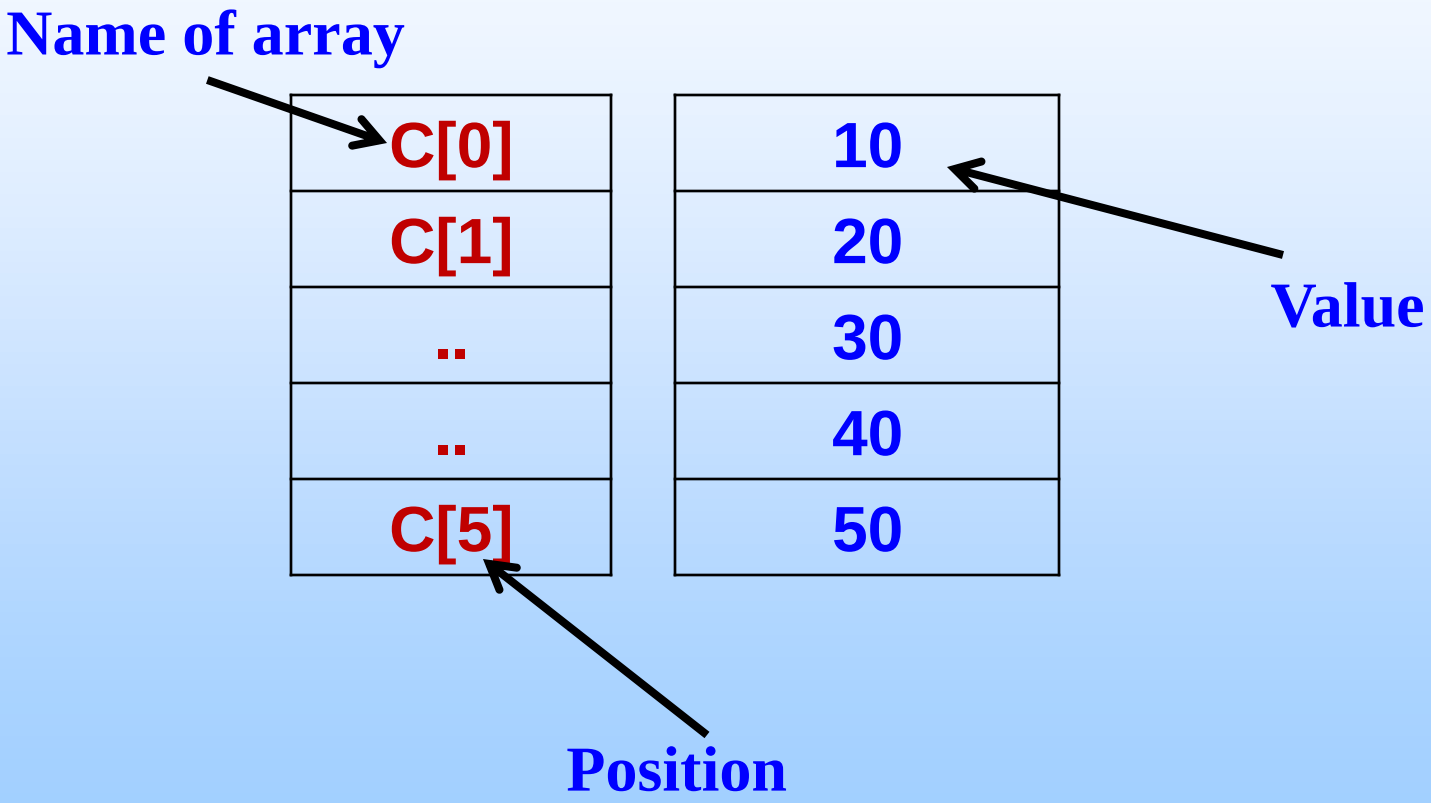
```
c[ 0 ] =  3;

cout << c[ 0 ];
```

- **Performing operations in subscript. If x = 3,**

```
c[ 5 – 2 ] == c[ 3 ] == c[ x ]
```

Array



Declaring Arrays

- Declaring arrays - specify:
  - Name
  - Type of array
  - Number of elements
  - Examples

```
int c[ 10 ];
float hi[ 3284 ];
```

- Declaring multiple arrays of same type
  - Similar format as other variables
  - Example

```
int b[ 100 ], x[ 27 ];
```

Initializing an Array

```
int n[ 5 ] = { 1, 2, 3, 4, 5 };
```

If not enough initializers, rightmost elements become 0.

```
int n[ 5 ] = { 0 }
```

Sets all the elements to 0.

```
int n[ ] = { 1, 2, 3, 4, 5 };
```

5 initializers, therefore n is a 5 element array

Example:1

```
//initializing an array with a declaration
#include <iostream.h>
#include <iomanip.h>
main( )
{
int n[10];
for (int i=0; i<10; i++) // initialize array
n[i] = 0;
cout << “Element” << setw(13) << “ value” << endl;
for (i=0 ; i< 10; i++) // print array
cout << setw(7) <<i<<setw(13) <<n[i]<<endl;
return 0;
}
```

Example:1

The output

Element	Value
	0
	0
	0
	0
	0
	0
	0
	0
	0

Example:2

```
// compute the sum of the elements of the array
#include <iostream.h>
main( ){
const int arraysize =12;
int a[arraysize] = {1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45};
int total = 0;
for (int i= 0; i<arraysize ; i++)
total += a[i];
cout <<" total of array element values is " << total << endl;
return 0;}
```

Example:2

- **The Output**

total of array element values is 383

Arrays and Functions

- **Passing Arrays to Function As Function Parameters:**

Function Syntax:

```
ReturnType function Name(data Type array Name[arraySize])
{
    // code
}
```

Call Function

```
functionName(arrayName)
```

Arrays and Functions

- **Example:**

Function Syntax:

```
int total(int m[5])
 {
    // code
  }
```

Call Function

```
Total(marks)
```

Example:

```
// C++ Program to display marks of 5 students
#include <iostream.h>
void display(int x[5])
  { cout << "Displaying marks: " << endl;
    for (int i = 0; i < 5; ++i)
    { cout << "Student " << i + 1 << ": " << x[i] << endl; }}
int main() {
    int marks[5] = {88, 76, 90, 61, 69};
    display(marks);
    return 0; }
```

Example:

- **The output:**

Displaying marks:

Student 1: 88

Student 2: 76

Student 3: 90

Student 4: 61

Student 5: 69

Arrays and Functions

- **Passing Multidimensional Array to a Function**
- **An array can be multidimensional and each dimension can be of a different size.**
- **we only pass the name of the two dimensional array as the function argument.**
- **It is not mandatory to specify the number of rows in the array. However, the number of columns should always be specified.**

Example:

```
// initializing multidimensional arrays
#include<iostream.h>
void printarray(int [ ] [3]);
int main( )
{
   int array1[2] [3] = { {1, 2, 3}, {4, 5, 6}},
   array2[2] [3] = {1, 2, 3, 4, 5},
   array3[2] [3] = { {1, 2}, {4} };
   cout << “values in array1 by row are : “ <<
   endl;
```

Example:

```
  printArray(array1);
  cout << “values in array2 by row are : “ <<
  endl;
  printArray(array2);
  cout << “values in array3 by row are : “ <<
  endl;
  printArray(array3);
  return 0;
}
```

Example:

```
 void printArray(int a[ ][3])
 {
   for (int i=0; i<1; i++) {
   for (int j=0; j<2; j++)
   cout << a[i][j] <<’ ‘ << endl;
 }
}
```

Example:

The Output:

values in array 1 by row are:

1 2 3

4 5 6

values in array 2 by row are:

1 2 3

4 5 0

values in array 3 by row are:

1 2 0

4 0 0