جامعة
القاهرة الجديدة
التكنولوجية

NEW CAIRO
TECHNOLOGICAL
UNIVERSITY

TUE – The Technological Universities in Egypt

NCTU – New Cairo Technological University

Faculty of Industry and Energy Technology

Information Technology Department

Second-Year

Course: Programming Essentials in C++

Lecture 6

Presented by

Dr. Ghada Maher

Contents:

- C++ Functions
  - Defining a Function
  - Function with value and parameters
  - Default values in parameters
  - Function Declarations
  - Functions with no type. The use of void

C++ Functions

- A function is a group of statements that together perform a task. Every C++ program has at least one function, which is main(), and all the most trivial programs can define additional functions.

- You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division usually is such that each function performs a specific task.

- A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

Defining a Function

The general form of a C++ function definition is as follows:

```
type name ( parameter1, parameter2, ...)
{ statements }
```

Where:

- type is the type of the value returned by the function.
- name is the identifier by which the function can be called.
- parameters (as many as needed): Each parameter consists of a type followed by an identifier, with each parameter being separated from the next by a comma. Each parameter looks very much like a regular variable declaration (for example: int x), and in fact acts within the function as a regular variable which is local to the function. The purpose of parameters is to allow passing arguments to the function from the location where it is called from.
- statements is the function's body. It is a block of statements surrounded by braces { } that specify what the function actually does.

Example 1: Function with value


```
1  // function example
2  #include <iostream>
3  using namespace std;
4
5  int addition (int a, int b)
6  {
7    int r;
8    r=a+b;
9    return r;
10 }
11
12 int main ()
13 {
14   int z;
15   z = addition (5,3);
16   cout << "The result is " << z;
17 }
```

```
The result is 8
```

Example 2: Function with values and parameters

A function can actually be called multiple times within a program, and its argument is naturally not limited just to literals:

```
1  // function example
2  #include <iostream>
3  using namespace std;
4
5  int subtraction (int a, int b)
6  {
7    int r;
8    r=a-b;
9    return r;
10 }
11
12 int main ()
13 {
14   int x=5, y=3, z;
15   z = subtraction (7,2);
16   cout << "The first result is " << z << '\n';
17   cout << "The second result is " << subtraction (7,2) << '\n';
18   cout << "The third result is " << subtraction (x,y) << '\n';
19   z= 4 + subtraction (x,y);
20   cout << "The fourth result is " << z << '\n';
21 }
```

```
The first result is 5
The second result is 5
The third result is 2
The fourth result is 6
```

Default values in parameters

In C++, functions can also have optional parameters, for which no arguments are required in the call, in such a way that, for example, a function with three parameters may be called with only two. For this, the function shall include a default value for its last parameter, which is used by the function when called with fewer arguments. For example:

Example 3:


```
1  // default values in functions
2  #include <iostream>
3  using namespace std;
4
5  int divide (int a, int b=2)
6  {
7    int r;
8    r=a/b;
9    return (r);
10 }
11
12 int main ()
13 {
14   cout << divide (12) << '\n';
15   cout << divide (20,4) << '\n';
16   return 0;
17 }
```

```
6
5
```

Function Declarations

The prototype of a function can be declared without actually defining the function completely, giving just enough details to allow the types involved in a function call to be known. Naturally, the function shall be defined somewhere else, like later in the code. But at least, once declared like this, it can already be called.

The declaration shall include all types involved (the return type and the type of its arguments), using the same syntax as used in the definition of the function, but replacing the body of the function (the block of statements) with an ending semicolon.

The parameter list does not need to include the parameter names, but only their types. Parameter names can nevertheless be specified, but they are optional, and do not need to necessarily match those in the function definition. For example, a function called proto function with two int parameters can be declared with either of these statements:

```
1 int protofunction (int first, int second);
2 int protofunction (int, int);
```

Example 4:


```
1  // declaring functions prototypes
2  #include <iostream>
3  using namespace std;
4
5  void odd (int x);
6  void even (int x);
7
8  int main()
9  {
10   int i;
11   do {
12     cout << "Please, enter number (0 to exit): ";
13     cin >> i;
14     odd (i);
15   } while (i!=0);
16   return 0;
17 }
18
19 void odd (int x)
20 {
21   if ((x%2)!=0) cout << "It is odd.\n";
22   else even (x);
23 }
24
25 void even (int x)
26 {
27   if ((x%2)==0) cout << "It is even.\n";
28   else odd (x);
29 }
```

```
Please, enter number (0 to exit): 9
It is odd.
Please, enter number (0 to exit): 6
It is even.
Please, enter number (0 to exit): 1030
It is even.
Please, enter number (0 to exit): 0
It is even.
```

Functions with no type. The use of void

The syntax shown above for functions:

type name (argument1, argument2 ...) { statements }

Example 5:

```
1  // void function example
2  #include <iostream>
3  using namespace std;
4
5  void printmessage ()
6  {
7    cout << "I'm a function!";
8  }
9
10 int main ()
11 {
12   printmessage ();
13 }
```

```
I'm a function!
```

Example 6:

```
#include <iostream>

using namespace std;

int addition (int x, int y)

{ int r;

  r=a+b;

 a++;

 b=b+3

  return r;}

int main ()

{ int x=5;

int y=3;

 int z;

  z = addition (x,y);

 cout << "The result is " << z;

cout << "The result is " << x;

cout << "The result is " << y;

Return 0; }
```

1. Write a C++ program with a function (fact) that allows the user several times to enter a number and return the factorial of this number.
2. Write a function (maximum) that return the maximum of three float numbers.

Recursivity

Recursivity is the property that functions have to be called by themselves. It is useful for some tasks, such as sorting elements, or calculating the factorial of numbers. For example, in order to obtain the factorial of a number (n!) the mathematical formula would be:

`n! = n * (n-1) * (n-2) * (n-3) ... * 1`
More concretely, `5!` (factorial of 5) would be:

`5! = 5 * 4 * 3 * 2 * 1 = 120`
And a recursive function to calculate this in C++ could be:

```cpp
// factorial calculator
#include <iostream>
using namespace std;

long factorial (long a)
{
  if (a > 1)
   return (a * factorial (a-1));
  else
   return 1;
}

int main ()
{
  long number = 9;
  cout << number << "! = " << factorial (number);
  return 0;
}
```

```
9! = 362880
```

Ec
&
Ru

Notice how in function factorial we included a call to itself, but only if the argument passed was greater than 1, since, otherwise, the function would perform an infinite recursive loop, in which once it arrived to 0, it would continue multiplying by all the negative numbers (probably provoking a stack overflow at some point during runtime).