University of Mosul

College of Computer Sciences and Mathematics

Department of Artificial Intelligence

Algorithms and Structured Programming (2)

First stage

Lecture 2

أ.م. بيداء سليمان بهنام

Assist.Prof. Baydaa Sulaiman Bahnam

Lecture 1 Outline

1. Variable Scope: Local vs Global VariablesFunctions
2. Default Parameter Value
3. Parameter Passing Techniques in C++
4. Pass by Value
5. Pass by Reference
6. Standard Library Functions (C++ Library Functions)

Variable Scope : Local vs Global Variables

نطاق المتغير: المتغيرات المحلية والعالمية

A scope is a region of the program and broadly speaking there are three places, where variables can be declared :

- Inside a function or a block which is called **local variables,**
- In the definition of function parameters which is called formal parameters.
- Outside of all functions which are called **global variables**.

Local variables can be used only by statements that are inside that function or block of code. Local variables are not known to functions on their own.

Example

```
#include <iostream>
using namespace std;
  int main ()
{
   // Local variable declaration:
   int a, b;
   int c;
   // actual initialization
   a = 10;
   b = 20;
   c = a + b;

   cout << c;
   return 0;
}
```

Output

This will give the output −

```
30
```

Variable Scope : Local vs Global Variables

Global variables are defined outside of all the functions, usually on top of the program. The global variables will hold their value throughout the lifetime of your program. A global variable can be accessed by any function.

Example

```
#include <iostream>
 using namespace std;

// Global variable declaration:
 int g;

int main ()
 {
   // Local variable declaration:
    int a, b;
// actual initialization
   a = 10;
   b = 20;
   g = a + b;
cout << g;
return 0;
}
```

Output

This will give the output −

```
30
```

Variable Scope : Local vs Global Variables

A program can have the same name for local and global variables but the value of a local variable inside a function will take preference. For accessing the global variable with same name, you'll have to use the **scope resolution operator (::)**.

Example

```
#include <iostream>
using namespace std;

// Global variable declaration:
int g = 20;

int main ()
{
   // Local variable declaration:
   int g = 10;
   cout << g; // Local
   cout << ::g; // Global
   return 0;
   }
```

Output

This will give the output −

```
10
20
```

متى نستخدم :: ومتى لانستخدمها في دوال البرنامج اوالبرنامج الرئيسي

- ✓ إذا كانت اسماء المتغيرات المحلية تختلف عن اسماء المتغيرات العامة فسوف نستخدم المتغيرات العامة مباشرة ولانحتاج الى استخدام :: قبل اسم المتغير العام.
- ✓ إذا كان هناك متغير محلي بنفس اسم المتغير العام فيجب استخدم :: قبل اسم المتغير العاممن اجل للوصول إلى ذلك المتغير العام.

Variable Scope : Local vs Global Variables

```
#include <iostream>
using namespace std;
int x = 8000; // Global variable x

// Define the function Fun1
void Fun1()
{
    int x = 500; // Local variable x in function Fun1
    cout << "The value of x inside the function is: " << x << endl;
    cout << "The global value of x inside the function is: " << ::x << endl; //
Access global x using scope resolution operator
}

int main()
{
    int x = 100; // Local variable x in main
    cout << "The local value of x inside Main is: " << x << endl;
    cout << "The global value of x inside Main is: " << ::x << endl; // Access
global x using scope resolution operator
    // Call the Fun1 function
    Fun1();

    ++::x;
    cout << "The global value of x inside Main is: " << ::x << endl; // Access
global x using scope resolution operator
    return 0;
}
```

ملاحظة إذا كان هناك متغير محلي داخل دالة main() (او داخل اي دالة) ومتغير محلي بنفس الاسم داخل دالة أخرى ، فكل دالة ستتعامل مع نسختها الخاصة من المتغير، لأن المتغيرات المحلية تكون محصورة داخل الدالة التي تم تعريفها تلك المتغيرات فيها ولا تؤثر على المتغيرات خارجها.

عندما تنتهي عمل الدالة تمحى قيمة المتغير x من الذاكرة

```
The local value of x inside Main is: 100
The global value of x inside Main is: 8000
The value of x inside the function is: 500
The global value of x inside the function is: 8000
The global value of x inside Main is: 8001
```

Write a C++ program that uses a function to calculate the square of numbers from 1 to 10.

```cpp
#include <iostream>
using namespace std;
int square(int y) // function named square with type integer
{
    int z;
    z = y * y;
    return (z);
}

int main()
{
    for (int x = 1; x <= 10; x++)
    {
        cout << square(x) << endl;
    }
    return 0;
}
```

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

return y * y;

Write a C++ program that uses a function to calculate the average of two numbers entered by the user in the main program.

```
#include <iostream>
using namespace std;
// Function named square with type float
float aver(int x1, int x2)
{
    float z;
    z = (x1 + x2) / 2.0;
    return (z);
}

int main()
{
    float x;
    int num1, num2;

    cout << "Enter 2 positive numbers: \n";
    cin >> num1 >> num2;
    if (num1 < 0 || num2 < 0)
    {
        cout << "Both numbers must be positive." << endl;
        return 1; // إرجاع قيمة خطأ إذا لم تكن الأرقام موجبة
    }
    x = aver(num1, num2);
    cout << "The average is: " << x << endl;
    return 0;
}
```

```
return (x1 + x2) / 2.0;
```

```
Enter 2 positive numbers:
5
7
The average is: 6
```

```
Enter 2 positive numbers:
-8
10
Both numbers must be positive.
```

Write a C++ program that uses a function to calculate the summation of the following series:

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + \cdots + n^2$$

```
#include <iostream>
using namespace std;

// Function to calculate the summation of squares from 1 to x
int summation(int x)
{
    int sum = 0;
    for (int i = 1; i <= x; i++)
    {
        sum += i * i;//  sum = sum + i * i;
    }
    return (sum);
}
```

```
int i=1, sum=0;
while (i<=x)
{
 sum += pow(i,2) ;
 i++;
 }return (sum);}
```

```
int main()
{
    int n, s;
    cout << "Enter positive number = ";
    cin >> n;
    s = summation(n);
    cout << "sum is : " << s << endl;
    return 0;
}
```

```
Enter positive number = 3
sum is : 14
```

Ex: Write a program includes a function receives a character and returns its type (number, lowercase alphabet, uppercase alphabet, or symbol)

```
#include <iostream>
using namespace std;
int chaletter( char c)
{
 if(c >= 'A' && c <= 'Z')
 return (1);
 else if(c >= 'a' && c <= 'z')
 return (2);
 else if (c>='0' && c<='9')
 return (0);
 else return(3);
}
main()
{
int type;
 char ch;
 cout << "Enter the character:"<<endl;
 cin >> ch;
 type = chaletter(ch );
 switch(type)
 {
 case 0: cout << "It is a number"; break;
 case 1: cout << "It is an uppercase alphabet"; break;
 case 2: cout << "It is a lowercase alphabet"; break;
 default: cout<<" It is a symbol";
 }
 return 0;
}
```

Default Parameter Value

You can also use a default parameter value, by using the equals sign (=).

If we call the function without an argument, it uses the default value such as ("Iraq"):

```
#include <iostream>
using namespace std;

// Function definition with a default parameter
void greet(string country = "Iraq")
{
    cout << "Hello from " << country << "!" << endl;
}

int main()
{
    greet();          // No argument passed → the default value " Iraq " will be used
    greet("Japan");// Argument passed → "Japan" will be used instead of the default value
    greet();          // No argument passed → the default value " Iraq " will be used
    greet("Turkey");  // Argument passed → "Turkey" will be used instead of the default value
    return 0;
}
```

```
Hello from Iraq!
Hello from Japan!
Hello from Iraq!
Hello from Turkey!
```

Default Parameter Value

You can also use a default parameter value, by using the equals sign (=).

```
#include <iostream>
using namespace std;
// Function definition with a default parameter for 'y'
void example(int x, int y = 20)
{
    cout << "x = " << x << ", y = " << y << endl;
}

int main()
{
    example(5);     // Uses the default value for y (which is 20)

    example(10, 30);// Uses the provided value for y (which is 30)
    return 0;
}
```

عند استدعاء example(5); لم يتم تمرير قيمة لـ y ، لذا استخدمت القيمة الافتراضية 20 الموجودة بالدالة

```
x = 5, y = 20
x = 10, y = 30
```

The result when uses the default value for y (which is 20)

1. القيم الافتراضية يجب تحديدها **فقط في التصريح (اي الاعلان)** وليس في التعريف عند الفصل بينهما.
2. يجب أن تكون المعاملات الافتراضية **في نهاية قائمة المعاملات**، ولا يمكن وضع معامل غير افتراضي بعد معامل افتراضي.

Parameter Passing Techniques in C++

In C++, data must be sent to functions when they are called in order to perform operations. This data is called parameters or arguments and there are various parameter passing methods available in C++, each with merits and demerits of its own.

Basic Terminologies

- **Function Parameters:** These variables, which indicate the data that the function anticipates receiving when called, are specified in the parameter list of a function.
- **Actual Parameters:** The expressions or values passed in during a function call. When the function is called, it receives these values as input.
- The parameters that the function signature declares. They serve as stand-ins for the values that are provided in when the function is called.

Parameter Passing Techniques in C++

There are 3 different methods using which we can pass parameters to a function in C++. These are:

1. **Pass by Value (call by value)**
2. **Pass by Reference (call by Reference)**
3. **Pass by Pointers (call by Pointers)**

Pass by Value

In Pass by Value method, the actual value that is passed as argument is not changed after performing some operation on it. This means the variable's actual value is copied and then passed to the function instead of the original variable, it creates a copy of that variable into the stack section in memory. W. As the result, any changes to the parameter inside the function will not affect the variable's original value outside the function. Although it is easy to understand and implement, this method is not so useful for large size of data structures at it involves copying the value.

التمرير بالقيمة (Pass by Value) يتم انشاء نسخة دائما : حيث يتم تمرير المتغيرات إلى الدالة اي انشاء نسخة جديدة من هذه المتغيرات مما يعني أن أي تغيير لهذه المتغيرات يحدث داخل الدالة يؤثر فقط على متغيراتها و لن يؤثر على المتغيرات الأصلية في البرنامج الرئيسي .

لاحظ جميع البرامج السابقة كانت من هذا النوع

Example of Pass by Value (call by value)

Write a c++ function named swap to swap two integer numbers. Read these numbers in main program and call the function to do swap.

نلاحظ هنا المتغيرات يتم تمريرها بالقيمة **(Pass by Value)** إلى الدالة **swap(int x, int y)**
هذا يعني أن أي تغيير يحدث على هذه المتغيرات داخل الدالة لن يؤثر على المتغيرات الأصلية في **main()**،
حيث يتم إنشاء نسخ جديدة من المتغيرات عند تمريرها إلى الدالة حتى لو كانت الأسماء مختلفة أو متشابهة.
نلاحظ هنا اسماء المتغيرات مختلفة.

```
#include <iostream>
using namespace std;

// Function to swap two integers using a temporary variable
void swap(int x, int y)
{
    cout << "Before swapping in function: x = " << x << ", y = " << y << endl;

    int temp = x;  // Temporary variable to hold the value of a
    x = y;         // Assign the value of b to a
    y = temp;      // Assign the value of temp (original a) to b
    cout << "After swapping in function: x = " << x << ", y = " << y << endl;
}

int main()
{
    int a, b;
    // Input values for a and b
    cout << "Enter the value of a: ";
    cin >> a;
    cout << "Enter the value of b: ";
    cin >> b;
    cout << "The variable in main program before call swap function : a = " << a << ", b = " << b <<
endl;

    // Call the swap function
    swap(a, b);
    cout << "The variable in main program after call swap function : a = " << a << ", b = " << b <<
endl;

    return 0;
}
```

```
Enter the value of a: 3
Enter the value of b: 7
The variable in main program before call swap function : a = 3, b = 7
Before swapping in function: x = 3, y = 7
After swapping in function: x = 7, y = 3
The variable in main program after call swap function : a = 3, b = 7
```

Example of Pass by Value (call by value)

Write a c++ function named swap to swap two integer numbers. Read these numbers in main program and call the function to do swap.

نلاحظ هنا المتغيرات يتم تمريرها بالقيمة (Pass by Value) إلى الدالة swap(int x, int y)
هذا يعني أن أي تغيير يحدث على هذه المتغيرات داخل الدالة لن يؤثر على المتغيرات الأصلية في main()،
حيث يتم إنشاء نسخ جديدة من المتغيرات عند تمريرها إلى الدالة حتى لو كانت الأسماء مختلفة أو متشابهة.
نلاحظ هنا اسماء المتغيرات متشابهة.

```
#include <iostream>
using namespace std;

// Function to swap two integers using a temporary variable
void swap(int a, int b)
{
    cout << "Before swapping in function: a = " << a << ", b = " << b << endl;

    int temp = a;  // Temporary variable to hold the value of a
    a = b;         // Assign the value of b to a
    b = temp;      // Assign the value of temp (original a) to b
    cout << "After swapping in function: a = " << a << ", b = " << b << endl;
}

int main()
{
    int a, b;
    // Input values for a and b
    cout << "Enter the value of a: ";
    cin >> a;
    cout << "Enter the value of b: ";
    cin >> b;
    cout << "The variable in main program before call swap function : a = " << a << ", b = " << b
<< endl;

    // Call the swap function
    swap(a, b);
    cout << "The variable in main program after call swap function : a = " << a << ", b = " << b <<
endl;

    return 0;
}
```

```
Enter the value of a: 5
Enter the value of b: 10
The variable in main program before call swap function : a = 5, b = 10
Before swapping in function: a = 5, b = 10
After swapping in function: a = 10, b = 5
The variable in main program after call swap function : a = 5, b = 10
```

Reference

When a variable is declared as a reference, it becomes an alternative name for an existing variable. A variable can be declared as a reference by putting ‘&’ in the declaration. Also, we can define a reference variable as a type of variable that can act as a reference to another variable. ‘&’ is used for signifying the address of a variable or any memory. Variables associated with reference variables can be accessed either by its name or by the reference variable associated with it.

عندما يتم إعلان متغير كمرجع، فإنه يصبح اسمًا بديلًا لمتغير موجود. يمكن إعلان متغير كمرجع عن طريق وضع "&" في الإعلان.كما يمكننا تعريف متغير مرجعي كنوع من المتغيرات التي يمكن أن تعمل كمرجع لمتغير آخر. يتم استخدام "&" للإشارة إلى عنوان متغير أو أي ذاكرة. يمكن الوصول إلى المتغيرات المرتبطة بمتغيرات مرجعية إما عن طريق اسمها أو عن طريق متغير المرجع المرتبط بها.

```cpp
#include <iostream>
using namespace std;

int main()
{
    int a = 10; // Declare an integer variable 'a' and assign it the value 10
    int &b = a; // 'b' is a reference to 'a'

    cout << "a = " << a << endl; // Print the value of 'a' (10)
    cout << "b = " << b << endl; // Print the value of 'b' (also 10, since 'b' refers to 'a')

    b = 20; // Modify 'b', which also changes 'a' since 'b' is a reference

    cout << "After modification-1:" << endl;
    cout << "a = " << a << endl; // Print the updated value of 'a' (20)
    cout << "b = " << b << endl; // Print the updated value of 'b' (20)

    a = 30; // Modify 'a', which also changes 'b' since 'a' is a reference
    cout << "After modification-2:" << endl;
    cout << "a = " << a << endl; // Print the updated value of 'a' (30)
    cout << "b = " << b << endl; // Print the updated value of 'b' (30)
    return 0;
}
```

```
a = 10
b = 10
After modification-1:
a = 20
b = 20
After modification-2:
a = 30
b = 30
```

Reference

```
//  demonstrate the use of references
#include <iostream>
using namespace std;

int main()
{
    int x = 10;
    // ref is a reference to x.
    int &ref = x;

    ref = 20; //The value of x is now changed to 20
    cout << "x = " << x << '\n';

    x = 30;// The value of x is now changed to 30
    cout << "ref = " << ref << '\n';
    return 0;
}
```

```
x = 20
ref = 30
```

Pass by reference (Call by reference)

Write a C++ program that swaps the values of two variables using call by reference and prints the values before and after swapping

```
#include <iostream>
using namespace std;
// Function to swap two numbers using pass-by-reference
void swap(int &x, int &y)
{
    int temp = x;  // Store x in temp
    x = y;         // Assign y to x
    y = temp;      // Assign temp (original x) to y
}
int main()
{
    int a, b;
    // Taking input from user
    cout << "Enter the value of a: ";
    cin >> a;
    cout << "Enter the value of b: ";
    cin >> b;
    // Print values before swapping
    cout << "\nBefore swapping: a = " << a << ", b = " << b << endl;

    // Call the swap function
    swap(a, b);

    // Print values after swapping
    cout << "After swapping: a = " << a << ", b = " << b << endl;
    return 0;
}
```

التمرير بالمرجع **(Pass by Reference)** لا يتم إنشاء نسخة حيث عند استخدام التمرير بالمرجع **void swap(int &x, int &y)**، لا يتم إنشاء نسخ جديدة، بل يتم التعامل مع المتغيرات الأصلية مباشرةً.

عند استدعاء الدالة:**swap(a, b)** ، فإن القيم الأصلية للمتغيرين **a** و **b** في **main()** سيتم تعديلها، وليس مجرد نسخها إلى **x** و **y** كما يحدث في التمرير بالقيمة **(Pass by Value)**

في كلا الحالتين سواء باختلاف او تشابه أسماء المتغيرات (في البرنامج الرئيسي والدالة)، لم يتم إنشاء نسخة، بل يتم تعديل **a** و **b** مباشرةً.(اي تعديل متغيرات البرنامج الرئيسي او الدالة التي تستدعي الدالة الاخرى)

```
Enter the value of a: 33
Enter the value of b: 44

Before swapping: a = 33, b = 44
After swapping: a = 44, b = 33
```

Pass by reference (Call by reference)

Using `&` (Pass by Reference):

- The function `swap(int &x, int &y)` takes references to `x` and `y` instead of copies.
- This ensures that changes made inside the function **affect the original variables** in `main()`.

ماذا يحدث عند تنفيذ swap(a, b); ؟

1.تمرير المتغيرات بالمرجع(&x, &y)

• x و y في swap(int &x, int &y) ليسا نسخًا جديدة، بل هما مجرد **أسماء مستعارة (References)** للمتغيرات الأصلية a و b في main()

2.عند تنفيذ جملة x = y; داخل swap() ، يتم تعديل a أيضًا في البرنامج الرئيسي

•لأن x هو مجرد مرجع لـ a ، وأي تغيير في x هو في الحقيقة تغيير في a

•وكذلك y هو مرجع لـ b ، لذا أي تغيير في y هو تغيير في b

Difference between call by value and call by reference in C++

No.	Call by value	Call by reference
1	A copy of value is passed to the function	An address of value is passed to the function
2	Changes made inside the function is not reflected on other functions	Changes made inside the function is reflected outside the function also
3	Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in same memory location

Functions

1.Standard Library Functions (C++ Library Functions)

Library functions are the built-in functions in C++ programming.

Programmers can use library functions by invoking the functions directly; they don't need to write the functions themselves.

Some common library functions in C++ are sqrt(), abs(), round(), etc.

In order to use library functions, we usually need to include the header file in which these library functions are defined.

For instance, in order to use mathematical functions such as sqrt(), round() and abs(), we need to include the header file cmath.

Examples of Standard Library Functions in C++

The C++ <cmath> header file declares a set of functions to perform mathematical operations.

C++ ceil()

Return ceiling value of number. The ceil() function in C++ returns the smallest possible integer value which is **greater** than or **equal** to the given argument.

`ceil() Parameters:` The ceil() function takes a single argument whose ceiling value is computed.

الدالة **ceil()** في **C++** تُستخدم لإرجاع أصغر عدد صحيح **أكبر من أو يساوي** العدد المُعطى. أي أنها تقوم بالتقريب دائمًا **نحو الأعلى**.

Ex:
ceil(3.2) = 4
ceil(3.7) = 4
ceil(-2.3) = -2
ceil(-2.9) = -2

Example 1: ceil() function for double, float and long double types

```cpp
#include <iostream>
#include <cmath>
using namespace std;

int main() {
    double x = 10.25, result;
    result = ceil(x);
    cout << "Ceil of " << x << " = " << result << endl;
    return 0;
}
```

```
Ceil of 10.25 = 11
```

Examples of Standard Library Functions in C++

floor()

The floor() function in C++ returns the largest possible integer value which is **less** than or **equal** to the given argument.

floor() Parameters: The floor() function takes a single argument whose floor value is computed.

دالة floor() في C++ تُستخدم لإرجاع أكبر عدد صحيح أقل من أو يساوي العدد المُعطى. أي أنها تقوم بتقريب العدد دائمًا نحو الأصغر.

floor() Return value

The floor() function returns the largest possible integer value which is **less than** or **equal** to the given argument.

Ex :
floor(3.7) = 3
floor(3.2) = 3
floor(-2.3) = -3
floor(-2.9) = -3

Example 1: How floor() works in C++?

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    double x = 10.25, result;
    result = floor(x);
    cout << "Floor of " << x << " = " << result << endl;
    return 0;
}
Output:
Floor of 10.25 = 10
```

round()

Returns integral value nearest to argument.

The round() function in C++ returns the integral value that is nearest to the argument, with halfway cases rounded away from zero.

round() Parameters: the round() function takes a single argument value to round.

Example 1: How round() works in C++?

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
    double x, result;

    x = 11.16;
    result = round(x);
    cout << "round(" << x << ") = " << result << endl;

    x = 13.87;
    result = round(x);
    cout << "round(" << x << ") = " << result << endl;

    x = 50.5;
    result = round(x);
    cout << "round(" << x << ") = " << result << endl;

    x = -11.16;
    result = round(x);
    cout << "round(" << x << ") = " << result << endl;

    x = -13.87;
    result = round(x);
    cout << "round(" << x << ") = " << result << endl;

    x = -50.5;
    result = round(x);
    cout << "round(" << x << ") = " << result << endl;

    return 0;
}
```

دالة round() في C++ تُستخدم لتقريب الأعداد العشرية إلى أقرب عدد صحيح.

- إذا كان الجزء العشري أقل من 0.5، يتم التقريب إلى العدد الصحيح الأصغر.
- إذا كان الجزء العشري أكبر من أو يساوي 0.5، يتم التقريب إلى العدد الصحيح الأكبر.
- تتعامل مع القيم الموجبة والسالبة بنفس الطريقة.

Ex :
round(3.3) = 3
round(3.5) = 4
round(-2.7) = -3
round(-2.5) = -2

```
round(11.16) = 11
round(13.87) = 14
round(50.5) = 51
round(-11.16) = -11
round(-13.87) = -14
round(-50.5) = -51
```

C++ trunc()

Truncates the decimal part of a number

Truncates a double value after the decimal point and gives the integer part as the result. The return value and the arguments are of the type double.

trunc() Parameters: The trunc() function takes a single argument whose trunc value is to be computed.

Example 1: How trunc() works in C++?

الدالة **trunc()** تُستخدم لحذف الجزء العشري من الرقم بدون التقريب، أي أنها تقرب الرقم **نحو الصفر** دائمًا.

Ex:
trunc(3.9) = 3
trunc(3.2) = 3
trunc(-2.7) = -2
trunc(-2.3) = -2

```cpp
#include <iostream>
#include <cmath>

using namespace std;

int main() {
    double x, result;

    x = 10.25;
    result = trunc(x);
    cout << "trunc(" << x << ") = " << result << endl;

    x = -34.251;
    result = trunc(x);
    cout << "trunc(" << x << ") = " << result << endl;

    return 0;
}
```

```
trunc(10.25) = 10
trunc(-34.251) = -34
```

Example of Standard Library Functions in C++

```
#include <iostream>
#include <cmath> //

using namespace std;
int main()
{
    // Define some decimal numbers
    double num1 = 3.14;
    double num2 = 7.5;
    double num3 = -2.7;
    double num4 = -5.5;
    double num5 = 75;

    // Round the numbers using the round function
    cout << "Original number: " << num1 << " After Round fun: " << round(num1) << endl;
    cout << "Original number: " << num2 << " After Round fun: " << round(num2) << endl;
    cout << "Original number: " << num3 << " After Round fun: " << round(num3) << endl;
    cout << "Original number: " << num4 << " After Round fun: " << round(num4) << endl << endl;
    // Square the numbers using the sqrt function
    cout << "Original number: " << num5 << " After sqrt fun: " << sqrt(num5) << endl << endl;

    // Square and round the numbers using the sqrt and round functions
    cout << "Original number: " << num5 << " After sqrt and round fun: " << round(sqrt(num5)) <<
endl << endl;

    // Absolute the numbers using the abs function
    cout << "Original number: " << num3 << " After abs fun: " << abs(num3) << endl << endl;

    // Round + Absolute the numbers using the round and abs functions
    cout << "Original numbers: " << num1<<"\t" << num3 << " After round + abs fun: " << round(num1)
+ abs(num3) << endl << endl;
return 0;
}
```

```
Original number: 3.14 After Round fun: 3
Original number: 7.5 After Round fun: 8
Original number: -2.7 After Round fun: -3
Original number: -5.5 After Round fun: -6

Original number: 75 After sqrt fun: 8.66025

Original number: 75 After sqrt and round fun: 9

Original number: -2.7 After abs fun: 2.7

Original numbers: 3.14  -2.7 After round + abs fun: 5.7
```

Example of Standard Library Functions in C++

Ex:
trunc(-2.7) = -2 // فقط يقطع الجزء العشري
round(-2.7) = -3 // يقرب إلى أقرب عدد صحيح
floor(-2.7) = -3 // يقرب للأسفل
ceil(-2.7) = -2 // يقرب للأعلى

Comparison Table of `trunc()`, `round()`, `floor()`, and `ceil()`

Number	`ceil()`	`floor()`	`round()`	`trunc()`
3.7	4	3	4	3
3.2	4	3	3	3
-3.7	-3	-4	-4	-3
-3.2	-3	-4	-3	-3
2.5	3	2	3	2
-2.5	-2	-3	-2	-2
2.0	2	2	2	2
-2.0	-2	-2	-2	-2