Programming in c++
Lecture_5

Functions

Dr. Sara Mohamed

Functions

- A function is a group of statements that is executed when it is called from some point of the program.
- **Functions types:**
  - Predefined Functions
  - User-Defined Functions

Predefined Functions

- C++ comes with libraries of predefined functions
- An include directive tells the compiler which library header file to include.
- C++ provides a library of predefined functions. The definitions of many common functions are found in **the cmath and cstdlib**.
- Functions are pow(parameter1, parameter2) ($like\ x^2$), sqrt(parameter1) (like $\sqrt{x}$), abs(parameter1), and floor(parameter1) such as floor(48.79) = 48.0

- Two components of a function definition
  - Function **declaration** (or function prototype)
    - Shows how the function is called
    - Must appear in the code before the function can be called
    - Syntax:
      Type_returned Function_Name(Parameter_List);
      //Comment describing what function does

      ;
  - Function **definition**
    - Describes how the function does its task
    - Can appear before or after the function is called
    - Syntax:
      Type_returned Function_Name(Parameter_List)
      {
      //code to make the function work
      }

Function Declaration

- Tells the return type
- Tells the name of the function
- Tells how many arguments are needed
- Tells the types of the arguments
- Tells the formal parameter names
  - Formal parameters are like placeholders for the actual arguments used when the function is called
  - Formal parameter names can be any valid identifier
- Example:

```
double total_cost(int number_par, double price_par);
// Compute total cost including 5% sales tax on
// number_par items at cost of price_par each
```

Function Libraries

To access these functions, your program must include camath and cstdlib library:

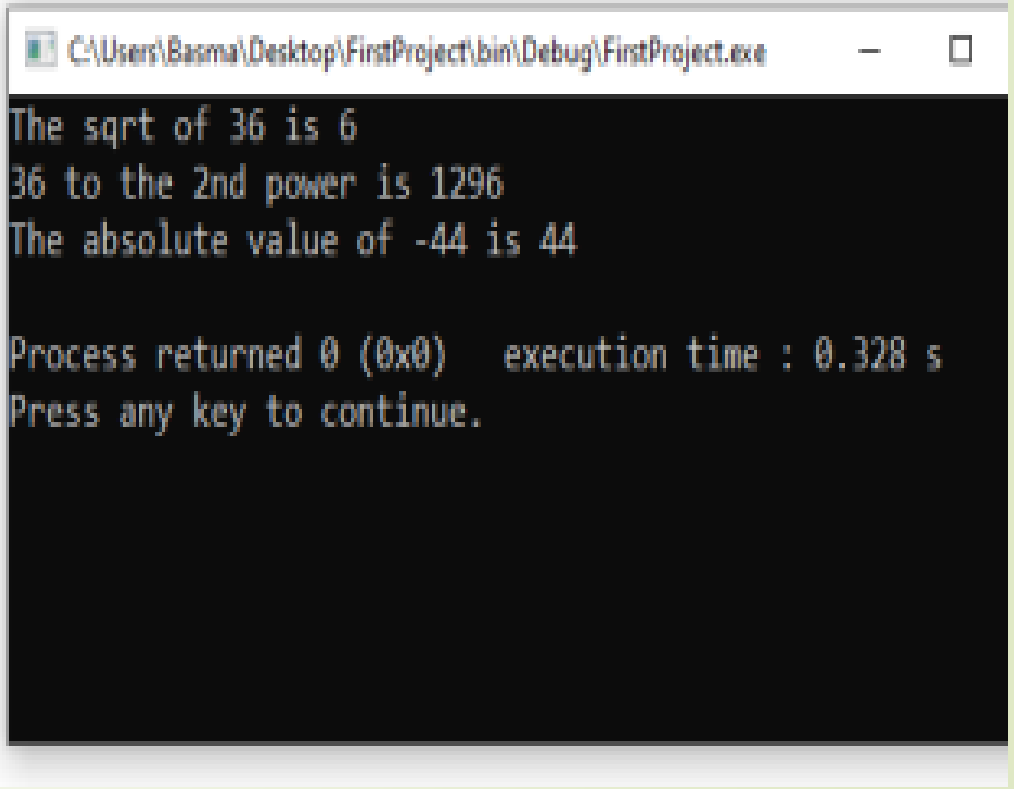
```
#include <cmath>

#include <cstdlib>
```

To call these functions:

```
Function_Name(Parameter1, Parameter2, ...)
```

Exercise 1: Use sqrt, pow, and abs function with cout function

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

using namespace std;

int main ( )
{
   cout << "The sqrt of 36 is " << sqrt( 36 ) << endl;

   cout << "36 to the 2nd power is " << pow( 36, 2 ) << endl;

   cout << "The absolute value of -44 is " << abs( -44.0 ) << endl;

   return 0;
}
```

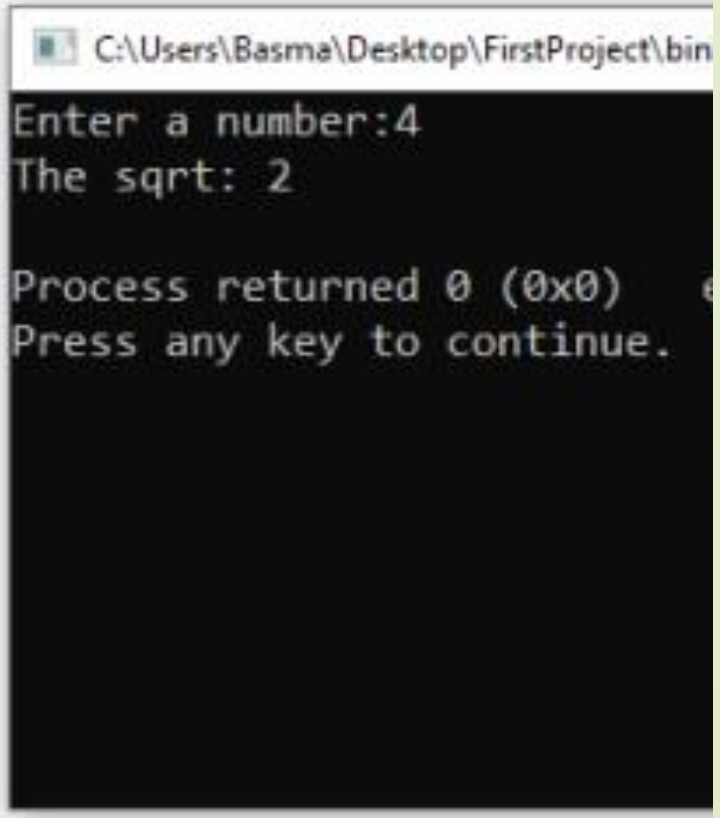


Exercise 2: Write a C++ program that take a number from user then output the square of this number?

Enter a number: 4

The square: 2

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

int main ( )
{
    int number;

    cout<<"Enter a number:";

    cin>> number;

    int result = sqrt(number);

    cout << "The sqrt: " << result << endl;

    return 0;
```



The **fabs()** function in C++ returns the absolute value of the argument. It is defined in the cmath header file.

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

int main() {

  // get absolute value of -5.5
  cout << fabs(-5.5);

  return 0;
}

// Output: 5.5
```

- The largest possible integer value which is less than or equal

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

#include <cmath>
int main() // Main function where the execution of the pr
{
  double num, result;
  num = 10.25;
  result = floor(num);

  cout << "Floor of " << num << " = " << result << endl;

  num = 34.5;
  result = floor(num);

  cout << "Floor of " << num << " = " << result << endl;

  num = 0.71;
  result = floor(num);

  cout << "Floor of " << num << " = " << result;

   return 0;

}
```

```
D:\app\dody\bin\Debug\dody.exe
Floor of 10.25 = 10
Floor of 34.5 = 34
Floor of 0.71 = 0
Process returned 0 (0x0)   execut
Press any key to continue.
```

round():the integral value that is nearest to num

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

#include <cmath>
int main() // Main function where the execution of the pr
{
  double num, result;
  num = 10.25;
  result = round(num);

  cout << "round of " << num << " = " << result << endl;

  num = 34.5;
  result = round(num);

  cout << "round of " << num << " = " << result << endl;

  num = 0.71;
  result = round(num);

  cout << "round of " << num << " = " << result;

   return 0;

}
```

```
D:\app\dody\bin\Debug\dody.exe
round of 10.25 = 10
round of 34.5 = 35
round of 0.71 = 1
Process returned 0 (0x0)   exe
Press any key to continue.
```

Defined Functions

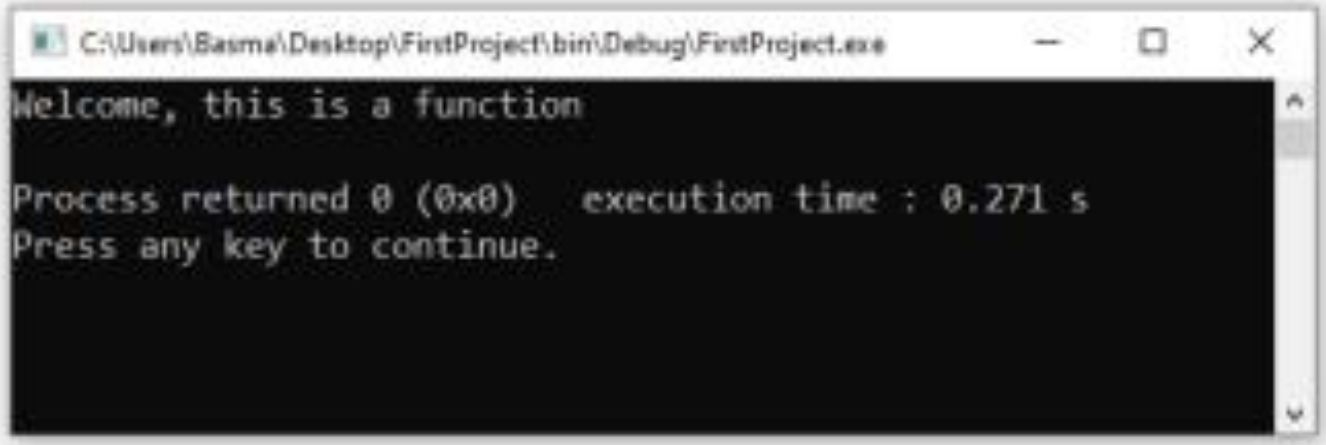
Syntax: type name (parameter1, parameter2, ...) { statements }

Where
- type is the data type specifier of the data returned by the function.
- name is the identifier by which it will be possible to call the function.
- parameters (as many as needed): Each parameter consists of a data type specifier followed by an identifier, like any regular variable declaration (for Exercise: int x) and which acts within the function as a regular local variable. They allow to pass arguments to the function when it is called. The different parameters are separated by commas.
- statements is the function's body. It is a block of statements surrounded by braces { }.

Function that doesn't take parameter and doesn't return result

- Exercise: Write a function that print a message: Welcome, this is a function.

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

void fun1()
{
    cout<<"Welcome, this is a function"<<endl;
}
int main()
{
    fun1();
}
```



Function that take a parameter and doesn't return value

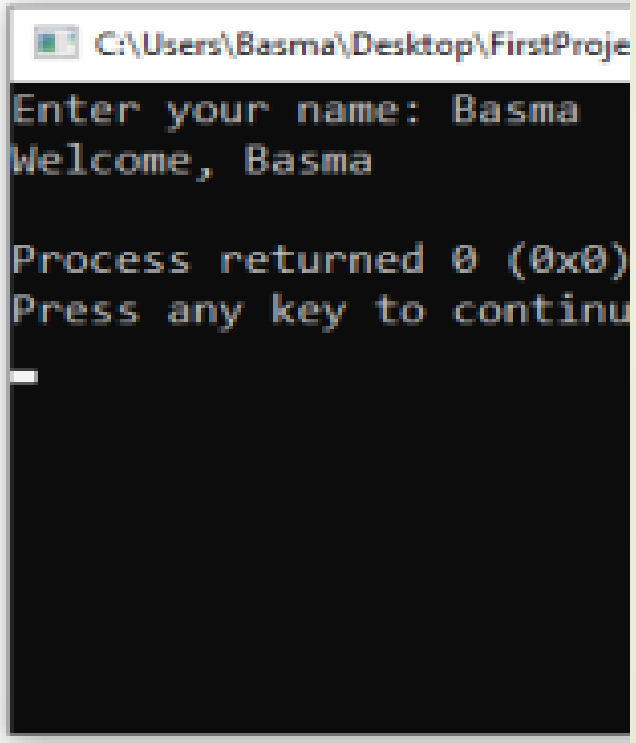
- Exercise: Write a function that take your name and print a message say: Welcome, your name

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

void printName(string name)
{

    cout<<"Welcome, "<< name<<endl;
}
int main()
{
    string n;

    cout<<"Enter your name: ";
    cin>> n;

    printName(n);
}
```

```
Select C:\Users\Basma\Desktop\FirstP
Enter your name: Basma
Welcome, Basma

Process returned 0 (0x0)
Press any key to continue.
```

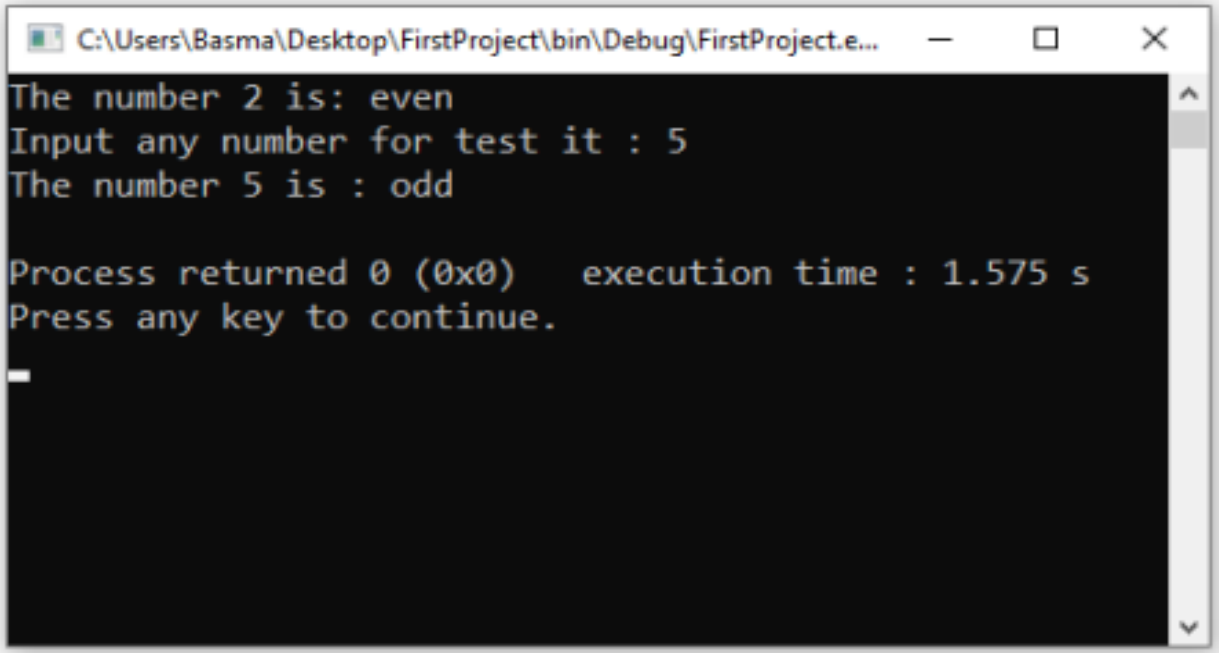
Function that doesn't take parameter and returns value

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

string getName()
{
    string n;

    cout<<"Enter your name: ";
    cin>> n;

    return n;

}
int main()
{
    string n = getName();
    cout<<"Welcome, "<< n<<endl;
}
```



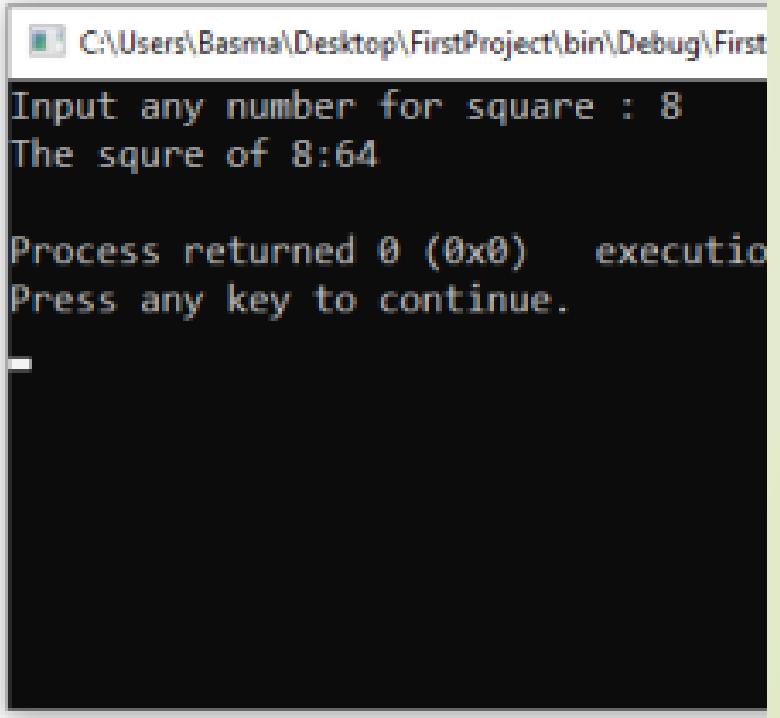
Function take parameters and returns value

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

string odd_even(int number)
{
    string result;

    if (number%2 == 0)
        result= "even";
    else
        result= "odd";
    return result;

}
int main()
{
    cout<<"The number 2 is: "<< odd_even(2)<<endl;
    int num;
    string result;

    cout<<"Input any number for test it : ";
    cin>>num;

    result =  odd_even(num);
    cout<<"The number "<< num<<" is : "<< result <<endl;

}
```



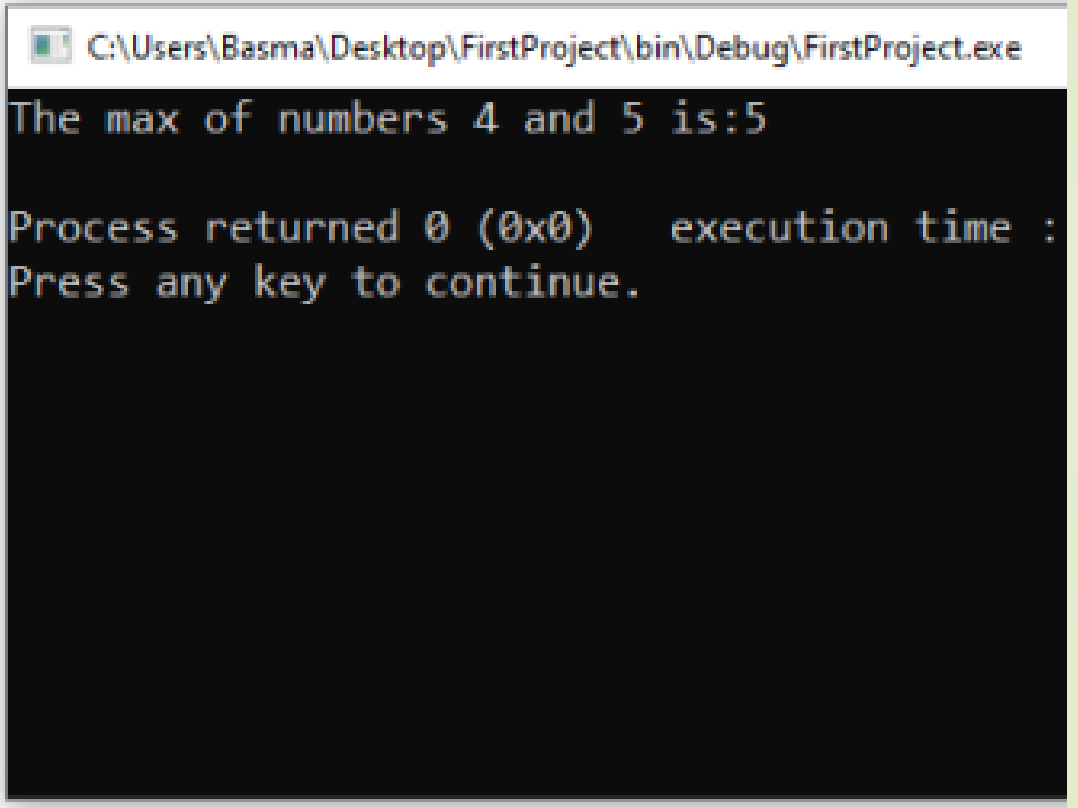
Write a function in C++ to find the square of any number using the function?

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

double square(double num)
{
    return (num * num);
}

int main()
{
    int num;
    double result;

    cout<<"Input any number for square : ";
    cin>>num;

    result = square(num);

    cout<<"The squre of "<< num<<':' << result<<endl;
    return 0;
}
```



Write a C++ function to find the Max of two numbers?

```cpp
#include <iostream>
using namespace std;
int getMax(int num1, int num2)
{
    int the_max= 0;
    if (num1 > num2)
      the_max= num1;

    else
        the_max= num2;

    return the_max;
}
int main()
{
    int get_max = getMax(4,5);
    cout<<"The max of numbers 4 and 5 is:"<<get_max<<endl;
}
```



Local and Global Variable

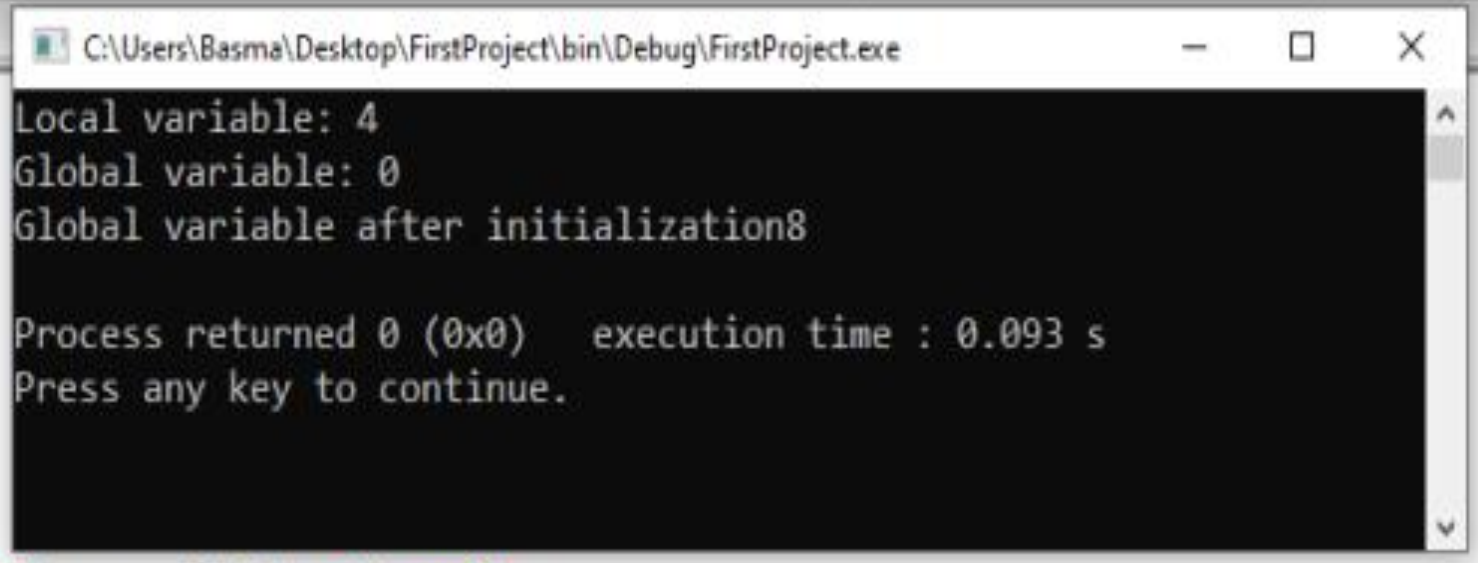
- Variables that are declared inside a function or a block are called local variables and are said to have local scope. These local variables can only be used within the function or block in which these are declared.

- Variables that are defined outside of all the functions and are accessible throughout the program are global variables and are said to have global scope.

Parameter	Local	Global
Scope	It is declared inside a function.	It is declared outside the function.
Value	If it is not initialized, a garbage value is stored	If it is not initialized zero is stored as default.
Lifetime	It is created when the function starts execution and lost when the functions terminate.	It is created before the program's global execution starts and lost when the program terminates.
Data sharing	Data sharing is not possible as data of the local variable can be accessed by only one function.	Data sharing is possible as multiple functions can access the same global variable.
Modification of variable value	When the value of the local variable is modified in one function, the changes are not visible in another function.	When the value of the global variable is modified in one function changes are visible in the rest of the program.
Accessed by	Local variables can be accessed with the help of statements, inside a function in which they are declared.	You can access global variables by any statement in the program.

The Garbage value is a random value at an address in the memory of a computer. Whenever a variable is defined without giving any value to it, it contains the leftover values from the previous program.

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

int g_variable;

int main()
{
    int loc_variable = 4;

    cout << "Local variable: " <<loc_variable << endl;
    cout << "Global variable: " <<g_variable << endl;

    g_variable = loc_variable * 2;

    cout << "Global variable after initialization" <<g_variable << endl;
    return 0;
}
```



C++ Function Overloading

- In C++, two functions can have the same name if the number and/or type of arguments passed is different.
- These functions having the same name but different arguments are known as overloaded functions. For example:

```
// same name different arguments
int test() { }
int test(int a) { }
float test(double a) { }
int test(int a, double b) { }
```

Overloaded functions may or may not have different return types, but they must have different arguments. For example:

```
// Error code
int test(int a) { }
double test(int b){ }
```

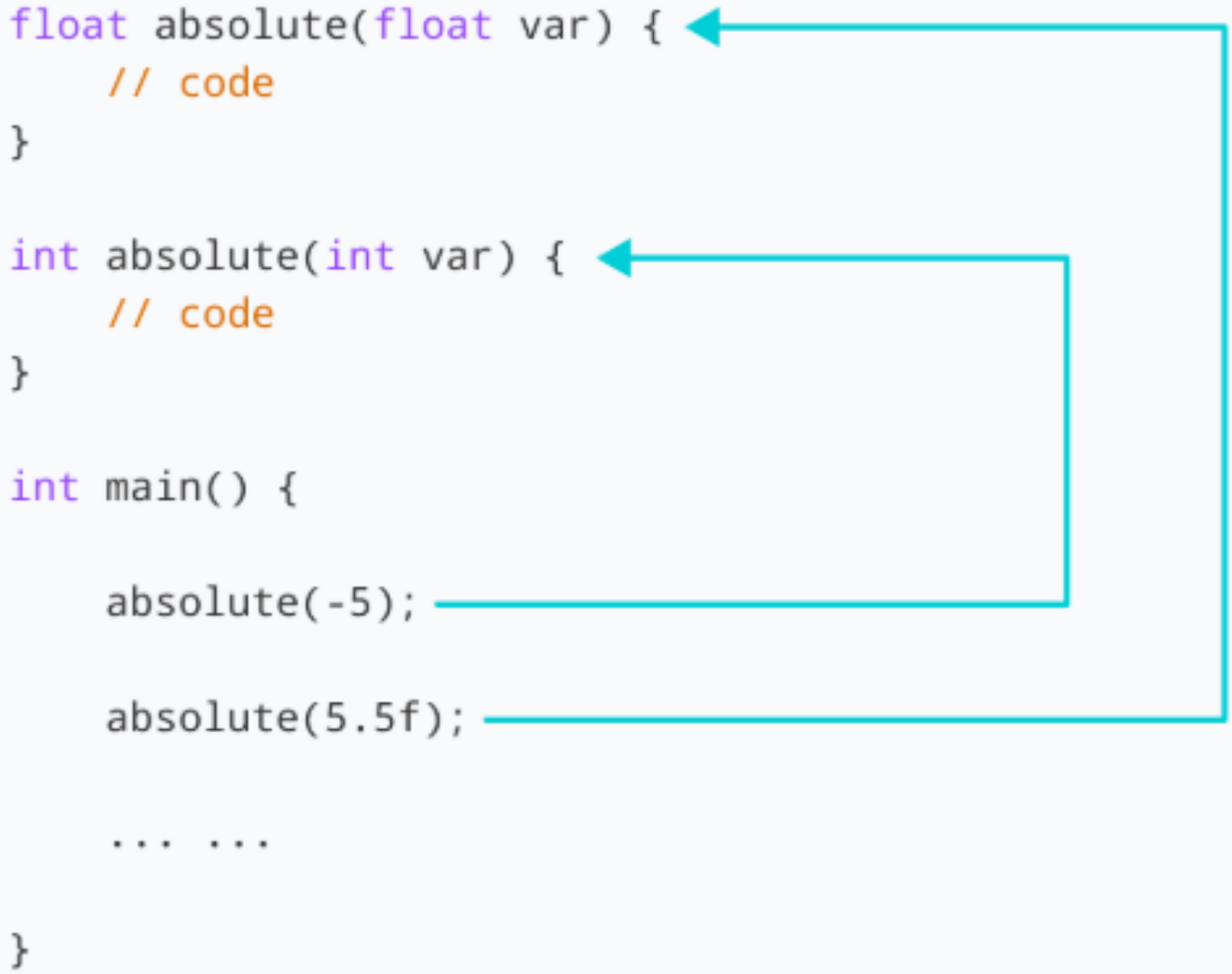
Function Overloading

Program to compute absolute value

```cpp
#include <iostream>

using namespace std;
// function with float type parameter
float absolute(float var){
    if (var < 0.0)
        var = -var;
    return var;
}

// function with int type parameter
int absolute(int var) {
    if (var < 0)
        var = -var;
    return var;
}
```

```cpp
int main()
{
    // call function with int type parameter
    cout << "Absolute value of -5 = " << absolute(-5) << en

    // call function with float type parameter
    cout << "Absolute value of 5.5 = " << absolute(5.5f) <<

    return 0;
}
```

```
D:\app\function\bin\Debug\function.exe
Absolute value of -5 = 5
Absolute value of 5.5 = 5.5

Process returned 0 (0x0)   execution time : 0.192 s
Press any key to continue.
```



Working of overloading for the absolute() function

Example 2: Overloading Using Different Number of Parameters

```
#include <iostream>

using namespace std;
// function with 2 parameters
void display(int var1, double var2) {
    cout << "Integer number: " << var1;
    cout << " and double number: " << var2 << endl;
}

// function with double type single parameter
void display(double var) {
    cout << "Double number: " << var << endl;
}

// function with int type single parameter
void display(int var) {
    cout << "Integer number: " << var << endl;
}
```

```
}
int main()
{
    int a = 5;
    double b = 5.5;

    // call function with int type parameter
    display(a);

    // call function with double type parameter
    display(b);

    // call function with 2 parameters
    display(a, b);


    return 0;
}
```

```
Integer number: 5
Double number: 5.5
Integer number: 5 and doub

Process returned 0 (0x0)
Press any key to continue.
```

C++ Recursion

- A function that calls itself is known as a recursive function. And this technique is known as recursion.
- Recursion is a programming technique where a function calls itself repeatedly until a specific base condition is met. A function that performs such self-calling behaviour is known as a **recursive function**, and each instance of the function calling itself is called a **recursive call**.

Working of Recursion in C++

```
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

int main()
{
    ... .. ...
    recurse();
    ... .. ...
}
```

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

void printHello(int n) {

    // Base Case
    if (n == 0) return;

    cout << "Hello" << endl;

    printHello(n - 1);
}

int main() {
    printHello(5);
    return 0;
}
```

Output

```
Hello
Hello
Hello
Hello
Hello
```

Explanation of the previous code

- This code demonstrates a simple recursive function that prints the word "Hello" five times.
- The function printHello(n) calls itself with a decremented value of n until it reaches the base case n == 0, at which point the recursion stops.
- Each recursive call prints "Hello" before making the next call, resulting in the message being printed once per call from n = 5 down to n = 1.

C++ Program to calculate the sum of first N natural

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

int nSum(int n)
{
    // base condition to terminate the recursion when N = 0
    if (n == 0) {
        return 0;
    }

    // recursive case / recursive call
    int res = n + nSum(n - 1);

    return res;
}

int main()
{
    int n = 5;

    // calling the function
    int sum = nSum(n);

    cout << "Sum = " << sum;
    return 0;
}
```

n=5 then
Res=5+nsum(4)=**5+10=15**
==========
n=4
Res=4+nsum(3)=**4+6=10**
==========
n=3
Res=3+nsum(2)=**3+3=6**
==========
n=2
Res=2+nsum(1)=**2+1=3**
==========
n=1
Res=1+nsum(0)=**1**
==========

C++ Recursion Factorial of a Number Using Recursion

- // Factorial of n = 1*2*3*...*n

```
int factorial(int n) {
    if (n > 1) {
        return n * factorial(n - 1);
    } else {
        return 1;
    }
}
int main()
{

    int n, result;

    cout << "Enter a non-negative number: ";
    cin >> n;

    result = factorial(n);
    cout << "Factorial of " << n << " = " << result;

    return 0;
```

```
factorial(5)
= 5 * factorial(4)
= 5 * 4 * factorial(3)
= 5 * 4 * 3 * factorial(2)
= 5 * 4 * 3 * 2 * factorial(1)
= 5 * 4 * 3 * 2 * 1 = 120
```

- **Advantages of C++ Recursion**
  - It makes our code shorter and cleaner.
  - Recursion is required in problems concerning data structures and advanced algorithms, such as Graph and Tree Traversal.
- **Disadvantages of C++ Recursion**
  - It takes a lot of stack space compared to an iterative program.
  - It uses more processor time.
  - It can be more difficult to debug compared to an equivalent iterative program.

Task

Write a C++ program that will display the calculator menu?

The program will prompt the user to choose the operation choice (from 1 to 5). Then it asks the user to input two integer vales for the calculation. See the sample below.

MENU

1. Add
2. Subtract
3. Multiply
4. Divide
5. Modulus

Enter your choice: 1

Enter your two numbers: 12 15

Result: 27

References

- https://www.programiz.com/cpp-programming/function-overloading
- https://www.geeksforgeeks.org/cpp/cpp-recursion/