



Void VS Value Returning Functions



اللَّهُمَّ ارْزُقْنِي عِلْمًا نَافِعًا وَاسِعًا عَمِيقًا

اَللّٰهُمَّ ارْزُقْنِيْ رِزْقًا وَّاسِعًا حَلَالًا طَيِّبًا
مُّبَارَكًا مِنْ عِنْدِكَ

Review

Function Call

Function
Definition

Function
Prototype

```
1  #include <iostream>
2  using namespace std;
3
4  void addition(int num1, int num2);
5
6  main() {
7      int number1, number2;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12
13     addition(number1, number2);
14
15     void addition(int num1, int num2)
16     {
17         int sum = num1 + num2;
18         cout << "Sum is: " << sum;
19     }
```

Review

Function Call

```
1  #include <iostream>
2  using namespace std;
3
4  void addition(int num1, int num2);
5
6  main() {
7      int number1, number2;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     addition(number1, number2);
13 }
14 void addition(int num1, int num2)
15 {
16     int sum = num1 + num2;
17     cout << "Sum is: " << sum;
18 }
```

Function
Prototype

2
Parameters

Objective of functions

Make an **independent** code that can be reused at multiple locations.

Objective of functions

Giving more than one responsibility to functions make them less independent and less reusable.

Function Tasks

How many
responsibilities of
this function ?

```
1  #include <iostream>
2  using namespace std;
3
4  void addition(int num1, int num2);
5
6  main() {
7      int number1, number2;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     addition(number1, number2);
13 }
14 void addition(int num1, int num2)
15 {
16     int sum = num1 + num2;
17     cout << "Sum is: " << sum;
18 }
```

Function
Prototype

void function

Function Tasks

The addition function has **two responsibilities**.

Can you identify ?

Function
Prototype

```
1  #include <iostream>
2  using namespace std;
3
4  void addition(int num1, int num2);
5
6  main() {
7      int number1, number2;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     addition(number1, number2);
13 }
14 void addition(int num1, int num2)
15 {
16     int sum = num1 + num2;
17     cout << "Sum is: " << sum;
18 }
```

void function

Function Tasks

The addition function has **two responsibilities**.

Sum the numbers and **printing** on the screen

Function
Prototype

```
1  #include <iostream>
2  using namespace std;
3
4  void addition(int num1, int num2);
5
6  main() {
7      int number1, number2;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     addition(number1, number2);
13 }
14 void addition(int num1, int num2)
15 {
16     int sum = num1 + num2;
17     cout << "Sum is: " << sum;
18 }
```

void function

Function Tasks

How to give it
single
responsibility so it
only sum the
numbers

Function
Prototype

```
1  #include <iostream>
2  using namespace std;
3
4  void addition(int num1, int num2);
5
6  main() {
7      int number1, number2;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     addition(number1, number2);
13 }
14 void addition(int num1, int num2)
15 {
16     int sum = num1 + num2;
17     cout << "Sum is: " << sum;
18 }
```

void function

Review

This function
doesn't return
anything.
Therefore, it is
called the **Void**
Function.

```
1  #include <iostream>
2  using namespace std;
3
4  void addition(int num1, int num2);
5
6  main() {
7      int number1, number2;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     addition(number1, number2);
13 }
14 void addition(int num1, int num2)
15 {
16     int sum = num1 + num2;
17     cout << "Sum is: " << sum;
18 }
```

Function
Prototype

void function

Review

It is like I have told you to go to admin and bring back the marker.



```
1  #include <iostream>
2  using namespace std;
3
4  void addition(int num1, int num2);
5
6  main() {
7      int number1, number2;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     addition(number1, number2);
13 }
14 void addition(int num1, int num2)
15 {
16     int sum = num1 + num2;
17     cout << "Sum is: " << sum;
18 }
```

Function
Prototype

void function

Review

But instead of bringing me back the marker you have started writing with that marker.

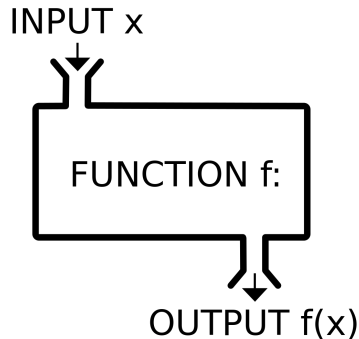
```
1  #include <iostream>
2  using namespace std;
3
4  void addition(int num1, int num2);
5
6  main() {
7      int number1, number2;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     addition(number1, number2);
13 }
14 void addition(int num1, int num2)
15 {
16     int sum = num1 + num2;
17     cout << "Sum is: " << sum;
18 }
```

Function
Prototype

void function

Review

Therefore, you have not fulfilled the true definition of function.



```
1  #include <iostream>
2  using namespace std;
3
4  void addition(int num1, int num2);
5
6  main() {
7      int number1, number2;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     addition(number1, number2);
13 }
14 void addition(int num1, int num2)
15 {
16     int sum = num1 + num2;
17     cout << "Sum is: " << sum;
18 }
```

Function
Prototype

void function

Functions

We must write a function that takes some input, perform some processing on it and returns an output.

```
1  #include <iostream>
2  using namespace std;
3
4  int addition(int num1, int num2);
5
6  main() {
7      int number1, number2, result;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     result = addition(number1, number2);
13     cout << "Sum is: " << result;
14 }
15
16 int addition(int num1, int num2)
17 {
18     int sum = num1 + num2;
19     return sum;
20 }
```


Functions

We must write a function that takes some input, does some processing on it and returns an output.

```
1 #include <iostream>
2 using namespace std;
3
4 int addition(int num1, int num2);
5
6 main() {
7     int number1, number2, result;
8     cout << "Enter First Number: ";
9     cin >> number1;
10    cout << "Enter Second Number: ";
11    cin >> number2;
12    result = addition(number1, number2);
13    cout << "Sum is: " << result;
14 }
15
16 int addition(int num1, int num2)
17 {
18     int sum = num1 + num2;
19     return sum;
20 }
```

Receive the result

Value
returning
function

Functions

We must write a function that takes some input, does some processing on it and returns an output.

```
1 #include <iostream>
2 using namespace std;
3
4 int addition(int num1, int num2);
5
6 main() {
7     int number1, number2, result;
8     cout << "Enter First Number: ";
9     cin >> number1;
10    cout << "Enter Second Number: ";
11    cin >> number2;
12    result = addition(number1, number2);
13    cout << "Sum is: " << result;
14 }
15
16 int addition(int num1, int num2)
17 {
18     int sum = num1 + num2;
19     return sum;
20 }
```

Displayed the
result

2 types of Functions

We have studied 2 types of functions.

1. Value Returning Function
2. Void Function (which returns nothing)

```

1 #include <iostream>
2 using namespace std;
3
4 int addition(int num1, int num2);
5
6 main(){
7     int number1, number2, result;
8     cout << "Enter First Number: ";
9     cin >> number1;
10    cout << "Enter Second Number: ";
11    cin >> number2;
12    result = addition(number1, number2);
13    cout << "Sum is: " << result;
14 }
15
16 int addition(int num1, int num2)
17 {
18     int sum = num1 + num2;
19     return sum;
20 }

```

**Value
Returning
Function**

Which
one is
better?

```

1 #include <iostream>
2 using namespace std;
3
4 void addition(int num1, int num2);
5
6 main(){
7     int number1, number2, result;
8     cout << "Enter First Number: ";
9     cin >> number1;
10    cout << "Enter Second Number: ";
11    cin >> number2;
12    addition(number1, number2);
13 }
14
15 void addition(int num1, int num2)
16 {
17     int sum = num1 + num2;
18     cout << "Sum is: " << sum;
19 }

```

**Void
Function**

Property of Functions


The **single-responsibility principle** is a computer-programming principle that states .

*“every function in a computer program should have **responsibility over a single part** of that program's functionality”*

```

1  #include <iostream>
2  using namespace std;
3
4  int addition(int num1, int num2);
5
6  main(){
7      int number1, number2, result;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     result = addition(number1, number2);
13     cout << "Sum is: " << result;
14 }
15 int addition(int num1, int num2)
16 {
17     int sum = num1 + num2;
18     return sum;
19 }

```

 **Value Returning Function**

Which
one is
better?

```

1  #include <iostream>
2  using namespace std;
3
4  void addition(int num1, int num2);
5
6  main(){
7      int number1, number2, result;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     addition(number1, number2);
13 }
14
15 void addition(int num1, int num2)
16 {
17     int sum = num1 + num2;
18     cout << "Sum is: " << sum;
19 }

```

Void Function

|| When the **Void functions** are used?

Previously, we have seen an example that value returning functions are better.

Then, the question is why and when **Void functions** are used?

When the Void functions are used?

Void functions can be used when we want to **print information** for the user to read.

For example,

1	<code>void printName(string name)</code>
2	<code>{</code>
3	<code> cout << "Username is: ", name;</code>
4	<code>}</code>

When the Void functions are used?

Void functions can be used when we want to **print information** for the user to read.

For example,

```
1 void printMenu()  
2 {  
3     cout << "*****Welcome*****";  
4     cout << "1. Login";  
5     cout << "2. Logout";  
6 }
```



Pre-Defined VS User-Defined Functions

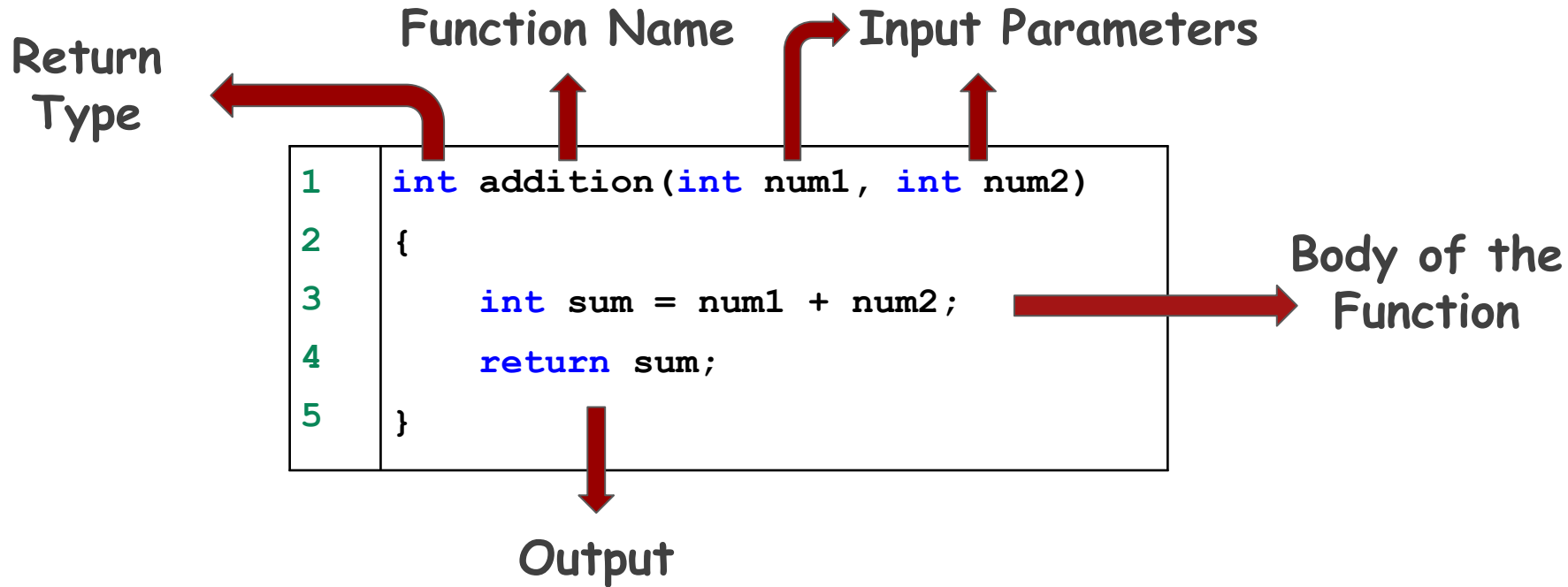
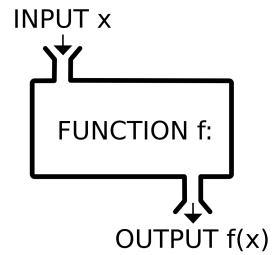


Types of Functions

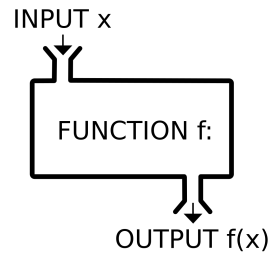
In C++, we have

1. User-Defined Functions
2. Pre-Defined (Library) Functions

User-Defined Functions in C++



User-Defined Functions in C++

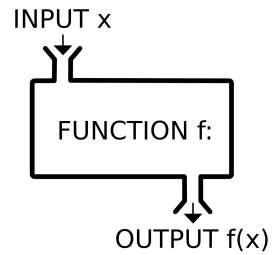


```
1 int addition(int num1, int num2)
2 {
3     int sum = num1 + num2;
4     return sum;
5 }
```

Function Call

```
1 main() {
2     int number1, number2, result;
3     cout << "Enter First Number: ";
4     cin >> number1;
5     cout << "Enter Second Number: ";
6     cin >> number2;
7     result = addition(number1, number2);
8     cout << "Sum is: " << result;
9 }
```

User-Defined Functions in C++

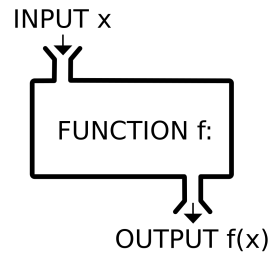


```
1 int addition(int num1, int num2)
2 {
3     int sum = num1 + num2;
4     return sum;
5 }
```

Parameter
Passing

```
1 main() {
2     int number1, number2, result;
3     cout << "Enter First Number: ";
4     cin >> number1;
5     cout << "Enter Second Number: ";
6     cin >> number2;
7     result = addition(number1, number2);
8     cout << "Sum is: " << result;
9 }
```

User-Defined Functions in C++



```
1 int addition(int num1, int num2)
2 {
3     int sum = num1 + num2;
4     return sum;
5 }
```

Function
returning the
output

```
1 main() {
2     int number1, number2, result;
3     cout << "Enter First Number: ";
4     cin >> number1;
5     cout << "Enter Second Number: ";
6     cin >> number2;
7     result = addition(number1, number2);
8     cout << "Sum is: " << result;
9 }
```

Pre-Defined Functions

- We can use **library functions** by invoking the functions directly; we don't need to write the functions ourselves.
- In order to use **library functions**, we usually need to include the **header file** in which these library functions are defined.

Pre-Defined Functions

- In C++, pre-defined functions are organized into separate libraries.
- For example, the header file `iostream` contains I/O functions; such as `cout` and `cin` functions.

1	<code>#include <iostream></code>
---	--

Pre-Defined Functions

- Similarly, the header file `cmath` contains math functions; such as `pow`, `sqrt`, `fabs` and `floor` etc.

1	<code>#include <iostream></code>
2	<code>#include <cmath></code>

Pre-Defined Functions

FunctionType	Header File	Purpose	Parameter(s) Type	Result
<code>pow(x, y)</code>	<code><cmath></code>	Returns x^y ; if x is negative, y must be a whole number <code>pow(0.16, 0.5) = 0.4</code>	double	double

Pre-Defined Functions

FunctionType	Header File	Purpose	Parameter(s) Type	Result
<code>pow(x, y)</code>	<code><cmath></code>	Returns x^y ; if x is negative, y must be a whole number <code>pow(0.16, 0.5) = 0.4</code>	double	double
<code>sqrt(x)</code>	<code><cmath></code>	Returns the nonnegative square root of x ; x must be nonnegative <code>sqrt(4.0) = 2.0</code>	double	double

Pre-Defined Functions

FunctionType	Header File	Purpose	Parameter(s) Type	Result
<code>pow(x, y)</code>	<code><cmath></code>	Returns x^y ; if x is negative, y must be a whole number <code>pow(0.16, 0.5) = 0.4</code>	double	double
<code>sqrt(x)</code>	<code><cmath></code>	Returns the nonnegative square root of x ; x must be nonnegative <code>sqrt(4.0) = 2.0</code>	double	double
<code>fabs(x)</code>	<code><cmath></code>	Returns the absolute value of its argument <code>fabs(-5.67) = 5.67</code>	double	double



Local VS Global Variables



Review

Both **main** function, and **addition** function has different set of variables.

```
1  #include <iostream>
2  using namespace std;
3
4  int addition(int num1, int num2);
5
6  main() {
7      int number1, number2, result;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     result = addition(number1, number2);
13     cout << "Sum is: " << result;
14 }
15 int addition(int num1, int num2)
16 {
17     int sum = num1 + num2;
18     return sum;
19 }
```

Function with no parameters

Instead of **passing parameters**, can we use same parameters?
i.e.,

number1, number2 and result

```
1  #include <iostream>
2  using namespace std;
3
4  int addition(int num1, int num2);
5
6  main() {
7      int number1, number2, result;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     result = addition(number1, number2);
13     cout << "Sum is: " << result;
14 }
15 int addition(int num1, int num2)
16 {
17     int sum = num1 + num2;
18     return sum;
19 }
```


Function with no parameters

Instead of **passing parameters**, can we use same parameters?
i.e.,

number1, number2 and result

```
1  #include <iostream>
2  using namespace std;
3
4  int addition();
5
6  main() {
7      int number1, number2, result;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     result = addition();
13     cout << "Sum is: " << result;
14 }
15
16 int addition()
17 {
18     result = number1 + number2;
19     return result;
20 }
```

Error

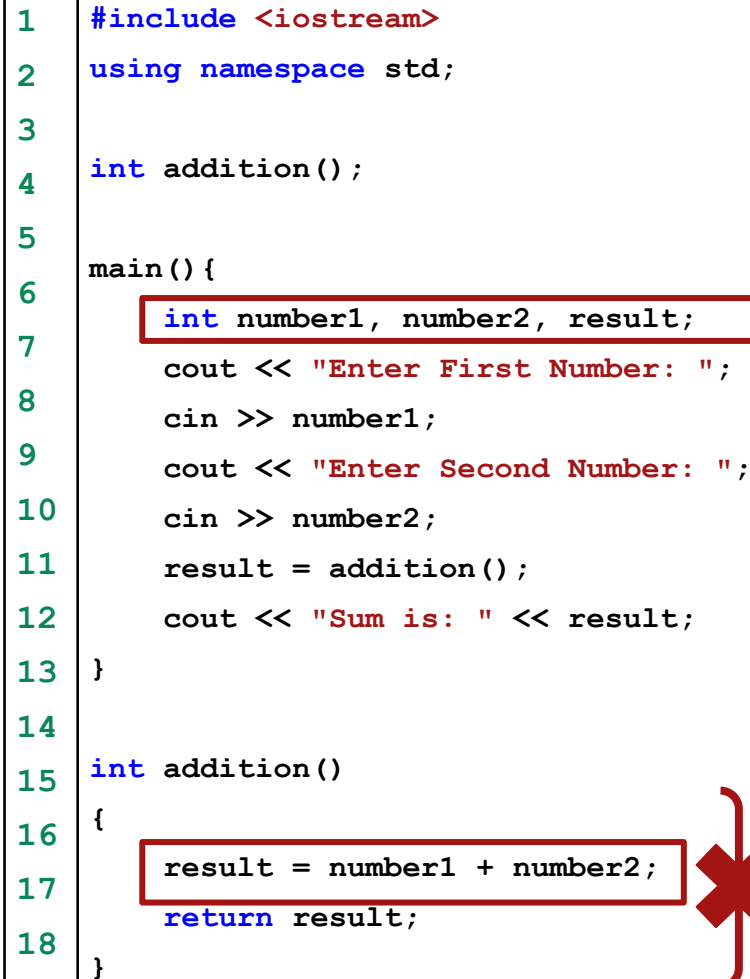
```
example.cpp: In function 'int addition()':
example.cpp:18:5: error: 'result' was not declared in this scope
  18 |     result = number1 + number2;
     |     ~~~~~
example.cpp:18:14: error: 'number1' was not declared in this scope
  18 |     result = number1 + number2;
     |             ~~~~~
example.cpp:18:24: error: 'number2' was not declared in this scope
  18 |     result = number1 + number2;
     |                       ~~~~~
```

```
1  #include <iostream>
2  using namespace std;
3
4  int addition();
5
6  main() {
7      int number1, number2, result;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     result = addition();
13     cout << "Sum is: " << result;
14 }
15
16 int addition()
17 {
18     result = number1 + number2;
19     return result;
20 }
```

Local Variables

Variables within a block `{ }` remain accessible only **within** that block and not outside that block. These are called **local variables** of block.

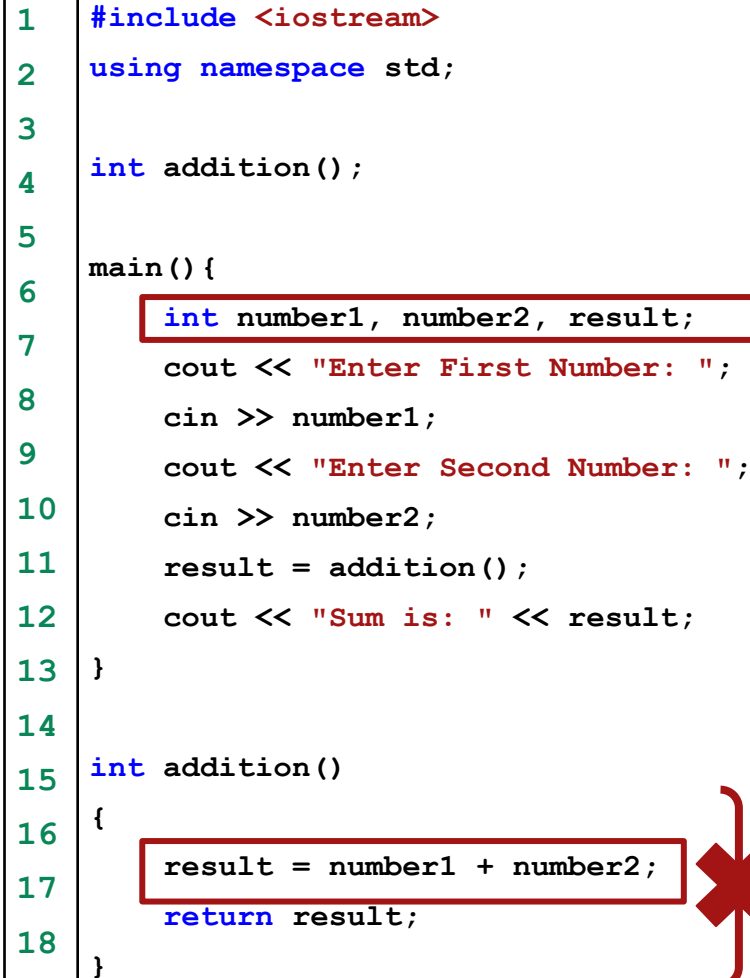
```
1  #include <iostream>
2  using namespace std;
3
4  int addition();
5
6  main() {
7      int number1, number2, result;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     result = addition();
13     cout << "Sum is: " << result;
14 }
15
16 int addition()
17 {
18     result = number1 + number2;
19     return result;
20 }
```



Scope of Variables

The area within a variable can be accessed is called **scope** of the variable.

```
1  #include <iostream>
2  using namespace std;
3
4  int addition();
5
6  main() {
7      int number1, number2, result;
8      cout << "Enter First Number: ";
9      cin >> number1;
10     cout << "Enter Second Number: ";
11     cin >> number2;
12     result = addition();
13     cout << "Sum is: " << result;
14 }
15
16 int addition()
17 {
18     result = number1 + number2;
19     return result;
20 }
```



Solution: Global Variables

We can Declare **Global Variables** before the main function.

```
1  #include <iostream>
2  using namespace std;
3
4  void addition();
5  int number1, number2, result;
6  main() {
7      cout << "Enter First Number: ";
8      cin >> number1;
9      cout << "Enter Second Number: ";
10     cin >> number2;
11     addition();
12     cout << "Sum is: " << result;
13 }
14 void addition()
15 {
16     result = number1 + number2;
17 }
18
```

Solution: Global Variables

We can Declare **Global Variables** before the main function.


```
C:\C++>c++ example.cpp -o example.exe  
  
C:\C++>example.exe  
Enter First Number: 5  
Enter Second Number: 9  
Sum is: 14  
C:\C++>
```

```
1  #include <iostream>  
2  using namespace std;  
3  
4  void addition();  
5  int number1, number2, result;  
6  
7  main() {  
8      cout << "Enter First Number: ";  
9      cin >> number1;  
10     cout << "Enter Second Number: ";  
11     cin >> number2;  
12     addition();  
13     cout << "Sum is: " << result;  
14 }  
15  
16 void addition()  
17 {  
18     result = number1 + number2;  
19 }
```

Local Variables

Which one is better?

```
1 #include <iostream>
2 using namespace std;
3
4 int addition(int num1, int num2);
5
6 main() {
7     int number1, number2, result;
8     cout << "Enter First Number: ";
9     cin >> number1;
10    cout << "Enter Second Number: ";
11    cin >> number2;
12    result = addition(number1, number2);
13    cout << "Sum is: " << result;
14 }
15 int addition(int num1, int num2)
16 {
17     int sum = num1 + num2;
18     return sum;
19 }
```



Low Coupled

Global Variables

```
1 #include <iostream>
2 using namespace std;
3
4 void addition();
5 int number1, number2, result;
6
7 main() {
8     cout << "Enter First Number: ";
9     cin >> number1;
10    cout << "Enter Second Number: ";
11    cin >> number2;
12    addition();
13    cout << "Sum is: " << result;
14 }
15
16 void addition()
17 {
18     result = number1 + number2;
19 }
```

High Coupled

Local Vs Global Variables

- Low Coupling is **Good** and **Always Desired**.
- In Some Cases, where multiple function need to share the same data we have to declare **GLOBAL** variables.

Global Variables Scope

Local
Variables
Scope

Learning Outcome

Differentiate between **Void** and **Value Returning** Functions, **Pre-defined** and **User-defined** functions, **Local** and **Global** Variables.



Self Assessment

1. What will be the output of the program?

```
1  #include <iostream>
2  using namespace std;
3
4  /* global variable declaration */
5  int g = 20;
6  main()
7  {
8      /* local variable declaration */
9      int g = 10;
10     cout << "Value of g = " << g;
11 }
```



Self Assessment

2. What will be the **sequence of the output** of the program?
How many **global variables**, **local variables of main**, **parameters of sum function** and **local variables of sum function** are there?

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

int a = 20;
int sum(int a, int b);
main ()
{
    int a = 10;
    int b = 20;
    int c = 0;
    cout << "value of a in main() = " << a << endl;
    c = sum( a, b);
    cout << "value of c in main() = " << c << endl;
}
/* function to add two integers */
int sum(int a, int b)
{
    cout << "value of a in sum() = " << a << endl;
    cout << "value of b in sum() = " << b << endl;
    return a + b;
}
```