Void VS Value Returning Functions

اَللّٰهُمَّ اُرْزُقْنِى عِلْمًا نَافِعًا وَاسِعًا عَمِيْقًا

اَللّٰهُمَّ اُرْزُقْنِى رِزْقًا وَاسِعًا حَلَالًا طَيِّبًا
مُبَارَكًا مِنْ عِنْدِكَ

Review

Function Prototype

Function Call

Function Definition

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

void addition(int num1, int num2);

main(){
    int number1, number2;
    cout << "Enter First Number:  ";
    cin >> number1;
    cout << "Enter Second Number:  ";
    cin >> number2;
    addition(number1, number2);
}

void addition(int num1, int num2)
{
    int sum = num1 + num2;
    cout << "Sum is:  " << sum;
}
```

Review

Function Call

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

void addition(int num1, int num2);

main(){
    int number1, number2;
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    addition(number1, number2);
}

void addition(int num1, int num2)
{
    int sum = num1 + num2;
    cout << "Sum is: " << sum;
}
```

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 ?

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

void addition(int num1, int num2);

main(){
     int number1, number2;
     cout << "Enter First Number: ";
     cin >> number1;
     cout << "Enter Second Number: ";
     cin >> number2;
     addition(number1, number2);
}

void addition(int num1, int num2)
{
     int sum = num1 + num2;
     cout << "Sum is: " << sum;
}
```

Function Prototype

void function

Function Tasks

The addition function has **two responsibilities**.

Can you identify ?

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

void addition(int num1, int num2);

main(){
    int number1, number2;
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    addition(number1, number2);
}

void addition(int num1, int num2)
{
    int sum = num1 + num2;
    cout << "Sum is: " << sum;
}
```

Function Prototype

void function

Function Tasks

The addition function has **two responsibilities**.

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

Function Prototype

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

void addition(int num1, int num2);

main(){
    int number1, number2;
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    addition(number1, number2);
}

void addition(int num1, int num2)
{
    int sum = num1 + num2;
    cout << "Sum is: " << sum;
}
```

void function

Function Tasks

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

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

void addition(int num1, int num2);

main(){
     int number1, number2;
     cout << "Enter First Number: ";
     cin >> number1;
     cout << "Enter Second Number: ";
     cin >> number2;
     addition(number1, number2);
}

void addition(int num1, int num2)
{
     int sum = num1 + num2;
     cout << "Sum is: " << sum;
}
```

Function Prototype

void function

Review

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

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

void addition(int num1, int num2);

main(){
     int number1, number2;
     cout << "Enter First Number: ";
     cin >> number1;
     cout << "Enter Second Number: ";
     cin >> number2;
     addition(number1, number2);
}

void addition(int num1, int num2)
{
     int sum = num1 + num2;
     cout << "Sum is: " << sum;
}
```

Function Prototype

void function

Review

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

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

void addition(int num1, int num2);

main(){
    int number1, number2;
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    addition(number1, number2);
}

void addition(int num1, int num2)
{
    int sum = num1 + num2;
    cout << "Sum is: " << sum;
}
```

Function Prototype

void function

Review

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

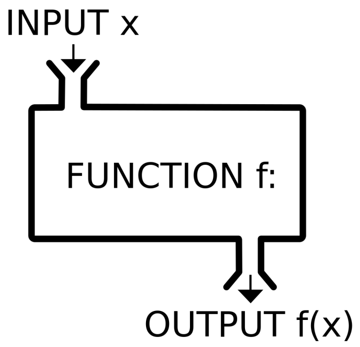
```
#include <iostream>
using namespace std;

void addition(int num1, int num2);

main(){
    int number1, number2;
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    addition(number1, number2);
}

void addition(int num1, int num2)
{
    int sum = num1 + num2;
    cout << "Sum is: " << sum;
}
```

Function Prototype

void function

Review

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



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

void addition(int num1, int num2);


main(){
     int number1, number2;
     cout << "Enter First Number: ";
     cin >> number1;
     cout << "Enter Second Number: ";
     cin >> number2;
     addition(number1, number2);
}

void addition(int num1, int num2)
{
     int sum = num1 + num2;
     cout << "Sum is: " << sum;
}
```

Function Prototype

void function

Functions

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

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

int addition(int num1, int num2);

main(){
    int number1, number2, result;
    cout << "Enter First Number:  ";
    cin >> number1;
    cout << "Enter Second Number:  ";
    cin >> number2;
    result = addition(number1, number2);
    cout << "Sum is:  " << result;
}

int addition(int num1, int num2)
{
    int sum = num1 + num2;
    return sum;
}
```

Functions

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

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

int addition(int num1, int num2);

main(){
    int number1, number2, result;
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    result = addition(number1, number2);
    cout << "Sum is: " << result;
}

int addition(int num1, int num2)
{
    int sum = num1 + num2;
    return sum;
}
```

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.**

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

int addition(int num1, int num2);

main(){
    int number1, number2, result;
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    result = addition(number1, number2);
    cout << "Sum is: " << result;
}

int addition(int num1, int num2)
{
    int sum = num1 + num2;
    return sum;
}
```

Displayed the result

2 types of Functions

We have studied 2 types of functions.

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

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

int addition(int num1, int num2);

main(){
    int number1, number2, result;
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    result = addition(number1, number2);
    cout << "Sum is: " << result;
}

int addition(int num1, int num2)
{
    int sum = num1 + num2;
    return sum;
}
```

Value Returning Function

Which one is better?

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

void addition(int num1, int num2);

main(){
    int number1, number2, result;
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    addition(number1, number2);
}


void addition(int num1, int num2)
{
    int sum = num1 + num2;
    cout << "Sum is: " << sum;
}
```

Void Function

Property of Functions

The single-responsibility principle is a computer-programming principle that states that every function in a computer program should have responsibility over a single part of that program's functionality.

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

int addition(int num1, int num2);

main(){
    int number1, number2, result;
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    result = addition(number1, number2);
    cout << "Sum is: " << result;
}

int addition(int num1, int num2)
{
    int sum = num1 + num2;
    return sum;
}
```

✓

Value Returning Function

Which one is better?

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

void addition(int num1, int num2);

main(){
    int number1, number2, result;
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    addition(number1, number2);
}

void addition(int num1, int num2)
{
    int sum = num1 + num2;
    cout << "Sum is: " << sum;
}
```

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,

```
void printName(string name)
{
    cout << "Username is: ", name;
}
```

When the Void functions are used?

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

For example,

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

Pre-Defined VS User-Defined Functions

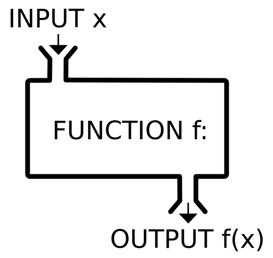
Types of Functions

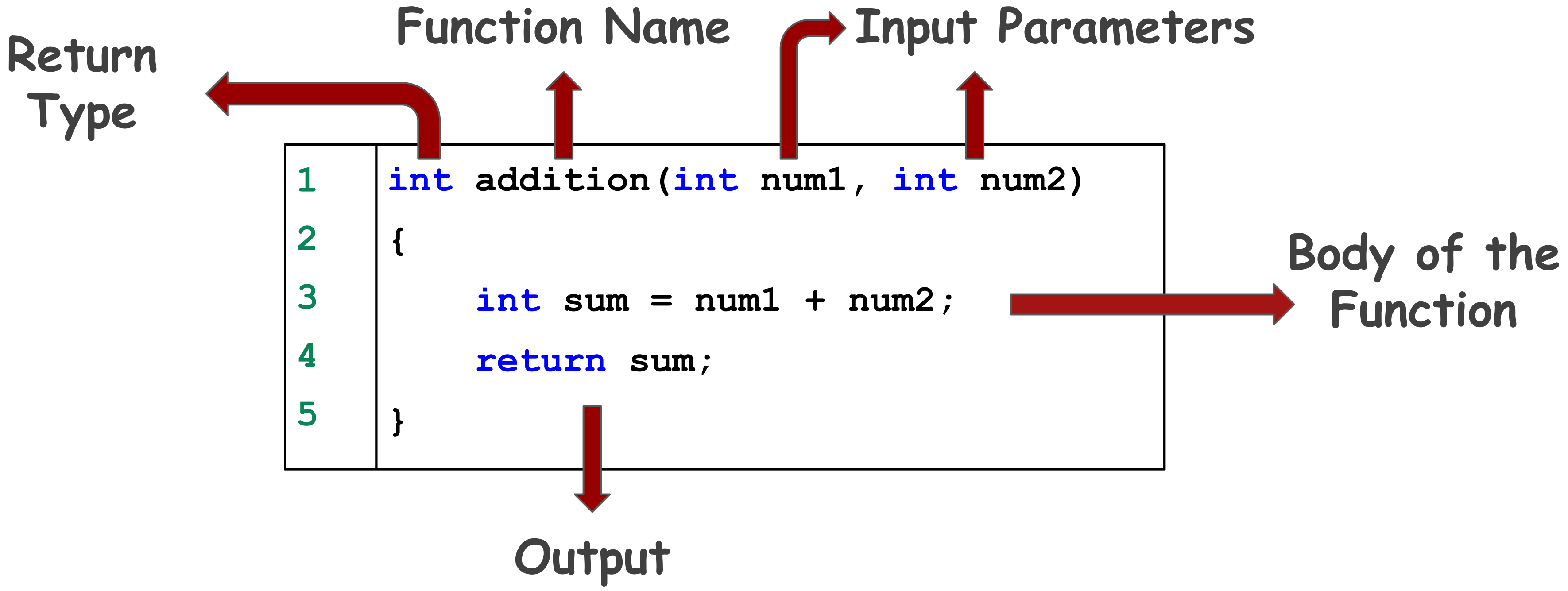
In C++, we have

1. User-Defined Functions

2. Pre-Defined (Library) Functions

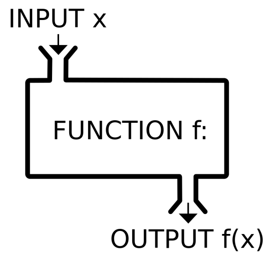
User-Defined Functions in C++





```cpp
int addition(int num1, int num2)
{
    int sum = num1 + num2;
    return sum;
}
```

User-Defined Functions in C++

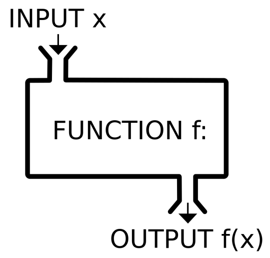


```cpp
int addition(int num1, int num2)
{
     int sum = num1 + num2;
     return sum;
}
```

Function Call

```cpp
main(){
     int number1, number2, result;
     cout << "Enter First Number: ";
     cin >> number1;
     cout << "Enter Second Number: ";
     cin >> number2;
     result = addition(number1, number2);
     cout << "Sum is: " << result;
}
```

User-Defined Functions in C++

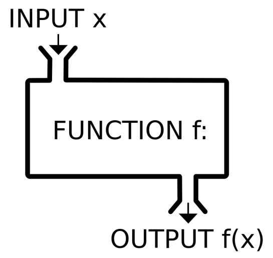


```cpp
int addition(int num1, int num2)
{
      int sum = num1 + num2;
      return sum;
}
```

Parameter Passing

```cpp
main(){
      int number1, number2, result;
      cout << "Enter First Number: ";
      cin >> number1;
      cout << "Enter Second Number: ";
      cin >> number2;
      result = addition(number1, number2);
      cout << "Sum is: " << result;
}
```

User-Defined Functions in C++



```cpp
int addition(int num1, int num2)
{
    int sum = num1 + num2;
    return sum;
}
```

Function returning the output

```cpp
main(){
    int number1, number2, result;
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    result = addition(number1, number2);
    cout << "Sum is: " << result;
}
```

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.

```
#include <iostream>
```

Pre-Defined Functions

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

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

Pre-Defined Functions

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

Pre-Defined Functions

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

Pre-Defined Functions

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

Local VS Global Variables

Review

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

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

int addition(int num1, int num2);

main(){
    int number1, number2, result;
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    result = addition(number1, number2);
    cout << "Sum is: " << result;
}

int addition(int num1, int num2)
{
    int sum = num1 + num2;
    return sum;
}
```

Function with no parameters

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

number1, number2 and result

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

int addition(int num1, int num2);

main(){
    int number1, number2, result;
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    result = addition(number1, number2);
    cout << "Sum is: " << result;
}

int addition(int num1, int num2)
{
    int sum = num1 + num2;
    return sum;
}
```

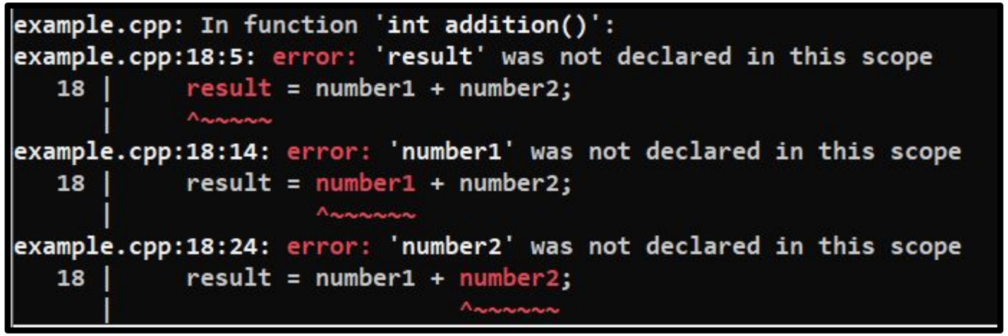
Function with no parameters

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

number1, number2 and result

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

int addition();

main(){
    int number1, number2, result;
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    result = addition();
    cout << "Sum is: " << result;
}

int addition()
{
    result = number1 + number2;
    return result;
}
```

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;
      |                        ^~~~~~~
```

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

int addition();

main(){
    int number1, number2, result;
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    result = addition();
    cout << "Sum is: " << result;
}

int addition()
{
    result = number1 + number2;
    return result;
}
```

Local Variables

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

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

int addition();

main(){
    int number1, number2, result;
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    result = addition();
    cout << "Sum is: " << result;
}

int addition()
{
    result = number1 + number2;
    return result;
}
```

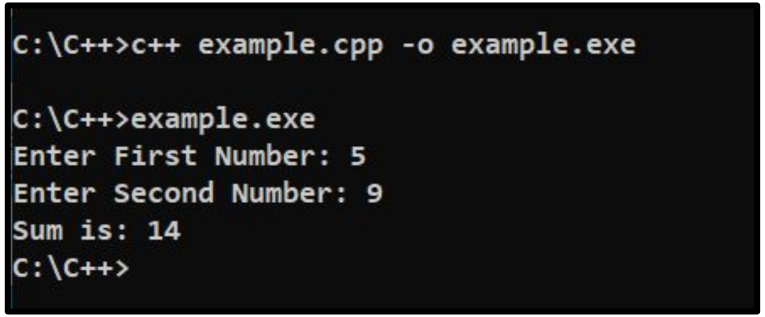
Solution: Global Variables

We can Declare Global Variables before the main function.

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

int addition();
int number1, number2, result;
main(){
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    result = addition();
    cout << "Sum is: " << result;
}

int addition()
{
    result = number1 + number2;
    return result;
}
```

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++>
```

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

int addition();
int number1, number2, result;
main(){
    cout << "Enter First Number: ";
    cin >> number1;
    cout << "Enter Second Number: ";
    cin >> number2;
    result = addition();
    cout << "Sum is: " << result;
}

int addition()
{
    result = number1 + number2;
    return result;
}
```

Local Variables

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

int addition(int num1, int num2);

main(){
    int number1, number2, result;
    cout << "Enter First Number:  ";
    cin >> number1;
    cout << "Enter Second Number:  ";
    cin >> number2;
    result = addition(number1, number2);
    cout << "Sum is:  " << result;
}

int addition(int num1, int num2)
{
    int sum = num1 + num2;
    return sum;
}
```

✓ Low Coupled

Which one is better?

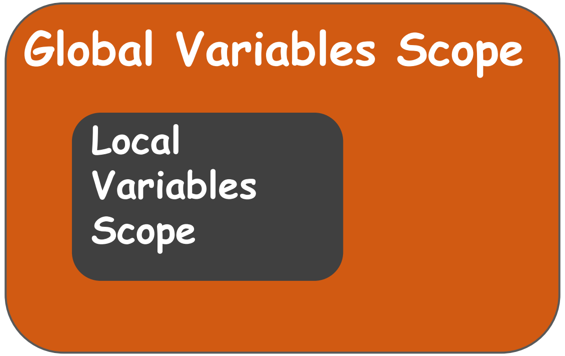
Global Variables

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

int addition();
int number1, number2, result;
main(){
    cout << "Enter First Number:  ";
    cin >> number1;
    cout << "Enter Second Number:  ";
    cin >> number2;
    result = addition();
    cout << "Sum is:  " << result;
}

int addition()
{
    result = number1 + number2;
    return result;
}
```

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.



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?**

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

/* global variable declaration */
int g = 20;
main()
{
     /* local variable declaration */
     int g = 10;
     cout << "Value of g = " << g;
}
```

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?

```cpp
#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;
}
```