CL1004 – Object-Oriented Programming Lab

Lab # 1

Procedural Programming

Instructor: Muhammad Saad Rashad

Email: saad.rashad@nu.edu.pk

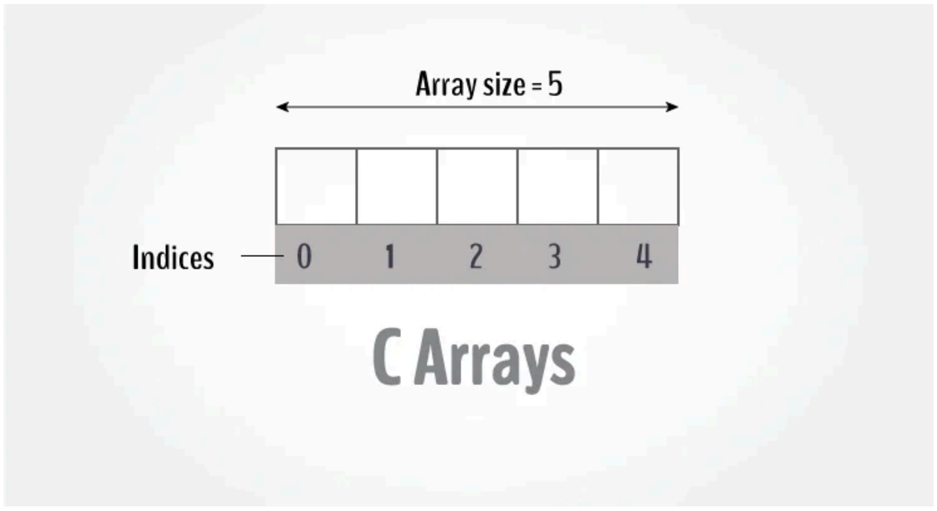
Department of Computer Science,

National University of Computer and Emerging Sciences FAST

Peshawar

1. Arrays in C/C++

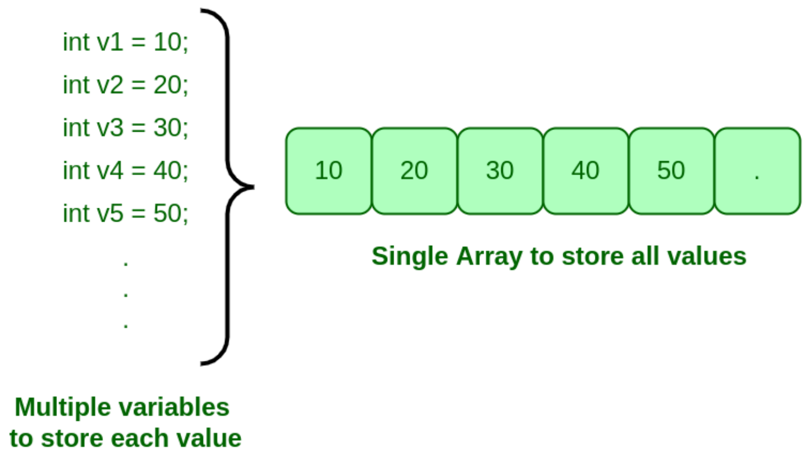
1. It is a group of variables of similar data types referred to by a single element.
2. Its elements are stored in a contiguous memory location.
3. The size of the array should be mentioned while declaring it.
4. Array elements are always counted from zero (0) onward.
5. Array elements can be accessed using the position of the element in the array.
6. The array can have one or more dimensions.



An array in C++ is a collection of similar data items stored at contiguous memory locations and elements can be accessed randomly using indices of an array

2. Why do we need arrays?

We can use normal variables (v1, v2, v3, ..) when we have a small number of objects, but if we want to store a large number of instances, it becomes difficult to manage them with normal variables. The idea of an array is to represent many instances in one variable.


int v1 = 10;
int v2 = 20;
int v3 = 30;
int v4 = 40;
int v5 = 50;
10
20
30
40
50
Single Array to store all values
Multiple variables to store each value

Advantages:-

- Code Optimization: we can retrieve or sort the data efficiently.
- Random access: We can get any data located at an index position.

Disadvantages:-

- Size Limit: We can store only the fixed size of elements in the array. It doesn't grow its size at runtime.

An array is a variable that can store multiple values. For example, if you want to store 100 integers, you can create an array for it.

```
int data[100];
```

3. How to declare an array?

```
dataType arrayName[arraySize];
```

For example:

```
float mark[5];
```

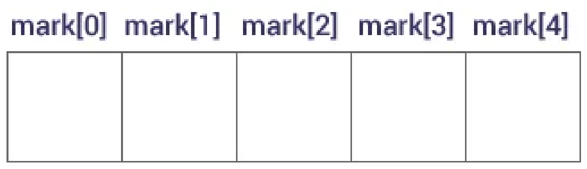
Here, we declared an array, mark, of floating-point type. And its size is 5. Meaning, it can hold 5 floating-point values.

It's important to note that the size and type of an array cannot be changed once it is declared.

4. Access Array Elements

You can access elements of an array by indices.

Suppose you declared an array mark as above. The first element is mark[0], the second element is mark[1] and so on.



Few keynotes:

- Arrays have 0 as the first index, not 1. In this example, **mark[0]** is the first element.
- If the size of an array is **n**, to access the last element, the **n-1** index is used. In this example, **mark[4]**

5. How to initialize an array?

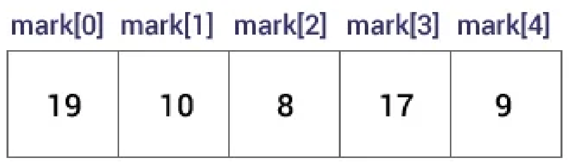
It is possible to initialize an array during declaration. For example:

```
int mark[5] = {19, 10, 8, 17, 9};
```

You can also initialize an array like this.

```
int mark[ ] = {19, 10, 8, 17, 9};
```

Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.



Here,

mark[0] is equal to 19

mark[1] is equal to 10

mark[2] is equal to 8

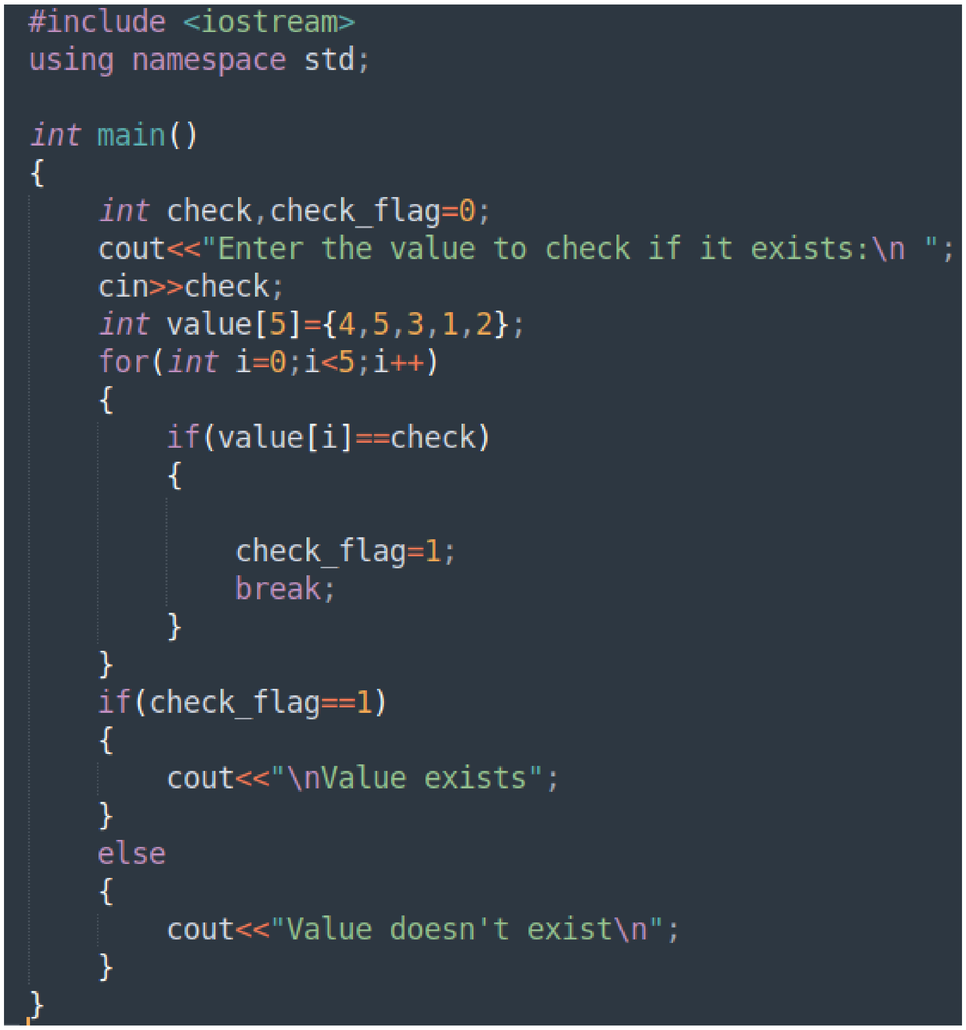
mark[3] is equal to 17

mark[4] is equal to 9

6. Change Value of Array elements

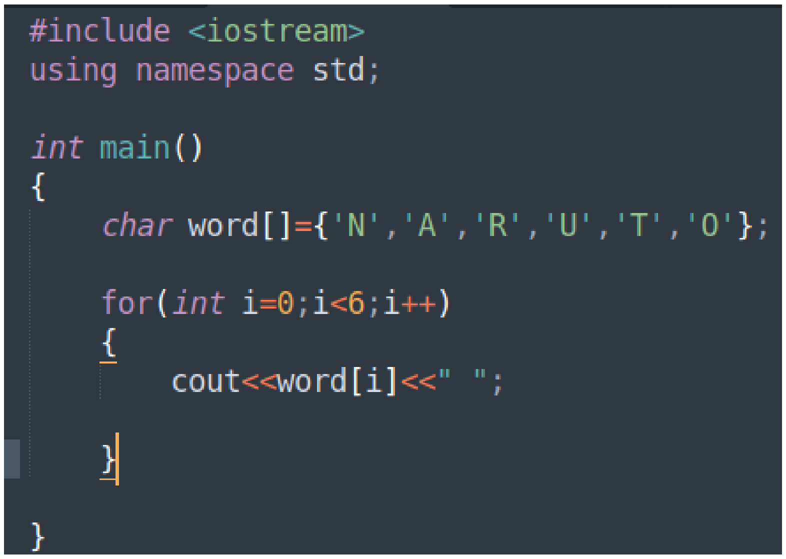
```
int mark[5] = {19, 10, 8, 17, 9}
// make the value of the third element to -1
mark[2] = -1;
// make the value of the fifth element to 0
mark[4] = 0;
```

Example 1



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

int main()
{
    int check,check_flag=0;
    cout<<"Enter the value to check if it exists:\n ";
    cin>>check;
    int value[5]={4,5,3,1,2};
    for(int i=0;i<5;i++)
    {
        if(value[i]==check)
        {

            check_flag=1;
            break;
        }
    }
    if(check_flag==1)
    {
        cout<<"\nValue exists";
    }
    else
    {
        cout<<"Value doesn't exist\n";
    }
}
```

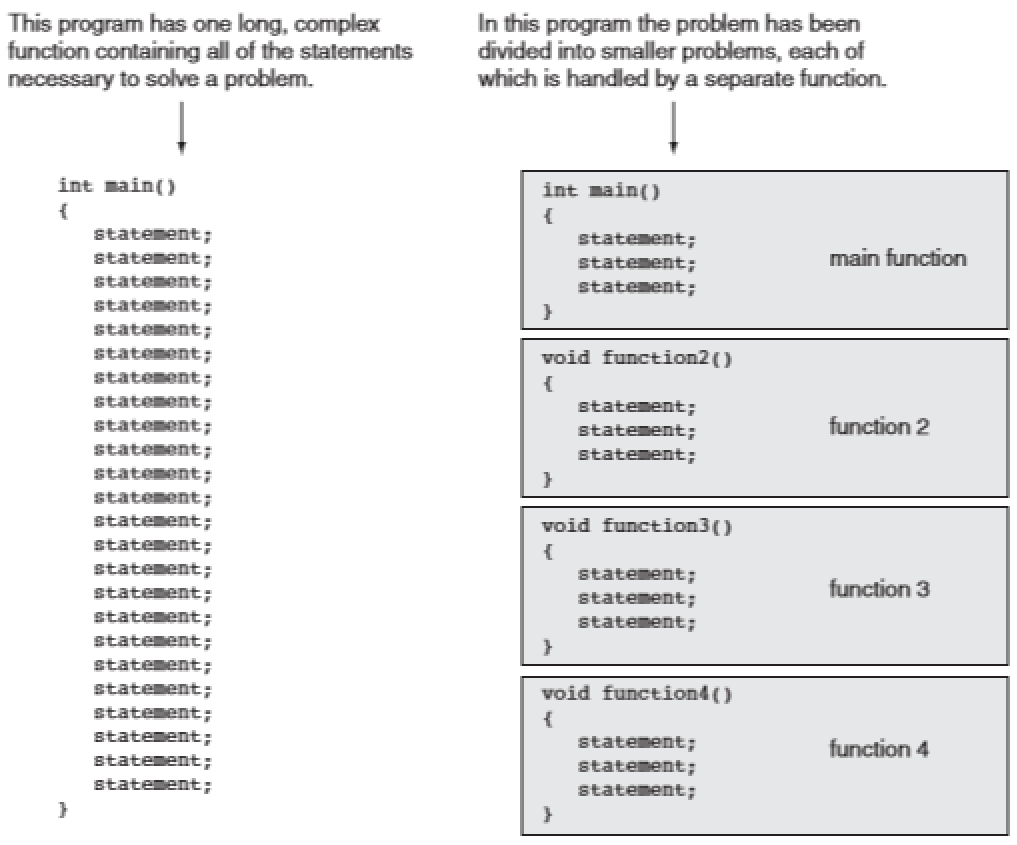
Example 2:



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

int main()
{
    char word[]={'N','A','R','U','T','O'};

    for(int i=0;i<6;i++)
    {
        cout<<word[i]<<" ";

    }

}
```

1. **Function:**

   A function is a block of code that performs a specific task. They let you divide complicated programs into manageable pieces. A function can be defined once and then called multiple times from different parts of the program. It allows for code reuse, modularity, and easier program maintenance.


This program has one long, complex function containing all of the statements necessary to solve a problem.
In this program the problem has been divided into smaller problems, each of which is handled by a separate function.
main function
function 2
function 3
function 4

```
int main()
{
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
   statement;
}
```

```
int main()
{
   statement;
   statement;
   statement;
}

void function2()
{
   statement;
   statement;
   statement;
}

void function3()
{
   statement;
   statement;
   statement;
}

void function4()
{
   statement;
   statement;
   statement;
}
```

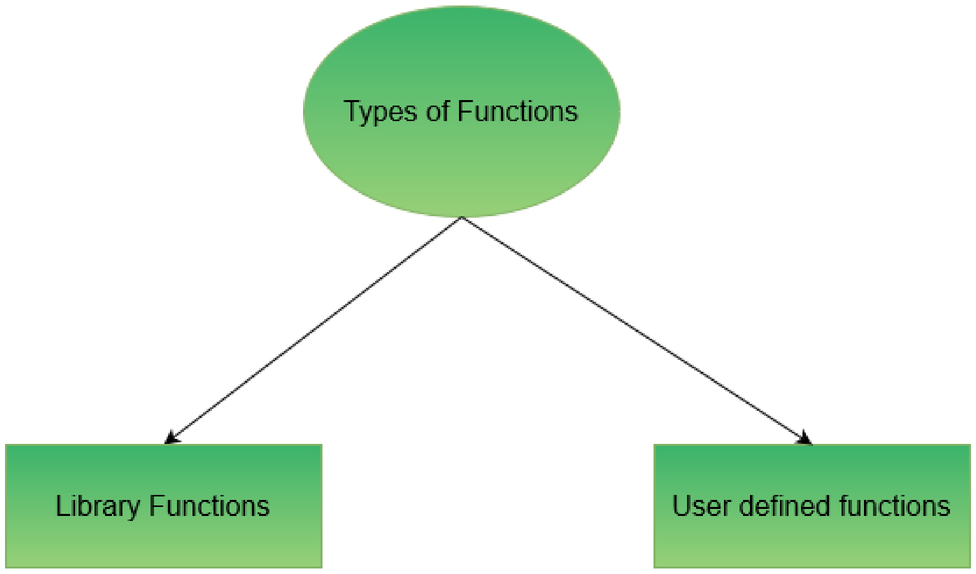
2. **Advantages of function:**
   - **Code Reusability:** Functions can be called from different parts of a program, allowing the same piece of code to be reused multiple times. . This can save time and effort in writing and maintaining code.
   - **Modularity:** Modularity: Functions allow programs to be broken down into smaller, more manageable pieces, making them easier to develop, debug, and maintain. Each function can be tested and debugged independently of the rest of the program.
   - **Efficient use of memory:** Functions can help reduce code duplication, which can help reduce the amount of memory required to store the program. This is particularly important in embedded systems or systems with limited memory.

3. **Types of functions:**

There are different types of functions in C programming language and few of them are as follows:

I. **Library Functions:** These are predefined functions that are included in the C library. **Example:** *strlen(), pow(), sqrt()* etc

II. **User-defined functions:** These are functions that are created by the programmer to perform a specific task. It reduces the complexity of a big program and optimizes the code and can be used multiple times in different part of the same program to achieve reusability.



4. **Function Definition:**

A function is defined by two main components: the ***function header*** and the ***function body.***

Function Header: The function header contains the function name, return type, and parameter list, and is usually placed at the beginning of the function definition. It has the following syntax:

return_type function_name()

OR

return_type function_name(data_type parameter1,data_type parameter 2)

Function Body:

The function body is enclosed within curly braces { } and contains the set of statements that perform the specific task of the function. It is located immediately after the function header and contains the following components:

1. **Declaration of local variables:** This involves declaring any variables that are used within the function body. These variables are only accessible within the function and are not visible outside the function.
2. **Statements to perform a specific task:** This involves writing the set of statements that perform the intended task of the function. These statements can include control structures, loops, conditional statements, and other function calls.
3. **Return statement:** If the function is designed to return a value, the return statement is used to send the computed value back to the calling function. The return statement terminates the function and sends the computed value to the calling function.

Syntax:

```
int add(int x, int y)

{        //Start of the body

int sum;        //Declaration of local variables

sum = x + y;  //Performing specific task i.e addition

return sum;   //return statement

}
```

5. **Return Value:**

A C function may or may not return a value from the function. If you don't have to return any value from the function, use void for the return type.

Let's see a simple example of C function that doesn't return any value from the function.

Example without return value:

```
void hello(){

printf("hello c");
}
```

If you want to return any value from the function, you need to use any data type such as int, long, char, etc. The return type depends on the value to be returned from the function.
Let's see a simple example of C function that returns int value from the function.

Example with return value:

```
int get(){

return 10;

}
```

In the above example, we have to return 10 as a value, so the return type is int. If you want to return floating-point value (e.g., 10.2, 3.1, 54.5, etc), you need to use float as the return type of the method.

```
float get(){

return 10.2;

}
```

Now, you need to call the function, to get the value of the function.

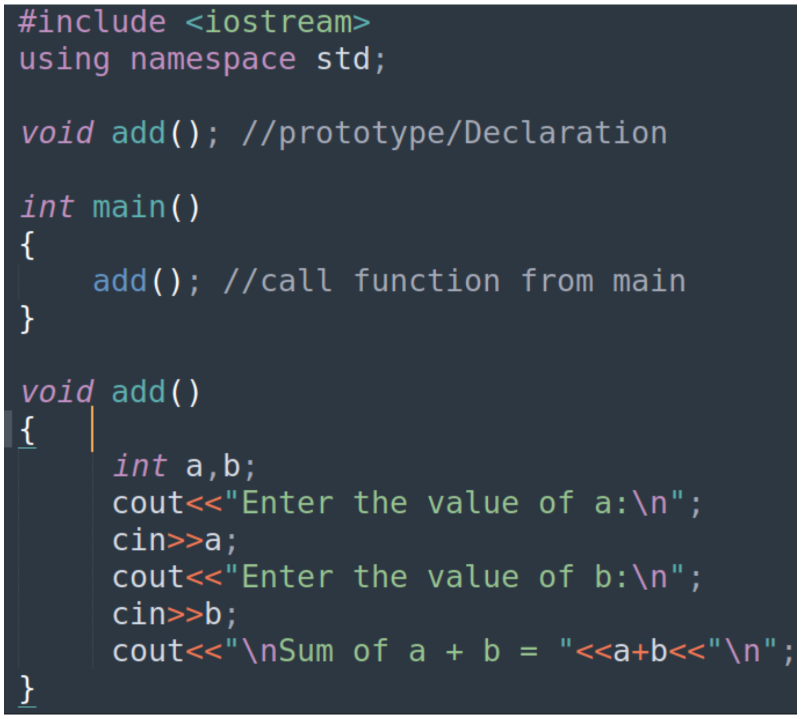
6. **Calling a function:**

   A function may or may not accept any argument. It may or may not return any value. Based on these facts, There are four different aspects of function calls.

1. function without arguments and without return value
2. function with arguments and without return value
3. function without arguments and with return value
4. function with arguments and with return value

1. **Function without arguments and without return value**

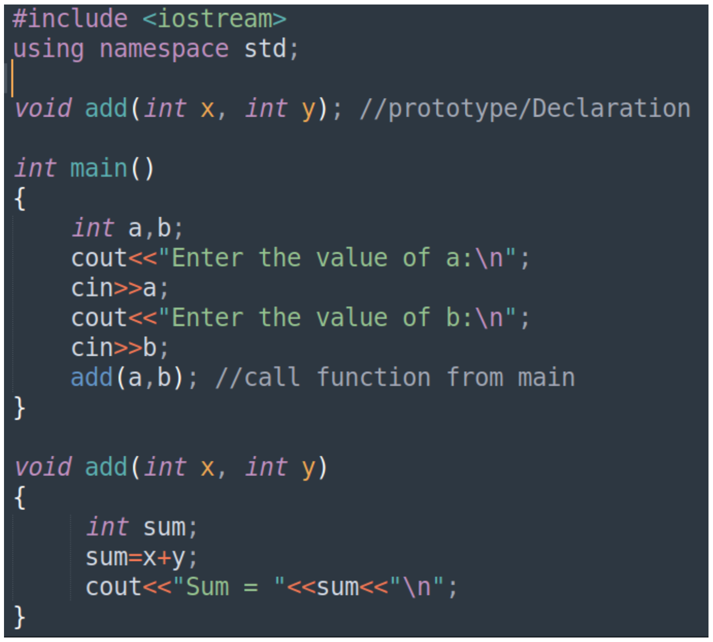
   Example 1:



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

void add(); //prototype/Declaration

int main()
{
    add(); //call function from main
}

void add()
{
    int a,b;
    cout<<"Enter the value of a:\n";
    cin>>a;
    cout<<"Enter the value of b:\n";
    cin>>b;
    cout<<"\nSum of a + b = "<<a+b<<"\n";
}
```

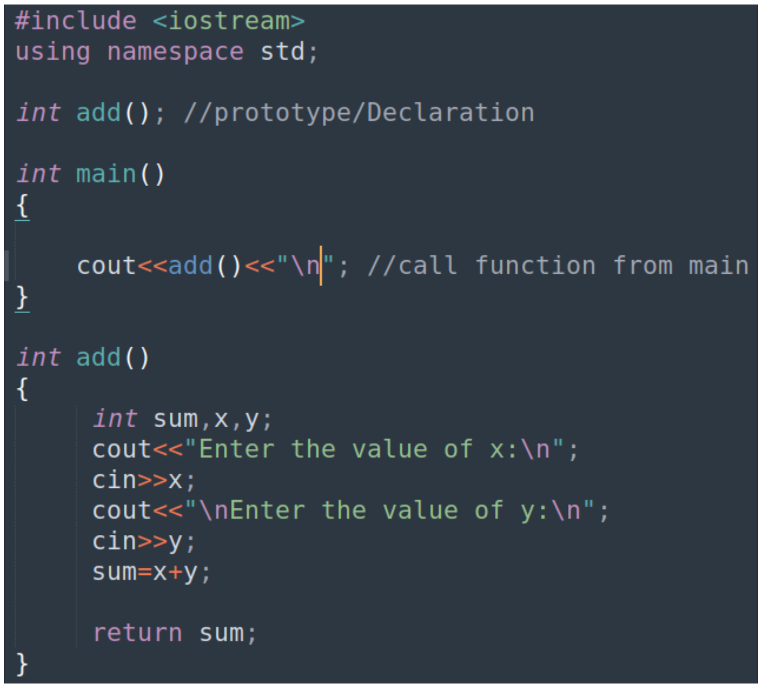
2. **Function with arguments and without return value**

Example 2:



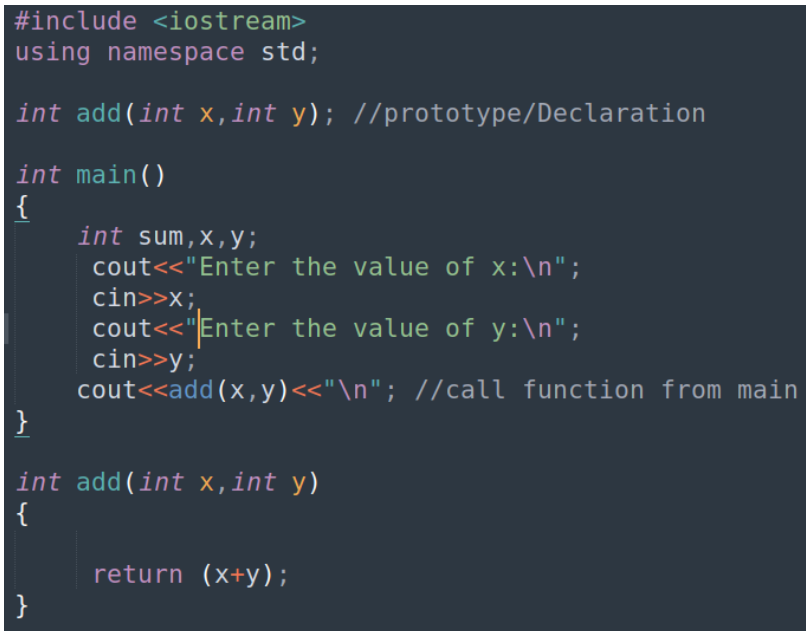
```cpp
#include <iostream>
using namespace std;

void add(int x, int y); //prototype/Declaration

int main()
{
    int a,b;
    cout<<"Enter the value of a:\n";
    cin>>a;
    cout<<"Enter the value of b:\n";
    cin>>b;
    add(a,b); //call function from main
}

void add(int x, int y)
{
     int sum;
     sum=x+y;
     cout<<"Sum = "<<sum<<"\n";
}
```

3. **Function without arguments and with return value**
   Example 3:



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

int add(); //prototype/Declaration

int main()
{
    cout<<add()<<"\n"; //call function from main
}

int add()
{
     int sum,x,y;
     cout<<"Enter the value of x:\n";
     cin>>x;
     cout<<"\nEnter the value of y:\n";
     cin>>y;
     sum=x+y;

     return sum;
}
```

4. **Function with arguments and return value**
   Example 4:

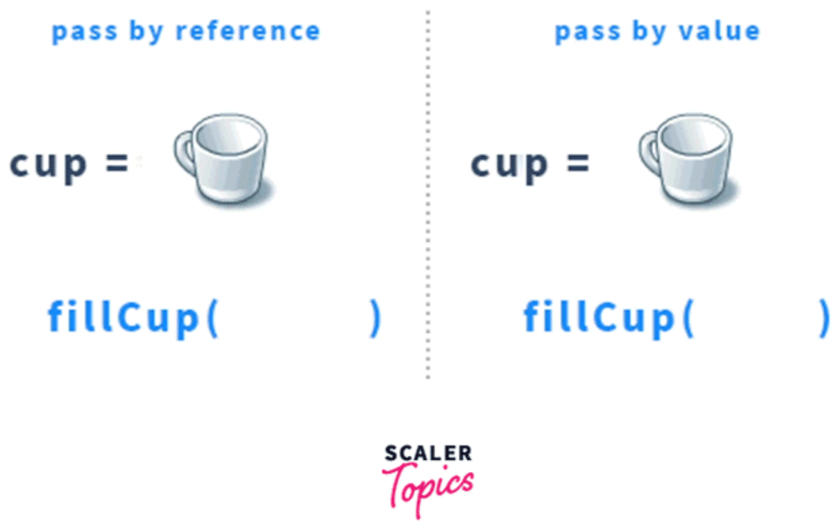


```cpp
#include <iostream>
using namespace std;

int add(int x,int y); //prototype/Declaration

int main()
{
    int sum,x,y;
     cout<<"Enter the value of x:\n";
     cin>>x;
     cout<<"Enter the value of y:\n";
     cin>>y;
    cout<<add(x,y)<<"\n"; //call function from main
}

int add(int x,int y)
{

     return (x+y);
}
```

5. **Pass by reference:**

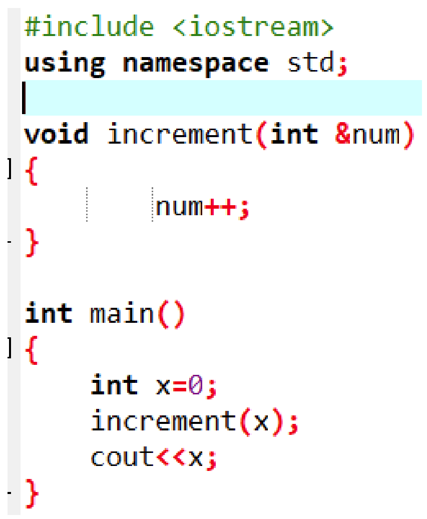
   Previously we passed a parameter to a function by value. However, In this lab we will see how a parameter is passed by reference.

   When you pass a parameter by value, a copy of the parameter's value is passed to the function, and any changes made to the parameter within the function have no effect on the original value outside the function. However, when you pass a parameter by reference, you pass a reference to the parameter's memory address instead of a copy of its value. This allows the function to access and modify the original value of the parameter.



https://blog.penjee.com/wp-content/uploads/2015/02/pass-by-reference-vs-pass-by-value-animation.gif

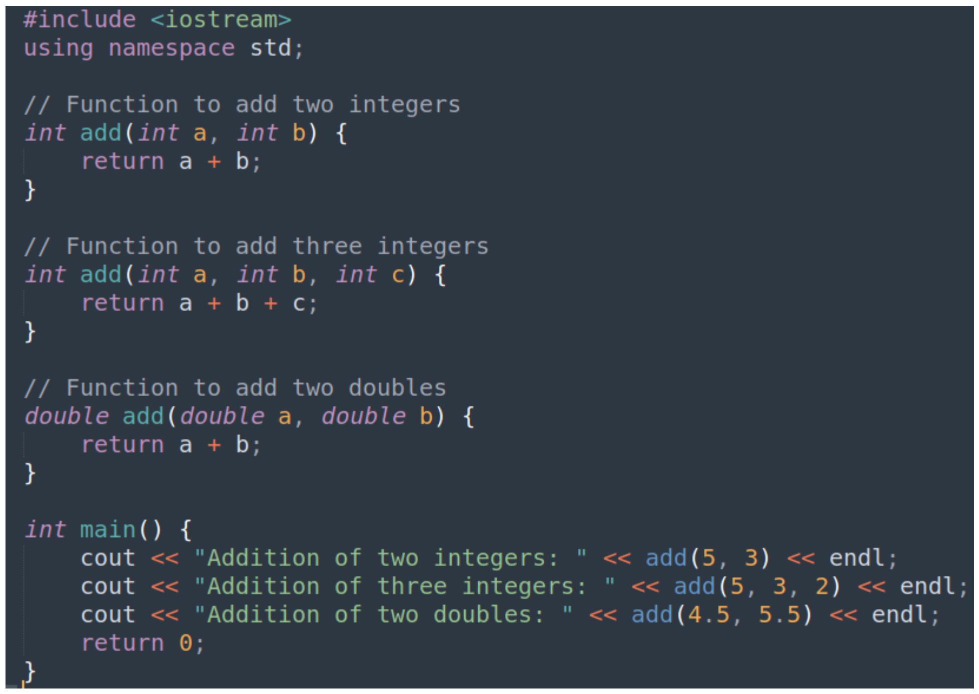
Example 5:



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

void increment(int &num)
{
        num++;
}

int main()
{
    int x=0;
    increment(x);
    cout<<x;
}
```

6. **Function Overloading:**
    - Function overloading in C++ is a feature that allows you to define multiple functions with the same name but different parameter lists (either in number, type, or both).
    - The compiler differentiates these functions based on their signatures (the number and types of parameters), allowing the same function name to be used for different tasks.

Example 6:



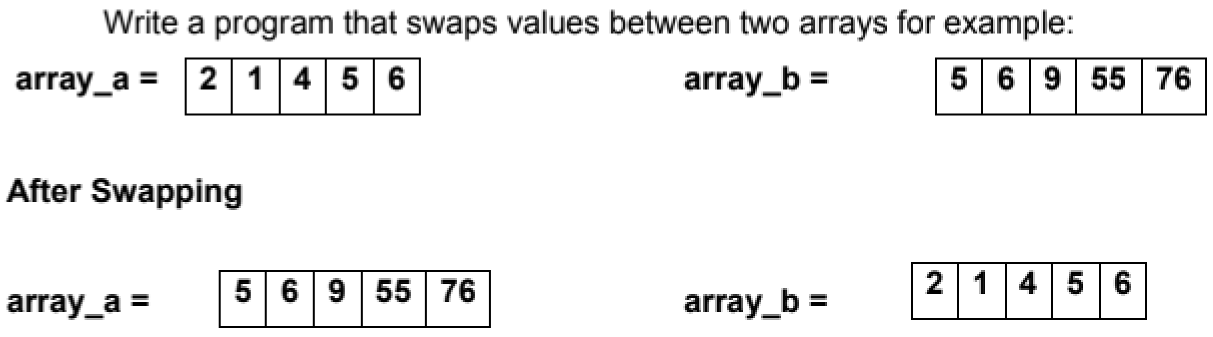
```cpp
#include <iostream>
using namespace std;

// Function to add two integers
int add(int a, int b) {
    return a + b;
}

// Function to add three integers
int add(int a, int b, int c) {
    return a + b + c;
}

// Function to add two doubles
double add(double a, double b) {
    return a + b;
}

int main() {
    cout << "Addition of two integers: " << add(5, 3) << endl;
    cout << "Addition of three integers: " << add(5, 3, 2) << endl;
    cout << "Addition of two doubles: " << add(4.5, 5.5) << endl;
    return 0;
}
```

NOTE: **Use functions to solve the problem**

Tasks:

1. **Use a one-dimensional array to solve the following problem. Read in 25 numbers, each of which is between 10 and 100, inclusive. As each number is read, validate it and store it in the array only if it isn't a duplicate of a number already read. After reading all the values, display only the unique values that the user entered.**

2.



Write a program that swaps values between two arrays for example:

array_a =

2	1	4	5	6

array_b =

5	6	9	55	76

After Swapping

array_a =

5	6	9	55	76

array_b =

2	1	4	5	6

3. **A car rental company charges a base fee of $25.00 per day, plus $0.15 per mile driven. Write a function that takes the number of days the car is rented and the number of miles driven as parameters, and returns the total cost of the rental. If the number of days is more than 30 the user gets a 20% discount, however, if the days are less than 15 extra $1.99 is charged per day.**

4. **Scenario:**

You are tasked with creating a program to manage a small student database for a school. The school wants to store the following information about each student:

- **Name**
- **Roll Number**
- **Marks in 3 subjects**

The system should perform the following tasks:

1. **Add new student records.**
2. **Display all student records.**

3. **Calculate and display the average marks of each student.**
4. **Find the student with the highest total marks.**

Procedure: Solving with Procedural Programming

Step 1: Break Down the Problem

- **Use arrays to store student information.**
  - **A 2D array for marks (e.g., marks[10][3] for 10 students and 3 subjects).**
  - **Separate arrays for names and roll numbers.**
- **Use functions to:**
  - **Input and store student data.**
  - **Calculate averages.**
  - **Find the student with the highest total marks.**
- **Iterate through the arrays for all operations.**