

EXPERIMENT NO. 04

DATE OF PERFORMANCE:

GRADE:

DATE OF ASSESSMENT:

SIGNATURE OF LECTURER/ TTA:

AIM: To Study Functions in C++.

THEORY:

FUNCTION:

A function is a group of statements that together perform a task. Every C++ program has at least one function, which is `main()`, and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division usually is so each function performs a specific task.

A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

The C++ standard library provides numerous built-in functions that your program can call. For example, function `strcat()` to concatenate two strings, function `memcpy()` to copy one memory location to another location and many more functions.

A function is known as with various names like a method or a sub-routine or a procedure etc.

Defining a Function:

The general form of a C++ function definition is as follows:

```
return_type function_name( parameter list )  
{  
    body of the function
```

}

A C++ function definition consists of a function header and a function body. Here are all the parts of a function:

- **Return Type:** A function may return a value. The `return_type` is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the `return_type` is the keyword `void`.
- **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body:** The function body contains a collection of statements that define what the function does.

Example:

Following is the source code for a function called `max()`. This function takes two parameters `num1` and `num2` and returns the maximum between the two:

```
// function returning the max between two numbers  
int max(int num1, int num2)  
{  
    // local variable declaration  
    int result;
```

```
if (num1 > num2)
    result = num1;
else
    result = num2;
return result;
}
```

Function Declarations:

A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts:

```
return_type function_name( parameter list );
```

For the above defined function max(), following is the function declaration:

```
int max(int num1, int num2);
```

Parameter names are not important in function declaration only their type is required, so following is also valid declaration:

```
int max(int, int);
```

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

Calling a Function:

While creating a C++ function, you give a definition of what the function has to do. To use a function, you will have to call or invoke that function.

When a program calls a function, program control is transferred to the called function. A called function performs defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.

To call a function, you simply need to pass the required parameters along with function name, and if function returns a value, then you can store returned value. For example:

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

// function declaration
int max(int num1, int num2);
int main ()
{
    // local variable declaration:
    int a = 100;
    int b = 200;
    int ret;

    // calling a function to get max value.
    ret = max(a, b);
    cout << "Max value is : " << ret << endl;
    return 0;
}

// function returning the max between two numbers
int max(int num1, int num2)
{
    // local variable declaration
    int result;
```

```

if (num1 > num2)
    result = num1;
else
    result = num2;

return result;
}

```

I kept max() function along with main() function and compiled the source code. While running final executable, it would produce the following result:

Max value is : 200

Function Arguments:

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the formal parameters of the function.

The formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit. While calling a function, there are two ways that arguments can be passed to a function:

Call Type	Description
<u>Call by value</u>	This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
<u>Call by pointer</u>	This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.
<u>Call by reference</u>	This method copies the reference of an argument into the formal parameter. Inside the function, the reference is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

By default, C++ uses call by value to pass arguments. In general, this means that code within a function cannot alter the arguments used to call the function and above mentioned example while calling max() function used the same method.

Default Values for Parameters:

When you define a function, you can specify a default value for each of the last parameters. This value will be used if the corresponding argument is left blank when calling to the function.

This is done by using the assignment operator and assigning values for the arguments in the function definition. If a value for that parameter is not passed when the function is called, the default given value is used, but if a value is specified, this default value is ignored and the passed value is used instead. Consider the following example:

```
#include <iostream>
using namespace std;
int sum(int a, int b=20)
{
    int result;

    result = a + b;

    return (result);
}
int main ()
{
    // local variable declaration:
    int a = 100;
    int b = 200;
    int result;

    // calling a function to add the values.
```

```
result = sum(a, b);  
cout << "Total value is :" << result << endl;  
// calling a function again as follows.  
result = sum(a);  
cout << "Total value is :" << result << endl;  
return 0;  
}
```

When the above code is compiled and executed, it produces the following result:

Total value is :300

Total value is :120

inline function:

C++ inline function is powerful concept that is commonly used with classes. If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.

Any change to an inline function could require all clients of the function to be recompiled because compiler would need to replace all the code once again otherwise it will continue with old functionality.

To inline a function, place the keyword inline before the function name and define the function before any calls are made to the function. The compiler can ignore the inline qualifier in case defined function is more than a line.

A function definition in a class definition is an inline function definition, even without the use of the inline specifier.

Function overloading in C++:

You can have multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. You can not overload function declarations that differ only by return type.

PROGRAM 1: User defined Function to add Two Number.

```
#include <iostream>
```

```
using namespace std;
```

```
int add(int, int); // Function prototype (declaration)
```

```
int main()
```

```
{
```

```
    int num1, num2, sum;
```

```
    cout << "Enter two numbers to add: ";
```

```
    cin >> num1 >> num2;
```

```
    sum = add(num1, num2); // Function call
```

```
    cout << "Sum = " << sum;
```

```
    return 0;
```

```
}
```

```
int add(int a, int b) // Function definition
```

```
{
```

```
    int add;
```

```
    add = a + b;
```

```
    return add; // Return statement
```



```
}
```

OUTPUT:

Enter two numbers to add: 5 8

Sum = 13

PROGRAM 2: Inline Function to find the multiplication values and the cubic values.

```
#include <iostream>
```

```
using namespace std;
```

```
class line
```

```
{
```

```
public:
```

```
    inline float mul(float x, float y)
```

```
    {
```

```
        return (x * y);
```

```
    }
```

```
    inline float cube(float x)
```

```
    {
```

```
        return (x * x * x);
```

```
    }
```

```
};
```

```
int main()
```

```
{
```

```
line obj;  
float val1, val2;  
cout << "Enter two values: ";  
cin >> val1 >> val2;  
cout << "\nMultiplication value is: " << obj.mul(val1, val2);  
cout << "\n\nCube value is: " << obj.cube(val1) << "\t" << obj.cube(val2);  
return 0;  
}
```

OUTPUT:

Enter two values: 2.5 3.2

Multiplication value is: 8

Cube value is: 15.625 32.768

PROGRAM 3: Find the sum of the given numbers using default argument declaration.

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

```
void sum(int a, int b, int c = 6, int d = 10);
```

```
int main()  
{  
    int a, b;  
    cout << "Enter any two numbers: ";  
    cin >> a >> b;  
    sum(a, b); // sum of default values  
    return 0;  
}
```

```
void sum(int a1, int a2, int a3, int a4)  
{  
    int temp;  
    temp = a1 + a2 + a3 + a4;  
    cout << "a = " << a1 << endl;  
    cout << "b = " << a2 << endl;  
    cout << "c = " << a3 << endl;  
    cout << "d = " << a4 << endl;  
    cout << "sum = " << temp;  
}
```

OUTPUT:

Enter any two numbers: 5 8

a = 5

b = 8

c = 6

d = 10

sum = 29

PROGRAM 4: Function overloading in C++

```
#include <iostream>
```

```
using namespace std;
```

```
class printData
```

```
{
```

```
public:
```

```
    void print(int i)
```

```
    {
```

```
        cout << "Printing int: " << i << endl;
```

```
    }
```

```
    void print(double f)
```

```
    {
```

```
        cout << "Printing float: " << f << endl;
```

```
    }
```

```
    void print(char* c)
```

```
    {
```

```
        cout << "Printing character: " << c << endl;
```

```
    }
```

```
};
```

```
int main()
{
    printData pd;

    // Call print to print integer
    pd.print(5);

    // Call print to print float
    pd.print(500.263);

    // Call print to print character
    pd.print("Hello C++");

    return 0;
}
```

OUTPUT:

Printing int: 5

Printing float: 500.263

Printing character: Hello C++

PROGRAM 5: Use of pow function.

```
#include <math.h>
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main()  
{  
    for (int i = 1; i < 5; i++)  
        cout << "Power of pow(3.2, " << i << "): " << pow(3.2, i) << endl;  
    return 0;  
}
```

OUTPUT:

Power of pow(3.2, 1): 3.2

Power of pow(3.2, 2): 10.24

Power of pow(3.2, 3): 32.768

Power of pow(3.2, 4): 104.8576

PROGRAM 6: Swap Two Variable using call by reference.

```
#include <iostream>
```

```
using namespace std;
```

```
void swap(int& first, int& second)
```

```
{
```

```
    int temp = first;
```

```
    first = second;
```

```
    second = temp;
```

```
}
```

```
int main()
```

```
{
```

```
    int a = 2, b = 3;
```

```
    swap(a, b);
```

```
    cout << a << " " << b;
```

```
    return 0;
```

```
}
```

OUTPUT:

3 2