Unit 3

Functions in C++

A function is a block of code that performs a specific task. There are two types of function:

Standard Library Functions: Predefined in C++

User-defined Function: Created by users

C++ User-defined Function

C++ allows the programmer to define their own function.

A user-defined function groups code to perform a specific task and that group of code is given a name (identifier).

When the function is invoked from any part of the program, it all executes the codes defined in the body of the function.

C++ Function Declaration

The syntax to declare a function is:

```
returnType functionName (parameter1, parameter2,...) {
    // function body
}
```

Here's an example of a function declaration.

```
// function declaration

void greet() {

   cout << "Hello World";

}
```

Here,

- the name of the function is greet()
- the return type of the function is void
- the empty parentheses mean it doesn't have any parameters
- the function body is written inside {}

- Calling a Function

In the above program, we have declared a function named greet(). To use the greet() function, we need to call it.

Here's how we can call the above greet() function.

```
int main() {
        // calling a function
   greet();
}
```

example:

```
#include <iostream>
using namespace std;
// declaring a function
void greet() {
   cout << "Hello there!";
}
int main() {
   // calling the function
   greet();
   return 0;
}
```

ex.

```
#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;
}
```

Example 2: Function with Parameters

```
// program to print a text
```

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

// display a number
void displayNum(int n1, float n2) {
   cout << "The int number is " << n1;
   cout << "The double number is " << n2;
}

int main() {

    int num1 = 5;
    double num2 = 5.5;

   // calling the function
   displayNum(num1, num2);

   return 0;
}
```

Call by value in C++

In call by value, **original value is not modified.**

In call by value, value being passed to the function is locally stored by the function parameter in stack memory location. If you change the value of function parameter, it is changed for the current function only. It will not change the value of variable inside the caller method such as main().

Example 3: Add Two Numbers

```
// program to add two numbers using a function

#include <iostream>

using namespace std;

// declaring a function
int add(int a, int b) {
   return (a + b);
```

```
}

int main() {

    int sum;

    // calling the function and storing
    // the returned value in sum
    sum = add(100, 78);

    cout << "100 + 78 = " << sum << endl;

    return 0;
}
```

- **Function Prototype**

In C++, the code of function declaration should be before the function call. However, if we want to define a function after the function call, we need to use the function prototype. For example,

```
// function prototype
void add(int, int);
int main() {
    // calling the function before declaration.
    add(5, 3);
    return 0;
}

// function definition
void add(int a, int b) {
    cout << (a + b);
}
```

In the above code, the function prototype is:

```
void add(int, int);
```

This provides the compiler with information about the function name and its parameters. That's why we can use the code to call a function before the function has been defined.

The syntax of a function prototype is:

```
returnType  functionName(dataType1, dataType2, ...);
```

Call by reference in C++

In call by reference, original value is modified because we pass reference (address).

Here, address of the value is passed in the function, so actual and formal arguments share the same address space. Hence, value changed inside the function, is reflected inside as well as outside the function.

example:

```
#include<iostream>
using namespace std;
void swap(int *x, int *y)
{
 int swap;
 swap=*x;
 *x=*y;
 *y=swap;
}
int main()
{
 int x=500, y=100;
 swap(&x, &y);  // passing value to function
 cout<<"Value of x is: "<<x<<endl;
 cout<<"Value of y is: "<<y<<endl;
 return 0;
}
```

Output:

```
Value of x is: 100
Value of y is: 500
```

No.	Call by value	Call by reference
1	A copy of value is passed to the function	An address of value is passed to the function
2	Changes made inside the function is not reflected on other functions	Changes made inside the function is reflected outside the function also
3	Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in same memory location

Inline Functions in C++

C++ provides inline functions to reduce the function call overhead. An inline function is a function that is expanded in line when it is called. When the inline function is called whole code of the inline function gets inserted or substituted at the point of the inline function call. This substitution is performed by the C++ compiler at compile time. An inline function may increase efficiency if it is small.

Syntax:

```
inline return-type function-name(parameters)
{
   // function code
}
```

Inline functions Advantages:

1. Function call overhead doesn't occur.
2. It also saves the overhead of push/pop variables on the stack when a function is called.
3. It also saves the overhead of a return call from a function.
4. When you inline a function, you may enable the compiler to perform context-specific optimization on the body of the function. Such optimizations are not possible for normal function calls. Other optimizations can be obtained by considering the flows of the calling context and the called context.
5. An inline function may be useful (if it is small) for embedded systems because inline can yield less code than the function called preamble and return.

Example:

```
#include <iostream>
using namespace std;
inline int cube(int s) { return s * s * s; }
int main()
{
        cout << "The cube of 3 is: " << cube(3) << "\n";
        return 0;
}
```

Example:

```
// C++ Program to demonstrate inline functions and classes
#include <iostream>

using namespace std;

class operation {
        int a, b, add, sub, mul;
        float div;

public:
        void get();
        void sum();
        void difference();
        void product();
        void division();
};
inline void operation ::get()
{
        cout << "Enter first value:";
        cin >> a;
        cout << "Enter second value:";
        cin >> b;
}

inline void operation ::sum()
{
        add = a + b;
        cout << "Addition of two numbers: " << a + b << "\n";
}

inline void operation ::difference()
```

```
{
        sub = a - b;
        cout << "Difference of two numbers: " << a - b << "\n";
}

inline void operation ::product()
{
        mul = a * b;
        cout << "Product of two numbers: " << a * b << "\n";
}

inline void operation ::division()
{
        div = a / b;
        cout << "Division of two numbers: " << a / b << "\n";
}

int main()
{
        cout << "Program using inline function\n";
        operation s;
        s.get();
        s.sum();
        s.difference();
        s.product();
        s.division();
        return 0;
}
```