

# Functions in C++

Branch:S6 ECE

Faculty: SREEDIVYA

# Functions

- A function is a block of code that performs some operation
- Dividing a program into functions is one of the major principles of top – down, structured programming.
- It is possible to reduce the size of the program by calling and using them at different places in the program
- C++ has added many new features to functions to make them more reliable and flexible
- C++ functions can be overloaded to make it perform different tasks depending on the arguments passed to it

# Structure of a function

```
void display();           /*Function Declaration*/
void main()
{
    -----
    display();           /*Function call*/
    -----
}
void display()           /*Function Definition*/
{
    -----
    -----
}
```

# Function Prototyping/Declaration

- Provides the compiler with details such as the number and type of arguments and the type of return values.
- Syntax:

`type function_name(argument_list);`

eg: `float volume(int x, float y, float z);`

# Function Definition

- The actual operation of the function is specified in the function definition.
- The arguments inside function definition are known as **formal parameters**
- Formal parameters are just a reference or copy of the actual values

# Function call

- It is an expression that passes control and arguments to a function
- The arguments(or variables) inside a function call are known as **actual parameters**
- Actual parameters represent the real values that are to be passed on to the function.
- There are two ways to call a function
  1. Call by value
  2. Call by reference

# Example(Sample program illustrating use of Function)

```
#include <iostream>
using namespace std;
int add(int, int);
int main() {
    int x, y, sum;
    cout << "Enter two numbers: ";
    cin >> x >> y;
    sum = add(x, y);
    cout << "The sum of "<<x<< " and " <<y<<" is: "<<sum;
    return 0;
}
int add(int num1, int num2) {
    int add;
    add = num1 + num2;
    return add;
}
```

# Call by value

- Actual values in the call are passed to the function definition
- Disadvantages: Copying overhead and no alteration can be made



# Call by reference

- Instead of passing values, the memory location is passed.

Function definition be,

```
void swap(int &a, int &b)
{
    int t=a;
    a=b;
    b=t;
}
```

Function call be,     swap(m,n);

# Return by reference

- A function can also return a reference
- Function definition be

```
int &max(int &x,int &y)
```

```
{
```

```
    if(x>y)
```

```
        return x;
```

```
    else
```

```
        return y;
```

```
}
```

# Inline functions

- These functions are expanded in line when it is invoked. i.e., compiler replaces the function call with the corresponding code.

## Syntax:

```
inline function header  
{  
    function body;  
}
```

## Example:

```
inline double cube(double a)  
{  
    return (a*a*a);  
}
```

This can be invoked by `c=cube(3.0);`

# Features of inline functions

- Inline functions must be defined before they are called.
- Functions are made inline when they are small enough to be defined in one or two lines
- Consumes more memory
- Inline functions may not work,
  - a) If they are recursive
  - b) If they contain static variables
  - c) For function returning values; if a loop, a switch or a goto exists.
  - d) For functions not returning values; if a return statement exists

# Default Arguments

- C++ allows to call a function without specifying all its arguments.
- In that case the function assigns a default value to the parameter which does not have a matching argument in the function call.
- Default values are specified when the function is declared.

Eg:

```
int date(int day, int month, int year=2021);
```

Function call:     calendar= set(20,12);  
                  calendar= set(20,12,2023)

# Function overloading in C++

- Two or more functions can have the same name but different parameters
- The number and type of arguments will be different
- When a function name is overloaded with different jobs it is called function overloading
- Function overloading can be considered as an example of polymorphism in C++
- This improves the readability of the program

# Example 1

```
#include<iostream>
using namespace std;
class A
{
    int num1=20,num2=10;
    public:
        void fun()
        {
            int sum=num1+num2;
            cout<<"Addition  "<<sum<<"\n";
        }
}
```

```
void fun(int a, int b)
```

```
{
```

```
    int sub=a-b;
```

```
    cout<<"Subtraction  " <<sub<<"\n";
```

```
}
```

```
};
```

```
int main()
```

```
{
```

```
    A obj;
```

```
    obj.fun();
```

```
    obj.fun(100,50);
```

```
    return 0;
```

```
}
```



## OUTPUT

Addition 30

Subtraction 50

## Example 2

```
#include<iostream>
using namespace std;
class Addition
{
    public:
        int sum(int a, int b)
        {
            return (a+b);
        }
}
```

```
int sum(int a, int b, int c)
```

```
{
```

```
    return (a+b+c);
```

```
}
```

```
};
```

```
int main()
```

```
{
```

```
    Addition obj;
```

```
    cout<<obj.sum(20,15)<<"\n";
```

```
    cout<<obj.sum(81,100,10);
```

```
    return 0;
```

```
}
```

## OUTPUT

35

191