Functions in C++

Prepared by Deepali Sonawane, MIT-WPU, Pune

Introduction

- Functions are a basic **building block** for writing CPP programs.
- Breaking a program into a **separate functions**, each of which performs a **particular task,** makes it easier **to develop and debug** a program.
- There are **two types** of function
  - Built in function / Predefined function
  - User defined function
- Examples of Built in functions:

  get(), put(), getline() etc.

Advantages of function

- Functions allow for **braking down** the program into discrete units.
- Programs that use functions are **easier to design**, debug and maintain.
- It is possible to perform separate **compilation** of functions.
- Functions can **return data** via arguments and can return a value.
- Functions have **local variables** and have access to **global variables.**
- Different programmers working on one large project can divide the workload by **writing different functions**.

main() function

- The **main() function** is the starting point for every CPP program.
- The operating system calls this function to **begin program execution**.
- It must have a return type of **int** to indicate the program's exit status to the operating system, with **0** typically indicating **successful execution**.
- Example:

```
int main()
{
        ……….
        return 0;
}
```

User Defined Function

- Besides the built-in functions, it is possible to create our **own functions**.
- A function will be executed by a **call to the function**.
- Three steps to write user defined function:
  - Function declaration
  - Function definition
  - Function call
- **Syntax of function declaration**:
  return_type function_name([list of parameters]);
- **Syntax of function definition :**
  return_type function_name([list of parameters])
  {
  //function body
  }
- **Syntax for function Call:**
  function_name([list of parameters]);

Types of functions

- **Non parameterized function** : Functions **without parameters** are called as **Non parameterized** function.

  Example : void display();

- **Parameterized function:** Functions **with the parameters** are called as **parameterized function**.

  Example : float area(float r);

Types of Parameters

- **There are two types of parameters**

1. **Actual Parameters :** These parameters are used in function call, that contains actual values of variables.

   Example : addition(a,b)

2. **Formal or Dummy Parameters :** These parameters are used in function definition.

   Example : void addition(int c, int d)

Default arguments

- While declaring a function, we can specify a **default value** for each **parameter.**
- This value will be used if that **parameter is left blank** when calling to the function.
- To do that we simply have **to assign a value** to the arguments in the function declaration.
- Default values are assigned from right to left.
- **Example:** void addition(int a, int b=1);

const arguments

- A function parameters and return type of function can be declared as **constant**.
- **Constant values** cannot be changed as any such attempt will generate a compile-time error.
- **Example**: void addition(const int a, const int b);

Methods of passing parameters to a function

There are two methods **to pass parameters** to the function:

- **Call by value** : If parameters are passed by value, only a copy of the variables has been passed to the function. Any changes to the value will **not be reflected back** to the calling function.

  Example: void swap(int a, int b);

- **Call by reference :** A function that uses call by reference arguments (reference variables) **reflects any changes to the values** of the arguments to the calling function.

  Example: void swap(int &a, int &b);

  We can also use **pointers** for call by reference.

Function with returning value

- **return statement** is used to return a value from the function.
- **Example:**

```
int multiplication(int a, int b)
{
        int c;
        c= a*b;
        return c;
}
```

Function overloading

- **Function Overloading** is a process in which a function with the same name as another function and have **different parameter types** or **different number of parameters**.
- You can give **same name** to **more than one function** if they have either a **different number of arguments** or **different types in their arguments.**
- This promotes **programming flexibility**.
- Example: int addition(int x, int y);

  double addition(double x,double y);

  int addition(int x, int y, int z);

Recursive Function

- **Recursive Function** : It is a function **which calls itself**.
- **Recursion** : It is a **process** in which **function calls itself** from its body.
- **Advantages of recursive function**:
  i. **Reduce unnecessary** calling of function.
  ii. Through Recursion one can **solve problems in easy way** while its iterative solution is very big and complex.
  iii. Recursion uses **stack** to store data so that we **get previous value** of a variable.
- **Disadvantages of recursive function**:
  i. Recursive solution is always logical and it is very **difficult to trace.**(debug and understand).
  ii. In recursive function we must have an **if statement** somewhere to force the function to **return value**.
  iii. Recursion uses **more processor time**.
  iv. Recursion takes a **lot of stack space**, usually not considerable when the program is small.

Recursive Function

- Example:

```
int factorial(int n)
{
        if (n= =1 || n= =0)
        return 1;
        else
        return n * factorial(n-1);
}
```

Inline Function

- Functions can be instructed to compiler to make them inline so that compiler can replace those **function definition** wherever those are being called.
- **Compiler replaces the definition of inline functions at compile time instead of referring function definition at runtime**.
- To make an **inline** function, the keyword, “inline” precedes the **function prototype** and **function definition**.
- Its advantage is only appreciated in **very short functions**, in which the resulting code from compiling the program may be faster.

Inline Function

- Example:

```
inline int area(int l, int b)
{
        return l * b;
}
a = area(10,20);
cout<<“\n Area of rectangle is ”<<a;
```

Math Library Function

In order to use Maths functions, we need to include a header file <math.h> or <cmath>.

1. abs(x) : Returns the absolute value of x
2. sqrt(x) : Returns the square root of x
3. cbrt(x) : Returns the cube root of x
4. ceil(x) : Returns the value of x rounded up to its nearest integer
5. floor(x) : Returns the value of x rounded down to its nearest integer
6. round(x) : Returns x rounded to the nearest integer
7. pow(x, y) : Returns the value of x to the power of y
8. remainder(x, y) : Return the remainder of x/y rounded to the nearest integer

Math Library Function

9. log(x) : Returns the natural logarithm of x
10. log10(x) : Returns the base 10 logarithm of x
11. sin(x) : Returns the sine of x (x is in radians)
12. cos(x) : Returns the cosine of x (x is in radians)
13. tan(x) : Returns the tangent of x (x is in radians)
14. trunc(x) : Returns the integer part of x

Thank you