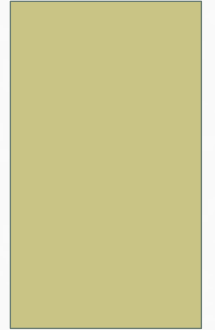


FUNCTION IN C++



CONTENT

- Function meaning
- Function definition
- Function Calling
- Function recursion
- Function overloading
- Function Overriding

FUNCTION

- A complex problem is often easier to solve by **dividing it into several smaller parts**, each of which can be solved by itself.
- **These parts are sometimes made into *functions* in C++.**
- *Functions* are **sections of code separate from the main program.**
- These code sections are **called (executed) when needed to perform specific actions in a program.**
- In c++ the main() it self is a function that means the main function is invoking the other functions to perform various tasks

WHY FUNCTION ???

- Functions **separate the concept (what is done) from the implementation (how it is done).**
- Functions make programs **easier to understand**
- Functions can be **called several times** in the same program, allowing the code to be reused.
- **Easier to debug, maintain, design and modify** etc..
 - Easy to write a correct small functions
 - Easy to read, write and debug a function
 - Easier to maintain or modify such a function

TYPES OF FUNCTION

- **Pre-defined or built-in**
 - **Is a function which is defined in External header file**
 - **To use predefined functions in a program, you must include the header file that contains the function's specification via the include statement.**
 - Have predefined task.
- **User- defined**
 - Are functions which are **declared by the programmer**
 - **The programmer is expected to declare and specify its body in detail in the program**

EXAMPLE ON PREDEFINED FUNCTION

Method	Description	Example
ceil(x)	rounds x to the smallest integer not less than x	ceil(9.2) is 10.0 ceil(-9.8) is -9.0
cos(x)	trigonometric cosine of x (x in radians)	cos(0.0) is 1.0
exp(x)	exponential function e^x	exp(1.0) is 2.71828 exp(2.0) is 7.38906
fabs(x)	absolute value of x	fabs(5.1) is 5.1 fabs(0.0) is 0.0 fabs(-8.76) is 8.76
floor(x)	rounds x to the largest integer not greater than x	floor(9.2) is 9.0 floor(-9.8) is -10.0
fmod(x, y)	remainder of x/y as a floating-point number	fmod(13.657, 2.333) is 1.992
log(x)	natural logarithm of x (base e)	log(2.718282) is 1.0 log(7.389056) is 2.0
log10(x)	logarithm of x (base 10)	log10(10.0) is 1.0 log10(100.0) is 2.0
pow(x, y)	x raised to power y (x^y)	pow(2, 7) is 128 pow(9, .5) is 3
sin(x)	trigonometric sine of x (x in radians)	sin(0.0) is 0
sqrt(x)	square root of x	sqrt(900.0) is 30.0

CONT...

- Although C++ is shipped with a lot of standard functions, these functions are not enough for all users, therefore, C++ provides its users with a way to define their own functions (or user-defined function).
- Therefore, we must declare and implement this function ourselves.

FUNCTION PROTOTYPE

- A *prototype* is a declaration of a function's appearance or a foreshadowing of its definition.
- The prototypes list only the return type, the function name, and the data type of the function's parameters.
- That is the minimum requirement for a function declaration.
- The *function prototype*, tells the compiler
 - **how many** parameters the function takes,
 - **the data type** of each **parameter**, and
 - **the data type of the return value** for the function.

```
int funcitonName(int x, int y);
```

FUNCTION DECLARATION

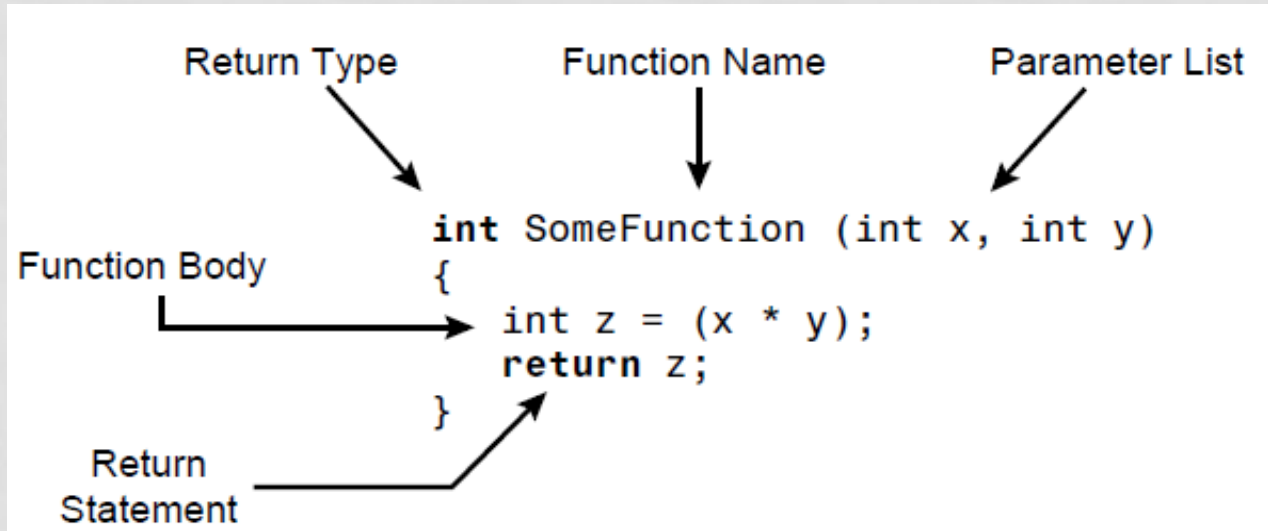
- **Before a function can be used, it must have first been declared.**
- A function definition has a name, a parentheses pair containing zero or more parameters and a body.

```
int funcitonName(int x, int y){  
    //Function body  
}
```

CONT...

- The function definition can be placed anywhere in the program after the function prototypes.
- If a function definition is placed in front of `main()`, there is no need to include its function prototype.

ANATOMY OF A FUNCTION.



- `type name ( parameter1, parameter2, ...)`
`{`
 statements
`}`

- **where:**

- **type** is the data type specifier of the data returned by the function.
- **name** is the identifier by which it will be possible to call the function.
- **parameters** (as many as needed): Each parameter consists of a data type specifier followed by an identifier, like any regular variable declaration (for example: `int x`) and which acts within the function as a regular local variable. They allow to pass arguments to the function when it is called. The different parameters are separated by commas.
- **statements** is the function's body. It is a block of statements surrounded by braces `{ }`.

EXAMPLE

```
// function example
#include <iostream>
using namespace std;

int addition (int a, int b)
{
    int r;
    r=a+b;
    return (r);
}

int main ()
{
    int z;
    z = addition (5,3);
    cout << "The result is " << z;
    return 0;
}
```

The result is 8

FUNCTION NAMING

- The function name can be any name conforming to the syntax rules of a variables.
- The function name made relevant to the function operations.

CONT...

- User-defined functions in C++ are classified into two categories.
- **Value-returning functions**
 - These functions have a return type.
 - These functions return a value of a specific data type using the **return** statement, which we will explain shortly.
- **Void functions**
 - **This** functions that do not have a return type(it has void return type).
 - These functions do not use a return statement to return a value.

RETURN STATEMENT

- The keyword **return** is used to terminate function and return a value to its caller.
- The return statement may also be used to exit a function without returning a value.
- The return statement may or may not include an expression.
- The general syntax of the return statement
return;
return (expression);

FUNCTION OVERLOADING

- Function overloading: **creating several functions with the same name**
- The signature of a function consists of the function name and its formal parameter list
- Two functions have different signatures if they have either different names or different formal parameter lists
- Note that the signature of a function does not include the return type of the function

EXAMPLE

```
void functionXYZ()  
void functionXYZ(int x, double y)  
void functionXYZ(double one, int y)  
void functionXYZ(int x, double y, char ch)
```

CONT...

```
void functionABC(int x, double y)  
int functionABC(int x, double y)
```

- Both of these function headings have the **same name** and **same formal parameter** list
- **Therefore, these function headings to overload the function functionABC are incorrect**
- **In this case, the compiler will generate a syntax error**
- Note that the return types of these function headings are different

INLINE FUNCTIONS

- Sometimes, we use the keyword ***inline*** to define user-defined functions
 - **Inline functions are very small functions, generally, one or two lines of code**
 - **Inline functions are very fast functions compared to the functions declared without the inline keyword**
- Example:-

```
double degrees ( double radian)
{
    return radian * 180.0 / 3.1415;
}
```

RECURSION AND ITERATION FUNCTIONS

- **A recursive function is one that calls itself.**
- A recursive function is called to solve a problem
- Thus the function launches (calls) a fresh copy of itself to work on the smaller problem –this is related as a Recursive-call/recursive step.

- Most of the time recursion uses selection statement and iteration uses repetition statement.
- But both uses control statement and any problem solved by recursion can be solved by iteration function.

EXAMPLE

```
#include<iostream.h>
#include<iomanip.h>
Unsigned long factorial (unsigned long);//prototype
int main()
{
int num;
cout<<"enter a positive integer:";
cin>>num;
cout<<"factorial="<<factorial(num);
return 0;
}
unsigned long factorial(unsigned long n)
{
if ( n <= 1) //the base case
return 1;
else
return n * factorial (n - 1);
}
```

