

Computer Programming(ECEg1052)

Chapter-5

Function

Why use Function?

- ▶ You can simplify programming tasks by breaking programs into smaller logical components.
- ▶ Function are used for divide a large code into module, due to this we can easily debug and maintain the code.
- ▶ E.g. if we write a calculator programs at that time we can write every logic in a separate function [For addition **sum()**, for subtraction **sub()**].
- ▶ Any function can be called many times.
- ▶ Code Re-usability
- ▶ Develop an application in module format.
- ▶ Easily to debug the program.
- ▶ **Code optimization:** No need to write lot of code.

Function Definitions

- ▶ **Function definition:** statements that make up a function
- ▶ *A function is a group of statements that together perform a specific task.*
- ▶ *Every C++ program has at least one function, which is main().*
- ▶ *Every function that is called in a program must be defined somewhere (once only).*
- ▶ *The syntax is:*

```
return_type  
function_name(parameter)  
{  
    function body;  
}
```

Cont...

- ▶ **Definition includes:**
 - ▶ return type: data type of the value that function returns to the part of the program that called it
 - ▶ name: name of the function. Function names follow same rules as variables
 - ▶ parameter list: variables containing values passed to the function
 - ▶ body: statements that perform the function's task, enclosed in {}

Cont...

Return type Parameter list (This one is empty)

Function name

Function body

```
int main ()  
{  
    cout << "Hello World\n";  
    return 0;  
}
```

- Note: The line that reads `int main()` is the *function header*.

Function Return Type

- ▶ If a function returns a value, the type of the value must be indicated:

```
int main()
```

- ▶ If a function does not return a value, its return type is ***void***:

```
void printHeading()  
{  
    cout << "Monthly Sales\n";  
}
```

Function Declarations

- ▶ *A function declaration is the process of tells the compiler about a function name.*
- ▶ *Then, the actual body of the function can be defined separately.*
- ▶ *The syntax*

return_type

function_name(parameter);

Function Calling

- *The syntax is:* `function_name();` // or
`variable=function_name(argument);`

```
#include <iostream>

void function_name() {
    ... ..
    ... ..
}

int main() {
    ... ..
    function_name();
    ... ..
}
```

The diagram illustrates the flow of control during a function call. A line from the closing brace of the `main` function points to the opening brace of the `function_name` function, indicating the call. Another line from the closing brace of the `function_name` function points back to the line following the `function_name()` call in the `main` function, indicating the return of control.

Cont...

- ▶ Example: `sum()` function
- ▶ `#include<iostream.h>`
- ▶ `void sum(); // declaring a function`
- ▶ `int a = 11, b = 22, c;`
- ▶
`void main()`
- ▶ `{`
- ▶ `sum(); // calling the function`
- ▶ `}`
- ▶ `void sum() // defining the function`
- ▶ `{`
- ▶ `c = a + b;`
- ▶ `cout << "Sum: " << c;`
- ▶ `}`

Print out:

Sum: 33

Cont...

Program 6-1

```
1  // This program has two functions: main and displayMessage
2  #include <iostream>
3  using namespace std;
4
5  /*******
6  // Definition of function displayMessage *
7  // This function displays a greeting. *
8  /*******
9
10 void displayMessage()
11 {
12     cout << "Hello from the function displayMessage.\n";
13 }
14
15 /*******
16 // Function main *
17 /*******
18
19 int main()
20 {
21     cout << "Hello from main.\n";
22     displayMessage();
23     cout << "Back in function main again.\n";
24     return 0;
25 }
```

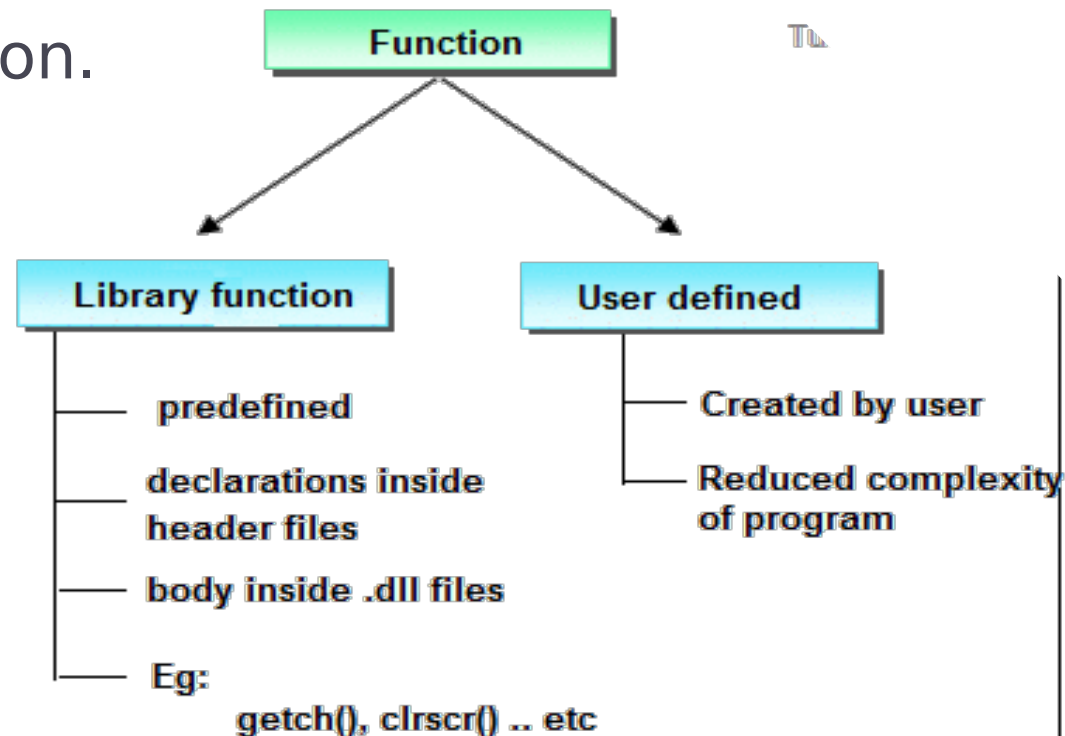
Program Output

```
Hello from main.
Hello from the function displayMessage.
Back in function main again.
```



Type of Function

- ▶ There are two type of function in C++ Language. They are:
 - ▶ Library function or pre-define function.
 - ▶ User defined function.



Library Function

- ▶ Library functions are the built-in function in C++ programming.
- ▶ Programmer can use library function by invoking

```
#include <iostream>
#include <cmath> // includes the content of cmath file.
using namespace std;
int main()
{
    double number, squareRoot;
    cout << "Enter a number: ";
    cin >> number;
    // sqrt() is a library function to calculate square root.
    squareRoot = sqrt(number);
    cout << "Square root of " << number << " = " << squareRoot;
    return 0;
}
```

Print out:

Enter a number: 26
Square root of 26 =
5.09902

Sending Data into a Function

- ▶ Can pass values into a function at time of call:

```
c = pow(a, b);
```

- ▶ Values passed to function are arguments
- ▶ Variables in a function that hold the values passed as arguments are parameters

A Function with a Parameter Variable

```
▶ void displayValue(int num)
▶ {
▶     cout << "The value is " << num << endl;
▶ }
```

The integer variable *num* is a parameter.

It accepts any integer value passed to the function.

Passing value or data to function

- ▶ There are two ways to pass value or data to function.
 - ▶ call by value
 - ▶ call by reference
- ▶ **Call by value:** In call by value, original value can **not** be **changed** or **modified**.
- ▶ **Call by reference:** In call by reference, original value is **changed** or **modified** because we pass a reference (an address). So actual and formal arguments shares the same address space. Hence, any value changed inside the function, is reflected inside as well as outside the function.

Cont...

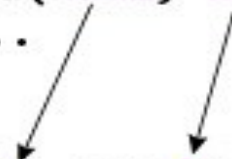
- ▶ The value of **num1** and **num2** are initialized to variables **a** and **b** respectively. These arguments **a** and **b** are called formal arguments. For

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

int add(int, int);

int main() {
    ... ..
    sum = add(num1, num2); // Actual parameters: num1 and num2
    ... ..
}

int add(int a, int b) { // Formal parameters: a and b
    ... ..
    add = a+b;
    ... ..
}
```



Return Statement

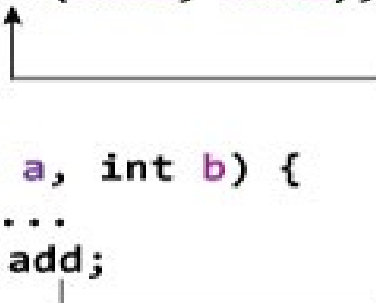
- ▶ A function can return a single value to the calling program using return statement. In the previous program, the value of add is returned from user-defined function to the calling program using statemer

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

int add(int, int);

int main() {
    ... .. ...
    sum = add(num1, num2);
}

int add(int a, int b) {
    ... .. ...
    return add;
}
```



The diagram illustrates the return flow. A line originates from the `return add;` statement in the `add` function, moves horizontally to the right, then vertically upwards, and finally horizontally to the left, ending with an arrow pointing to the `add(num1, num2);` call in the `main` function.

Returning a Value From a Function

- ▶ A function can return a value back to the statement that called the function.
- ▶ You've already seen the pow function, which returns a value:

```
double x;  
x = pow(2.0, 10.0);
```

Cont...

- ▶ In a value-returning function, the return statement can be used to return a value from function to the point of call. Example:

```
int sum(int num1, int num2)
{
    double result;
    result = num1 + num2;
    return result;
}
```

A Value-Returning Function

Return Type



```
▶ int sum(int num1, int num2)
{
    double result;
    result = num1 + num2;
    return result;
}
```

Cont...

```
▶ int sum(int num1, int num2)
{
    return num1 + num2;
}
```

Functions can return the values of expressions, such as
num1 + num2

Cont...

▶ Example: Call by Value function

▶ `#include<iostream.h>`

▶ `void swap(int a, int b){`

▶ `int temp;`

▶ `temp = a;`

▶ `a = b;`

▶ `b = temp;`

▶ `}`

▶ `void main(){`

`int a=111, b=222;`

`swap(a, b); // passing value to function`

▶ `cout<<"Value of a: "<< a <<endl;`

▶ `cout<<"Value of b: "<< b <<endl;`

`}`

Print out:

Value of a: 111

Value of b: 222

Cont...

▶ **Example: Call by Reference function**

▶ `#include<iostream.h>`

▶ `void swap(int *a, int *b){`

▶ `int temp;`

▶ `temp = *a;`

▶ `*a = *b;`

▶ `*b = temp;`

▶ `}`

▶ `void main(){`

`int a=111, b=222;`

`swap(&a, &b); // passing value to function`

▶ `cout<<"Value of a: "<< a <<endl;`

▶ `cout<<"Value of b: "<< b <<endl;`

`}`

Print out:

Value of a: 222

Value of b: 111

Summary: Argument Passing

- ▶ Arguments passed **by value** and **by reference**.

For example

```
{  
    val++;  
    ref++;  
}
```

For example:

```
int main() {  
    int i=1;  
    int j=1;  
    f(i, j);  
}
```

When **f()** is called, **val++** increments a local copy of the first actual argument, whereas **ref++** increments the second actual argument.

- It will increment **j** but not **i**.

Difference Between **Call by Value** and **Call by Reference**.

call by Value	call by Reference
This method copy original value into function as a arguments.	This method copy address of arguments into function as a arguments.
Changes made to the parameter inside the function have no effect on the argument.	Changes made to the parameter affect the argument. Because address is used to access the actual argument.
Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in same memory location

Note: By default, C++ uses call by value to pass arguments.

Function Overloading

- ▶ Two or more functions having same name but different argument(s) are known as overloaded functions.
- ▶ In Function Overloading “Function” name should be the same and the arguments should be different.
- ▶ Here, all 4 functions are overloaded functions called “**test**” because argument(s) passed to these functions are different.

▶ For example:

```
int test() { }  
int test(int a) { }  
float test(double a) { }  
int test(int a, double b) { }
```

- ▶ N.B., the **return type** of all these 4 functions **are not same**. Overloaded functions may or may not have different return type but it should have different argument(s).

Cont...

```
► double test(int a) { }  
int test(int b) { } // ERROR CODE!!!
```

