Chapter 9 User-defined Functions

A **function** is a block of code that performs a specific task. With functions, a large programming project can be divided into small blocks of code makes our program easy to understand, reuse, and can be divided among many programmers.

There are two types of functions:

- **Standard library functions**: are predefined functions declared in standard libraries or header files such as `pow()`, `to_string()`, `toupper()`, `isalpha()`, `stoi()`, etc.
- **User-defined functions**: User-defined functions refer to functions that we define by ourselves to do any task.

In this chapter, you will learn about User-defined functions.

1. Function Definition

To create a function, use the following syntax:

```
returnType functionName(type1 parameter1, type2 parameter2, ...){  //header

    // body
}
```

In the above syntax, a function definition contains the following parts:

- The first line `returnType functionName(type1 parameter1, type2 parameter2, ...)` is known as **function header**, and the statement(s) within curly braces is called **function body**.
- **Return Type** defines the return data type of a value in the function. The return data type can be `integer`, `float`, `char`, etc.
- **Function Name** defines the actual name of a function that contains some parameters.
- **Formal parameters** are the variables defined by a function that receives values when the function is called. Parameters can be any type, in any order, and any number of parameters.
- **Function Body** is the collection of the statements to be executed for performing the specific tasks in a function.

2. `return` Statement

The **`return`** statement has two forms:

Functions with return type `void` use the following form:

```
return;   // to terminate the execution of the function
```

Functions with non-void return type use the following form:

```
return expression;  // to return an expression and also terminate the execution
                    // of the function
```

3. Returning Value

A function may or may not return any value. If you do not want a function to return any value, use `void` for the return type.

Example 01: Create a function that displays `Hello`.

```
void hello(){

    cout << "Hello";
}
```

If you want a function to return a value, you need to use data type such as `int`, `long`, `char`, etc., for the return type.

The data type of the returned value has to be same as the return type of the function, otherwise, you will get compilation error.

Example 02: Create a function that return an integer value.

```
int get(){

    return 10;
}
```

In the above example, we have to return `10` as a value, so the return type we use is `int`. If you want to return a floating-point value, you need to use `float` as the return type of the function.

Note, function names are identifiers and should be unique. The function name must be unique which includes also the function names defined in the included head files.

`return` statement also ends the function execution, hence it must be the last statement of any function. If you write any statement after the return statement, it will not be executed.

Example 03:

```
void hello(){
    cout << "Hello";

    return;

    cout << "How are you?"; // This statement will not be reached
}
```

4. Functions with Parameters

Function parameters are variables that accepts the values that a function receives when being called. If you want a function that can accept values, then create a function with parameters.

Example 04: Create a function with parameters.

```
float square(float x){
    return x * x;
}
```

5. Function Calls

A function can be called or invoked by its name. It can be called in the main() function, in any other functions, or even in itself.

As already mentioned, a function may or may not accept any parameters, and it may or may not return any value. Based on these facts, there are different aspects of function calls.

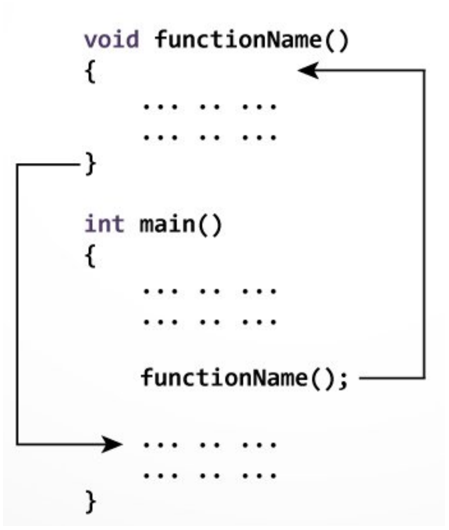
Example 05: Call a function that has no parameters and does not return any value.

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

void functionName(){
    cout << "John";
}

int main(){
    functionName(); // call the function

    return 0;
}
```

Output

```
John
```

Here explains how the function works:



```
void functionName()
{
    ... .. ...
    ... .. ...
}

int main()
{
    ... .. ...
    ... .. ...

    functionName();

    ... .. ...
    ... .. ...
}
```

- The execution of a C++ program begins from the **main()** function.
- When the compiler encounters **functionName()**;, control of the program jumps to void **functionName()**{}
- And, the compiler starts executing the codes inside **functionName()**.
- The control of the program jumps back to the **main()** function once code inside the **functionName()** has been executed.

Example 06: Call a function that has no parameters but returns a value.

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

int sum(){
    int a, b;

    cout <<"Enter two numbers: ";
    cin >> a >> b;

    return a + b;
}

int main(){
    int total = sum();  // call the function

    cout << total;

    return 0;
}
```

Output

```
Enter two numbers: 2 5
7
```

Example 07: Call a function that has parameters and does not return any value.

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

void sum(int a, int b){       // formal parameters
    cout << a + b;
}

int main(){
    int n1, n2, result;

    cout <<"Enter two numbers: ";
    cin >> n1 >> n2;

    // call the function
    sum(n1, n2);              // actual parameters or arguments

    return 0;
}
```

Output

```
Enter two numbers: 3 9
12
```

In the above example, two variables n1 and n2 are passed during the function call. The parameters a and b accept the passed arguments in the function definition.

Example 08: Call a function that has parameters and returns a value.

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

float square(float x){
    return x * x;
}

int main(){
    cout << square(2);

    return 0;
}
```

Output

```
4
```

In C++, having a function sum(void) and sum() is the same thing. They are the functions that have no arguments.

Example 09:

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

void sayHello(void){
    cout <<"Hello";
}

int main(){
    sayHello();

    return 0;
}
```

Output

```
Hello
```

Functions can be called by other functions, not only by the `main()` function.

Example 10:

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

int sum(int a, int b){
    return a + b;
}

int multiply(int a, int b){
    int c = sum(1, 2);

    return a * b * c;
}

int main(){
    cout << multiply(2, 3);

    return 0;
}
```

Output

```
18
```

Example 11: Nested function calls.

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

int sum(int a, int b){

    return a + b;
}

int multiply(int a, int b){

    return a * b;
}

int main(){
    cout << multiply(3, sum(1, 2));

    return 0;
}
```

Output

```
9
```

Note that a value-returning function also can be invoked as a statement. In this case, the return value is ignored. This is rare but is permissible if the caller is not interested in the return value.

Example 12:

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

int sum(){
    int a = 10, b = 20;

    cout << "Hello";

    return a + b;
}

int main(){
    cout << "Going to call the sum function:" << endl;
    sum();

    return 0;
}
```

Output

```
Going to call the sum function:
Hello
```

Exercises

For each of the following exercises, you are asked to write a function and its test programs that call the function to show that it works as intended.

1. Write a function with the following header to display the even digits in an integer:
   `void display_even(int number)`
   Write a test program that asks the user to enter an integer and displays the even digits in it.
2. Write a function with the following header to display the largest of three numbers:
   `void display_largest(int num1, int num2, int num3)`
   Write a test program that asks the user to enter three numbers and call the function to display the largest one.
3. Write a function called `sum_series(n)` to return the result of the following series:
   $$f(n) = \frac{1}{3} + \frac{1}{8} + \frac{1}{15} \ldots \quad + \frac{1}{n(n+2)}$$
   Write a test program that asks the user to enter an integer, n, and displays the result.
4. Write a function called `string_upper` that accepts a string and returns it in uppercase.
5. Write a function called `string_char` that accepts a string and a character, and returns `true` if the string contains the character, otherwise, returns `false`.

6. Write a function called `string_comparei` that accepts two strings, and returns `true` if the both strings are equal, otherwise, returns `false`. Make it case-insensitive.

6. Overloading Functions

In C++, two different functions can have the same name if they have either:

- different number of parameters
- different data type for their parameters.

Example 13: Overloading functions

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

int calculate(int a){
    return a * a;
}

int calculate(int a, int b){
    return a + b;
}

float calculate(float a, float b){
    return a / b;
}

int main(){
    int x = 5, y = 2;
    float n = 5, m = 2;

    cout << calculate(x) << endl;
    cout << calculate(x, y) << endl;
    cout << calculate(n, m);

    return 0;
}
```

Output

```
25
7
2.5
```

7. Function Prototype

In C++, a function must be declared above the location where you use it. As the compiler reads a program from top to bottom, if by the time it gets to a function call, but it has not been told about the function definition, it would believe that no such function exists.

A **Function Prototype**, also called **Function Declaration**, is simply the declaration of a function that specifies the function's name, parameters and return type. It does not contain function body.

A function prototype tells the compiler that the function will later be defined in the program.

Example 14: Using Function Prototype.

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

void functionName();        // function prototype

int main(){
    functionName();

    return 0;
}

void functionName(){

    printf("John");
}
```

Output

```
John
```

Example 15: Using Function Prototype.

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

int sum(int a, int b);   // function prototype

int main(){
    cout << sum(10, 25);

    return 0;
}

int sum(int a, int b){

    return a + b;
}
```

Output

```
35
```

In the above example, `int` is a return data type of `sum`() function that contains two integer parameters as `a` and `b`. We can also write the above function prototype as below:

Example 16: Using Function Prototype.

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

int sum(int, int);      // function prototype

int main(){
    cout << sum(10, 25);

    return 0;
}

int sum(int a, int b){

    return a + b;
}
```

Output

```
35
```

8. Default Parameter Value

C++ allows you to declare functions with default values for function parameters. The default values are passed to the parameters when a function is invoked without the arguments.

Example 17: In this example, only the first argument is passes.

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

void display(string name = "John", int age = 20){

    cout << "I'm " << name << ", " << age << " years old." << endl;
}

int main(){
    display();
    display("Jack");
    display("Lucy", 25);

    return 0;
}
```

Output

```
I'm John, 20 years old.
I'm Jack, 20 years old.
I'm Lucy, 25 years old.
```

If the above example, if the second argument is passed without the first argument, it will result in an error.

```
display(20); // Error
```

When a function contains a mixture of parameters with and without default values, **the parameters with default values must be declared last**. For example,

```
void func1(int x, int y = 0, int z);         // Invalid
void func2(int x = 0, int y = 0, int z);     // Invalid

void func3(int x, int y, int z = 0); .       // Valid
void func4(int x, int y = 0, int z = 0);     // Valid
void func5(int x = 0, int y = 0, int z = 0); // Valid
```

The following calls are NOT valid:

```
func4(1, , 20);  // Error
func5(, , 20);   // Error
```

The following calls are valid:

```
func4(1);    // Parameters y and z are assigned a default value
func5(1, 2); // Parameter z is assigned a default value
```

9. Creating and Using Your Own Header Files

Step1: Let's create an empty file, and write two function definitions as below:

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

int sum(int a, int b){
    return a + b;
}

void say_hi(string name){
    cout <<"Hi " << name;
}
```

Step 2: Save the file as myheader (no extension).

Step 3: Using Header Files

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

int main(){
    int result = sum(5, 10);
    cout <<"Sum of two numbers: " << result;

    say_hi("John")

    return 0;
}
```

Output

```
Sum of two numbers: 15
Hi John
```

Note: Instead of writing `<myheader>`, you have to use this terminology `"myheader"`. Also, if both files are not in the same folder, you will need to give the path, for example, `"myfolder/myheader"`.

10. Types of Function Calls

To execute a function, you need to call it. In the case of a function with parameters, arguments need to pass during function call. There are two types of function calls:

- Call by value
- Call by reference

10.1 Call by Value

In Call by Value, aka. Pass by Value, the values of arguments are passed to the formal parameter of a function. If we make some changes in the formal parameters, it will not affect the original value of the arguments.

Example 18: Call by Value

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

void update(int x, int y){
    x = 10;
    y = 20;
}
```

```
int main(){
    int x = 1, y = 2;

    update(x, y);

    cout << "After calling the function\n";
    cout << "x = " << x << ", y = " << y;

    return 0;
}
```

Output

```
After calling the function
x = 1, y = 2
```

main::x	main::y	update::x	update::y
1	2	~~1~~ 10	~~2~~ 20
0x6dfed4	0x6dfed5	0x9rg768	0x9rg769

10.2 Call by Reference

In Call by Reference, aka. Pass by Reference, the addresses of the arguments are copied into the formal parameters. If we make some changes in the formal parameters, it will show the effect in the original value of the arguments.

In this case, the address operator & is used with function parameters to hold addresses of arguments passed during the function call.

Example 19: Call by Reference using the address operator &

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

void update(int &x, int &y){
    x = 10;
    y = 20;
}

int main(){
    int x = 1, y = 2;

    update(x, y);

    cout << "After calling the function" << endl;
    cout << "x = " << x << ", y = " << y;
    return 0;
}
```

Output

```
After calling the function
x = 10, y = 20
```

main::x, update::x
~~1~~ 10
0x6dfed4

main::y, update::y
~~2~~ 20
0x6dfed5

Call by Value and Call by Reference Comparison

Example 01: Create a function that accepts two values and returns their sum:

Call by Value	Call by Reference

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

int sum(int a, int b){
    int result = a + b;

    return result;
}
int main(){
    int result = sum(5, 10);

    cout << "Result = " << result;

    return 0;
}
```
	```
	#include <iostream>
	using namespace std;

	void sum(int a, int b, int &result){
	    result = a + b;
	}


	int main(){
	    int result;

	    sum(5, 10, result);
	    cout << "Result = " << result;

	    return 0;
	}
	```

Example 02: Create a function that accepts two values and returns their sum and subtraction:

Call by value

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

int sum(int a, int b){
    int sum_result = a + b;
    return sum_result;
}

int sub(int a, int b){
    int sub_result = a - b;
    return sub_result;
}

int main(){
    int sum_result = sum(5, 10);
    int sub_result = sub(5, 10);

    cout << "Sum = " << sum_result << endl;
    cout << "Sub = " << sub_result;

    return 0;
}
```

Call by reference

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

void calculate(int a, int b, int &sum_result,
               int &sub_result){
    sum_result = a + b;
    sub_result = a - b;
}

int main(){
    int sum_result;
    int sub_result;

    calculate(5, 10, sum_result, sub_result);

    cout << "Sum = " << sum_result << endl;
    cout << "Sub = " << sub_result;

    return 0;
}
```

Every function in C/C++ can return only one value. In order to create a function that returns more than one value in C/C++, you will need to use Call by Reference.

11. Arrays and Functions

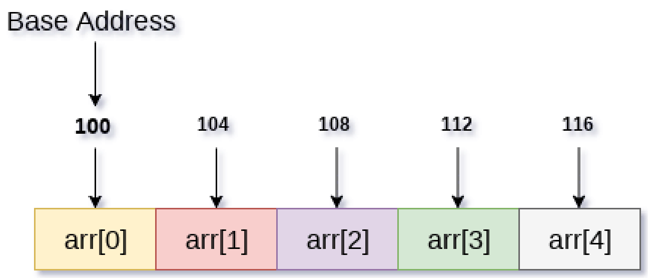
11.1 Declaring Function with Arrays as Parameters

```
int sum1(int arr[], int size);

int sum2(int arr[][10], int row, int col);
```

11.2 Passing Arrays to a Function

In C++, if you pass an array as an argument to a function, what actually gets passed is the base address of the array.

```
int arr[] = {1, 2, 3, 4, 5};
```



Example 20: Passing a one-dimensional array to a function.

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

void update(int marks[], int size){
    marks[1] = 100;
    marks[2] = 200;
}

int main(){
    int marks[] = {50, 90, 87, 70, 95};
    int size = sizeof(marks) / sizeof(int);

    update(marks, size);

    // Display the array
    for(int i = 0; i < size; i++)
        cout << marks[i] << "\t";

    return 0;
}
```

Output

```
50      100     200     70      95
```

Note: As you can see, when an array is passed to a function, **we also have to pass the size of the array to the function** because the function does not know how many elements the array has. Also, as arrays are passed by address, any changes to the formal parameters will also change the original arrays.

Example 21: Passing a Multi-dimensional array to a function.
```cpp
#include <iostream>
using namespace std;

void display_array(int arr[2][3], int row, int column){

    cout << "The complete array is:" << endl;

    for (int i = 0; i < row; ++i){
        for (int j = 0; j < column; ++j)
            cout << arr[i][j] << "\t";

        cout << endl;
    }
}

int main(){
    int arr[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };
    display_array(arr, 2, 3);   // pass the array as argument

    return 0;
}
```

Output
```
The complete array is:
1	2	3
4	5	6
```

Exercises

For each of the following exercises, you are asked to write a function and its test programs that call the function to show that it works as intended.

7. Write a function called `string_reverse` that accepts a string and reverse it.
8. Write a function called `find_int_in_array` that accepts an integer and an array of integers, then returns the index of first occurrence of the integer within the array, or returns -1 if not found.
9. Write a function called `sort_array` that sorts the array that pass to it in ascending order.
10. Write a method called `get_largest` that accepts a 2D array of integers and returns the largest number stored in the array.
11. Write a method called `triangle_calculator` that accepts the three sides, a, b and c, of a triangle, and returns the perimeter and area.

12. Write a header file that contains two functions:

    `celsius_to_fahrenheit` that accepts a Celsius and returns it in Fahrenheit.
    `fahrenheit_to_celsius` that accepts a Fahrenheit and returns it in Celsius.

    The formulas for the conversion are:
    celsius = (5 / 9) * (fahrenheit – 32)
    fahrenheit = (9 / 5) * celsius + 32

    Write a test program that will ask the user to select the menu:

    1. Celsius to Fahrenheit
    2. Fahrenheit to Celsius

    After the user selected the menu, ask the user to enter the value and convert it by calling the function in the header file.

Reference

[1] Y. Daniel Liang. 'Introduction to Programming with C++', 3e – 2014

[2] Dey, P., & Ghosh, M. 'Computer Fundamentals and Programming in C', 2e – 2013

[3] https://cplusplus.com/doc/tutorial/

[4] https://www.programiz.com/cpp-programming

[5] https://www.studytonight.com/cpp/