TOPIC 5: FUNCTIONS

At the end of the chapter, the students should be able to:

- *use predefined functions*
- *built independent functions or user-defined functions*
- *understand the scope of global and local variables*
- *learn how to use function with parameter*

5.1 INTRODUCTION

As programs get longer and complicated, it is common to *group the program statements into its interrelated modules or segments or subprograms*. Each module or segment will only perform a *particular task*. The module or segment of codes is referred to as *function*.

A function is a mini program that performs a particular task. Each function may include its *own variables* and its *own statements*, just like writing the `main` function. This mini program can be *built, compiled and tested independently*.

Benefits of using functions:

(i) The `main` program is *simplified* where it will contain the function call statements. By doing that planning, coding, testing, debugging, understanding and maintaining a computer will be easier.

(ii) The same program can be *reused* in another program, which will prevent code duplication and reduce the time of writing the program.

```
1   #include<iostream>
2   using namespace std;
3
4   int sum (int x , int y);        // function declaration
5   int sum(int x , int y)          // funciotn definition
6   {
7      int result;
8     result = x  + y;
9     return (result);
10  }
11  int main()
12  {
13     int x , y , output ;
14    x = 20;
15    y = 100;
16    output = sum(x,y);        /* calling a function and storing the
17                               value from funciton to variable output*/
18   cout<<output;
19
20  return 0;
21  }
```

Figure 5.1 Example of function in C++

SOURCE FROM:
[1]MALIK, D.S, *C++ PROGRAMMING FROM PROBLEM ANALYSIS TO PROGRAM DESIGN*, THOMSON COURSE TECHNOLOGY, 2013,
[2]NORIZAN M., & MAZIDAH P., *PROBLEM SOLVING WITH C++*, UPENA UITM SHAH ALAM, 2006

Functions can be classified into two types:

(a) Predefined or built-in functions
(b) Programmer defined or independent functions

5.2 PREDEFINED FUNCTIONS

Predefined or built-in functions are functions used by the programmers in order to *speed up program writing*. Programmers may use the *existing code* to perform tasks without having to rewrite any code. These functions are predefined by the producer of a compiler and are stored in the header files called *libraries*.

In order for a program to use a predefined function, the appropriate library file must be included in the program using the `#include` directive.

Function	Header File	Purpose	Parameter(s) Type	Result
`abs(x)`	`<cmath>`	Returns the absolute value of its argument: `abs(-7) = 7`	`int` (`double`)	`int` (`double`)
`ceil(x)`	`<cmath>`	Returns the smallest whole number that is not less than `x`: `ceil(56.34) = 57.0`	`double`	`double`
`cos(x)`	`<cmath>`	Returns the cosine of angle `x`: `cos(0.0) = 1.0`	`double` (`radians`)	`double`
`exp(x)`	`<cmath>`	Returns e^x, where `e = 2.718`: `exp(1.0) = 2.71828`	`double`	`double`
`fabs(x)`	`<cmath>`	Returns the absolute value of its argument: `fabs(-5.67) = 5.67`	`double`	`double`

Table 5.1 Commonly used predefined functions in C++

Function	Header File	Purpose	Parameter(s) Type	Result
`floor(x)`	`<cmath>`	Returns the largest whole number that is not greater than `x`: `floor(45.67) = 45.00`	`double`	`double`
`islower(x)`	`<cctype>`	Returns `true` if `x` is a lowercase letter; otherwise, it returns `false`; `islower('h')` is `true`	`int`	`int`
`isupper(x)`	`<cctype>`	Returns `true` if `x` is a uppercase letter; otherwise, it returns `false`; `isupper('K')` is `true`	`int`	`int`
`pow(x, y)`	`<cmath>`	Returns x^y; if `x` is negative, `y` must be a whole number: `pow(0.16, 0.5) = 0.4`	`double`	`double`
`sqrt(x)`	`<cmath>`	Returns the nonnegative square root of `x`; `x` must be nonnegative: `sqrt(4.0) = 2.0`	`double`	`double`
`tolower(x)`	`<cctype>`	Returns the lowercase value of `x` if `x` is uppercase; otherwise, it returns `x`	`int`	`int`
`toupper(x)`	`<cctype>`	Returns the uppercase value of `x` if `x` is lowercase; otherwise, it returns `x`	`int`	`int`

Table 5.2 Commonly used predefined functions in C++

SOURCE FROM:
[1]MALIK, D.S, *C++ PROGRAMMING FROM PROBLEM ANALYSIS TO PROGRAM DESIGN*, THOMSON COURSE TECHNOLOGY, 2013,
[2]NORIZAN M., & MAZIDAH P., *PROBLEM SOLVING WITH C++*, UPENA UITM SHAH ALAM, 2006

```
//How to use predefined functions.
#include <iostream>
#include <cmath>
#include <cctype>

using namespace std;

int main()
{
    int   x;
    double u, v;

    cout << "Line 1: Uppercase a is "
         << static_cast<char>(toupper('a'))
         << endl;                                    //Line 1

    u = 4.2;                                         //Line 2
    v = 3.0;                                         //Line 3
    cout << "Line 4: " << u << " to the power of "
         << v << " = " << pow(u, v) << endl;         //Line 4

    cout << "Line 5: 5.0 to the power of 4 = "
         << pow(5.0, 4) << endl;                     //Line 5

    u = u + pow(3.0, 3);                             //Line 6
    cout << "Line 7: u = " << u << endl;             //Line 7

    x = -15;                                         //Line 8
    cout << "Line 9: Absolute value of " << x
         << " = " << abs(x) << endl;                 //Line 9

    return 0;
}
```

Sample Run:

```
Line 1: Uppercase a is A
Line 4: 4.2 to the power of 3 = 74.088
Line 5: 5.0 to the power of 4 = 625
Line 7: u = 31.2
Line 9: Absolute value of -15 = 15
```

Figure 5.2 How to use predefined function in C++

5.3 INDEPENDENT FUNCTIONS

Programmers are also able to create their own functions since the built-in functions in C++ libraries are limited to perform other programming tasks.

Independent functions or also known as *programmer-defined functions* are functions whose tasks are
determined by the programmer and *defined within the program in which the function is used*.

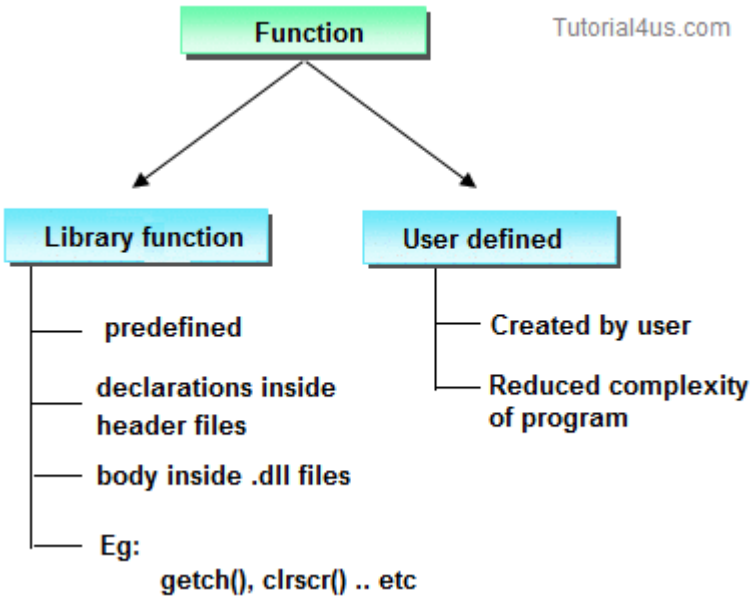


Figure 5.1 Differences between predefined functions and user-defined functions in C++

5.3.1 Program Structure for Independent Functions

In order to use the independent function, a programmer must satisfy three (3) requirements:

(i) Function prototype declaration
(ii) Function call
(iii) Function definition

```
void show();   //function declaration
main()
{
  ....
   show();      //function call
  ....
}
void show();   //function definition
{
...            //function body
...
}
```

Figure 5.3 Function structure in C++

-- *Function Prototype Declarations* --

All functions must be *declared* before they can be used in a program. Placed just before the `main` program and sometimes at the beginning of the `main` program, a function declaration specifies:

(i) The name of the function
It can be any names but the naming convention must follow the rules in naming the
identifiers in C++.

(ii) The order and type of parameters
- Tells the calling function what type of data value the parameter will receive when the function is called.
- Tells the calling function the order of the values to be transmitted to the called function.
- If it contains more than one data type, parameters must be separated by comma(s).
- Parameters are *optional*.
- Parameter list is also known as *arguments*.

(iii) The return value
- Indicates the type of value that the function will return
- The return value may be of any valid data type (`int`, `float`, `char`)
- `void` is used when a function does NOT return any value to the calling program

```
double func(double, double); //prototype

main(void){
  double result = func(10,20);
}

double func(double x, double y){
  return x * y;
}
```

Figure 5.4 Example of function prototype in C++

SOURCE FROM:
[1]MALIK, D.S, *C++ PROGRAMMING FROM PROBLEM ANALYSIS TO PROGRAM DESIGN*, THOMSON COURSE TECHNOLOGY, 2013,
[2]NORIZAN M., & MAZIDAH P., *PROBLEM SOLVING WITH C++*, UPENA UITM SHAH ALAM, 2006

-- Parameters --

There are two types of parameters:

(a) Formal parameters
- A formal parameter is a placeholder variable which is defined at the *called function* and *local to the called function.*
- The number of the formal parameters must agree with the number of actual parameters passed to the function.

(b) Actual parameters
- Actual parameter is a constant, variable or expression in a *function call* that corresponds to the formal parameter.
- Actual parameters contain *values that will be passed* to the function.

```
# 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;
    ... .. ...
}
```

Figure 5.5 Actual parameters vs. formal parameters

Rules for parameter passing:

(i) The *type* of actual must be the same with the formal parameter
(ii) The *order* of parameter must be the same
(iii) The *number* of actual parameter must be the same with the number of formal parameter

-- Function Definition --

A function must be *defined* before it can carry out the task assigned to it. In other words, the statements that perform the task are written inside the function definition.

A function is written once in the program and can be used by any other functions in the program.

The function definition is placed either before the `main()` or after the `main()`.

A function definition consists of a function *header* and a function *body*.

Function header	Function body
The function header defines the *return type*, *function name* and list of *parameters*. It is written the same way as the function prototype, except that the header does not end with a semicolon.	The function body contains declaration of local variables and executable statements needed to perform a task assigned to the function.

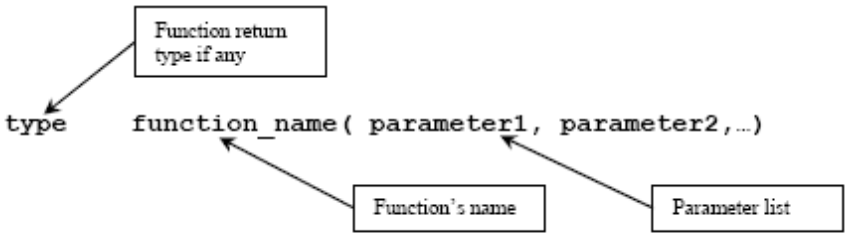


Figure 5.6 Function header in C++

-- Function call --

A function is made to perform its designated task by a function call in main function or in other function. When a function is called, the *program control* is passed to the called function and the statement inside the function will be executed to the end until control is passed back to the calling function.

To call a function, specify the function name and the values of the parameters that the function needs to do its job.

When completing a function call, there are two possibilities:

(i) Function call that returns no value

This type of function does not return any value to the calling function, as given in Figure 5.7. Upon completing the last statement in the function definition, the control is passed back to the statement that calls the function in the calling function. Thus, next statement in the calling function will be executed.

(ii) Function call that returns a value

SOURCE FROM:
[1]MALIK, D.S, *C++ PROGRAMMING FROM PROBLEM ANALYSIS TO PROGRAM DESIGN*, THOMSON COURSE TECHNOLOGY, 2013,
[2]NORIZAN M., & MAZIDAH P., *PROBLEM SOLVING WITH C++*, UPENA UITM SHAH ALAM, 2006

A function may also return a value. In this type of function, when the execution in the called function is complete, the control is passed back to the calling function with a value. This value will be used in the calling function.

There are four ways how a returned value can be used in the calling function:

(a) In an arithmetic expression
(b) In a logical expression
(c) In an assignment statement
(d) In an output statement

```
#include <iostream.h>

void welcome(void)          //function definition of welcome()
{
   cout << "Welcome to C++!\n";
}

void main(void)
{
   welcome();               // a function call to welcome()
}
```

Figure 5.7 Function definition and function call in C++

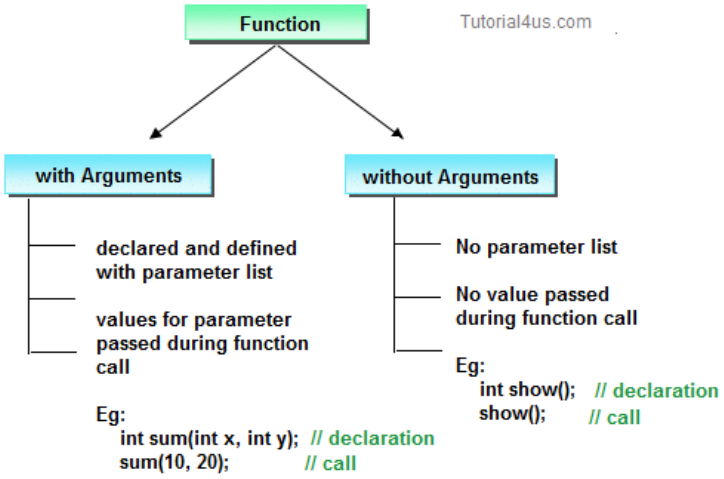


Figure 5.8 Differences between function with arguments and without arguments during function calls

```
//function prototype that returns a value
int calcSum(int, int, int);

void main()
{
   int num1, num2, num3, total;

   cout << "Enter 3 numbers:";
   cin >> num1 >> num2 >> num3;

   // in an assignment statement
   total = calcSum(num1, num2, num3);

   // in an output statement
   cout << "\nThe sum is :" << calcSum (num1, num2,
   num3);

   // in a logical expression
   if (calcSum(num1,num2,num3) < 0)
      // in an arithmetic expression
      total = calcSum(num1,num2,num3) / 3;
}

//function definition
int calcSum(int a, int b, int c)
{
   int sum = 0;
   sum = a + b + c;
   return sum;
}
```

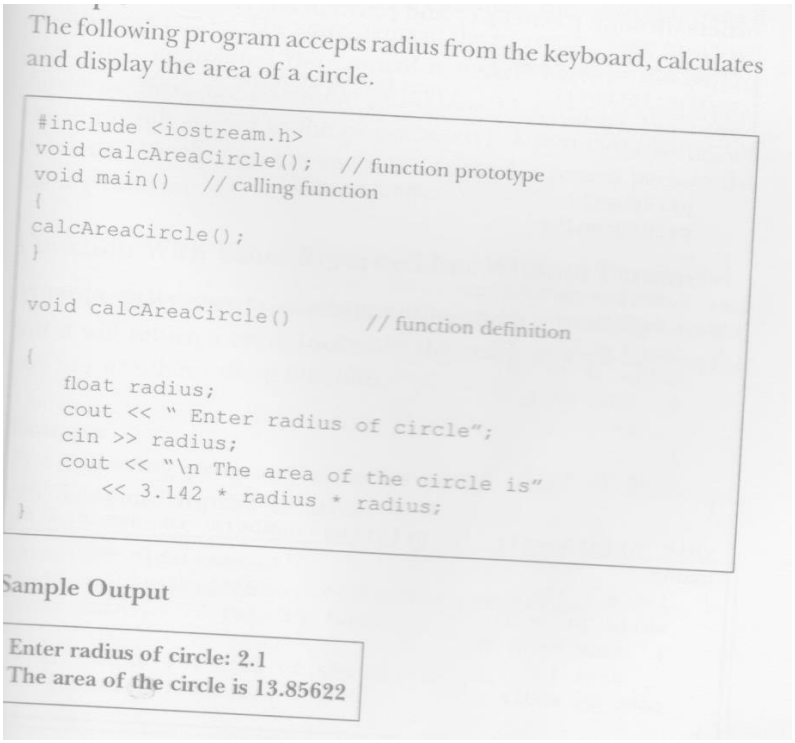
Figure 5.9 Example of how a returned value can be used in the calling function

5.3.2 Types of Independent Functions

In summary, when using functions, we can have four types of functions:

(i) Function without value returned and parameter
(ii) Function with value returned but without parameter
(iii) Function without value returned but with parameter
(iv) Function without value returned and parameter

SOURCE FROM:
[1]MALIK, D.S, *C++ PROGRAMMING FROM PROBLEM ANALYSIS TO PROGRAM DESIGN*, THOMSON COURSE TECHNOLOGY, 2013,
[2]NORIZAN M., & MAZIDAH P., *PROBLEM SOLVING WITH C++*, UPENA UITM SHAH ALAM, 2006



The following program accepts radius from the keyboard, calculates and display the area of a circle.

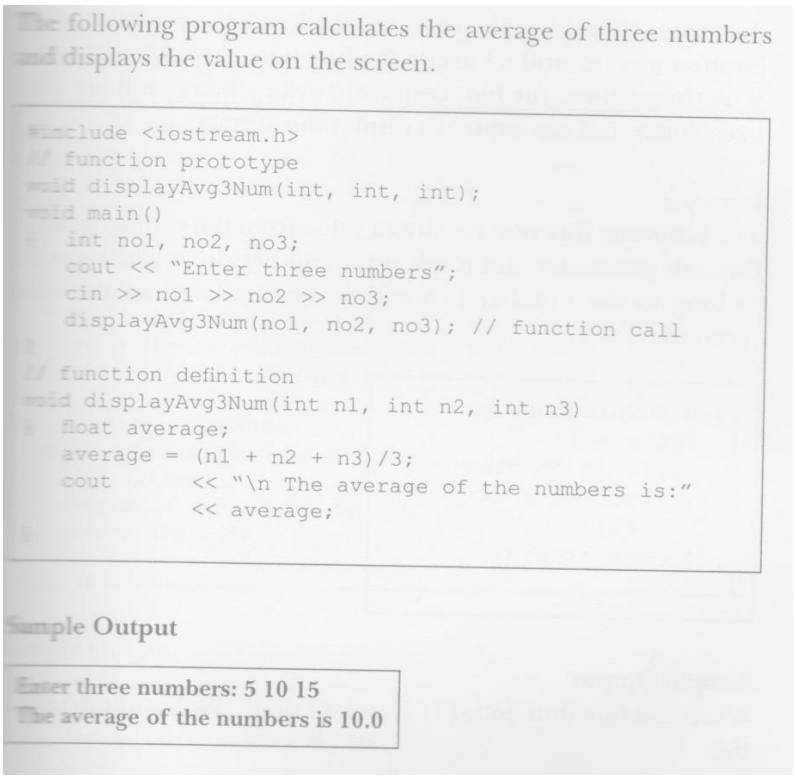
```
#include <iostream.h>
void calcAreaCircle();  // function prototype
void main()   // calling function
{
calcAreaCircle();
}

void calcAreaCircle()        // function definition
{
   float radius;
   cout << " Enter radius of circle";
   cin >> radius;
   cout << "\n The area of the circle is"
      << 3.142 * radius * radius;
}
```

Sample Output

```
Enter radius of circle: 2.1
The area of the circle is 13.85622
```

Figure 5.10 Function without value returned and parameter



The following program calculates the average of three numbers and displays the value on the screen.

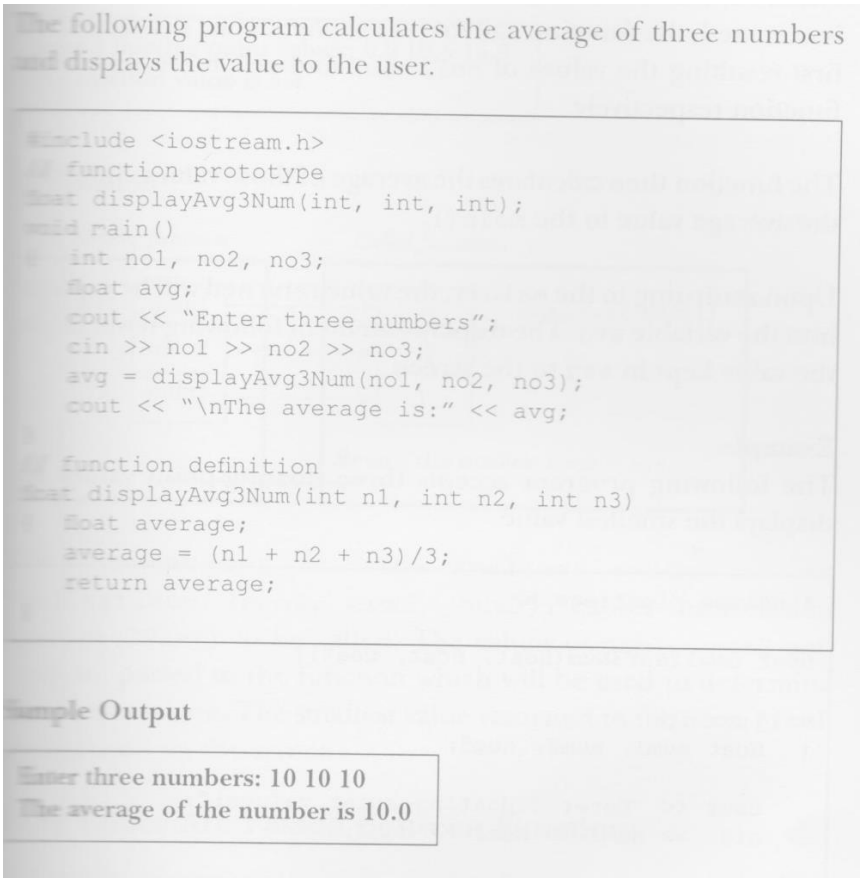
```
#include <iostream.h>
// function prototype
void displayAvg3Num(int, int, int);
void main()
{  int no1, no2, no3;
   cout << "Enter three numbers";
   cin >> no1 >> no2 >> no3;
   displayAvg3Num(no1, no2, no3); // function call
}
// function definition
void displayAvg3Num(int n1, int n2, int n3)
{  float average;
   average = (n1 + n2 + n3)/3;
   cout      << "\n The average of the numbers is:"
             << average;
}
```

Sample Output

```
Enter three numbers: 5 10 15
The average of the numbers is 10.0
```

Figure 5.11 Function with parameter and no value returned

SOURCE FROM:
[1]MALIK, D.S, *C++ PROGRAMMING FROM PROBLEM ANALYSIS TO PROGRAM DESIGN*, THOMSON COURSE TECHNOLOGY, 2013,
[2]NORIZAN M., & MAZIDAH P., *PROBLEM SOLVING WITH C++*, UPENA UITM SHAH ALAM, 2006



The following program calculates the average of three numbers and displays the value to the user.

```
#include <iostream.h>
// function prototype
float displayAvg3Num(int, int, int);
void main()
{
   int no1, no2, no3;
   float avg;
   cout << "Enter three numbers";
   cin >> no1 >> no2 >> no3;
   avg = displayAvg3Num(no1, no2, no3);
   cout << "\nThe average is:" << avg;
}
// function definition
float displayAvg3Num(int n1, int n2, int n3)
{
   float average;
   average = (n1 + n2 + n3)/3;
   return average;
}
```

Sample Output

```
Enter three numbers: 10 10 10
The average of the number is 10.0
```

Figure 5.12 Function with value returned and parameter

5.3 HOW VALUES ARE PASSED BETWEEN FUNCTIONS

A function *communicates* with another function by *passing values* between them. A function can send data to other function, and it can also accept values sent by other functions.

Data can be passed between functions through:

(i) Global variable
(ii) Parameter passing
(iii) A return value

5.3.1 Global Variable

When a variable is declared as global, its value can be changed at any point during the program execution because the *value is accessible to all function definitions* in the programs. The new value assigned to the global variable will replace the previous value stored and this new value will then be used throughout the whole program.

The global variable is typically *declared outside* of any function in the program and they remain in memory until the program ends.

```
#include<iostream>
using namespace std;
// global variable
int global = 5;

// main function
int main()
{
    // local variable with same
    // name as that of global variable
    int global = 2;

    cout << global << endl;
}
```

Global Variable

Local variable

Figure 5.13 Global variable vs. local variable

Local variables are used inside the function that declared them.

Global variables are sometimes useful in creating variables and constants that must be shared between many functions. However, declaring a variable as a global should be avoided due to the following *drawbacks*:

1 - It allows *unintentional errors* to occur when a function that does not need to access the function accidentally changes the variable's contents.
2 - Functions allow the tasks to be independently executed and reusable, on the other hand the use of global variables makes the program more *dependent* on them.

Thus, if more than one function needs access to the same variable, it is better to create a local variable in one of the functions and the pass that variable only to the function that need it.

5.3.2 Parameter Passing by Value

When passing by value, only *a copy of that value* is passed from the calling function to the called function through the parameters. The changes are made to the copied value do not change the original value.

Example #1(Pass by Value)

```
#include <iostream.h>

// function prototype of passOneValue
void passOneValue(int);

void main(void)
{
        int number;

        passOneValue(50);  // a function call to passOneValue() with a constant argument 50
        number = 100;
        passOneValue(number);  /*a function call to passOneValue() with a variable
        argument number*/
}

// function definition with a parameter named number of type integer
void passOneValue(int number)
{
        number = number * 2;
        cout << "The Number is " << number << "\n";
}
```

Figure 5.14 Example of pass by value

SOURCE FROM:
[1]MALIK, D.S, *C++ PROGRAMMING FROM PROBLEM ANALYSIS TO PROGRAM DESIGN*, THOMSON COURSE TECHNOLOGY, 2013,
[2]NORIZAN M., & MAZIDAH P., *PROBLEM SOLVING WITH C++*, UPENA UITM SHAH ALAM, 2006