United International University

Dept. of Electrical and Electronic Engineering (EEE)

Course No. : **EEE 122**
Course Title: **Structured Programming Laboratory**

Lab Sheet 5
Functions in C

Outcomes

After finishing this lab students should be able to ...

1. Construct programs modularly from small pieces called functions.
2. create new functions.
3. use the mechanisms that pass information between functions.
4. learn how the function call/return mechanism is supported by the function call stack and stack frames.
5. write and use functions that call themselves

Contents

1 C - Functions

In C programming, a function is a block of code that performs a specific task.Every C program has at least one function, which is **`main( )`**, and all the most trivial programs can define additional functions.

Suppose you are building an application in C language and in one of your program, you need to perform a same task more than once. So in such scenario you have two options

i. Use the same set of statements every time you want to perform the task

ii. Create a function, which would do the task, and just call it every time you need to perform the same task.

Using option (ii) is a good practice and a good programmer always uses functions while writing codes.

1.1 Uses of C functions

The uses of function in C programming is as follows -

i. C functions are used to avoid rewriting same logic/code again and again in a program.

ii. A large C program can easily be tracked when it is divided into functions.

iii. The core concept of C functions are, re-usability, dividing a big task into small pieces to achieve the functionality and to improve understandability of very large C programs.

iv. There is no limit in calling C functions to make use of same functionality wherever required.

v. We can call functions any number of times in a program and from any place in a program.

1.2 Types of C functions

Depending on whether a function is defined by the user or already included in C compilers, there are two types of functions in C programming.

i. Standard library functions

ii. User defined functions

Standard library functions

The standard library functions are built-in functions in C programming to handle tasks such as mathematical computations, I/O processing, string handling etc.

These functions are defined in the header file. When you include the header file, these functions are available for use.

The `printf()` is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in `"stdio.h"` header file.

There are other numerous library functions defined under `"stdio.h"`, such as `scanf()`, `fprintf()`, `getchar()` etc. Once you include `"stdio.h"` in your program, all these functions are available for use.

User defined functions

C allow programmers to define functions. Such functions created by the user are called user-defined functions.

Depending upon the complexity and requirement of the program, you can create as many user-defined functions as you want.

1.3 Defining a Function

The general form of a function definition in C programming language is as follows:

```
return_type function_name( parameter list );//function prototype or declaration
int main void(){
  main program
}
//defining function
return_type function_name( parameter list ) {
  body of the function
}
```

A function definition in C programming consists of a function header and a function body. function header consists of Return type, Function name, Parameters. Here are all the parts of a function

- **Return Type:** A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.
- **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body:** The function body contains a collection of statements that define what the function does.

1.4 How does function work

```
#include <stdio.h>

functionName(); //function prototype or declaring

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

  functionName();
  ... .. ...
  ... .. ...
}
//defining function
void functionName(){
  ... .. ...
  ... .. ...
}
```

The execution of a C program begins from the `main( )` function. When the compiler fetch prototype `functionName( );` it memories and when after inside the main function encounter this function name it understood that a function with the same name has been declared after the main function, then control of the program jumps to

```
 void functionName()
```

And, the compiler starts executing the codes inside the user-defined function.

The control of the program jumps to statement next to `functionName( );` once all the codes inside the function definition are executed.

Function Declarations

A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts

```
return_type function_name( parameter list );
```

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

Calling a Function

While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.

When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.
To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value. For example:

```
#include <stdio.h>

/* function declaration */
int max(int num1, int num2);

int main (void) {
  /* local variable definition */
  int a = 100;
  int b = 200;
  int ret;

  /* calling a function to get max value */
  ret = max(a, b);

  printf( "Max value is : %d\n", ret );

  return 0;
}

/* function returning the max between two numbers */
int max(int num1, int num2) {
  /* local variable declaration */
  int result;

  if (num1 > num2)
  result = num1;
  else
```

```
  result = num2;

  return result;
}
```

We have kept `max( )` along with `main( )` and compiled the source code. While running the final executable, it would produce the following result

1.5 Function Arguments

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the formal parameters of the function.

Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

While calling a function, there are two ways in which arguments can be passed to a function

i. **Call by value** This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

ii. **Call by reference** This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

By default, C uses call by value to pass arguments. In general, it means the code within a function cannot alter the arguments used to call the function.

2 Programming Examples

Example: 1

Description: Write a C program that asks the user to enter two integers and find their sum

Source Code

```
#include<stdio.h>
int sum(int x, int y);
int main(void){
  int a,b,s;
  printf ("Enter two numbers: ") ;
  scanf("%d %d",&a,&b);
  s=sum (a,b);/*Function call */
  printf("Sum of %d and %d is %d\n",a,b,s);
  return 0;
}
int sum(int x, int y){
  int s;
  s =x+y;
  return s;
}
```

Output

```
Enter two numbers:6 9
Sum of 6 and 9 is 15.
```

Example: 2

Description:Write a C program that returns the sum of squares of all odd numbers from 1 to 25.

Source Code

```
#include <stdio.h>
int func (void);
int main(void){
  printf ("The sum is: %d\n", func( ) );
  return 0;
}
int func(){
  int num, s=0;
  for(num=1;num<=25;num++){
    if(num%2!=0)
    s+=num*num;
  }
  return s;
}
```

Output

```
The sum is: 2925
```

Example: 3

Description: Write a C program to find sum of digits of an integer.

Source Code

```
#include <stdio.h>
int sum (int n);
int main(void){
  int num;
  printf ("Enter the number: ") ;
  scanf("%d",&num) ;
  printf ("Sum of digits of %d is %d\n",
            num, sum(num) );
  return 0;
}
int sum (int n){
  int i, sum=0, rem;
  while(n>0){
    rem=n%10;
    sum+=rem;
    n/=10;
  }
  return sum;
}
```

Output

```
Enter the number: 2437
Sum of digits of 2437 is 16
```

Example: 4

Description:Write a C program to find factorial of an integer using recursion.

Source Code

```
#include<stdio.h>
int fact(int n);
int main(void){
  int n,result;
  printf("\n Enter the number:");
  scanf("%d",&n);
  result=fact(n);
  printf("\n The factorial is=%d",result);
  return 0;
}
int fact(int m){
  if(m == 0) return 1 ;
  return m*fact(m-1);
}
```

Output

```
Enter the number:5

The factorial is=120
```

3 Practice session

Sl	Source Code

Practice 1	`#include <stdio.h>`
	`void func(int a,int b);`
	`int main(void){`
	`  func(2,3) ;`
	`  return 0;`
	`}`
	`void func(int a,int b){`
	`  int s;`
	`  s=a+b;`
	`  printf("Sum=%d",s);`
	`}`

Practice 2	`#include<stdio.h>`
	`int func(int a);`
	`int main(void){`
	`  int x=5,y;`
	`  y=func(x) ;`
	`  printf("%d\n",y);`
	`  return 0;`
	`}`
	`int func(int a){`
	`  a=a*2;`
	`  return a;`
	`}`

Practice 3	`#include<stdio.h>`
	`void func (int a, int b);`
	`int main (void){`
	`  int i=5,j=10;`
	`  func(i/2,j%3) ;`
	`}`
	`void func(int a,int b){`
	`  a=a/2;`
	`  b--;`
	`  printf("%d\t",a+b) ;`
	`  return;`
	`}`

Practice 4	`#include<stdio.h>`
	`int fun(int n);`
	`int main(void){`
	`  int n=8;`
	`  printf("%d\n",fun(n)) ;`
	`  return 0;`
	`}`
	`int fun(int n){`
	`  if(n==0)`
	`  return 0;`
	`  else`
	`  return(n+fun(n-1));`
	`}`

4 Lab Assignments

1. Write a program using user defined function to print all the Armstrong numbers between 1 and 500. [An Armstrong Number is a number which is equal the sum of cubes of its digits. For example, 153 is an Armstrong number, because $153 = 1^3 + 5^3 + 3^3$]
2. Write a program using a recursive function to find the gcd (greatest common divisor) of two given numbers.
3. Write a program using a recursive function to findthe lcm (lowest common multiple) of two given numbers.
4. Write a program using a recursive function to findthe nth Fibonacci number. n is taken from user.
5. Write a recursive function to find the sum of digits of a number.
   Use the logic:

$$sumd(n) = \begin{cases} 0 & n = 0 \quad (1) \\ \text{n\%10+sumd(n/10)} & n > 0 \quad (2) \end{cases}$$

Acknowledgment

First OBE version, prepared by:
B.K.M. Mizanur Rahman,
Assistant Professor,
Department of EEE, UIU

Second Update , prepared by:
Nazmul Alam,
Part Time Faculty,
Department of EEE, UIU