Experiment No. 4

Familiarization with Functions and Recursion

1. **Objective:** This experiment is designed to demonstrate the use of function and its effectiveness in C programming and also demonstrates how recursion makes function definition easier.

2. **Theoretical Background:** A function is a set of statements to perform a certain task. In C program there are two types of function
    - Library function : scanf, printf, gets, puts, getch, sqrt etc.
    - User defined function

    C program contains at least one function which is main(), and main() must be present exactly once in a program. There is no limit on the number of functions defined and there can be functions which are defined but not called.

    Defining a Function:

    While declaring a function we need to specify certain things: function's name, return type, and parameters. In C programming the general form of a function definition is as follows:

```
return_type name_of_function( parameter list )
 {
   body of the function
}
```

- **Return Type** − A function may return or may not return a value. The return_type is the type of data that the function is supposed to return. If no return type is specified then default int is assumed. When the return statement is encountered: the function returns immediately and does not execute whatever is written after the return statement. Return statement can be used without return value, return; . This type of return is used mostly by void functions (no return of value). In C programming more than one value can not be returned.

- **Function Name** − It is the name of the function. Two or more functions can not have the same name, also function and variable names can not be the same.

- **Parameters** − To perform a certain task a function may need some input from the user. For this reason, to take arguments a function must have special variables known as formal parameters and parameters are like a placeholder. While invoking a function the value that is passed to the function is known as argument. Parameters are optional so there can be functions without any parameters.

- **Function Body** − The function body contains a set of statements that are designed to perform a certain task. No statements can be written outside of a function

Note: In C programming one function can not be defined inside another function but one function can be called from another function.

Function Examples:

A function definition in C programming consists of a function header and a function body. Here are all the parts of a function −

```
int sub(int x, int y)
{
  return x-y;
}
```

This function takes two integer numbers x & y and returns x-y. In main function it can be called with values as,

```
sub(2, 6)
```

Local Variables: Local variables are variables that are declared inside a function or block and they can be used only by statements within that function. Local variables of one function can not be accessed by another function or outside. The following example shows how local variables are used. Here all the variables a, b, and c are local to main() function.

For example the following code will generate error as variable i local to main only,

```
#include <stdio.h>

void myfunc(void)

{

  printf("i is %d\n", i);

}

int main(void)
```

```
{
  int i=10;
  myfunc();
  return 0;
}
```

Global Variables: Global variables are defined outside a function, usually on top of the program and they can be accessed by any of the functions.

For example the following code will generate no error as variable i is global now and can be accessed by any function.

```
#include <stdio.h>
int i;
void myfunc(void)
{
  printf("i is %d\n", i);
}
int main(void)
{
  i=10;
  myfunc();
  return 0;
}
```

Functions Calling: Functions can be called by value also by reference. Function that is called by value will have no effect on the argument used to call. On the other hand, in case of call by reference address of an argument is copied into the parameter. By default C uses 'call by value'

Returning multiple values: Multiple values can be returned using a global variable or call by reference.

Recursion: In programming languages, if a function is defined in terms of itself, then it is called a recursive function. This type of definition is also known as circular definition. Here, the function is made to call itself. When a function calls itself recursively, each invocation gets a fresh set of all the automatic variables. Recursive code is more compact and often much easier to write. To define a recursive function it must have a recursive definition and a terminating condition. An example of recursive function will be demonstrated in this experiment.

3. Software requirements:

- Any IDE that supports a C compiler (preferably Code::Blocks; any other IDE or text editor with a C compiler installed will also be accepted.)

4. Example Problems:

a. Write a function power(a, b), to calculate the value of a raised to b

Code:

```
#include<stdio.h>
int main()
{
    int num, pow;
    int val;
    printf("Enter the number: ");
    scanf("%d", &num);
    printf("\nEnter the power: ");
    scanf("%d", &pow);

    val = power(num,pow);
    printf("\nThe result is = %d", val);

    return 0;
}

int power(int a, int b)
{
    int i, res = 1;
    for (i = 1; i <= b; i++)
    {
        res = res*a;
    }
    return res;
}
```

b. A positive integer is entered through the keyboard. Write a function to obtain the prime factors of this number. For example, prime factors of 24 are 2, 2, 2 and 3, whereas prime factors of 35 are 5 and 7.

Code:

```
#include <stdio.h>

int main()
{   int i, num, isPrime;

    /* Input a number from user */
    printf("Enter any number to print Prime factors: ");
    scanf("%d", &num);

    printf("All Prime Factors of %d are: \n", num);

    /* Find all Prime factors */
    for(i=2; i<=num; i++)
        {
            /* Check 'i' for factor of num */
            if(num%i==0)
                {
                /* Check 'i' for Prime */
                isPrime = check_prime(i);

                /* If 'i' is Prime number and factor of num */
                if(isPrime==1)
                    {
                    printf("%d\n", i);
                    }
                }
        }


    return 0;
}
```

```
/*check_prime() is a function which returns if the input number
is prime, otherwise it returns 0 */
int check_prime(int n)
{
    int flag = 1,j;

    for(j=2; j<=n/2; j++)
        {
        if(n%j==0)
            {
            flag = 0;
            break;
            }
        }
        return flag;
}
```

c. Write a program to determine the factorial of a positive integer using recursive function

Code:

```
# include <stdio.h>
int main( )
{
    int a, fact ;
    printf ( "Enter any number " ) ;
    scanf ( "%d", &a ) ;
    fact = rec (a) ;
    printf ( "Factorial value = %d\n", fact ) ;
    return 0 ;
}

int rec(int  x)
{

    int f ;
    if (x == 1)
        {
        return 1;
        }

    else
        {
        f = x * rec (x - 1) ;
        }
```

```
    return f;
}
```

5. **Results after running the example programs:**
   Screenshots of the results:

6. **Lab report assignments (submit source code and screenshots of results):**
   a. Write a C program to find whether a number is Palindrome or Not using functions.
   b. Write a C program to get the nth Fibonacci term using recursion.
   c. Write a C program to find the sum of digits of a given number using recursion.

7. **Comment/Discussion on the obtained results and discrepancies (if any).**