# LAB 6: Functions in C Language

**OBJECT:**

*To study the implementation and operation of functions in C.*

**THEORY:**

The best way to develop and maintain a large program is to construct it from smaller pieces or modules each of which is more manageable than the original program. The modules are called differently in different programming languages. The common terms that are used are subroutines, procedures, subprograms and functions. In C, these modules are called functions. C functions can be classified into two categories.

**a.** Standard library functions

**b.** User defined functions (UDF)

A function is a self -contained block of code that performs a particular task. Once a function has been designed and packed, it can be treated as a 'black box' that takes some data from the main program and returns some value.

**An Example of C-Function**

Each program we have presented has contained a function called main() that called standard library functions to accomplish its tasks. We now consider how programmers write their own customized functions.

```
                        /*User Defined Function */
void printline( );
void main ( )
{
        clrscr( );
        printline ( );
        printf( "This illustrates the use of C functions\");
        printline( );
        getch( );
}

void printline( )
{
        int k;
        for ( k=1 ; k< 40; k + +)
        printf ( "-");
        printf("\n");
}
```

Any function that is called by another function is known as a called function. A function that calls another function is known as a calling function. When a function is called, control is transferred from the calling function to the called function. Execution starts from the first

executable statement in the called function. Once the called function finishes execution, control is handed back to the calling function. Execution resumes from the statement following the function call.
Every function in C Language has three parts which are:

**1. Function Declaration:**

Every function that is being referenced should be declared for its type in the calling function. The function definition combines the argument list and the argument declaration is called function prototype. Using function prototype in function declaration is that they enable the compiler check any mismatch of type and number of arguments between the function calls and the function definitions. They also help the compiler to perform automatic type conversions on function parameters.

**2. Function Call:**

Every function is invoked by the function call. Here Function **printline** is invoked or called by the main() function.

**3. Function Definition:**

It contains all the statements or the operations to be performed by the function and is defined outside the main( ).

**Return values and their types:**

A function may or may not send back any value to the calling function. The called function can return only one value per call.

***return;***

***return(expression);***

The 'plain' return does not return any value; it acts as the closing brace or the function. When a return is encountered, the control is immediately passed back to the calling function.

```
                              /* Calculator*/
#include<conio.h>
#include<stdio.h>
int square(int);
void main (void) /* Defining main function*/
 {
        clrscr(); /* Clears previous contents of screen*/
        int x;
        for(x=1;x<=5;x++)
        printf( Square of %d is %d\n ,x,square(x));
        getch();
 }
int square(int y)
 {
        return y*y;
 }
```

By default a function returns int type of data. We can force a function to return a particular type of data by using a type specifier in the function header.

***double product ( double x, double y);***

***float sqroot( float p);***

***Common Programming Errors***

- Forgetting the semicolon at the end of the prototype is a syntax error

return_type *fun_name* (parameter_type)**;**

- Defining a function inside another function is a syntax error.
- Defining a function parameter again as a local variable with in the function is a syntax error.

```
int square (int y)
{
int y=5;
}
```

- Omitting the *return-value-type* in a function definition is a syntax error if the function prototype specifies a return type other than int.
- A function prototype placed outside any function definition applies to all calls to the function appearing after the function prototype.

**EXERSISE**

**Q#01:** C Language functions can be categorized into two types called

_Library functions and _User defined Functions_

. **Q#02: __A function___**is a self -contained block of code that performs a particular task.

**Q#03:** A function that calls another function is known as _Calling Function_

**Q#04:** Every function has three parts called: __**Function Declaration, Function Definition, and Function Call**.

**Q#05:** Create a function that when called take two numbers as input and print their sum.

Answer: #include <stdio.h>

```
void sum(int a, int b) {

   printf("Sum: %d\n", a + b);

}
```

```
int main() {

    int num1, num2;

    printf("Enter two numbers: ");

    scanf("%d %d", &num1, &num2);

    sum(num1, num2);

    return 0;

}
```

**Q#06:** Write a program using functions to calculate the average of two numbers and returns the maximum number from the two numbers (HINT: Define two functions avg and max)

Answer: #include <stdio.h>

```
float avg(float a, float b) {

    return (a + b) / 2;

}

int max(int a, int b) {

    return (a > b) ? a : b;

}
```

```
int main() {

    float num1, num2;

    printf("Enter two numbers: ");

    scanf("%f %f", &num1, &num2);



    printf("Average: %.2f\n", avg(num1, num2));

    printf("Maximum: %.2f\n", max(num1, num2));



    return 0;

}
```

**Q#07:** Write a program for calculator that performs simple calculations: Add, Subtract, Multiply, and Divide. Furthermore, the operation continues until the user enters character ‘n’ in that case the program ends. (HINT: You may use switch or else-if construct for implementing calculator.)

Answer:
```
#include <stdio.h>


void calculator() {
```

```
char choice;

do {

    int num1, num2;

    char op;


    printf("Enter an operation (+, -, *, /): ");

    scanf(" %c", &op);

    printf("Enter two numbers: ");

    scanf("%d %d", &num1, &num2);


    switch (op) {

        case '+': printf("Result: %d\n", num1 + num2); break;

        case '-': printf("Result: %d\n", num1 - num2); break;

        case '*': printf("Result: %d\n", num1 * num2); break;

        case '/':

            if (num2 != 0)

                printf("Result: %.2f\n", (float)num1 / num2);

            else

                printf("Error! Division by zero.\n");

            break;

        default: printf("Invalid operator!\n");

    }
```

```
        printf("Do you want to continue? (y/n): ");

        scanf(" %c", &choice);

    } while (choice == 'y' || choice == 'Y');

}



int main() {

    calculator();

    return 0;

}
```

## LAB 7: Passing arguments to a Functions

**OBJECT:**

*To study the mechanism of passing arguments to a function.*

**THEORY:**

The term argument refers to any expression or value within the parentheses of a function call. The term parameter refers to any declaration within the parentheses following the function name in a function declaration or definition. Arguments are separated by commas. Arguments are passed by value; that is, when a function is called, the parameter receives a copy of the argument's value, not its address. This rule applies to all scalar values, structures, and unions passed as arguments. Modifying a parameter does not modify the corresponding argument passed by the function call. However, because arguments can be addresses or pointers, a function can use addresses to modify the values of variables defined in the calling function. The scope of function parameters is within the function itself. Therefore, parameters of the same name in different functions may be used.

**Functions with arguments and no return values:**

Supposing we wanted to write a function that is capable of drawing a line of variable length and with any character, then we have to write a function that will receive two parameters - the character to be used for drawing the line and the length of the line.

```
                    /* Passing Arguments to a Function */
void printline ( char ch, int k );
void main ( )
{
        char ch2 = '*';
        int i = 45;
        clrscr( );
        printline (ch2, i );
        printf("This text is sandwiched between two lines \n");
        printline('*', 45);
        getch( );
}
void printline ( char ch, int k )
{
        int j;
        for ( j = 0; j < length; j + +)
        printf( "%c", line_char);
        printf ("\n");
}
```

The arguments ch and k called the formal arguments. The calling function can now send values to these arguments using function calls containing appropriate arguments.
The function call
***printline('*', 45);***
would send values '*' , 45 to the function and
***printline (char ch, int k)***
assign '*' to ch , and 45 to k.
The values 45 and '*' are the actual arguments which become the values of the formal arguments inside the called function. The actual and formal arguments should match in number, type and order. The values of actual arguments are assigned to the formal arguments on a one to one basis. If, the actual arguments are more than the formal arguments ( m > n) the extra actual arguments are discarded. If the actual arguments are less than the formal arguments, the unmatched formal arguments are initialized to some garbage values. Any mismatch in data type may also result in passing of garbage values. The formal arguments must be valid variable names. The actual arguments may be variable names, expressions, or constants.

**Functions with arguments and return values:**
On some occasions, a two way communication is required. The calling function provides some information to the called function. The called function then uses this information to perform some computation and then returns the result of the computation back to the calling function. In order for a function to return a value, the function must be declared accordingly. ***Ret_type function_name ( argument list)***

***{***
 ***declarations***
 ***statements***
***}***

Where ret_type specifies the data type of the value that is returned by the function. If the return type of a function is not specifies, it is assumed to return a value of int. If the function does not return any values to its called, its return type should be specified as void.

```
                    /* prints the largest of two integers */

 double bigger(double n1, double n2);
 void main(void) {
  double number1,number2, max;
  clrscr( );
  printf("Please input two numbers :");
  scanf("%lf %lf", &number1, &number2);
  max = bigger(number1, number2); // function call
  printf("The max of %.lf and %.lf is %.1f\n",number1,number2,max);
  getch( );}
 // function definition
 double bigger(double n1, double n2)
 {
  double larger;
  if(n1 > n2)
  larger = n1;
  else
  larger = n2;
  return larger;
  }
```

**EXERCISE**:

**Q#01:** Write a program using function to swap the values of variables.

Answer: #include <stdio.h>

```
void swap(int *a, int *b) {

   int temp = *a;

   *a = *b;
```

```
    *b = temp;
}

int main() {
    int x, y;

    printf("Enter two numbers: ");
    scanf("%d %d", &x, &y);

    printf("Before swapping: x = %d, y = %d\n", x, y);
    swap(&x, &y);
    printf("After swapping: x = %d, y = %d\n", x, y);

    return 0;
}
```

**Q#02:** Write a program using function to calculate the cube of a given number using pass by reference technique.

Answer:
```
#include <stdio.h>

void calculateCube(int *num) {
    *num = (*num) * (*num) * (*num);
}
```

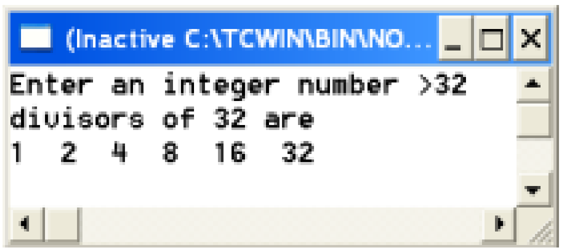
```
int main() {
    int number;


    printf("Enter a number: ");
    scanf("%d", &number);


    calculateCube(&number);
    printf("Cube of the given number is: %d\n", number);


    return 0;
}
```

**Q#03:** Write a function that receives an integer number from the user and displays its divisors on the screen including 1 and itself.

Answer:
```
#include <stdio.h>



void findDivisors(int num) {

    printf("Divisors of %d: ", num);

    for (int i = 1; i <= num; i++) {

        if (num % i == 0) {
```

```c
            printf("%d ", i);
        }
    }
    printf("\n");
}


int main() {
    int num;


    printf("Enter an integer: ");
    scanf("%d", &num);


    findDivisors(num);


    return 0;
```

}



```
Enter an integer number >32
divisors of 32 are
1   2   4   8   16   32
```