MODULE-4
USER DEFINED FUNCTIONS AND RECURSION

Definition

A function is a collection of statements that perform a specific task

These functions are very useful to read write and debug complex programs

Types of Functions

These can be broadly classified into two types

(i) Built-in functions

(ii) User defined functions

C provides various built-in functions, few of them are as follows

String manipulation functions:- The header file which includes string manipulation functions is "#include<string.h>"

Some of the string manipulation functions are:

- strlen( ) :- returns the length of the string.
- strcmp( ) :-used to compare two strings.
- strcpy( ) :-used to copy one string to another
- strcat( ) :- used to concatenate two strings
- strrev( ) :- used to find reverse of the string
- strlwr( ) and strupr( ) :- to convert strings to lower case and uppercase respectively

Character manipulation functions :- the header file that includes character manipulation functions is "#include<ctype.h>"

Some of the character manipulation functions are:

- isalnum(ch) :- returns true if ch is an alphabet or a number
- isalpha(ch) :- returns true if ch is an alphabet
- isascii(ch) :- returns the ASCII value in the range of 0 through 127

Mathematical functions: - the header file that includes Mathematical functions is "#include<math.h>"

Some of the Mathematical functions are:

- fabs(n) :- returns absolute value of the floating point number
- ceil(n) :- returns smallest integer value grater than or equal to n
- floor(n) :- returns largest integer value smaller than or equal to n
- exp(n) :- returns e raised to the given power n
- sqrt(n) :- returns the square root of n

User defined functions: - The user defined function is defined by the user according to its requirements instead of relying only on the built-in functions C allows us to create our own function called user defined function

Parts of user defined function.

(i)Function Declaration or Function prototype

(ii)Function call or calling Function

(iii)Function Definition or defining a function

Function Declaration or Function prototype :- It will inform the compiler about the return type, function name and number of arguments along with the data types.

syntax:

```
return_type  function_name(argument _list);
```

Where,

return_type :- is the data type of the value that is returned or sent from the function.

function_name :-function should be given a descriptive name

argument _list :- contains type and names of the variables that must be passed to the function.

Example:-

```
int  large (int x, int y);
```

is a function declaration with function_name “large” with return _type “integer” and has two arguments “x” and “y” of integer type.

***NOTE:*-**

if we define a function before main ( ) function the there is no need of function declaration

if we define the function after main ( ) function then it is mandatory to declare the function because it will inform the compiler.

Function call or calling function_:- invoking the function with valid number of arguments and valid data type is called as function call.

To call a function one simply needs to pass the required parameters along with the function name and if the function returns the value then one can store the returned value.

Syntax:

```
        function_name(argumement_list);
```

Where,

function_name :- descriptive name given to the function

argumement_list :- consist of constant(s) , variable(s), or Expression(s).

Example:-

```
        large (m,n);
```

The function can be invoked in various ways

- large(m,n); //m and n are variables.
- large(5,8); //5 and 8 are constants
- large(5+2,6); // The first argument is an expression which is evaluated to 7
- large(2*3,5+3); //is an expression which is equivalent to large(6,8);

Function definition or Defining a function :- The declared function must define the same to perform the specific task.

Syntax

```
        return_type  function_name(argument _list)
        {
                local_variable_declaration;
                Body of the function;
        }
```

Where,

return type :- when the function is called the function may or may not return a value

If the function returns a value then the return_type will be any appropriate data type (int, float, char etc) and we use the keyword “return” to return the value.

If the function does not return a value then the return_type will be “void” and no need to use the keyword “return”

function_name:- is the name of the function.

argument _list :- these are also called as parameters. the argument_list refers to the type order and number of parameters of the function.

local_variable_declaration:-these are temporary variables which are required only within this function.

function _body: - The body of the function contains the collection of statements that define what the function does.

when the program makes the function call the program control is transferred to the called function. This called function performs the defined task and returns the program control back to the main( ) function.

Example for Program written using Functions

```
/* C program to find area of circle using functions */

#include<stdio.h>
float area(float r);
void main()
{
        float r,x;
        printf("Enter the radius\n");
        scanf("%f",&r);
        x=area(r);
         printf("Area ofcircle= %f\n",x);
}
float area(float r)
{
        float x;
        x=3.142*r*r;
        return x;
}
```

Types of user defined function or Categories of functions based on arguments and return values:

- Function with argument with return value.
- Function with argument without return value.
- Function without argument with return value.
- Function without argument without return value.

1. Function with argument with return value.

The arguments are passed from calling function to the called function.
Based on the received argument values, the called function performs the required action and returns the value back to calling function (main( ) function).

```
/* C program to demonstrate Function with argument with return value */

#include<stdio.h>
int add(int a, int b);
```

```
void main()
{
        int a,b,sum;
        printf("Enter two numbers\n");
        scanf("%d%d",&a,&b);
        sum=add(a,b);
        printf("The Sum of two numbers=%d\n",sum);
}


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

2. Function with argument without return value.

The arguments are passed from calling function to the called function.
Based on the received argument values the called function performs the required action but does not return any value back to calling function (main( ) function).

```
/* C program to demonstrate Function with argument without return value */

#include<stdio.h>
void sum(int a, int b);
void main()
{
        int a,b;
        printf("Enter two numbers\n");
        scanf("%d%d",&a,&b);
        sum(a,b);
}
void sum(int a, int b)
{
        int x;
        x=a+b;
        printf("Result=%d\n",x);
}
```

3. Function without argument with return value.

Here no arguments are passed from calling function to the called function.

The called function performs the required action by taking the necessary arguments and returns the value back to calling function (main( ) function).

/* C program to demonstrate Function without argument with return value */

```
#include<stdio.h>
int sum();
void main()
{
        x=sum();
        printf("Result=%d\n",x);
}
int sum()
{
        int a,b,x;
        printf("Enter two numbers\n");
        scanf("%d%d",&a,&b);
        x=a+b;
        return x;
}
```

4. Function without argument without return value.

Here no arguments are passed from calling function to the called function.
The called function performs the required action by taking the necessary arguments but does not return any value back to calling function (main( ) function).

/* C program to demonstrate Function without argument without return value */

```
#include<stdio.h>
void sum();
void main()
{
        sum();
}
void sum()
{
        int a,b;
        printf("Enter two numbers\n");
        scanf("%d%d",&a,&b);
        x=a+b;
        printf("Result=%d\n",x);
}
```

Local variables : - local variables are also called internal variables

Functions including main( ) function declare the variables to store temporary values. These variables are called local variables.

The scope of the variable is only within the function it is declared with.

Global variables :- global variables are also called external variables

These are the variables which are used by or accessible from any function present in the program

These variables are declared before the main( ) function.

Register and Static variable :- register keyword is used to define a local variable that should be stored in register instead of RAM **.**They should be used for the variables that requires quick access

Static variable is used to store the variable in statically allocated memory instead of automatically allocated memory

Calling function and called function :-

The function main( ) that calls another function is called calling function

The function being called by the calling function is known as called function.

Actual arguments and Formal arguments :-

When the function is called, the values that are passed in the call are called as actual parameters.

The formal parameters are written in the function prototype or function declaration and function header of the definition .

These are called as dummy parameters which are assigned the values from the arguments when the function is called.

/* C program to demonstrate actual arguments and formal arguments */

```
#include<stdio.h>
int perimeter(int x,int y);        // int x, int y are formal parameters
void main()
{
        int l,b,p;
        printf("Enter length and breadth\n");
        scanf("%d%d",&l,&b);
        p=perimeter(l,b);        // function call with actual parameters l and b
        printf("Perimeter of Rectangle=%d\n",p);
}
int perimeter(int x,int y)
{
        int per;                 //int per is a local variable
        per=2*(x+y);
        return per;
}
```

Parameter passing mechanism :- There are two methods by which parameters or arguments can be passed to the function

(i) Call by value or Argument passing by value

(ii) Call by reference or Argument passing by reference

Call by value or Argument passing by value :-When an variable or value is passed to an function during its invocation such function invocation is called as call by value.

/*C program to demonstrate call by value or Argument passing by value */

```
#include<stdio.h>
int sum(int n);
void main()
{
        int n,x;
        printf("Enter the value of n\n");
        scanf("%d",&n);
        x=sum(n);
        printf("Sum of natural numbers=%d\n",x);
}
int sum(int n)
{
        int res=0,i;
        for(i=1;i<=n;i++)
        res=res+i;
        return res;
}
```

Call by reference or Argument passing by reference :- when the address of the variable is passed to the function during its invocation such a function is called as call by reference

C does not support directly passing by reference. It is done indirectly by passing the address of the variable and changing the value of the variable through its address.

/*C program to demonstrate call by reference or Argument passing by reference */

```
#include<stdio.h>
void swap(int *a,int *b);
void main()
{
        int a,b;
        printf("Enter two numbers\n");
         scanf("%d%d",&a,&b);
        printf("Before Swapping\n a=%d\t b=%d\n",a,b);
        swap(&a, &b);
        printf("After Swapping\n a=%d\t b=%d\n",a,b);
}
void swap(int *a, int *b)
{
         int temp;
        temp=*a;
        *a=*b;
         *b=temp;
}
```

Passing Arrays to functions :- Array elements or an entire array can be passed to a function such a mechanism is called a s passing array to the function.

```
/* program to demonstrate passing array to the functions */

#include<stdio.h>
int largest(int a[20],int n);
void main()
{
        int a[20],n,i,max;
         printf("Enter the value of n\n");
        scanf("%d",&n);
        printf ("Enter %d values\n",n);
         for(i=0;i<n;i++)
                scanf("%d",&a[i]);
         max=largest(a,n);
        printf("Largest element in array=%d\n",max);
}
int largest(int a[20],int n)
{
        int max,i;
        max=a[0];
        for(i=1;i<n;i++)
        {
                if(a[i]>max)
                        max=a[i];
        }
        return max;
}
```

Recursion :- when a function calls itself again and again it is called as recursion

Here the calling function and called function are same.It is a powerful technique of writing complex algorithms in an easier way. The recursive function must always have a stopping condition or an exit condition in the function else the program will go into infinite loop

Factorial of a given Number:-we can calculate the factorial of a given number using the formula

n!= (n) *(n-1)*(n-2)*… *1

There are several approaches to solve a given problem like useage of while do- while or for loop. Now, let us try to find the solution to this classic example using the concept of recursion.

```
/* C program to find factorial of the number using recursion */

#include<stdio.h>
int factorial(int n);
void main()
{
        int n,res;
        printf("Enter the value of n\n");
        scanf("%d",&n);
        res=factorial(n);
        printf("The factorial of the number=%d\n",res);
}


int factorial(int n)
{
        if(n==0)
                return 1;
        else
                return n*factorial(n-1);
}
```

Generation of Fibonacci series :- In Mathematics Fibonacci series or Fibonacci numbers are thenumbers that are displayed in the following sequence:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ………….

If you observe the above pattern sequence the First number is 0 the Second number is 1and the subsequent number is the result of sum of the previous two numbers,(Example Third number is 0+1=1,Fourth number is 1+1=2 and so on..) There are several approaches to solve a given problem like useage of while do- while or for loop. Now, let us try to find the solution to this classic example using the concept of recursion.

```
/* C program to generate fibonacci series upto N numbers using recursion */

#include<stdio.h>
int fibonacci(int n);
void main()
{
        int num,i;
        printf("Enter the number up to which you want\n");
        scanf("%d",&num);
        printf("\nFibonacci Series\n");
        for(i=0;i<num;i++)
        {
                printf("%d\t",fibonacci(i));
         }
}
int fibonacci(int n)
{
        if(n==0)
                return 0;
        else if(n==1)
                 return 1;
        else
                        return (fibonacci(n-1)+fibonacci(n-2));
}
```

/* C program to compute binomial coefficient using recursion */

$$nCr = \frac{n!}{(n-r)!\,r!}$$

```
#include<stdio.h>
int bino_coefficient(int n, int r)
{
          if(r==0||r==n)
                return1;
          else
               return bino_coefficient(n-1,r-1)+ bino_coefficient(n-1,r);
}
```

```
void main()
{
        int n,r res;
        printf("Enter the value of n and r\n");
        scanf("%d%d",&n,&r);
        res= bino_coefficient(n,r);
        printf("The Result of binomial coefficient =%d\n",res);
}
```