UNIT IV

FUNCTIONS

Functions:

Functions are the building blocks of C..

All programs definitely consist of one function –main()

Normally a big program is broken down into a number of functions to reduce complexity.

If a program is divided into functional parts, then each part may be independently coded and later combined into a single unit.

Subprograms are called 'functions' which are much easier to understand, debug, and test.

Functions give modularity to the code

Elements of a Function

1.Function declaration or Function prototype
 the function should be declared prior to its usage
2.Function definition
 implementing the function or writing the function consists of function header and function body
3.Function call or function invocation
 to use the function, the function should be called

User Defined Functions

A function consists of two parts:

1. Function Header
   - function name
   - function return type
   - list of parameters
2. Function Body
   - local variable declarations
   - function statements
   - return statement

Function Declaration and Function Prototypes

A function prototype is a function declaration that specifies the data types of its arguments in the parameter list. The compiler uses the information in a function prototype to ensure that the corresponding function definition and all corresponding function declarations and calls within the scope of the prototype contain the correct number of arguments or parameters, and that each argument or parameter is of the correct data type.

All identifiers in C need to be declared before they are used. This is true for functions as well as variables. For functions the declaration needs to be before the first call of the function. A full declaration includes the return type and the number and type of the arguments. This is also called the function prototype.

Having the prototype available before the first use of the function allows the compiler to check that the correct number and type of arguments are used in the function call and that the returned value, if any, is being used reasonably.

The function definition itself can act as an implicit function declaration

Note: Older versions of the C language didn't have prototypes, the function declarations only specified the return type and did not list the argument types.

Old style:

- Functions can be declared implicitly by their appearance in a call.
- Arguments to functions undergo the default conversions before the call.
- The number and type of arguments are not checked.

Prototype style:

- Functions are declared explicitly with a prototype before they are called. Multiple declarations must be compatible; parameter types must agree exactly.
- Arguments to functions are converted to the declared types of the parameters.
- The number and type of arguments are checked against the prototype and must agree with or be convertible to the declared types. Empty parameter lists are designated using the void keyword.

Prototype Syntax

A function prototype has the following syntax:

***function-prototype-declaration*:**

declaration-specifiers(opt) declarator;

- Function prototypes are always declared at the beginning of the program indicating the **name of the function**, the **data type of its arguments** which is passed to the function and the **data type of the returned value** from the function.
- **Examples with a function prototype**

  Ex 1:

```
#include<stdio.h>
 int add(int,int);
 main()
 {  }
```

The parameters are valid variable names separated by commas and are declared after the function header with proper data types.

Ex 2:

```
#include<stdio.h>
int add(int a,int b);// function prototype
main()
{
     int i=10,j=20,k;
     add(i,j); //parameter passing
     getch();
     }
     int add(int a,int b)
          { int c;
         c=a+b;
         printf("%d",c);
         }
```

Ex 3:

```
#include<stdio.h>

main()

{

    int i=10,j=20,k;

    add(i,j);

    getch();

    }

    add(a,b)

       { int c;

      c=a+b;

      printf("%d",c);

      }
```

Ex.4:

```
  #include <stdio.h>

  int sum (int, int);

  int
  main (void)
  {
    int total;

    total = sum (2, 3);
    printf ("Total is %d\n", total);

    return 0;
  }

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

The prototype gives a lot of information about the function. One, it tells it the return type. Two, it tells it how many parameters there are, and what their types are. The actual names of the parameter values (a and b in our example) can be left in or out of the prototype. Generally, leaving them out leaves you with the flexibility of renaming variables at will.

Prototypes are often in a separate header file which can be included in other C source files that wish to use the functions. The header stdio.h, for example, contains prototypes for the functions scanf and printf. This is why our examples have been including this file, so that the compiler will let us call those functions. The actual code for these functions is kept in a library elsewhere on the system, the header file only says how to interface with the functions

ADVANTAGES OF FUNCTIONS:

Reusability

- A function can be used by many other programs

Compactness

– Each function is a self contained block of code that performs a particular task.

– Complexity reduces and debugging becomes easier.

– The length of a source program can be greatly reduced by using functions at appropriate places

- *The general form of a function:*

function_name(parameter list)

parameter declaration;

{

local declarations;

executable statements;

return(expression);

}

– Some of them are optional and are listed below:

a) parameter list and their declarations

b) return statement

A function can have any name followed by ()

```
main()

{

printf(“My first program using functions\n”);

msg(); //function call

}

msg() // function definition

{

printf(“This is my new function \n”);

}
```

Actual parameters and formal parameters

The variables declared in the function header are called formal
arguments
The variables or constants that are passed in the function call are called
actual arguments
The formal and actual arguments names can be same or different

There are two other categories that you should know about that are also referred to as "parameters". They are called "parameters" because they define information that is passed to a function.

- *Actual parameters* are parameters as they appear in function calls.
- *Formal parameters* are parameters as they appear in function declarations.

A parameter cannot be both a formal and an actual parameter, but both formal parameters and actual parameters can be either value parameters or variable parameters.

Calling User Defined Functions
A function is called by its name and passing the required arguments.
The constants can be passed as arguments to functions
result = add(5,10);
The variables can also be sent as arguments to functions
x=5; y=10; result = add(x, y);
Calling a function which does not return a value and without arguments
display(); //* display is the name of the function //*

Parameter passing and return values

```
_ _ int add(int,int);
int main(){
int x, y, res;
x=10; y=20;
res=add(x, y);
/* print the result using printf statement */
}
int add(int a, int b){
int c;
c=a+b;
return c;
}
```

No parameters and no return value

```
_ _ void add();
int main(){
add();
/* print the result using printf statement */
}
void add(){
int a,b,c;
a=10;b=20;
c=a+b;
/* print the result using printf statement */
}
```

CATEGORIES OF FUNCTIONS:

Depending on whether arguments are present or not whether a value is returned or not , a function may belong to one of the following categories:

- Functions with no arguments and no return values
- Functions with arguments and no return values
- Functions with arguments and one return value
- Functions with no arguments and one return value
- Function that returns multiple values

SAMPLE PROGRAMS USING FUNCTIONS:

1) Write a program to print “Hello World” in the main function and “Welcome To C” in another function.

```
#include <stdio.h>

#include <conio.h>

void message(void);

main()

{

        clrscr();

        printf("\nHELLO WORLD");

        message();

        getch();

}

void message()

{

        printf("\nWELCOME TO C");

}
```

2) Write a program to add three given numbers using function.

```
#include <stdio.h>

#include <conio.h>

sum(int,int,int);

void main()

{
        int a,b,c,d;

        clrscr();

        printf("\nACCEPT VALUE FOR a,b,c:\n");

        scanf("%d %d %d",&a,&b,&c);

        d=sum(a,b,c);

        printf("\nSUM OF %d,%d and %d IS %d",a,b,c,d);

        getch();

}

sum(int x,int y,int z)

{

        int temp;

        temp=x+y+z;

        return(temp);

}
```

3) Write a program to calculate the tax of n given number of employees. Use a separate function to calculate the tax. Tax is 20% of basic if basic is less than 9000 otherwise tax is 25% of basic.

```
#include <stdio.h>

#include <conio.h>

void cal(void);

void main()

{

        int i,n;

        clrscr();

        printf("\nENTER THE NUMBER OF THE EMPLOYEES: ");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
                cal();
        getch();
}
void cal()
{
        int basic;
        float tax;
        printf("\nENTER THE AMOUNT OF BASIC: ");
        scanf("%d",&basic);
        if(basic<9000)
                tax=basic*20/100;
        else
                tax=basic*25/100;
        printf("\nTHE AMOUNT OF TAX IS %0.2f\n",tax);
}
```

4) Write a program to compute the root of a number using function.

```
#include <stdio.h>
#include <conio.h>
unsigned long root(int);
void main()
{
        int x;
```

```
    unsigned long res;

    clrscr();

    printf("\nENTER A NUMBER: ");

    scanf("%d",&x);

    res=root(x);

    printf("\nROOT OF %d IS %lu",x,res);

    getch();

}

unsigned long root(int a)

{

    unsigned long b;

    b=a*a;

    return(b);

}
```

5) Write a program to evaluate a^b using function.

```
#include <stdio.h>

#include <conio.h>

unsigned long power(int,int);

void main()

{

    int num,pow;

    unsigned long res;

    clrscr();

    printf("\nENTER THE NUMBER: ");
```

```
        scanf("%d",&num);
        printf("\nENTER THE POWER: ");
        scanf("%d",&pow);
        res=power(num,pow);
        printf("\n%d TO THE POWER %d IS %lu",num,pow,res);
        getch();
}
unsigned long power(int a,int b)
{
        int i;
        unsigned long prod=1;
        for(i=1;i<=b;i++)
                prod=prod*a;
        return(prod);
}
```

6) Write a program to print the following series using function.

        9 25 57 121 249 505 1017 2041…

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int num=9,i;
        clrscr();
```

```
	printf("%d ",num);
	for(i=4;i<=10;i++)
	{
		num=num+pow(2,i);
		printf("%d ",num);
	}
	getch();
}
pow(int a,int b)
{
	int prod=1,j;
	for(j=1;j<=b;j++)
		prod=prod*a;
	return(prod);
}
```

7) Write a program to check a number is prime or not using function.

```
#include <stdio.h>
#include <conio.h>
void main()
{
	int num,res=0;
	clrscr();
	printf("\nENTER A NUMBER: ");
	scanf("%d",&num);
	res=prime(num);
	if(res==0)
		printf("\n%d IS A PRIME NUMBER",num);
	else
		printf("\n%d IS NOT A PRIME NUMBER",num);
	getch();
}
int prime(int n)
{
	int i;
	for(i=2;i<=n/2;i++)
	{
		if(n%i!=0)
			continue;
		else
			return 1;
	}
	return 0;
}
```

8) Write a program to find out the maximum number in an array using function.

```
#include <stdio.h>
#include <conio.h>
max(int [],int);
void main()
```

```
{
        int a[]={10,5,45,12,19};
        int n=5,m;
        clrscr();
        m=max(a,n);
        printf("\nMAXIMUM NUMBER IS %d",m);
        getch();
}
max(int x[],int k)
{
        int t,i;
        t=x[0];
        for(i=1;i<k;i++)
        {
                if(x[i]>t)
                        t=x[i];
        }
        return(t);
}
```

9) Write a program to evaluate 1/Factorial of 1 + 2/Factorial of 2 … + n/Factorial of n using function.

```
#include <stdio.h>
#include <conio.h>
fact(float);
void main()
```

```
{
        float i,sum=0.0,n;
        clrscr();
        printf("\nENTER A NUMBER: ");
        scanf("%f",&n);
        for(i=1;i<=n;i++)
                sum=sum+(i/fact(i));
        printf("\nTHE SUMMATION IS: %0.2f",sum);
        getch();
}
fact(float x)
{
        if(x==1)
                return(1);
        else
                return(x*fact(x-1));
}
```