6. FUNCTION

6.1. What is Function

A function is defined as a self contained block of statements that performs particular task/jobs. This is a logical unit composed of a number of statements grouped into a single unit. It is defined as a section of a program performing a specific task. The function main is always present in each program which is executed first and other function are optional.

The separate program segment is called functional that is called main program when ever required. For example to calculate prime numbers we should have to calculate each number generated form loop should have to test to be prime or not, in such case we have to repeat same logic of finding prime numbers up to defined times. Instead of such case we have to make general function which calculates factorial of a given number and this function is called function which calculates prime of a given numbers by passing them to the function to be tested. Therefore function is a special code block that has some functionality . it is called inside main methods when it required.

6.1.1. Advantage of Functions:

Manageability: Using function for specific job, it is easier to write programs and keep track of what they are doing. It makes programs significantly easier to understand and maintain by breaking them into easily manageable sub units. The code for each job or activity is placed in individual functions such that testing and debugging because easy and efficient. Thus use of functions in a program helps to manage the code.

Code Re-usability: A single function can be used multiple times by a single program form different places or different programs. It avoid re writing same code again that recuce large efforts and reuse code of computer designer. Thus it helps to reuse code once written and tested. The C standard library is an example of functions being reused.

Avoid Redundant: While using function same function is called whenever needed inside main program. Thus particular job or activity is to be accessed repeatedly form several different place within or outside the program. The repeated activity can be placed within a single function, which can then be accessed whenever it is needed. If the function is not used in such situation the code must have to redesign for same activity is to be rewritten the same instruction again and again.

Logical Clarity: When single program is decomposed into various well-defined functions, the main program consists of a series of function calls rather than rewrite lines of code so that size of main program seems smaller and becomes logical clarity understandable.

Easy to Divide the Work Among Different Programmers: Different programmers working at large project can divide the working by writing different functions easily.

6.2. User Defined Function and Library Functions

Library Functions (Built in Functions): There are some functions which are already written compiled and placed in C library. They are not required to be re-written by a developer. The function name its return type and their arguments number and types have been already defined. We can use these functions whenever we are required them. For example printf (), scanf (), sqrt(), getch() are library functions generally used.

User Defined Function: These are functions which are defined by user at time of writing a program. The user has choice to choose its name, return type, arguments and their types. In case of above example the function factorial () is user defined function. The job of each user-defined function is as defined by user. A complex program can be divided into a number of user defined functions.

About main () Function: The function main is a user defined function except that name of function is defined or fixed by the language. The return type arguments and body of function are defined by programmer as required. This function is executed first, when the program starts execution.

Function Definition: The collection of program statement that describes specific task to be done by function is called a function definition. It consists of function header which defines functions name its return type and arguments list and body of function which are the block of code enclosed in parenthesis.

Syntax:

```
    return type function name ( data type variable name, data type variable  name)
    {
    ….;
    ….;
    }
```

The first line of function definition is known as function declaration or header. This is followed by function which is composed of statement that makes up function, delimited by braces.

```
float areaofcircle(float radius){   return(3.14*radius*radius);}
void display(int a){ printf("%d",a);}
```

The return type is optional the default is int type. The arguments variable 1, variable 2, variable 3 presents in function definition are called arguments or parameters because they represent name of data items that are transferred into function from calling function/program. There are local variable for function.

Function Declaration/Prototype: The function declaration or prototype is model or blueprint of function. If function are used before they are defined, then function declaration or prototype necessary which provides following information to computer.

- Name of function and type of value returned by function
- The number and type of arguments
- When a user defined function is defined before used

Syntax:

return type function name (arguments, arguments);

Here return type specifies data types of value returned by function. A function can return value of any data type. If there is not return value, the keyword void is used. The function deceleration and in header function must use the same function name, numbers of arguments and its type and returned types.

Note: It is important to note that the function prototype has semicolon at the end and to specify the same name of formal arguments is not mandatory but type is mandatory.

Return Statement : It is statement that is executive just before function completes its job and control is transferred back to the calling function. The job of return statement is to hand over some value given form where the call was called. The return statement serves mainly two purposes. They are:

- It immediately transfers control back to the calling program after executed after the return statement i.e. no statement within function body is executed after return statement.
- It returns the value to calling function.

Syntax:

return (expression);

When expression must evaluated to value of type specified in function header from the return value. The expression can be any desired expression as long as it ends up with value of required type. The value of expression is returned is omitted, return statement simply cause control to revert back to calling portion of return statement each containing a different expression, but a function can return only one value to calling portion of the program via return statement.

Function may or may not return any value. If a function doesn't return a value type in the function definition and deceleration is specified as void. Otherwise return type is specified as a valid data type. A return statement at end is optional for function without return value.

Accessing Function: A function can be called or accessed by specifying its name, followed by a list of arguments enclosed in parentheses must separate by comma. For example, function add () with two arguments is called by add (int a, int b) to add two numbers. If function call does not required any argument it has empty pair of parenthesis must followed the name of function. The arguments appearing in function call are referred as actual argument. In function call there will be one argument for each formal argument. The value of each argument is transferred into function and assignment to corresponding formal argument. If functions a value returned value can be assigned to a variable of type same as return type of function. When a function is called the program controls is passed to function. Once the function completes its task, program control is passed back to the calling function. The general form of function call statement is:

- If function has parenthesis but it does not return value
- If function has no arguments and it does not return value
- If function has parameters and it returns value
- If function has do not parameters but returns value

The following points to be taken into consideration wish function calls statement are used:

- The function name type & number of variable listed in function statements must that of function deceleration statement and the header of function definition.
- The names of the variables in the function deceleration function call statement and that in the header of definition may be different.
- By default arguments are always passed by value in C function call. This means that local copies of the values of arguments are passed function.
- Arguments presents in the form of expression are evaluated and converted to the type of formal parameters at the beginning body of the function.
- A variable may be assigned value returned by function after it is executed using function call statement, provided return type where the function is not void type.

Function Parameters: Function parameters are means for communication between calling and called functions. They can be classified into formal parameters and actual parameters. The formal parameters are the parameters given in the function declaration and function definition, the actual arguments or permanents are specified in the function are called. In above example a and b are actual arguments which are used in calling unction while parameters of x and y are formal arguments which are defined in function definition. The name arguments or parameters which must be same in function definition. The name of formal and actual arguments need not be same but data type and numbers of them must be matched.

6.2.1 Category of Function According to Return Value and Arguments

According to the arguments and return type present in function definition and design can categorized function in three type:

- Function with no arguments and no return value
- Function with arguments and no return type
- The function with arguments and return values

Function with on Arguments and No Return Values: When a function has no arguments it does not receive any data from calling function similarly it does not return a value. The function do not receive any data from the calling function. Thus in such function there is not data transfer between the calling function and called function.

This type of function is defined as: **void function name () { }**

They keyword void means the function does not return any values there is not arguments within the parenthesis which implies function has no argument and it does not receive any data for the called function.

WAP to illustrate function with no arguments and no return value

```
#include<stdio.h>
#include<conio.h>
void addition()
{
int a,b,sum;
printf("Enter two numbers"); scanf("%d%d",&a,&b);
sum=a+b;
printf("the sum is%d",sum);
}
void main()
{
clrscr();
addition();
getch();
}
```

Function with Arguments but has Arguments but No Return Value: The function has argument and reveries the data from calling function. The function completes task and does not return any values to calling function.

Such type of functions are defined as: **void function (arguments) { }**

WAP to illustrate the functions with arguments but no return type

```
#include<stdio.h>
#include<conio.h>
void addition(int a,int b)
{
int sum;
sum=a+b;
printf("the sum is%d",sum);
}
void main()
{
int a,b;
clrscr();
printf("Enter two numbers");
scanf("%d%d",&a,&b);
addition(a,b);
getch();
}
```

Function with Arguments and Return Value: The function of this category has arguments and reveries data from calling function. After completing its task, it returns results to calling function through return statement. Thus there is data transfer between called function and calling function using return values and arguments.

These types of functions are defined as: **return_types function_name (arguments, ..., ...) { body}**

WAP to illustrate the function with arguments and return values.

```
#include<stdio.h>
#include<conio.h>
int addition(int a,int b)
{
int sum;
sum=a+b;
return sum;
}
void main()
{
int a,b;
clrscr();
printf("Enter two numbers"); scanf("%d%d",&a,&b);
printf("the sum is%d",addition(a,b));
getch();
}
```

6.2.2. Different Type of Function Calls

The arguments in function can be passed in two ways:

- pass arguments by value
- pass arguments by address or references for pointer

Function Call by Value (Pass Arguments by Value)

When values of actual arguments are passed to function as argument it is known as function call by value. In this call, value of each actual argument is copied into corresponding formal arguments of the function definition. The content of arguments int calling functions are not altered even if they are changed into called function.

WAP to illustrate function call by value

```
#include<stdio.h>
#include<conio.h>
void swap(int,int);
void main()
{
int a,b;
clrscr();
a=90;b=89;
printf("\nbefroe swap a and b are %d  and %d",a,b);
swap(a,b);
printf("\nafter swap are %d and %d",a,b);
getch();
}
void swap(int x,int y)
{
int temp;
temp=x;
x=y;
y=temp;
printf ("\nthe value within function %d and %d",x,y);
}
```

In this program the value a and b are copy to void swap(int x , int y) then calculates inside function only, when again original value s are not changed in the main program so this process is similar to pass argument by value.

Function Call by Reference (Pass Arguments by Address)

In this type of function call, address of variable or arguments is passed to the function as arguments instead of actual value of variable.

Pointer: A pointer is a variable that store a memory address of a variable. Pointer can be any name that is legal for other variable and it is declared in same fashion like other variable but it always proceeded by*(asterisk) operator. Thus pointer variables are defined as: int *s; float *b; char*c;

Here int *a, int *b, int *c is pointer variables which stores address of another integer, float and character variables.

Thus the following are valid:

- a=&p;
- b=&y;
- c=&y;

WAP to swap the values of two variables using call by reference

```
#include<stdio.h>
#include<conio.h>
void swap(int*,int*);
void main()
{
int a,b;
clrscr();
a=99;
b=9;
printf("\n before swap a and b are %d  and %d",a,b);
swap(&a,&b);
printf("\n after swap are %d and %d",a,b);
getch();
}
void swap(int *x,int *y)
{
int temp;
temp=*x;
*x=*y;
*y=temp;
}
```

6.3. Recursive Functions

Recursion is a technique for defining a problem in terms of one or more smaller sub unit of same problem. The solution of problem is built on result from the simpler versions. The recursive function is one that calls itself directly or indirectly to solve a smaller version of its task until a final call which does not require a self call. Thus function is called recursive function if it calls to itself. Recursion is a process by which a function calls itself repeatedly until some specified condition will be satisfied. Thus process is used for repetitive calculation in which each action is stated in term of previous result. Many iterative or repetitive problems can be written in this form.

- To solve problem using recursion methods two conditions must be satisfied therefore
- Problems could be written or defined in term of its previous result.
- Problems statement must include a stopping condition. We must have an if statement somewhere to force the function to return without the recursive call being executed otherwise the function will never return.

WAP to find the factorial of number using recursive method

```
#include<stdio.h>
#include<conio.h>
int factorial(int n)
{
        if(n==1)
                return(1);
        else
                return(n*factorial(n-1));
}
void main()
{
        int num;  clrscr();
        printf("Enter a number");  scanf("%d",&num);
        printf("the factorial is%d", factorial (num));
        getch();
}
```

Recursive	Iteration
A function is called for definition of same function to do repeated task	Loop is used to do repetitive action.
Recursion is the top down approach to problems solving it divides the problem int pieces.	Iteration is like a bottom up approach; it begins with what is known and form this if constructs the solution step by step procedure.
In recursion a function calls to itself until some condition will be satisfied.	Iteration a function does not call to itself
Problem could be written or defined in term of its previous result to solve a problem.	It is not necessary to define a problem in terms of its previous result to solve in iteration
All problems cannot be solved using recursion	All problems cannot be solved using iteration

6.4. Concept of Local, Global, Static & Register Variables

6.4.1. Local /Automatic Variables

The automatic variables are always declared within a function/block. They are local to the particular function/block in which they are declared. As local variables are defined within body of function/block, other functions cannot access the variables. The compiler shows errors in case other function   cannot access the variables. The local variables are created when function is called and destroyed automatically when the function exit. The keyword auto may be used for storage class specification while deceleration of variable. A variable declared inside a function without storage class specification auto is by default, an automatic local variable.

Default initial value of local variable is unpredictable value which is often called garbage value. The scope of it is local to the block in which variable is defined. Again its life is till the control remains within in which the variable is declared. The system supply the garbage value to it.

Initial Value: The garbage value assigned to a variable if no value is assigned to local variable.

Scope: Scope of variable can be defined as region open which variables are visible.

Lifetime: The period of variable availability function on time duration which memory is associated with as variable. Its life time as available only inside function block only.

WAP to illustrate local variable.

```
#include<stdio.h>
#include<conio.h>
int fact(int n)
{
int f=1;  int i;
for(i=1;i<=n;i++)
        f*=i;
return (f);
}
void main()
{
int num=5;
clrscr();
printf("the factorial of 5 is%d",fact(num));
getch();
}
```

6.4.2. Global / External Variable

Variable that are both are available and active thorough entire program are known as global variable. They are also known external variable. The external variable or global variable are declared outside the block of functions. Unlike local variables, global variables can be accessed by any functions in the program. The default initial variables of variables are zero. The scope of variable is within whole program. The life times are as long as the program's execution.

Initial Value: ZERO

Scope: Throughout the program.

Lifetime: As long as program's execution.

WAP to illustrate global variables

```
#include<stdio.h>
#include<conio.h>
void fun();
int a=10;
void main()
{
printf("\t%d",a);  fun();
printf("\t%d",a);
getch();
}
void fun()
{
a=20;
printf("\t%d",a++);
}
```

6.4.3. Static Variable

This is another class of local variable. A static variable can only be accessed from function in which it was declared, like a local variable. The static variable is not destroyed on exit from function termination instead its value is preserved and becomes available again when the function is next called. Static variables are declared as local variables, but declaration is preceded by word static. Static variable can be initialized as normal; the initialization is done once only when program starts. The default initial value for this type of variables is zero (if user doesn't initialize it). The scope is similar local variable is only available to block in which variable is defined. But life time is global

Initial Value: ZERO.

Scope: Similar to Local Variables.

Lifetime: Similar to Global Variables.

WAP to illustrate static variable.

```
#include<stdio.h>
#include<conio.h>
void staticc(void);
void main()
{
        staticc();
        staticc();
        staticc();
        staticc();
        getch();
}
void staticc(void)
{
        static int i=1;
        printf("\n The value is %d",i);
        i++;
}
Output:
The value is 1
The value is 2
The value is 3
The value is 4
```

```
#include<stdio.h>
#include<conio.h>
void staticc();
void main()
{
        staticc();
        staticc();
        staticc();
        getch();
}
void staticc()
{
        int i=1;
        printf("\nthe value of %d",i); i++;
}

Output
The value of 1
The value of 1
The value of 1
The value of 1
```

6.4.4. Register Variable

Register variables are special type of automatic variables. Automatic variables are allocated storage in memory of computer however for most computers accessing data in memory is considerably slower than processing in the CPU. Those computers often having small amount of storage within CPU itself. Where data can be stored and accessed quickly. Theses storage calls are called register. Normally the computer determines what is to be stored in register of CPU at what times .However C language provides the storage class register so that the programmer can suggest to the compiler that particular automatic variables should be allocated to CPU register. Thus register variable provide a certain control over efficient of program execution is faster accessed. Variables which are used repeatedly and have good access times are critical may be declared to register.

Register variables behaves way just like automatic variables. The storage block and storage is freed when block is exited. The scope of register variable is local to block where they are declared. Rule for initialization for register variables are same as for automatic variables.

6.4.5. Difference between Local, Global and Static variables

Local	**Global**	**Static**
The variables are declared within function /block. the scope is only within the function in which they are defined	Floral variables are defined outside the function so that their scope is through the program	Static variables are special case of local variables thus static variables are also defined inside the functions or block
The initial value is unpredictable or garbage value	The initial value is zero	The initial value is zero
The lifetime is till the control remains within the block or function in which the variable is defined. The variable is destroyed when function returns to the calling functions or block ends	The lifetime is till programs execution does not come to an end variables are created when n program starts and destroyed when program ends	The value persists between different function calls. Thus lifetime is same as global variables
The keyword “auto” is used	The keywords “extern” is used	The keyword “static” is used.

6.5. Passing Pointer to a Function

A pointer can be passed to a function as an argument. Passing a pointer means passing address of variable instead of value to a function. As address is passed in this case, this mechanism is also known as call by address or call by reference. When pointer is passed to a function the formal argument of the function must be compatible with the passing pointer i.e. if integer is being passed, the formal argument in function must be pointer of type. As address of variable is passed in mechanism, if value in passed to address is changed within function definition, the value of actual variable also changed.

WAP to illustrate the use of passing pointer to a function

```
#include<stdio.h>
#include<conio.h>
void add(int *m)
{
        *m=*m+10;
}
void main()
{
        int marks;
        clrscr();
        printf("Enter actual marks\t");
        scanf("%d",&marks);
        add(&marks);
        printf("\n the grace marks is\t%d",marks);
        getch();
}
```

6.6. Passing Arrays to Function

Like ordinary variables it is possible to pass value of an array elements i.e. entire array argument to a function. To pass entire array to a function array name must appear by itself, without brackets or subscript, an actual argument in function call statement. The corresponding formal argument in function definition is written in same manner. When declaring a one dimensional array as a formal argument, the array name is written with a pair of empty square brackets. The size of array is not specified within pair of empty square brackets. The size of array is not specified within the formal argument deceleration.

Syntax:

```
function_name (array name); //for function call.
return_type function_name (data_type array_name) //for function prototype;
return_type function_name (data_type* pointer_variable);
```

When an array is passed to function the value of array elements are not passed to the function. Rather the array name is interpreted as the address of element. This address is assigned to the corresponding formal argument when the function is called. The formal argument therefore becomes a pointer to first array element. Thus array is passed using call by reference.