CHAPTER
∞ 8 ∞
(Your Brain On Functions)

Introduction-

Function is a small program which take some input and give us some output. Function allow a large program to be broken down into a number of smaller self contained components, each of which has a definite purpose. It avoids rewriting the code over & over. Breaking down of logic into separate functions make the entire process of writing & debugging easier.

Or another hand you can say Function is a self contained block of statements that is used to perform some task. A function is assigned some work once and can be called upon for the task any number of times. Every C program uses some functions, the commonly used functions are printf, scanf, main, etc.

Functions can be library functions or user defined functions. Library functions are those functions which come along with the compiler and are present in the disk. The user defined functions are those which the programmer makes by himself to make his program easier to debug, trace.

Drill Note-

printf (), scanf(), exit(), pow() are library functions. Every library function

has a header file. ***There are a total of 15 header files in C***. main() is a user defined function.

Every program must have a main function. This main function is used to mark the beginning of the execution. It is possible to code any program using only the main program but this leads to many problems. It becomes too large and complex thus difficult to trace, debug and test.

But if the same program is broken into small modulus coded independently and then combined into a single unit then these problems can be solved easily. These modulus are called as functions. Thus a function can be defined as a small program which takes some input and gives us some output.

Drill Note- I am not going to type header files again and again, so when writing programs please don't forget to start your programs with header files.

Function to calculate the sum of two numbers:

```
int sum(int, int);              /* Function Prototype  or Declaration*/
main()
{
        int a,b,ans;
        printf("Enter two numbers: ");
        scanf("%d %d",&a,&b);
        ans= sum(a,b);              /*Function Call*/
        printf(" sum is %d", ans);
        getch();
}
int sum( int x, int y)              /*Function Definition or Process*/
{
        int z;
        z = x + y;
        return(z);
}
```

Now let us see some of the features of this program:

1. **The first statement is the declaration of the function which tells the compiler the name of the function and the data type of the arguments**

passed.

2. **The declaration is also called as prototype.**

3. **The declaration of a function is not necessary if the output type is an integer value. In some C compilers declaration is not required for all the function.**

4. **The function call is the way of using the function. A function is declared once, defined once but can be used a number of times in the same program.**

5. **When the compiler encounters the function call, the control of the program is transferred to the function definition the function is then executed line by line and a value is returned at the end.**

6. **At the end of the main program is the definition of the function, which can also be called as process.**

7. **The function definition does not terminate with a semicolon.**

8. **A function may or may not return any value. Thus the presence of the return statement is not necessary. When the return statement is encountered, the control is immediately passed back to the calling function.**

9. **While it is possible to pass any number of arguments to a function, the called statement returns only one value at a call. The return statement can be used as:**

```
return;
or
return(value);
```

The first return without any value, it acts much as the closing of the braces of the function definition.

10. **A function may have more than one return statement. It can be used as :**

```
if (a!=0)
        return(a);
else
        return(1);
```

11. **All functions by default return int. But if the function has to return a particular type of data the type specifiers can be used along with the function name.**

```
long int fact(n)
```

12. **If function main() calls a function sum() then main() is the calling function and sum() is called function.**

No arguments and no return values:

Some functions do not receive any value from the calling function. Thus the function prototype will be as:

prn( ) i.e. no arguments will be passed. This can also be achieved as prn(void) And similarly the calling function does not get any value from the function. This is made possible by using the keyword void before the function name. to illustrate this point let us consider the following program:

```
void prn( );            /*declaration can also be made as void prn(void);*/
main( )
{
        prn( );
        prn( );
        prn( );
}
void prn( )
{
        printf("Hello");
}
```

Argument Passing Mechanism -

- **(i) Call by value -**

When arguments are passed by value then the copy of the actual parameters is transferred from calling function to the called function definition in formal parameters.

Now any changes made in the formal parameters in called function definition will not be reflected in actual parameters of calling function. Like in the above function to calculate the sum of two numbers the calling statement was written as:

ans = sum(a, b);

Here the values of variables a, b are passed from the main function to the calling function's definition.

In the definition variables x, y accept the values of a, b respectively. Here, the variables a, b are called the actual arguments while x, y will be called the formal arguments.

The scope of the actual and formal arguments is different so any change made in the formal arguments will not be seen in the actual arguments.

e.g:

```
void swap(int, int);
main()
{
        int a,b;
        printf("Enter 2 numbers");
        scanf("%d%d",&a,&b);
        swap(a,b);          /*In this call statement a,b are the actual parameters*/


printf("%d\t%d"a,b);
        }
void swap(int x,int y)   /*In this function definition x & y are the formal parameters */
        {
                int t;
                t=x; x=y; y=t;
```

```
}
```

Drill Note –

In the above e.g.: changes made in x, y will not be reflected in a,b.

(ii) Call by reference -

When arguments are passed by reference then the address of the actual parameters is transferred from calling function to the called function definition in formal parameters.

Now any changes made in the formal parameters in called function definition will be reflected in actual parameters of calling function.

Sometimes it is not possible to pass the values of the variables, for example while using an array it will not be possible to pass all the values of the array using call by value.

So, another type of function calling mechanism is used call by reference where the *address of the variable is passed*.

Here the definition would work by reaching the particular addresses. This method is generally used for the *array and pointers*.

e.g.:

```
main()
{
        int a,b;
        printf("Enter 2 numbers");
        scanf("%d%d",&a,&b);
        swap(&a, &b);                    /*In this call statement address
                                    of a, b gets transferred*/
        printf("%d\t%d"a,b);
}
                    void swap(int *p1, int *p2)        /*In this function
                    definition p1 & p2 are the pointers which receive the
                    address of a, b*/
{
```

```
        int t;
        t=*p1; *p1=*p2; *p2=t;
}
```

Drill Note –
In the above e.g.: changes made in p1, p2 will be reflected in a, b automatically.

Drill Note –
Arrays are also passed by reference. When we pass the name of the array then only the base address is transferred in the function definition.

Type of Functions -

1. **Library Functions - Functions defined previously in the library** are called as library functions.

e.g.

```
#include<math.h>
main()
{
        int n, p, ans;
        printf("Enter number and its power");
        scanf("%d%d",&n,&p);
        ans = pow(n,p);
        printf("%d",ans);
        getch();
}
```

Common Library Functions:

stdio.h functions-

fclose( )	Closes a stream
fcloseall()	Closes all open streams
feof()	Tests if end-of-file has been reached on a stream
fflush()	Flushes a stream

fgetc()	Gets a character from a stream
fgetpos()	Gets the current file pointer position
fsetpos()	Positions the file pointer of a stream
fgetchar()	Gets a character from stdin
fgets()	Gets a string from a stream
fopen()	Opens a stream
fprintf()	Sends formatted output to stream
fputc()	Outputs a character to a stream
fputs()	Outputs a string to a stream
fread()	Reads data from a stream
fscanf()	Scans and formats input from a stream.
fseek()	Sets the file pointer to a particular position.
ftell()	Returns the current position of the file pointer.
fwrite()	Writes to a stream.
getc()	gets one character.
getchar()	gets a character from stdin.
gets()	Get a string from stdin.
getw()	gets an integer from stream.
printf()	Sends the formatted output to stdin.
putc()	Outputs a character to stdout.
putchar()	Outputs a character on stdout.
puts()	Outputs string and appends a newline character.
putw()	Outputs an integer on a stream
remove()	Removes a file
rename()	Renames a file
Rewind()	Brings the file pointer to stream's beginning
scanf().	Scans and formats input from stdin.

conio.h

clrscr()	Clears text mode window
getch()	gets a character from console but does not echo to the screen
getche()	gets a character from console, and echoes to the screen
putch()	Oututs character to the text window on the screen

cgets()	Reads string from console
getchar()	Inputs a character from stdin.

stdlib.h

itoa()	converts an integer to a string.
atoi()	Converts string of digits to integer.
Random()	Returns a random number between 0 and number – 1
randomize()	initializes random number generator.
exit()	Terminates the program.
min()	Returns the smallest of two numbers.
max()	Returns the largest of two numbers.
ltoa()	converts a long to a string
ultoa()	converts an unsigned long to a string
atof()	converts a string to a floating point
_atold()	converts a string to a long double

math.h

abs()	gets the absolute value of an integer
acos()	Calculates the inverse of cos Accepts the angle value in radians
asin()	Calculates the inverse of sin Accepts the angle value in radians
atan()	Calculates the inverse of tan Accepts the angle value in radians
ceil()	Returns the largest integer in given list.
cos()	Calculates the cosine Accepts the angle value in radians
cosh()	Accepts the angle value in radians
exp()	Calculates the exponent
floor()	Returns the smallest integer in given list.
log()	Calculates the natural logarithm
log10()	Calculates the log of base 10
pow()	Calculates the power of a number
sin()	Calculates the sine value of an angle. Accepts the angle value in radians
sqrt()	Calculates the square root of a number

tan()	Calculates the tangent value of an angle. Accepts the angle value in radians
tanh()	Calculates the tangent hyperbolic value.

string.h	string.h
strcat()	Function to concatenate(merge) strings.
strcmp()	Function to compare two strings.
strcpy()	Function to copy a string to another string
stricmp()	Function to compare two strings ignoring their case.
strlen()	Function to calculate the length of the string
strlwr()	Converts the given string to lowercase
strrev()	Function to reverse the given string.
strupr()	Converts the given string to uppercase
strdup	Duplicates a string.
strnicmp()	Compares the first n characters of one string to another without being case sensitive.
strncat()	Adds the first n characters at the end of second string.
strncpy()	Copies the first n characters of a string into another.
strchr()	Finds the first occurrence of the character.
strrchr()	Finds the last occurrence of the character.
strstr()	Finds the first occurrence of string in another string.
strset()	Sets all the characters of the string to a given character.
strnset()	Sets first n characters of the string to a given character.

2. User Defined Functions -

Functions defined by us are known as User Defined Functions. main() function is also user defined function because the definition of main() is defined by us.

e.g.

```
int power(int, int);
main()
{
        int n, p, ans;
        printf("Enter number and its power");
        scanf("%d%d",&n,&p);
```

```
        ans = power(n,p);
        printf("%d",ans);
        getch();
}
int power(int n, int p)
{
        in ans=1, i;
        for(i=1;i<=p;i++)
        {
                ans = ans * n;
        }
        return(ans);
}
```

Function whose argument is a two dim array:-

```
void mat_sum(int m1[ ][10], int r1, int c1, int m2[][10], int r2, int c2, int m3[][10])

{
        int i,j;
        if(r1 != c1 && r2 != c2)
        {
                printf("Can't sum");
                exit();
        }
        for(i=0;i<r1;i++)
                for(j=0;j<c1;j++)
                        m3[i][j] = m1[i][j] + m2[i][j];

    }
```

The above function can be called from main() as mat_sum(m1,r1,c1,m2,r2,c2,m3);

Recursion -

A function is called recursive if a statement within the body of a function calls the same function. Sometimes called as 'circular definition', recursion is a function calling itself in the definition. A recursive function should have two parts recursive statement & a termination condition.

Suppose we want to calculate the factorial of an integer. As we know, the factorial of a number is the product of all the integers between 1 and that number. Factorial of 4 can be expressed as 4! = 4 * 3! Where ! stands for factorial. Thus the factorial of a number can be expressed in the form of itself. Hence this can be programmed using recursion.

e.g.:

```
int fact (int);                          /*function definition */
main()
{
int n,ans;
printf("Enter a number: ");
scanf("%d",&n);
ans = fact(n);                           /*function call */
printf("factorial = %d",ans);
getch();
}
int fact(int n)
{
        if(n= =0)                        /*terminating condition*/
        return (1);
        return (n*fact(n-1));            /*recursive statement*/
}
```

Now let us evaluate this program:

Assuming the value of n is 3 when the control of the program is passed from the main() function the function fact. Since n is not equal to 0 so the condition is false and the recursive statement is executed.

3*fact(2)

now fact(2) is the calling function and thus the control of the program again reaches the beginning of the definition. Still the terminating condition is false so the recursive statement is executed.

2*fact(1)

Again fact(1) is the calling function and thus the control of the program again reaches the beginning of the definition. Still the terminating condition is false so the

recursive statement is executed.

1 * fact(0)

Now the condition is true so the answer to the calling function(fact(0)) will be 1 and so on. Thus the sequence of acts will be:

fact(3)=3 * fact(2)
fact(2)=2 * fact(1)
fact(1)=1 * fact(0)

When we use a recursive program a stack is used to organize the data. Stack is a Last In First Out (LIFO) data structure. This means that the last item to be stored (push operation) in the stack will be the first one to come (pop operation) out.

In the above program when the fact(2) is called the value 3 will be stored in the stack. Similarly when fact(1) is called the value 2 will be stored at the top of 3 on the stack.

Now when the fact(0) returns 1. it will be multiplied to the first value in the stack i.e.

1. **This result will be multiplied to the second waiting value of the stack i.e. 2 and so on.**

When a function in its definition calls another function it is called *chaining*. Recursion is a special type of chaining where a function calls itself.

e.g.:

```
main()
{
        printf("Harry\n");
        main();
}
```

when executed the program will give the output as :

Harry
Harry
Harry

_

The execution of any recursive function can continue indefinitely so to bring the execution to the end a terminating condition is applied.

Use of recursive functions is to solve the problem where the solution is expressed in terms of successively applying the same solution to subsets of problems.

But there are also some disadvantages of the recursive functions:

1. **These functions are more time consuming, so the execution speed of the program is slow.**

2. **More memory space is occupied due to the formation of stack to keep the waiting values.**

Fill in the blanks -

1. Function returns the control to the calling function on the final __or __.
2. To not return anything to the calling function, _____can be used.
3. List of the parameters passed to a function are separated using _____.
4. The default return type of a function in C is _____.
5. In C all the arguments passed in a function are by _____.
6. Call–by–reference can be achived through _____.
7. main is a _____ function.
8. _____ are mandatory in a function.
9. _____ is the ability of a function to call by

itself.

10. The point at which a program stops recursion is called _____

11. Recursion uses _____ memory than iterative method.

Answers -

1. } or return statement.
2. void
3. ,
4. int
5. call-by-value
6. pointers
7. user defined
8. arguments
9. recursion
10. terminating condition
11. more

Solved programs -

1. WAP to calculate the power p of a number n by user defined function.

```
int pow(int,int);/*declaration*/
main()
{
        int n,i,ans,p;
        printf("Enter two numbers: ");
        scanf("%d %d",&n,&p);
        ans=pow(n,p);/*call*/
        printf("%d to the power %d is %d",n,p,ans);
}
int pow(int n,int p)/*defination*/
{
        int i,ans=1;
        for(i=1;i<=p;i++)
                ans=ans*n;
        return(ans);
}
```

2. WAP to calculate the Greatest common divisor of two numbers using recursive function.

```
int gcd(int, int);
main()
{
```

```
        int n,m;
        int ans;
        printf("\n Enter two integer numbers");
        scanf("%d %d",&n,&m);
        ans = gcd(n,m);
        printf("GCD of %d and %d is %d",n,m,ans);
}
int gcd(int n, int m)
{
        if(n >= m && n%m == 0)
                return(m);
        else
                return gcd(m,n%m);
}
```

3. WAP to calculate the Greatest common divisor of two numbers using non-recursive function.

```
int gcd(int,int);
main()
{
        int n,m,ans;
        printf("Enter two integers");
        scanf("%d %d",&n,&m);
        ans = gcd(n,m);
        printf("GCD of %d and %d is %d", n,m,ans);
}
int gcd(int n, int m)
{
        int t;
        while(m!=0)
        {
                t = m;
                m = n % m;
                n = t;
        }
        return (n);
}
```

4. Write a program to search a sub-string using pointers.

```
int strsearch(char [ ],char [ ]);
main()
{
        char s1[50],s2[10];
        int ans=0;
        printf("Enter a string");          /*input of string */
        gets(s1);
        printf("Enter a string");          /*input of substring to search*/
        gets(s2);
        ans = strsearch(s1,s2);            /*function call*/
        if(ans == -1)                      /*using the returned value*/
                printf("String is not found");
        else
                printf("String is found at pos %d ",ans+1);
}
int strsearch(char s1[ ],char s2[ ])          /*definition*/
{
        int i,j;
        i=j=0;
        while(s1[i] != '\0')
        {
                if(s1[i] == s2[j])
                {
                        while(s1[i+j] == s2[j] && s2[j] != '\0')
                                j++;
                        if(s2[j] == '\0')
                                return (i);
                        j=0;
                }
                i++;
        }
        return (-1);
}
```

5. WAP to calculate the determinants of a matrix*/

```
#define MAXROW 4
#define MAXCOL 4
```

```
int det_mat(int [ ][MAXCOL],int,int);
main()
{
        int mat[MAXROW][MAXCOL]={0};
        int r,c,i,j,sum;
        printf("Enter dimension of matrix");
        scanf("%d%d",&r,&c);
        /*Input Matrix */
        for(i=0;i<r;i++)
                for(j=0; j<c;j++)
                {
                        printf("Enter element %d %d", i+1, j+1);
                        scanf("%d",&mat[i][j]);
                }
        /*function call*/
        sum = det_mat(mat,r,c);
        /*output*/
        printf("%d",sum);
}
int det_mat(int mat[ ][MAXCOL], int r, int c)
{
        int i,j,k,sign,sum,a;
        int mat2[MAXROW][MAXCOL]={0};
        sign = 1;
        sum = 0;
        if(c = = 1)
                return(mat[0][0]);

        for(i=0 ; i<c; i++,sign *= -1)
        {
                for(j=1; j<r; j++)
                {
                        for(k=0;  k<c; k++)
                        {
                                if(k == i)
                                        continue;
                                if(k>i)
                                        mat2[j-1][k-1]=mat[j][k];
                                else
                                        mat2[j-1][k]=mat[j][k];
```

```
                    }
                }

                    sum = sum + mat[0][i] * sign * det_mat(mat2,r-1,c-1);
            }
            return (sum);
        }
```

Fill in the blanks –

The return statement returns only _____.

An _____ can be assigned initial values by including appropriate expressions by transferring control to some other part of program.

An automatic variable does not retain _____ once the control is transferred out of its defining function.

An extern variable declaration must begain with storage class specifiers _____.

If the line int sp;
Occurs outside any function, it _____ the external variable sp.

if the line
static char buffer[max];
Appears in one file of a program, then the variable name will not conflict with the same name in _____ of the same program.

An external variable definition must not begin with storage class specifies _____.

What would be the output of the following program?

1.
```
main( )
{
printf("\n welcome in c:");
        display( );
}
```

5.
```
main( )
{
        int i = 40,c;
        c = check(i);
        printf("\n%d",c);
```

```
display( )
        {
        printf("History of C:");
                main( );
        }
```

Output: Both massage will get printed indefinitely.

Because when main( ) calls display control jumps to the display and because in the body of display main is again call then control again jumps to the main and this process will be repeat repetedly.

2. main( )

```
        {
                float area;
                int radius = 1;
                area = circle (radius);
                printf("\n%f",area);
        }
        circle( int r)
        {
                float a;
                a = 3.14*r*r;
                return(a);
        }
```

Output: 3.000000

Because in function definition there is no explicit return type and default return type is int so when we return *a* that is 3.14 will demote to 3. And when 3 is printed as float it will promote in floating point format and gets printed 3.000000.

3. main( )

```
        {
        printf("\n Go for Matrix:");
        }
        check(int ch)
        {
                if(ch>=40)
                        return(100);
                else
                        return(10*10);
        }
```

Output: 100

Because the value of i is passed to ch that is 40. and when condition is checked in function definition it becomes true so if block is executed and returns 100.

6. main( )

```
        {
                int i = 5, j = 2;
                hello (i, j);
                printf("\n%d %d",i,j);
        }

        hello(int i, int j)
        {
                i = i*i;
                j = j*j;
        }
```

Output: 5 2

The function here is call by value so any change in the formal argument will not effect the actual argument. So in main( ) function i and j are still same.

7. main( )

```
        {
                int x = 5, y = 2;
                hello(&x ,&y);
                printf("\n%d %d",x,y);
        }
        hello(int *i,int *j)
        {
```

```
        main();
    }
```

Output: The massage will print indefinitely.

Because main again call main( ) that will be recursive call without any terminating condition,so massage gets printed indefinitely.

4.
```
main( )
    {
        static int i = 0;
        i++;
        if ( i<=5)
            {
printf("%d\t",i);
                main(
);
            }
        else
            exit( );
        }
```

Output: 1 2 3 4 5

Here first i++ is executed and each time the value of i increment by 1. now for the first time it will be 1 and condition becomes true and gets printed the value of i that is 1 now again main function is called and we know i is declared as static so it's value will not again initialize and now this time when increment take place it becomes 2 and in this way until i will not be 5, i gets printed and call main function repeatedly but when i becomes 6 condition will be false and control jumps

```
        *i = *i * *i;
        *j=*j * *j;
    }
```

Output: 25 4

In the above program because function is call by reference and the address of x any y is passed to i and j and by dereference [*i = *i * *i] becomes [x = x*x] that gives 25 and similarly next statement gives 4. because here function is called by reference so it will change the actual argument and then x =25 and y = 4 gets printed through printf( ).

8.
```
main( )
    {
        int i = 5, j = 3;
        calc( &i , j);
        printf("\n %d %d",i,j);
    }
    calc( int *i, int j)
    {
        *i =*i * *i;
         j = j*j
    }
```

Output: 25 3

Here to argument is passed to the called function first the address of i and second the value of j so when we dereference i. it change the actual argument but the operation on j doesn't effect to actual argument so the value of j remains same and i gets change.

to the else block and because of exit( ) it stops the program.

Point out the errors, if any, in the following programs:

1.
```
main( )
{
        int a = 3,b = 4, c,k;
        c = summult(i,j);
        d = summult(i,j);
        printf("\n%d %d",c,d);
}
summult(int x, int y)
        {
                int a1, b1;
                a1= x+y;
                b1 = x*y;

return(a1,b1);
        }
```
Output: Error. More than one value can't be returned by a function.

2.
```
main( )
{
        int x;
        x = massage( );
}
message( )
        {
        printf("\n your computer can be effected by Viruses:");
                return;
        }
```
Output: No Error.

7.
```
main( )
{
        message( );
        message( );
}
messae( );
{
        printf("Whole world is
                waiting for you:");
}
```
Output: Error. Function definition must not be present immediately after the function definition.

8.
```
main( )
{
        matrix( )
                {
        printf("Welcome in the world
                of computer:");

}
}
```
Output: Error. One function can't be defined into the body of another function.

9.
```
main( )
{
```

Because here no explicit return type is prefixed in function definition so default return type is int and here return statement is also exist in function definition so it will return a garbage integer value. But if there will be void type in function definition then error is encounterend.

3.
```
main( )
{
        float a = 185.5;
        char ch = 'C';
        display(a,ch);
}

display(a,ch)
{
        printf("\n%f %c",a,ch);
}
```
Output: Error.
Formal argument don't have any data type in function definition.

4.
```
main( )
{
        int a = 35,b;
        b = check (a);
        printf("\n %d",b);
}
check( m)
{
        int m;
        if( m>40)
                return(1);
        else

return(0);
        Matrix(computer(
));
}
void computer( )
{
printf("Beginning with C:");
}
```
Output:

9.
```
main( )
{
        int i = 135, a =
135, k;
        k = demo( i , a);
        printf("\n %d",k);
}
demo(int j, int b)
int c;
{
        c = j + b;
        return (c);
}
```
Output: Error. The declaration int c should be inside the body of demo( ).

10.
```
main( )
{
        int p  = 23,f = 24;
        out(&p, &f);
        printf("\n%d %d",p,f);
}

out (int q, int g)
{
        q = q + q;
        g = g + g;
}
```
Output: Error. The variable q and g in function definition should be declared as integer pointer.

```
}
```

Output: Error. The variable m must be declared before the braces.The formal argument can be declared by two following ways:

(a) Check(int m)

```
{

}
```

(b)check( )

```
int m;
{

}
```

5.

```
main()
{
        display();
}
void display()
{
        printf("Matrix");
}
```

Output: Error of redeclaration because display() is called before it is defined. In such cases the compiler assumes that the function display() is declared as **int display ();**
that is, an undeclared function is assumed to return an int and accept an unspecified number of arguments. Then when we define the function the compiler finds that it is returning void hence the compiler reports the discrepancy.

6.

```
main()
{
   int a=1;
    while(a<=5)
```

11.

```
main( )
{
        int i = 35, *z;
        z = fun( &i);
        printf("\n%d",z);
}

fun( int *m)
{
        return(m + 2);
}
```

Output: Unpredictable Output.
Here in the above program the address of i(&i) suppose 2005 is passed to m so m contains the address of i and returns m+2 is equal to 2007. so this function return 2007 to the main( ) that will store in z and z gets printed through printf( ). Here address is assume by the compiler so we can't say what will be the output but there will print an int value.

13.

```
main()
{
        int a,b;
        a=sumdig(123);
        b=sumdig(123);
        printf("%d %d",a,b);
}
sumdig(int n)
      {
            static int s=0;
            int d;
            if(n!=0)
            {

d=n%10;
                              n=(n-
```

```
        {
            printf("%d",a);
            if( a>2)
                        goto
abc;
        }
    }
    fun()
    {
            abc:
        printf("Author Harry");
    }
```

Output: goto cannot take control to a different function.

```
main()
{
        int a=10;
        void f();
        a=f();
        printf("%d",a);
}
void f()
{
        printf("Hello");
}
```

Output: The function has been declared as void but still the program is trying to collect the value returned by f() in variable *a*.

```
d)/10;
                                    s=s+d;

sumdigit(n);
            }
            else
                                    return
(s);
        }
```

Output: 6 12

14.
```
f(int a, int b)
{
    int a;
    a=20;
    return a;
}
```

Output: Error The variable *a* is redeclared

15.
```
main()
{
        int b=10;
            b=f(20);
        printf("%d",b);
}
int f(int a)
{
            a>20?
return(10):return(20);
}
```

Output: return statement cannot be used as shown with the conditional operators. Instead the following can be used:

```
return(a>20?10:20);
```

State whether the following statements are True or False:

1. **The variables commonly used in C functions are available to all functions in**

a program.

2. **To return the control back to the calling function we must use the keyword return.The same variable names can be used in different functions without any conflict.**

3. **Every called function must contain a return statement.**

4. **A function may contain more than one return statement.**

5. **Each return statement in a function may return a different value.**

6. **A function can still be useful even if you don't pass any arguments to it and the function doesn't return any value back.**

7. **Same names can be used for different functions without any conflict.**

8. **A function may be called more than once from any other function.**

9. **It is necessary for a function to return some value.**

10. **A function can have several declarations but only one definition.**

11. **A function cannot be defined inside another function.**

12. **Will the following function work:**

```
main()
{
        f1(int a, int b);
{
        return(f2(20));
}
f2 (int a)
{
        return(a*a);
}
```

13. **In a function two return statements should never occur.**

14. **In a function two return statements should never occur successively.**

15. **In C all functions except main() can be called recursively.**

16. **Usually recursion works slower than loops.**

17. **Too many recursive calls may result in stack overflow.**

Answers:

1. False	2. False	3. True	4. False
5. True	6. True	7. True	8. False
9. False	10. True	11. True	12. True
13. False	14. True	15. False	16. True
17. True			

Answer the following:

1. **When we mention the prototype of a function we are defining or declaring the function?**

   Output: declaring

2. **There is a mistake in the following code, add a statement to remove it.**

```
main()
{
        int a;
        a=f(10,3,14);
        printf("%d",a);
}
f(int aa, float bb)
{
        return((float)aa+bb);
}
```

   Output:
   The declaration of the function *f* is missing add the following above main()

```
        float f(int,float);
```

3. **What are the two notations of defining functions commonly known as:**

```
int f( int a, float b)
{
        /*some code*/
}
int f(a,b)
int a;
float b;
{
        /*some code*/
}
```

Output:

The first one is known as ANSI notation and the second is known as Kernighan and Ritchie or simply K&R notation.

Lab Exercise –

1. Write a function which takes to integer as argument and return there sum. WAP to test this function.

2. Write a function which takes to integer as argument and return there average in float. WAP to test this function.

3 . WAP that uses a function that converts a lowercase character to its uppercase.

4. WAP that uses a function that calculates factorial of a given number.

5. WAP that uses a function power that calculates the power of a given number.

6. WAP that uses a function that finds the largest of three integer quantities.

7. WAP that receives any year from the keyboard and uses a function to determine whether the year is a leap year or not.

8. WAP that uses a function which receives a float and an int from main(),finds the product of these two and returns the product which is printed through main().

9. WAP that uses a function to calculate the sum of n odd integers.

10. WAP that uses a function to calculate the sum of n even integers starting from a given even integer.

11. WAP that uses a function to determine whether a given positive integer is a prime number or not.

12. WAP that uses a function to determine whether a given positive integer is a fibonacci number or not.

13. WAP that uses a function that finds the length of the largest monotonically increasing subsequence in a sequence of real numbers.

14. WAP that uses a function for finding the absolute value of the integer parameter passed to it (do not use any library function).

15. WAP that uses a function that takes two arguments: a character and an integer and prints the character given number of times. if however the integer is missing the function prints the character twice.

16. WAP that uses a function to sum n natural numbers starting from a given number.

17. WAP that uses a function that takes a character argument and prints it number of times equal to number of times that function has been called to the input.

18. WAP that uses a function which takes a real number as its argument and returns the sum of digits(complete including fraction parts) of this number.

19. WAP that uses a function that checks whether the given number is divisible by 11 or not by using the algorithm which states that a number is divisible by 11 if and only if the difference of the sums of digits at odd positions and even positions is either zero or divisible by 11.

20. WAP that uses a function isdigit which should return a non-zero if the given number is a digit and 0 if not.

21. WAP that uses a function isalpha which should return a non-zero if the

given number is a alphabet and 0 if not.

22. WAP that uses a function isalnum which should return a non-zero if the given number is a alpha numeric and 0 if not.

23. WAP that uses a function isupper which should return a non-zero if the given number is a uppercase character and 0 if not.

24. WAP that uses a function toupper which accepts a character argument and return its equivalent uppercase character.

25. WAP that uses a function that returns the gcd(greatest common divisor) of two integers.

26. WAP that invokes a function satis() to find whether four integers a, b, c, d sent to satis() satisfy the equation a^3+b^3+c^3=d^3 or not. The function satis() returns 0 if the above equations satisfied with the given four numbers otherwise it returns -1.

27. WAP that uses a function called carea() to calculate area of a circle. The function carea() receives radius of float type and returns area of double type. the function main() gets radius value from the user, calls carea(),and displays the result. the function carea() is local to main().

28. WAP that uses various functions to sum following series:

a) (1)+(1+2)+(1+2+3)+(1+2+3+4)+_____ upto n terms

b) (2^2)+(2^2+4^2)+(2^2+4^2+6^2)+(2^2+4^2+6^2+8^2)+_____ upto n terms

c) 1+1/3+1/5+1/7+1/9+_____ upto n terms

d) 1+1/x+1/(x^2)+1/(x^3)+1/(x^4)+_____ upto n terms

29. WAP that receives a positive number from the keyboard and uses a functions to obtain the prime factors of this number. for eg, prime factors of 24 are 2,2,2,and 3,whereas prime factors of 35 are 5 and 7.

30. WAP that uses a function that can compute sum of any geometric series.

31. WAP that uses a function that generate every third integer, beginning

with i=1 and continuing for all integers that are less than 100. Calculate sum of those integers that are evenly divisible by 5.

32. WAP that uses a function for calculating volume and surface area of a sphere given diameter of the sphere.

33. WAP to print the largest element of an array using a function.

34. WAP that uses a function that takes a double array name and an array size as arguments and that SWAP the first and last value in that array.

35. WAP that uses a function which will accept an array of integers as an argument. it should find and return the smallest element in the array after sorting it. the calling program requires the sorted array. The size of the array can be defined to be a global constant.

36. WAP that uses a function that receive an int array, its size and a character '+' or '-'.by default the character should be '+'.for the character '+',the function returns the sum of positive numbers stored in the array and for the character '-',the function returns the sum of negative numbers stored in the array.

37. WAP that reads a float array having 15 elements. the program uses a function reverse() to reverse this array. make suitable assumptions wherever required.

38. WAP that uses various functions to express the following algebraic formulas in a recursive form:

a) y=(x1+x2+x3+_____+xn)
b) y=1-x+(x^2)/2-(x^3)/6+(x^4)/24+_____+(-1)nx^n/n!
c) p=(f1*f2*f3*_____*ft)

39. WAP that uses a function that takes a decimal number as a parameter and returns its octal equivalent.

40. WAP that uses a function that takes a decimal number as a parameter and returns its hexadecimal equivalent.

41. WAP that uses a function that takes a six digit integer as a parameter. if the number is even, then adds up its digits else multiply the individual digits and

returns the result.

42. WAP that uses a function to send all -ve elements of an array to the end without altering the original sequence i.e. if array contains 5,-4,3,-2,6,-11,12,-8,9 then the return array will be 5,3,6,12,9,-4,-2,-11,-8

43. WAP that uses a function for the binary search algorithm without using recursion technique.

44. WAP that uses a function to print the length of a string.

45. WAP that uses a function to copy a string into another string.

46. WAP using following functions to reverse a string:
a) func1() to reverse string using another array.
b) func2() to reverse string without using another array.

47. WAP that uses a function to concatenate two strings.

48. WAP that uses a function to a string into uppercase.

49. WAP that uses a function to find the first occurrence of a string into another string.

50. WAP that uses a function to find the last occurrence of a string into another string.

51. WAP that uses a function to compares two strings and returns 0 if the strings are equal and -1 if the strings are unequal.

52. WAP that uses two functions, code() and decode(),that accepts a string for an argument. the code() function should modify the argument string adding 1 to all characters in it except the null terminator. the decode() function restores the coded string to its original form. this program should accept a string on the command line, print the string coded, and then print it decoded. if no string is specified on the command line, prompt for none.

53. WAP that uses a function that accept two strings as the arguments and compare them to find the length of the greatest common substring between the two.