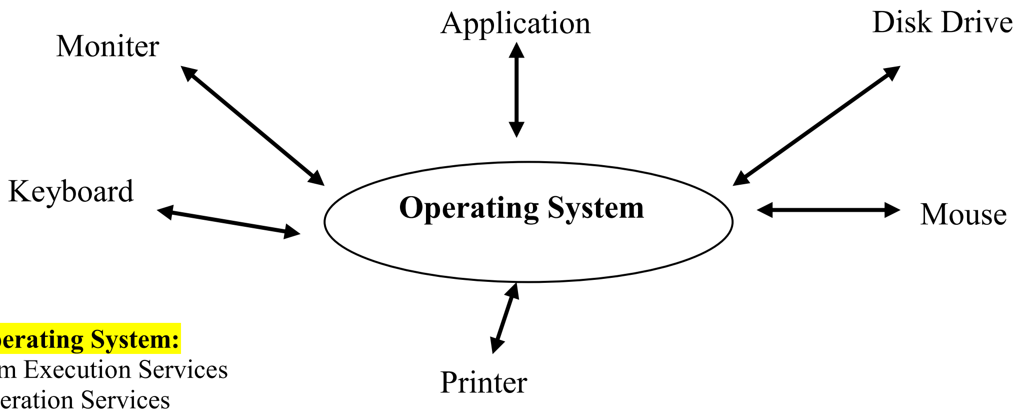
Computer System and Programming in C

Q. Explain Operating System and its functions

- OS is system software that helps the users to operate the computer and manage its resources.
- OS is a resource allocator that allocates the resources according to the CPU or I/O bound processes.
- OS acts as a scheduler and thus a process synchronizer.
- The primary purpose of the OS is to act as an interface between the computer and the user.
- Operating system is a continuously running program which provides an environment for the user to execute his programs in an efficient manner.
- Operating system is a all time running kernel program and all else being application programs.
- It provides a set of commands through which user can type, copy, print etc. Without OS, a computer cannot be used.

Exp: OS/2, Windows 98, Windows 2000, Windows XP, Vista.



Function of Operating System:

- Program Execution Services
- I/O Operation Services
- File System Services
- Communication Services
- Error detection Services
- Accounting Services

Types of Operating System:

Multi-user operating system: A multi-user operating system allows more than one user to use a computer system either at the same time or different time. Exp: Linux, Windows 2000.

Multi-tasking operating system: A multi-tasking operating system allows more then one program to run at the same time. Exp: UNIX, Windows 2000.

Multithreading operating system: A multithreading operating system allows the running of different parts of a program at a same time. Exp: UNIX, Linux.

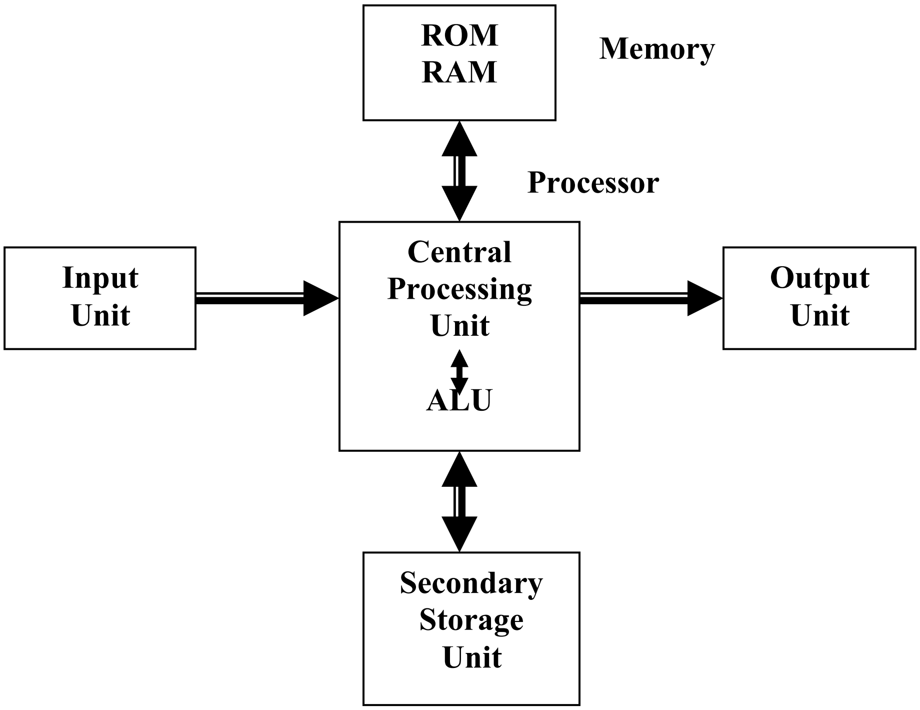
Q. Explain Storage Classes.

S.no.	Storage class & Declaration	Where stored	Initial value	Scope	Life span
1.	Automatic: Example: auto int x; By default a variable is auto if its storage class is not specified.	In memory	garbage	Local to the block in which it is declared.	The variable is accessible only while program is inside the block in which variable has been declared and used.
2.	Static: Example: static int x; Variables which are	In memory	Static variables are initialized to 0.	Same as auto variables.	Same as auto variables, except that the previous value

	required to maintain their values until the next function call are declared as static.				is retained and can be used.
3.	Register: Example: register int x; Only integer variables can be declared as register.	Such variables are stored in the CPU registers for quick access provided a register is free , otherwise it will be treated as auto variable.	garbage		Same as auto variables.
4.	External: Example: Extern int x;	In memory	Initialized to 0.	Such variables are declared on the top of the program after the preprocessor directives; and are globally accessible.	As long as program runs.

Q. Explain Block diagram of digital computer and its components.

Block diagram of Digital Computer:



Digital computer is an electronic device. It comprises of many units. These units work in coordination with each other to perform the given task. Each unit exists in the form of electronic equipments called devices.

Input Unit: The role of input unit in computer is to provide means for supplying data or instructions to the computer.

Exp: Keyboard, Mouse, CD-ROM Drive, Bar code reader, Scanner & OCR, Web camera, Light pen, Touch Screen, Joystick, Microphone, Digital Camera.

Output Unit: The role of the output unit is to show the result of processing. In other words, we can say that computer displays all the results on its output unit.

Exp: Monitor, Speakers, Printers (Dot Matrix, Inkjet & Laser)

Storage Devices: All input / output units which provides means for storing the data, instruction or results permanently are called storage devices (input / output units).

Exp: Floppy Drive, Hard Disk, CD-Writer (CD-R, CD-R/W), Magnetic Tape Drive, Pen Drive

Central Processing Unit (CPU):The CPU consists of Control Unit (CU) and Arithmetic Logical Unit (ALU). CU stores the instruction set, which specifies the operations to be performed by the computer. CU transfers the data and the instruction to the ALU for an arithmetic operation. ALU performs arithmetical or logical operation on the data received. The CPU register stores the data to be processed by the CPU and the processed data also.

Memory: Memory holds data, instruction or results temporarily we can say that the unit which holds data instruction or results in it, is called memory.The memory Unit of a computer are classified as Primary memory and Secondary memory

Primary Memory: - RAM, ROM and Cache Memory

- RAM:- RAM is volatile memory that temporarily stores the data and applications as long as they are in use.
- ROM:- It is the permanent memory of the computer where the contents cannot be modified by an end user.It is generally used to store the basic Input/Output system(BIOS) which performs the Power On Self Test(POST).
- Cache:- Cache memory is used for faster processing of the instruction.This memory lies between the processor and the main memory.It is used to store the data and the related application that was last processed by the CPU.

Secondary Memory:-Secondary memory are referred as secondary storage devices .These storage devices provides a non volatile area wherein the information can be stored which is not in use currently.

Eg:Magnetic storage device,Optical storage device.

Q. Explain Data types used in C programming.

(a) Data Types - C language is rich in its data types.Storage representations and machine instructions to handle constants differ from machine to machine. The variety of data types available allow the programmer to select the type appropriate to the needs of the application as well as the machine.

ANSI C supports three classes of data types:-

User-defined data type:

1) type –defined data type: It is used to rename lengthy and existing data type to make a program easy to understand.

Syntax: typedef existing data type new data type

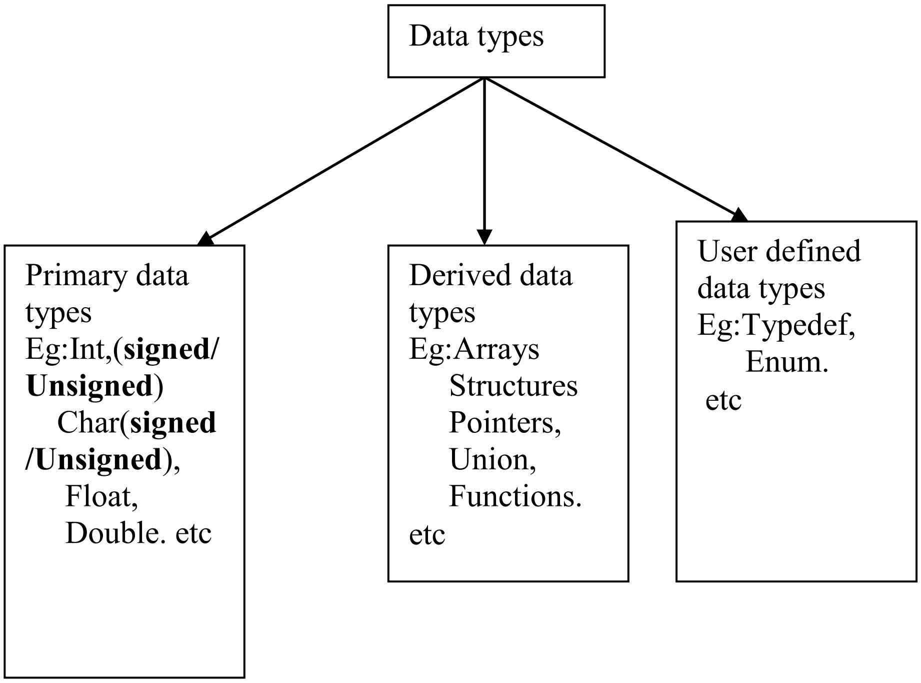
For example: typedef long int f

Here typedef is a keyword used to rename long int data type into short data type such as f.

Now there is no need to write every time long int in a program, we can just write f instead of long int to declare a variable of long int type.

2) Enumeration data type: It allows us to create our own data type with predefined constant Values.

Here a is enum name and first,second and third are constant values which will will be treated as 0, 1 , 2…n so on if no values are assigned to them. r is a enum variable.

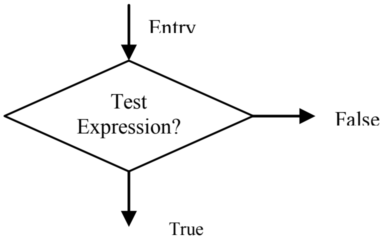


Data type	Range	Bytes	Format
Char or Signed char	-128 to +127	1	%c
Unsigned char	- 0 to 255	1	%c
signed int or int	- 32768 to + 32767	2	%d
unsigned int	0 to 65535	2	%u
Signed short int or Short int	-128 to 127	1	%h
Unsigned short int	0 to 255	1	%hu
Signed Long int	- 2,147,483,648 to + 2,147,483,647	4	%ld
Unsigned long int	0 to 4294967295	4	%lu
Float	3.4e-38 to 3.4e+38	4	%f
Double	1.7e-308 to1.7e +308	8	%lf
Long double	3.4e-4932 to 1.1e + 4932	10	%Lf

Q. Explain various decision Control Statements with examples.

The if statement is a powerful decision making statement and is used to control the flow of execution of statement.

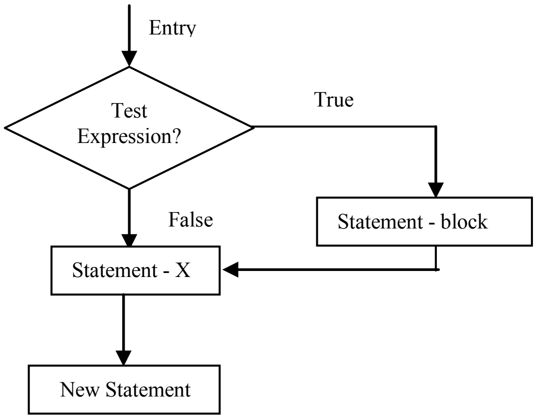
if (test expression)

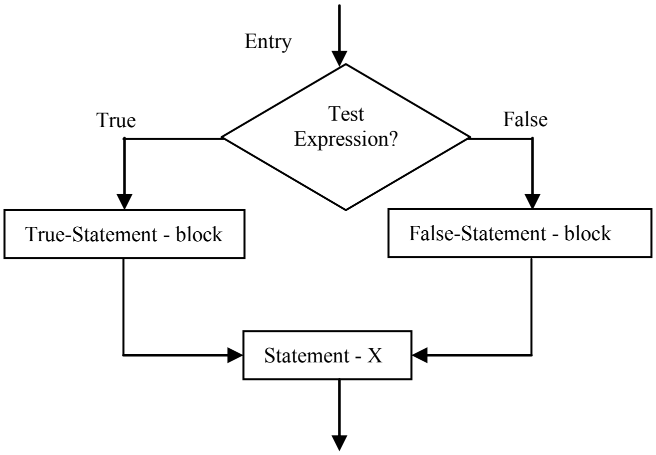


Simple If statement:	IF….Else statement

if (test expression)
{
Statement-block;
}
Statement-X;
	if (test expression)
	{
	True-Statement-block;
	}
	else

```
{
 False-Statement-X;
}
```





Exp: number is even or odd

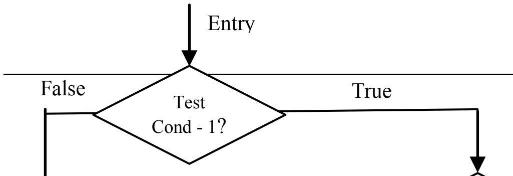
```
#include<stdion.h>
#include<conio.h>
void main( )
      {
        int num;
        clrscr( );
        printf("Enter the number");
        scanf("%d",& num);
        if ( num % 2 = = 0)
               {
                Printf("number is even");
               }
        if ( num % 2!= 0)
               {
                Printf("number is odd");
               }
        getch( );
      }
```

Exp: number is even or odd

```
#include<stdion.h>
#include<conio.h>
void main( )
      {
        int num;
        clrscr( );
        printf("Enter the number");
        scanf("%d",& num);
        if ( num % 2 = = 0)
               {
                Printf("number is even");
               }
        else
               {
                Printf("number is odd");
               }
        getch( );
      }
```

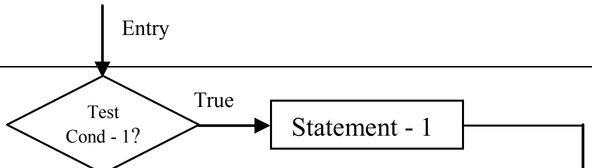
Nested If … else statement:

```
if ( test condition-1)
  {
    if ( test condition-1)
      {
        Statement-1;
      }
    else
      {
        Statement-2;
      }
  }
else
  {
    Statement-3;
  }
 Statement - X
```



Else if ladder statement

```
if (condition 1)
    {
      Statement - 1;
    }
  else if (condition 2)
     {
       Statement - 2;
     }
        .
        .
    else if (condition n)
        {
         Statement - n;
        }
       else
          default – statement;
```



Exp: find the largest of three numbers.

```
#include<stdio.h>
#include<conio.h>

void main( )
      {
       int A,B,C;
       clrscr( );

       printf(“Enter the numbers:”);
       scanf(“%d%d%d”,&A, %B, %C);

       printf(“Largest value is:”);


                                          cont..

     if(A > B)
         {
          if(A > C)
              printf(“%d”,A);
          else
              printf(“%d”,C);

         }
      else
         {
           if(c > B)
              printf(“%d”,C);
           else
               printf(“%d”,B);
         }
      getch();
      }
```

Exp: find the average marks & grade of students. Mark of five subjects input by keyboard.

```
#include<stdio.h>
#include<conio.h>

 void main( )
      {
       int math, phy, chem, bio, eng;
       float avg;
       clrscr( );

       printf("Enter the marks:");
       scanf("%d%d%d%d%d",&math, &phy,
            &chem, &bio, &eng);

                                            Cont..
       avg=(math+phy+chem+bio+eng)/5.0;

       printf("Average: %f\n",avg);


       if(100>=avg && avg>=80)
          printf("Honours");

       else if(79>=avg && avg>=60)
           printf("First Division");

       else if(59>=avg && avg>=50)
           printf("Second Division");

       else if(49>=avg && avg>=39)
           printf("Third Division");

       else
           printf("Fail");

       getch();
}
```

Q.2 What do you mean by looping? Explain for, while and do-while loop.

Looping

A looping process would include the following four steps:
1. Setting and initialization of a condition variable.
2. Execution of the statements in the loop.
3. Test for a specified value of the condition variable for execution of the loop.
4. Incrementing or updating the condition variable.

C provides three types of loops:
1. The while loop / statement
2. The do - while loop / statement
3. The for loop / statement

The While Statement

```
Syntax:
```

```
--
----
initialization;

while( test condition)
      {

       body of the loop;

      incrementing / decrementing;
      }
-----
--
```

```
--
----
sum = 0;
n = 1;        /* initialization */

while(n<=10)       /* condition */
   {               /* loop body */

    sum = sum + n;
    n = n+1;    /* incrementing */

   }
printf(“sum =%d\n”,sum)
----
--
```

The Do - - While Statement

```
Syntax:
```

```

--
----
initialization;

  do
    {

    body of the loop;

    incrementing / decrementing;
    }
  while( test condition);
-----
--
```

```
----
sum = 0;
n = 1;        /* initialization */

 do
   {               /* loop body */

    sum = sum + n;
    n = n+1;    /* incrementing */

   }
   while(n<=10);   /* condition */

printf(“sum =%d\n”,sum)
----
```

Exp: Sum of n numbers using while loop

Exp: Sum of n numbers using do – while loop

```
#include<stdio.h>
#include<conio.h>
void main()
   {
    int i,n,sum;
    clrscr();
    printf(“Enter the value of n”);
    scanf(“%d”,&n);
    sum = 0;
    i = 1;
    while(i<=n)
        {
        sum = sum + i;
        i = i+1;
        }
    printf(“sum =%d\n”,sum)
    getch();
   }
```

```
#include<stdio.h>
#include<conio.h>
void main()
   {
    int i,n,sum;
    clrscr();
    printf(“Enter the value of n”);
    scanf(“%d”,&n);
    sum = 0;
    i = 1;
      do
        {
         sum = sum + i;
         i = i+1;
        }
      while(i<=n);
    printf(“sum =%d\n”,sum)
    getch();
   }
```

Exp: Factorial of n using while loop

```
#include<stdio.h>
#include<conio.h>
void main()
   {
    int i,num,fact;
    clrscr();
    printf(“Enter the value of num”);
    scanf(“%d”,&num);
    fact = 1;
    i = 1;
    while(i<=num)
        {
        fact = fact * i;
        i = i+1;
        }
   printf(“factorial =%d\n”,fact);
   getch();
   }
```

Exp: Factorial of n using do – while loop

```
#include<stdio.h>
#include<conio.h>
void main()
   {
    int i,num,fact;
    clrscr();
    printf(“Enter the value of num”);
    scanf(“%d”,&num);
    fact = 1;
    i = 1;
      do
        {
         fact = fact * i;
         i = i+1;
        }
      while(i<=num);
   printf(“factorial =%d\n”,fact);
   getch();
   }
```

The For Statement

Syntax:

```
for(initialization; test-condition; increment)
    {

      Body of the loop;

    }
```

```
for(i=1; i<=10; i++)
  {

   printf(“%d”,i);

  }
```

Exp: Factorial of n using for loop.

```
#include<stdio.h>
#include<conio.h>

void main()
   {
    int i, num, fact=1;
    clrscr();
    for(i=1;i<=num;i++)
       {
         fact=fact*i;
       }
```

```
     printf("factorial : %d",fact);
     getch();
   }
```

Comparison of the three loops

for	while	do - while

for(n = 1; n <= 10; n++)
{
------- 	
------- 	
------- 	
}
	n=1;
	while(n<=10)
	{
	----- 	
	----- 	
	n = n + 1;
	}
		n=1;
		do
		{
		------ 	
		-------- 	
		n = n + 1;
		}
		while(n<=10);

The Nested For Statement

```
--------
      for(i=1;i<=10;i++)
          {
            -----
            -----
          for(j=1;j<=10;j++)
              {
                -----           inner       outer
                -----           loop        loop
              }
            -----
            -----
          }
--------
```

Q.3 WAP to print the following pattern.

```
        *
        * *
        * * *
        * * * *

#include<stdio.h>
#include<conio.h>
void main()
  {
   int n,i,j;
   clrscr();
   printf("\nEnter the no of rows:");
   scanf("%d",&n);

    for(i=1;i<=n;i++)
       {
       for(j=1;j<=i;j++)
          {
         printf("*");
          }
      printf("\n");
       }
   getch();

  }
```

Q.4 WAP to print the following pattern.

```
      *
    * * *
  * * * * *
* * * * * * *

#include<stdio.h>
#include<conio.h>
void main()
  {
   int n,i,j,k;
   clrscr();
   printf("\nEnter the no of rows:");
   scanf("%d",&n);

    for(i=1;i<=n;i++)
       {
      for(j=1;j<=n-i;j++)
        {
         printf(" ");
        }
      for(k=1;k<=(2*i-1);k++)
        {
          printf("*");
        }

      printf("\n");
       }
   getch();
  }
```

Q.5 WAP to print the following pattern.

```
    1
    2 3
    4 5 6
    7 8 9 10
#include<stdio.h>
```

q.6 WAP to print the following pattern.

```
    1
    0 1
    1 0 1
    0 1 0 1
#include<stdio.h>
```

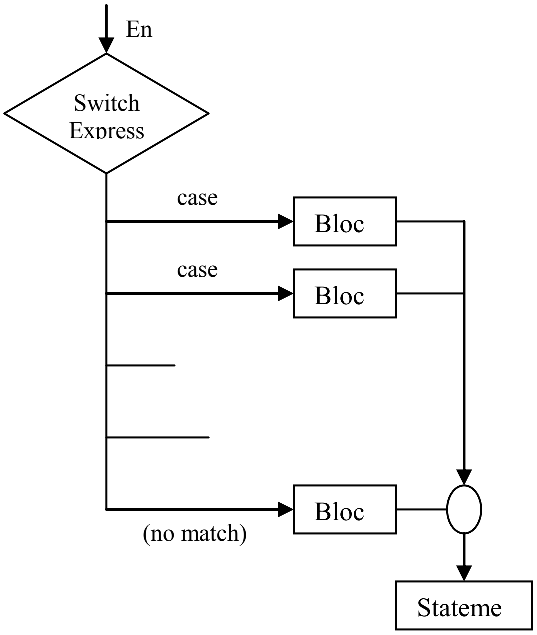
```
#include<conio.h>
void main()
   {
    int i,j,k,n;
    clrscr();
    printf("Emter the value of n: ");
    scanf("%d",&n);
    k=1;
    for(i=1;i<=n;i++)
      {
       for(j=1;j<=i;j++)
         {
          printf("%d\t",k);
          k++;
         }
       printf("\n");
      }
   getch();
  }
```

```
#include<conio.h>
void main()
  {
   int i,j,n;
   clrscr();
   printf("Enter the n: ",&n);
   scanf("%d",&n);
   printf("\n");
   for(i=1;i<=n;i++)
     {
       for(j=1;j<=i;j++)
        {
         if(i%2!=0&&j%2!=0)
            printf("%d",1);
         else if(i%2==0&&j%2==0)
            printf("%d",1);
         else
            printf("%d",0);
        }
       printf("\n");
     }
   getch();
  }
```

Q7: Write the syntax of 'switch' statement and explain how it is different from 'if' statement.

Ans: The general syntax is:

```
  switch ( expression )
     {
     case 1:
             block-1;
              break;
    case 2:
            block-2;
            break;
   …….
   ………
   default:
           default-block;
           break;
     }
   statement-x;
```



In the above example, the expression is first evaluated and the value id checked with the *constant1* in the case statement. If this is evaluated with the same value, then the group of statements is executed. When a *break* statement is encountered, the control is switched to the end of the *switch* statement. If the value of the expression is not equal to *connstant1*, it is checked with *constant2*. If it evaluates the same value, then the group of statements (in the case of *constant2*) are executed. When a break statement is encountered, the control is then switched to the end of the *switch* statement. This proceeds until the value of expressions equal one of the constant values given. If the value of the expression is not equal to any of the constant values given, then, by default, the statements present are executed.

S.No.	Switch	If
1.	In switch condition the condition is checked only once and jumps to required block	In multiple if statements the conditions are to be checked as many times the if statements are written
2.	A switch statement is generally best to use when you have more than two conditional expressions based on a single variable of *numeric & character* type.	A if else statement can take any kind of value in conditional expression.

3.	CASE statement is definitely better when there are more than 2 conditions because it makes the code more structured, systematic and easy to understand in terms of programming efficiency.	In if statements to get required result, we have to check many conditions first. this is time consuming
4.	Switch is better to use provided the results of the check-condition are discreet values.	A if else is better to use when multiple condition is to be evaluated.
5.	Logical operators are not allowed in cases.	All operators are allowed in if-else.

Q8: What is meant by jump statement?

Ans: Jump statements: The jump statements unconditionally transfer program control anywhere within a program. There are four types of unconditional statements in C.

1) break 2) continue 3) goto 4) return( )

1) break: It takes the control outside the block in which it is placed. It provides
An early exit from for, while, do-while, if and switch constructs.

2) Continue: It can be used only with the loops. It causes the loop to be continued with the next iteration after skipping any statement in between.

3) goto statement:

It can unconditionally transfer program control anywhere in a program. The target destination of a goto statement is marked by a label. The target label and goto must appear in the same program or function.

The syntax of goto statement is

```
 goto label;
    ……
    Statements;
   ……..
 label:
    Statements;
    ………..
```

Where label is a user supplied identifier and can appear either before & after goto.

For example:

```
  a=0;
  start :
      printf("\n hi");
  if(a>0)
     goto start;
```

Q9: Explain difference between following terms:

a) While and do-while
b) Call by value and Call by reference
c) Local variables and global variables
d) exit( ) and return( )
e) function and macro

Ans: Difference between the following terms:

While & DO-While:

S.No.	**While**	**Do-While**
1.	A while statement checks test-condition at the beginning of a loop to see whether the next loop iteration should occur or not.	A do-while statement checks test-condition at the end of a loop to see whether the next iteration of a loop should occur or not.

2.	It will not execute any statements written in loop block until condition is evaluated true.	The do- while statement will always execute the body of a loop at least once, even if the condition is false.

Call by value and call by reference:

S.No	Call by Value	Call by Reference
1.	Only a copy of the parameters is passed in function call. e.g. : swap(a, b); // function call	Address of the parameter is passed in the function call. e.g. : Swap(&a, &b); // function call
2.	Called function can not access the original parameters of the calling function, so changes made to the formal arguments of called function will not be reflected on the actual arguments of calling function.	Called function can access the original parameters of the calling function, so changes made to the formal arguments of called function will be reflected on the actual arguments of calling function.
3.	The function can return at most one value.	Since function has the access to original parameters, the function can indirectly return several values.

Local variables and global varaiables:

S.No.	Local Variables	Global Variables
1.	Variables that are declared inside a function are called local variables.	Variables that are declared outside a function are called global variables.
2.	It can be used only by the statements that are inside the block in which the variables are declared, i.e. local variables are not known to outside their own code block.	It can be used throughout the program in any function, regardless of what block of code that expression is in., i.e. global variables are known to outside their own code block.

3.	Local variables hold their values till that block in which it is declared.
	e.g.
	#include<stdio.h>
	int main(void)
	{
	int x=10; // local variables
	if(x==10)
	{ int x; // this hides the outer x
	x=99;
	printf(Inner x: %d\n", x);
	}
	printf("outer x: %d\n",x);
	return 0;
	}
	o/p: Inner x: 99
	outer x: 10
		Global variables hold their values throughout the program's execution. For example:
		#include<stdio.h>
		int x=10; // global variables
		
		int main(void)
		{
		if(x==10)
		{ int x; // this hides the outer x
		x=99;
		printf(Inner x: %d\n", x);
		}
		printf("outer x: %d\n",x);
		return 0;
		}
		o/p: Inner x: 99
		outer x: 10

Difference between exit( ) & return( ) :

S. No.	exit( )	return( )
1.	It is used to exit the program as a whole. In other words it returns control to the operating system.	The return () statement is used to return from a function and return control to the calling function.
2	After exit () all memory and temporary storage areas are all flushed out and control goes out of program.	It does not flush out the memory space which is reserved by a program.
3	Also in a program there can be only one exit () statement.	But a function can have number of return( ) statements.
4	exit () statement is placed as the last statement in a program since after this program is totally exited.	Return( ) statement can take its presence anywhere in the function. It need not be presented as the last statement of the function

Difference between function and macro

S.No.	macro	Function

1.	In it, we can define function definition in one line.	In function, we can write as many statements as we want in function definition.
2.	It has faster execution speed.	It has slower execution speed.
3.	No function call overhead.	There is function call overhead every time when function is called.

(Note: you can include example of each yourself for making one more point in differences)

Q10: What is the modular programming (structured programming)?

Ans: Modular programming is a strategy applied to the design & development of Software systems, it is defined as organizing a large program into small, independent program segments called modules that are separately named & individually callable program units. These modules are carefully integrated to become a software system that satisfies the system requirements. It is basically based on divide-&-conquer approach to problem solving.

Modules (functions) are identified and designed such that they can be organized into top-down hierarchical structure.

Some characteristics of modular programming are:

1) Each module should do only one thing.
2) Communication between modules is allowed only by a calling module.
3) A module can be called by one & only one higher module.
4) No communication can take place directly between modules that do not have calling- called relationship.
5) All modules are designed as single-entry, single – exit systems using control Structures.

Q11: What is the function?

Ans: A function is a self-contained block of code that performs a particular task.

Once a function has been defined, it can be treated as a 'black box' that takes some data from main program and returns a value. The inner details (coding) of a function are invisible to the rest of the program. All that the program knows about a function is : What goes in & what comes out.

Three steps are required before using a function:

1) Function Declaration(i.e., prototype): to specify the function's interface to the program.
2) Function Definition – It is used to tell the program about what & how a function is doing.
3) Function call - it is used to invoke the function at a required place.

(Note: Calling function or calling program: The program or function that calls the required function is referred to as the calling function or calling program. The calling function should declare the function (like declaration of a variable) that is used later in the program. This is known as function declaration or function prototype.)

Q12:What is an argument? Differentiate between formal arguments and actual arguments?

Ans: An argument is an entity used to pass the data from calling function to the called function. Formal arguments are the arguments available in the function definition. They are preceded by their own data types. Actual arguments are available in the function call.

Q13: What is function prototype?

Ans: A function prototype specifies a information about a function to a compiler such as function's name, input's information(data type, number,order) to a called function and returning type of a called function.

A function prototype in C or C++ is a declaration of a function that omits the function body but does specify the function's name, arity, argument types and return type. While a function definition specifies what a function does, a function prototype can be thought of as specifying its interface.

As an example, consider the following function prototype:

```
int fac(int n);
```

This prototype specifies that in this program, there is a function named "fac" which takes a single integer argument "n" and returns an integer. Elsewhere in the program a function

Advantages of function prototype:

--Informing the compiler

If a function is not previously declared and its name occurs in an expression followed by a left parenthesis, it is implicitly declared as a function that returns an `int` and nothing is assumed about its arguments. In this case the compiler will not be able to perform compile-time checking of argument types and arity when the function is applied to some arguments. This can potentially cause problems. The following code illustrates a situation in which the behavior of an implicitly declared function is unspecified.

```
#include <stdio.h>

/*
 * If this prototype is provided, the compiler will catch the error
 * in main(). If it is omitted, then the error will go unnoticed.
 */
int fac(int n);              /* Prototype */

int main() {                 /* Calling function */
    printf("%d\n", fac());   /* ERROR: fac is missing an argument! */
    return 0;
}

int fac(int n) {             /* Called function  */
    if (n == 0) {
        return 1;
    }
    else {
        return n * fac(n - 1);
    }
}
```

The function "fac" expects an integer argument to be on the stack when it is called. If the prototype is omitted, the compiler will have no way of enforcing this and "fac" will end up operating on some other datum on the stack (possibly a return address or the value of a variable that is currently not in scope). By including the function prototype, you inform the compiler that the function "fac" takes one integer argument and you enable the compiler to catch these kinds of errors.

-- Creating library interfaces: By placing function prototypes in a header file, one can specify an interface for a library.

Q14: What is recursion?

Ans: Recursion: Recursive function is a function that calls itself. When a function calls another function and that second function calls the third function then this kind of a function is called nesting of functions. But a recursive function is the function that calls itself repeatedly until base condition occurs.

 base case(s), in which the problem is simple enough to be solved directly, and

 recursive case(s). A recursive case has three components:

1. ***divide*** the problem into one or more simpler or smaller parts of the problem,
2. ***call*** the function (recursively) on each part, and
3. ***combine*** the solutions of the parts into a solution for the problem.

for example:

```
 main()
{
printf(“this is an example of recursive function”);
main();
}
```

When this program is executed. The line is printed repeatedly and indefinitely. We might have to abruptly terminate the execution.

Q38: What is the function main( ) in C?

Ans: The function main( ) is a user-defined function which specifies to the compiler that program's execution will be started from here. So each program in C must have at least one function named as main( ) . It contains all those instructions which a user wants a computer to perform to get a desired result.

significance of main( ) function:

The function main() is the first function in the program which gets called when the program executes. The startup code contains runmain() function which calls main() function. we can't change the name of the main() function.

The function main( ) invokes other functions within it. It is the first function to be When the program starts the execution.

- It is the starting function.
- It returns an int value to the environment that called the program
- It is a user-defined function.
- Recursive call is allowed for main( ) also.
- Program execution ends when the closing brace of the function main( ) is reached.
- It has two arguments 1)argument count and 2) argument vector (represents strings passed.
- Any user-defined name can also be used as parameters for main( ) instead of argc and argv.

Q.15: What is stack? Discuss the applications of stack.

Ans: Stack: The stack is also an ordered collection of elements like arrays, but it has a special feature that deletion and insertion of elements can be done only from one end, called the top of stack(TOP).Due to this property it is also know as LIFO(last in first out)type of data. It can be thought of just like a stack of plates placed on table in a party.

The stack is where all the functions' local (auto) variables are created. The stack also contains some information used to call and return from functions.

A "stack trace" is a list of which functions have been called, based on this information. When you start using a debugger, one of the first things you should learn is how to get a stack trace.

The stack is very inflexible about allocating memory; everything must be deallocated in exactly the reverse order it was allocated in. For implementing function calls, that is all that's needed. Allocating memory off the stack is extremely efficient. One of the reasons C compilers generate such good code is their heavy use of a simple stack. The memory was automatically deallocated when the calling function returned.

Stack is a last in first out linear data structure. For example, compare it with a box full of books -- you can take out only the uppermost book and can put a new book only at top.

Stack has its two functions

push( ) --> to insert an element at the top in the stack

pop( ) --> to delete an element from the top of the stack

OS uses stack for function calls -- if F1() calls F2() then on the stack F1() will be below F2(). So F1() can continue execution only when F2() completes and returns control back to F1The stack is where all the functions' local (auto) variables are created. The stack also contains some information used to call and return from functions.

Applications of stack:

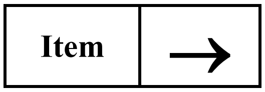
- Reversing Data: We can use stacks to reverse data. For example: files, strings. (Very useful for finding palindromes.)
- Useful in working of recursion.
- Keeping track of function calls
- ConPostponement: Evaluating arithmetic expressions.
  - Prefix: + a b
  - Infix: a + b (what we use in grammar school)
  - Postfix: a b +
- In high level languages, infix notation cannot be used to evaluate expressions. We must analyze the expression to determine the order in which we evaluate it. A common technique is to convert a infix notation into postfix notation, then evaluating it.
- Converting Decimal to Binary
- Stacks can be used to backtrack to achieve certain goals.

Q.16: What is the linked list? Discuss various application of linked list.

Ans: Linked list: It is one of the fundamental data structures used in computer programming. A linked list is called so because each of items in the list is a part of a structure, which is linked to the structure containing the next item. This type of list is called a linked list since it can be considered as a list whose order is given by links from one item to the next.

In other words, it consists of a sequence of nodes, each of which stores two items of information – an element of the list and a link. A link is a pointer and an address that indicates explicitly the location of the node containing the successor of the list element. Null in a link part of a node indicates last node of a linked list.

Structure

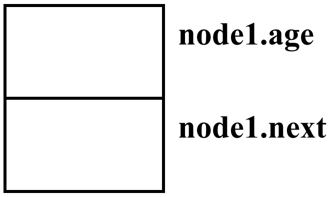


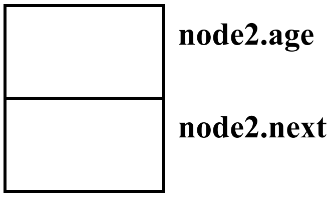
Each item has a node consisting two fields one containing the variable and another consisting of address of the next item(i.e., pointer to the next item) in the list. A linked list is therefore a collection of structures ordered by logical links that are stored as the part of data. Consider the following example to illustrator the concept of linking. Suppose we define a structure as follows

```
struct linked_list
{
float age;
struct linked_list *next;
}
struct Linked_list node1,node2;
```

this statement creates space for nodes each containing 2 empty fields

ode1 node2





The next pointer of node1 can be made to point to the node 2 by the same statement.

```
node1.next=&node2;
```

This statement stores the address of node 2 into the field node1.next and this establishes a link between node1 and node2, similarly we can combine the process to create a special pointer value called null that can be stored in the next field of the last node.

types of linked list:

There are different kinds of linked lists they are

Linear singly linked list Circular singly linked list Two way or doubly linked list Circular doubly linked list.

Advantages of Linked List:

A linked list is a dynamic data structure and therefore the size of the linked list can grow or shrink in size during execution of the program. A linked list does not require any extra space therefore it does not waste extra memory. It provides flexibility in rearranging the items efficiently.

Applications of linked lists:

Linked lists concepts are useful to model many different abstract data types such as queues , stacks and trees. If we restrict the process of insertions to one end of the list and deletions to the other end then

Limitations of the linked list:

The limitation of linked list is that it consumes extra space when compared to a array since each node must also contain the address of the next item in the list to search for a single item in a linked list is cumbersome and time consuming.

Q: Explain difference between following terms:

- **Structure and Union**

- **malloc( ) and calloc( )**
- **s) Dynamic and static memory allocation**
- **t) Array and Linked list**
- **u) array and stack**
- **w) actual and formal arguments**

Ans:

Q.17 Difference between the following terms:

a) Structure and Union:

S.No.	Union	Structure
1.	All members share same memory storage area within an union.	In structure, each member has its own unique storage area.

2.	union s
	{
	int age;
	char name[20];
	
	}e;
	void main( )
	{
	printf("%d", sizeof(e));
	}
	o/p: 20
		Struct s
		{
		int age;
		char name[20];
		
		}e;
		void main( )
		{
		printf("%d", sizeof(e));
		}
		o/p: 22

b)

S.No.	Array	Stack
1.	**Random access order:** The items can be entered or removed in any order.	**LIFO order (last in first out):** The item that is first entered would be the last removed.
2.	Fixed Size: The size of array remains fixed after declaring at the time of compilation.	Variable size: With the help of dynamic memory allocation, we can grow and shrink the size of stack at the time of execution.
3.	For example: int a[10];	For example: ptr = (int *) calloc(10, sizeof(int))

S.No.	Array	Linked list
1.	Array's elements occupy contiguous memory locations.	A linked list is not constrained to be stored in adjacent location. The individual elements are stored "somewhere" in memory, rather like a family dispersed, but still bound together.
2.	Arrays allow random access.	Linked lists allow only sequential access elements.
3.	Fixed Size: The size of array remains fixed after declaring at the time of compilation.	Variable size: With the help of dynamic memory allocation, we can grow and shrink the size of stack at the time of execution.

Q1: Write a short note on the file handling in C programming.

Ans: Many applications require that information to be written to or read from an auxiliary memory device. Such information is stored in the form of a data file which provides many benefits:

- It allows us to store large amount of information permanently.
- Easy to access

- Easy to alter the information whenever necessary.
- The transfer of input-data/ output-data from one computer to another can be done easily.

There are different operations that can be carried out on a file. These are:

a) Creation of a new file
b) Opening an existing file
c) Reading from a file
d) Writing to a file
e) Moving to a specific location in a file(seeking)
f) Closing a file.

Q2: What is the purpose of a buffer area when working with a stream- oriented data file? How is a buffer area defined?

Ans: When working with a stream-oriented data file, the purpose of a buffer area is to store information temporarily while data is being transferred between the computer's memory and data file.

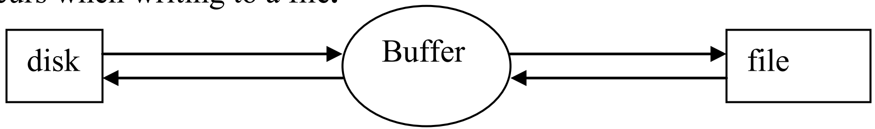
--The buffer area allows information to be read from and written to the data file more rapidly.

-- It leads to efficient disk I/O operations cause of fewer disk accesses.

When disk I/O is performed on entities then accessing the disk every time a read or write Operation is to be performed would be inefficient. Hence a buffer is used to hold the data In transit to and from a file.

-- In a buffered read operation a disk file, a fixed chunk of bytes is read from the disk into a buffer. Thus the function requesting data from the file actually read from the buffer.

-- When the buffer has no characters left, it is automatically refilled by a disk read operation. A similar sequence occurs when writing to a file**.**



Q3: What is a stream pointer? What is the relationship between a stream pointer and a buffer area?

Ans: A stream pointer holds the address of a file structure which is used to represent the Open I/O stream. It is used in ANSI C library call to identify the file.

Before opening a file, A file pointer (stream pointer) should be declared which will point to the structure containing the details of a file (file name, its status, and the current position of a file) that you are opening. The declaration of a file pointer is done in the following way:

FILE *fp;

Here fp is declared as a file pointer to the type FILE. Where file pointer need not necessarily point to the beginning the file.

- It is possible to position the file pointer to the user defined positions using inbuilt functions such as fseek().
- It identifies a specific file.
- It is used by the associated stream to direct the operation of the I/O functions.

Q4: Explain difference between following terms:

a) fprintf ( ) and fwrite( ) :

S.No	fprintf( )	fwrite( )
1.	It is a text mode function, i.e. stores data in text format.	It is a binary mode function, i.e. stores data in binary format.
2.	In text mode function, the numbers are stored as strings of characters, so numbers would occupy more number of bytes in memory. For instance, 20000, an int, which occupies two bytes in memory, would be treated as a string of characters '2','0','0','0','0',which would occupy five bytes in the file, one for each	In it, the numbers occupy as much as storage area it is required to store a single number. For example, 20000, an int, it would occupy 2 bytes in memory.

	character. Thus, three bytes would be wasted in storing a two byte number	
3.	If number of fields in the structure increase (say, by adding address, house rent etc.), writing structure using fprintf( ) becomes quite clumsy.	The format of fwrite( ) would remain same even if the number of fields belonging to the structure increases.

4.	Syntax:
	fprintf(file pointer, “control string”, variable list);
	for example:
	fprintf(fp,”%d%f”, age, salary);
		Syntax:
		fwrite(&e, sizeof(e), 1, fp);
		here, &e is the address of the structure to be written to the disk.
		sizeof(e) is the size of the structure in bytes.
		Third argument is the number of such structures that we want to write at a time.
		fp is the pointer to the file we want to write to.

Q5: What is the preprocessor?

Ans: A preprocessor is a program that processes its input data to produce output that is used as input to another program. The output is said to be a preprocessed form of the input data, which is often used by some subsequent programs like compilers. It operates under the control of preprocessor command lines and directives. Preprocessor directives are placed in the source program before the main line before the source code passes through the compiler it is examined by the preprocessor for any preprocessor directives. If there is any appropriate actions are taken then the source program is handed over to the compiler. Preprocessor directives follow the special syntax rules and begin with the symbol # and do not require any semicolon at the end. A set of commonly used preprocessor directives

Preprocessor directives:

Directive	Function
#define	Defines a macro substitution
#undef	Undefines a macro
#include	Specifies a file to be included
#ifdef	Tests for macro definition
#endif	Specifies the end of #if
#ifndef	Tests whether the macro is not def
#if	Tests a compile time condition
#else	Specifies alternatives when # if test fails

The preprocessor directives can be divided into three categories

1. Macro substitution division
2. File inclusion division
3. Compiler control division

Macros:

Macro substitution is a process where every occurrences of an identifier in a program is replaced by a macro expansion by preprocessor. The definition should start with the keyword #define and should follow on identifier and macro-expansion with at least one blank space between them.

Syntax: #define macro-identifier macro-expansion

There are different forms of macro substitution. The most common form is

1. Simple macro substitution
2. Argument macro substitution
3. Nested macro substitution

```
#define  MAX    1
#define SQUARE(x)    x*x
#define CAL(y)   y+y
Void main( )
{
   if(CAL(SQUARE(5)) > MAX)
    printf("hello");
   else
    printf("bye");
}
```

File inclusion: The preprocessor directive "`#include file name`" can be used to include any file in to your program if the functions or macro definitions are present in an external file they can be included in your file. In the directive the filename is the name of the file containing the required definitions or functions alternatively.

```
#include< filename >
```

Without double quotation marks. In this format the file will be searched in only standard directories.

Conditional Compilation:The c preprocessor also supports a more general form of test condition #if directive. This takes the following form

```
#if constant expression
{
statement-1;
statemet2;
….
….
}
#endif
```

(Note:the constant expression can be a logical expression such as test < = 3 etc)

If the result of the constant expression is true then all the statements between the #if and #endif are included for processing otherwise they are skipped.

Q6:What will the preprocessor do for a program?

Ans: The C preprocessor is used to modify your program according to the *preprocessor directives* in your source code. A preprocessor directive is a statement (such as #define) that gives the preprocessor specific instructions on how to modify your source code. The preprocessor is invoked as the first part of your compiler program's compilation step. It is usually hidden from the programmer because it is run automatically by the compiler.

The preprocessor reads in all of your include files and the source code you are compiling and creates a preprocessed version of your source code. This preprocessed version has all of its macros and constant symbols replaced by their corresponding code and value assignments. If your source code contains any
conditional preprocessor directives (such as #if), the preprocessor evaluates the condition and modifies your source code accordingly.

Q7: Discuss the following functions:

a) fread( ) b) fwrite( )

fread( ):

It is used to read blocks of any type of data from a file. It reads a specified number of equal sized data from an input stream into a block.

Syntax: size_t fread(void *buf, size_t size, size_t count, FILE *stream);

buf- It points to a block into which data is read.

Size- length of each data items read, in bytes
Count – number of data items read
Stream – points to a file stream.

On success, it returns number of data items actually read.

Program - 1

Aim: Write a program to convert Fahrenheit temperature into centigrade.
c=5*(f-32)/9.0

```
#include<stdio.h>
#include<conio.h>
void main()
  {
   int f;
   float c;
   clrscr();
   printf("Enter the temp in F:");
   scanf("%d",&f);
   c=(5*(f-32)/9.0);
   printf("temp in centigrade: %f",c);
   getch();
  }
```

Program - 2

Aim: Write a program to find the area of triangle.
Area = sqrt(s*(s-a)*(s-b)*(s-c))

```
#include<stdio.h>
#include<conio.h>
#include<math.h>

void main()
  {
   int a,b,c;
   float s,d,area;
   clrscr();
   printf("Enter the sides of tringle:");
   scanf("%d%d%d",&a,&b,&c);
   s=(a+b+c)/2.0;
   d=(s*(s-a)*(s-b)*(s-c));
   area=sqrt(d);

   printf("Area of:%f",area);

   getch();
  }
```

Program -3

Aim: Write a program to find the sum of series till n elements
1^1 + 2^2 + 3^3 +----

```
#include<stdio.h>
#include<conio.h>
#include<math.h>

void main()
  {
   int n,i;
   static temp=0;
   clrscr();
   printf("Series: 1^1 + 2^2 +3^3 +4^4 + ------");
   printf("\nEnter the no of element:");
   scanf("%d",&n);

    for(i=1;i<=n;i++)
       {
      temp=temp+pow(i,i);
       }
     printf("sum:%d",temp);
   getch();
  }
```

Program – 4

**Aim: Write a program to find the sum of series till n elements
1/!1 + 2/!2 + 3/!3 +----**

```
#include<stdio.h>
#include<conio.h>
void main()
  {
   int n,i,j;
   float fact=1;
   float sum=0;
   clrscr();
   printf("sum of series: 1/!1 + 2/!2 + ");
   printf("\nEnter the no of element for sesies:");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
      {
       for(j=1;j<=i;j++)
      {
       fact=fact*j;
      }
       sum=sum+(i/fact);
      }
    printf("Sum : %f",sum);
  getch();
  }
```

Program – 5

Aim: Write a program to find number is even or odd.

```
#include<stdio.h>
#include<conio.h>
void main()
    {
     int n;
     clrscr();
     printf("Enter the number:");
     scanf("%d",&n);

     if(n%2==0)
      {
       printf("Number is even");
      }
    else

      {
       printf("Number is odd");
      }
     getch();

    }
```

Program – 6

Aim: Write a program for finding year is leap or not.

```
#include<stdio.h>
#include<conio.h>

void main()
 {
  int year;
  clrscr();
  printf("Enter the year");
  scanf("%d",&year);
    if((year%400==0) ||(year%100!=0 && year%4==0))
       {
        printf("%d is leap year");
         }
      else
        {
        printf(" %d is not a leap year");
        }
  getch();
 }
```

Program – 7

Aim: Write a program to find ASCII Code of the entered character.

```
#include<stdio.h>
#include<conio.h>

void main()
 {
  char ch;
  clrscr();
  printf("Enter the chracter:");
  scanf("%c",&ch);
  printf("%c 's ASCII code is: %d",ch,ch);
  getch();
 }
```

Program – 8

Aim: Admission to a professional course is subject to the following condition
math>=60, phy>=50, chem>= 150.
Write a program for finding that candidtes are eliglible or not

```
#include<stdio.h>
#include<conio.h>

void main()
 {
  int math,phy,chem;
  clrscr();
  printf("Enter the marks math:");
  scanf("%c",&math);
  printf("Enter the marks phy:");
  scanf("%c",&phy);
  printf("Enter the marks chem:");
  scanf("%c",&chem);


    if(math >= 60 && phy >= 50 && chem >= 40)
       {
        printf("Candidte is eligble");
       }
     else
        {
        printf("Candidte is not eligble");
        }
  getch();
 }
```

Program – 9

Aim: write a program to find gross salary of employee if DA is 40% of basic
Salary and HRA is 20% of basic salary. Basic salary input by keyboard.

```
#include<stdio.h>
#include<conio.h>

void main()
 {
  int bs;
  float da,hra,gross;
  clrscr();
  printf("Enter the basic salary:");
  scanf("%d",&bs);

  da = (0.4*bs);
  hra = (0.2*bs);
  gross = bs+da+hra;
  printf("\nDearness allowance: %f",da);
  printf("\nHouse rent allowance: %f",hra);
  printf("\nGross salary: %f",gross);
  getch();
 }
```

Program - 10

Aim: Write a program for performing all arithmetic operation using switch case.

```
#include<stdio.h>
#include<conio.h>
void main()
  {
   int a,b,n;
   float c;
   clrscr();
   printf("Enter the number:");
   scanf("%d%d",&a,&b);
   printf("1.Addition\n");
   printf("2.Subtraction\n");
   printf("3.Division\n");
   printf("4.Multiply\n");
   printf("5.Modulo Division\n");
   printf("Enter ur choce:");
   scanf("%d",&n);

     switch(n)
       {
      case 1: c=a+b;
             printf("Add:%f",c);
             break;
      case 2: c=a-b;
             printf("Sub:%f",c);
             break;
      case 3: c=a/b;
             printf("Div:%f",c);
             break;

      case 4: c=a*b;
             printf("Muli:%f",c);
             break;
      case 5: c=a%b;
             printf("Modulo Div:%f",c);
             break;
      default:
         printf("U enter worng choice");

      }
   getch();
  }
```

Program - 11

WAP to print all prime numbers less than 500.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int count,i=1;
    int a;
    clrscr();
    while(i<=500)
      {
        count=0;
         a=1;
        while(a<=i)
              {
                         if(i%a==0)
                         count++;
                         a++;
               }
      if(count==2)
```

```
            printf("%d\t",i);
      i++;
      }
      getch();
}
```

Program – 12

WAP to search for an item from a given array.

```
    #include<stdio.h>
    #include<conio.h>
    void main()
   {
     int a[5],i;
     int ele,temp=0,pos=0;
     clrscr();
     printf("enter the array elements\n");
     for (i=0; i<5; i++)
     scanf("%d",&a[i]);
     printf("Enter the element to be search\n");
     scanf("%d",&ele);

     // searching for the element

     for (i=0; i<5; i++)
     {
         if (a[i]==ele)
         {
           temp=1;
           pos=i;
         }
      }

     if (temp==1)
     printf("Element found %d , position==%d",ele,pos);
     else
     printf("Element not found\n");
     getch();
    }
```

Program – 13

Explain a "C" program to test whether a given number is prime number or not.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,c=0,i,n;
printf("enter the number to be checked");
scanf("%d",&n);
for(i=1;i<=n;i++)
   {
     a=n%i;
     if(a==0)
        {
           c=c+1;
        }
    }
if (c==2)
  { printf("the given number is prime"); }
else
    printf("the given number is not prime");
getch();
}
```

Program – 14

Explain a "C" program to test whether a given year is leap year or not.

```
#include<stdio.h>
#include<conio.h>

void main( )
{
    int year;

    printf("Enter the year: ");
  scanf("%d",&year);

    if(year%400 ==0 || (year%100 != 0 && year%4 == 0))
    {
        printf("Year %d is a leap year",year);
    }
    else
    {
        printf("Year %d is not a leap year",year);
    }
    getch();
}
```

Program – 15

Write a 'C' program to swap two numbers: using third variable and without using third variable.

Swaping using 3rd variable

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
clrscr();
printf("enter the value for a & b\n");
scanf("%d%d",&a,&b);
printf("value of a is %d & b is %d before swapping\n",a,b);
c=a;
a=b;
b=c;
printf("value of a is %d & b is %d after swapping\n",a,b);
getch();
}
```

Swaping Without using 3rd variable

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b;
clrscr();
printf("enter the value for a & b\n");
scanf("%d%d",&a,&b);
printf("value of a is %d & b is %d before swapping\n",a,b);
a=a+b;
b=a-b;
a=a-b;
printf("value of a is %d & b is %d after swapping\n",a,b);
getch();
}
```

Program – 16

WAP to find the reverse of a number and check whether it is palindrome or not.

```
#include<stdio.h>
#include<conio.h>
void  main()
{
int   num,mod,rev=0;
printf("Enter     a     number:");
scanf("%d", &num);
while(num>0)
{
mod=num%10;
rev=(rev*10)+mod;
num=num/10;
}
printf("Reverse of the given number: %d", rev);
if(num==rev)
printf("No  is palindrome");
else
printf("No is not palindrome");
getch();
}
```

Program – 17

WAP to accept any number n and print the sum of square of all numbers from 1 to n.

```
#include<stdio.h>
#include<conio.h>
void  main()
{
int   sqr,n,sum=0;                                         ;
printf("Enter     a     number:");
scanf("%d", &n);
while(n>=1)
{
sqr=n*n;
sum=sum+sqr;
n=n-1;
}
printf("sum of square of all numbers from 1 to %d is %d",n,sum);
getch();
}
```

Program – 18

Write a program to accept three numbers and smallest number among them.

```
 # include<stdio.h>
#include<conio.h>
void main()
{
   int a,b,c;
   printf("enter 3 nos");
scanf("%d%d%d",&a,&b,&c);
if((a<b)&&(a<c))
printf("smallest no among given three no is %d",a);
else
{if(b<c)
 printf("smallest no among given three no is %d",b);
else
printf("smallest no among given three no is %d",c);}
getch();
}
```

Program – 19

Write a program to print odd numbers starting from 1 to 10 and printing their sum also.

```
 # include<stdio.h>
 #include<conio.h>
```

```
    void main()
{ int i sum=0;
   for(i=1;i<=10;i++)
      {  if (i%2!=0)
           {printf( "%d\n",i);
          sum=sum+i;
            }
}
printf("sum of odd nos :%d",sum);
getch();
}
```

Program – 20

Write a program, which accepts salary of a person. If salary is greater than Rs. 10,000 but less than 20,000 then it should print "executive". If the salary is greater than or equal to 20,000 then it should print "manager". In rest of condition, it should print "worker".

```
 #include<stdio.h>
#include<conio.h>
void main()
{int sal;
printf("enter salary of a person");
scanf("%d",sal);
if (sal<=10000)
printf("WORKER");
else
{if ((sal>10000)&&(sal<20000))
printf("EXECUTE");
else
printf("MANAGER");
}
getch();
}
```

Program – 21

Print week days according to user choice using switch statement.

```
#include<stdio.h>
#include<conio.h>

 void main( )
      {
       int choice;
       printf("Enter Choice:");
       scanf("%d",&choice);

       switch(choice)
           {
           case 1:
                 printf("Monday");
                 break;
           case 2:
                 printf("Tuesday");
                 break;
           case 3:
                 printf("Wednesday");
                 break;
           case 4:
                printf("Thursday");
                break;
           case 5:
                printf("Friday");
                break;
           case 6:
                printf("Saturday");
                break;
           case 7:
                printf("Sunday");
                break;
           default:
              printf("U enter wrong choice");

           }
       getch();
   }
```

Program – 22

Write a program to print the all even numbers starting for 1 to 100 in reverse order i.e. 100, 98, and 96 and so on.

```
 #include<stdio.h>
#include<conio.h>
```

```
void main()
{ int i;
for (i=100; i>1;i--)
  { if(i%2==0)
printf(“%d”,i);}
getch();
}
```

Program - 23

Write a program to accept value of N and the sum of N term Fibonacci series.

```
  #include<stdio.h>
#include<conio.h>
void main()
{ int i, n, a=0,b=1,sum=0,s=1;
   printf(“enter the no “);
scanf(“%d”,&n);
for(i=1;i<=n;i++)
{ sum=a+b;
 a=b;
 b=sum;
s=s+sum;
}
printf(“sum of %d term Fibonacci series is %d”,n,s);
getch();
}
```

Program - 24

WAP to count even and odd numbers in an array.

```
#include<stdio.h>
#include<conio.h>
 void main()
{
  int a[5],count_even=0,count_odd=0,i;

   for(i=0;i<5;i++)

       scanf("%d",&a[i]);
/* display the number of odd and even intergers */
for(i=0;i<5;i++)
{
   if((a[i]%2 ==0))
          count_even++;
    if((a[i]%2==1))
          count_odd++;
}
printf("%d %d",count_even,count_odd);
getch();
}
```

Program - 25

What is pointer? Write a C program to swap the values of two variables making use of pointers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b;
printf(“enter 2 values”);
scanf(“%d%d”,&a,&b);
swap(a,b);
printf("%d%d",b,a);
getch();
}
swap(int *x,int *y)
{
int t;
t=*x;
*x=*y;
*y=t;
printf("%d%d",x,y);
}
```

Program – 26

Write a program to calculate the addition of two matrix of size NxN

```
 #include<stdio.h>
 #include<conio.h>

 void main()
 {
      int a[50][50],b[50][50],c[50][50],i,j,n;
      clrscr();
      printf("Enter the size of matrix n i.e. n xn not more than 50");
      scanf("%d",&n);

      /* Entry of values into First Matrix */
      printf("\nEnter the Data in first Matrix\n");
      for(i=0;i<n;i++)
      {
            for(j=0;j<n;j++)
            {
                  printf("Enter the value in matrix");
                  scanf("%d",&a[i][j]);
            }
      }

      /* Entry of values into Second Matrix */
      printf("\nEnter the Data in Second Matrix\n");
      for(i=0;i<n;i++)
      {
            for(j=0;j<n;j++)
            {
                  printf("Enter the value in matrix");
                  scanf("%d",&b[i][j]);
            }
      }

      /* Addition of Matrix */
      for(i=0;i<n;i++)
      {
            for(j=0;j<n;j++)
            {
                  c[i][j]=a[i][j] + b[i][j];
            }
      }

      /* Printing Of Resultant Matrix */
      printf("\nThe Resultant matrix is\n");
      for(i=0;i<n;i++)
      {
            for(j=0;j<n;j++)
            {
                  printf("\t%d",c[i][j]);
            }
            printf("\n");
      }
 getch();
 }
```

Program – 26

Write a program to calculate the multiplication of two matrix of size NxN

```
#include<stdio.h>
#include<conio.h>

void main()
{
     int a[50][50],b[50][50],c[50][50],i,j,k,n;
     clrscr();
     printf("Enter the size of matrix n i.e. n xn not more than 50");
     scanf("%d",&n);

     /* Entry of values into First Matrix */
     printf("\nEnter the Data in first Matrix\n");
     for(i=0;i<n;i++)
     {
           for(j=0;j<n;j++)
           {
                 printf("Enter the value in matrix");
                 scanf("%d",&a[i][j]);
           }
     }

     /* Entry of values into Second Matrix */
     printf("\nEnter the Data in Second Matrix\n");
     for(i=0;i<n;i++)
     {
           for(j=0;j<n;j++)
           {
                 printf("Enter the value in matrix");
                 scanf("%d",&b[i][j]);
           }
     }

     /* Initialization of Resultant Matrix */

     for(i=0;i<n;i++)
     {
           for(j=0;j<n;j++)
           {
                 c[i][j]=0;
           }
     }

     /* Multiplication of Matrix */
     for(i=0;i<n;i++)
     {
           for(j=0;j<n;j++)
           {
                 for(k=0;k<n;k++)
                 {
                       c[i][j]=c[i][j]+ a[i][k] * b[k][j];
                 }
```

```
            }
        }
        /* Printing Of Resultant Matrix */
        printf("\nThe Resultant matrix is\n");
        for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                printf("\t%d",c[i][j]);
            }
            printf("\n");
        }
    getch();
    }
```

Program – 25

Write a program to find out the transpose of a matrix of size NxN

```
    #include<stdio.h>
    #include<conio.h>
    void main()
    {
        int a[50][50],b[50][50],i,j,n;
        clrscr();
        printf("Enter the size of matrix n i.e. n xn not more than 50");
        scanf("%d",&n);
        /* Entry of values into Matrix */
        printf("\nEnter the Data in Matrix\n");
        for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                printf("Enter the value in matrix");
                scanf("%d",&a[i][j]);
            }
        }
        /* Transpose of Matrix */
        for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                b[i][j]=a[j][i];
            }
        }
        /* Printing Of Resultant Matrix */
        printf("\nThe Resultant matrix is\n");
        for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                printf("\t%d",b[i][j]);
            }
            printf("\n");
        }
    getch();
    }
```

Program – 26

Write a program to calculate the factorial of a number without using function, using function and using recursion.

Without function

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

```
{
     int n,i;
     long int f=1;
     clrscr();
     printf("Enter the number");
     scanf("%d",&n);
     for(i=1;i<=n;i++)
     {
 f=f*i;
     }
     printf("The factorial is %ld",f);
     getch();
}
```

Using function

```
'#include<stdio.h>
 #include<conio.h>
 long int fact(int n);
 void main()
 {
      int n,i;
      long int f;
      clrscr();
      printf("Enter the number");
      scanf("%d",&n);
      f=fact(n);
      printf("The factorial is %ld",f);
      getch();
 }
 long int fact(int n)
 {
      long int f=1;
      int i;
      for(i=1;i<=n;i++)
      {
               f=f*i;
      }
      return(f);
 }
```

Using Recursion

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

Program – 26
Wap for sorting an array

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[100],n,i,j,temp;
clrscr();
printf("How many elements");
scanf("%d",&n);
printf("Enter the element of array");
for(i=0;i<=n-1;i++)
 {
  scanf("%d",&a[i]);
 }
for(i=0;i<=n-1;i++)
{
  for(j=0;j<=n-1-i;j++)
     {
      if(a[j]>a[j+1])
      {
       temp=a[j];
       a[j]=a[j+1];
       a[j+1]=temp;
      }
      }
}
printf("Element of array after the sorting are:\n");
for(i=0;i<=n-1;i++)
{
printf("%d\n",a[i]);
}
getch();
}
```

Program – 27
wap for merging of two array

```
#include<stdio.h>
#include<conio.h>
void main()
{
int p,q,m,n,c;
int array1[100],array2[100],array3[200];
clrscr();
puts("Enter number of element of the first sorted array");
scanf("%d",&p);
puts("Enter element of the first sorted array");
for(m=0;m<=p;m++)
scanf("%d",&array1[m]);
puts("Enter number of element of the second sorted array");
scanf("%d",&q);
puts("Enter element of the second array");
for(n=0;n<=q;n++)
scanf("%d",&array2[n]);
c=0;
m=0;
n=0;
while((m<q)&&(n<q))
{
if(array1[m]<=array2[n])
array3[c]=array1[m++];
else
array3[c]=array2[n++];
c++;
}
while(m<p)
{
array3[c]=array1[m];
c++;
m++;
}
while(n<q)
{
array3[c]=array2[n];
c++;
n++;
}
puts("merged array in ascending order");
for(m=0;m<=c;m++)
printf("%d",array3[m]);
getch();

}
```

Program – 28

Wap for storing the 100 student detail and retrieved them with the help of structure

```
#include<stdio.h>
#include<conio.h>
struct student
{
char name[20];
int roll;
char branch[10];
};
struct student std[100];
void main()
{
clrscr();
int i;
for(i=0;i<100;i++)
{
printf("enter student detail");
scanf("%s%d%s",std[i].name,&std[i].roll,std[i].branch);
}
for(i=0;i<100;i++)
printf("%s%d%s",std[i].name,std[i].roll,std[i].branch);
getch();
}
```

Program – 28

Wap for Copy a file to another file.

```
      #include<stdio.h>
   #include<conio.h>
   #include<stdlib.h>
   void main()
   {
       FILE *fs,*ft;
      char ch;
       clrscr();
       fs = fopen(“Source.c”,"r");
     if(fs==NULL)
       {
       printf("Cannot open source file ! Press key to exit.");
       getch();
       exit(0);
      }
      ft = fopen(“Destination.c”,"w");
       if(ft==NULL)
       {
       printf("Cannot copy file ! Press key to exit.");
       fclose(fs);
     getch();
       exit(0);
       }

       while(1)
       {
       ch = getc(fs);
       if(ch==EOF)
       {
      break;
       }
       else
       putc(ch,ft);
       }

       printf("File copied succesfully!");
       fclose(fs);
       fclose(ft);
   }
```