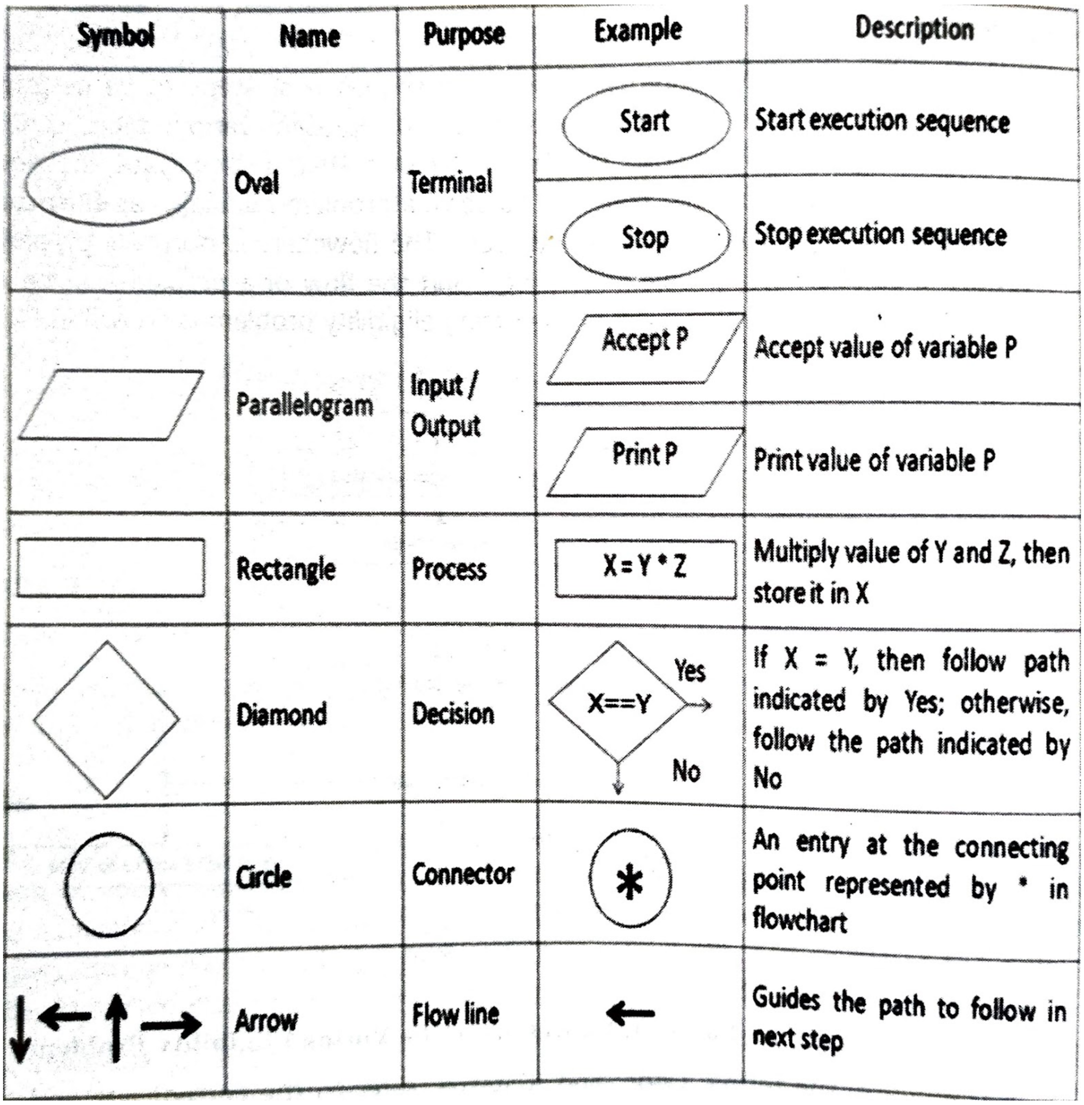
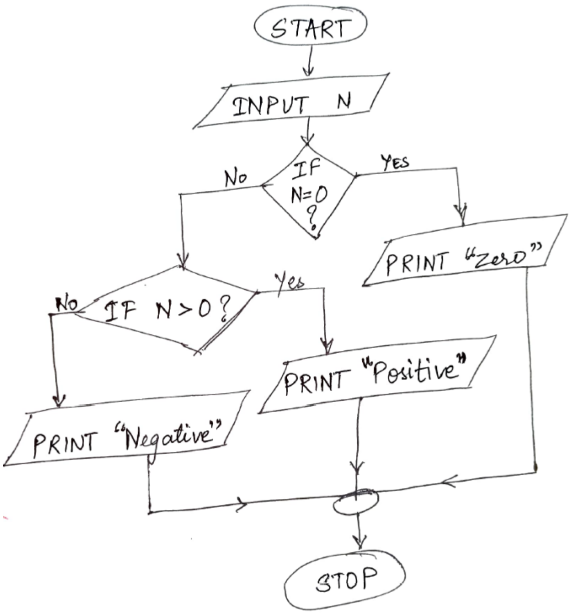
BRANCH : COMP/IT/ECS

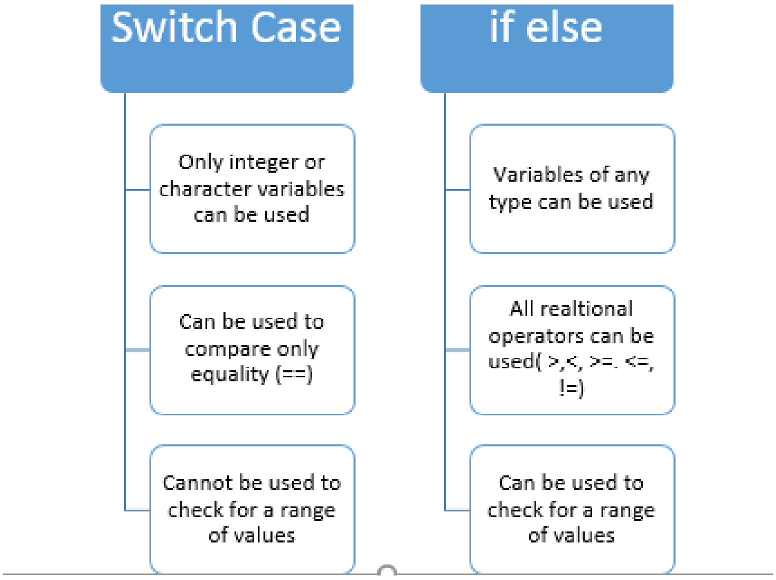
Subject: Programming in C **Time: 02.00 Hours**
Max. Marks: 60 **Date: 15-03-2022**
N.B 1. Q.1 is compulsory
2. Attempt any two from the remaining three questions

Q.1.	**Attempt all**	**M**	**BT**	**CO**
a)	**What is a flowchart ? Explain the symbols used in flowchart. Draw a flowchart to display if a given integer is positive, negative or zero.** A flowchart is a pictorial representation of a sequence of steps to be performed to solve a given problem. (1 mark) 	**5**	**1**	**1**

Symbol	Name	Purpose	Example	Description
⬭	Oval	Terminal	Start	Start execution sequence
			Stop	Stop execution sequence
▱	Parallelogram	Input / Output	Accept P	Accept value of variable P
			Print P	Print value of variable P
▭	Rectangle	Process	X = Y * Z	Multiply value of Y and Z, then store it in X
◇	Diamond	Decision	X==Y Yes No	If X = Y, then follow path indicated by Yes; otherwise, follow the path indicated by No
○	Circle	Connector	*	An entry at the connecting point represented by * in flowchart
↓←↑→	Arrow	Flow line	←	Guides the path to follow in next step



b) **Distinguish between switch-case and if-else ladder with example.** 5 1 3


Switch Case

Only integer or character variables can be used

Can be used to compare only equality (==)

Cannot be used to check for a range of values

if else

Variables of any type can be used

All realtional operators can be used(>,<, >=. <=, !=)

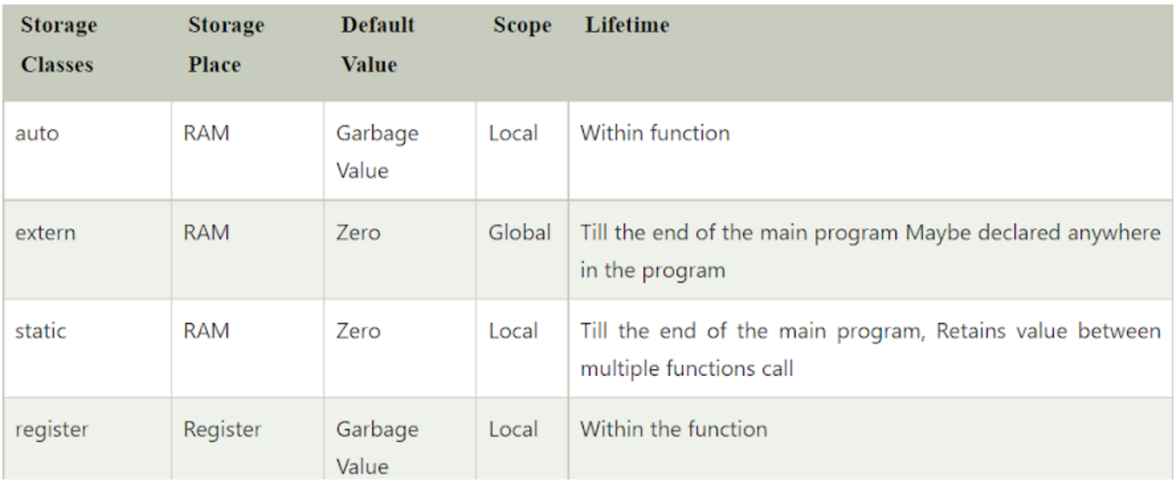
Can be used to check for a range of values

4. syntax

5. any example program

c) **What is storage class in C? Explain all storage classes with example.** 5 1 1

Storage Classes are used to describe the features of a variable/function. These features basically include the scope, visibility and life-time which help us to trace the existence of a particular variable during the runtime of a program.



Storage Classes	Storage Place	Default Value	Scope	Lifetime
auto	RAM	Garbage Value	Local	Within function
extern	RAM	Zero	Global	Till the end of the main program Maybe declared anywhere in the program
static	RAM	Zero	Local	Till the end of the main program, Retains value between multiple functions call
register	Register	Garbage Value	Local	Within the function

d)	**Explain following library functions with example :** **(i) pow(x,y)** - returns x raised to the power of y i.e. x^y. - math.h - any valid example **(ii) strrchr(s,c)** - searches for the **last** occurrence of the character c in the string s - string.h - any valid example **(iii) toupper(c)** - converts lowercase letter to uppercase - ctype.h - any valid example **(iv) strlen(s)** - returns the length of the string s up to, but not including the terminating null character. - string.h - any valid example	5	1	2
Q.2.	**Attempt all**			
a)	**What do you mean by dynamic memory allocation? Explain the different functions used for it.** An array is a collection of a fixed number of values. Once the size of an array is declared, you cannot change it. Sometimes the size of the array you declared may be insufficient. To solve this issue, you can **allocate memory manually during run-time**. This is known as dynamic memory allocation in C programming. Library functions defined in the **<stdlib.h>** header file used are : **1. malloc()** ● Stands for memory allocation ● It reserves a block of memory of the specified number of bytes. ● It returns a pointer of void which can be casted into pointers of any form. ● Syntax : ptr = (castType*) malloc(size); ● Example : ptr = (float*) malloc(100 * sizeof(float)); **2. calloc()** ● Stands for contiguous allocation. ● The malloc() function allocates memory and leaves the memory uninitialized, whereas the calloc() function allocates memory and initializes all bits to zero. ● Syntax: ptr = (castType*)calloc(n, size); ● Example: ptr = (float*) calloc(25, sizeof(float)); **3. realloc()** ● To change the size of previously allocated memory. ● Syntax: ptr = realloc(ptr, newSize); **4. free()** ● Dynamically allocated memory created with either calloc() or malloc() doesn't get freed on their own. You must explicitly use free() to release the space. ● Syntax: free(ptr);	4	1	5

b)	**Write a program to check if user entered matrix is symmetric matrix or not.**	4	4	5

```
#include <stdio.h>
int main()
{
    int A[10][10], r1, c1, i, j, flag = 1;  // flag set to 1

     printf("Enter r1 & c1 : ");
     scanf("%d%d", &r1, &c1);

     if(r1 == c1)   //check square matrix
    {
          // Take a matrix A as input from user
          printf("Enter the elements in matrix: \n");
          for(i=0; i<r1; i++)
          {
               for(j=0; j<c1; j++)
               {
                    scanf("%d", &A[i][j]);
               }
          }
// Checks whether matrix A is equal to its transpose or not
          for(i=0; i<r1; i++)
          {
               for(j=0; j<c1; j++)
               {
                    if(A[i][j] != A[j][i])
                    {
                         flag = 0;
                         break;
                    }
               }
          }
          // If the given matrix is symmetric.
          if(flag == 1)
          {
               printf("\n Matrix is Symmetric. \n");
          }
          else
          {
               printf("\n Matrix is not Symmetric.");
          }
     }
     else
     {
          printf("Not square matrix");
     }
    return 0;
}

//OUTPUT

Enter r1 & c1 : 3
3
```

```
Enter the elements in matrix:
1  2  3
2  4  5
3  5  6

 Matrix is Symmetric.
```

c)	**Write a program to check if user entered number is magic number. A magic number is that number whose sum of the digits is when multiplied by the reverse of the same sum results back the original number. Example: 1729 (1+7+2+9 = 19, reverse = 91, 19*91 = 1729)**	6	6	3

```c
#include <stdio.h>
int main()
{   int num, temp, rev=0, sumOfDigits=0;
    printf("Enter a Number \n");
    scanf("%d",&num);

    temp = num;     //take backup

    //Calculating Sum of digits
    while(temp > 0)
    {
        sumOfDigits += temp % 10; //Extract digit & add them
        temp = temp / 10;
    }

    temp = sumOfDigits;

    while(temp > 0)
    {
        rev = rev*10 + temp % 10; //Compute reverse of Sum
        temp = temp / 10;
    }

    if(rev*sumOfDigits == num)
        printf("Magic Number \n");
    else
        printf("Not a Magic Number \n");

    return 0;
}

//OUTPUT 1
Enter a Number
1729
Magic Number

//OUTPUT 2
Enter a Number
1234
Not a Magic Number
```

d) **What do you mean by recursive function ? Write a program to find sum of N natural numbers using recursion.** 6 5 4

- Recursion - The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function.
- Syntax for recursive function:

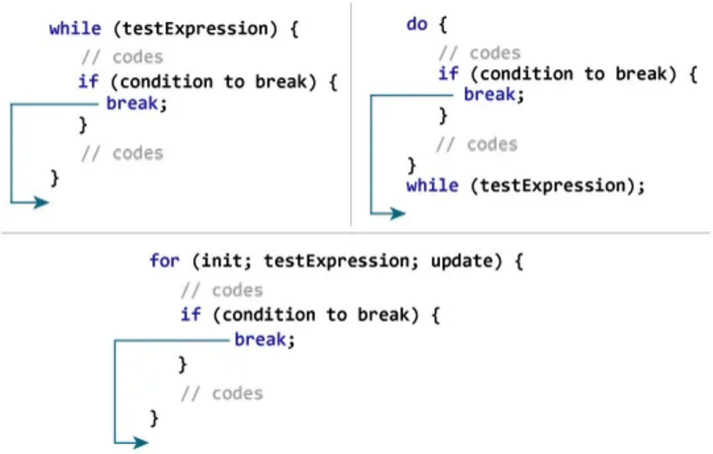
```
          void recursiveFunc()
          {
                    if(base case)
                              //something
                    else
                              //something
                              recursiveFunc();        //recursive call
          }
```

```
#include <stdio.h>
int sum(int);
int main()
{   int number;
    printf("Enter a positive integer :");
    scanf("%d", &number);
    printf("The sum of first %d numbers is %d.", number,
sum(number));
    return 0;
}

int sum(int n)
{   if (n == 0)
        return 0;
    else
        return n + sum(n - 1);
}

//OUTPUT
Enter a positive integer :10
The sum of first 10 numbers is 55.
```

Q.3. Attempt all

a) **Explain break and continue statements with example.** 4 1 2

- break - terminates the loop in which it is written or transfers control out of switch block.

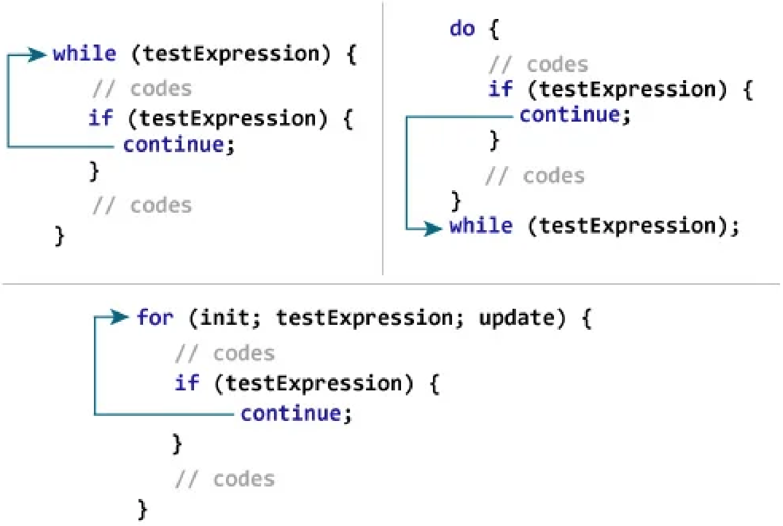


```
while (testExpression) {
   // codes
   if (condition to break) {
     break;
   }
   // codes
}
```

```
do {
   // codes
   if (condition to break) {
     break;
   }
   // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
   // codes
   if (condition to break) {
     break;
   }
   // codes
}
```

- continue - skips some lines of code inside the loop and continues with the next iteration.



```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

- any valid example

b) **Write a program to display following pattern for N lines :** 4 4 3

```
   1
  A21
 AB321
```

```
#include<stdio.h>
int main()
{     int n,i,j;

      printf("Enter no. of lines = ");
      scanf("%d",&n);

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

            for(j=1; j<i; j++)
            {
                  printf("%c",64+j);
            }

            for(j=i; j>=1; j--)
            {
                  printf("%d",j);
            }
            printf("\n");
      }

      return 0;
}
```

```
//OUTPUT

Enter no. of lines = 5
    1
   A21
  AB321
 ABC4321
ABCD54321
```

c) **Write a program to check whether two given strings are anagram of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different. For example, “silent” and “listen” are an anagram of each other.** **6** **5** **4**

Method 1 (Use Sorting)

1. Sort both strings
2. Compare the sorted strings

Method 2 (Count characters)
This method assumes that the set of possible characters in both strings is small. In the following implementation, it is assumed that the characters are stored using 8 bit and there can be 256 possible characters.

1. Create count arrays of size 256 for both strings. Initialize all values in count arrays as 0.
2. Iterate through every character of both strings and increment the count of character in the corresponding count arrays.
3. Compare count arrays. If both count arrays are the same, then return true.

Method 3 (count characters using one array)
The above implementation can be further to use only one count array instead of two. We can increment the value in count array for characters in str1 and decrement for characters in str2. Finally, if all count values are 0, then the two strings are anagram of each other.

Method 4:
Count the frequency of alphabets in both the strings and store them in respective arrays. If the two arrays are equal, return true. Else, return false.

```
#include <stdio.h>
#include <string.h>
int check_anagram(char a[], char b[])
{
     int first[26] = {0}, second[26] = {0}, c=0;
     // Calculating frequency of characters of first string
     while (a[c] != '\0')
     {
          first[a[c]-'a']++;
          c++;
     }
     c = 0;
     while (b[c] != '\0')
     {    second[b[c]-'a']++;
          c++;
     }
     // Comparing frequency of characters
     for (c = 0; c < 26; c++)
     {    if (first[c] != second[c])
               return 0;
     }
     return 1;
}

int main()
{
     char a[100], b[100];
     printf("Enter two strings : \n");
     gets(a);
     gets(b);

     if (check_anagram(strlwr(a), strlwr(b)) == 1)
          printf("The strings are anagrams\n");
     else
          printf("The strings are not anagrams\n");

     return 0;
}

//OUTPUT

Enter two strings :
SILENT
listen
The strings are anagrams
```

d)	**Write a program to count the number of vowels in a string using switch-case.**	**6**	**3**	**3**

```
#include <stdio.h>
int main()
{
    char str[50];
    int j, vowel=0;

    printf("Enter any string: ");
    gets(str);

    for(j=0; str[j] != '\0'; j++)
    {
        if((str[j]>='a' && str[j]<='z') || (str[j]>='A' &&
str[j]<='Z'))
        {
            switch(str[j])
            {
                case 'a':
                case 'e':
                case 'i':
                case 'o':
                case 'u':
                case 'A':
                case 'E':
                case 'I':
                case 'O':
                case 'U':
                    vowel++;
            }
        }
    }
    printf("Total number of vowels = %d\n", vowel);
    return 0;
}

//OUTPUT
Enter any string: hello
Total number of vowels = 2
```

Q.4. **Attempt all**

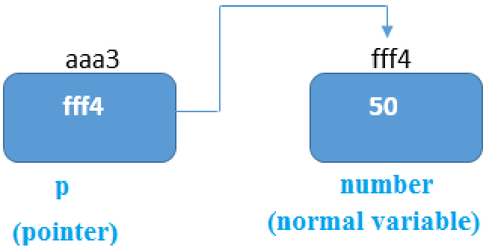
a) **Explain reference and dereference operator with example.** 4 1 2

Reference operator:

- Address of operator (“&”) is known as referencing operator.
- This operator returns the address of the variable associated with the operator.
- For e.g., if we write “&x”, it will return the address of the variable “x’.
- Hence, if we have a pointer “p”, which we want to point to a variable x, then we need to copy the address of the variable “x” in the pointer variable “p”.
- This is implemented by the statement: p = &x;

Dereference operator:

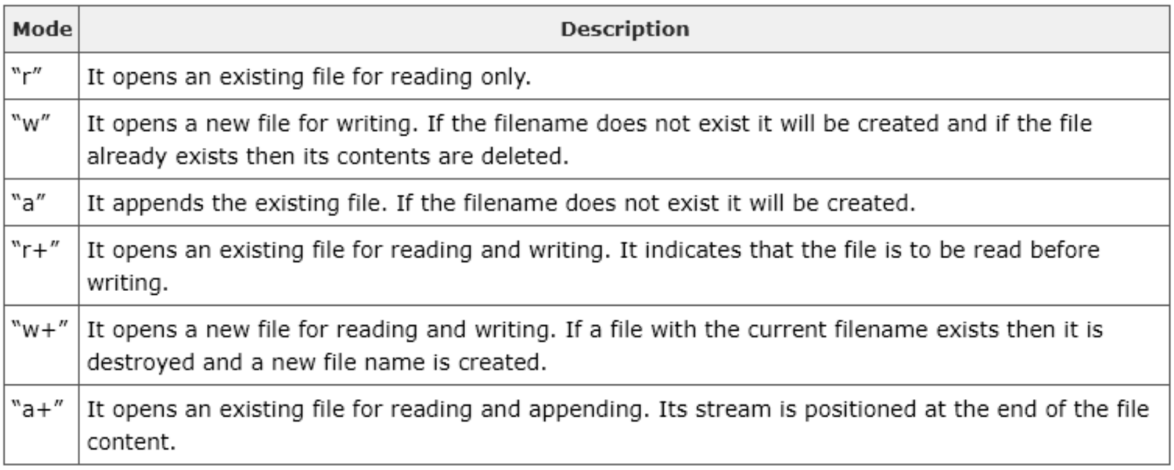
- Value of operator (“*”) is known as dereference operator.
- This operator returns the value stored in the variable pointed by the specified pointer.
- For e.g., if we write “*p”, it will return the value of the variable pointed by the pointer “p”.
- Hence, if we want the value of the variable pointed by the pointer “p” to be stored in a variable “y”, then the expression can be written as: y = *p;



b) **What is FILE ? Explain the different file opening modes.** 4 1 6

- A file is a collection of bytes which is stored on a secondary storage device like a hard disk to store data permanently.

- **FILE** is a **predefined structure** which is defined in the header file **<stdio.h>**.



Mode	Description
"r"	It opens an existing file for reading only.
"w"	It opens a new file for writing. If the filename does not exist it will be created and if the file already exists then its contents are deleted.
"a"	It appends the existing file. If the filename does not exist it will be created.
"r+"	It opens an existing file for reading and writing. It indicates that the file is to be read before writing.
"w+"	It opens a new file for reading and writing. If a file with the current filename exists then it is destroyed and a new file name is created.
"a+"	It opens an existing file for reading and appending. Its stream is positioned at the end of the file content.

c) **Write a program to store details for 'N' BankCustomer(account no., name, balance) and display list of customers whose balance is less than 5000.** 6 3 4

```
#include<stdio.h>
struct BankCustomer
{
     char name[20];
     long acc_no;
     double balance;
};

int main()
{    struct BankCustomer c[10];
     int n,i;
     double bal;

     printf("Enter the number of customers : ");
     scanf("%d", &n);

     for(i=0;  i<n;  i++)
     {    printf("\nEnter the customer's name  : ");
          fflush(stdin);
          gets(c[i].name);

          printf("Enter account no. : ");
          scanf("%ld", &c[i].acc_no);

          printf("Enter balance : ");
          scanf("%lf", &bal);
          c[i].balance = bal;
     }

     printf("\nName \t Account No. \t Balance\n");
     printf("----------------------------------------\n");

     for(i=0; i<=n-1; i++)
     {    if(c[i].balance < 5000)
               printf(" %s\t  %ld\t %.2lf \n", c[i].name,
c[i].acc_no, c[i].balance);
     }
     return 0;
}

/* OUTPUT :

Enter the number of customers : 3

Enter the customer's name  : abc
Enter account no. : 122201
Enter balance : 2340

Enter the customer's name  : xyz
Enter account no. : 122203
Enter balance : 5000

Enter the customer's name  : pqr
Enter account no. : 122202
Enter balance : 1000

Name     Account No.     Balance
----------------------------------------
 abc      122201         2340.00
 pqr      122202         1000.00

*/
```

d) **Write a program to remove the duplicate elements from an integer array. For ex: A = {2,1,3,2,1,4,4}, after removing duplicates, A={2,1,3,4}.** 6 4 4

```
#include<stdio.h>
int main()
{
   int a[50],i,j,k,size;

   printf("Enter size of the array\n");
   scanf("%d",&size);

   printf("Enter Elements of the array:\n");
   for(i=0;i<size;i++)
   {
      scanf("%d",&a[i]);
   }

   printf("Entered elements are: \n");
   for(i=0;i<size;i++)
   {
      printf("%d ",a[i]);
   }

   for(i=0;i<size;i++)
   {
      for(j = i+1; j < size; j++)
      {
         if(a[i] == a[j])
         {
            for(k = j; k <size; k++)
            {
               a[k] = a[k+1];
            }
            j--;
            size--;
         }
      }
   }

   printf("\nAfter deleting the duplicate element the Array
is: \n");
   for(i=0;i<size;i++)
   {
      printf("%d ",a[i]);
   }
}

//OUTPUT

Enter size of the array
10
Enter Elements of the array:
1
5
8
2
5
8
1
5
4
6
Entered elements are:
1 5 8 2 5 8 1 5 4 6
After deleting the duplicate element the Array is:
1 5 8 2 4 6
```

CO1: Understand the basic terminology used in computer programming.

CO2: Use different data types, operators and keywords to write programs.

CO3: Able to logically code using control statements and loops.

CO4: Use the concepts of arrays, strings, functions and Structures to structure complex programs.

CO5: Use of pointers to access different user defined data types like arrays, Strings and Structures.

CO6: Use different data structures and open/create/update basic data files.

BT Levels: - 1 Remembering ,2 Understanding, 3 Applying,4 Analyzing, 5 Evaluating, 6 Creating.

M-Marks, **BT**- Bloom's Taxonomy, **CO**-Course Outcomes.