UNIT 5

Files

File definition:-

A file can be defined as a collection of records where each record contains group of related data in the form of fields.

A file is a collection of alphabets, numbers and symbols stored on secondary memory such as hard disk.

Types of files:- Files are classified into 2 types based on its content.

They are, 1.Text files

2. Binary files

1. Text files:-

A text file stores textual information like alphabets, numbers, symbols, etc.

A text file can store different characters such as:

- Upper case alphabets (A to Z)
- Lower case alphabets(a to z)
- Numbers (like 1,2,3, etc)
- Symbols (like ? , : : ‘ . etc)

Actually ASCII code of textual characters is stored in the textual files.

Ex:- biodata**.**txt(A text file has **.**txt extension)

biodata**.**doc(A word document has **.**doc extension)

sum**.**c(A c file has **.**c extension)

2.Binary files:-

As the name suggests, binary file stores information in binary form i.e. in the form of 2 bits (0 and 1).

Binary files can’t be understood by humans where as Text files can be understood by humans.

Eg:- sum.exe (Executable file)

flower.jpg (An image file)

Based on Accessing files are of 2 types.

1. Sequential Access file

2. Random Access file

1. Sequential Access file:-

In this type data are stored sequentially. If we want to read the last record of the file, then we need to read all the records before that record. It takes more time.

For example to access the 10^{th} record of the file then the first 9 records should be read sequentially for reaching to the 10^{th} record.

2. Random Access file:-

In this type data can be read randomly. If we want to read the last

Record of the file then we can read it directly. It takes less time compared to sequential file.

Steps for file operations:- There are 3 steps for file operations.

1. Declaring and opening a file
2. Process the file using functions
3. Close the file

1.Declaring and opening a file:- If we want to store data in a file then we must specify certain things about the file to the operating system. They are,

i)file name

ii)Structure

iii)purpose

i)file name:- It is a string and it may contain 2 parts such as primary name , optional period (.) with an extension.

Eg:- sum.c where sum is primary name, period(.) and c is the extension.

Biodata.txt where Biodata is a primary name, period(.) and c is the extension.

ii)Structure:- Structure of a file is defined as FILE in stdio header file.

Therefore , all the files should be declared as type FILE before they are used.

iii)purpose:- It denotes the purpose of opening a file or what we want to do with the opened file.

Syntax to declare a file:- FILE *filepointer_variable;

Syntax to opening a file:- filepointer_variable=fopen(“file_name” , ”mode”);

Filepointer:-It is a pointer variable of FILE type.

fopen:- It is a function used to open a file for reading or writing the data.

Mode:- It is a string which specifies the purpose of opening a file.

Various Text file modes:-

1. write (w)
2. read (r)
3. append (a)
4. w+
5. r+
6. a+

1.write (w):- This mode opens a file for writing the data in to the file. If the file already exists, then the contents of the file are overwritten. If file doesn't exist then a new file will be created with that name.

Eg:- FILE *fp;

fp=fopen("sample.txt" , "w");

2.read (r) :- This mode opens a file for reading the data of an existing file. If the file doesn't exist then the compiler returns NULL to the file pointer.

Eg:- FILE *fp;

fp=fopen("sample.txt" , "r");

3.append(a):- This mode opens a file for appending or adding the data to the existing file. If file doesn't exists then a new file will be created with that name.

Eg:- FILE *fp;

fp=fopen("sample.txt" , "a");

4.w+ (Write + read) :- This mode opens a file for reading and writing.

Eg:- FILE *fp;

fp=fopen("sample.txt" , "w+");

5.r+ (read + append) :- This mode opens a file for reading and appending the data.

Eg:- FILE *fp;

fp=fopen("sample.txt" , "r+");

6.a+ (append + read):- This mode opens a file for appending and reading the data.

Eg:- FILE *fp;

fp=fopen("sample.txt" , "a+");

Various Binary file modes:-

1. write (wb)
2. read (rb)
3. append (ab)
4. wb+ or w+b
5. rb+ or r+b
6. ab+ or a+b

1.write (wb):- This mode opens a file for writing the data in to the file. If the file already exists, then the contents of the file are overwritten. If file doesn't exist then a new file will be created with that name.

Eg:- FILE *fp;

fp=fopen("sample.dat" , "wb"); Here sample.dat is a binary file.

2.read (rb) :- This mode opens a file for reading the data of an existing file. If the file doesn't exist then the compiler returns NULL to the file pointer.

Eg:- FILE *fp;

fp=fopen("sample.dat" , "rb");

3.append(ab):- This mode opens a file for appending or adding the data to the existing file. If file doesn't exists then a new file will be created with that name.

Eg:- FILE *fp;

fp=fopen("sample.dat" , "ab");

4.wb+ (Write + read) :- This mode opens a file for reading and writing.

Eg:- FILE *fp;

fp=fopen("sample.dat" , "wb+");

5.rb+ (read + append) :- This mode opens a file for reading and appending the data.

Eg:- FILE *fp;

fp=fopen("sample.dat" , "rb+");

6.ab+ (append + read):- This mode opens a file for appending and reading the data.

Eg:- FILE *fp;

fp=fopen("sample.dat" , "ab+");

3.Closing the file:- A file must be closed as soon as all operations on it have been completed. So file data will be destroyed.

Syntax to close file:- fclose(filepointer);

Eg:- fclose (fp);

Process the file using functions or Input/Output operations on files Or File I/O :-

To perform operations on the file File Input/output functions are used.

These are of 2 types.

1. Formatted file I/O functions
2. Unformatted file I/O functions or Standard file I/O functions

1.Formatted file I/O functions:- These functions specify in which format the data has to be read or displayed. There are 2 formatted file I/O functions.

1.fscanf()

2.fprintf()

1.fscanf() :- This function is used to read data from the file. fscanf() works similar to scanf().

Syntax:- fscanf(filepointer, "format specifier" , &variable);

2.fprintf() :- This function is used to write data to the file. fprintf() works similar to printf().

Syntax:- fprintf(filepointer, "format specifier" , variable);

c program to read data from keyboard and write the same data on to a file using formatted file I/O functions

```
#include<stdio.h>

int main()

{

    int a,b;

    char ch,str[20];

    FILE  *fp;

    printf(" \nEnter 2 numbers\n");

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

```
        fflush(stdin);

        printf("\nEnter a character\n");

        scanf("%c",&ch);

        fflush(stdin);

        printf("\nEnter a line of text\n");

        gets(str);

        fp=fopen("sample.c", "w");

        fprintf(fp,"%d%d%c%s",a,b,ch,str);

        fclose(fp);

        return 0;

}
```

Result:-
Input:-
Enter 2 numbers
10 20
Enter a character
A
Enter a line of text
Welcome to c programming
Output:-
sample.c file content:-
10 20 A Welcome to c programming

2.Unformatted file I/O functions:- These functions doesn't uses format specifiers like %d , %c , %f , %s ,etc.

1.fgetc()

2.fputc()

3.fgets()

4.fputs()

5.getw()

6.putw()

1.fgetc() :- This function is used to read a single character form the file. This function works like getchar( ). It returns EOF, if end of file is reached.

Syntax:- variable_name=fgetc(filepointer);

Ex:- ch=fgetc(fp);

2.fputc() :- This function is used to write a single character in to the file. This function works like putchar( ).

Syntax:- fputc(variable_name,filepointer);

Ex:- fputc(ch,fp);

Program 1:- C program to copy contents of one file to another file

```
#include<stdio.h>
int main()
{
        char ch;
        FILE  *fp1,*fp2;
        fp1=fopen("input.c","r");
        fp2=fopen("output.c","w");
        ch=fgetc(fp1);
        while(ch!=EOF)
        {       fputc(ch,fp2);
                ch=fgetc(fp1);
        }
        Printf("\nFile copied successfully");
        fclose(fp1);
        fclose(fp2);
        return 0;
}
```

Result:--

File copied successfully.

output.c file contains contents of input.c

Program 2:-C program to read data from console, write it to a file and read data from file and display on monitor.

```
#include<stdio.h>

int main()

{       char ch;

        FILE *fp;

        fp=fopen("output.c","w");

        printf("\nEnter the data & press ctrl+z to stop\n");

        ch=getchar();

        while(ch!=EOF)

        {       fputc(ch,fp);

                ch=getchar();

        }

        fclose(fp);

        fp=fopen("output.c","r");

        ch=fgetc(fp);

        while(ch!=EOF)

        {       putchar(ch);

                ch=fgetc(fp);

        }

        return 0;

}
```

Result:-

Input:- Enter the data & press ctrl+z to stop

Welcome to c programming

Ctrl+z

Output:-

Welcome to c programming

Now output.c file contains Welcome to c programming

3.fgets() :- This function is used to read a line of text or string from file.

Syntax:- fgets(variable, size, file pointer);

Ex:- fgets(str,50 , fp);

4.fputs() :- This function is used to write a line of text or string into the file.

Syntax:- fputs(variable, file pointer);

Ex:- fputs(str, fp);

C program to demonstrate fputs( )

```
#include<stdio.h>
#include<conio.h>
main()
{
        char str[50];
        FILE *fp;
        clrscr();
        fp=fopen("output.c","w");
        printf("\nenter a line of text\n");
        gets(str);
        fputs(str,fp);
        fclose(fp);
        getch();
}
```

Result:-

Input:- enter a line of text

Welcome to c programming

output.c file contains Welcome to c programming

5.getw() :- This function is used to read an integer from the file.

Syntax:-variable=getw(filepointer);

Ex:- n=getw(fp);

6.putw() :- This function is used to write an integer into the file.

Syntax:-putw(variable,filepointer);

Ex:- putw(n,fp);

C program to demonstrate putw()

```
#include<stdio.h>
#include<conio.h>
main()
{
        int n;
        FILE *fp;
        clrscr();
        fp=fopen("output.c","w");
        printf("\nenter integer\n");
        scanf("%d",&n);
        putw(n,fp);
        fclose(fp);
        getch();
}
```

Binary I/O functions:-

These functions are used to perform operations on binary file instead of text file.

There are 2 binary I/O functions. 1.fread( ) 2.fwrite( )

1.fread( ):- This function is used for reading data from binary file.

Syntax:- fread(&structure_variable, sizeof(structure_variable), number of structure_variables,file_pointer);

2.fwrite( ):- This function is used to write data to binary file.

Syntax:- fwrite(&structure_variable, sizeof(structure_variable), number of structure_variables,file_pointer);

c program to read the contents of a binary file.

```
#include<stdio.h>
#include<conio.h>
main()
{
        struct num
        {
                int a,b,c;
        }s;
        FILE *fp;
        clrscr();
        fp=fopen("sachin.dll","wb");
        s.a=10;
        s.b=20;
        s.c=30;
        fwrite(&s,sizeof(s),1,fp);
        fclose(fp);
        fp=fopen("sachin.dll","r");
        fread(&s,sizeof(s),1,fp);
        printf("\na=%d\nb=%d\nc=%d",s.a,s.b,s.c);
        getch();
}
```

Random Access Files :- ftell, rewind and fseek functions are used to access a particular record from the file.

ftell:- This function returns the current position of file pointer. The position will always be returned as long int.

Syntax:-- ftell(filepointer_variable);

Rewind():-This function is used to reset the file pointer position to the beginning of the file.

Syntax: rewind(filepointer_varaible);

fseek() :- This function is used to move the file pointer to a desired position in a file.

syntax:- fseek(file pointer, offset, position);

file pointer:- It is a pointer to the concerned file.

offset :- It is a long integer which specifies number of bytes or characters to be moved . Offset value can be either positive or negative. Positive means move file in forward direction otherwise in backward direction.

Position:- It can take one of the following three values.

Value	Meaning
0	Beginning of file
1	Current position
2	End of file

Statement	Meaning
1.fseek(fp,0L,0);	Go to the beginning (similar to rewind)
2.fseek(fp,0L,1);	Stay at the current position
3.fseek(fp,0L,2);	Go to the End of file.
4.fseek(fp,m,1);	Go forward by m bytes from the current position
5.fseek(fp,-m,1);	Go backward by m bytes from the current position.
6.fseek(fp,-m,2);	Go backward by m bytes from the end of file.

Error Handling in files:-

It is possible that an error may occur during I/O operations on a file.
Typical errors are:-
1.File is opened with invalid name.
2.when the data accessed is beyond the EOF character from the file.
3.when a file is opened for one purpose and trying to use for another purpose.
4.when a hidden file is trying to open then also we get error.
5.If the file is write protected then also we get error.

To handle these errors in c language error handling functions are available.
There are 2 Error handling functions.

1.feof( )
2.ferror( )

1.feof( ) :- This function is used to detect EOF (end of file) character in the file.
If EOF is successfully detected then it returns a non zero value otherwise it returns zero(0).It takes file pointer as argument.

Syntax:- feof(filepointer_variable);
Ex:- feof(fp);

2.ferror( ):- This function is used to detect the errors in opening of a file. It returns a non zero value when an error occurred otherwise it returns zero(0).
It also takes file pointer as argument.

Syntax:- ferror(filepointer_variable);
Ex:- ferror(fp);

C program to demonstrate Error handling functions (feof( ) and ferror( ))

```
#include<stdio.h>
#include<conio.h>
main()
{
        FILE *fp;
        char ch;
        clrscr();
        fp=fopen("st.c","r");
        if(ferror(fp))
                printf("\nError occured");
        else
        {
                while(!feof(fp))
                {
                        ch=fgetc(fp);
                        printf("%c",ch);
                }
        }
        fclose(fp);
        getch();
```

}

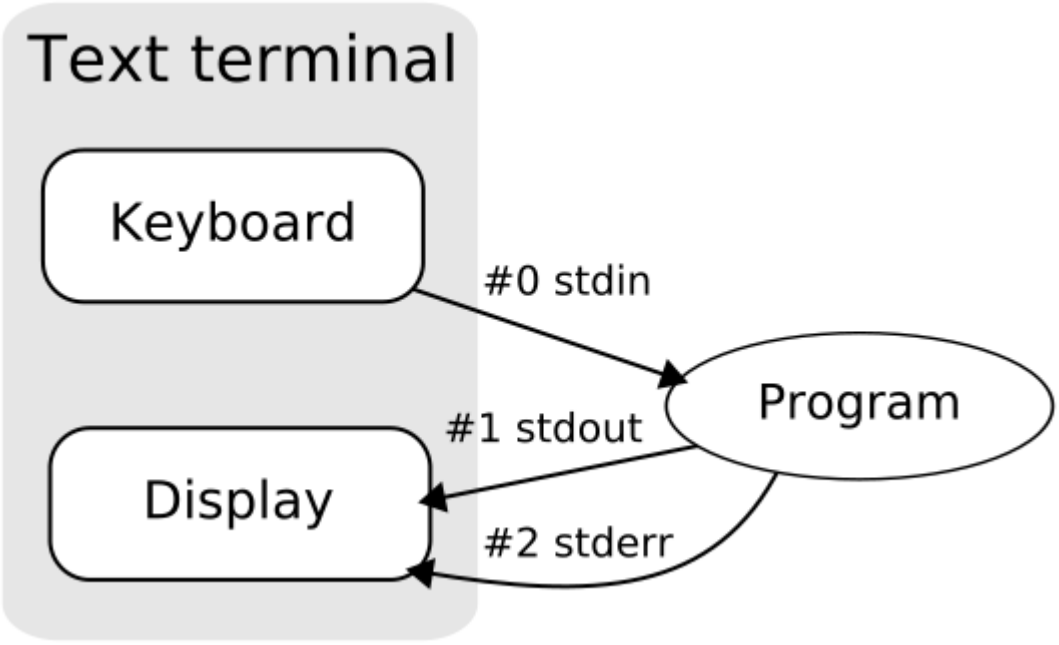
Streams:-

File I/O Streams in C Programming Language :

1. In C all **input and output** is done with streams
2. Stream is nothing but the **sequence of bytes of data**
3. A sequence of bytes flowing into program is called **input stream**
4. A sequence of bytes flowing out of the program is called **output stream**
5. Use of Stream make I/O machine independent.

Predefined Streams :

stdin	Standard Input
stdout	Standard Output
stderr	Standard Error



Standard Input Stream Device :

1. **stdin** stands for (**St**an**d**ard **In**put)
2. Keyboard is **standard input device** .
3. Standard input is data **(Often Text) going into a program**.

4. The program requests data transfers by use of the read operation.
5. Not all programs require input.

Standard Output Stream Device :

1. **stdout** stands for (**St**an**d**ard **Out**put)
2. Screen(Monitor) is **standard output device** .
3. Standard output is data **(Often Text) going out from a program**.
4. The program sends data to output device by using write operation.

Difference Between Std. Input and Output Stream Devices :

Point	**Std i/p Stream Device**	**Standard o/p Stream Device**
Stands For	Standard Input	Standard Output
Example	Keyboard	Screen/Monitor
Data Flow	Data (Often Text) going into a program	data (Often Text) going out from a program
Operation	Read Operation	Write Operation

Program 1:-
A file called NUMBERS.dat contains sequence of numbers. Write a program to read the number from the file NUMBERS.dat, write the even numbers into a file called EVEN.dat & odd numbers in to a file called ODD.dat

```
#include<stdio.h>

#include<conio.h>

main( )
```

```
{
        FILE    *fp1,*fp2,*fp3;
        int   n;
        clrscr();
        fp1=fopen("NUMBERS.dat" , "r");
        fp2=fopen("EVEN.dat" , "w");
        fp2=fopen("ODD.dat" , "w");

        while( (n=getw(fp1))!=EOF)
        {
                if(n%2= =0)
                        putw(n,fp2);
                else
                        putw(n,fp3);

        }
        fclose(fp1);
        fclose(fp2);
        fclose(fp3);
        getch( );
}
```

Program 2 :-

Write a program to count the number of vowels, number of digits present in a text file**.**

```
#include<stdio.h>
#include<conio.h>
main( )
{
        FILE    *fp;
```

```
                char    ch;
                int     nv=0, nd=0;
                fp=fopen("input.txt" , "r");
                if(fp == NULL)
                {
                        printf("\nFile    is not opened");
                        exit(1);
                }
                else
                {
                        ch=fgetc(fp);
                        while(ch != EOF)
                        {

if(ch=='a'||ch=='e'||ch=='i'||ch=='o'||ch=='u'||ch=='A'||ch=='E'||ch=='I'||ch=='O'||ch=='U')
                                nv++;
        else  if( ch>='0'  &&  ch<<='9')
                                nd++;
                                ch=fgetc(fp);

                                }
            }
                                fclose(fp);
                                printf("\nNumber   of  vowels in the file = %d", nv);
                                printf("\nNumber   of  digits  in the file = %d",nd);
                }
                getch( );
}
```

Program 3:-

Write a program to merge the contents of two files into another file.

```
#include<stdio.h>
#include<conio.h>
main( )
{
        FILE   *fp1,*fp2,*fp3;
        char    ch1,ch2;
        clrscr( );
        fp1=fopen("input1.txt" , "r");
        fp2=fopen("input2.txt",  "r");
        fp3=fopen("merge.txt" , "w");
        ch1=fgetc(fp1);
        while(ch1 != EOF)
        {
                fputc(ch1, fp3);
                ch1=fgetc(fp1);
        }
```

```
        fclose(fp1);
        ch2=fgetc(fp2);
        while(ch2 != EOF)
        {
                fputc(ch2, fp3);
                ch2=fgetc(fp2);
        }
        fclose(fp2);
        fclose(fp3);
        getch();
}
```

Program 4:-

Write a program to append data to the existing text file.

```
#include<stdio.h>
#include<conio.h>
main( )
{
        FILE   *fp1,*fp2;
        char    ch;
        clrscr();
        fp1=fopen("input.txt" , "r");
        fp2=fopen("output.txt", "a");
        ch=fgetc(fp1);
        while( ch != EOF)
        {
                fputc(ch,fp2);
                ch=fgetc(fp1);
        }
        fclose(fp1);
        fclose(fp2);
        getch( );
}
```

Program 5:-

Write a program to store N integers in text file

```
#include<stdio.h>

#include<conio.h>

main()

{

        FILE *fp;
```

```
        int i,n;
        clrscr();
        fp=fopen("output.txt","w");
        printf("\nEnter n\n");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                putw(i,fp);
        }
        getch();
}
```

Distinguish between structure and array

Array	Structure
1.Array is defined as a collection of Homogeneous(similar) data type elements.	1.Structure is def ined as a collection of Heterogeneous (different) data type Elements.
2.An Array is a derived data type	2.Structure is a user defined data type
3 Array elements are accessed using subscript and index value.	3.Structure members are accessed using Structure variable and either dot(.) or period(->) operators.
4.Memory will be allocated to array using Static memory allocation.	4.Memor will be allocated to structure Using dynamic memory allocation.
5.Arrays can't have bit fields.	5.structures can have bit fields
6.Aceesing array element takes relatively Less time than accessing structure members.	6.Accessing structure member takes Relatively more time than accessing array element.