FILES

FILES(VI)

6.0 FILES introduction :

File is a collection of bytes that is stored on secondary storage devices like hard disk. There are two kinds of files in a system. They are,

1. ASCII Text Files
2. Binary Files

A file has a beginning and an end; it has a current position, typically defined as on many bytes from the beginning. You can move the current position to any other point in file. A new current position can be specified as an offset from the beginning the file.

Types of files:

ASCII Text Files

- ✓ A text file can be a stream of characters that a computer can process sequentially.
- ✓ It is processed only in forward direction.
- ✓ It is opened for one kind of operation (reading, writing, or appending) at any give time.
- ✓ It can read only one character at a time.

Binary files :

- ✓ A binary file is collection of bytes.
- ✓ In „C" both a both a byte and a character are equivalent.
- ✓ A binary file is also referred to as a character stream, but there are two essential differences.

6.1 File streams

A stream is a series of bytes of data flow from your program to a file or vice-versa. A stream is an abstract representation of any external source or destination for data, so the keyword, the command line on your display and files on disk are all examples of stream. „C" provides numbers of function for reading and writing t o or from the streams on any external devices. There are two formats of streams which are as follows:

1. Text Stream
2. Binary Stream

Text Stream

- ✓ It consists of sequence of characters, depending on the compilers.
- ✓ Each character line in a text stream may be terminated by a newline character.
- ✓ Text streams are used for textual data, which has a consistent appearance from one environment to another or from one machine to another.

Binary Stream

- ✓ It is a series of bytes.
- ✓ Binary streams are primarily used for non-textual data, which is required to keep exact contents of the file.

6.2 File Operations

In C, you can perform four major operations on the file, either text or binary:

1. Opening a file
2. Closing a file
3. Reading a file
4. Writing in a file

6.12.1 Opening a file

To perform any operation (read or write), the file has to be brought into memory from the

storage device. Thus, bringing the copy of file from disk to memory is called opening the file.

The *fopen()* function is used to open a file and associates an I/O stream with it. The general form of fopen() is

fopen(const char *filename, const char * mode);

This function takes two arguments. The first argument is the name of the file to be opened while the second argument is the mode in which the file is to be opened.

NOTE: Before opening a file, we have to create a file pointer that is created using **FILE** structure is defined in the "stdio.h" header file.

For example

```
FILE *fp;
if(fp = fopen("csefile.txt","r")) == NULL)
{
        printf("Error opening a file");
        exit(1);

 }
```

This opens a file named *csefile.txt* in *read* mode.

File opening modes

Mode	Meaning
R	Open a text file for reading only. If the file doesn‟t exist, it returns null.
W	Opens a file for writing only. If file exists, then all the contents of that file are destroyed and new fresh blank file is copied on the disk and memory with same name. If file doesn‟t exists, a new blank file is created and opened for writing. Returns NULL if it is unable to open the file
A	Appends to the existing text file. Adds data at the end of the file. If file doesn‟t exists then a new file is created. Returns NULL if it is unable to open the file.
Rb	Open a binary file for reading
Wb	Open a binary file for reading
Ab	Append to a binary file
r+	Open a text file for read/write
w+	Opens the existing text file or Creates a text file for read/write
r+b	Open a binary file for read/write
w+b	Create a binary file for read/write
a+b	Append a binary file for read/write

6.12.2 Closing a file

To close a file and dis-associate it with a stream, use *fclose()* function. It returns zero if successful else returns EOF if error occurs. The *fc–loseall()* closes all the files opened previously. The general form is:

fclose(file pointer);

–

–6.13. Reading from the text file:

When we want to read contents from existing file, then we require opening that file into read mode that means "r" mode. To read file follow the below steps

1. Initialize the file variable
2. Open a file in read mode
3. Accept information from file
4. Write it into the output devices
5. Close the file

Example

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

        char ch;
        clrscr();
        fp=fopen("file1.c","r"); // file opened in read mode
        if(fp==NULL)
                printf("Unable to open test.txt");
        else
        {
                do
                {
                        ch = getc(fp);
                        putchar(ch);
                }while(ch!=EOF);
                fclose(fp);
        }
        getch();
        return 0;
}
```

Output

Welcome to Files. You opened a file name test.txt

Note: EOF means **End of File** that denotes the end-of-file.

Function for reading from the text files:

- ✓ There are five functions to read text from the file.

1. fscanf()
2. fgets()
3. fgetc( )
4. fread()
5. fgetw()

fscanf ():-The scanf() is used to read formatted data from the stream.

Prototype:-

————int scan–f(FILE *stream, const char *format,)

- It is used to read data from the stream and store from according to the parameter format.

Type Specifiers:-

- ✓ C → for single character
- ✓ D →decimal value–
- ✓ E,e,F,g,G →for floating
  →Octal
- ✓ S →String
- ✓ U →for unsigned
- ✓ X,x →hexadecimal

Note:-

The fscanf( ) function is similar to the scanf( ) function, except that the first argument of fscanf() specified as a stream from which to read, whereas scanf() can only read from standard.

Ex:-

```
main()
{
        FILE *fp;
        char name[70];
        int sid;
        fp=fopen("student.txt","r");
        if(fp==NULL)
        {
                printf("Unable to open");
                exit(1);
        }
        printf("Enter name and roll no:");
        fscanf(stdin,"%s %d",name, &sid);
        printf(Name:%s \Roll no="%d",name,sid);
        fscanf(fp, "%s %d", name, &sid);
        printf("In name:%s Roll number = %d",name,sid);
        fclose(fp);
}
```

fgets( ) :

The function fgets() stands for file get string.

- The function is used to get string from a stream.

Syntax:-

```
char* fgets(char *str, int size, *stream);
```

- This function reads the a line of characters/
- The fgets() function reads at most one less than the no. of characters specified by size from the given stream and stores them in the string str.
- It is terminated as soon as it encounters either newline character, EOF or any other error.
- It is a new line character is encountered it is retained.
- When all the characters are read without any error, a '\0' character is appended to the end of the string.
- Difference between gets and fgets all most same but the difference is

gets	fgets
➢ Infinite size and stream of stdin. ➢ When encounters a newline character, it does not retain it.	➢ Specified size. ➢ When encounters a newline it retains.

- On successfully completes ,the fgets() will return str.
- On failure, the stream is at EOF.
- If fgets() encounters any error while reading, the error indicator for stream will be set a null pointer.

Ex:

```
#include<stdio.h>
main()
{
        FILE *fp;
        char str[70];
        fp=fopen("abc.txt","r");
        if(fp==NULL)
        {
                printf("Unable to open");
                exit(1);
        }
        while(fgets(str,70,fp)!=NULL)
        {
                printf("%s",str);
        }
                printf("File read computer");
        fclose(fp);
```

fgetc( ):-

- The fgetc( ) function returns next character from streams.

Prototype:- int fgetc(FILE *stream);

- fgetc() returns the character read as an int.
- fgetc() reads a single character from the current position of a file. After reading the character, the function increments the associated file pointer to point to the next character.
- If the stream is reached the End Of File, the EOF indicator for the stream for the steam is set.

Ex:

```
#include<stdio.h>
main()
{
        FILE *fp;
         int ch;
        fp=fopen("file.txt","r");
          if(fp==NULL)
            {
              printf("Unable to open");
            }

        while((ch=fgetc(fp))!=EOF)
        {
                printf("%c",ch);
        }
}
```

fread( ):-

- The fread() function is used to read data from a file.

Prototype : int fread(void *str, size_t size, size_t num, FILE *stream);

- The function on successful, fread() returns the no. of bytes successfully read. The no. of objects will be less than num if a read error or end of file is encountered.
- If size or num is 0, fread() will return 0.

Note:-

The fread() function does not distinguish between end-of-file and error. The program must use feof and ferror to determine which of the two occurs.

Ex:

```
#include<stdio.h>
main()
{
        FILE *fp;
        char str[20];
        fp=fopen("Letters.txt","r+");
        if(fp==NULL)
        {
                printf("Unable to open");
                exit(1);
        }
        fread(str,1,10,fp);
        str[10]='\0';
        printf("\n The first character of a file %s",str);
        fclose(fp);
}
```

fgetw( ):-

- ✓ The fgetw() is used to read an integer from the file stream.

  Prototype: int fgetw(fp);

```
#include<stdio.h>
int main()
{
        int num;
        FILE *fp;
        Fp=fopen("Num.txt","r");
        printf("The numbers in the file are:");
        num=getw(fp);
        while(num!=EOF)
                printf("%d",num);
        return 0;
}
```

6.14 Writing to the file :

When we want to write new data to the file, then we require opening that file in write mode that means "w" mode. To write in to the file follow the below steps

1. Initialize the file variable
2. Open a file in write mode
3. Accept information from the user
4. Write in to the file
5. Close the file

Example

```
#include <stdio.h>
int main()
{
        FILE *fp;
        char c;
                                                // file opened in write mode

        fp = fopen("test.txt", "w");
        printf("Enter your Input\n");
        while((c =getchar()) != EOF)            //writing data to the file
                putc(c,fp);
        fclose(fp);

        printf("\nYou have entered\n");
        fp = fopen("test.txt","r");
        while((c =getc(fp)) != EOF)             //reading data from file
                printf("%c ",c);
        fclose(fp);
        return 0;
}
```

Output

Enter your Input
Hi Students…! How are you?
All the best for your end exams. **(press CTRL+Z for stopping)**

You have entered
Hi Students…! How are you?
All the best for your end exams.

Functions to write a text to file:

- ✓ There are five functions to write text into files they are :
    1. fprintf( )
    2. fputs( )
    3. fputc( )
    4. fwrite( )
    5. fputw( )

(1) fprintf():-

- ✓ The fprintf() is used to write formatted output to stream. It same as printf( ), but here first argument is file pointer.
- ✓ This function write the information to file.

Prototype : int fprintf(FILE *stream, const char *formal);

```
#include<stdio.h>
main()
{
        FILE *fp;
        int i,n;
        char sname[20];
        int sid;
```

```
        float smarks;
        printf("Student details:");
        for(i=0;i<n;i++)
        {
                puts("Enter student sID:");
                scanf("%d",&sid);
                puts("Enter student Name:");
                scanf("%d",sname);
                puts("Enter student marks:");
                scanf("%f",smarks);
                printf(fp, "sid:%d\t Name:%s \t marks:%f \t", sid,sname,smarks);
        }
        fclose(fp);
}
```

fputs():-

The opposite of fgets() is fputs(). The fputs() is used to write a line to a file.

Syntax:-

```
int fputs(const char *str, File *stream);
```

Ex:-

```
#include<stdio.h>
main()
{
        FILE *fp;
        char greeting[100];
        fp=fopen("wishes.txt","w");
        if(fp==NULL)
                printf("\n Enter your wishes");
        gets(greeting);
        fflush(stdin);
        fputs(greeting, fp);
        fclose(fp);
}
```

fputc():-

- ✓ It is opposite to fgetc() and is used to write a character to the stream.

  Syntax:- int fputc(int c, FILE *stream);

- ✓ On successful completion, fputc() will return the value it has written otherwise error, the function will return EOF.

```
#include<stdio.h>
main()
{
        FILE *fp;
        char welcome[100];
        int i;
        fp=fopen("Welcome.txt","w");
        if(fp==NULL)
                printf("\n Provide welcome message");
        gets(welcome);
for(i=0;i<feedback[i];i++)
fputc("feedback[i]", fp);
```

```
fclose(fp);
}
```

fwrite():-

- ✓ It is used to write data to a file.

Prototype: int fwrite(const void *str, size_t, size, size_t count, FILE *stream);

- ✓ The fwrite() function will write objects of size specified by size.
- ✓ The file position indicator for the stream will be advanced by two no. of bytes successfully written.
- ✓ fwrite() function returns the no. of successfully written. The no. of objects will be less than count if an error is encountered.
- ✓ If size or count is 0, fwrite() will return 0.

Ex:-

```
#include<stdio.h>
main()
{
        FILE *fp;
        size_t count;
        char str[]="GOOD MORNING";
        fp=fopen("Welcome.txt","wb");
        count=fwrite(str, 1, strlen(str), fp);
        printf("\n%d bytes were written to the file",count);
        fclose(fp);
        return 0;
}
```

fputw():-

The putw() is used to write an integer to file system.

Syntax :

int fputw(word, fpt);

```
#include<stdio.h>
main()
{
        FILE *fp;
        int i, n, val;
        fp=fopen("file1", "wb");
        if(fp==NULL)
        {
                printf("File does not exists");
                exit(1);
        }
        else
        {
                printf("\nEnter the no. of values to be stored:");
                scanf("%d",&n);
                printf("\nEnter the values:");
                for(i=1;i<=n;i++)
                {
                        scanf("%d",&val);
                        fputw(val, fr);
                }
        } }
```

<table>
<tr><td></td><td></td></tr>
<tr><td>Formatted I/O functions allow to supply input or display output in user desired format.</td><td>Unformatted I/O functions are the most basic form of input and output and they do not allow to supply input or display output in user desired format.</td></tr>
<tr><td><code>printf()</code> and <code>scanf()</code> are examples for formatted input and output functions.</td><td><code>getch(), getche(), getchar(), gets(), puts(), putchar()</code> etc. are examples of unformatted input output functions.</td></tr>
<tr><td>Formatted input and output functions contain format specifier in their syntax.

printf (format, data1, data2,……..);

printf (“%c”, data1);

The character specified after % is called a conversion character because it allows one data type to be converted to another type and printed.

See the following table conversion character and their meanings.

<table><tr><th>Conversion Character</th><th>Meaning</th></tr><tr><td>d</td><td>The data is converted to decimal (integer)</td></tr><tr><td>c</td><td>The data is taken as a character.</td></tr><tr><td>s</td><td>The data is a string and character from the string , are printed until a NULL, character is reached.</td></tr><tr><td>f</td><td>The data is output as float or double with a default Precision 6.</td></tr></table></td><td>Unformatted input and output functions do not contain format specifier in their syntax.

Puts();

Gets();</td></tr>
<tr><td>Formatted I/O functions are</td><td>Unformatted I/O functions are used for</td></tr>
</table>

used for storing data more user friendly.	storing data more compactly.
Formatted I/O functions are used with all data types.	Unformatted I/O functions are used mainly for character and string data types.

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

void main()
{
 int a;
 clrscr();
 printf("Enter value of a:");
 scanf("%d", &a);
 printf(" a = %d", a);
 getch();
}
```

```
Enter value of a:5↵
a = 5
```

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

void main()
{
 char ch ;
 clrscr();
 printf("Press any character:");
 ch = getche();
 printf("\nYou pressed :"
 putchar(ch);
getch();
}
```

```
Press any character: L
You pressed: L
```

6.15 RANDOM ACCESS FILES:

- ✓ We can access the file in any place that's from beginning , ending and middle of the file also.
    1. fseek( )
    2. ftell( )
    3. rewind( )
    4. fgetpos( )
    5. fsetpos( )

We will be able to change file pointer position using these functions.

(1) fseek():-

The function fseek() is used to reposition a binary stream.

Prototype:-

int fseek(FILE *stream, long offset, int origin);

- fseek() is used to set the file position pointer for the given stream.
- It is used to move the file pointer to different position using fseek function.

Syntax:-

fseek(file pointer, displacement, pointer position);

File Pointer --> It is the pointer which points the file.
Displacement--> It is positive or negative. This is the no. of bytes which one skipped backward (if negative) or forward (if positive) from the current position that is attached with long integer.

Pointer Position:-

- ✓ This sets the pointer position in the file. Origin values should have one of the following.

0	Beginning of file	SEEK_SET
1	Current position	SEEK_CUR
2	End of file	SEEK_END

Ex:-

fseek(fp, 10, SEEK_SET) **OR** fseek(fp, 10, 0);
fseek(fp, -10, SEEK_END) **OR** fseek(fp, -10, 2);
fseek(fp, 2, SEEK_CUR) **OR** fseek(fp, 2, 1);

Program:-

```
#include<stdio.h>
main()
{
        FILE *fp;
        fp=fopen("abc.txt","w+");
        fputs("RGUKT NUZVID", fp);
        fseek(fp, 7, SEEK_SET);
        fputs("ANDHRA PRADESH", fp);
        fclose(fp);
        return 0;
}
```

OUTPUT:

abc.txt

RGUKT ~~NUZVID~~
RGUKT ANDHRA PRADESH

ftell():-

The ftell() function used to know the current position of file pointer.

Syntax:-

long int ftell(File Pointer);

Here, file pointer points to the file whose file position indicator has to be determined.

- If Successful, ftell() returns current file position.
- If Fail, it returns -1.

Ex:-

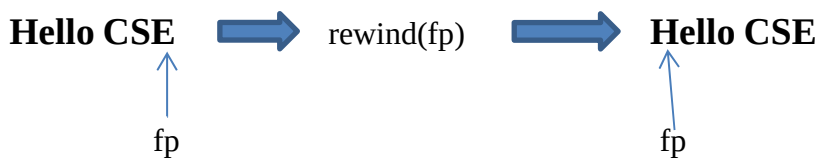
Hello CSE

```
printf("%d", ftell(fp));  --> 9
```

```
#include<stdio.h>
main()
{
        FILE *fp;
        int len;
        fp=fopen("file.txt","r");
        fseek(fp, 0, SEEK_END);
        len=ftell(fp);
        printf("%d", len);
        fclose(fp);
}
```

rewind():- rewind() is used to move the file pointer to the beginning of the file.

Syntax:- rewind(fp);
where fp is a file pointer.



• rewind() is equivalent to calling fseek() with following arguments.
fseek(fp, 0, SEEK_SET);

```
#include<stdio.h>
main()
{
        FILE *fp;
        char ch;
        fp=fopen("abc.txt", "r");
        while((ch=fget(fp))!=EOF)
        {
                printf("%c", ch);
        }
        rewind(fp);
}
```

fgetpos():-

- ✓ The fgetpos() is used to determine the current position of the file pointer.

Syntax:-

```
int fgetpos(FILE *stream, fpos_t *pos);
```

stream --> This is a point to a file.
Pos --> Pointer to fpos_t object.

- ✓ The function returns zero on success, else non-zero value in case of an error.
- ✓ getpos( ) can be used by two fsetpos() to return to this same position.

fsetpos():-

The fsetpos() is used to move the file position indicator of a stream to the location indicated by the information obtained in position by making a call to the fgetpos().

Syntax:-

```
int fsetpos(file *stream, const fpos_t pos);
```

- Stream is a file pointer.
- The pos is a position given by function fgetpos.
- This function return zero if successful.
- It returns a non-zero value and sets the global variable errno to positive value.

```
#include<stdio.h>
main()
{
        FILE *fp;
        fpos_t  position;
        fp=fopen("file.txt", "w+");
        fgetpos(fp, &position);
        fputs("Hello CSE!", fp);
        fsetpos(fp, &position);
        fputs("This is going to override previous content", fp);
}
```

6.16 Remove( ):-

- ✓ The function remove() is used to erase a file.
  Prototype:
  int remove(const, char *filename);
- ✓ On success, it return 0.
- ✓ On failure, it returns non-zero value.

Ex:

```
#include<stdio.h>
main()
{
        remove("abc.txt");
        return 0;
}
```

6.17 Renaming the file:-

It is used to rename a file.

Syntax:-

int remove(const char *old name, const char *new name);

- On success, it return 0.
- On failure, it return non-zero.

```
#include<stdio.h>
main()
{
        int success=0;
        success=rename("abc.txt", "xyz.txt")
        if(success==0)
        {
                printf("The file name renamed");
        }
        else
        {
                printf("File name not changed");
        }
}
```

Type definition (typedef)

- ✓ The typedef keyword enables the programmer to create a new data type(duplicate data type) from existing data type.
- ✓ No new name is created, rather than alternate name is given to a known data type

Syntax: typedef existing_data_type new_data_type

Ex : typedef int cse

- ✓ Here, cse is a new name of data type int.
- ✓ By using typedef, we can create synonymous for available data type.
- ✓ typedef statement doesn't occupy any memory ,it simply define new type.
- ✓ it is also local and global.

Example:

```
#include<stdio.h>
int main( )
{
typedef int cse;
cse a=10,b=20,c=30;
printf("sum:%d",a+b+c);
}
```

OUTPUT: sum: 30

typedef using Array:

```
#include<stdio.h>
int main( )
{
int i=0;
 typedef  int Array[5];  // Array is a new name for integer array with specified size;
Array x={1,2,3,4,5};
printf("Array elements are:");
for( i=0;i<5;i++)
{
```

```
printf("%d\n",x[i]);
}
}
```

OUTPUT:
1
2
3
4
5

- ✓ In above program int changes to new data type i.e Array , it stores the array elements with specified size.
- ✓ When we are declare the variable with Array data type . that variable holds the elements .
- ✓ Array x; x is a variable it holds the five elements.

typedef using Structures:

- ✓ We know that, the structure data type have two words i.e first one struct keyword and name, both two words together a structure data type.
- ✓ Here when space occurred between two words peoples may think both are different keywords. To simplify the structure datatype as a single word we can use the typedef statement.
- ✓ typedef using in structures in two ways.1. after defining the structure 2. Along with structure definition.

Ex:

```
struct student
{
 int sid;
char name[20];
float marks;
};

typedef struct student std; // std is a new name for struct datatype

std s; // declaration of struct variable
```

Ex:

```
typedef  struct
{
 int x;
 int y;
} student; // new name for struct

main( )
{
student st={10,20};
pf("%d",st.x+st.y);
}
```

Here no need to provide structure name, and here student is a not a variable. It is a new name for struct data type. student is a data type.

typedef using in pointers:

- ✓ In c language, there is a no data type for strings . so here taking a character type it want change to string data type.

Ex:-

```
#include<stdio.h>
typedef char* string;
char* read(void);
void main( )
{
string name;
name=read( );
printf("Welcome %s"name);
}
string read( )
{
string name;
printf("Enter Name:");

gets(name);
return name;
}
```

OUTPUT:
Enter Name: CSE
Welcome CSE

Enumerated Data type:-

- User defined data type based on standard integer.
- Working with a set of elements using constant integer values.
- The functionality of a particular element we are accessing just with help of constant integer value called as enum.
- It contains set of named integer constant. Each integer value is assigned an identifier.
- It provide symbolic name to make the program more readable.

Syntax:-

enum enumeration name {identifier1, ------ , identifier n};

- The enum keyword basically used to declare and initialize a sequence of integer constants. Here enumeration name is optional.

Ex:-

enum E1CSE{CSE1, CSE2, CSE3, CSE4, CSE5}; **ouput : 1,2,3,4,5**

enum E1CSE{CSE1=2, CSE2, CSE3=10, CSE4, CSE5=24}; **ouput : 2,3,10,11,24**

Rules:-

- An Enumeration list may contain duplicate constant values. Therefore, two different identifiers may be assigned the same value.
- The identifiers in the enumeration list must be different from other identifiers in the same scope with same visibility including ordinary variable names and identifiers in other enumeration lists.
- Enumeration names follow the normal scoping rules. So, every enumeration must be different from other enumeration, structures and unions.

Ex:-

```
#include<stdio.h>

int main()

{

 enum E1CSE{CSE1=2, CSE2, CSE3=10, CSE4, CSE5=24};

 prinnf("\n CSE1=%d",CSE1);

prinnf("\n  CSE2=%d",CSE2);

prinnf("\n CSE3=%d",CSE3);
```

```
prinnf("\n CSE4=%d",CSE4);

prinnf("\n CSE5=%d",CSE5);

return 0;

}
```

OUTPUT:
CSE1=2
CSE2=3
CSE3=10
CSE4=11
CSE5=24

Typedef with num:-

typedef enum E1CSE E2CSE;

Enumeration type conversion:-

enum colors{red, blue, black, green, yellow, purple, white};

enum colors c;

c=black+white

2 + 6 = 8

Two things

- To declare c as int
- Cast the right handisde of

c=enum colors(black+white);

c=black;

//c=2; // error

c=(enum colors) 2;

enum colors{r,b,b,g,y,p,w};

enum colors c; scanf("%d",&c); printf("%d = %d", c);