FILES

File: A file is a collection of related data treated as a unit stored on external storage devices. Some important external(auxiliary or secondary) storage devices are disks(hard disks, cd, dvds) and magnetic tapes.

Advantages of files :

- Since contents of primary memory are lost when the computer is shut down, files can be used to store data permanently.
- Can store large amounts of data in external devices using files. If data is so large it cannot be fit in primary memory.
- Once available file can be accessed and modified and restored whenever required.

Types of files : There are two types of files.

1. Binary files .
2. Text files**.**

Text files :

- Text files are those files that contain letters, digits and symbols.
- A text file contains characters coded from ASCII character set.
- Uses only 7 bits of a byte and last bit is 0.
- These are in Human readable form.
- Conversion is required to convert data to binary format.

Binary files:

- Binary files can be read and understood by a computer system.
- They are not understood by human beings.
- They uses all 8 bits of a byte.
- All executable files are examples of binary files.
- No conversion is required.

Stream :

Stream is a sequence of bytes, used to read and write data to a file. streams contains input data of a program are **input streams** and that contain o/p data of a program are **o/p streams.**

Steps in processing a file : There are four steps in processing a file**.**

1. Create the stream.
2. Open the file to associate stream name with physical file name.
3. Read/write the data
4. Close the file

1.Creating a stream :

Stream (also known as file pointer or stream pointer) is created when we declare it. The declaration uses FILE type which is a structure contains information needed for reading and writing a file in stdio.h. We must include stdio.h.

Synatax :
FILE *ptrvariable;
Eg : FILE *fp;

2. <u>Opening a File</u> : Once the stream has been created, we can associate the stream to a file. This can be done using a standard library function fopen as follows.

fp = fopen("filename", "mode");

eg :

fp = fopen("st.dat","w");

Filename : filename is the physical name given to the file. It can have two parts, primary name and extension. Some eg filenames are
x.txt, student.dat, xyz.c etc

File mode : When we open the file we must specify the mode of the file. The mode shows how we will use the file : for reading, for writing or for appending. C has six different modes**.**
ropen text file in read mode, if file exists the pointer positions beginning of file, if not exists it returns error.

r+ Open for reading and writing. The pointer positioned at the beginning of the file.

w create text file for writing. The pointer is positioned at the beginning of the file. If file already existed , it will be erased

w+ Open for reading and writing. The file is created if it does not exist, otherwise it is erased. The pointer is positioned at thebeginning of the file.

a Open for appending (writing at end of file). The file is created if it does not exist. The pointer is positioned at the end.

a+ Open for reading and appending (writing at end of file). The file is created if it does not exist. The initial file position for reading is at the beginning of the file, but output is always appended to the end of the fil**e.**
To work with binary files the letter b is appended. i.e., modes will be rb, wb, ab, rb+, wb+, ab+ for binary files.

<u>Closing of a file</u> :

When file is not needed, if we finish the work, the file must be closed, frees buffer space.
A file is closed using fclose function.
fclose(fp); // closes the opened file
fcloseall(); // closes all open files

<u>Input /output operations on files (reading/writing) :</u>

File I/O

Formatted i/o unformatted i/o

<u>Formatted i/o :</u>These functions read and write all types of data values. The functions fprintf() and fscanf() perform a formatted i/o operations on files.

Syntax: fprintf(fp,"format string",list of variables);
eg : fprintf(fp,"%s %d %d ",name,m1,m2); to print the values of name, m1 and m2 on file.
fscanf(fp,"format string",list of variables);
eg : fscanf(fp,"%s %d %d ",name,&m1,&m2); reads data from file and place in variables name , m1 and m2.

Q. write a program to create a file with item code, item name , price, qty and calculate total value

```
#include <stdio.h>
main()
{
  FILE *fp;
  fp=fopen("item.dat","w");  // file is opened for writing
  do
  {
       printf("enter item name, code, price and qty ");
       scanf("%s%d%d%d",name,&code,&price,&qty);
                                                  // read data from keyboard
       fprintf(fp, "%s\t%d\t%d\t%d",name,code,price,qty);
                                                      // write data to file
       printf("\nAny more data ");
       scanf("%c",&ans);
  }
  while(ans=='y'); // if ans is yes repeats loop to read data and write to a file
  fclose(fp);  // closes the file
}
```

//Example program to read data from a file calculate total value and print

```
#include <stdio.h>
main()
{
  FILE *fp;
  int total;
  fp=fopen("item.dat","r");  // file is opened for writing
  if (fp==NULL)
   { printf("File not existed ");  exit(0);  }

 while(fscanf(fp,"%s%d%d%d",name,&code,&price,&qty))
                                  // read data from file until data exists
   {
        total = price * qty;
        printf( "%s\t%d\t%d\t%d\t%d",name,code,price,qty,total);
                                                      // write data on screen
  }
  while(ans=='y'); // if ans is yes repeats loop to read data and write to a file
  fclose(fp);  // closes the file
```

}

Unformatted File I/O :

Character i/o : These functions read/write one character at a time.

getc() : Get one character from a file (macro)
fgetc() : Get a character from a file (function)
putc() : Put a character to a file (macro)
fputc() Put a character from a file (function)

syntax:

```
char c;
c=getc(fp); //get a character from file and store in variable c
putc(c,fp); // print the value of variable c in file
  or
c=fgetc(fp);
fputc(c,fp);
```

eg :

Q. write a program to copy the contents of one file to other

```
//  program to copy data of one file to another file (File copy program)
#include<stdio.h>
main()
{
 FILE *fp1,*fp2;
 fp1= fopen("x.dat","r"); // existed file
 fp2= fopen("y.dat","w"); // new file
if (fp1==NULL)
 {
   printf("\nFile not exist ");  exit(0);
 }
if (fp2==NULL)
 {
   printf("\nFile cannot be created, disk may be full ");  exit(0);
 }

while ((c=getc(fp1))!=EOF)  // read a character from fp1 and write it to fp2 until EOF
  putc(c,fp2);
fcloseall();
}
```

o/p : After executing the above program the contents of x.dat will be copied to y.dat.

String File i/o : These functions read/write a string at a time.

fgets() Get a string from a file
syntax : fgets(string,size,fp);
fputs() Writes a string to a file
syntax : fputs(string,fp);

Eg program using fgets, fputs :

```
#include<stdio.h>
main()
{
 FILE *fp1;
 char s[50];
 fp1= fopen("x.dat","r"); // existed file
 fgets(s,20,fp1);  // reads 20 characters from file and write to string s
 printf("\nFirst 20 characters of file = %s ",s);
fclose(fp);
fp=fopen("x.dat","w"); //
 fputs(s,fp1); // writes string to file
}
```

integer i/o : These functions read/write one integer number at a time.

getw() – read an integer from a file
putw() – write an integer to a file
eg:

```
  int n;
  n= getw(fp);   // get a number from file and store in n
  putw(n,fp);// writes the number n to file
```

```
#include<stdio.h>
main()
{
 FILE *fp1,*fp2,*fp3;
 if (fp1==NULL)
 {    printf("\nFile not exist ");  exit(0);
 }
 char s[50];
 fp1= fopen("nos.dat","r"); // existed file
 fp2= fopen("even.dat","w"); // new file
 fp3= fopen("odd.dat","w"); // new file
 while((!feof(fp1)) // feof returns true if end of file , false other wise
 {
    n=getw(fp1); // read a no from file
```

```
     if (n%2==0)  // if it is even write to even.dat
     putw(n,fp2); // else write to odd.dat
    else
     putw(n,fp3);
}
fcloseall();
}
```

Binary i/o : These functions read/write block of data from binary files.

fread() : Read a block of characters (for binary files)
fwrite() : Write a block of characters

syntax : fread(ptr-name, sizeof element, no of elements, filepointer);
fwrite(ptr-name, sizeof element, no of elements, filepointer);

eg :

```
 char x[50]="ABCDEFGHI";
  FILE *fp = fopen("x.txt","wb");
  fwrite(x,sizeof(x),1,fp);  // write x to file
```

Random access of files : These functions are used to access file randomly.

ftell() : Returns file position
fseek() : To go to the required position
rewind() : Moves file position to the start of file
eg :

```
  long int n= ftell(fp);
  rewind(fp);
  fseek(fp,offset,position);  // offset is no of bytes to be moved, position may be
```

0 from beginning of file 1 from current position of file 2 from end of file

eg :

```
fseek(fp,10,0);  // places fp  10 bytes from beginning
fseek(fp,10,1); // moves 10 bytes forward from current position
fseek(fp,-10,2); // move 10 bytes backward from end of file
```

Other file handling functions :

feof() : **The function feof() tests the end-of-file indicator for the stream. This function returns true if end of file reached other wise returns false.**

```
FILE *fp;
while (!feof(fp))  // if not eof
  {
  ch = getc(fp);
  }
```

ferror() : The function ferror() tests the error indicator for the stream. If any error occurred it returns true else return false.

conversion (format) specifiers for printf() function :

d	signed integer	u	Unsigned denary integer
x	Hexadecimal integer	o	Octal integer
s	String	c	Single character
f	Fixed decimal floating point	e	Scientific notation floating point
g	Use f or e, whichever is shorter		

conversion (format) specifiers for scanf() function :

d	integer	ld	Long int
x	Hexadecimal integer	o	Octal integer
h	short integer	f	Float type
lf	Long float or double	e	Float type
le	Double	c	Single character
s	Character string		

Assignment Questions :

1. Write the syntax for opening a file with various modes and closing a file
2. Explain about file handling functions
3. Write a program to read series of digits and store them into a file, then read the numbers from the file and wite all odd & even numbers into different files.
4. a) Describe types of files with an example.
   b) Explain the following operations
   1. fseek() 2. ftell () 3. rewind() 4 ferror()
5. Explain about different formatted and unformatted I/O operation function on files

Detailed Notes

A file represents a sequence of bytes, does not matter if it is a text file or binary file. C programming language provides access on high level functions as well as low level (OS level) calls to handle file on your storage devices. This chapter will take you through important calls for the file management.

Opening Files

You can use the **fopen( )** function to create a new file or to open an existing file, this call will initialize an object of the type **FILE**, which contains all the information necessary to control the stream. Following is the prototype of this function call:

```
FILE *fopen(constchar* filename,constchar* mode );
```

Here, **filename** is string literal, which you will use to name your file and access **mode** can have one of the following values:

Mode	Description
r	Opens an existing text file for reading purpose.
w	Opens a text file for writing, if it does not exist then a new file is created. Here your program will start writing content from the beginning of the file.
a	Opens a text file for writing in appending mode, if it does not exist then a new file is created. Here your program will start appending content in the existing file content.
r+	Opens a text file for reading and writing both.
w+	Opens a text file for reading and writing both. It first truncate the file to zero length if it exists otherwise create the file if it does not exist.
a+	Opens a text file for reading and writing both. It creates the file if it does not exist. The reading will start from the beginning but writing can only be appended.

If you are going to handle binary files then you will use below mentioned access modes instead of the above mentioned:

```
"rb","wb","ab","rb+","r+b","wb+","w+b","ab+","a+b"
```

Closing a File

To close a file, use the fclose( ) function. The prototype of this function is:

```
int fclose( FILE *fp );
```

The **fclose( )** function returns zero on success, or **EOF** if there is an error in closing the file. This function actually, flushes any data still pending in the buffer to the file, closes the file, and releases any memory used for the file. The EOF is a constant defined in the header file **stdio.h**.

There are various functions provide by C standard library to read and write a file character by character or in the form of a fixed length string. Let us see few of the in the next section.

Writing a File

Following is the simplest function to write individual characters to a stream:

```
int fputc(int c, FILE *fp );
```

The function **fputc()** writes the character value of the argument c to the output stream referenced by fp. It returns the written character written on success otherwise **EOF** if there is an error. You can use the following functions to write a null-terminated string to a stream:

```
int fputs(constchar*s, FILE *fp );
```

The function **fputs()** writes the string **s** to the output stream referenced by fp. It returns a non-negative value on success, otherwise **EOF** is returned in case of any error. You can use **int fprintf(FILE *fp,const char *format, ...)** function as well to write a string into a file. Try the following example:
Make sure you have **/tmp** directory available, if its not then before proceeding, you must create this directory on your machine.

```
#include<stdio.h>

main()
{
  FILE *fp;

  fp = fopen("/tmp/test.txt","w+");
  fprintf(fp,"This is testing for fprintf...\n");
  fputs("This is testing for fputs...\n", fp);
  fclose(fp); }
```

When the above code is compiled and executed, it creates a new file **test.txt** in /tmp directory and writes two lines using two different functions. Let us read this file in next section.

Reading a File

Following is the simplest function to read a single character from a file:

```
int fgetc( FILE * fp );
```

The **fgetc()** function reads a character from the input file referenced by fp. The return value is the character read, or in case of any error it returns **EOF**. The following functions allow you to read a string from a stream:

```
char*fgets(char*buf,int n, FILE *fp );
```

The functions **fgets()** reads up to n - 1 characters from the input stream referenced by fp. It copies the read string into the buffer **buf**, appending a **null** character to terminate the string.
If this function encounters a newline character '\n' or the end of the file EOF before they have read the maximum number of characters, then it returns only the characters read up to that point including new line character. You can also use **int fscanf(FILE *fp, const char *format, ...)** function to read strings from a file but it stops reading after the first space character encounters.

```
#include<stdio.h>

main()
{
  FILE *fp;
char buff[255];

  fp = fopen("/tmp/test.txt","r");
  fscanf(fp,"%s", buff);
```

```
   printf("1 : %s\n", buff );

   fgets(buff,255,(FILE*)fp);
   printf("2: %s\n", buff );

   fgets(buff,255,(FILE*)fp);
   printf("3: %s\n", buff );
   fclose(fp);

}
```

When the above code is compiled and executed, it reads the file created in previous section and produces the following result:

```
1 : This
2: is testing for fprintf...
3: This is testing for fputs...
```

Let's see a little more detail about what happened here. First **fscanf()** method read just **This** because after that it encountered a space, second call is for **fgets()** which read the remaining line till it encountered end of line. Finally last call **fgets()** read second line completely.

Binary I/O Functions

There are following two functions, which can be used for binary input and output:

```
size_t fread(void *ptr,size_t size_of_elements,
size_t number_of_elements, FILE *a_file);

size_t fwrite(const void*ptr,size_t size_of_elements,
size_t number_of_elements, FILE *a_file);
```

Both of these functions should be used to read or write blocks of memories - usually arrays or structures.

Write a C program to read name and marks of n number of students from user and store them in a file

```
#include<stdio.h>
int main(){
char name[50];
int marks,i,n;
   printf("Enter number of students: ");
   scanf("%d",&n);
   FILE *fptr;
   fptr=(fopen("C:\\student.txt","w"));
if(fptr==NULL){
      printf("Error!");
```

```
exit(1);
  }
for(i=0;i<n;++i)
  {
    printf("For student%d\nEnter name: ",i+1);
    scanf("%s",name);
    printf("Enter marks: ");
    scanf("%d",&marks);
    fprintf(fptr,"\nName: %s \nMarks=%d \n",name,marks);
  }
  fclose(fptr);
return0;
}
```

Write a C program to read name and marks of n number of students from user and store them in a file. If the file previously exits, add the information of n students.

```
#include<stdio.h>
int main(){
char name[50];
int marks,i,n;
  printf("Enter number of students: ");
  scanf("%d",&n);
  FILE *fptr;
  fptr=(fopen("C:\\student.txt","a"));
if(fptr==NULL){
      printf("Error!");
exit(1);
  }
for(i=0;i<n;++i)
  {
    printf("For student%d\nEnter name: ",i+1);
    scanf("%s",name);
    printf("Enter marks: ");
    scanf("%d",&marks);
    fprintf(fptr,"\nName: %s \nMarks=%d \n",name,marks);
  }
  fclose(fptr);
return0;
}
```

Write a C program to write all the members of an array of strcures to a file using fwrite(). Read the array from the file and display on the screen.

```
#include<stdio.h>
struct s
{
```

```
char name[50];
int height;
};
int main(){
struct s a[5],b[5];
   FILE *fptr;
int i;
   fptr=fopen("file.txt","wb");
for(i=0;i<5;++i)
   {
      fflush(stdin);
      printf("Enter name: ");
      gets(a[i].name);
      printf("Enter height: ");
      scanf("%d",&a[i].height);
   }
   fwrite(a,sizeof(a),1,fptr);
   fclose(fptr);
   fptr=fopen("file.txt","rb");
   fread(b,sizeof(b),1,fptr);
for(i=0;i<5;++i)
   {
      printf("Name: %s\nHeight: %d",b[i].name,b[i].height);
   }
   fclose(fptr);
       }
```

- Besides reading or writing one character at a time, you can also read or write one character line at a time. There is a pair of C I/O functions, fgets() and fputs(), that allows you to do so.
- The prototype for the fgets() function is:

char *fgets(char *s, int n, FILE *stream);

- s, references a character array that is used to store characters read from the opened file pointed to by the file pointer stream. n specifies the maximum number of array elements. If it is successful, the fgets() function returns the char pointers s. If EOF is encountered, the fgets() function returns a null pointer and leaves the array untouched. If an error occurs, the function returns a null pointer, and the contents of the array are unknown.

- The fgets() function can read up to n-1 characters, and can append a null character after the last character fetched, until a newline or an EOF is encountered.

- If a newline is encountered during the reading, the fgets() function includes the newline in the array. This is different from what the gets() function does. The gets() function just replaces the newline character with a null character.

- The prototype for the fputs() function is:

int fputs(const char *s, FILE *stream);

- s points to the array that contains the characters to be written to a file associated with the file pointer stream. The const modifier indicates that the content of the array pointed to by s cannot be changed. If it fails, the fputs() function returns a nonzero value, otherwise, it returns zero.

- The character array must include a null character at the end as the terminator to the fputs() function. Also, unlike the puts() function, the fputs() function does not insert a newline character to the string written to a file.

- You can also read or write a block of data at a time. There are two C I/O functions, fread() and fwrite(), that can be used to perform block I/O operations.

- The prototype for the fread() function is:
size_t fread(void *ptr, size_t size, size_t n, FILE *stream);

- The ptr is a pointer to an array in which the data is stored. size indicates the size of each array element.

-n specifies the number of elements to be read. stream is a file pointer that is associated with the opened file for reading.

- size_t is an integral type defined in the header file stdio.h. The fread() function returns the number of elements actually read.

- The number of elements read by the fread() function should be equal to the value specified by the third argument to the function, unless an error occurs or an EOF is encountered.

- The fread() function returns the number of elements that are actually read, if an error occurs or an EOF is encountered.

- The prototype for the fwrite() function is:
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *stream);

- ptr references the array that contains the data to be written to an opened file pointed to by the file pointer stream.
-size indicates the size of each element in the array. n specifies the number of elements to be written.

- The fwrite() function returns the number of elements actually written.
- If there is no error occurring, the number returned by fwrite() should be the same as the third argument in the function. The return value may be less than the specified value if an error occurs.
- That is the programmer's responsibility to ensure that the array is large enough to hold data for either the fread() function or the fwrite() function.
- In C, a function called feof() can be used to determine when the end of a file is encountered.
This function is more useful when you are reading a binary file because the values of some bytes may be equal to the value EOF.
- If you determine the end of a binary file by checking the value returned by fread(), you may end up at the wrong position.
- Using the feof() function helps you to avoid mistakes in determining the end of a file.
The prototype for the feof() function is:

int feof(FILE *stream);

- Here, stream is the file pointer that is associated with an opened file. The feof() function returns 0 if the end of the file has not been reached, otherwise, it returns a nonzero integer.

Random Access To Disk Files

- Before this you have learned how to read or write data sequentially. In many cases, however, you may need to access particular data somewhere in the middle of a disk file.
- Random access is another way to read and write data to disk file. Specific file elements can be accessed in random order.
- There are two C I/O functions, fseek() and ftell(), that are designed to deal with random access.
- You can use the fseek() function to move the file position indicator to the spot you want to access in a file. The prototype for the fseek() function is:

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

- stream is the file pointer with an opened file. offset indicates the number of bytes from a fixed position, specified by whence, that can have one of the following integral values represented by SEEK_SET, SEEK_CUR and SEEK_END.

- If it is successful, the fseek() function return 0, otherwise the function returns a nonzero value.
- whence provides the offset bytes from the file location. whence must be one of the values 0, 1, or 2 which represent three symbolic constants (defined in stdio.h) as follows:

Constants whence File location

SEEK_SET 0 File beginning
SEEK_CUR 1 Current file pointer position
SEEK_END 2 End of file

- If SEEK_SET is chosen as the third argument to the fseek() function, the offset is counted from the beginning of the file and the value of the offset is greater than or equal to zero.

- If however, SEEK_END is picked up, then the offset starts from the end of the file, the value of the offset should be negative.
- When SEEK_CUR is passed to the fseek() function, the offset is calculated from the current value of the file position indicator.
- You can obtain the value of the current position indicator by calling the ftell() function. The prototype for the ftell() function is,

long ftell(FILE *stream);

- stream is the file pointer associated with an opened file. The ftell() function returns the current value of the file position indicator.

- The value returned by the ftell() function represents the number of bytes from the beginning of the file to the current position pointed to by the file position indictor.

- If the ftell() function fails, it returns –1L (that is, a long value of minus 1).

- C function, called rewind(), can be used to rewind the file position indicator. The prototype for the rewind() function is:

void rewind(FILE *stream);

- Here, stream is the file pointer associated with an opened file. No value is returned by rewind() function. In fact the following statement of rewind() function:

```
rewind(fptr);
```

- is equivalent to this:

```
(void) fseek(fptr, 0L, SEEK_SET);
```

- The void data type is cast to the fseek() function because the rewind() function does not return a value.

- As you learned, two C library functions scanf() and printf() can be used to read or write formatted data through the standard I/O (that is, stdin and stdout).

For C disk file I/O functions, there are two equivalent functions; fscanf() and fprintf() functions allow the programmer to specify I/O streams.

- The prototype for the fscan() function is:

```
int fscanf(FILE *stream, const char *format,…);
```

- stream is the file pointer associated with an opened file. format, which usage is the same as in the scanf() function, is a char pointer pointing to a string that contains the format specifiers. If successful, the fscanf() function returns the number of data items read. Otherwise, the function returns EOF.

- The prototype for the fprintf() function is:
int fprintf(FILE *stream, const char *format, …);

- Here, stream is the file pointer associated with an opened file. format, whose usage is the same as in the printf() function, is a char pointer pointing to a string that contains the format specifiers.

- If successful, the fprintf() function returns the number of formatted expressions. Otherwise, the function returns a negative value.

File Management Functions

- It refers to dealing with existing files, not reading or writing to them, but renaming, deleting and copying them. Normally the file management functions are provided in the standard library function.

Deleting A File

- We use function remove() to delete a file. Its prototype is in stdio.h file and the prototype is as follows:

```
int remove(const char *filename);
```

- The variable filename is a pointer to the name of the file to be deleted. The specified file must not be opened. If the file exists, it is deleted and remove() return 0.

- If the file doesn't exist, if it's read only, if you don't have sufficient access rights or permission, or if some other error occurs, remove() return –1. Be careful if you remove a file, it is gone forever.
- Let try a program example.

Renaming A File

- The rename() function changes the name of an existing disk file. The function prototype, in bstdio.h, is as follows:

```
int rename(const char *oldname, const char *newname);
```

- Both names must refer to the same disk drive; you can't rename a file to a different disk drive means if the old name is in drive C:\test.txt, you can't rename it to D:\testnew.txt.

- The function rename() returns 0 on success, or –1 if an error occurs. Errors can be caused by the following conditions (among others):

1. The file oldname does not exist.
2. A file with the name newname already exists.
3. You try to rename to another disk.