Concept of Data File:

Many applications may require a large amount of data to be read, processed and also saved for later use. Such information is stored on the auxiliary memory device in the form of file. Data file is a file that we use to store, retrieve and alter information whenever necessary.

C language provides set of library functions for creating and processing data files. The prototype for all the file handling functions are declared in <stdio.h>.

Types of file:

There are two types of files.

1. Sequential File:

In this type of file, data are kept sequentially. If we want to read the last records of the file, we need to read all the records before that record. It takes more time. If we desire to access the 100^{th} records then first 99 records should be read sequentially for reading to the 100^{th} record.

2. Random Access File:

In this type of file, data can be read and modified randomly. If we want to read the last records of a file, we can read it directly. It takes less time as compared to sequential file.

In random access file, following functions are used.

- ***fseek( ):*** To set the file position.

  Syntax:

  fseek(fp,character position, seek from);

where, ***fp*** is a **file pointer** , ***character position*** can be any **long integer** from **seek from** and ***seek from*** can be **SEEK_SET, SEEK_CUR** or **SEEK_END.**

SEEK_SET : seeks from **beginning** of the **file**.

SEEK_CUR : seeks from **current position**.

SEEK_END : seeks from **end of file**.

- ***ftell( ):*** To return the current value of the file position.

  Syntax:

  ftell(fp);

where, ***fp*** is a **file pointer** .

- ***rewind( ):*** To reset the current value of the file position.

  Syntax:

  rewind(fp);

where, ***fp*** is a **file pointer** .

File operations:

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

1. Creation of a new file.

2. Opening an existing file.

3. Reading from a file.

4. Writing to a file.

5. Renaming a file.

6. Deleting a file.

7. Closing a file.

1. Creation of a new file:

Creation of a **new file** creates link between the **file** and **operating system**, where we have to mention **data file name** and **mode** of operation. The link between **file** and **operating system** is controlled by a structure called **FILE**. So, it is necessary to declare **FILE** before creating **data file**.

Syntax:

FILE *fp;

where, ***fp*** is a pointer variable which contains the address of ***FILE*** structure.

2. Opening an existing file.

After creation of a **new file**, we need to **open** the **file** either for **reading** or **writing**.

Syntax:

FILE *fp;

fp=fopen("data file name", "mode");

where, ***fp*** is a ***file pointer, fopen( )*** is a function for opening a **new file** or **existing file ,** ***data file name*** is the **name of the file** and ***mode*** defines the **purpose** of opening the data file.

Different modes of opening a data file:

Mode	Meaning
r	**Used to *open file* for *reading* provided that the file must exist to read it.**
w	**Used for *opening the file* for *writing.* If the *file exists,* its contents will be *erased* and if the file *doesn't exist* then it will be *created.***
a	**Used for *opening the file* to *append*(to add content to the *end of file*) if the *file exist* but if the file *doesn't exist,* the *new file* will be created and *pointer* will be at *first* i.e. 0 location for putting the *content* from the *beginning of the file.***
r+	**Used for *opening the file* for both *reading* and *writing.* The *file* must already *exist* before performing operation.**
w+	**Used for *opening file* for both *writing* and *reading.* If the *file exists,* its contents are *erased* and if the file *doesn't exist,* then it will be *created*.**
a+	**Used for *opening file* for both *reading* and *appending* if the *file exists,* but if the file *doesn't exist,* then *new file* will be created and *pointer* will be at *first* i.e. 0 location for putting the *content* from the *beginning of the file.***

3. Reading from a file.

To read the content from a data file, we use the following functions.

- ***fgetc( ):*** To read character from a data file.

  Syntax:

  fgetc(fp);

  where, ***fp*** is a file pointer.

- ***fgets( ):*** To read string from a data file.

Syntax:

fgets(str, n, fp);

where, ***str*** is the address of the string, ***n*** is the number of characters and ***fp*** is a file pointer.

- ***fgetw( ):*** To read integer value from a data file.

  Syntax:

  fgetw(fp);

  where, ***fp*** is a file pointer.

- ***fscanf( ):*** To read formatted data from a data file.

  Syntax:

  fscanf (fp,"control string", argument list);

  where, ***fp*** is a file pointer.

- ***fread( ):*** To read records or blocks from a data file.

  Syntax:

  fread (&v, sizeof(v), numv, fp);

where, ***&v*** is the address of structure, ***sizeof(v)*** is size of structure in bytes, ***numv*** is the number of structure and ***fp*** is a file pointer.

4. Writing to a file.

To write the content to a data file, we use the following functions.

- ***fputc( ):*** To write character to a data file.

  Syntax:

  fputc(c, fp);

  where, ***c*** is a character variable and ***fp*** is a file pointer.

- ***fputs( ):*** To write string to a data file.

  Syntax:

  fputs(str, fp);

  where, ***str*** is the address of the string and ***fp*** is a file pointer.

- ***fputw( ):*** To write integer value to a data file.

  Syntax:

  fputw(i, fp);

  where, ***i is an integer variable*** and ***fp*** is a file pointer.

- ***fprintf( ):*** To write formatted data to a data file.

  Syntax:

  fprintf (fp,"control string", argument list);

  where, ***fp*** is a file pointer.

- ***fwrite( ):*** To write records or blocks to a data file.

  Syntax:

  fwrite (&v, sizeof(v), numv, fp);

where, ***&v*** is the address of structure, ***sizeof(v)*** is size of structure in bytes, ***numv*** is the number of structure and ***fp*** is a file pointer.

5. Renaming a file.

To change the data filename from **old name** to **new name**, we use ***rename( )*** function.

Syntax:

rename ("old file name", "new file name");

6. Deleting a file.

To **remove** the **data file** from a **disk drive**, we use ***remove( )*** function.

Syntax:

remove ("file name");

7. Closing a file.

A **data file** must be closed at the **end** of the **program**. We use ***fclose( )*** function.

Syntax:

fclose(fp);

where, ***fp*** is a file pointer.

Programs on File Handling:

1. WAP to store std_no, name and marks of 'n' students in a data file. Display the records in appropriate format reading from the file.

```
#include<stdio.h>
#include<conio.h>
struct student
{
    char name[80];
    int std_no,marks;
};
void main( )
{
    struct student rec;
    FILE *fptr;
    int i,n;
    clrscr( );
        if((fptr=fopen("studinfo.txt","w+"))==NULL)
        {
            printf("Error opening file\n");
            exit(1);
        }
        printf("How many students ?");
        scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\n Enter records for student %d:\n",i+1);
        printf("Name :  ");
        scanf("%s",rec.name);
        printf("Student Number:  ");
scanf("%d",&rec.std_no);
```

```
        printf("Marks:  ");
        scanf("%d",&rec.marks);
        fprintf(fptr,"%s\n%d\n%d\n",rec.name,rec.std_no,rec.marks);
    }
    rewind(fptr);
    printf("\n\n Records stored in data file: \n");
    for(i=0;i<n;i++)
        {
         fscanf(fptr,"%s%d%d",rec.name,&rec.std_no,&rec.marks);
         printf("\n The records of student %d:\n",i+1);
         printf("Name: %s\n",rec.name);
         printf("Student Number: %d\n",rec.std_no);
         printf("Marks: %d\n",rec.marks);
        }
         fclose(fptr);
         getch();
}
```

2. WAP that reads successive records from the new data file and display each record on the screen in an appropriate format.

```
#include<stdio.h>
#include<conio.h>
    void main()
        {
        char name[15];
         int std_no,marks;
         FILE *fptr;
         clrscr();
        fptr=fopen("studinfo.txt","r");
        fscanf(fptr,"%s%d%d",name,&std_no,&marks);
            while (!feof(fptr))
                {
                    printf("\n Name :%s",name);
                    printf("\n Student Number :%d",std_no);
                    printf("\n Marks :%d",marks);
                     printf("\n ------------------------");
                fscanf(fptr,"%s%d%d",name,&std_no,&marks);
                }
        getch( );
}
```

3. WAP to delete and rename the data file using remove and rename command.

```
#include<stdio.h>
#include<conio.h>
void main( )
{
 char filename[15],oldname[15],newname[15];
 clrscr();
 printf("Enter file name to be removed: ");
 gets(filename);
 printf("Enter old file name :");
 gets(oldname);
 printf("Enter new file name  :");
gets(newname);
        if(remove(filename)= =0)
                printf("File %s is removed",filename);
        else
                printf("File %s cannot be removed",filename);
 if(rename(oldname,newname)= =0)
printf("File %s is renamed to %s\n",oldname,newname);
        else
         printf("File %s cannot be renamed\n",oldname);
   getch( );
        }
```

4. WAP to show data writing and reading operation to/from a data file using fwrite( ) and fread( ) function.

```
#include<stdio.h>
#include<conio.h>
void main( )
{
 FILE *fp;
 struct info
 {
        char name[50];
        int phone;
        int sal;
 }p;
char choice='y';
clrscr( );
 fp=fopen("abc.txt","w");
 while(choice=='y' || choice=='Y')
```

```
{
        printf("Enter name,phone and salary\n");
        scanf("%s%d%d",p.name,&p.phone,&p.sal);
        fwrite(&p,sizeof(p),1,fp);
        printf("Want to add more records?");
        fflush(stdin);
        scanf("%c",&choice);
      }
      fclose(fp);
       fp=fopen("abc.txt","r");
        while(fread(&p,sizeof(p),1,fp)==1)
      {
printf("\nName=%sPhone=%d Salary=%d ",p.name,p.phone,p.sal);
       }
       fclose(fp);
       getch();
}
```

Note 1: Important function used in above programs is

*feof(FILE *)*

Description

Used to determine if the end of file specified, has been reached. When a file is being read sequentially one line or one piece of data at a time (by means of a loop). This is the function you check every iteration to see if the end of the file has come.

Return values

Nonzero(true) if the end of file has been reached, zero(false) otherwise.

Note 2: Text mode and Binary mode

In **text mode** every digit or text are stored as a character and while reading the content back, the conversion is required from character to appropriate format and takes lots of space. While in **binary mode**, there is no conversion and every digit or text stored in binary format**(i.e. stored without performing translation, it means whatever format has been used to store the value in the memory, in the same format the content are stored in a file without conversion thereby while reading content no conversion is necessary).** Character I/O, String I/O and Formatted I/O use **text** mode whereas **fread( )** and **fwrite( )** use **binary** mode.

For e.g. suppose we want to store the value of **PI 3.14158** in **float**, then what would be the file size? If this value is stored in **Text** mode then file size would be **8** bytes **(including decimal and EOF**) whereas if the same content is stored in **Binary** mode, then only **4** bytes is occupied because **float** uses **4** bytes.