CP MODULE 6

Data files – formatted, unformatted and text files. Command line arguments – examples.

DATA FILES

When we run in a program, we used to give inputs through keyboard and the results appear in the computer screen. These inputs and outputs are not stored permanently in the computer but they are in memory and will be not be available after that program terminates. To store data permanently in the computer, we can use data files.

File can be used to **store the data permanently** so that they can be easily retrieved later. If data is stored in memory they are volatile and so it is not permanent. But data stored in files are permanent.

A file is a collection of bytes stored on a permanent medium like a hard drive, flash memory, or CD-ROM.

There are two types o**f data files**,

1. **stream-oriented** (or standard) data files, - easier to work with
2. **system-oriented** (or low-level)data files. - complicated- more closely related to the computer's operating system. A separate set of procedures and library functions are required to process system-oriented data files.

Stream-oriented data files can be subdivided into two categories.

1. ***Text files***
   - Consist of consecutive characters.
2. ***Unformatted data files***
   - Organizes data into blocks containing contiguous bytes of information.
   - blocks represent more complex data structures, such as arrays and structures

OPENING AND CLOSING A DATA FILE

When working with a stream-oriented data file, the first step is to **establish a buffer area**,

- information transferred between the computer's memory and the data file are temporarily stored in buffer area.
- This buffer area allows information to be read from or written to the data file rapidly.

The buffer area is established by writing

```
FILE *ptvar;
```

where FILE (uppercase letters required) is a *special structure type t*hat establishes the buffer area, and ptvar is a pointer variable that indicates the beginning of the buffer area.

- The *structure type**FILE*** is defined within a system include file, typically *stdio.h*

- Here the pointer ***ptvar*** is often referred to as a **stream pointer**, or simply a **stream**.

This declaration is needed for communication between the file and the program.

OPENING A FILE

- A data file must be opened before it can be processed.
- Opening a file
    - associates the file name with the buffer area (i.e., with the stream).
    - also specifies how the data file will be utilized,
        - i.e., as a read-only file, a write-only file, or a read write file, in which both operations are permitted.

The library function **fopen** is used to open a file. This function is typically written as

```
ptvar = fopen( file-name, file-type);
```

where file-name and file- type are strings that represent the name of the data file and the manner in which the data file will be utilized.

The file- type must be one of the following:-

"r"	Open an **existing file** for reading only.
"r+"	Open an **existing file** for both*reading and writing.*
"w"	Open a new file for writing only. If a file with the specified file-name currently exists, it will be destroyed and a new file created in its place.
"w+"	Open a new file for both*reading and writing.* If a file with the specified file-name currently exists, it will be destroyed and a new file created in its place.
"a"	Open an **existing file** for appending (i.e., for adding new information at the end of the file). A new file will be created if the file with the specified file-name does not exist.
"a+"	Open an **existing file** for both *reading and appending*. A new file will be created if the file with the specified file-name does not exist.

- The fopen function returns a pointer to the beginning of the buffer area associated with the file.
- A NULL value is returned if the file *cannot be opened* as, for example, when an existing data file cannot be found.

CLOSING A FILE

Finally, a data file must be closed at the end of the program.
This can be accomplished with the library function **fclose**. The syntax is simply

```
fclose (ptvar);
```

- It is good programming practice to close a data file explicitly using the fclose function,
- Most C compilers will automatically close a data file at the end of program execution if a call to fclose is not present.

Simple file program in C can contain the following statements.

```
#include <stdio.h>
FILE * fpt ;
fpt = fopen("sample.dat", " w " ) ;
. . . . …..
………..
fclose(fpt) ;
```

- The first statement **#include <stdio.h>** causes the header file stdio.h to be included in the program.
- The second statement **FILE * fpt ;** defines a pointer called fpt which will point to a structure of type FILE, indicating the beginning of the data-file buffer area.FILE is defined in stdio. h.
- The third statement **fpt = fopen("sample.dat", " w ") ;** opens a new data file called sample.dat as a write-only file. The fopen function returns a pointer to the beginning of the buffer area and assigns it to the pointer variable fpt.
    - Thus, fpt points to the buffer area associated with the data file sample. Dat.
- All subsequent file processing statements (not shown in this example) will access the data file via the pointer variable fpt rather than by the file name.
- Finally, the last statement closes the data file. The argument is the pointer variable fpt, not the file name sample. Dat.

C Program to check whether a file is there(existing) or not

```
#include <stdio.h>
#define NULL 0
main( )
{
FILE * fpt ;
fpt = fopen("sample.dat", “r" ) ;

if(fpt == NULL)
{
        printf("\nERROR - Cannot open the f i l e \ n " ) ;
}
else
{
        printf("\nFile is existing\n " ) ;
        fclose(fpt) ;
}
}
```

The following code attempts to **open an existing data file called sample.dat for both reading and writing**. An error message will be generated if this data file cannot be found. Otherwise the data file will be opened and processed, as indicated.
The fopen and the if statements are often combined, as follows:-

```
if( ( fpt = fopen(“sample.dat”,"r") )= = NULL)
{
printf("\nERROR - Cannot open the designated f i l e \n " ) ;
}
```

CREATING A FILE
A data file must be created before it can be processed.
A stream-oriented data file-TEXT FILE can be created in two ways.

1. Create the file directly, using a text editor or a word processor.
2. Write a program that enters information into the computer and then writes it out to the data file.

UNFORMATTED data file can only be created

- with such specially written programs.

When creating a new data file with a specially written program,

- the usual approach is to enter the information from the keyboard
- and then write it out to the data file.

To write characters to a file through a program

- If the data file consists of individual characters, the library functions **getchar** can be used to enter the data from the keyboard and and **putc** to write it out to the data file.

To read characters from the data file and display its contents.

- the library function **getc** will read the individual characters from the data file, and **putchar** will display them on the screen.

Q. Enter a line of text through keyboard and store it in a file and display(use library functions getchar,putchar, putc getc)

```
#include<stdio.h>
# define NULL 0;
main()
{
        char c=' ';
        FILE *f;

f=fopen("sample.txt","w");
        printf("\nEnter a line to store in file and press enter key\n");
        while(c!='\n')
        {
        c=getchar();
        putc(c,f);
        }
fclose(f);

c=' ';
printf("\nDISPLAYING CONTENTS IN FILE\n");
f=fopen("sample.txt","r");
if(f==NULL)
{
printf("\n ERROR File does not exist");
}
else
{
         while(c!='\n')
         {
         c=getc(f);
         putchar(c);
         }
fclose(f);

}
}
```

Q. Enter a line through keyboard and convert it into UPPERCASE and store in sample.txt

STORE AND DISPLAY A LINE OF TEXT IN FILE

```
#include<stdio.h>

main()
{
        char c=' ';
        FILE *f;

f=fopen("sample.txt","w");
        printf("\nEnter a line to store in file and press enter key\n");
        while(c!='\n')
        {
        c=getchar();
        putc(toupper(c),f);
        }

fclose(f);

c=' ';
printf("\n CONTENTS IN FILE\n");
f=fopen("sample.txt","r");
if(f==NULL)
{
printf("\n ERROR File does not exist");
}
else
{
         while(c!='\n')
         {
         c=getc(f);
         putchar(c);
         }

fclose(f);
}
}
```

Data files consisting entirely of stringscan be created and read more easily with programs.

- Some commonly used functions of this type are **gets, puts, fgets and fputs.**
  - The functions **gets** read strings from the standard input devices
  - **puts** write strings to the standard output devices
  - **fgets**retrieve(read) strings from **file**
  - **fputs** writes string to **file**

fgets and fputs

The fputs() function writes a line of characters into file. It outputs string to a stream
fputs(const char *s, FILE *stream)
E.g.

fputs("This is a system programming language.", fp);

The fgets() function reads a line of characters from file

char* fgets(char *s, size , FILE *stream)

- char *fgets(char *str, int n, FILE *stream) reads a line from the specified stream and stores it into the string pointed to by str. It ***stops*** when either (n-1) characters are read, the newline character is read, or the end-of-file is reached, whichever comes first.

WRITE and READ A STRING IN A FILE(using fputs fgets)

```
#include<stdio.h>
#include<ctype.h>
main()
{
        FILE *f;
        char s[100],d[100];
        f=fopen("sample.txt","w");
        printf("\nEnter a line to store in file and press enter key\n");
        gets(s);
        fputs(s,f) ;
fclose(f);

printf("\nCONTENTS IN FILE\n");
f=fopen("sample.txt","r");
        fgets(d,1000,f) ;
        printf("\nString is %s",d);
fclose(f);}
```

- Many data files consist of <u>complex data structures, such as **structures**</u>**.**
  - Such data files can be processed using the library functions **fscanf** and **fprintf.**
  - **fscanf** function helps to <u>**read formatted data** from a data file</u> associated with a particular stream,
  - **fprintf** permits helps to <u>**write formatted data t**o the data file</u>.
- The fprintf() function is used to write mixed type data items into a file.
  - Its syntax is

    fprintf(fp,format-string,var-list);

  where fp is the file pointer, format-string is the list of format specifiers and var-list is the list of variables whose values are to be written to file.

- The fscanf() function is used to read mixed type data from the file.
  - The syntax is similar tothat of fprintf():

    fscanf(fp,format-string,addr-list);

<u>Using fscanf and fprintf to manipulate file</u>

```
#include<stdio.h>
struct student
{
char name[15];
int roll;
int mark;
}s;
main()
{

        FILE *f;
        char nam[100];
        int r,m;
        clrscr();

f=fopen("sample.txt","w");
        printf("\nEnter name and rollnumber and mark\n");
        scanf(" %[^\n]%d%d",s.name,&s.roll,&s.mark);

        fprintf(f,"\n%d\n",s.roll);
        fprintf(f,"\n%d\n",s.mark);
        fprintf(f,"\n%s\n",s.name);
fclose(f);
printf("\nCONTENTS IN FILE\n");
f=fopen("sample.txt","r");

        fscanf(f,"%d",&r);
        fscanf(f,"%d",&m);
        fscanf(f," %[^\n]",nam);

        printf("\nRoll is %d",r);
        printf("\nMArk is %d",m);
        printf("\nNAme is %s",nam);

fclose(f);
}
```

Reading and writing integers

- The **putw**() function is used to write integers to the file. The syntax is

putw(num, fp);

The putw() function takes two arguments, first is an integer value num to be written to the file and second, fp is the corresponding file pointer.

- The **getw**() is used to read integers from a file. The syntax is

n=getw(fp);

The getw() function takes the file pointer as argument from where the integer value will be read and returns the read value n. It returns EOF if it has reached the end of file.

Q. **The program to inputs numbers from the user and writes them to a file. Later the numbers are printed by reading from the file**.

```
#include<stdio.h>
main()
{
FILE *fp;
int num,count,i;
printf("How many numbers?");
scanf("%d",&count);
fp=fopen("file.txt","w");
if(fp==NULL)
printf("Sorry! Error in opening file\n");
else
{
for(i=0;i<count;i++)
{
```

```
printf("Enter any number:\n");
scanf("%d",&num);
putw(num,fp);
}
fclose(fp);
}
fp=fopen("file.txt","r");
if(fp==NULL)
printf("Sorry! Error in opening file\n");
else
{
num=getw(fp);
while(num!=EOF)
{
printf("%d ",num);
num=getw(fp);
}
fclose(fp);
}
}
```

UNFORMATTED DATA FILE

Unformatted files are *created using special programs*. They are *binary files*. They are *used to store blocks of data*.

When processing large amounts of data, it is often easier to group information together, instead of dealing with lots of individual variables.

- Some applications involve the use of data files to **store blocks of data**,
    - each block consists of a fixed number of contiguous bytes.
    - Each block will generally represent a complex data structure, such as a **structure** or an **array**.
        - For example, a data file may consist of multiple structures having the same composition, or it may contain multiple arrays of the same type and size.
- The library functions fread and fwrite are intended to be used in situations of this type.
- For example, in the case of structure variables, it is good to read/write the structure data at a single step.
    - To read and write data that are longer than 1 byte, the C language provides the functions: **fread()** and **fwrite().**
    - fread() and fwrite() functions are called**unformatted read** and **write functions.**
    - Similarly, data files of this type are often referred to as **unformatted data files.**

- fwrite and fread require four arguments:
    - a pointer to the data block,
    - the size of the data block,
    - the number of data blocks being transferred, and
    - the stream pointer.

The syntax of fwrite() is
fwrite(address,size,count,fptr)
where

- address denotes the address of the data block (structure variable) whose contents are to be written to file.
- size denotes the size of the block .
- count denotes the number of data blocks (number of structure variables) that you need to write.
- fptr represents the file pointer associated with the file

fread() is used to read blocks of data from a file.
The syntax of fread() is same as that of fwrite():
fread(address,size,count,fptr)

E.g. fwrite function might be written as
fwrite(&customer, sizeof(record), 1, fpt) ;
where customer is a structure variable of type record, and fpt is the stream pointer associated with a data file that has been opened for output.
E.g.
/* create an unformatted data f i l e containing customer records */
Inputs the details of a bank customer and writes to the file and display.

```
#include<stdio.h>
typedef struct
{
char name[15];
int accno;
float balance;
}account;

main()
{
account acc;
FILE *fp;
fp=fopen("struct.txt","w");
if(fp==NULL)
{
printf("Sorry! Error in opening file\n");
}
else
{
printf("Enter customer name\n");
scanf(" %s",acc.name);
printf("Enter account number\n");
scanf("%d",&acc.accno);
printf("Enter balance\n");
scanf("%f",&acc.balance);
fwrite (&acc, sizeof(account), 1, fp);
        /*count is 1 because you are writing the contents of only 1 struct variable*/
fclose(fp);
}

fp=fopen("struct.txt","r");
if(fp==NULL)
printf("Sorry! Error in opening file\n");
else
{
fread(&acc, sizeof(account), 1, fp);
printf("Customer name: %s\n",acc.name);
printf("Account number: %d\n",acc.accNo);
printf("Balance: %f\n",acc.balance);
fclose(fp);
}
}
```

EXAMPLE Inputs the details of a student and writes them to a file. The details are then printed by reading from the file.(without using structure)

```
#include<stdio.h>
main()
{
int m1,m2,roll,n,i;
char name[15];
FILE *fp;
fp=fopen("marks.txt","w");
if(fp==NULL)
printf("Sorry! Error in opening file\n");
else
{
printf("Number of students please\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter the details of student %d\n",i+1);
scanf(" %s %d %d %d",name,&roll,&m1,&m2);

fprintf(fp,"%s %d %d %d",name,roll,m1,m2);
}
fclose(fp);
}
fp=fopen("marks.txt","r");
if(fp==NULL)
printf("Sorry! Error in opening file\n");
else
{
for(i=0;i<n;i++)
{
printf("Details of student %d\n",i+1);
fscanf(fp,"%s %d %d %d",name,&roll,&m1,&m2);
printf("Name: %s\n",name);
printf("Roll: %d\n",roll);
printf("Marks: %d and %d\n",m1,m2);
}
fclose(fp);
}}
```

Random access to files

If you want to access specific locations in a file, C supports three functions for random access to files.

- **ftell()**

This function tells the current position of the file pointer in the file. It takes a file pointer as argument and returns the byte location of current position of the pointer. The statement

len = ftell(fp);

means that the file pointer is currently at byte numbered len. i.e., bytes 0 to len - 1 are to the left of the current pointer position.

- **rewind()**

The rewind() function resets the file pointer to the beginning of the file with the pointer specified as its argument. That is, it ‘‘rewinds” the file. Its syntax is:
rewind(fp);
where fp is a file pointer. Thus, after executing the statements,
rewind(fp);
n=ftell(fp);
the value of n will be zero (indicating the beginning of file).

- **fseek()**

fseek() is used to move the file pointer to a desired location in the file. Its syntax is:
fseek(fp,offset,position);
where
• fp is the file pointer.
• offset is the number of bytes fp is to be moved.
• position denotes the position from which the file pointer is moved.
position can take three values:
• 0 – beginning of file
• 1 – current position
• 2 – end of the file
position can also be specified in terms of constants rather than in terms of integers as follows:
• SEEK SET – beginning of file
• SEEK CUR – current position
• SEEK END – end of the file
offset can be positive or negative. If it is positive, the file pointer is moved in forward direction. If it is negative, file pointer is moved backwards.

<u>fseek() function and their interpretations.</u>
fseek(fp,0,0) – Moves fp zero bytes from the beginning of the file. In other words, it moves the file pointer to start of the file.

fseek(fp,0,SEEK CUR) – Moves fp zero bytes from current position. In other words, the pointer stays in its current position.

fseek(fp,0,SEEK END) – Moves fp to the file end.
fseek(fp,m,SEEK SET) – Moves fp m bytes from the beginning - sets the pointer at (m + 1) th byte in the file.

fseek(fp,m,1) – Moves fp forward by m bytes from current position.
fseek(fp,−m,1) – Moves fp backward by m bytes from current position.
fseek(fp,−m,2) – Moves fp backward by m bytes from end of the file.

COMMAND LINE ARGUMENTS

Usually we write main function as main(). (main with empty parentheses)
But the parentheses of main function may contain special arguments that help to pass parameters from the operating system to main function.
C permit two such arguments,

- **argc**
- **argv**

The first argument, **argc**,

- must be an <u>integer variable</u>,
- The value of argc will indicate the <u>*number of parameters passed*</u>.
- The argc parameter holds the number of arguments on the command line and is an integer.
    - If we want to execute a program in Linux we write **./a.out**
        - Here argc is 1 because only ./a.out is there
    - agc is always at least 1 because the ./a.out command itself is the first argument.

The second argument, **argv**, is an <u>array of pointers to characters</u>; i.e., an ***array of strings***.
If we execute using **./a.out the value of rgv[0] is ./a.out**
Each string in this array will represent a parameter that is passed to main.

```
void main(int argc, char *argv[])
{ . . . . .
}

main(int argc, char *argv[])
{
……………………….
}
```

To execute the program in Linux which contains main function with arguments we can type
./a.out parameter1   parameter2 ……parametern
The individual items(parameters) must be separated from one another either by blank spaces or by tabs.

- The <u>*program name*</u> will be stored as the first item in argv that is **argv[0]**
    - In Linux argv[0] will be **./a.out**
- if n parameters are there after **./a.out**, there will be (n+ 1) entries in argv, ranging from argv[0] to argv[n].

- **argc** will automatically be assigned the value (n + 1).(total number of parameters including program name or ./a.out
  E.g Suppose the we are executing show.c
  ./a.out hello hai ok
  Here argv[0]is ./a.out
  argv[1] is hello
  argv[2] is hai
  argv[3] is ok
  Number of entries in argv is 4(argv[0] to argv[3]). So **argc** is 4.

E.g Suppose the name of the following program is ***sample.c***

```
#include<stdio.h>
main(int argc,char *argv[])
{
printf("Hello %s",argv[1]);
}
```

When we run the above code as follows:
./a.out Anu
the output will be
Hello Anu

E.g.

```
#include<stdio.h>
#include<stdlib.h>
main(int argc,char *argv[])
{
int i;
printf("argc is %d\n",argc);
for(i=0;i<argc;i++)
{printf("argv[%d] is %s\n",i,argv[i]);}
}
```

If you run the above code as:
./a.out CSE IT
the output is

```
argc is 3
argv[0] is ./a.out
argv[1] is CSE
argv[2] is IT
```

E.g. To receive name, roll number and age of a student as command line arguments and then print them.

```
#include<stdio.h>
#include<stdlib.h>
main(int argc,char *argv[])
{
printf("STUDENT DETAILS\n");
if(argc<4)
{printf("Insufficient number of arguments\n");}
else
{
printf("Name:%s\n",argv[1]);
printf("Roll:%s\n",argv[2]);
printf("Age:%s\n",argv[3]);
}
}
```

We execute this program linke

```
./a.out Anu 11 20
```

Output is

```
Name:Anu
Roll:11
Age:20
```

E.g To add two numbers received as command line arguments.

```
#include<stdio.h>
#include<stdlib.h>
main(int argc,char *argv[])
{
int a,b;
a=atoi(argv[1]);
b=atoi(argv[2]);
printf("The sum is %d",a+b);
}
```

```
./a.out 2 3
```

Output is

```
The sum is 5
```

- Since the numbers are received as command line arguments, they will be string constants. To convert them to integers, the function **atoi()** is used.

E.g.

Read a line of text from a data file and display it on the screen. Enter file name as a command line parameter

```
#include <stdio.h>
#define NULL 0
main(int argc, char *argv[])
{
FILE * fpt ;
char c;

i f ( ( fpt = fopen(argv[l], "r" ) ) == NULL)
{
printf("\nERROR - Cannot open the designated f i l e \ n " ) ;
}
else
{
        c=getc(fpt);
        while (c!='\n')
        {
        putchar(c) ;
        c = getc(fpt);
        }
fclose(fpt ) ;
}
```

To execute this program we can type

```
./a.out sample.txt
```