F.Y.B.C.A. (Science) Sem-II
Advanced C Programming

Unit 6
File Handling

Mrs. Madhuri Abhijit Darekar
Assistant Professor
MIT Arts, Commerce and Science College, Alandi (D)

Introduction

- A program always operates on input data. Usually input is given to the program from standard input devices like keyboard. Where input and output functions like printf() and scanf() are used.
- Now a day many applications require a large amount of input data. If this data we accept through the keyboard, it will take a lot of time and when the program ends, the data get lost. So we need different form of data.
- **A file stores data in a permanent form** so it can be used whenever required which will solve our problem.
- Files can be used to **provide input data to a 'C' program**. The **output of the program can also be stored in a file.**

Streams

- A **stream** is a **sequence of bytes of data** which a program can read from or write to.
- All C **input/output is done with streams**.
- Stream has two types :
- **Input Stream** : A sequence of bytes flowing into a program.
- **Output Stream**: A sequence of bytes flowing out of program is an output stream

Predefined streams

- There are 5 **predefined streams**, which are **automatically opened** when a C program starts executing **and are closed** when the program terminates.
- **Types of predefined streams**

Name	Stream	Device
i. stdin	Standard input	Keyboard
ii. stdout	Standard output	Screen
iii. stderr	Standard error	Screen
iv. stdprn	Standard Printer	Printer (LPT1)
v. stdaux	Standard auxiliary	Serial Port (COM1)

Files

- **Definition:** A **file** represents a **sequence of bytes on the disk** where a group of related data is stored. File is **created for permanent storage of data.** It is a ready made structure. In **C** language, we use a **structure pointer of FILE type to declare a file**.
- C programming language **provides** access on high level **functions to handle file** on your storage devices.

Types of Files

- Streams are of two types - text and binary. Either type of stream can be associated with a file.
- **Types of Files**
- I. **Text File**

  In a text file, all **data is stored as a sequence of characters**.
- ii. **Binary File**

  In a binary file, **the data is stored in the file as it is stored in memory** i.e. a character will be stored in 1 byte, an integer in 2 bytes, float in 4 bytes, etc.
- **Example : 876A**
- In the text file, the 876 is stored as three characters. The ASCII value of character '8' is 56, 7 is 55 and '6' is 54. The ASCII value of 'A' is 65.
- In the binary file, 876 is stored as an integer in the binary form i.e. 0000001101101100 and ‘A’ is stored as 65.

Operations on a File

- C provides various **functions to perform operations on files.** These operations include:
  1. Open a file
  2. Read data from a file
  3. Write data to a file
  4. Close a file
  5. Detect end-of-file
- **To perform any operation** using files in 'C', a **pointer must be used**. This pointer is called a **file pointer.**

The File Pointer

- A **file pointer** is a pointer variable of type **FILE** which is **defined in stdio.h**. The type FILE defines various properties about the file including its name, status and current position.
- Basically, a **file pointer identifies a specific disk file.** This **pointer is used to specify where to perform the operations in the file**
- **Example**

  FILE *fp:

  Here, **fp is declared as a pointer to a file**. All operations on the file will be performed using the pointer.

Opening a File

- **Before performing any file operation, the file must be opened.** The process of opening a stream for use and linking a disk file to it, is called **opening a file**.
- The fopen( ) function is used to open a file which requires two arguments:
- 1. file name to be opened.
- 2. mode in which it should be opened.
- **If the open operation is successful, fopen returns a pointer to type FILE. If it fails, it returns NULL**.
- **Example:**

  FILE *fp;

  fp=fopen(“a.txt”, “r”);

File opening mode for Text File

A file can be opened in different modes. Most commonly used modes for opening a file.

r : opens a text file in reading mode.

w : opens or creates a text file in writing mode.

a : opens a text file in append mode.

r+ : opens a text file in both reading and writing mode. The file must exist.

w+ : opens a text file in both reading and writing mode. If the file exists, it's truncated first before overwriting. Any old data will be lost. If the file doesn't exist, a new file will be created.

a+ : opens a text file in both reading and appending mode. New data is appended at the end of the file and does not overwrite the existing content.

File opening mode for Binary File

rb : opens a binary file in reading mode.

wb : opens or creates a binary file in writing mode.

ab : opens a binary file in append mode.

rb+ : opens a binary file in both reading and writing mode, and the original content is overwritten if the file exists.

wb+ : opens a binary file in both reading and writing mode and works similar to the w+ mode for binary files. The file content is deleted first and then new content is added.

ab+ : opens a binary file in both reading and appending mode and appends data at the end of the file without overwriting the existing content.

Closing a File

- After the operations on the file have been performed, the file should be closed using the **fclose()** function.
- It ensures that all **information associated with the file is flushed out of buffers**.
- Prevents accidental misuse of the file.
- It is necessary to **close a file before it can be reopened in a different mode**.
- **Example :**

  fclose(fp); //to close single file

  fcloseall(); //to close all open files

Detecting End of File

- If we know exactly how long a file is, there is no need to detect the end of file. if we don't know how big the file is it is necessary to detect end of file.

  This can be done in two ways.
- In **text files**, a special **character EOF**(defined in stdio.h having value -1) denotes the end of file. As soon as this character is encountered, the end-of-file can be detected.
- In **binary files**, the EOF is not there. Instead we can use the library **function feof()** which returns TRUE if end of file is reached. **It can be used for text file also**.

Reading and Writing File data

File I/O operations can be performed in three ways:

1. Character/string input output **to read/write characters or line of characters** are commonly **used with text files only.**

   Functions: fgetc(), fputc(), fgets(), fputs(), getw(), putw()

2. Formatted I/O to **read/write formatted data**. This can **only be used with text mode files**.

   Functions: fprintf(), fscanf()

3. Direct input/output to **read /write blocks of data directly**. This method is **used only for binary files.**

   Functions: fread(), fwrite()

Reading and Writing Characters

1. getc() and fgetc()

Both are used to **read a single character from a file or stream.** The difference is that **getc is a macro while fgetc is a function**

They return a single character. When used with the stream stdin, they input a character from the keyboard.

Example:

ch=fgetc (fp);

ch=getc(stdin);

ch=getc(fp);

Reading and Writing Characters

2. putc() and fputc()

Both are used to **write a single character to the specified stream.** If the stream is stdout, they display it on the standard output device

Example:

```
char ch=' A';
putc (ch, fp);
putc(ch, stdout);
fputc (ch, fp);
```

Reading and Writing Characters

3. fgets()

It **reads a line of characters** from a file.

Syntax:

char *fgets(char *str, int n, FILE *fp);

str points to the string where the read string has to be stored.

n is the maximum number of characters to be read.

fp is a file pointer from which we want to read a line of characters.

Example:

```
char str[80];
fgets(str,80,fp);
```

Reading and Writing Characters

4. fputs()

Writes a line of characters to a stream. It does not add a new-line to the end of the string. If it is required, then the new line has to be explicitly put.

Syntax:

char fputs(char * str, FILE *fp);

Example:

char name[]="ABCD";

fputs(name,fp);

C program to open a text file and count number of characters, words and lines in file.

```
#include<stdio.h>
void main()
{
    FILE *fp;
    char fname[30],ch;
    int ccnt=0,wcnt=0,lcnt=0;
    printf("\nEnter file name:");
    scanf("%s",fname);
    fp=fopen(fname,"r");
    while((ch=fgetc(fp))!=EOF)
    {
      ccnt++;
      if(ch=='\n')
            lcnt++;
      if(ch==' '||ch=='\0'||ch=='\n')
            wcnt++;
    }
    printf("\nTotal number of characters = %d \nTotal number of words = %d \nTotal
    number of lines = %d",ccnt, wcnt, lcnt);
    fclose(fp);
}
```

Formatted File Input / Output Functions

- Let us now see how these functions can be used to perform various operations on text files.
- The **functions seen above can handle only single data types. In order to deal with multiple data types, formatted file I/O functions are used.** fprintf () is used for output and fscanf() is used for input.

1. fprintf() :

- This is similar to printf() except that a pointer to a file must be a specified. The data is written to the file associated with the pointer.
- **Syntax:**

  int fprintf (FILE *fp, char *format, argument list);

  The format string is the same as used printf(). The argument list are the names of variables to be output to the specified stream.
- **Example:**

  fprintf (fp, "%s %d %f", name, age, salary);

Formatted File Input / Output Functions

2. fscanf() :

- This is similar to scanf() except that input comes from a specified stream instead of stdin.
- **Syntax:**

  int fscanf (FILE *fp, char *format, address list);

  The format string is the same as used for scanf(). The address list contains the addresses of the variables where fscanf() is to assign the values.
- **Example:**

  fscanf (fp,“%s%d%f”, &name, &age, &salary);

C program to store n Student Information such as (Rollno,Name,Percentage) in file and display the same

```
#include<stdio.h>
void main()
{
    char name[50],c;
    int rno,per,i,n,j;
    FILE *fp;
    printf("\n Enter Number of Students : ");
    scanf("%d",&n);
    fp=fopen("student.txt","w");
    if(fp==NULL)
    {
     printf("\n ERROR! ");
     exit(1);
    }
```

```
    printf("\n For %d Students : ",n);
    for(i=0;i<n;i++)
    {      printf("\n Enter Name : ");
     scanf("%s",name);
     printf("\n Enter Roll No and Percentage : ");
     scanf("%d %d",&rno,&per);
     fprintf(fp,"\n Name : %s \n Roll No : %d \n Percentage : %d
    \n",name,rno,per);
    }
    fclose(fp);
    fp=fopen("student.txt","r");
    c=fgetc(fp);
    while (c != EOF)
    {      printf("%c",c);
     c=fgetc(fp);
    }
    fclose(fp);
}
```

Direct File Input /Output

This is **used with binary-mode files**. It is used to read or write blocks of data.

1. **fwrite() :**

   This function writes a block of data from memory to a binary file. **It returns the number of elements written.**

   int fwrite(void *buf, int size, int count, FILE *fp);

   - buf is a pointer to the region of memory which **holds the data to be written to the file**.
   - size specifies the **size in bytes of individual data items.**
   - count specifies the **number of items to be written**.
   - fp is a **file pointer**

Direct File Input /Output

2. fread():

It **reads a block of data from binary file** and assigns it to the specified memory address. It returns the number of values read.

int fread(void *buf, int size, int count, FILE *fp);

- buf is the pointer to which memory **receives data read from the file**(i.e., it is the address of the variable).
- size specifies the **size in bytes of individual data items being read**.
- count specifies the **number of items to be read**.
- fp is a **file pointer**.

Example

fread(&num, sizeof(int),1, fp);

This **reads an integer from the file and assigns it to num**. If we have a **structure variable emp** and its members have to be read from the file, fread can be used.

fread (&emp, sizeof (emp), 1, fp);

C Program to store n employee information such as (eno, ename, salary) in file and display same.

```
#include<stdio.h>
struct employee
{
    int eno;
    char ename[10];
    int sal;
}emp[10],emp1[10];
void main()
{
    int  i,n;
    FILE *fp1;
    printf("\nEnter how many employee :");
    scanf("%d",&n);
    fp1=fopen("emp.txt","w");
```

```
for(i=0;i<n;i++)
    {     printf("\nEnter the employee no.  : ");
     scanf("%d",&emp[i].eno);
     printf("\nEnter the name :");
     scanf("%s",emp[i].ename);
     printf("\nEnter the salary :");
     scanf("%d",&emp[i].sal);
     fwrite(&emp[i],sizeof(emp[i]),1,fp1);
    }
    fclose(fp1);
```

```
fp1=fopen("emp.txt","r");

    for(i=0;i<n;i++)
    {     fread(&emp1[i],sizeof(emp1[i]),1,fp1);
     printf("\nEmployee no.  : ");
     printf("%d",emp1[i].eno);
     printf("\nEmployee name :");
     printf("%s",emp1[i].ename);
     printf("\nEmployee salary :");
     printf("%d",emp1[i].sal);
    }
    fclose(fp1);
}
```

Other Functions

1. **fflush():**

   This causes the buffer associated with an open output stream to be written to the specified file. If it is an input stream, **buffer contents cleared**.

   fflush(stdin);

2. **remove():**

   This **deletes the file specified**. If it is open, be sure to close it before removing it.

   remove(“a.txt”);

3. **rename():**

   This function **changes the name of an existing disk file**.

   rename(“a.txt”, ”a_new.txt”);

Random Access to Files

- Every **open file has a position pointer** or a position indicator associated with it. This **indicates the position where read and write operation takes place.**
- C provides functions to control the position pointer by means of which data can be read from or written to any position in the file. These functions are:

1. ftell():

This function is **used to determine the current location of the file pointer.**

long ftell(FILE *fp);

It **returns a long integer that gives the current pointer position** in bytes from the start of the file. The beginning of the file is considered at position 0.

Random Access to Files

Example:

fp=fopen(“a.txt”,”r”); printf(“%ld”, ftell(fp));

Here the **output will be 0,** because when a file is opened, the pointer points to the beginning of the file .

fp=fopen(“a.txt”,”a”); printf(“%ld”, ftell(fp));

Here the **output will be the number of bytes in the file** because when a file is opened in the append mode, the pointer points to the end of the file.

2. rewind():

This function **sets the pointer to the beginning of the file**. This function can be used **if we have read some data from a file and want to start reading from the beginning of the file**, **without closing and reopening the file.**

- **Example:**

rewind(fp);

printf(“%ld”, ftell(fp));

This will yield 0 since rewind positions the pointer at the start of the file:

Random Access to Files

3. fseek() :

The function fseek **allows the pointer to be set to any position in the file**.

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

offset indicates the distance in bytes that the **position pointer has to be moved by**.

origin indicates the **reference point in the file** with respect to which the pointer is moved offset number of bytes.

Constant	**Value**	**Moves the pointer offset bytes from**
SEEK_SET	0	beginning of file
SEEK_CUR	1	its current position
SEEK_END	2	the end of the file.

- **Example:**

1. fseek (fp, 0,SEEK_END);

This positions the pointer to the end of the file.

2. fseek (fp, 0,SEEK_END);

printf("%ld", ftell(fp));

This will display the size of the file in bytes.