9. File Handling

The input /output function printf() scanf() getchar(),puthar() are known as console oriented i/o functions which always use keyboard is input device and monitor are out devices . While using these library functions the entire data is lost when either program is terminated or the computer is turned off. Again it becomes bored and time consuming to handle large volume of data through keyboard. It takes a lot of time to enter entire data. These problems of data files in which data can be stored memory than read/write whenever necessary, without destroying data.

A file is place on disk where a group of related data are stored. The data file allows us to store information permanently and to access and alter these information when necessary. Programming language C has various library functions for creating and processing data files. Mainly, there are two types of data files one is stream oriented high-level and low level data files. In high-level data files functions do their own buffer management where as the programmer should do it explicitly in the case of system oriented level files.

The standard data files are again subdivided into text files and binary files. The test files consists consecutive characters by characters be inter operated as individual data item. The binary file origination data into blocks containing continues byte of information. For each binary and text files, there are a number of Formatted and Unformatted library functions.

9.1. Opening and Closing a Files

Before a program write to a file or read form a file, the program must be open. Opening a file establishes a link between program and operating system. This provides the operating system the name of the file and mode in which file is to be opened. While working with high level data file we need buffer are where information is stored temporarily in course of transferring data between computer memory to data file. The buffer area is established as: FILE * ptr variable;

Here, FILE is special structure declared in header stdio.h . Each file we opened as its own file is special structure that contains information about file being used. Such as its current size, its location in memory etc. The ptr variable is a pointer type. That stores beginning address and buffer area allocated after a file opened. Thus pointer contains all the information about file and its use and communication link between system and the program. A data file is opened as:

ptr variable=open (filename, file mode);

The associate file name with buffer area and specifics how data files will be open in specified mode. The function open () returns a pointer to beginning of buffer area associated with file. A file NULL value is returned if file cannot be opened due to some reasons. After opening a file we can processes data files as required. Finally data file must be closed. The closing a file ensures that all outstanding information associated with file is flushed out form buffer and links to file are broken. It also prevent any accidental misuse of file. The file is closed using other library functions fclose() as:

fclose(ptr variable);

9.1.1 File Opening Modes

The file opening mode specifies way in which a file should be opened. In other words, it specifies the purpose of openings a file.

a) "r" read mode: This opens existing files for reading mode only. It searches specified file. If file exists it loads it into memory and sets up a pointer to first character in it. If file does not exist, it returns NULL. The possible operation reading from the file only.

b) "w" write mode: This mode of file opens a file for writing only. It searches specified file, If file already exist, the contents are overwritten. If file does not exist new file is crated. It returns NULL, if it is unable to open file in write mode. The possible operation is only writing to file.

c) "a" append mode: It opens an existing file for appending. It searches specified file, If specified exists it, loads into memory and setup a pointer that points last character in it. If file does not exist, new file is created. It returns NULL. if unable to open file. The possible operation is adding new content at end of file.

d) "r+" read+write mode: It opens existing file for reading and writing. It searches specified file. If file exists it loads into memory and a pointer first character. It will returns NULl, if the file does not pointer exist. The possible operations are reading existing contents writing contents, modifying existing contents can be done in this mode.

e) "w+" read+write mode: It opens file for reading and writing. If specified file exists. It contents are destroyed. If file does not exist t new file is created. It returns NULL. If unable to open file. The possible operations are writing new contents, reading them back and modifying contents of the file.

f) "a+" append and read mode: It opens an existing file for both reading and appending. A file will be created it specified file does not exist. The possible operations are reading existing contents appending new contents to end of file but it cannot modify existing content.

9.2. Library Functions for Reading File

String Input /Output:

fgets(): This function is used to read from file. **Syntax:** ***fgets (string,int, fptr variable)***

fputs (): It is used to write a string to the file. **Syntax:** ***fputs(string, file ptr variable);***

Character Input/Output:

Using these functions, data can be read form file or written to file one character at a time.

fgetc(): This function is used to read a character form a file.

Syntax: ***char vari= fgetc(file variable);***

fputc(): This function is used to write a character form a file.

Syntax: ***fputc(char or char variable, fle variable);***

Formatted I/O:

These functions are used to read/write number, character or string form file in format as user requirement.

fprint(): This function is formatted output function, which is used to write some integer, float, char or string to specific file.

Syntax: ***fprint(file variable, "control string" list variable);***

fscanf(): This function is formatted input function which is used to read some integer, float, char or string form a file.

Syntax: ***fscanf(file variable," control string", &list variables);***

End of File (EOF):

The EOF represents End of File and determines whether the file associated with a file handle has reached end of file or not . This unique integer is sent to program by operating system and is defined in a header file stdio.h . While creating a file, operating system transmits EOF signal when it finds last character of file. In text mode, a special character 1A hex decimal is inserted after the last character in the file. This character is used to indicate EOF. If character last encountered an any pint in file, the read function will return EOF signal to program. An attempt to read after EOF might either case the program to terminate with errors or result in a infinite loop situations. Thus the last point of file is detected using EOF while reading data form file. Without this mark we cannot detect last character at the file such that is difficultly to find up to what time character is to be read while reading data from the file.

Binary Data Files:

The binary data files organize data into block containing continuous bytes of information. A binary file is file of any length that holds bytes with variables in range 0 to 0xff (0 to 225).These bytes have other meaning like value of 13 means, carriage return, 10 means, line feed, 26 means, end of file. In modern terms we recognized binary file as stream of bytes file opening mode of text file is appended by a character.

1. "r" is replaced by rb
2. "w" is replaced by wb
3. "a" is replaced by ab
4. "r+" is replaced by r+b
5. "w "+ is replaced by w+b
6. "a+" is replaced by a+b;
7. fptr=fopen("text, ‘’wb");

The default mode is text mode. Thus when simple r is written as mode of opening, this is assumed as text mode. We can write "rb" in place of r and file text mode of file. This syntax of all above mode in the case of binary mode is similar to the previous.

Difference Between Binary Mode and Text Mode

There are mainly three difference between binary and text mode, they are as follows:

1. In text mode, a special character EOF whose ASCII is 1A hex (26 in decimal) is inserted after last character in file. The binary mode files keep track of end from numbers present in directory entry of file ranges from 0 to 255. In this mode numbers of characters presents in directory entry of the file its mode. In this mode, numbers are stored in binary format. Therefore a number stored in memory happens to be 1A hex.(bytes have other means). Therefore a number stored in memory may be 1A but in text mode it represents the end of file.

2. In text mode, numbers are stored assuring of character where as in binary format they are stored the same way as hey are stored in computer main memory. The number 1234 occupies 2 bytes but it occupies 4 bytes in text mode because 1 character for one bytes in binary mode. If occupies 2 bytes. The solution of file opens to fread() function and replaced fprintf() and fscanf() functions.

3. The end of line single character, the new line charcter is marked by '\n' with ASCII value 10 in decimal whereas in dos e end of line is marked by two characters, the carriage return CR with ASCII value 13 in decimal and line feed (LF) with ASCII value 10 in decimal which is same as of c's new line character. In text mode a new line character is automatically converted into carriage return combination before being written to the disk of file. Like the carriage return line feed combination on the disk is converted into a new line when the file is read by C program. However these conversions don't take place in this case of binary character. Similarly & while reading the combination of CR/LF will be treated as two separate character as r and n character.

Record Input/Output

It is clear that character i/o and string i/o permits reading and writing of character data only formatted i/o permits reading and writing of characters and number. However the numbers are stored as sequential of character not in memory as result they take a lot of disk space in addition these methods provides no direct way to read write complex data types such array and structures. Array and structure can be handled by reading and writing each array element one at time but this approach is very inefficient.

The solution of these problems is record i/o also known as block i/o record i/o writes number to file in binary format so that integer are stored in 2 bytes are stored in 4 bytes with single precisions floating point numbers are stored in 4 bytes the same format used to store number data in memory record i/o permits reading and writing of data once. This process is not limited to single character or string of the few values in fact arrays structure array of structure can be read and rewritten as single unit.

The function fwrite() is used for record output while fread() is used for record input.

Syntax: ***fwrite (&ptr,size of array, number of structure array);***

fread(&ptr, sixe of array, number of structure array, fptr);

DIRECT /RANDOM Access

The reading and writing in all previous programs was in sequential. While reading data form a file, data items are read from beginning of file in sequence until end of file. Similarly while writing data to a file data items are placed one after the other in sequence. We can access a particular data items place in any location without starting beginning. Such type of access to a data items is called direct or random access.

For random access in a file, knowledge of file pointer is necessary. A file pointer to particular bytes in a file. While opening a file in write mode, the file pointer is at beginning of file. Every time when we write to a file, file pointer moves to the end of data items so that writing can conscious from that point while opening a file in read mode file pointer is at the beginning file. After reading data items the file pointer moves to the beginning of next data item, which can subsequently are read. However if file is io end in aped mode, then file pointer will be positioned at end of existing file, so that new data items can be written form there onward. Therefore we can read any data items form a file or write a file randomly if we would be successful to move this file pointer as our requirement. Important functions used in random access area as:

fseek(): It sets the file pointer associate with a stream file handle to a new position. This function is used to move he file pointer to different positions.

Syntax: ***fseek (fpt, offset, mode);***

rewind (): One way of positioning the file pointer to the beginning of the file to close the file and them reopen it again . This same stack can accomplished without closing ah file using rewind() function . This function positions hat file pointer in the beginning of the file. The use of functions rewind() is equivalent or using fssek()fptr,0,seek-*set*)

ftell(): This function determines current location of file pointer . It returns integer values.

Syntax: ***ftell(fptr)***