• Chapter 14 : File Handling in C

14.1 Overview

Q : 14-01-01 : Define permanent storage / file ?

Answer :

Permanent Storage / File : When we write programs to work with temporary data, the user has to enter data each time the program is executed. The data is stored on permanent storage in the form of files. [A file is a set of related records stored on a permanent storage medium].

14.2 The Stream

Q : 14-02-01 : Explain The Concept of Streams in C ?

Answer :

The Stream : C does not have any built-in method of performing file I/O, however the C standard library (stdio.h) contains a very rich set of I/O functions providing an efficient, powerful and flexible approach for file handling. A very important concept in C is the stream. [A stream is a logical interface to a file. A stream is associated to a file using an open operation. A stream is disassociated from a fife using a close operation]. There are two types of streams :

Text Stream : A text stream is a sequence of characters. In a text stream, certain character translations may occur (e.g., a new line may be converted to a carriage return/line-feed pair). This means that there may not be a one-to-one relationship between the characters written and those in the external device.

Binary Stream : A binary stream is a sequence of bytes with a one-to-one correspondence to those on the external device (i.e., no translations occur). The number of bytes written or read is the same as the number on the external device. (However, an implementation-defined number of bytes may be appended to a binary stream (e.g., to pad the information so that it fills a sector on a disk).

Important Note : In C, a file refers to a disk file, the screen, the keyboard, a port, a file on tape, etc.

14.3 NewLine and EOF Marker

Q : 14-03-01 : Explain NewLine and EOF Marker in C ?

Answer :

NewLine and EOF Marker : A text file is a named collection of characters saved in secondary storage e.g. on a disk. A text file has no fixed size. To mark the end of a text file, a special end-of-file character is placed after the last character in the file (denoted by EOF in C having ASCII Value 255 or xFF or all bits of the byte are 1s). When we create a text file using a text editor such as notepad, pressing the ENTER

key causes a newline character (denoted by \n in C) to be placed at the end of each line, and an EOF marker is placed at the end of the file. A specimen text file layout (Organization of text in a text file .txt) on storage is :

I		l	O	v	E		P	a	K	I	s	t	A	n	\n		
I		a	M		A		S	t	U	D	e	n	T	\n			
I		w	O	r	K		H	a	R	D	\n						
M	a	y		A	L	L	A	H		B	l	e	S	s		u	S	EOF

14.4 Opening a File

Q : 14-04-01 : Explain File Opening in C, its Modes, and File Pointer with an example program ?

Answer :

File Opening : Before reading from or writing to a file, it must be opened. All standard file handling functions of C are declared in stdio.h. Thus it is included in almost every program. To open a file and associate it with a stream, the fopen( ) function is used. Its prototype is :

```
FILE* fopen(const char* filename, const char* mode) ;
```

The fopen( ) function takes two parameters. The first is the name of the file. If the file is not in the current directory then its absolute path is be given. In this case, we need to escape the backslashes (i.e., use \\ instead of \) in the absolute path. For example :

```
fopen("c:\\Program Files\\MyApplication\\test.txt", "r") ;
```

The second parameter of fopen( ) function is the open "mode". It needs to be a string - not just a character. (Use double quotes, not single quotes). The "r" means we wish to open the file for reading (input). We could use a "w" if we wanted to open the file for writing (output). The fopen( ) function returns the NULL pointer if it fails to open the file for some reason. The most common reason for fopen( ) to fail is that the file does not exist. There can, however be, other reasons for failure so don't assume that is what went wrong for certain. Example is :

```
FILE *fp ;
if ((fp = fopen("myfile", "r")) == NULL)
{
        printf("Error Opening File\n");
        exit(1);
}
```

File Opening Modes : A file can be opened in following modes :

r	Open a text file for reading. The file must already exist.
W	Open a text file for writing. If the file already exists its contents are overwritten. If it does not exist, it will be created.
A	Open a text file for append. Data is added to the end of the existing file. If the file does not exist, it is created.
R+	Open a text file for both reading and writing. The file must already exist.
W+	Open a text file for reading and writing and its contents are overwritten. If the file does not exist, it is created.
A+	Open a text file for both reading and appending. If the file does not exist, it is created for both reading and writing.

The File Pointer : A file pointer is a variable of type FILE that is defined in stdio.h. To obtain a file pointer variable, a statement like the following is used :

FILE* fp ;

We know the symbol "*" as the arithmetic multiplication operator. But, it has entirely different meaning when used with a data type such as int, double, or FILE. It represents a pointer to the variable of type with which it is used e.g. int* represents a pointer to an integer, float* represents a pointer to a float variable, and FILE* represents a pointer to a variable of type FILE. Conceptually, a pointer is a memory cell whose content is the address of another memory cell.

Consider the following line of code:

```
int* var;
```

The variable *var* is a *pointer* to an integer type variable. It contains the address of a memory location (i.e. 0002) where an integer value can be stored. The contents of the memory location pointed to by the pointer *var* are referred to as *var. A pointer is a memory location that contains the address of another memory location. The file pointer i.e. FILE* is a pointer to the *file information* which defines various properties of the file (including name, status and current position).

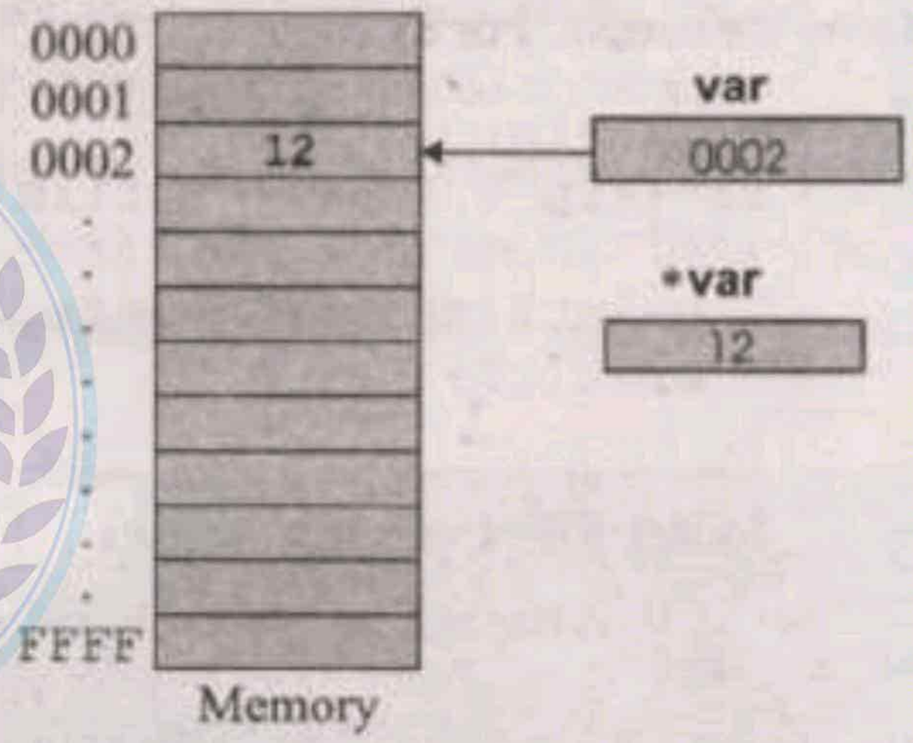

Fig. 14.1 Understanding the Pointer

Example : Write a program to demonstrate the use of pointers :

```
#include<stdio.h>
#include<stdio.h>
void main (void)
{
        int* var ;
```

```
                                    int num = 25 ;
                                    clrscr( ) ;
                                    var = &num ;
                                    printf ("\nAddress of variable num is %x", &num) ;
                                    printf ("\nContents (i.e. Value) of num is %d", num) ;
                                    printf("\nAddress of memory location pointed to by
                                    var is %x", var);
                                    printf("\nContents of memory pointed to by var is
                                    %d", *var) ;
                            }
```

It is clear from the program that a pointer type variable stores the address of a memory location containing the value, not the value itself. The address of the variable num i.e., fff4 may be different when you would execute this program on your computer. This is because a different memory location may be assigned to the variable num each time the program is executed.

14.5 Closing A File

Q : 14-05-01 : Explain File Closing in C ?

Answer :

File Closing : When a program has no further use of a file, it should close it with fclose( ) library function. The syntax of fclose( ) is as follows :

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

The fclose( ) function closes the file associated with fp, which must be a valid file pointer previously obtained using fopen( ), and disassociates the stream from the file. It also destroys structure that was created to store information about file. The fclose( ) function returns 0 if successful and EOF (end of file) if an error occurs.

14.6 Reading And Writing Characters To A File

Q : 14-06-01 : Explain the procedure for Reading and Writing Characters to a File in C ?

Answer :

Reading And Writing Characters To A File : Once a file has been opened, depending upon its opening mode, a character can be read from or written to it by using the following two functions.

```
int getc(FILE* fp)
int putc(int ch, FILE* fp)
```

The getc( ) function reads the next character from the file and returns it as an integer and if error occurs returns EOF. The getc( ) function also returns EOF when the end of file is encountered. The putc( ) function writes the character stored in the variable ch to the file associated with fp as an unsigned char. Although ch is defined as an int yet we may use a char instead. The putc( ) function returns the character written if successful or EOF if an error occurs.

Example : Write a program that reads a file and then writes its contents to another file (copy file).

```
#include<stdio.h>
void main (void)
{
        FILE *input ;
        FILE *output ;
        int ch ;
        // Try to open the input file. If it fails, print an ERROR
        message.
        if ((input = fopen("afile.txt", "r")) == NULL)
                printf("Can't open afile.txt for reading ! \n") ;
        // Now try to open the output file. If it fails, close the
        input file
        else if ((output = fopen("bfile.txt", "w") == NULL)
        {
                printf("Cant open bfile.txt for writing ! \n") ;
                fclose(input) ;
        }
        // If the files opened successfully, loop over the input
        one character
        // at a time.
        else
        {
                while ((ch = getc(input)) != EOF) // Process ch
                and     output it.
                        putc(ch, output) ;
        }
                // Close both the files
                fclose(input) ;
                fclose(output) ;
        }
```

Output : This program copies the contents of afile.txt to bfile.txt, both files are in current directory (i.e. the directory in which this .c file resides).

14.7 String Handling

Q : 14-07-01 : Explain String Handling (including declaration, initialization and assignment) in C ?

Answer :

String Handling in C : We display strings on screen with printf( ) function. String variables - the way C stores a string in a variable. Unlike variables of different numeric data types, C follows a different approach to handle strings. [C stores a string as an array of characters]. [An array is a group of contiguous memory locations, which can store data of the same data type]. The general form of declaring arrays in C :

Data_type array_name[n]

The data_type specify the type of data that is stored in every memory location of the array, array_name describe the array name, and 'n' is the subscript of array which

shows the total number of memory locations in the array. For example, the statements :

```
int balls[6] ;
double temperature[10] ;
```

define two arrays named balls and temperature (two sets of six and ten contiguous memory locations). In balls we can store six integer values, whereas in temperature we can store ten floating point values. Each value of array can be accessed via its subscripts. For example, consider the following statements :

```
balls[0]      = 4 ;   temperature[0]      = 37.0 ;
balls[l]      = 0 ;   temperature[2]      = 26.5 ;
balls[4]      = 6 ;   temperature[3]      = 19.6 ;
```

Declaring and Initializing String Variables : A string in C is implemented as an array. So
declaring a string variable is the same as declaring an array of type char; such as :

```
char name[16] ;
```

the variable name can hold string from 0 to 15 characters long. The last character of every string in C is '\ 0', the null terminator which indicates the end of the string. In this way the C let us manipulate each character of the string individually. Like variables of other data types, the strings can also be initialized :

```
char CityName[16] = "Lahore" ;
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
L	a	h	O	r	e	\0									

Notice the above figure showing the memory arrangement for the string variable name; the name[6] contains the character '\0'. This is the null character that marks the end of the string. This end marker allows the strings to have variable lengths. The rest of the memory locations in the array remains empty and are not allocated to any other variables. All of the C's string handling functions simply ignore whatever is stored in the cells following the null character. The following figure shows another string, longer than the previous, that the variable name can, store.

```
char name[l6] = "I love Pakistan";
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
I	\32	L	O	v	e	\32	P	a	k	i	s	t	A	n	\0

Notice that, in the initialization statement of the string we did not put a null character (\0) at the end. When we initialize a string, a null character is added at the end of it by default (space is ASCII 32).

String Assignment : Assigning a value to a string variable is not as simple as assignment to other variables. For example, we can assign an integer value to a variable of type int and a floating point value to a variable of type float by using assignment operator (i.e., =). But, it does not work with strings. So the following statement will cause an error :

```
name = "I Love Pakistan" ;
```

As name does not consist of a single memory location - it is an array. So, different characters are put in different memory locations of the array. This is done by copying every character of the string to respective index (subscript) of the array. For this purpose C provide a library for handling string manipulation i.e., library of string.h.

Most of the string manipulation functions of C are part of this library. There is a function named strcpy( ) which is used to copy a string to an array of characters (i.e., string variable). The syntax of strcpy( ) is as follows :

char* strcpy(char* dest, const char* source) ;

Hence, the following statement will successfully copy the string to the variable name.

strcpy(name, "I love Pakistan") ;

14.8 String Handling in Text Files

Q : 14-08-01 : Explain String Handling in Text Files (including fputs( ) and fgets( ) functions) in C ?

Answer :

String Handling in Text Files in C : When working with text files, C provides four functions which make file operations easier. The first two are called fputs( ) and fgets( ), which write or read a string from a file, respectively. Their prototypes are :

int fputs(char *str, FILE *fp)
char *fgets (char *str, int num, FILE *fp)

The fputs( ) function writes the string pointed to by str to the file associated with fp. It returns EOF if an error occurs and a non-negative value if successful. The null that terminates str is not written and it does not automatically append a carriage return / linefeed sequence.

The fgets() function reads string of characters from the file associated with fp into a string pointed to by str until num-1 characters have been read, a new line character (\n) is encountered, or the end of file (EOF) is encountered. The function returns str if successful and a null pointer if an error occurs.

Example : Write a program that accepts name and telephone numbers of your friends and write them in a file "Contacts.txt".

```
#include<stdio.h>
#include<string.h>
void main (void)
{
        FILE *ptrFile ;
        char name[30] ;
        char tel[12] ;
        if ((ptrFile = fopen("d:\\Contacts.txt", "w")) == NULL)
                printf("Can't open Contacts.txt for writing ! \n") ;
// If the file opened successfully, Get the name and telephone number
// and store them in the file
else
{
        do
        {
                printf("Enter the name(or press ENTER to quit) :
                gets (name) ;
```

```
                if (strlen(name) > 0)
                {
                        printf("Enter telephone number (max 10
characters) :  ") ;
                        gets(tel) ;
                        // write name and telephone number to file
                        fputs(name, ptrFile) ;
                        fputs("!", ptrFile) ;
                        fputs(tel, ptrFile) ;
                        fputs ("\n", ptrFile) ;
                }
                        }while(strlen(name) > 0) ;
                        // Close the file
                        fclose (ptrFile) ;
                }
        }
```

This program demonstrates the typical use of strings in text files. A sentinel loop reads name and telephone numbers unless the user enters an empty string for the name. In addition to fgets( ) and fputs( ), this program makes use of a new string handling function i.e., gets( ). The gets function accepts a string from keyboard and assigned it to the variable tel (an array of characters). The contents of the file contacts.txt are as follows :

```
Amir ! 8547348
nasir ! 7833129
aslam Hameed ! 2206301
Hamad Rehan !9214578
```

Here an exclamation sign (!) separates the name and the telephone number fields in each record. We may use another symbol such as a colon (:), as a separator. In text files, a separator is used to mark the end of the data for one filed, whereas the data for the next field follow this separator.

Example : Write a program that will read the contacts.txt file, and displays its contents on the screen.

```
#include<stdio.h>
#include<conio.h>
void main (void)
{
        FILE* ptrFile ;
char ch ;
int line = 3 ;
clrscr() ;
if((ptrFile = fopen("d:\\contacts.txt", "r")) == NULL)
        printf("can not open file") ;
else
{
        printf("Name") ;
        gotoxy (35, 1) ;
        printf( "Phone#\n") ;
```

```
printf("----------------------------------------
--\n") ;
while ((ch = getc(ptrFile)) != EOF)
{
        if (ch == '!')
        gotoxy(35, line) ;
        else if (ch == '\n')
        gotoxy(1, ++line) ;
        else
        printf("%c", ch) ;
}
        }
        fclose(ptrFile) ;
        getch() ;
    }
```

The function gotoxy() moves the cursor to a specified location on the screen. To use this function, the conio.h file must be included in the program. Its syntax is :

gotoxy(int col, int row) ;

The arguments of the gotoxy() function specify the coordinates of the screen where the cursor should move to.

Output : This program reads the file contacts txt and displays its contents on the screen.

```
NAME                    Phone#
-------------------------------------------
Amir                    8547348
nasir                   7833129
aslam Hameed            2206301
Hammad Rehan            9214578
```

The process of appending a file is same as that of writing a file, just open the file in append mode.

Example : Write a program that will append records in contacts.txt file.

```
#include<stdio.h>
#include<conio.h>
void main (void)
{
        FILE* ptrFile ;
char name[30] ;
char tel [12] ;
if ((ptrFile = fopen("d:\\Contacts.txt", "a")) == NULL)
        printf("Can't open Contacts.txt for writing / appending !\n")
;
// If the file opened successfully, get the name and telephone
number
// and append, them in the file
else
{
```

```
do
{
        printf("Enter the name(or press ENTER to quit) : ") ;
        gets (name) ;
        if (strlen(name) > 0)
        {
                printf("Enter telephone number (max 10
characters) :  ") ;
                gets(tel) ;
                // write name and telephone number to file
                fputs(name, ptrFile) ;
                fputs("!", ptrFile) ;
                fputs(tel, ptrFile) ;
                fputs ("\n", ptrFile) ;
        }
    }while(strlen(name) > 0) ;
    // Close the file
    fclose (ptrFile) ;
}
}
```

Important Note : This program seems very much similar to the program in previous example except that it opens contacts.txt in append mode, so new records are added at the end of the contacts.txt file.

14.9 Formatted I/O

Q : 14-09-01 : Explain Formatted I/O in Files (including fprintf( ) and fscanf( ) functions) in C ?

Answer :

Formatted I/O in Files in C : The other two file handling functions to be covered are fprintft( ) and fscanf( ). These functions operate exactly like printf() and scanf() except that they work with files.

Their prototypes are :

int fprintf (FILE *fp, char *control_string, ...)

int fscanf (FILE *fp, char *control_string, …)

Instead of directing their I/O operations to the console, these functions operate on the file specified by fp. Otherwise their operations are the same as their console based relatives. The advantages to fprintf( ) and fscanf( ) is that they make it very easy to write a wide variety of data to a file using a text format.

Example : Write a program (using Formatted I/O) that accepts name and telephone numbers of your friends and write them in a file "Contacts.txt".

```
#include<stdio.h>
#include<string.h>
void main (void)
{
```

```
FILE *ptrFile ;
char name[30] ;
char tel[12] ;
if ((ptrFile = fopen("d:\\Contacts.txt", "w")) == NULL)
        printf("Can't open Contacts.txt for writing ! \n") ;
// If the file opened successfully, Get the name and telephone
number
// and store them in the file
else
{
        do
{
        printf("Enter the name(or press ENTER to quit) :
        gets (name) ;
        if (strlen(name) > 0)
        {
                printf("Enter telephone number (max 10
characters) :  ") ;
                gets(tel) ;
                // write name and telephone number to file
                fprintf(ptrFile, "%s!%s\n", name, tel) ;
                // this line replaces 4 lines commented below
                // fputs(name, ptrFile) ;
                // fputs("!", ptrFile) ;
                // fputs(tel, ptrFile) ;
                // fputs ("\n", ptrFile) ;
        }
        }while(strlen(name) > 0) ;
        // Close the file
        fclose (ptrFile) ;
}
}
```

Exercise 14

Q-8. Write a program to merge the contents of two text files.

Answer :

Program Merge Files :

Important Note : Open Files a.txt for input and b.txt for appending. Read a.txt character by character and append each character at te end of b.txt (merge files).

```
#include<stdio.h>
void main (void)
{
        FILE *input ;
        FILE *appendoutput ;
        int ch ;
```

```
// Try to open the input file. If it fails, print an ERROR
message.
if ((input = fopen("a.txt", "r")) == NULL)
        printf("Can't open a.txt for reading ! \n") ;
// Now try to open the output file. If it fails, close the
input file
else if ((appendoutput = fopen("b.txt", "a") == NULL)
{
        printf("Cant open b.txt for merging / appending
        ! \n") ;
        fclose(input) ;
        }
        // If the files opened successfully, loop over the input
        one character
        // at a time.
        else
        {
                while ((ch = getc(input)) != EOF) // Process ch
                and     output it.
                        putc(ch, appendoutput) ;
        }
        // Close both the files
        fclose(input) ;
        fclose(appendoutput) ;
}
```

Q-9. Write a program that counts the total number of characters in a text file. [Note: consider the blank. space a character]

Answer :

Program Count Characters in a File :

Important Note : Open File a.txt for input read it character by character and keep counting. At the end of the file print / display # of characters counted.

```
#include<stdio.h>
void main (void)
{
        FILE *input ;
int ch, count = 0 ;
// Try to open the input file. If it fails, print an ERROR
message.
if ((input = fopen("a.txt", "r")) == NULL)
        printf("Can't open a.txt for reading/counting
characters! \n") ;
else
{
        while ((ch = getc(input)) != EOF) // count ch.
                count++ ;
```

```
                    }
                    printf("\n # of Characters in a.txt = %d", count) ;
                    // Close the file
                    fclose(input) ;
          }
```

Q-10. Write a program that counts the number of words in a text files and display the count on the screen.

Answer :

Program Count Words in a File :

Important Note : Open File a.txt for input read it character by character and keep counting spaces only that separate words from each other. At the end of the file print / display # of words counted.

```
#include<stdio.h>
void main (void)
{
        FILE *input ;
        int ch, count = 0 ;
        // Try to open the input file. If it fails, print an ERROR
        message.
        if ((input = fopen("a.txt", "r")) == NULL)
                printf("Can't open a.txt for reading/counting
        characters! \n") ;
        else
        {
                while ((ch = getc(input)) != EOF) // count ch.
                        if (ch == ' ') count++ ;
        }
        count++ ;
        printf("\n # of Words in a.txt = %d", count) ;
        // Close the file
        fclose(input) ;
}
```