School of CSE, KLE TECH

Data Structures and Algorithms (17ECSC204)

Chapter 07 - File Structures

Prakash B Hegade

2017-18

Files

What are files?

Definition 01: a collection of data or information that has a name. Almost all information stored in a computer must be in a file.

Reference: webopedia.com

Definition 02: A file is a place on the disk where a group of related data is stored.

Reference: Text book

Why use files?

Can we save the input data?

Giving a large input to a program every time is a time consuming job

Can we save the output data?

A programs calculation may be required for future run

The inference is,

"Can we save the data?"

A file name has ***two*** parts:

- A name and
- Extension

Extension depends on the type of the file.

Examples:

Contents

==========

"Losing data is a risk,
Hence we put them to the disk.
Study the operations for a while,
And that is how we build a file."

- PH

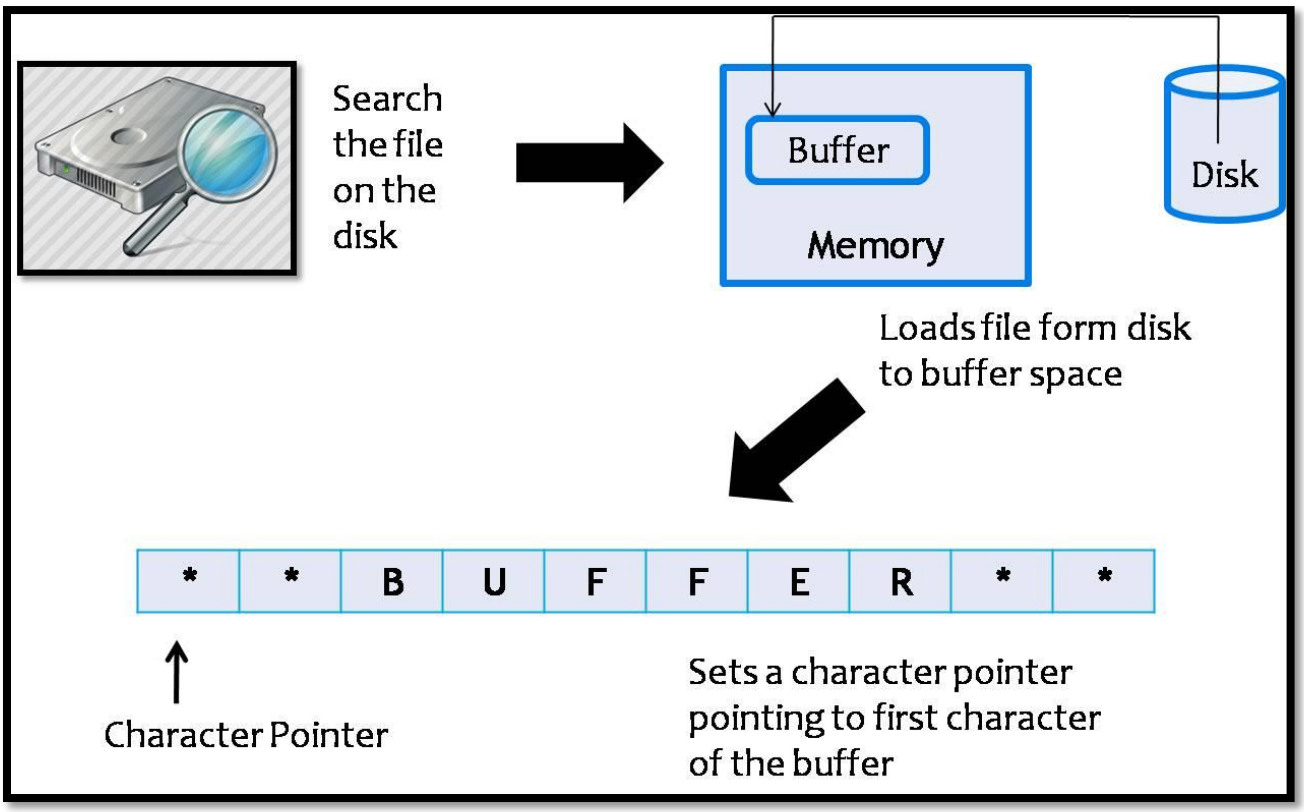
==========

Basic File Operations

The basic operations that can be performed on the file are:

- Opening
- Reading
- Writing
- Moving to specified location
- Closing

What happens when we try to open a file?



Operations:

- The file to be opened is searched on the disk
- If found,
  - Loads the file from the disk into place in memory called buffer
  - Sets up the character pointer that points to the first character of the buffer
- If not found (not highlighted in figure),
  - The open operation fails
  - Suitable error is thrown to the user

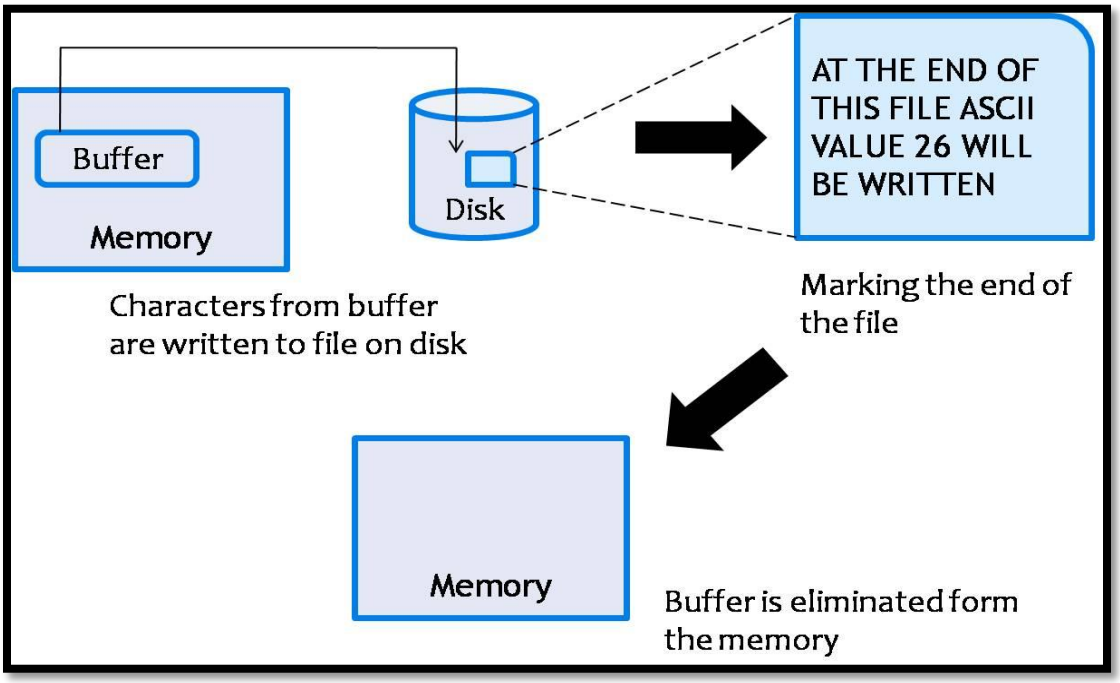
Note:

- Disk access is costlier than RAM access. Hence we load all the data into RAM buffer.
- All the information related to file in contained in a structure called FILE. File open returns the address of this structure, which we collect in structure pointer. FILE structure is defined in header <stdio.h>
- The structure of the FILE can be seen below:

```
typedef struct  {
        int level;              /* fill/empty level of buffer   */
        unsigned  flags;        /* File status flags            */
        char  fd;               /* File descriptor              */
        unsigned char hold;     /* Ungetc char if no buffer     */
        int bsize;              /* Buffer size                  */
        unsigned char  *buffer; /* Data transfer buffer         */
        unsigned char  *curp;   /* Current active pointer       */
        unsigned  istemp;       /* Temporary file indicator     */
        short token;            /* Used for validity checking   */
} FILE;
```

- Hence we declare a file pointer by writing FILE *fp before performing any file operations. This means we have created a pointer of type FILE
- Whenever we perform read / write operation on file, the file pointer is automatically incremented to its next position

What happens when we try to close a file?



Operations:

- Characters in buffer is written to file on disk
- Special character with ASCII value 26 is written at the end of the file
- Buffer associated with file is removed from the memory

File Modes

A file can be opened in several modes. The different available modes are:

"r" - searches a file. If opened successfully loads it into memory and sets up a pointer which points to first character in it.
If file cannot be opened returns NULL
Operations possible: reading from the file

"w" - searches a file. If file exists, its contents are overwritten. If file does not exist, a new file is created. NULL, if unable to open file.
Operations Possible: writing to the file

"a" – searches file. If opened successfully loads it into memory and sets up a pointer which points to last character in it. If file doesn't exist a new file is created. Returns NULL, if unable to open file.
Operations Possible: adding new contents at the end of the file

"r +" - searches a file. If opened successfully loads it into memory and sets up a pointer which points to first character in it.
If file cannot be opened returns NULL
Operations possible: reading from the file, writing new contents, modifying existing contents

"w+" - searches a file. If file exists, its contents are overwritten. If file does not exist, a new file is created. NULL, if unable to open file.
Operations Possible: writing to the file, reading them back and modifying existing contents of the file

"a+" – Searches file. If opened successfully loads it into memory and sets up a pointer which points to last character in it. If file doesn't exist a new file is created. Returns NULL, if unable to open file.
Operations Possible: reading existing contents, appending new contents to end of the file. Cannot modify existing contents

File API's

Basic API's

Operation	Syntax	Example	Return values
File Declare	FILE * fp;	FILE * f;	
File open	fp = fopen(char *filename, char * mode);	*fp = fopen("file.txt", "a");* *opening a file "file.txt" in append mode*	File pointer if successful NULL if failure
File close	int fclose(FILE *fp);	*fclose(fp);*	0 if successful EOF on error

File Read API's

Operation	Syntax	Example	Return values
Unformatted read	int getc(FILE * fp); read a character from file	*int ch = getc(fp);*	Character/integer pointed to by file if successful EOF on error or end of file
	num = getw(FILE *fp); read a integer from file	*int num = getw(fp);*	
Formatted read	fscanf(FILE *fp, "control string", variable_list);	*fscanf(fp, "%d %s", &number, string);*	Function returns the number of items that are successfully read from the file identified by FP

File Write API's

Operation	Syntax	Example	Return values
Unformatted write	int putc(int ch, FILE * fp); write a character to file	*putc(ch, fp);*	Writes character / integer to stream pointed by FP if successful. EOF on error or end of file
	putw(int num, FILE *fp); write an integer to file	*putw(10, fp);*	
Formatted write	fprintf(FILE *fp, "control string", variable_list);	*fprintf(fp, "%d %s", number, string);*	Function returns the no of items that are successfully written into the file identified by FP

Other API's

Operation	Syntax	Example	Return values
End of the file	feof(FILE *fp);	*feof(fp);*	Non Zero on success 0 on failure
File error	ferror(FILE *fp);	*ferror(fp);*	Non Zero on success 0 on failure
Random access	int fseek(FILE *fp, long offset, int position); offset → 0, positive or negative value, number of bytes to move position → 0: beginning , 1: current 2: eof	*fseek(fp, 20, 0);* → *move the file pointer by 20 bytes from beginning of the file* *fseek(fp, -40, 2);* → *move file pointer 40 bytes backwards from end of the file*	0 on success Non Zero on failure
File pointer position	long ftell(FILE *fp);	*int position;* *position = ftell(fp);*	current position on success EOF on error
Reset fp to start	rewind (FILE *fp);	*rewind(fp);*	

Text Files and Binary Files

Text files: contains only textual information like alphabets, digits, and special symbols. In actuality the ASCII codes of these characters are stored in text files.

Binary files: collection of bytes

Difference Table:

Concept	Text Files	Binary Files
Handling of new lines	\n to \r\n while writing to file and \r\n to \n while reading from file	No such conversion
Representation of EOF	EOF ASCII value 26	They track EOF from number of characters present in the directory entry of the file
Storage of numbers	In the form of characters. Each digit will occupy 1 byte	Stores the number in binary format. Each number would occupy same number of bytes on disk as it occupies in memory

File Access (Sequential or Random)

A file can be accessed sequentially by reading from first byte until we reach the end of the file. However in most of the cases it may not be the essential scenario. We might require searching a specific data in the file and displaying it. Or modify only certain required part in the file. Now after studying all the API's you should be able to easily guess which of the API's will support random access.

Indexed Sequential Files

The below explanation is referenced from Veegan Technologies.

In Sequential file structure, fixed format is used to store records. All records are of fixed length and arranged in a physical order. Next record is the one that physically follows the previous record. Field name and length are the attributes of this structure. A field known as key field identifies each record in sequential file.

Read and write operations on sequential file are done in sequential order. Sequential files become inadequate for interactive applications that use queries for reading and writing records because queries require random access and in sequential access, satisfying a single request takes a lot of time as every time searching is done sequentially. Unnecessary delays occur, if the user accesses large sequential files.

Index sequential file is advancement over sequential file. Index sequential file adds the feature of indexing of records to the sequential file. Indexing of records provides the facility of searching records randomly. Index to a file is a simple sequential file that contains index as its record.

An entry in index files is made up of two fields. Key field and a pointer pointing to some record in the main file. The pointer related to key field starts searching the record at the location it indicates. Index sequential file reduces the searching time to a great extent keeping the sequential nature of file. To increase the efficiency multiple levels of index are possible in index sequential file. Lower level of index is considered as the sequential file and its higher level is considered as its index file. Index sequential files are stored on the disks.

To sum up all,

- An **index** is an additional file that contains information about where records are stored. In the simplest case, an index would contain the key values of the records and the address where the record is stored.
- The index file would be a **sequential file**, with the index records stored in key value order. This makes the index easy to search.
- An **indexed sequential file** is a sequential file that also has an index. This would allow sequential access of records as well as indexed access.

Exercise

This section envelops programs covering the concepts studied in the chapter.

Program 7.1

Program to copy contents of one file to another

```
#include<stdio.h>
#include<stdlib.h>

int main()
{
   FILE *fp1, *fp2;
   char inputfile[20], outputfile[20];
   int ch;

   printf("Enter the input file name\n");
   scanf("%s", inputfile);
   if((fp1 = fopen(inputfile, "r")) == NULL)
   {
      printf("Error opening the file\n");
      exit(0);
   }

   printf("Enter the output file name\n");
   scanf("%s", outputfile);
   fp2 = fopen(outputfile, "w");
   if(fp2 == NULL)
   {
      printf("Error opening the file\n");
      exit(0);
   }

   while((ch=getc(fp1)) != EOF)
   {
      putc(ch,fp2);
   }

   fclose(fp1);
   fclose(fp2);
}
```

Program 7.2

Write a program to display the contents of the file with following conditions:

- **Print all the data in the file other than digits**
- **Print the string "There was a A here" wherever the character 'a' / 'A' is found**
- **At the end print the size of the file.**

Assume that there exists a file named "ReadPrint.txt" from which the data needs to be displayed.

```
#include <stdio.h>
#include <string.h>
int main(int argc, char *argv[])
{
   FILE *fp;
   char ch;
   int count = 0;

   fp = fopen("ReadPrint.txt", "r");
   if(fp == NULL) {
         printf("Error in Opening File\n");
         return 0;
   }

   while((ch = getc(fp)) != EOF) {
         if(isdigit(ch))
               ;
         else if(ch == 'a' || ch == 'A')
               printf(" There was a A here ");
         else
               printf("%c", ch);
         // the file size is number of bytes in file. So, let's keep a count
         count++;
   }
   printf("\nThe file size is... %d\n", count);
   fclose(fp);
   return 0;
}
```

Program 7.3

Program to demonstrate the usage of Binary files and Binary file API's

```
#include<stdio.h>
#include<stdlib.h>
```

```
struct data
{
   char name[20];
   int age;
};

int main()
{
   struct data d;
   FILE *fp1, *fp2;
   int choice;

   // Mode is suffixed with a 'b'
   fp1 = fopen("file.txt","ab");
   if(fp1 == NULL)
   {
      printf("Error opening the file\n");
      exit(0);
   }

   while(1)
   {
      printf("Enter name and Age");
      scanf("%s%d", d.name, &d.age);
      fwrite(&d, sizeof(d), 1, fp1);

      printf("Add one more data?\n1-Yes");
      scanf("%d", &choice);

      if(choice != 1)
         break;
   }

   fclose(fp1);

   fp2 = fopen("file.txt", "rb");
   if(fp2 == NULL)
   {
      printf("Error opening the file\n");
      exit(0);
   }
```

```
  while( fread(&d, sizeof(d), 1, fp2) == 1)
  {
    printf("%s\t%d\n", d.name, d.age);
  }
  fclose(fp2);
}
```

Program 7.4

Write a program to generate cyclic words for a given word and store the results in a file. Following operations have to be carried out.
In main function: accept a string from the user of length 04 and generate all cyclic words. For example if user enters "care", then cyclic words are:
arec
reca
ecar
care

These generated 04 words have to be written into a file. The format to be written inside the file is as follows:

```
arec # reca # ecar # care #
```

Delimiter "#" has to be inserted after every data. Read the file name from the user. Then from the main function provide options for the following until user wants to exit:

- Display the file data
- Delete a file data

'Delete a file data' is a function which performs following task:

- Open the file in read mode
- Get the input from the user on which word has to be deleted
- Search for the word in the file and record the position if found
- Close the file
- Open the file in read+ mode
- Jump to the recorded position and replace the word with special characters, say "@@@@"
- Close the file

'Display a file data' is a function which performs following task:

- Open the file in read mode
- Until the end of the file is reached print only the valid data from the file (do not print deleted data)
- Close the file

Solution:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void display()
{
   FILE * fp2;
   char name[5];
   char delimiter;

   // Open the file in read mode
   fp2 = fopen("data.txt", "r");
   if(fp2 == NULL) {
      printf("Cannot open the file\n");
      exit(0);
   }

  while(1)
  {
     if(feof(fp2))
      break;
     fscanf(fp2, "%s %c ", name, &delimiter);
      // Dont display if the record is already deleted
      if(strcmp(name, "@@@@")!=0)
     printf("%s\n",  name);
  }
   fclose(fp2);
}

void deletedata()
{
   FILE * fp1;
   FILE * fp4;

   int flag = 0;
   char delimiter;
   int curpos = 0;
   char cycle[5];
   char word[5];

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

```
  if(fp1 == NULL) {
    printf("Cannot open the file\n");
    exit(0);
  }
  printf("Enter the word to be deleted\n");
  scanf("%s", word);

  while(1)
  {
    // record the position of entry
    curpos = ftell(fp1);
    // Is it end of the file
    if(feof(fp1))
      break;

    fscanf(fp1, "%s %c ", cycle, &delimiter);
    if(strcmp(cycle, word)==0) {
      printf("found at position %d\n", curpos);
      flag =1;
      break;
    }
  }

  if(flag ==0) {
    printf("No such matching record was found\n");
    return;
  }
  fclose(fp1);
  // Mark as deleted if there is a entry found
  if(flag ==1) {
    fp4 = fopen("data.txt", "r+");
    if(fp4 == NULL)
    {
      printf("Cannot open the file\n");
      exit(0);
    }
    fseek(fp4, curpos,0);
    fprintf(fp4, "%s", "@@@@");
    fclose(fp4);
  }
}
```

```
int main()
{
   FILE * fp;
   char anagram[5];
   char cycle[5];
   char delimiter = '#';

   fp = fopen("data.txt", "w");
   if(fp == NULL) {
      printf("Unable to open the file\n");
      exit(0);
   }

   printf("Enter the string to generate anagrams\n");
   printf("The length of the string should be strictly 4 characters\n");
   scanf("%s", anagram);
   int count  = 0;
   while(count!=4)
   {
      cycle[0] = anagram[1];
      cycle[1] = anagram[2];
      cycle[2] = anagram[3];
      cycle[3] = anagram[0];
      cycle[4] = '\0';
      printf("The cycle is %s\n", cycle);
      strcpy(anagram, cycle);
      count++;

       fprintf(fp, "%s %c ", cycle, delimiter);
   }
   printf("\nFile will be closed. Data is being added successfully to file\n");
   fclose(fp);

   int choice = 0;
   while(1)
   {
      printf("Enter your choice\n");
      printf("1- Display file contents\n");
      printf("2- Delete file contents\n");
      printf("3- Exit\n");

      printf("Enter your choice\n");
```

```
      scanf("%d", &choice);

      switch(choice)
      {
         case 1: display();
               break;

         case 2: deletedata();
               break;

         case 3: exit(0);
      }
   }
   return 0;
}
```

~*~*~*~*~*~*~*~*~*~*~*~