REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université de M'sila
Faculté des Mathématiques et de l'Informatique
Département d'informatique

جامعة المسيلة
كلية الرياضيات والإعلام الآلي
قسم الإعلام الآلي

Chapter 2 :
Files

Algorithms and data structure

Presented by : Dr. Benazi Makhlouf
Academic year : 2023/2024

Chapter 06 Contents:

1. Introduction
2. The definitions
3. File Types
4. **Handling** files
5. Example

1. Introduction

- Information is stored as **variables** in central **memory**.
- It provides a very fast way for data storage and retrieval.
- **Volatile** memory retains data as long as there is electrical current; however, as soon as the electricity is cut off, this information disappears.
- The computer needs other devices to store information **permanently** and does not need electricity to store it. These devices are called **storage** units. For example: floppy disk, hard disk, CD, USB key, memory card, etc.
- This type of memory requires a lot of time to save and recover data, but it provides a large amount of storage space and does not lose its contents when the power is turned off.
- Data is stored as **files**.

2. Definitions

- A **file** is a structured set of information stored on volumes in a specific format. The operating system (e.g., Windows) controls how to store, retrieve, organize, and manage these files and the data they contain.
- A **folder** (or directory) is a virtual file organization container that can hold a group of files and/or other folders. This structure allows us to organize files in a tree format.
- **File name** is a character string used to specify the location of a specific file stored on the computer. It is utilized by the user to save and retrieve the file.
- A **file extension** is a set of letters added to the file name and comes after the last dot. This addition makes it easier for both the operating system and the user to determine which program can process the file. For example, 'myFile.mp3' is played by an audio player, while 'myFile.doc' is opened by MS Word.

2. Definitions (continued)

- **Path** is a set of folder names that specify the precise location of a particular file. These names are separated by / or , depending on the operating system. In Windows, the starting point is the main folder, identified by the name of the storage device and a Latin letter followed by a colon (:).

For example:

C:\myFolder1\myFolder2\myFile.doc

<table>
<tr><td>C:</td><td>/</td><td>myFolder1</td><td>/</td><td>myFolder2</td><td>/</td><td>myFile</td><td>.</td><td>doc</td></tr>
<tr><td>Volume name</td><td></td><td>folder name</td><td></td><td>folder name</td><td></td><td>Short file name</td><td></td><td>extension</td></tr>
<tr><td colspan="6">path</td><td colspan="3">file name</td></tr>
<tr><td colspan="9">The full name of the file</td></tr>
</table>

3. File Types:

Files are stored as bytes, with the only difference being how they are interpreted. From the user's perspective, files come in various types (audio, image, video, text, application, etc.), but from the programmer's point of view, there are two types: text and binary.

- **Text files:** All data in the file is stored as symbols (characters). Even numeric values (int, float, double, etc.) are formatted and stored as strings. These characters are typically organized as lines, with each line ending with '\n'. Each character is stored on one byte (if ASCII encoding is used). Text files can be opened with any text editor because each byte is translated into a character (char).
- **Binary files:** These files require interpretation by the program that created them or understands their specific formatting. Each byte or group of bytes can be translated in different ways, such as to different data types (int, float, bool, char, etc.).

Example

The number 124 can be stored as a single byte if represented as a numerical value, or as text, requiring 3 bytes.

- Files with the following extensions are typically considered text: txt, c, h, cpp, htm, xml, js.
- Files with the following extensions are generally considered binary: exe, bin, obj, zip, doc, mp3, jpg.

Observations:

- In the programming language, all input and output devices, such as the keyboard and monitor, are treated as files.
- Two special files are automatically configured for all programs:
  - `stdin` is the standard input file, usually associated with the keyboard.
  - `stdout` is the standard output file, typically associated with the screen.

4. File handling:

- While the process of storing files varies from one storage device and system to another and is often complex, the operating system hides these complexities. It provides comprehensive functions enabling the creation, opening, reading, or writing of files.
- A file can be envisioned as a series of records stored consecutively in the order they were entered, and they can only be read in that specific order. Random access to a record is not possible; all preceding recordings must be read first.
- When a file is opened, the read or write head positions itself on the first record. During subsequent reading or writing, the head moves directly to the following record. The end of the file is identified during playback when it encounters the special **EOF** symbol, known as "End Of File.

Typically, file processing follows these steps:

1. Declare a `FILE*` type variable for each file you wish to manipulate. This variable will point to the memory address associated with the file on the storage device.
2. Open the file for reading or writing using `fopen` and assign the address it returns to the previously declared variable.
3. Utilize this variable in all read and write operations on the file.
4. Close the file using `fclose`.

- In C, the input and output functions are part of the `<stdio>` library, so they must be included using the directive:

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

Declare a variable

statement	`FILE* f;`
Explanation	To create a variable

Opening the file

Function	`FILE * fopen (char * filename , char *mode);`
Return	If fopen fails, it returns **NULL**.
Explanation	The file named by **FileName** is opened for reading or writing depending on the value of the second **mode parameter.** Returns a pointer to this to use to locate the file. It should be used in place of the file name throughout the rest of the program.
Example	`f= fopen ("c:/test.txt","r+");`

Closing file

Function	`int fclose (FILE*f);`
Return	Returns 0 if the operation was successful.
Explanation	Close the file to clear temporary memory.
Example	`fclose (f);`

File Opening Mode

value	Explanation
r	Opens a file as read-only.
w	Creates a new file and opens it for writing only. If the file exists, it replaces it.
a	Opens a file to add (to write at the end of the file) and if the file does not exist it creates it.
r+	Open an existing file for updating (reading and writing).
w+	Creates a new file for updating (reading and writing). If a file with this name exists, it will be replaced.
a+	Open a file to update (read and write). at the end of the file. If the file does not exist, it will be created.

Functions for text files

Function	`int` **`fscanf`** `(FILE *f, char *format[, address , ...]);`
Return	Returns the number of variables read (input) and, if the end of the file is reached, returns **EOF**.
Explanation	It reads and formats input (converts and stores it into variables) from the file, with spaces considered separators between the data.
Example	**`fscanf`** `(f,"%d",&x );`
Function	`int` **`fprintf`** `(FILE *f, char *format[, argument, ...]);`
Return	Returns the number of bytes written, and on failure, **EOF**
Explanation	Writes formatted text to the file.
Example	**`fprintf`** `(f,"%.2f",y );`

Functions for text files (continued)

Function	`int` **`fgetc`** `(FILE *f);`
Return	Returns character, and if the end of file is reached or an error , it returns EOF.
Explanation	reads a character from the file.
Example	`x=` **`fgetc`**`(f);`
Function	`int` **`fputc`**`(int c, FILE *f);`
Return	If it fails, it returns **EOF**
Explanation	Writes a character to the file.
Example	**`fputc`**`(c,f);`

Functions for text files (continued)

Function	`char * fgets (char *s, int n, FILE *f);`
Return	If the end of the file is reached or an error is reached, it returns **NULL**.
Explanation	It reads an array of ‘n-1’ characters and stores it in variable ‘s’ and adds '\0' at the end, if the function encounters '\n' (end of line) it stores it in ‘s’ and stops reading.
Example	`fgets(s, 30, f);`
Function	`int fputs( const char *s, FILE *f);`
Return	If it fails, it returns EOF
Explanation	Writes a character string ‘s’ to the file but it does not store the character '\0' and does not append '\n' (carriage return) if it does not exist.
Example	`fputs( s,f );`

Functions for binary files

Function	`size_t fread(void* ptr , size_t size, size_t n, FILE* f);`
Return	Returns the number of items read.
Explanation	fread reads ‘n’ elements of data from the file, each element of size size bytes, and stores them in the location indicated by ‘ptr’. Where ‘ptr’ is any pointer. The total size to read is (n x size) bytes.
Example	`fread (&x, sizeof x, 1, f);`
Function	`size_t fwrite(const void* ptr, size_t size, size_t n, FILE* f);`
Return	Returns the number of items entered.
Explanation	fwrite writes ‘n’ elements of data to the file, the size of each element is size byte, where ‘ptr’ refers to its location in memory.
Example	`fwrite(x, sizeof x, 1, f);`

Other functions

Function	`int` **`feof`** `(FILE* f);`
Return	0 means the end has been reached
Explanation	To check whether the end of the file has been reached or not,
Example	**`If`** `(!`**`feof`**` (f))…`
Function	`int` **`fseek`** `(FILE *f, long offset, int whence );`
Return	Returns 0 on success
Explanation	Steps through the file a certain number of bytes or a certain number of characters of offset size from the start of the file if whence is equal to SEEK_SET.
Example	**`fseek`** `(f, 0, SEEK_SET);`
Function	`int` **`fflush`** `(FILE* f);`
Return	Returns 0 on success
Explanation	It writes data from temporary memory to the file.
Example	**`fflush`**`(f);`

fprintf(**stdout**,"text") ⇔ printf(" text ");	fscanf(**stdin** ,"% d",&x) ⇔ scanf("% d",&x);
fputc (c, **stdout**) ⇔ putchar (c);	c=fgetc (**stdin**) ⇔ c=getchar();
fputs (s, **stdout**) ⇔ puts (s);	fgets (s, n,**stdin**) ⇔ gets (s);

5. Example:

Declare a data structure used to store each student's information and it contains:

```
typedef struct {
  int num ; // num is a serial number
  char name[30]; //  name is a character string (array)
  char gender ; //  gender takes either the letter 'm' for a man, or 'w' for a woman
  float avg ; //  for the average a real number
} student ;
student e, t[10]; //  t array e single record
char str [50]; //  A character string used to read a line from a file.
FILE* f; //  A pointer indicating the file to open
```

S

// Populate the table with students information

```
for (int i=0; i<10; i++) {
  t[i].num =i;
  gets(t[i].name);
  t[i].gender = getchar();
  scanf("%f", &t[i]. avg );
}
f=fopen("c:/test.txt","w+"); //  Create the test.txt file, open it for reading and writing
for (int i=0;i<10;i++)
fprintf(f,"%d %s %c %.2f\n", t[i].num, t[i].name, t[i].gender, t[i].avg );
```

// Return the cursor (the reading position) to the beginning of the file.

```
fseek(f, 0, SEEK_SET);
```

// Reading the file line by line and displaying on the screen until the end of the file (NULL return).

```
while(fgets(str,50,f))
  printf("%s", str);
fclose(f); // to Close the file
f= fopen("c:/test.txt","w"); // recreate the file, everything it contained will be lost.
fwrite(&t, sizeof t, 1, f); // Write the table t to the file
fclose(f); // Close and reopen the read-only file
f= fopen("c:/test.txt","r");
// The file is read record by record at the variable e
while(fread(&e, sizeof(e),1,f))
  printf("%d %s %c %.2f\n", e.num, e.name , e.gender, e.avg );
fclose (f); // to close the file
```

End Chapter 02