Basis and Practice in Programming

Chapter 13: File Input/Output

Prof. Tamer ABUHMED
College of Software

Class Objectives

- **What is a File?**
- **How C Views a File?**
- **Explain How to Access a Text/Binary File.**
- **Discussions**

What is a File?

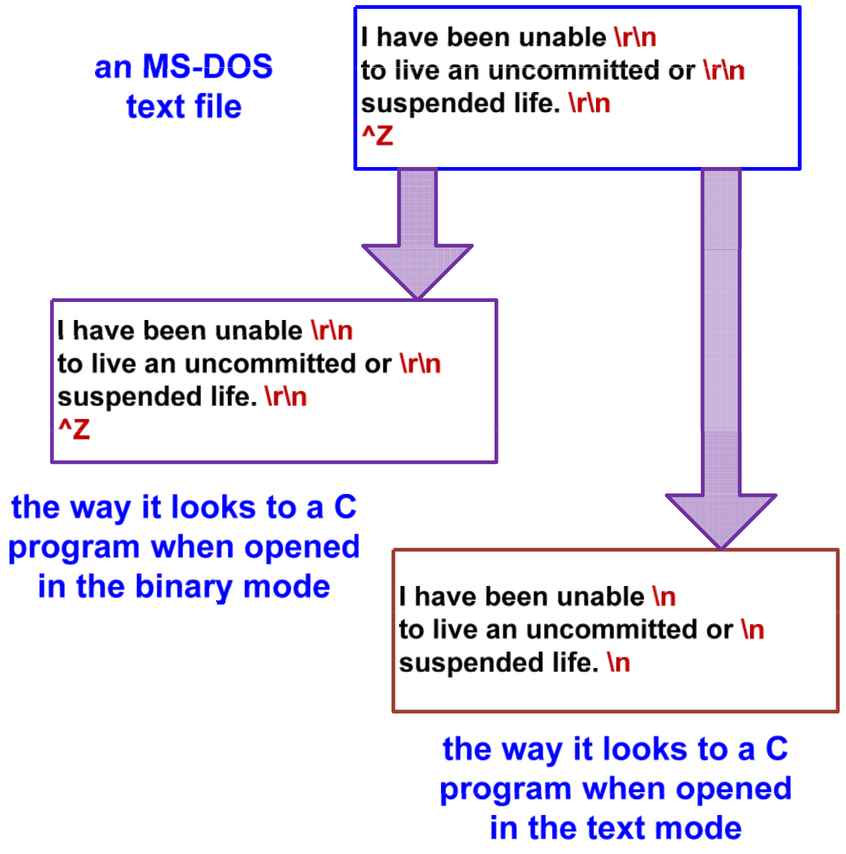
- **Definition of a File**
  - **A file is a named section of storage, usually on a disk.**

  - **How does C views a file?**
    - **C views a file as a sequence of bytes.**
    - **Each byte can be read individually.**

  - **File Views**
    - **Binary view**
      - **Each and every character is accessible to the program.**
    - **Text view**
      - **What the program sees can differ from what is in the file.**
        - » **How!**

How C Sees a File?

- **Viewing Modes**


an MS-DOS text file
I have been unable \r\n
to live an uncommitted or \r\n
suspended life. \r\n
^Z
I have been unable \r\n
to live an uncommitted or \r\n
suspended life. \r\n
^Z
the way it looks to a C program when opened in the binary mode
I have been unable \n
to live an uncommitted or \n
suspended life. \n
the way it looks to a C program when opened in the text mode

How C Sees a File? – contd.

- **Levels of Input/Output (I/O)**
  - **Low-level I/O**
    - **It uses the fundamental I/O services provided by the OS.**

  - **Standard High-level I/O**
    - **All I/O operations in C must be carried out through function calls.**
    - **It uses the standard package `#include <stdio.h>` of C library functions**
    - **Advantages:**
      - Portability: they work in a wide variety of computer environments
      - General character: they generalize to using files for I/O
    - **Disadvantage:**
      - Less performance: they don't take advantage of features peculiar to a particular system.

How C Sees a File? – contd.

- **Standard Files**
  - **C opens three files on behalf of the programmer**
    - **Standard Input: File read by getchar(), gets(), and scanf()**
    - **Standard Output: It is where normal output goes. Used by putchar(), puts(), and printf()**
    - **Standard Error Output: Provides logically distinct place to send error messages.**

Redirect output

- For example, if you want to write all your program results into a file called `data.txt`:
  - all that you need to do under Unix or Windows, if running in a terminal window, is to *redirect the output* from the program `prog` into the file `data.txt` by executing the program with the following command at the command prompt:

```
prog > data.txt
```

- This command instructs the system to execute the program `prog` but to redirect the output normally written to the terminal into a file called `data.txt` instead.
- You can have the program get its input from a file called `input.txt`, for example, by *redirecting the input* when the program is executed. If the program is called `prog`, the following command line works:

```
prog < input.txt
```

I/O redirection example

```c
/* pecho.c -- repeats input */

#include <stdio.h>
int main(void) {
  char ch;
  while ((ch = getchar()) != '*')
    putchar(ch);
  return 0;
}
```

F1.txt

```
This is fun to see
This data go to another file*
Yes ok
```

What is the result of running following command ?

```
pecho  <f1.txt >f2.txt
```

Files in C

- A file must first be opened properly before it can be accessed for reading or writing. When a file is opened, a stream is associated with the file.

- Successfully opening a file returns a pointer to (i.e., the address of) a file structure, which contains a file descriptor and a file control block.

```
FILE *fptr1, *fptr2 ;
```

  declares that `fptr1` and `fptr2` are pointer variables of type `FILE`. They will be assigned the address of a file descriptor, that is, an area of memory that will be associated with an input or output stream.

- Whenever you are to read from or write to the file, you must first open the file and assign the address of its file descriptor (or structure) to the file pointer variable.

Opening Files

- The statement:

```
fptr1 = fopen ( "mydata", "r" ) ;
```

  would open the file *mydata* for input (reading).

- The statement:

```
fptr2 = fopen ("results", "w" ) ;
```

  would open the file *results* for output (writing).

- Once the files are open, they stay open until you close them or end the program (which will close all files.)

Accessing a File

- **Writing in a File**
  - Write a string to a file

```
# include <stdio.h>
# include <stdlib.h>
void main(void)
{
    FILE *fp; // 1) "file pointer"
    char name[] = "My name is Huey.";
    long count = 0;
    fp = fopen("name.txt", "w");
     if(fp == NULL){// 2)Testing for Successful Open
        printf("Can't open the file\n");
        exit(1);
    }
    while(name[count]!='\0'){// 3) Start writing
        putc(name[count],fp);
        count++;
    }
   fclose(fp); // 4)Closing File
}
```

Accessing a File – contd.

- **Reading a File**
  - **Read a string from a file**

```
# include <stdio.h>
# include <stdlib.h>
void main(void)
{
    FILE *fpi; // 1)"file pointer"
    char ch;
    fpi = fopen("name.txt", "r");
    if(fpi == NULL){// 2) Testing for Successful Open
        printf("Can't open the file\n");
        exit(1);
    }
    while( (ch = getc(fpi)) != EOF){// 3)Start reading
     putchar(ch);
    }
    printf("\n");
    fclose(fpi); // 4) Closing File
}
```

Accessing a File – contd.

- **fopen() function**

Mode String	Meaning
"r"	Open a text file for reading.
"w"	Open a text file for writing. If file exists: file's data are lost. Otherwise, file is created.
"a"	Open a text file for writing. If file exists: data is appended. Otherwise, file is created.
"r+"	Open a file for update (reading and writing.)
…	… … … …
"rb", "wb", "ab", " ab+", "a+b", "wb+", "w+b", "rb+", "r+b"	Same as above for binary mode.

Accessing a File – contd.

- **getc() and putc()**
  - **Same as getchar() and putchar()**
  - ► **With difference that they deal with files.**

- **fclose(fp)**
  - **It closes the file identified by fp.**
  - will close the files and release the file descriptor space and I/O buffer memory.

  - **Example**

```
FILE *fp;
char ch = ‘F’;
fp = fopen(“out.txt”, ‘w’);
putc(ch, fp);
fclose(fp)
```

Accessing a File – contd.

fprintf() and fscanf() Functions

```c
# include <stdio.h>
# include <stdlib.h>
# define MAX 40

void main(void)
{
    FILE *fp;
    char words[MAX];
    if((fp = fopen("wordy", "a+")) == NULL){
      fprintf(stdout,"Can't open \"words\" file.\n");
        exit(1);
    }
    puts("Enter words to add to the file; press the Enter");
    puts("key at the beginning of a line to terminate.");
    while(fgets(words, 13, stdin)!= NULL   && words[0] != '\0')
        fprintf(fp, "%s ", words);

    puts("File contents:");
    rewind(fp);             /* go back to beginning of file */
    while(fscanf(fp,"%s", words) == 1)
      puts(words);
    if (fclose(fp) != 0)
        fprintf(stderr,"Error closing file\n");
}
```

Output

```
Enter words to add to the file; press the
n Enter
key at the beginning of a line to termina
te.
Hello
My name is Huey.
[ENTER]
File contents:
Hello
My name is Huey.
```

Accessing a File – contd.

- **fprintf() and fscanf() Functions**
  - **The function fprintf()**
    - ► **It uses the append mode.**
- **The function rewind()**
  - **Takes the program to the file beginning.**

Accessing a File – contd.

- **The Function fseek()**
  - **This function allows you to deal with the file as an array.**
  - **Therefore,**
    - **► You can move to whenever you need.**

- **The function ftell()**
  - **This function returns the current position in a file**
    - **► The returned value is of type long.**

Accessing a File – contd.

Moving in the File

- Note
  - 0L means 0 bytes as long value.

```
fseek(fp, 0L, SEEK_SET);        /* go to the beginning of the file */

fseek(fp, 10L, SEEK_SET);       /* go 10 bytes into the file */

fseek(fp, 2L, SEEK_CUR);        // advance 2 bytes from the current pos.

fseek(fp, 0L, SEEK_END);        // go to the end of the file

fseek(fp, -10L, SEEK_END);      // back up 10 bytes from the end of the file
```

  - SEEK_SET: Beginning of file
  - SEEK_CUR: Current position
  - SEEK_END: End of file

Accessing a File – contd.

- **Using fseek() and ftell()**

```
long last;  int count;
fseek(fp, 0L, SEEK_END);/* go to the end of the file */

last = ftell(fp);   /* number of bytes from beginning to end */

for(count = 1L; count <= last; count++)
{
     fseek(fp, -count, SEEK_END);
     ch = getc(fp);
     putchar(ch);
}
```

Accessing a File – contd.

- **Binary I/O: fread() and fwrite()**
  - **Mainly for dealing with non-text data**

Writing Binary file

```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
/* Our structure */
struct rec {
char name[10];
int age; float gpa;
};
int main() {
    int counter;
    FILE* ptr_myfile;
    struct rec my_record;
    ptr_myfile = fopen("test.bin", "wb");  // open binary file for writing
    if (!ptr_myfile) {/* check file is available */
      printf("Unable to open file!");
      return 1;
    }
    for (counter = 0; counter <3; counter++) {/* start writing loop*/
      scanf("%s", &my_record.name); fflush(stdin);
      scanf("%d", &my_record.age); fflush(stdin);
      scanf("%f", &my_record.gpa); fflush(stdin);
      fwrite(&my_record, sizeof(struct rec), 1, ptr_myfile);
    }
    fclose(ptr_myfile);
    return 0;
}
```

Accessing a File – contd.

- **Binary I/O: fread() and fwrite()**
  - **Mainly for dealing with non-text data**

Reading Binary file

```
#include<stdio.h>
/* Our structure */
struct rec {
char name[10];
int age;
float gpa;
};
int main() {
   int counter;
   FILE* ptr_myfile;
   struct rec my_record;
   ptr_myfile = fopen("test.bin", "rb"); // open binary file for reading
   if (!ptr_myfile) {
     printf("Unable to open file!");
     return 1;
   }
   fread(&my_record, sizeof(struct rec), 1, ptr_myfile);
   while(!feof(ptr_myfile)) {/* Check not end of the file*/
     printf("%s\n", my_record.name);
     fread(&my_record, sizeof(struct rec), 1, ptr_myfile);
   }
   fclose(ptr_myfile);
   return 0;
}
```

Summary & Discussion

- **What is a File?**
- **How C Views a File?**
- **Explain How to Access a Text/Binary File.**
- **Discussions**