

File manipulation in C

LIU, Yannan

ynliu@cse.cuhk.edu.hk

Overview

- What is a file
- Open files
- Basic file operations
 - Reading, Writing, Appending
- Close a File
- Binary file operation
- Practice

What is a file

- A file is a collection of information.
- A file can be processed as a char stream.
 - Line break is just a symbol in the stream.
 - File processor may "display" lines

Text file in Unix line break format:

```
This is an apple.\nThis apple is red.\nHello world!\n...
```

Data Structure for Representing a File in C

- The pre-defined data structure is **FILE**:

```
typedef struct {  
    ...buffer...;           // inside details are  
    ...buffer_size...;      // implementation dependent,  
    ...pointer...;          // we DO NOT touch the inside  
} FILE;
```

- We use **FILE *** (a structure pointer) rather than using the structure directly by value.
- File related data structures and functions are defined in “stdio.h”

Example on Using Files

Don't panic if you don't understand now!

```
1 #include <stdio.h>
2
3 int main() {
4     FILE *fptr;
5     char name[30];
6
7     fptr = fopen("data.txt", "r"); /* open file */
8
9     fscanf(fptr, "%s", name);      /* process file */
10    printf("Name = %s\n", name);
11
12    fclose(fptr);                  /* close file */
13 }
```

File data.txt:

Programming in C

Name = Programming

Open files

```
FILE *fopen(char *filename,  
            //file name(including the path)  
            char *mode  
            //open mode  
            );
```

- Opens the file named by **filename** in the way specified by the string **mode**.
- Return values:
 - Returns a non-**NULL** pointer (of type **FILE ***, usually referred to as a *stream*), if the file can be opened successfully;
 - Returns **NULL** pointer otherwise.

Open modes

Mode	Description
"r"	Open for reading
"w"	Truncate to zero length or create file for writing
"a"	Append; open for writing at end-of-file, or
"rb"	Open binary file for reading
"wb"	Over-write or create binary file for writing
	Pay attention when using fopen(): " double quote STRING MODE! "

Representing file path in different OSs

- Windows
 - `fopen("c:\\document\\student.txt", "r");`
- Linux
 - `fopen("/home/root/student.txt", "r");`

Example on fopen()

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

int main() {
    FILE *fp;

    fp = fopen("data.txt", "r");
    if (fp == NULL) {
        printf("Cannot open file.\n");
        exit(0);
    }
    .....
}
```

Reading: fscanf()

```
int fscanf(FILE *fp, char *format, args, ...);
```

- Similar to `scanf()`, except that the data comes from the stream `fp` (the file), instead of standard input(the console).
- Advances the "*file position indicator*" associated with `fp`.

fscanf()

```
int fscanf(FILE *fp, char *format,  
           args, ...);
```

- Return values:

- Returns the number of fields read and assigned from the stream `fp`.
- Returns `EOF` if the file position indicator reaches the end-of-file before the first assignment.

Example on fscanf()

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

int main() {
    FILE *fp;
    int x;

    fp = fopen("data.txt", "r");
    if (fp == NULL) {
        printf("Cannot open file.\n");
        exit(0);
    }

    fscanf(fp, "%d", &x);
    .....
}
```

Reading fgetc()

```
int fgetc(FILE *fp) ;
```

- Reads and returns the next character from the stream `fp`.
- Advances the file position indicator associated with `fp` one character ahead.
- Return values:
 - Returns the read character as an `int`.
 - Returns `EOF` at end-of-file or upon an error.

Example on fgetc()

```
#include <stdio.h>

int main() {
    FILE *fp;
    int c;

    fp = fopen("data.txt", "r");
    if (fp == NULL) {
        .....
    }

    c = fgetc(fp);
    if (c == EOF) {
        ..... /* error processing */
    }
    .....
}
```

Reading: fgets()

```
char *fgets(char *s, int n, FILE *fp) ;
```

- Reads characters from the stream `fp` into an array pointed to by `s`, until:
 - `n` – 1 characters are read, or
 - a newline character is read and transferred to `s`, or
 - an end-of-file condition is encountered.
- The string `s` is then NULL-terminated.

fgets()

```
char *fgets(char *s, int n, FILE *fp) ;
```

- Return values:

- Upon successful completion, **s** is returned.
- If end-of-file is encountered and no characters have been read, no characters are transferred to **s** and a **NULL** pointer is returned.
- If a read error occurs (e.g., reading a file that has not been opened), a **NULL** pointer is returned.

Example on fgets()

```
#include <stdio.h>

int main() {
    FILE *fp;
    char line[100];

    fp = fopen("data.txt", "r");
    if (fp == NULL) {
        .....
    }

    if (fgets(line, 100, fp) == NULL) {
        ..... /* error processing */
    }
    .....
}
```

Writing & Appending

```
int fprintf(FILE *fp, char *format,  
            args, ...);
```

- Similar to `printf()`, except that the data is written to the stream `fp` (instead of the stdout output).
- Advances the **file (buffer) position indicator** associated with `fp` after writing.

fprintf()

```
int fprintf(FILE *fp, char *format,  
            args, ...);
```

- Return values:
 - Returns the number of characters written to the stream `fp`.
 - Returns `EOF` on error.

Example on fprintf()

```
#include <stdio.h>

int main() {
    FILE *fp;

    fp = fopen("data.txt", "w");
    if (fp == NULL) {
        .....
    }

    fprintf(fp, "%s\n", "Hello!");
    .....
}
```

fputc()

```
int fputc(int c, FILE *fp);
```

- Writes a single character **c** to the stream **fp**.
- Advances the file position indicator associated with **fp** one character ahead.
- Return values:
 - Returns the value written (i.e., **c**).
 - Returns **EOF** on error.

Example on fputc()

```
#include <stdio.h>

int main() {
    FILE *fp;

    fp = fopen("data.txt", "w");
    if (fp == NULL) {
        .....
    }

    fputc('A', fp);
    .....
}
```

Closing Files

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

- Causes the buffered data associated with the stream `fp` to be written out to disk and the corresponding file to be closed.
- Return values:
 - Returns zero (`0`) upon successful completion.
 - Returns `EOF` otherwise.

Examples on fclose()

```
#include <stdio.h>

int main() {
    FILE *fp;
    fp = fopen("data.txt", "r");
    if (fp == NULL) {
        .....
    }

    .....    /* processing the file data */

    if (fclose(fp) == EOF)
        printf("Cannot close file.\n");

    return 0;
}
```


Inquiring End-of-File

```
int feof(FILE *fp) ;
```

- Tests for end-of-file on the stream `fp`.
- Return values:
 - Returns non-zero if end-of-file has previously been detected for the stream `fp`,
 - Returns zero (`0`) otherwise.

Example on feof()

```
#include <stdio.h>

int main() {
    FILE *fp;
    int c;

    fp = fopen("data.txt", "r");
    if (fp == NULL) {
        .....
    }

    if (feof(fp) == 0)
        c = fgetc(fp);
    .....
}
```

Binary Files

```
size_t fread( void *buffer_ptr,  
              size_t item_size, size_t no_of_items, FILE *fp );
```

```
size_t fwrite( const void *buffer_ptr,  
               size_t item_size, size_t no_of_items, FILE *fp );
```

- Consider the file stream as a sequence of bytes.

Example on reading binary file

```
#include <stdio.h>
int main() {
    FILE *fptr;
    double rainfall[12];
    int numRead;

    fptr = fopen("rain.dat", "rb");
    numRead = fread(rainfall, sizeof(double), 12, fptr);
    if (numRead != 12) {
        fputs ("Reading error", stderr); exit (3);
    }
    fclose(fptr);
    printf("%f", rainfall[3]);
    return 0;
}
```

```
size_t fread( void *buffer_ptr, size_t item_size,
              size_t no_of_items, FILE *fp );
```

Example on writing binary file

```
#include <stdio.h>
int main() {
    FILE *fptr;
    char buffer[] = { 'x' , 'y' , 'z' };
    int numWritten;

    fptr = fopen("myfile.bin", "wb");

    numWritten = fwrite(buffer, sizeof(char),
                        sizeof(buffer), fptr);
    if (numWritten != sizeof(buffer)) {
        fputs ("Writing error",stderr); exit (3);
    }
    fclose(fptr); return 0;
}
```

Summary

File processing is closely related to Operating System.

File operations **MAY FAIL**, we should always check the return value of a file function.

There are often cross-platform issues.

File format, file reading procedure, and file writing procedure should be designed **TOGETHER**.

Practice

- Remember the practices in structure session?
- Requirements:
 - ▶ Reading the student information from a file, instead of console.
 - ▶ Appending the results to the same file above.
- You can modify the complete code for lab4 directly.

Practice for reading file

- Define a structure type, which can record student ID, student name, and student age.
- Reading the below information from file

ID	Name	Age
11345	Tim	18
60765	John	17
19146	Jerry	20
20984	Lucy	22
57862	William	19

Practices for file appending

- After collecting all these information,
 - Practice 1
 - Record the student ID in the same file, for the each student whose age is larger than 19.
 - Practice 2
 - Record the student name for the each student, whose name's second letter is 'e'.
 - Practice 3
 - Record the student's name, whose age is the largest.