Systems Architecture

8. Working with files in C

Boni García

boni.garcia@uc3m.es

Telematic Engineering Department
School of Engineering

2024/2025

uc3m | Universidad Carlos III de Madrid

Table of contents

1. Introduction

- A **file** represents a sequence of bytes on the disk where a group of related data is **persistently** stored
- In C programming, we handle two types of files:
  - **Text** files:
    - Sequence of characters (letters, numbers, and other symbols) encoded using a given character set (e.g. ASCII, UTF-8)
    - Lines are separated by the newline character (this character is `\n` in UNIX-like systems but `\r\n` in Windows)
    - Human readable (we can use a text editor for creation and modification)
  - **Binary** files:
    - Sequence of bytes (binary data) typically interpreted as something different than text
    - Computer readable (we need a computer program to interpret the data in the file)

Table of contents

2. Text files

- The typical procedure to read/write text files in C is:
  1. Declare a pointer to the file (type `FILE`):

```
FILE *fp;
```

  2. Open the file using `fopen`:

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

  3. Perform read or write operations
  4. Close the file using `fclose`:

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

2. Text files

- The following table summarizes the access modes for **text** files:

Mode	Description	Behavior
`r`	Open for **reading**	If the file does not exist, `fopen()` returns `NULL`
`w`	Open for **writing**	If the file exists, its contents are overwritten. If the file does not exist, it is created
`a`	Open for **append** (new data is added to the end of the file)	If the file does not exist, it is created
`r+`	Open for both **reading** and **writing**	If the file does not exist, `fopen()` returns `NULL`
`w+`	Open for both **reading** and **writing**	If the file exists, its contents are overwritten. If the file does not exist, it is created
`a+`	Open for both **reading** and **appending**	If the file does not exist, it is created

2. Text files

- The following functions are used to **read** and **write** text from/to files:

Prototype	Description	
`int fgetc(FILE *fp);`	Reads and returns a single character at a time from a file. It returns EOF (end of file) when there are no more characters	read
`char *fgets(char *buf, int max, FILE *fp);`	Reads a line from the file. It stops when either (n-1) characters are read, the newline character is read, or EOF is reached	
`int fscanf(FILE *fp, const char *format,...);`	Reads formatted input from a file (same as `scanf` but from a file)	
`int fputc(int ch, FILE *fp);`	Writes a single character into a file	write
`int fputs(const char *str, FILE *fp);`	Writes a text line into a file	
`int fprintf(FILE *fp, const char *format, ...);`	Write formatted text from a file (same as `printf` but from a file)	

2. Text files

- Basic example for writing a text file:

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

int main() {
    FILE *fp = fopen("file.txt", "w");
    if (fp == NULL) {
        printf("Error opening file\n");
        exit(1);
    }

    // Write a line to the file
    fprintf(fp, "I am writing into the file\n");

    int i;
    printf("Enter integer: ");
    scanf("%d", &i);

    // Write another line to the file
    fprintf(fp, "You entered: %d\n", i);

    fclose(fp);

    return 0;
}
```

2. Text files

- Basic example for reading a text file line by line:

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

int main() {
    FILE *fp = fopen("file.txt", "r");
    if (fp == NULL) {
        printf("Error opening file\n");
        exit(1);
    }

    int buf_size = 255;
    char buffer[buf_size];
    while (fgets(buffer, buf_size, fp) != NULL) {
        printf("%s", buffer);
    }

    fclose(fp);

    return 0;
}
```

Table of contents

3. Binary files

- In C programming, we use can **binary files** to persist any kind of **data type** in the C program
- The procedure to read/write binary files in C is the same than we used in text finales, but using different functions for reading/writing:
  1. Declare a pointer to the file (type `FILE`)
  2. Open the file using **`fopen`** (using an access mode for binary files)
  3. Perform read (**`fread`**) or write (**`fwrite`**) operations
  4. Close the file using **`fclose`**

3. Binary files

- The following table summarizes the access mode for **binary** files:

Mode	Description	Behavior
`rb`	Open for **reading**	If the file does not exist, `fopen()` returns NULL
`wb`	Open for **writing**	If the file exists, its contents are overwritten. If the file does not exist, it is created
`ab`	Open for **append** (new data is added to the end of the file)	If the file does not exist, it is created
`rb+`	Open for both **reading** and **writing**	If the file does not exist, `fopen()` returns NULL
`wb+`	Open for both **reading** and **writing**	If the file exists, its contents are overwritten. If the file does not exist, it is created
`ab+`	Open for both **reading** and **appending**	If the file does not exist, it is created

3. Binary files

- The following functions are used to **read** and **write** from/to **binary files**:

Prototype	Description
`size_t fwrite(const void *ptr, size_t size, size_t count, FILE *fp);`	Writes an array of **`count`** elements in **`fp`**, each one with a size of **`size`** bytes, from the block of memory pointed by **`ptr`**. The *file position* advances the total number of bytes written. It returns the total number of elements successfully written
`size_t fread(void *ptr, size_t size, size_t count, FILE *fp);`	Reads an array of **`count`** elements, each one with a size of **`size`** bytes from **`fp`**, and stores them in the block of memory specified by **`ptr`**. The *file position* advances the total amount of bytes read. It returns the total number of elements successfully read (which is different than **`count`** when EOF is reached)

The ***file position*** of a file describes where in the file the stream is currently reading or writing. This file position is represented as an integer which counts the number of bytes from the beginning of the file

3. Binary files

- Basic IO examples with binary files (write 1):

The perror function prints error messages to the stderr stream based on the error state in the *errno* (i.e., a standard global variable that stores an error code occurred during any function call)

```c
#include <stdio.h>

int main() {
    FILE *fp = fopen("file1.bin", "wb");
    if (!fp) {
        perror("An error occurred opening the file");
        return 1;
    }

    int i = 100;
    float f = 20.5;

    fwrite(&i, sizeof(int), 1, fp);
    fwrite(&f, sizeof(float), 1, fp);

    fclose(fp);

    return 0;
}
```

We can use the command hexdump to visualice the hexacimal content

```
$ hexdump -C file1.bin
00000000  64 00 00 00 00 00 a4 41                           |d......A|
00000008
```

3. Binary files

- Basic IO examples with binary files (read 1):

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

int main() {
    FILE *fp = fopen("file1.bin", "rb");
    if (!fp) {
        perror("An error occurred opening the file");
        return 1;
    }

    int i;
    fread(&i, sizeof(i), 1, fp);
    float f;
    fread(&f, sizeof(f), 1, fp);

    printf("The content of the binary file is:\n");
    printf("%d\n", i);
    printf("%f\n", f);

    fclose(fp);

    return 0;
}
```

```
The content of the binary file is:
100
20.500000
```

3. Binary files

- Basic IO examples with binary files (II):

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

#define MAX_STR 255
#define SIZE 10

int main() {
    char content[10][MAX_STR];

    for (int i = 0; i < SIZE; i++) {
        sprintf(content[i], "This is line %d", i + 1);
    }

    FILE *fp = fopen("file2.bin", "wb");
    if (!fp) {
        perror("An error occurred opening the file");
        return 1;
    }
    fwrite(content, MAX_STR, SIZE, fp);
    fclose(fp);

    return 0;
}
```

write

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

#define MAX_STR 255

int main() {
    FILE *fp = fopen("file2.bin", "rb");
    if (!fp) {
        perror("An error occurred opening the file");
        return 1;
    }

    printf("The content of the binary file is:\n");
    char record[MAX_STR];
    while (fread(&record, sizeof(record), 1, fp) == 1) {
        puts(record);
    }

    fclose(fp);

    return 0;
}
```

read

3. Binary files

- Basic IO examples with binary files (III):

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

#define MAX_STR 255
#define SIZE 10

struct data {
    char str[MAX_STR];
    int integer;
};

int main() {
    struct data *content = (struct data*) calloc(SIZE, sizeof(struct data));

    for (int i = 0; i < SIZE; i++) {
        sprintf(content[i].str, "This is line %d", i + 1);
        content[i].integer = i + i;
    }
    FILE *fp = fopen("file3.bin", "wb");
    if (!fp) {
        perror("An error occurred opening the file");
        return 1;
    }
    fwrite(content, sizeof(struct data), SIZE, fp);
    fclose(fp);
    free(content);
    return 0;
}
```

write

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

#define MAX_STR 255

struct data {
    char str[MAX_STR];
    int integer;
};

int main() {
    FILE *fp = fopen("file3.bin", "rb");
    if (!fp) {
        perror("An error occurred opening the file");
        return 1;
    }

    printf("The content of the binary file is:\n");
    struct data record;
    while (fread(&record, sizeof(struct data), 1, fp) == 1) {
        printf("String: %s -- Integer: %d\n", record.str, record.integer);
    }

    fclose(fp);

    return 0;
}
```

read

3. Binary files

- The following functions are used to manipulate the **file position** associated with a file stream:

Prototype	Description
`int fseek(FILE *fp, long int offset, int origin);`	Changes the file position associated with **fp**. For binary files, the new position is defined by adding **offset** to a reference position specified by **origin**. For text files, **offset** shall either be zero or a value returned by a previous call to **ftell**, and **origin** shall necessarily be **SEEK_SET**. The parameter **origin** specifies the position used as a reference for the **offset**: - **SEEK_SET**: Starts the offset from the beginning of the file. - **SEEK_END**: Starts the offset from the end of the file. - **SEEK_CUR**: Starts the offset from the current location of the cursor in the file.
`long int ftell(FILE *fp);`	Returns the current file position. For binary streams, this is the number of bytes from the beginning of the file.
`void rewind(FILE *fp);`	Sets the file position associated with an stream to the beginning of the file.

3. Binary files

- Basic IO examples with binary files (using `fseek`):

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

#define MAX_STR 255

int main() {
    FILE *fp = fopen("file2.bin", "rb");
    if (!fp) {
        perror("An error occurred opening the file");
        return 1;
    }

    printf("The content of the binary file (showing only odd values) is:\n");
    char record[MAX_STR];
    while (fread(&record, sizeof(record), 1, fp) == 1) {
        puts(record);
        fseek(fp, sizeof(record), SEEK_CUR);
    }

    fclose(fp);

    return 0;
}
```

```
The content of the binary file (showing only odd values) is:
This is line 1
This is line 3
This is line 5
This is line 7
This is line 9
```

3. Binary files

- Basic IO examples with binary files (using `ftell`):

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

#define MAX_STR 255

int main() {
    FILE *fp = fopen("file2.bin", "rb");
    if (!fp) {
        perror("An error occurred opening the file");
        return 1;
    }

    printf("The content of the binary file is:\n");
    char record[MAX_STR];
    int file_pos = ftell(fp);
    printf("[file position at the beginning is: %d]\n", file_pos);
    while (fread(&record, sizeof(record), 1, fp) == 1) {
        file_pos = ftell(fp);
        printf("%s [file position is: %d]\n", record, file_pos);
    }
    printf("[file position at the end is: %d]\n", file_pos);

    fclose(fp);
    return 0;
}
```

```
The content of the binary file is:
[file position at the beginning is: 0]
This is line 1 [file position is: 255]
This is line 2 [file position is: 510]
This is line 3 [file position is: 765]
This is line 4 [file position is: 1020]
This is line 5 [file position is: 1275]
This is line 6 [file position is: 1530]
This is line 7 [file position is: 1785]
This is line 8 [file position is: 2040]
This is line 9 [file position is: 2295]
This is line 10 [file position is: 2550]
[file position at the end is: 2550]
```

Table of contents

4. Takeaways

- A **file** represents a sequence of bytes on the disk where a group of related data is **persistently** stored
- In C programming, we typically handle two types of files: **text** (sequence of characters) and **binary** (sequence of bytes) files
- The typical procedure to read/write files in C is:
  1. Declare a pointer to the file (type **`FILE`**)
  2. Open the file using **`fopen`**
  3. Perform read/write operations:
     - Functions for text files: **`fprintf`**, **`fscanf`**, …
     - Functions for binary files: **`fwrite`**, **`fread`**
  4. Close the file using **`fclose`**