

Programming 1

Lesson 8. Files in C

Degree in Computer Engineering

Syllabus

2

1. Introduction
2. Types of files in C
3. Common functions for files in C
4. Functions for text files in C
5. Functions for binary files in C

1. Introduction

3

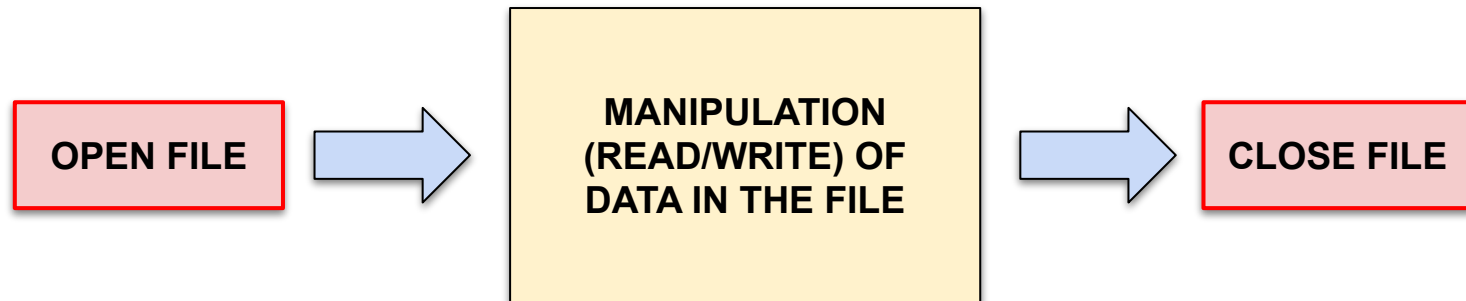
- **Files** (or archives) are data structures stored in **secondary memory**, unlike the ones seen so far, which use main memory.
- They allow **data** to be **stored permanently** and can be accessed and modified at any time.
- The file data type in C is the **FILE** type and the data is accessed through a file pointer.

```
FILE *file; // we need to declare a variable of type  
             // pointer to a file to get access to all  
             // the data in a file
```

1. Introduction

4

- To access the contents of a file, **the file must first be opened.**
- **Access to data** in C files is done **sequentially.**
- When the manipulation of the data in a file is complete, **it is necessary to close the file to keep its integrity.**



2. Types of files in C

5

■ Text files.

- They store text strings consisting of alphanumeric characters.
- They can be opened and modified using any text editor.

■ Binary files.

- They store information in the same way as it is done in memory.
- The contents of the files cannot be viewed with a text editor.

3. Common functions for files in C

6

Common functions for the handling of text files and binary files

■ Open a file

FILE ***fopen**(const char ***name**, const char ***mode**)

It allows us to open the file whose name, with relative or absolute path, is specified in the **name** parameter. The **mode** parameter is a text string that allows us to specify how the file is opened:

- "**r**" (text) / "**rb**" (binary): opens the file in read mode. The file must exist. If the file does not exist the function returns **NULL**. If it exists, a pointer to the file is returned.
- "**w**" (text) / "**wb**" (binary): opens the file in write mode. If the file does not exist, it creates a new file. If the file exists, it deletes the file and creates a new one. It returns the pointer to the file.
- "**a**" (text) / "**ab**" (binary): opens the file in write mode. If the file does not exist, it creates a new file. If the file exists, it opens the file keeping its contents and appending the new data after the existing data. It returns the pointer to the file.

3. Common functions for files in C

7

Common functions for the handling of text files and binary files

- Close a file

```
int fclose(FILE *file)
```

It closes the file whose pointer is passed in the `file` parameter.

- Knowing whether the end of the file has been reached

The following function allows us to know if the end of the file passed in the parameter `file` has been reached, when accessing in read mode.

```
int feof(FILE *file)
```

It returns a non-zero value (boolean value true) if and only if the end of the file has been reached, otherwise it returns 0 (boolean value false).

3. Common functions for files in C

8

Common functions for the handling of text files and binary files

- Move the cursor of the file to a specific location

```
int fseek(FILE *file, long int offset, int orig)
```

It allows us to move the file cursor relative to the position specified by **orig**, as many bytes as indicated by **offset**. The **offset** value can be positive or negative, and the possible values of **orig** are:

- **SEEK_SET**: Offset relative to the start of the file.
- **SEEK_CUR**: Offset relative to the current position of the file cursor.
- **SEEK_END**: Offset relative to the end of the file.

- Move the cursor to the beginning of the file

```
int rewind(FILE *file)
```

It moves the cursor to the beginning of the file.

4. Functions for text files in C

9

Basic functions

- Write a character to a text file

```
int putc(int ch, FILE *file)
```

It writes the character passed in the **ch** parameter to the file specified in the **file** parameter. If **ch** is of type **char**, it is automatically converted to **int**.

- Write a string to a text file

```
int fputs(const char *string, FILE *file)
```

It writes the character string passed in the **string** parameter to the file specified in the **file** parameter. It does not write the end-of-string character (**\0**).

4. Functions for text files in C

10

Basic functions

Example:

```
#include<stdio.h>

int main(){
    FILE *file;
    char op;
    char string[30];

    file = fopen("data.txt", "w");

    if(file){ // if it was possible to open/create ...
        printf("Enter your name: ");
        scanf(" %[^\\n]s", string);
        printf("Choose option (A, B, C): ");
        scanf(" %c", &op);

        fputs(string, file);
        fputc('\\n', file); // new line
        fputc(op, file);

        fclose(file);
    }

    return 0;
}
```

```
$ ./program
Enter your name: José Antonio
Choose option (A, B, C): B
```

File data.txt:

```
José Antonio
B
```

4. Functions for text files in C

11

Basic functions

- Read a character from a text file

```
int getc(FILE *file)
```

It reads and returns the next available character in the file specified by the **file** parameter, moving the file pointer to the next character. The value returned is the ASCII code of the character read and is converted to **char** automatically if necessary.

- Read a string of characters from a text file

```
char *fgets(char *string, int n, FILE *file)
```

It reads a line of text from the file and stores it in the string passed in the **string** parameter. It stops reading characters when it reads **n-1** characters, or an end-of-line (**\n**) character, or reaches the end of the file. It returns **string** if everything went well, or **NULL** if an error occurred.

4. Functions for text files in C

12

Basic functions

Example:

```
#include<stdio.h>

int main(){
    FILE *file;
    char op;
    char string[30], sresult[30];

    file = fopen("data.txt", "r");

    if(file){
        if (fgets(string, 30, file) != NULL) {
            op = getc(file);

            printf("Name: %s", string);
            printf("Option: %c\n", op);

        }

        fclose(file);
    }

    return 0;
}
```

File data.txt:

```
José Antonio
B
```

```
$ ./program
Name: José Antonio
Option: B
```

4. Functions for text files in C

13

Basic functions

- Write data of any type to a text file

To write one or more values to a text file, whether or not they are of character type, the file version of `printf` is used:

```
int fprintf(FILE *file, const char *format, arguments)
```

- `file`. Name of the file to which the values are to be written.
- `format`. A string of text to be written. It may optionally contain *flags* (the same flags type as for `printf`), which begin with % and allow interpolation of values specified in `arguments`.
- `arguments`. List of values to interpolate in the string `format` where the corresponding flags are located.

4. Functions for text files in C

14

Basic functions

Example:

```
#include<stdio.h>

int main(){
    FILE *file;
    char op;
    char string[30];
    int age;

    file = fopen("data.txt", "w");

    if(file){
        printf("Enter your name: ");
        scanf(" %[^\\n]s", string);
        printf("Enter your age: ");
        scanf(" %d", &age);
        printf("Choose option (A, B, C): ");
        scanf(" %c", &op);

        fprintf(file, "%s\\n", string);
        fprintf(file, "%d %c\\n", age, op);

        fclose(file);
    }

    return 0;
}
```

```
$ ./program
Enter your name: José Antonio
Enter your age: 32
Choose option (A, B, C): B
```

File data.txt:

```
José Antonio
32 B
```

4. Functions for text files in C

15

Basic functions

- Read data of any type from a text file

To read one or more values from a text file, whether or not they are of character type, the file version of `scanf` is used:

```
int fscanf(FILE *file, const char *format, arguments)
```

- `file`. Name of the file to which the values are to be written.
- `format`. A text string with *flags* (the same flags type as for `printf`), starting with %, which indicate the order of the variables that are given in `arguments` where the read values will be stored.
- `arguments`. List of variables that will take the values read in the order indicated in the string `format`. Variables whose data type is simple need to be preceded by the address symbol, `&`, since `fscanf` needs to access the memory address of the variable to write the value read.

4. Functions for text files in C

16

Basic functions

Example:

```
#include<stdio.h>

int main(){
    FILE *file;
    char op;
    char string[30];
    int age;

    file = fopen("data.txt", "r");

    if(file){
        fscanf(file, "%[^\\n]s", string);
        fscanf(file, "%d %c", &age, &op);

        printf("Name: %s\\n", string);
        printf("Age: %d\\n", age);
        printf("Option: %c\\n", op);

        fclose(file);
    }

    return 0;
}
```

File data.txt:

José Antonio
32 B

```
$ ./program
Name: José Antonio
Age: 32
Option: B
```


5. Functions for binary files in C

17

Basic functions

■ Write to a binary file

```
size_t fwrite(const void *ptr, size_t size,  
              size_t numElements, FILE *file)
```

Where the different parameters are as follows:

- **ptr**: pointer to the array of elements to write.
- **size**: size in bytes of each element of the array. The **sizeof()** function can be used to find out the size of any data type.
- **numElements**: number of array elements to write.
- **file**: pointer to the file (an object of type FILE) where the information will be written.

As a result, the function returns the number of bytes written.

5. Functions for binary files in C

18

Basic functions

■ Read from a binary file

```
size_t fread(const void *ptr, size_t size,  
             size_t numElements, FILE *file)
```

Where the different parameters are as follows:

- **ptr**: pointer to the block of memory where the read elements of the file will be stored.
- **size**: size in bytes of each element of the array. The **sizeof()** function can be used to find out the size of any data type.
- **numElements**: number of items to read from the file.
- **file**: pointer to the file (an object of type FILE) from which the elements (bytes) will be read.

As a result, the file pointer is moved forward by the number of bytes read. The function returns the number of bytes read.

5. Functions for binary files in C

19

Basic functions

Example:

```
#include<stdio.h>

#define MAX_PEOPLE 100
#define MAX_STRING 30
#define FILENAME "people.txt"

typedef char TString[MAX_STRING];
typedef struct{
    TString dni;
    TString name;
    int age;
} TPerson;
typedef TPerson TPeopleList[MAX_PEOPLE];

int main(){
    TPeopleList list;
    int nPeople = 0;

    ...
    readData(list, &nPeople);
    ...
    writeData(list, nPeople);
    ...

    return 0;
}
```

```
int readData(TPeopleList l, int *n){
    int read = 0;
    FILE *file;
    TPerson person;

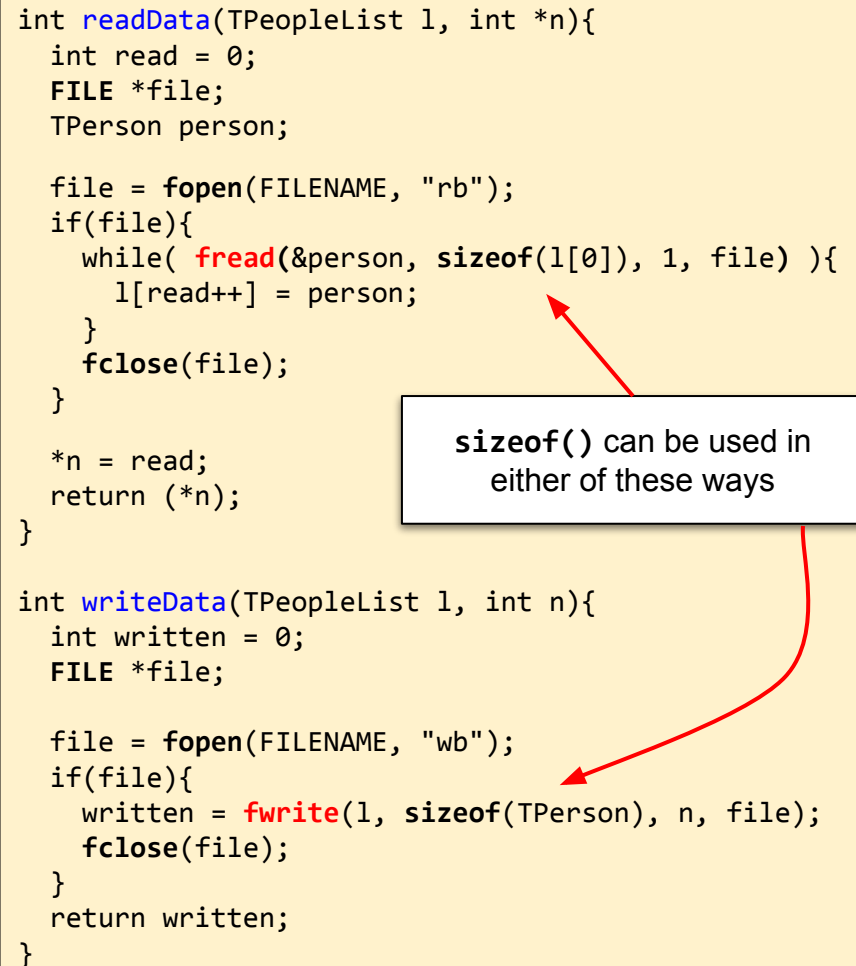
    file = fopen(FILENAME, "rb");
    if(file){
        while( fread(&person, sizeof(l[0]), 1, file) ){
            l[read++] = person;
        }
        fclose(file);
    }

    *n = read;
    return (*n);
}

int writeData(TPeopleList l, int n){
    int written = 0;
    FILE *file;

    file = fopen(FILENAME, "wb");
    if(file){
        written = fwrite(l, sizeof(TPerson), n, file);
        fclose(file);
    }
    return written;
}
```

sizeof() can be used in either of these ways



5. Functions for binary files in C

20

Basic functions

Example: Code to read a specific record from the file

```
// reads the record of the position indicated in pos.
TPerson readRecord(int pos){
    FILE *file;
    TPerson person;

    file = fopen(FILENAME, "rb");
    if(file){
        fseek(file, pos * sizeof(TPerson), SEEK_SET);
        fread(&person, sizeof(TPerson), 1, file);
        fclose(file);
    }

    return person;
}
```

```
// reads the last record in the file
TPerson readLastRecord(){
    FILE *file;
    TPerson person;

    file = fopen(FILENAME, "rb");
    if(file){
        fseek(file, -1 * sizeof(TPerson), SEEK_END);
        fread(&person, sizeof(TPerson), 1, file);
        fclose(file);
    }

    return person;
}
```