C Developer

File Handling

SUPINFO

Course Objectives

- ✓ Understand the usefulness of being able to store data other than temporarily
- ✓ Know the different ways of reading and writing in a file

Course Plan

1. Opening and Closing Files

2. Read and Write Operations

1. Opening and Closing Files

1. Opening and Closing Files

Overview

- It is necessary to keep data in memory in a **permanent** way
- For the moment, our data is only available during the execution of the program
- A file will thus be a support to keep a bulk of data

1. Opening and Closing Files

Overview

- **Text file**: representation of a file as a sequence of lines, each composed of a certain number of characters and terminated by "**\n**"

- **Binary file**: sequence of bytes, which can represent all kinds of data

1. Opening and Closing Files

Overview

#include <stdio.h>

- The program does not read/write its information directly from/to the file: it passes through a **buffer**
- This buffer is emptied automatically when it is full, or when it contains the character "**\n**"
- It can be explicitly dumped with the **fflush** function

1. Opening and Closing Files

Overview

- The memory location of the input/output buffer of a file is given by a variable of **FILE** type.

- It is a structure defined in **stdio.h**

```
struct FILE {
    char *buffer;
    char *ptr;
    int cnt;
    int flags;
    int fd;
};
```

1. Opening and Closing Files

Overview

- To access a file, we declare a pointer to the **FILE** type

- The function to open a file will assign to this pointer the address of a variable of **FILE** type containing information about the file

1. Opening and Closing Files

Opening a file

- We use the **fopen** function which returns a pointer to the **FILE** type
- To do this we pass it two arguments: the name of the file and the opening mode
- If the opening failed, the returned pointer is **NULL**
- We will check that this opening has been successful: to do so, we must just test the value of the pointer

1. Opening and Closing Files

Opening a file

- Function signature:

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

  – **filename**: is the name of the file to open, possibly prefixed by the path
  – **accessMode**: specifies both the access mode (**read**, **write**, **append**) and the opening mode (**ASCII** or **binary**)

1. Opening and Closing Files

Opening a file

- The different access modes:
  - "**r**" read: you can read the contents of the file but not write to it, the file must exist before
  - "**r+**" read extended (read and write): you can read the contents of the file and write to it, the file must exist before

➢ In both cases, if the file does not exist, **fopen** returns the NULL pointer

1. Opening and Closing Files

Opening a file

- The different access modes:
  - “**w**” write: the content of the file is deleted, you can then write but not read
  - “**w+**” write extended (read and write): the content of the file is deleted, you can then write and read

➢ In both cases, if the file does not exist, it is created

1. Opening and Closing Files

Opening a file

- The different access modes:
  - “**a**” append: the content of the file is kept, so you can write to the end of it but not read
  - “**a+**” append extended (read and append): the content of the file is kept, so you can write to the end of it and read

➢ In both cases, if the file does not exist, it is created

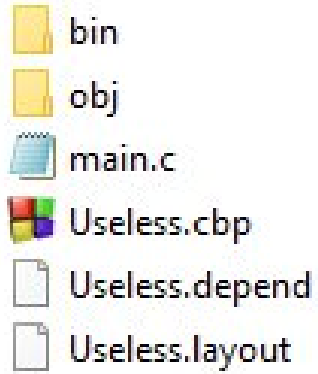
1. Opening and Closing Files

Opening a file

- The previous access modes are **ASCII** opening modes
- Their equivalents in binary opening mode are: “**rb**”, “**rb+**”, “**wb**”, “**wb+**”, “**ab**”, and “**ab+**”
- If you do not specify a path when you use the **fopen** function, your file will have to be (or will be) placed in the same directory as your project (or in the execution path)

1. Opening and Closing Files

Opening a file

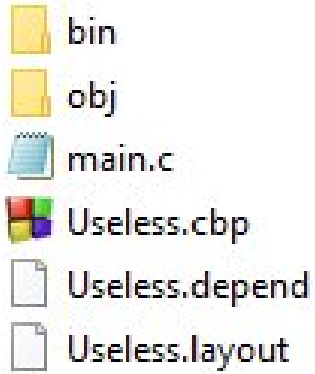



```c
#include <stdio.h>

int main()
{
    FILE *myFile = NULL;
    myFile = fopen("data.txt", "r");
    if(myFile != NULL) {
        printf("Succeeded!\n");
    } else {
        printf("Failed!\n");
    }
    return 0;
}
```

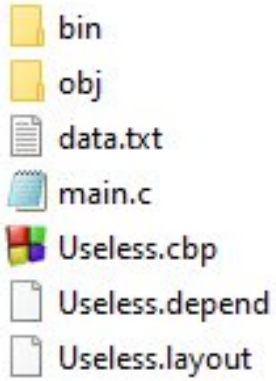
```
Failed!
```

1. Opening and Closing Files

Opening a file




```c
#include <stdio.h>

int main()
{
    FILE *myFile = NULL;
    myFile = fopen("data.txt", "w");
    if(myFile != NULL) {
        printf("Succeeded!\n");
    } else {
        printf("Failed!\n");
    }
    return 0;
}
```

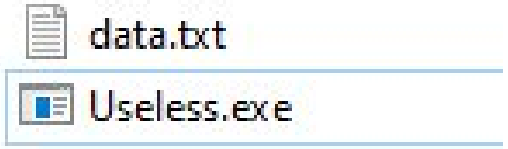


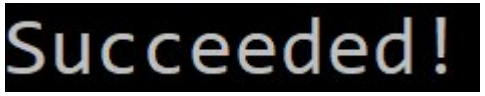

```
Succeeded!
```

1. Opening and Closing Files

Opening a file

- And if you run directly from the **\bin\Debug** executable folder:





1. Opening and Closing Files

Closing a file

- We use the **fclose** function which returns a value of **int** type, and which takes as a parameter a pointer to the **FILE** type
- If the closing has failed, the returned value is **EOF**, otherwise it is **0**
- **EOF** is a symbolic constant defined in **stdio.h**, a failure corresponds most of the time to a file already closed or non-existent
- We check that this closing was successful: it is enough to test the value of the returned integer

1. Opening and Closing Files

Closing a file

- Closing a file will empty the write buffer: useful in case of later abnormal program interruption

- This also allows to free memory by deleting the dynamic variable of **FILE** type created at the opening of the file (**fclose** calls the **free** function)

1. Opening and Closing Files

Closing a file

```c
#include <stdio.h>

int main()
{
    FILE *myFile = NULL;
    myFile = fopen("data.txt", "r");
    if(myFile != NULL) {
        printf("Succeeded!\n");
        if(fclose(myFile) == 0) {
            printf("Closed!\n");
        } else {
            printf("Not closed!\n");
        }
    } else {
        printf("Failed!\n");
    }
    return 0;
}
```

```
Succeeded!
Closed!
```

1. Opening and Closing Files

Renaming a file

- Use the **rename** function

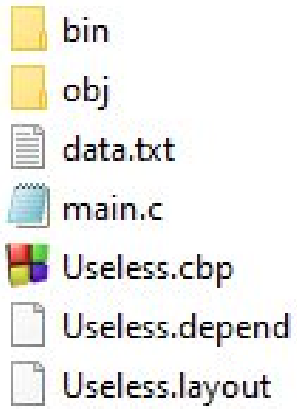
- Function signature:

```
int rename(const char *oldName, const char *newName);
```

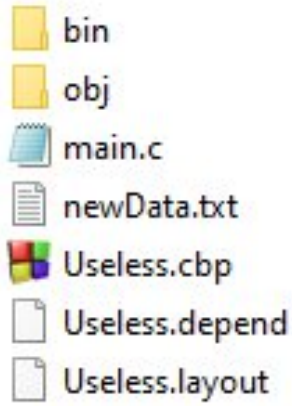
  - The parameters are the old name of the file and its new name
  - This function returns **0** if it succeeded in renaming the file and **-1** otherwise

1. Opening and Closing Files

Renaming a file




```c
#include <stdio.h>

int main()
{
    rename("data.txt", "newData.txt");
    return 0;
}
```



1. Opening and Closing Files

Removing a file

- Use the **remove** function

- Function signature:

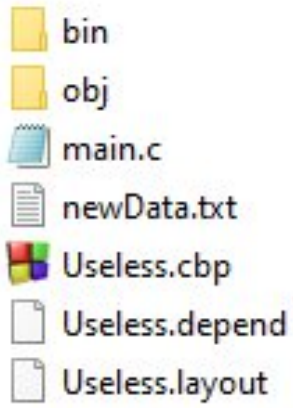
```
int remove(const char *file);
```

  - The only parameter is the name of the file to delete
  - This function returns **0** if it succeeded in removing the file and **-1** otherwise

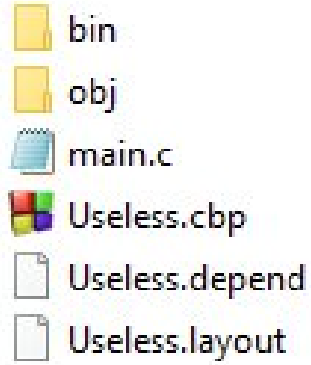
- The deletion is definitive, it is not a *move to the Recycle Bin*

1. Opening and Closing Files

Removing a file




```c
#include <stdio.h>

int main()
{
    remove("newData.txt");
    return 0;
}
```



1. Opening and Closing Files

Questions

2. Read and Write Operations

2. Read and Write Operations

Writing

- To write a character, use the **fputc** function

- Function signature:

```
int fputc(int character, FILE *file);
```

  – It takes as parameter the character to write, and a pointer to the file in which we want to write
  – The returned value is equal to the value of the written character or to **EOF** in case of failure

2. Read and Write Operations

Writing

- The failure causes can for example be an attempt to write in a closed file, or in a non-existent file

- We can therefore test the proper functioning of our operation

- The pointer in parameter is obviously the pointer to the **FILE** structure which has been returned by the **fopen** function

2. Read and Write Operations

Writing

- Writing is done at the **current position** if the file was opened in **write** mode, this position is incremented by 1 at the end of this operation (when opening, the current position is the beginning of the file)

- Writing is done at the **end** of the file if it was opened in **append** mode

2. Read and Write Operations

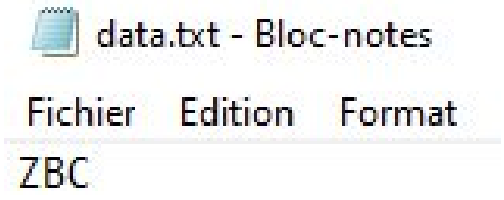
Writing

```c
#include <stdio.h>

int main()
{
    FILE *myFile = NULL;
    myFile = fopen("data.txt", "w");
    if(myFile != NULL) {
        fputc('A', myFile);
        fputc('B', myFile);
        fputc('C', myFile);
        fclose(myFile);
    } else {
        printf("Error!\n");
    }
    return 0;
}
```

2. Read and Write Operations

Writing

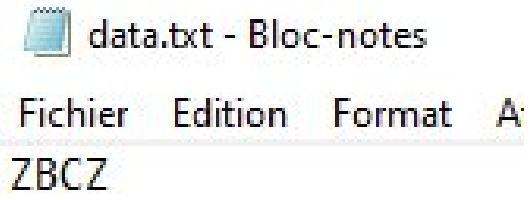
```c
#include <stdio.h>

int main()
{
    FILE *myFile = NULL;
    myFile = fopen("data.txt", "r+");
    if(myFile != NULL) {
        fputc('Z', myFile);
        fclose(myFile);
    } else {
        printf("Error!\n");
    }
    return 0;
}
```



2. Read and Write Operations

Writing

```c
#include <stdio.h>

int main()
{
    FILE *myFile = NULL;
    myFile = fopen("data.txt", "a");
    if(myFile != NULL) {
        fputc('Z', myFile);
        fclose(myFile);
    } else {
        printf("Error!\n");
    }
    return 0;
}
```



2. Read and Write Operations

Writing

- To write a string, use the **fputs** function

- Function signature:

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

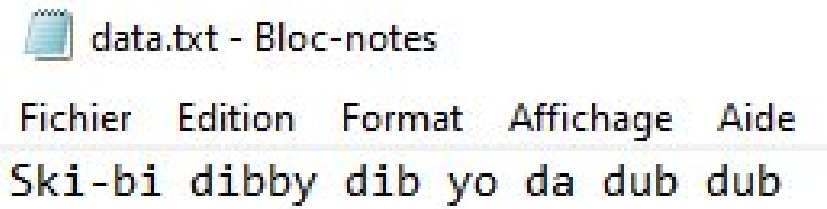
  – It takes as parameter the string to write, and a pointer to the file in which we want to write
  – The returned value is equal to the written string or to **EOF** in case of failure

- Writing is done the same way (as **fputc**) except that the position is incremented by the string size; the "**\0**" terminal is not considered

2. Read and Write Operations

Writing

```c
#include <stdio.h>

int main()
{
    FILE *myFile = NULL;
    myFile = fopen("data.txt", "w");
    if(myFile != NULL) {
        fputs("Ski-bi dibby dib yo da dub dub", myFile);
        fclose(myFile);
    } else {
        printf("Error!\n");
    }
    return 0;
}
```



2. Read and Write Operations

Writing

- To write a formatted string, use the **fprintf** function

- Function signature:

```
int fprintf(FILE *file, const char *formattedString, type para1, type para2, ...);
```

  - It takes as parameter a pointer to the file in which we want to write, the formatted string to write, and the different values
  - The returned value is equal to the number of characters written or to **EOF** in case of failure

- Writing is also done the same way (as **fputs**) and the position is incremented by the number of characters written

2. Read and Write Operations

Writing

```c
#include <stdio.h>

int main()
{
    FILE *myFile = NULL;
    int i = 123;
    myFile = fopen("data.txt", "a");
    if(myFile != NULL) {
        fprintf(myFile, " test %d test", i);
        fclose(myFile);
    } else {
        printf("Error!\n");
    }
    return 0;
}
```

data.txt - Bloc-notes

Fichier Edition Format Affichage Aide

Ski-bi dibby dib yo da dub dub test 123 test

2. Read and Write Operations

Reading

- To read a character, use the **fgetc** function

- Function signature:

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

  – It takes as parameter the file in which we want to read
  – The returned value is equal to the value of the character or to **EOF** in case of failure

2. Read and Write Operations

Reading

- Reading is done at the **current position**

- This position is incremented by 1 at the end of this operation (when opening, the current position is the beginning of the file)

2. Read and Write Operations

Reading

```c
#include <stdio.h>

int main()
{
    FILE *myFile = NULL;
    int c;
    myFile = fopen("data.txt", "r");
    if(myFile != NULL) {
        c = fgetc(myFile);
        printf("%c\n", c);
        fclose(myFile);
    } else {
        printf("Error!\n");
    }
    return 0;
}
```

S

2. Read and Write Operations

Reading

```c
#include <stdio.h>

int main()
{
    FILE *myFile = NULL;
    int c;
    myFile = fopen("data.txt", "r");
    if(myFile != NULL) {
        do {
            c = fgetc(myFile);
            printf("%c", c);
        } while(c != EOF);
        fclose(myFile);
    } else {
        printf("Error!\n");
    }
    return 0;
}
```

```
Ski-bi dibby dib yo da dub dub test 123 test
```

2. Read and Write Operations

Reading

- To read a string, use the **fgets** function
- Function signature:

```
char *fgets(char *string, int length, FILE *file);
```

  – It takes as parameter an array of char intended to receive the read string, an integer corresponding to the maximum **number of characters to read + 1**, and a pointer to the file in which we want to read
  – The returned value is equal to the read string or to **EOF** in case of failure

2. Read and Write Operations

Reading

- Reading is done the same way (as **fgetc**) except that the position is incremented by the number of read characters: equal to **length-1** or less if we met "**\n**" or **EOF**

- If we want to read 10 characters, **length** must be set to 11, in order to leave a space for the "**\0**" terminal

2. Read and Write Operations

Reading

data.txt - Bloc-notes

Fichier Edition Format Affichage Aide

```
Hello 1
Hello 2
Hello 3
```

```c
#include <stdio.h>

int main()
{
    FILE *myFile = NULL;
    char str[30];
    myFile = fopen("data.txt", "r");
    if(myFile != NULL) {
        fgets(str, 30, myFile);
        printf("%s", str);
        fgets(str, 30, myFile);
        printf("%s", str);
        fgets(str, 5, myFile);
        printf("%s", str);
        fclose(myFile);
    } else {
        printf("Error!\n");
    }
    return 0;
}
```

```
Hello 1
Hello 2
Hell
```

2. Read and Write Operations

Reading

- To read a formatted string, use the **fscanf** function

- Function signature:

```
int fscanf(FILE *file, const char *formattedString, para1, para2, ...);
```

  – It takes as parameter a pointer to the file in which we want to read, the formatted string to read, and the different variables
  – The returned value is equal to the number of characters read or to **EOF** in case of failure

- Reading is also done the same way (as **fgets**) and the position is incremented by the number of characters read

2. Read and Write Operations

Reading

data.txt - Bloc-notes

Fichier Edition Format Affichage Aide

```
10 11 12
```

```c
#include <stdio.h>

int main()
{
    FILE *myFile = NULL;
    int i, j, k;
    myFile = fopen("data.txt", "r");
    if(myFile != NULL) {
        fscanf(myFile, "%d %d %d", &i, &j, &k);
        printf("%d\n%d\n%d\n", i, j, k);
        fclose(myFile);
    } else {
        printf("Error!\n");
    }
    return 0;
}
```

```
10
11
12
```

2. Read and Write Operations

Moving in a file

- Considering the previous operations, when a file has been opened, there is a pointer that indicates the current position in the file

- We will be able to know its value, and modify it manually

2. Read and Write Operations

Moving in a file

- To get the current position in the file, use the **ftell** function

- Function signature:

```
long ftell(FILE *file);
```

  - It takes as parameter a pointer to the file
  - It returns the current value of the position indicator or **-1** in case of failure

2. Read and Write Operations

Moving in a file

data.txt - Bloc-notes

Fichier Edition Format Affichage

```
Lorem ipsum dolor sit
amet, consectetur
adipiscing elit, sed do
eiusmod tempor
incididunt ut labore et
dolore magna aliqua.
```

```c
#include <stdio.h>

int main()
{
    FILE *myFile = NULL;
    char str[10];
    myFile = fopen("data.txt", "r");
    if(myFile != NULL) {
        fgets(str, 9, myFile);
        printf("%s\n", str);
        printf("%ld\n", ftell(myFile));
        fclose(myFile);
    } else {
        printf("Error!\n");
    }
    return 0;
}
```

```
Lorem ip
8
```

2. Read and Write Operations

Moving in a file

- To set the position in the file, use the **fseek** function

- Function signature:

```
int fseek(FILE *file, long offset, int whence);
```

  – It takes as parameter a pointer to the file, an offset (positive or negative) from a reference position, and the reference position
  – It returns **0** or **-1** in case of failure

2. Read and Write Operations

Moving in a file

- **whence** constants:
  - **SEEK_SET**: beginning of the file
  - **SEEK_CUR**: current position of the file pointer
  - **SEEK_END**: end of the file

2. Read and Write Operations

Moving in a file

data.txt - Bloc-notes

Fichier Edition Format Affichage

```
Lorem ipsum dolor sit
amet, consectetur
adipiscing elit, sed do
eiusmod tempor
incididunt ut labore et
dolore magna aliqua.
```

```c
#include <stdio.h>

int main()
{
    FILE *myFile = NULL;
    char str[10];
    myFile = fopen("data.txt", "r");
    if(myFile != NULL) {
        fgets(str, 9, myFile);
        printf("%s\n", str);
        printf("%ld\n", ftell(myFile));
        fseek(myFile, -3, SEEK_CUR);
        printf("%ld\n", ftell(myFile));
        fgets(str, 9, myFile);
        printf("%s\n", str);
        fclose(myFile);
    } else {
        printf("Error!\n");
    }
    return 0;
}
```

```
Lorem ip
8
5
 ipsum d
```

2. Read and Write Operations

Moving in a file

- To go back to the beginning of the file, you can also use the **rewind** function

- Function signature:

```
void rewind(FILE* file);
```

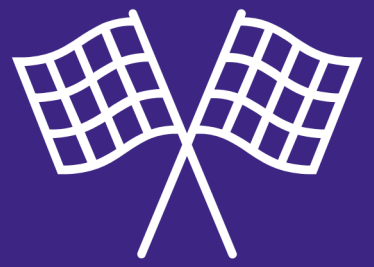
  – It takes as parameter a pointer to the file
  – Unlike the **fseek** function used with **SEEK_SET**, it does not return any value and therefore cannot be tested

2. Read and Write Operations

Questions

C Developer

File Handling



Thank you for your attention