Mila University center
Institute of Mathematics and Computer Science
1st Year LMD [redacted]

Practical Work series N° 3: Files

1. Definition :

A computer file is in the common sense, a collection, or a set of digital data (0,1) gathered under the same name, recorded on a permanent storage medium, called mass memory, such as a hard disk, a CD-ROM, flash memory, etc., and handled as a unit.

A file has **a file name** that is used to designate and access the content. This name often includes a **suffix** called the **extension**, which provides information on the nature of the information contained in the file and therefore the software that can be used to manipulate it.

2. File types:

In programming, there are mainly two types of files:

- **The text file:** They are made up of a series of characters forming a text (character string). They are used to record texts but also numerical values with a view to exchanging them with other software. They are readable by a simple text editor.
- **The binary file:** containing data in the form **of bytes** which therefore only have meaning for the software that uses them, this type of files is **unreadable by a text editor** , it is made up of a collection **of records** , each record containing a collection of logical units of information also called **fields** .

3. Handling binary files:

Most current programming languages, **particularly C++** , have instructions for manipulating files. These instructions can be classified as follows:

- Opening and creating a file,
- Closing a file
- Reading and writing records from the file,
- Positioning in the file,
- End of file detection.

Noticed :

To use a physical file **F** in a program, this program had to include a file variable **f** . The association between **f** and **F** will therefore be carried out by means of a process called **assignment** , such that the modifications made to **f** in the program will directly affect **F** on its support .

In C++ language, this **step is integrated** into the file opening.

3.1 Opening and creating a file

To open a file in C++, we use the predefined " **fopen" function** ,

The syntax is: **FILE* fopen(** " *file-name* **",** " *mode* " **);**

FILE*: the return value of the function is a **pointer** to the file type.

" *file-name* ": The first argument to fopen is the name of the file concerned, provided in the form of a character string (exp: " f.txt ").

" *mode* " **:** The second argument, ***mode*** , is a character string which specifies the file access mode, the table below shows the different modes:

The mode	interpretation	If the file exists	If does not exist
"r"	Open for reading	Read from the beginning	Error
"w"	Open for writing	Overwrite content	Create file
"a"	Open read/write	Write at the end	Create file
"r+"	Open read/write	Read from the beginning	Error
"w+"	Open read/write	Overwrite content	Create file
"a+"	Open read/write	Write at the end	Create file

Example: open for reading a file named “data.txt” which is located in the C partition of the hard drive.

File *f; // declare a pointer to a file

Char name = "C:\data.txt"; // a string contains the path

f = fopen(name,"r");

Note: the fopen function in this case returns the address of the FILE structure associated with the file. It returns **NULL** if it cannot open the file.

Therefore: Testing the value returned by fopen is essential to prevent errors: non-existent file, defective or saturated physical media, excessive number of open files, etc.

Here is a complete example:

```
# include <iostream>
using namespace std;

char name[6]= "p.txt";
FILE *f;
f = fopen(name,"r"); // open the file for reading
if (f == NULL) // tests if there is an opening problem
cout << "error opening file " << name << endl; // show error
else // successful opening
{
// here we read data from the file
}
```

3.2 Closing a file

To close a file in C++, we use the predefined function “ **fclose”** ,

The syntax is: **fclose** (`FILE*`);
Example: fclose(f); // close file, f is a pointer to a file.

Noticed :
It is essential to close a file before the end of the program that uses it to avoid data loss.

3.3 Reading and writing from a file

After opening the file, several possibilities are offered: read the information it contains, modify some of it, delete some of it, or add others.

a) Reading: To read data, we use the **fread() function** as follows:

int **fread** (void* **adr_buffer** , int **element_size** , int **number_elements** , FILE* **file**);

- **fread** function returns the number of elements read.
- **adr_buffer:** the address of the variable which serves as a buffer where to store the data to be read.
- **element_size:** the size in bytes of an element
- **nb_elements:** an integer which specifies the number of elements that we will read
- **file** : The pointer to the file to read

Example: nb_lu = fread(&b, sizeof(int) , 1 ,f); // reading an element of integer type from the file pointed to by f, and putting the data read in the variable b which is of integer type.

Important note: this function allows us to read one or more elements of the file, so to read all the file we must repeat the execution of this function until the end of the file, to detect the end of the file, we have two ways :

1) the fread() function returns the number of elements actually read. If the returned value differs from the number of elements to read, it is because the **end of the file has been encountered** . We use it as a condition of a while loop.

2) the second is to use the predefined function **feof(),** it returns the NULL value if it is the end of the file, and another value otherwise.

b) Writing: To write data, we use the **fwrite() function** as follows (it is similar to reading):

int **fwrite** (void* **adr_buffer** , int **element_size** , int **element_number** , FILE* **file**);

c) Modification: to modify an element of a file, you must position the cursor on this element, then you overwrite its values with the new ones.

So, we read the file, element by element using the **fread() function** up to the target element.

fseek function allows you to position the cursor at the desired position.

Syntax: int **fseek (** FILE * **Stream,** long **Offset,** int **Origin);** Or :

- **fseek:** returns a **0** if the operation is successful, **another value** otherwise
- **Stream:** this is the pointer to the file
- ***Offset*** is the number of bytes of the move, counted algebraically from ***Origin*** .
- ***Origin*** is a constant which is **SEEK_SET** ("from the beginning of the file") or **SEEK_END** ("from the end of the file") or, **SEEK_CUR** ("from the current position").

Application example:

Here is a program that contains three functions:

- The first allows you to write a certain number of points in a file, each point contains a name, and the X, and Y coordinates.
- The second function allows you to read the values of these points.
- The third modify the coordinates of a point.

First of all, we must define a **Point structure** which contains 3 fields.

```
#include <iostream>
//#include <stdio.h>
using namespace std;
// the definition of the Point type
struct Point
{
char n[5];
float X,Y;
};

const int MAX = 10; // the maximum number of points to enter
char name[6]= "p.txt"; // character string which contains the name of the file
Point v[MAX];
FILE *f; // the pointer to a file
//-------------the GrabPoint function-----------
Point GrabPoint()
{
Point P;
cout << "Enter the point name ";
cin >> Pn;
cout << "Enter the coordinates of the point ";
cin >> P.X >> P.Y ;
return P;
}
//-------------the write function----------------------------
void write (FILE *f)
{

int i,m;

cout << ">>>>> insert points into file:"<< name << endl;
cout << "how many points do you want to enter:";
cin >> m;

if ((f = fopen(name,"w")) == NULL) // open the file for writing
cout << "error opening (creating) file " << name << endl;
else
{
for (i=0; i<m; i++) // we fill the vector
v[i] = GrabPoint(); // call to the EnterPoint function
fwrite(v,sizeof(Point),m,f); // write v to the file
fclose(f); // close the file
}
}
//---------------- The read function -------------------
void read (FILE* f)
{
int nb_lu,i;
Point ww;
cout << ">>>>>>>>>>reading points from the file: " << name << endl;
if ((f = fopen(name,"r")) == NULL) // open file for reading
cout << "error opening file " << name << endl;
else
{
```

```
// while the end of the file is not reached
while ( (nb_lu = fread(&ww,sizeof(Point),1,f)==1))
{
// the fread function is integrated into the loop condition
cout << ww.n << " -> " << ww.X <<" " <<ww.Y<<endl; // display of the point read

}
fclose(f); // close the file
}
}
//--------------the modify function-------------------------
void modify (FILE* f)
{
char val[5],;
Point p[0],n_val[0];
bool find;
cout << ">>>>>>modification of a point in the file: " << name << endl;
if ((f = fopen(name,"r+")) == NULL) // open the file in "update"
cout << "error opening file " << name << endl;
else
{
cout << "Enter the value to update?" << endl;
cin >> val;
find = false;
//while is not the end of the file and val is not found
while ((! feof(f)) && (! find)) // search for val
{
fread(&p,sizeof(Point),1,f); // read a point
if (strcmp(val,p[0].n)==0) // compare with val
find = true;
}
if (find) // if val exists, so the cursor is on the next point
{

fseek(f,-sizeof(Point),SEEK_CUR); // position the cursor on the previous point

cout << "Enter the new value?" << endl;
cin >> n_val[0].n;

cout << “Enter the new X coordinate?” << endl;
cin >> n_val[0].X;
cout << “Enter the new Y coordinate?” << endl;
cin >> n_val[0].Y;

fwrite(&n_val,sizeof(Point),1,f); // insert the new point
}
else cout << val << "not found" << endl;
fclose(f); // close the file
}

}
//-----------------------------------------
int main()
{
// calls to functions
write(f);
read (f);
modify(f);
read (f); // to display the modifications

system("PAUSE");
}
```

Exercise:

You are asked to create a **student management system** . This system allows us to:

- Enter a student's data; Each student is identified by his/her: **last name** , **first name** , **date of birth** , **group** , **notes** for algorithmic, algebra, and analysis.
- display a student's data
- View student data in a group.
- Edit student data.