Universidad
Carlos III de Madrid

Industrial Informatics I Lab Session

Bachelor in Industrial Electronics and Automation

2016-2017

Teaching staff :

Juan Carlos Gonzalez Victores

Mohamed Abderrahim

Lab 7 – Files

1. Opening and closing files

In this lab session we are going to look at how to work with system files. Keep in mind that I/O (Input/Output) files are just a special case of the general I/O system, and therefore what we see here can also be applied to any data stream connected to other types of device.

To work with files in C++, it is necessary to include the header file: `fstream`.

The first thing to do to open a file is to create an object of file class depending on the "Direction" of data flow required [i.e, read from a file, write into a file, or both]. There are three types:

- Input: `ifstream`
- Output: `ofstream`
- Input/Output: `fstream`

A file can be associated to the file class object by using the function `open()`or directly through the corresponding constructor, to which the file path and open mode is passes as parameters. The mode is determined by one or more of a series of the following values:

- `ios::app` → To add data to the end of the file.
- `ios::ate` → To be positioned at the end of the file.
- `ios::in` → Indicates file to be used for input.
- `ios::out` → Indicates file to be used for output.
- `ios::binary` → Binary mode (default is text mode).
- `ios::trunc` → Changes the file size to 0 by deleting any data it may contain.

The above mentioned functions are assigned with default modes, and so it may not be necessary to mention the mode as a parameter.

In the following code snippet, you can see an example of how to use it:

```
#include <iostream>
#include <fstream>

Using namespace std;

int main(intargc, char *argv[]) {
      ifstream input("input.txt"); //reading file in text mode
      if(!input){
            cerr << "Error opening input file!" << endl;
      }

      ofstream output;
      //writing file in binary mode
      output.open("output.bin", ios::out | ios::binary);
      if(output.is_open()) { //check if file is opened
            cerr << "Output file ready" << endl;
      }

      // ... do things with files ...
```

```
    input.close();
    output.close();

    return 0;
}
```

2. Reading and writing text files

To read from and write into text files, you can directly use operators ">>" and "<<" respectively, just like you did so on a console. For example you could do something like:

```
output << "Writing into output file using outflow." << endl;
```

Exercise

Write a program that asks the name and surname of several people from the user, and writes the same to a file using one line for each individual data. Then write another program that reads and displays the content of this file on the screen.

How to detect end of file? You can use the function `eof()` (end of file) of the corresponding flow, or be aware that when you reach the end of the file, the associated flow returns `false`.

Another interesting function that allows you to read data from a file is `getline()`, whose prototypes are:

```
istream&getline(char *buf, streamsize num);
istream&getline(char *buf, streamsize num, char delim);
```

These functions read characters from the stream and stores them in an array pointed to by `buf` until `num-1` characters have been read, or until it encounters an end-of-line (or the specified `delim` character, as in the second prototype), or until it reaches the end of file. The delimiter character is extracted and added to `buf`.

The `get` function has two forms that function very much like the `getline` function. The prototypes are:

```
istream & get(char *buf, streamsize num);
istream & get(char *buf, streamsize num, char delim);
intget();
```

The only difference between the first two prototypes of `get` and that of the `getline` function is that the `get` functions do not discard the delimiter, and continues by reading the next character. On the other hand, `get` just returns the next character, and returns `EOF` on reaching the end of the file.

Write a program for reading the previously created *`Names`* file, and display its contents on the screen, line by line.

3. Binary files

Text files are not always the most effective format. Also, you may need to write data with no format, i.e. raw (1s and 0s). For these reasons it is necessary to work with binary files.

To work with binary files you need to open the file in binary mode, using `ios::binary`.

Reading and writing can be done byte by byte (char by char) using functions `get()` and `put()`.The same can be done in blocks by using functions `read()` and `write()`.which are used for reading and writing, respectively, `num` amount of characters starting from the location pointed to by `buf`.

```
istream&read(char *buf, streamsizenum);
ostream&write(char *buf, streamsizenum);
```

Exercise

Write a program that writes all the characters between 0 and 255, one character at a time, into a binary file. Then read this file, and display on the screen, all the even numbers.

4. Random access

In C++ there are always two pointers associated with files: One `get` pointer, that points to the position from where the next input operation (reading) should be performed, and one `put`, pointer, as you may guess, that specifies the position from where the next output operation (write) should be performed.

There exists two functions `seekg` y `seekp`, which are used to modify these pointers:

```
istream & seekg(off_type offset, seekdir origin);
ostream & seekp(off_type offset, seekdir origin);
```

These function moves their respective pointers `offset` amount of characters from the position indicated by origin. The position parameter origin may be one of the following values:

- `ios::beg` → The beginning of the file.
- `ios::cur` → The current position.
- `ios::end` → The end of the file.

Random access operations to a file are usually utilized in binary operations, and special care must be taken as it could easily result in data synchronized error in the file.

To obtain the current position of the pointers (`get` or `put`) you could use the following functions:

```
pos_typeseekg();
pos_typeseekp();
```

The results of the above functions can be applied directly to functions `seekg` and `seekp`, respectively, in the following way:

```
istream & seekg(pos_type pos);
istream & seekp(pos_type pos);
```

Exercise

Write a program that saves, in binary format, integer values between 0 and 10 in a file. Then read the same file and print the content of the file on the screen. Finally, move the `get` *pointer of the file one character from the beginning of the file, then read and display the content of the file again. What result do you obtain? Why?*

Exercise

Let us continue to extend the functionality of the Computer Simulator project. In this lab session, you have to create two new classes `InputFile` and `OutputFile`, which are inherited from classes `Input` and `Output` respectively.

`InputFile` will read a text file and send its content to the corresponding Processor object. In turn, create two more classes inherited from the class `InputFile`, , where one class would send data line by line, and the other class would send data word by word.

`OutputFile` will write strings into a text file, appending each line to the end of the file. The format of each line written should be of the form `[date+time] [received string]`.

To obtain the date and time information from the system, you can use code similar to the following:

```
#include <time.h>
#include <string.h> // strtok: used to remove the line scape (‘\n’)

...

time_t rawtime;
struct tm *timeinfo;
time(&rawtime);
timeinfo = localtime(&rawtime);
cout << “TIME: “ << strtok(asctime(timeinfo), ”\n”) << endl;
```

The output file must look something like this:

```
MonSep 09 18:39:05 2013:Processed by Uppercaseprocessor: GHOST
MonSep 09 18:39:05 2013:Processed by Uppercaseprocessor: IN
MonSep 09 18:39:05 2013:Processed by Uppercaseprocessor: THE
MonSep 09 18:39:05 2013:Processed by Uppercaseprocessor: HOUSE,
MonSep 09 18:39:05 2013:Processed by Uppercaseprocessor: LIMPBIZKET
MonSep 09 18:39:05 2013:Processed by Uppercaseprocessor: IS
MonSep 09 18:39:05 2013:Processed by Uppercaseprocessor: IN
MonSep 09 18:39:06 2013:Processed by Uppercaseprocessor: THE
MonSep 09 18:39:06 2013:Processed by Uppercaseprocessor: HOUSE
```