Object Oriented Programming Lab

Lab #05: File Handling in C++

Agenda

- Why use File Handling?
- How to Achieve File Handling?
- Functions Used in File Handling.
- File Opening modes.

File Handling in C++

File Handling concept in C++ language is used for store a data permanently in computer. Using file handling we can store our data in Secondary memory (Hard disk).

File: A named collection of data, stored in secondary storage (typically).

Typical operations on files:

– **Open**

– **Read**

– **Write**

– **Close**

Why use File Handling

- For permanent storage.
- The transfer of input - data or output - data from one computer to another can be easily done by using files.

For read and write from a file you need another standard C++ library called **fstream**, which defines three new data types:

Datatype	Description
ofstream	This is used to create a file if it doesn't exists and write data on files
ifstream	This is used to read data from files
fstream	This is used to both read and write data from/to files

These classes are derived directly or indirectly from the classes istream, and ostream. We have already used objects whose types were these classes: cin is an object of class istream and cout is an object of class ostream. Therfore, we have already been using classes that are related to our file streams. And in fact, we can use our file streams the same way we are already used to use cin and cout, with the only difference that we have to associate these streams with physical files. Let's see an example:

```cpp
// basic file operations
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile;
  myfile.open ("example.txt");
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
}
```

```
[file example.txt]
Writing this to a file
```

This code creates a file called example.txt and inserts a sentence into it in the same way we are used to do with cout, but using the file stream myfile instead. But let's go step by step:

How to achieve File Handling

For achieving file handling in C++ we need follow following steps

- Naming a file
- Opening a file
- Reading data from file
- Writing data into file
- Closing a file

Functions use in File Handling

Function	Operation
open()	To create a file if it doesn't exists and to open an existing file
close()	To close an existing file
get()	Read a single character from a file
put()	Write a single character into file.
read()	Read block of binary data from file
write()	Write block of binary data into file.
eof()	Returns true if a file open for reading has reached the end.
Is_open()	Returns true if a file is opened successfully otherwise returns false

Open a file

The function **open()** can be used to open multiple files that use the same stream object.

In order to open a file with a stream object we use its member function **open():**

open (filename, mode);

Where **filename** is a null-terminated character sequence of type const char * (the same type that string literals have) representing the name of the file to be opened, and **mode** is an optional parameter with a combination of the different input modes.

Example

```
ofstream outfile;     // create stream
outfile . open ("data1.txt"); // connect stream to data1
```

To check if a file stream was successful opening a file, you can do it by calling to member **is_open()** with no arguments. This member function returns a **bool** value of true if file opening is successful, or false otherwise:

```
If(outfile.is_open()) {                    //OK, proceed to next          }
```

Closing a File

A file must be close after completion of all operation related to file. For closing file we need **close()** function.

Syntax

```
outfile.close();
```

File Opening mode

Mode	Meaning	Purpose
ios :: out	Write	Open the file for write only.
ios :: in	Read	Open the file for read only.
ios :: app	Appending	Open the file for appending data to end-of-file.
ios :: ate	Appending	take us to the end of the file when it is opened.
ios::binary	Binary File	Open a file in binary mode.
ios::trunc	Deleting previous data & Replacing with new data	If the file opened for output operations already existed before, its previous content is deleted and replaced by the new one.

Both **ios :: app** and **ios :: ate** take us to the end of the file when it is opened. The difference between the two parameters is that the **ios :: app** allows us to add data to the end of file only, while **ios :: ate** mode permits us to add data or to modify the existing data anywhere in the file.

The mode can combine two or more parameters using the bitwise OR operator (symbol |)

Example

```
fstream file;
file.Open("data . txt", ios :: out | ios :: in);
```

Each one of the open() member functions of the classes ofstream, ifstream and fstream has a default mode that is used if the file is opened without a second argument:

class	default mode parameter
ofstream	ios::out
ifstream	ios::in
fstream	ios::in | ios::out

For ifstream and ofstream classes, ios::in and ios::out are automatically and respectively assumed, even if a mode that does not include them is passed as second argument to the open() member function.

The default value is only applied if the function is called without specifying any value for the mode parameter. If the function is called with any value in that parameter the default mode is overridden, not combined.

Text files

Text file streams are those where we do not include the **ios::binary** flag in their opening mode. These files are designed to store text and thus all values that we input or output from/to them can suffer some formatting transformations, which do not necessarily correspond to their literal binary value.

Data output operations on text files are performed in the same way we operated with cout:

```
// writing on a text file
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile ("example.txt");
  if (myfile.is_open())
  {
    myfile << "This is a line.\n";
    myfile << "This is another line.\n";
    myfile.close();
  }
  else cout << "Unable to open file";
  return 0;
}
```

```
[file example.txt]
This is a line.
This is another line.
```

Data input from a file can also be performed in the same way that we did with **cin:**

```
// reading a text file
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main () {
  string line;
  ifstream myfile ("example.txt");
  if (myfile.is_open())
  {
    while (! myfile.eof() )
    {
      getline (myfile,line);
      cout << line << endl;
    }
    myfile.close();
  }

  else cout << "Unable to open file";

  return 0;
}
```

```
This is a line.
This is another line.
```

This last example reads a text file and prints out its content on the screen. Notice how we have used a new member function, called **eof()** that returns true in the case that the end of the file has been reached. We have created a while loop that finishes when indeed **myfile.eof()** becomes true (i.e., the end of the file has been reached).

Tasks

1. Copy the given paragraph into a file and compute the following things:

    - Count the total number of characters including blanks, dots etc. from file?
    - Count the lower case and upper case letters separately?
    - Count the total number of lines?
    - Count the total number of words?
    - Count the total number of lines from file?
    - Copy the data from one file to another file and also display on console?

International business entities, which are interested in economic results achieved by some local companies, may receive economic summaries containing anonymous data relating to certain employees, and they may also receive – as part of contractually guaranteed cooperation (i.e., in particular make random collections of) individual data about specific employees in relation to their performances. Any potential economic interest of such international entities, however, is not relevant in respect of the legal position of employees employed in the Czech Republic because, in keeping with the Labour Code, the employer is always authorised to assess employees' conduct and evaluate their performances (both of those activities undoubtedly also involve processing of personal data). Therefore, controllers intending to transfer personal data of employees abroad should be recommended to take into consideration also the following criteria:

2. A file named as "students.txt" contains the student first name, last name and marks of an CP course. Write a program to read the data from file and show the following results on console;
   a. Student name with Highest & Lowest marks?
   b. Average marks?

"Student.txt"

```
Duckey Donald 85
Goof Goofy 89
Brave Balto 93
Snow Smitn 93
Alice Wonderful 89
Samina Akthar 85
Simba Green 95
Donald Egger 90
Brown Deer 86
Johny Jackson 99
Greg Gupta 75
```