**Logo, company name

Description automatically generated**

**Department of (Computer Science)**

**Pak-Austria** Fachhochschule**: Institute of Applied Sciences and Technology, Haripur, Pakistan**

**COMP-112L Object Oriented Programming Lab**

**Lab Journal**

**Class: BS Computer Science**

**Name: Ahmed Raza**

**Registration No.: B20F0436CS031**

**Semester: 3rd**

**Submission Date: 08 February, 2022**

**Submitted to: Engr. Rafi-Ullah**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Instructor Signature**

**Lab No. 12**

**File Handling in C++**

**Objectives:**

In this lab we will be discussing about File Handling in detail. One of the most important concepts in object-oriented programming is that of inheritance. In C++, files are mainly dealt by using three classes fstream, ifstream, ofstream available in fstream headerfile.

**Ofstream: Stream class to write on files**

**Ifstream: Stream class to read from files**

**fstream:** Stream class to both read and write from/to files.

**Tools/Software Required:**

* All the tasks are implemented on DEV C++.

**Introduction:**

**Open a file:**

The first operation generally performed on an object of one of these classes is to associate it to a real file. This procedure is known as to *open a file*. An open file is represented within a program by a stream and any input or output operation performed on this stream object will be applied to the physical file associated to it.

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 flags.

All these flags can be combined using the bitwise operator OR (|). For example, if we want to open the file example.bin in binary mode to add data we could do it by the following call to member function open():

ofstream myfile;

myfile.open ("example.bin", ios::out | ios::app | ios::binary);

**Closing a file:**

When we are finished with our input and output operations on a file we shall close it so that its resources become available again. In order to do that we have to call the stream's member function close(). This member function takes no parameters, and what it does is to flush the associated buffers and close the file:

**myfile.close();**

Once this member function is called, the stream object can be used to open another file, and the file is available again to be opened by other processes.

In case that an object is destructed while still associated with an open file, the destructor automatically calls the member function close().

**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.

**Checking state flags**

In addition to eof(), which checks if the end of file has been reached, other member functions exist to check the state of a stream (all of them return a bool value):

**bad()**

Returns true if a reading or writing operation fails. For example in the case that we try to write to a file that is not open for writing or if the device where we try to write has no space left. Page 101

**fail()**

Returns true in the same cases as bad(), but also in the case that a format error happens, like when an alphabetical character is extracted when we are trying to read an integer number.

**eof()**

Returns true if a file open for reading has reached the end.

**good()**

It is the most generic state flag: it returns false in the same cases in which calling any of the previous functions would return true.

In order to reset the state flags checked by any of these member functions we have just seen we can use the member function clear(), which takes no parameters.

**get and put stream pointers:**

All i/o streams objects have, at least, one internal stream pointer:

ifstream, like istream, has a pointer known as the *get pointer* that points to the element to be read in the next input operation.

ofstream, like ostream, has a pointer known as the *put pointer* that points to the location where the next element has to be written.

Finally, fstream, inherits both, the get and the put pointers, from iostream (which is itself derived from both istream and ostream).

These internal stream pointers that point to the reading or writing locations within a stream can be manipulated using the following member functions:

***tellg() and tellp()***

These two member functions have no parameters and return a value of the member type pos\_type, which is an integer data type representing the current position of the get stream pointer (in the case of tellg) or the put stream pointer (in the case of tellp).

***seekg() and seekp()***

These functions allow us to change the position of the get and put stream pointers. Both functions are overloaded with two different prototypes. The first prototype is:

**seekg ( position ); seekp ( position );**

Using this prototype the stream pointer is changed to the absolute position position (counting from the beginning of the file). The type for this parameter is the same as the one returned by functions tellg and tellp: the member type pos\_type, which is an integer value.

The other prototype for these functions is:

**seekg ( offset, direction ); seekp ( offset, direction );**

Using this prototype, the position of the get or put pointer is set to an offset value relative to some specific point determined by the parameter direction. offset is of the member type off\_type, which is also an integer type.

**Lab Tasks:**

**Task # 01:**

Declare a 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..

**Code:**

**// basic file operations**

**#include <iostream>**

**#include <fstream>**

**using namespace std;**

**int main ()**

**{**

**ofstream myfile;**

**myfile.open ("example.txt");**

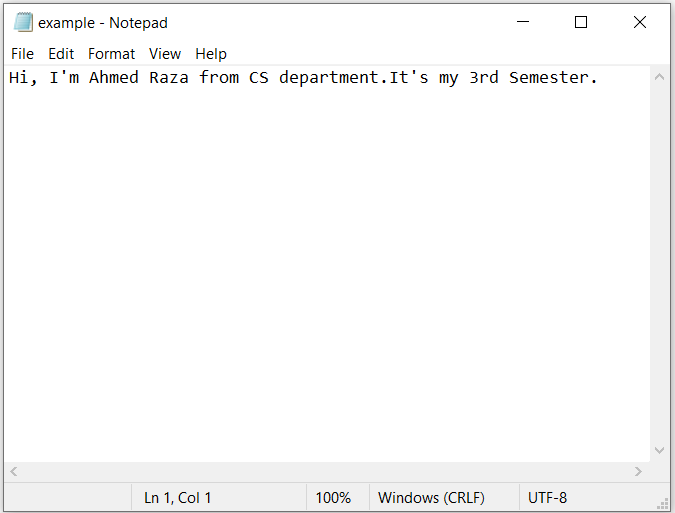
**myfile << "Hi, I'm Ahmed Raza from CS department.It's my 3rd Semester.\n";**

**myfile.close();**

**return 0;**

**}**

**Output:**

****

**Task #02:**

First create pointer and then create file by using pointer and then open file write something and then check if file is opening or not after that also print the text on the console screen.

**Code:**

**(1)**

**// 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;**

**}**

**(2)**

**// 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();**

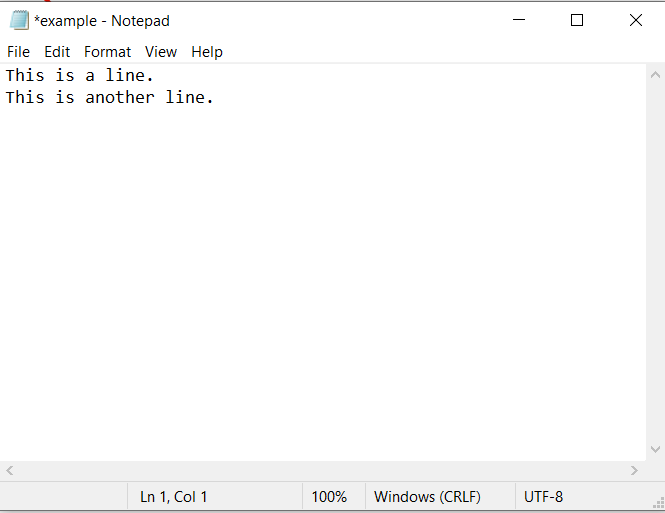
**}**

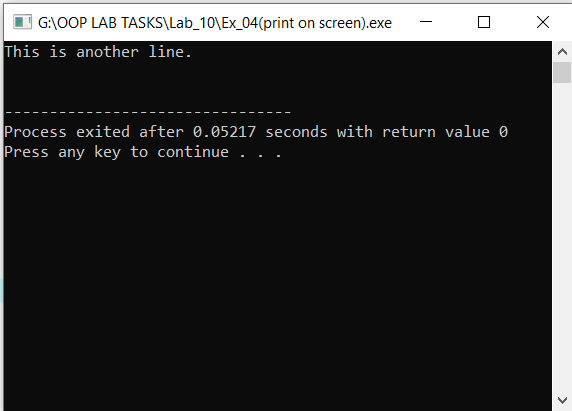
**else cout << "Unable to open file";**

**return 0;**

**}**

**Output:**

****

****

**Task #03:**

Write a code that tell the size of the file that you created.

**Code:**

**// obtaining file size**

**#include <iostream>**

**#include <fstream>**

**using namespace std;**

**int main () {**

**long begin,end;**

**ifstream myfile ("example.txt");**

**begin = myfile.tellg();**

**myfile.seekg (0, ios::end);**

**end = myfile.tellg();**

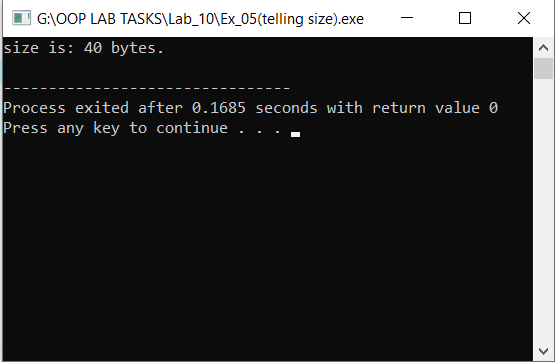
**myfile.close();**

**cout << "size is: " << (end-begin) << " bytes.\n";**

**return 0;**

**}**

**Output:**

****

**Results & Observations:**

In this Lab I’ve learned about the concept of file handling & also I understand that how to create file and open file & close file. After that I’ve learned that how to write a file. Then I understand to find the size of that created file. At last I learned that how to print the text on the console screen that’s already saved in the file.