Lab 2: Exception Handling

Information

Topics:	Exception handling, file streams
Turn in:	All source files (.cpp and .hpp), question document as .txt file.

Getting started

You will start with the following code, which contains an object that handles file streams, as well as the main source file that contains tests. At the moment, it should compile and run, but it will crash during the tests because the RecordManager object doesn't have error checking.

In this lab, you will implement the exception handling within the RecordManager, as well as checking for exceptions from the main file.

The following code is the starter code that you will work with; you can download the starter code, or type it out manually. There are descriptions for how the functions work.

Once entered, the next part of the lab is to add the error checking and exception handling.

Starter code

lab2_RecordManager.hpp

The RecordManager contains an array of output-file-streams (`ofstream`). When the `OpenOutputFile` function is called to open a new file, a free spot in the array is found and its index is used for opening the file, writing to the file, and closing the file.

```
#ifndef _RECORD_MANAGER_HPP
#define _RECORD_MANAGER_HPP

#include <string>
#include <iostream>
#include <fstream>
#include <stdexcept>
using namespace std;

class RecordManager
{
    public:
    ~RecordManager();

    void OpenOutputFile( string filename );
    void CloseOutputFile( string filename );
    void WriteToFile( string filename, string text );
    void DisplayAllOpenFiles();

    private:
    int FindAvailableFile();
    int FindFilenameIndex( string filename );

    ofstream m_outputs[5];
    string m_filenames[5];
    const int MAX_FILES = 5;
};

#endif
```

lab2_RecordManager.cpp

Destructor The destructor `~RecordManager` is a function that gets called automatically once the object is destroyed. This is a good place to close any currently-open file streams, if any are open.

Include the hpp file!
Don't forget to include the header file!

`#include "lab2_RecordManager.hpp"`

```
RecordManager::~RecordManager()
{
    for ( int i = 0; i < MAX_FILES; i++ )
    {
        if ( m_outputs[i].is_open() )
        {
            m_outputs[i].close();
        }
    }
}
```

OpenOutputFile When the user wants to open a file, they pass in the filename to open. Via the member function, `FindAvailableFile`, an available index is found. Then, the file strema at that position is opened for usage.

If the `FindAvailableFile` function cannot find an available index (that is, all files in the array are currently open), then it will return -1. For this function, trying to access `m_outputs[ index ]` will cause the program to crash.

```
void RecordManager::OpenOutputFile( string filename )
{
    int index = FindAvailableFile();
    m_outputs[ index ].open( filename );
    m_filenames[ index ] = filename;
}
```

CloseOutputFile This function receives the filename that it should close. Then, this function calls `FindFilenameIndex` to search through the array of filenames, and if it is found, the index is returned. If the filename is not found in the array, then -1 is returned. Again, if `m_outputs[ index ]` is accessed when index is -1, the program will crash.

```cpp
void RecordManager::CloseOutputFile( string filename )
{
    int index = FindFilenameIndex( filename );
    m_outputs[ index ].close();
    m_filenames[ index ] = "";
}
```

WriteToFile This function takes in a filename to be written to, as well as some text to write out. Like with `CloseOutputFile`, it uses `FindFilenameIndex` to find an index to work with. Then, it writes the `text` to that output file.

```cpp
void RecordManager::WriteToFile( string filename, string
    text )
{
    int index = FindFilenameIndex( filename );
    m_outputs[ index ] << text << endl;
}
```

DisplayAllOpenFiles This function iterates through the list of filenames and outputs them, if the corresponding file stream is open.

This function doesn't cause a crash.

```cpp
void RecordManager::DisplayAllOpenFiles()
{
    cout << "Open files: " << endl;
    for ( int i = 0; i < MAX_FILES; i++ )
    {
        if ( m_outputs[i].is_open() )
        {
            cout << i << ". " << m_filenames[i] << endl;
        }
    }
}
```

FindAvailableFile This function iterates through all the files, and for the first element it finds that isn't open, it will return its index for usage by other functions in the class. If all files are open (none are available), -1 is returned.

This function doesn't cause a crash.

```
int RecordManager::FindAvailableFile()
{
    for ( int i = 0; i < MAX_FILES; i++ )
    {
        if ( m_outputs[i].is_open() == false )
        {
            return i;
        }
    }
    return -1;
}
```

FindFilenameIndex This function will search for a filename in the array that matches the one passed in as input. If that filename is found, it returns the corresponding index. Otherwise, it returns -1.

This function doesn't cause a crash.

```
int RecordManager::FindFilenameIndex( string filename )
{
    for ( int i = 0; i < MAX_FILES; i++ )
    {
        if ( m_filenames[i] == filename )
        {
            return i;
        }
    }
    return -1;
}
```

lab2_main.cpp

The main file contains 5 tests, each called from main(). Initially, the program will crash while running the tests, but as you implement the error checking and exception handling, it should run through all 5 tests.

```cpp
#include "lab2_RecordManager.hpp"

#include <iostream>
using namespace std;

void Test1()
{
    cout << endl << "-------------------------------------" << endl;
    cout << "TEST 1: Open one file and write to it" << endl << endl;
    RecordManager record;
    record.OpenOutputFile( "Test1.txt" );

    record.DisplayAllOpenFiles();

    record.WriteToFile( "Test1.txt", "Hello world!" );

    record.CloseOutputFile( "Test1.txt" );

    cout << endl << "END OF TEST 1" << endl;
}

void Test2()
{
    cout << endl << "-------------------------------------" << endl;
    cout << "TEST 2: Open 5 files and write to them" << endl << endl;
    RecordManager record;
    record.OpenOutputFile( "Test2_A.txt" );
    record.OpenOutputFile( "Test2_B.txt" );
    record.OpenOutputFile( "Test2_C.txt" );
    record.OpenOutputFile( "Test2_D.txt" );
    record.OpenOutputFile( "Test2_E.txt" );

    record.DisplayAllOpenFiles();

    record.WriteToFile( "Test2_A.txt", "ABCDE" );
    record.WriteToFile( "Test2_B.txt", "FGHIJ" );
    record.WriteToFile( "Test2_C.txt", "KLMNO" );
    record.WriteToFile( "Test2_D.txt", "PQRST" );
    record.WriteToFile( "Test2_E.txt", "UVWXYZ" );

    record.CloseOutputFile( "Test2_A.txt" );
    record.CloseOutputFile( "Test2_B.txt" );
    record.CloseOutputFile( "Test2_C.txt" );
    record.CloseOutputFile( "Test2_D.txt" );
    record.CloseOutputFile( "Test2_E.txt" );

    cout << endl << "END OF TEST 2" << endl;
}

void Test3()
{
    cout << endl << "------------------------------------" << endl;
    cout << "TEST 3: Write to a file that isn't opened" << endl << endl;
    RecordManager record;

    record.DisplayAllOpenFiles();

    record.WriteToFile( "Test2.txt", "How are you?" );

    cout << endl << "END OF TEST 3" << endl;
}

void Test4()
{
    cout << endl << "------------------------------------" << endl;
    cout << "TEST 4: Close a file that isn't opened" << endl << endl;
    RecordManager record;

    record.DisplayAllOpenFiles();

    record.CloseOutputFile( "Test3.txt" );

    cout << endl << "END OF TEST 4" << endl;
```

```cpp
}

void Test5()
{
    cout << endl << "----------------------------------------" << endl;
    cout << "TEST 5: Try to open more than max # of files" << endl << endl;
    RecordManager record;

    record.OpenOutputFile( "Test5_A.txt" );
    record.OpenOutputFile( "Test5_B.txt" );
    record.OpenOutputFile( "Test5_C.txt" );
    record.OpenOutputFile( "Test5_D.txt" );
    record.OpenOutputFile( "Test5_E.txt" );
    record.OpenOutputFile( "Test4_F.txt" ); // too many

    record.DisplayAllOpenFiles();

    cout << endl << "END OF TEST 5" << endl;
}

int main()
{
    Test1();
    Test2();
    Test3();
    Test4();
    Test5();


    return 0;
}
```

Adding error checking

Within the RecordManager class functions, you will add error checking and `throw` commands. Outside the class, in main and the tests, you will add the `try/catch` blocks.

Throw, try, and catch

INSIDE a function is where the `throw` command is used. After checking for some error state, the response is to `throw` the type of exception that has occurred, as well as an error message.

OUTSIDE a function (at the function-call level) is where `try` and `catch` commands are used. When a function that may throw an exception is being called, it should be wrapped within a `try` block. Then, the `catch` block(s) follow, to catch different types of exceptions, resolve them, clean up, and allow the program to continue.

Updating RecordManager::OpenOutputFile

In this function, add an error check before `m_outputs[ index ]` is accessed: Check to see if the *index* value is equal to -1. If so, throw a `runtime_error` like this:

```
if ( index == -1 )
{
    throw runtime_error( "No available files" );
}
```

Updating RecordManager::CloseOutputFile

This function should also throw a `runtime_error` if the returned index is -1.

Updating RecordManager::WriteToFile

This function should also throw a `runtime_error` if the returned index is -1.

Updating RecordManager::DisplayAllOpenFiles

This function won't cause any exceptions. Therefore, you should add the function specifier, `noexcept`, to the end of the signature - both in the declaration and the definition.

Updating RecordManager::FindAvailableFile

Add the `noexcept` function specifier.

Updating RecordManager::FindFilenameIndex

Add the `noexcept` function specifier.

Testing it out

Now when you run the program, it won't flat out crash like before, but it will abort the program once the first exception is hit.

```
-------------------------------------
TEST 1: Open one file and write to it

Open files:
0. Test1.txt

END OF TEST 1

-------------------------------------
TEST 2: Open 5 files and write to them

Open files:
0. Test2_A.txt
1. Test2_B.txt
2. Test2_C.txt
3. Test2_D.txt
4. Test2_E.txt

END OF TEST 2

-------------------------------------
TEST 3: Write to a file that isn't opened

Open files:
terminate called after throwing an instance of 'std::runtime_error'
  what():  File Test2.txt is not open
Aborted
```

(Note: This is the example output for the program running in Linux, with the g++ compiler. The result text might look different in Visual Studio.)

Next, `try/catch` blocks will need to be added in our tests so that the program will complete its execution, even if exceptions are discovered.

Adding try/catch

When we are calling a function that may cause an exception, that's when we should have a `try/catch` statement. The functions from `RecordManager` that may cause exceptions are:

- `OpenOutputFile`
- `CloseOutputFile`
- `WriteToFile`

Test1 and Test2 themselves won't cause any exceptions to be thrown, but if we were to add the try/catch to Test1, it would look like this:

```
// (...)
try
{
    record.DisplayAllOpenFiles();

    record.WriteToFile( "Test1.txt", "Hello world!" );

    record.CloseOutputFile( "Test1.txt" );
}
catch( runtime_error& ex )
{
    cout << "Error: " << ex.what() << endl;
}
// (...)
```

We could wrap each individual function in a try/catch, or wrap all three of them together. This is a design decision to make. In general, it is considered good design to wrap your try/catch around the smallest possible amount of code, but it also doesn't need to wrap just one line at a time. Since these three functions are related, we can wrap the three in a single try/catch block and it would be fairly clean.

Mainly, don't wrap an entire function in a try/catch - only a section working with "risky" areas.

Updating Test3, Test4, and Test5

For each of these tests, surround only the function calls that may return with an exception. Remember that any functions marked as `noexcept` will never throw an exception.

Once you've updated the entire program, the output should look like this:

```
-------------------------------------
TEST 1: Open one file and write to it

Open files:
0. Test1.txt

END OF TEST 1

-------------------------------------
TEST 2: Open 5 files and write to them

Open files:
0. Test2_A.txt
1. Test2_B.txt
2. Test2_C.txt
3. Test2_D.txt
4. Test2_E.txt

END OF TEST 2

-------------------------------------
TEST 3: Write to a file that isn't opened

Open files:
Error: File Test2.txt is not open

END OF TEST 3

-------------------------------------
TEST 4: Close a file that isn't opened

Open files:
Error: File Test3.txt is not open

END OF TEST 4

-------------------------------------
TEST 5: Try to open more than max # of files

Error: No available files
Open files:
0. Test5_A.txt
1. Test5_B.txt
2. Test5_C.txt
3. Test5_D.txt
4. Test5_E.txt

END OF TEST 5
```

Questions and Answers

Answer the following questions in a .txt file and turn them in with the rest of the lab. You will also want to reference the exception handling lecture for these questions.

1. What are the four levels of exception safety?
2. How many `catch` blocks can there be with a single `try` block?
3. Which exceptions are children of the `logic_error` exception?
4. Which exceptions are children of the `runtime_error` exception?