

CS 3721: Programming Languages Lab

Lab #10: Exceptions and I/O in C++

Due date: April 8, 3:30pm. At the beginning of the next recitation.

Goals of this lab:

- Be able to use eclipse to write C++ programs
 - Be able to read/write simple procedural C++ programs using exceptions, I/O streams, and namespaces.
1. *Namespace*. Namespaces group entities like classes, objects and functions under a name. In other words, it is used to structure a program into *logical units*. Its format is:

```
namespace identifier
{
  entities // set of classes, objects, and functions
}
```

Example:

```
namespace A {
    void f1() {std::cout << "A::f1()" << endl;}
    void f2() {std::cout << "A::f2()" << endl;}
}

namespace B {
    void f1() {std::cout << "B::f1()" << endl;}
    void f2() {std::cout << "B::f2()" << endl;}
}

// Accessing the members of namespace:
void g() {A::f1();}
// or:
void g()
{
    {
        using namespace A;
        f2(); // calling the function in namespace A.
    }
    {
        using namespace B;
        f2(); // calling the function in namespace B.
    }
}

// or:
void g()
{
    using A::f1;
    using B::f2;
    f1(); // calls the function in namespace A.
    f2(); // calls the function in namespace B.
}
```

```
}
```

Also, note that, you can separate the implementation and the interface of a namespace using header files. In this way, the implementation details are hidden from the outer world.

Here's the question: The following is the interfaces of two namespaces:

```
namespace mathPackage
{
    int add(int n); // Adds 5 to the argument n, and returns the result.
}

namespace setPackage
{
    int add(int n); // This is set operation. Adds the argument n to a set
                  // that should have 5 positive elements at most.
                  // If the set operation is successful, return the added
                  // number. Otherwise, return -1 (if the set is already
                  // full, or the argument n is nonpositive.).

    void displaySet(); // display the elements in the set.
}
```

- (a) Let's follow a top-down approach. The following is a main function which uses the namespaces that you will implement. The parts accessing the functions in the namespaces are missing. Please fill them according to the comments in the code.

```
#include <iostream>
#include "add.h"

using namespace std;

int main()
{
    for (int i = 0; i < 8; i++)
    {
        cout << _____ << endl; //call the function "add"
                                   //in the namespace mathPackage
                                   //with the argument i.
        if ( _____ != -1) //call the function "add" in the
                               //namespace setPackage with the argument
                               _____ //call the function "displaySet"
        else
            cout << "can not add" << i << endl;
    }
    return 0;
}
```

- (b) Copy the interfaces at the beginning of the question to the header file, "add.h". Then, **implement** them in file "add.cpp". For the *namespace setPackage*, you may simply use the

following array for your set:

```
int set[5] = {-1, -1, -1, -1, -1};
```

(Hint: Use the main function in part a to test your code. To do that, copy it to a file named `main.cpp`. Now, you have three files “`add.h`, `add.cpp`, and `main.cpp`”. Compile your files using the makefile on the webpage, and run the executable.)

2. *Exceptions*. Without exceptions, error conditions are generally handled by detecting them with some sort of *if* test, and then taking appropriate action when the condition is true. Often this action is to print an error message and exit the program. Alternatively, we can handle error conditions by transferring control to special functions called *handlers*. The C++ syntax involves *try* block, *throw* statement, and *catch* block to handle exceptions that have been thrown within the associated *try* block. Example:

```
...
try
{
    throw 10;
}
catch (int e)
{
    cout << "Exception No: " << e << endl;
}
...
```

A *throw* statement contains an expression and passes the value of the expression. A *catch* block may immediately follow a *try* block and receive any thrown exceptions.

```
try
{
    // statements that may throw exceptions
}
catch (int param) { cout << "int exception"; }
catch (char param) { cout << "char exception"; }
catch (...) { cout << "default exception"; }
```

Only the handler that matches its type with the argument specified in the throw statement is executed.

Here comes the question: Rewrite the main function in question 1 using exceptions. The new one should not contain any *if* statements. In order to achieve this, you should also make some changes in the implementation of the function *add* in the namespace `setPackage`. The outputs of the new program and the old one should match exactly.

3. I/O. C++ provides the following classes to perform input and output of characters from/to files:
 - Stream class to write on files
 - Stream class to read from files
 - Stream class to both read and write from/to files.

Following is an example:

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

```

using namespace std;

int main () {
    ofstream myfile;
    myfile.open ("example.txt");
    if (myfile.is_open())
    {
        myfile << "Writing this to a file.\n";
        myfile.close();
    }
    else
        cout << "error opening file" << endl;
    return 0;
}

```

This code creates a file, and writes a string into the file. Following is the version using exceptions.

```

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

int main () {
    ofstream myfile;
    // following sets the error state flag bits.
    myfile.exceptions( ofstream::eofbit | ofstream::failbit | ofstream::badbit );
    try {
        myfile.open ("example.txt");
        myfile << "Writing this to a file." << endl;
    }
    catch (ofstream::failure e) {
        cout << "Exception opening/writing file" << e.what() << endl;
    }
    myfile.close();
    return 0;
}

```

Now using **exceptions** and i/o streams, write a program which first creates a file “raw.dat” containing numbers from 1000 to 1050 with the following format:

```

1000
1001
.
.
.
1050

```

Then, read the values from the newly created file “raw.dat”, increment each value, and create another file “manipulated.dat” with the following format:

```

1001 1002 . . . 1051

```

(*Hint:* Use two file streams. One for input and one for output. Also, check for end of file using eof() while reading from file)