Lab 08
Exception, Templates and STL

Section 1: Guess program outputs.

1.

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

class IntRange
{
private:
   int input;     // For user input
   int lower;
   int upper;
public:
   // Exception class
   class OutOfRange
   {
   public:
       int value;
       OutOfRange(int i) { value = i; }
   };

   IntRange(int low, int high)
   {
      lower = low;
      upper = high;
   }
   int getInput()
   {
      cin >> input;
      if (input < lower || input > upper)
         throw OutOfRange(input);
      return input;
   }
};

int main()
{
   IntRange range(5, 10);
   int userValue;

   cout << "Enter a value in the range 5 - 10: ";
   try
   {
       userValue = range.getInput(); // software testing: please enter 12
       cout << "You entered " << userValue << endl;
   }
   catch (IntRange::OutOfRange ex)
   {
       cout << "That value " << ex.value
            << " is out of range.\n";
   }

   cout << "End of the program.\n";
   return 0;
}
```

2.

```
#include <iostream>
#include <cstdlib>
#include <memory>

using namespace std;

class IndexOutOfRangeException
{
public:
    const int index;
    IndexOutOfRangeException(int ix) : index(ix) {}
};

template <class T>
class SimpleVector
{
private:
    unique_ptr<T []> aptr;
    int arraySize;
public:
    SimpleVector(int s); // Constructor
    SimpleVector(const SimpleVector &obj); // Copy constructor

    int size() const { return arraySize; }
    T &operator[](int);                  // Overloaded [] operator
    void print() const;                  // outputs the array elements
};

template <class T>
SimpleVector<T>::SimpleVector(int s)
{
    arraySize = s;
    aptr = make_unique<T[]>(s);
    for (int count = 0; count < arraySize; count++)
        aptr[count] = T();
}

template <class T>
SimpleVector<T>::SimpleVector(const SimpleVector &obj)
{
    arraySize = obj.arraySize;
    aptr = make_unique<T[]>(obj.arraySize);
    for (int count = 0; count < arraySize; count++)
        aptr[count] = obj[count];
}

template <class T>
T &SimpleVector<T>::operator[](int sub)
{
    if (sub < 0 || sub >= arraySize)
        throw IndexOutOfRangeException(sub);
    return aptr[sub];
}

template <class T>
void SimpleVector<T>::print() const
{
    for (int k = 0; k < arraySize; k++)
        cout << aptr[k] << "  ";
```

```
    cout << endl;
}



template <class T>
class SearchableVector : public SimpleVector<T>
{
public:
    SearchableVector(int s) : SimpleVector<T>(s) { } // Constructor
    SearchableVector(const SearchableVector &obj); // Copy constructor

    // Overloaded constructor with base class object parameter
    SearchableVector(const SimpleVector<T> &obj):SimpleVector<T>(obj) { }
    int findItem(T);
};

template <class T>
SearchableVector<T>::SearchableVector(const SearchableVector &obj) :
    SimpleVector<T>(obj)
{
}

template <class T>
int SearchableVector<T>::findItem(T item)
{
    for (int count = 0; count < this->size(); count++)
    {
        if ((*this)[count] == item)
            return count;
    }
    return -1;
}


int main()
{
    const int SIZE = 5;
    SearchableVector<int> intTable(SIZE);
    SearchableVector<double> doubleTable(SIZE);

    for (int x = 0; x < SIZE; x++)
    {
        intTable[x] = (x * 2);
        doubleTable[x] = (x * 0.2);
    }

    cout << "These values are in intTable:\n";
    for (int x = 0; x < SIZE; x++)
        cout << intTable[x] << " ";
    cout << endl;
    cout << "These values are in doubleTable:\n";
    for (int x = 0; x < SIZE; x++)
        cout << doubleTable[x] << " ";
    cout << endl;

    int result;
    cout << "Searching for 6 in intTable.\n";
    result = intTable.findItem(6);
    if (result == -1)
        cout << "6 was not found in intTable.\n";
```

```
    else
        cout << "6 was found at subscript "
             << result << endl;

    cout << "Searching for 1.23 in doubleTable.\n";
    result = doubleTable.findItem(1.23);
    if (result == -1)
        cout << "1.23 was not found in doubleTable.\n";
    else
        cout << "1.23 was found at subscript "
            << result << endl;
    return 0;
}
```

Section 2: Review Questions and Exercises

Each of the following declarations or code segments has errors. Locate and explain the errors.

1.
```
catch
{
quotient = divide(num1, num2);
cout << "The quotient is " << quotient << endl;
}
try (string exceptionString)
{
cout << exceptionString;
}
```

2.
```
try
{
quotient = divide(num1, num2);
}
cout << "The quotient is " << quotient << endl;
catch (string exceptionString)
{
cout << exceptionString;
}
```

3.
```
template <class T>
T square(T number)
{
return T * T;
}
```

4.
```
template <class T>
int square(int number)
{
return number*number;
}
```

5.
```
template <class T1, class T2>
T1 sum(T1 x, T1 y)
{
return x + y;
}
```

Section 3: Programming Challenges

1. Rotate Left

The two sets of output below show the results of successive circular rotations of a vector. One set of data is for a vector of integers, and the second is for a vector of strings.

```
1 3 5 7
3 5 7 1
5 7 1 3
7 1 3 5

a b c d e
b c d e a
c d e a b
d e a b c
e a b c d
```

Write two template functions that can be used to rotate and output a vector of a generic type:

```
void rotateleft(vector<T>& v)
void output(vector<T> v)
```

The first function performs a single circular left rotation on a vector, and the second prints out the vector passed to it as parameter. Write a suitable driver program that will allow you to test the two functions by generating output similar to the above. Verify that the program works with vectors whose element types are char, int, double, and string.

2. Two-Dimensional Data

Suppose that data representing a list of people and places they would like to visit is stored in a file as follows:

```
3
0 Ali
1 AhKao
2 Muthu

0 3 Putrajaya Cyberjaya KL
1 1 Cyberjaya
2 0
```

The first number n in the file indicates how many people there are in the list. Here n is 3, so there are three people. Each person in the list is assigned a number in the range 0 .. n – 1 that is used to identify that person. For each person, the file lists the numerical identifier of the person, followed by the number of places the person wants to visit, followed by the names of those places. For example, Cyberjaya is the only place that AhKao cares to visit, while Muthu wants to visit no places.
Write a program that reads in this type of data from a file and stores it in appropriate STL data structure. For example, you might use vectors, as well as vectors of vectors, to represent this information. The program allows users to type in the name of a person whose list of favorite destinations is to be printed out. The program prints an error message if the person is not in the database.

2. Code Hint

```
#include <iostream>
#include <string>
#include <vector>
#include <fstream>

using namespace std;

void getData(ifstream& inFile, vector<string>& tourists,
             vector<vector<string> >& destinations)
{
// fill in this body
}


// Returns the index of a string in a vector of strings, or
// -1 if the searched for string is not found.
int indexOf(vector<string> vec, string name)
{
    for (int k = 0; k < vec.size(); k++ )
    {
        if (name == vec[k]) return k;
    }
    return -1;
}


int main(int argc, char* argv[])
{
    ifstream inFile("lab08pc02_tourists.txt");
    if (!inFile)
    {
        cout << "Cannot open the file lab08pc02_tourists.txt";
        return 1;
    }

    vector <string> tourists;
    vector<vector<string> > destinations;
    getData(inFile, tourists, destinations);

    // Print names of tourists
    cout << "Names of tourists are:\n";
    for (int k = 0; k < tourists.size(); k++)
    {
        cout << tourists[k] << endl;
    }

    // Interact with the user
    cout << "Enter name of tourists as prompted, type 'quit' to end.\n";
    string name;
    while (true)
    {
        cout << "Tourist? ";
        cin >> name;
        if (name == "quit") break;
        int index = indexOf(tourists, name);
        if (index == -1)
        {
            cout << "There is no such tourist." << endl;
            continue;
        }
```

```
        // print the list of destinations
        vector<string>::iterator iter = destinations[index].begin();
        while (iter != destinations[index].end())
        {
            cout << *iter << "  ";
            iter ++;
        }
        cout << endl;
    }
    cout << endl;

    return 0;
}
```