PS9

Deadline: You have to submit before 17:30

Q1.

Modify PFArray from Chapter 10 to throw an exception named OutOfRange if the index supplied is out of range, or if an insertion is attempted that causes the array to exceed its capacity.

```
//This is the HEADER FILE pfarrayd.h. This is the INTERFACE for the
//class PFArrayD. Objects of this type are partially filled arrays
//of doubles.
//The class PFArrayD has been modified by moving the member
//function definitions that are longer than one line to the
//implementation file, PFArrayD.cpp.
//The definition of a class OutOfRange has been added. The <string>
//header, needed for OutOfRange and its member functions, has been
//included.

#ifndef PFARRAYD_H
#define PFARRAYD_H

#include <string>

using std::string;


class Exception
{

};

class OutOfRange : public Exception
{
public:
  OutOfRange(string thisMessage, int thisIndex);
  string getMessage() const;
  int getIndex() const;
private:
  string message;
  int index;
};

//Objects of this class are partially filled arrays of doubles.
class PFArrayD
{
public:
    PFArrayD( );
    //Initializes with a capacity of 50.
    PFArrayD(int capacityValue);
    PFArrayD(const PFArrayD& pfaObject);
    void addElement(double element);
    //Precondition: The array is not full.
    //Postcondition: The element has been added.
    bool full( ) const { return (capacity == used); }
    //Returns true if the array is full, false otherwise.

    int getCapacity( ) const { return capacity; }
    int getNumberUsed( ) const { return used; }
    void emptyArray( ){ used = 0; }
    //Empties the array.
    double& operator[](int index) throw(OutOfRange);
    //Read and change access to elements 0 through numberUsed - 1.
    PFArrayD& operator =(const PFArrayD& rightSide);
    ~PFArrayD( );
private:
    double *a; //for an array of doubles.
    int capacity; //for the size of the array.
    int used; //for the number of array positions currently in use.
};
#endif //PFARRAYD_H
```

Use the main function below, make the necessary changes to the function testPFArrayD and write pfarrayd.cpp.

```
//Display 10.12, Demonstration Program for PFArrayD
//The main program has not been changed, but the
//testPFArrayD() function has been changed by adding
//try-catch blocks.

//Program to demonstrate the class PFArrayD.
#include <iostream>
#include "pfarrayd.h"

using std::cin;
using std::cout;
using std::endl;

void testPFArrayD( );
//Conducts one test of the class PFArrayD.

int main( )
{
    cout << "This program tests the class PFArrayD.\n";

    char ans;
    do
    {
        testPFArrayD( );
        cout << "Test again? (y/n) ";
        cin >> ans;
    } while ((ans == 'y') || (ans == 'Y'));

    return 0;
}
```

```
void testPFArrayD( )
{
    int cap;
    cout << "Enter capacity of this super array: ";
    cin >> cap;
    PFArrayD temp(cap);

    cout << "Enter up to " << cap << " nonnegative numbers.\n";
    cout << "Place a negative number at the end.\n";

    double next;
    cin >> next;

    while ((next >= 0) && (!temp.full( )))
    {
        temp.addElement(next);
        cin >> next;
    }

    cout << "You entered the following "
          << temp.getNumberUsed( ) << " numbers:\n";
    int index;
    int count = temp.getNumberUsed( );

    for (index = 0; index < count; index++)
    {
        cout << temp[index] << " ";
        cout << endl;
        cout << "(plus a sentinel value.)\n";
    }

    //Test index out of range
    cout << temp[-1];//2*count];

    //Test insert beyond declared size
    temp.addElement(99999);
}
```

Q2.

Write a program consisting of functions calling each other to a maximum depth of 10. Give each function an argument specifying a level at which it is to throw an exception. The main function prompts for and receives input that specifies the calling depth at which an exception is thrown. Don't forget the depth 0, where the main both throws and catches the exception. Hint: Use main and one other function that calls itself recursively.

Q3.

Modify the source code for the Stack class from Chapter 17. If the user of the class attempts to pop from an empty stack the program currently prints out an error message and exits. Modify the program so that it instead throws an exception named PopEmptyStackException.

Write a main method that tests the new Stack class, attempts to pop from an empty stack, and catches the PopEmptyStackException exception.

```
//stack.h
//
//A template-based class for a stack based on StackSavitch
//  from chapter 17.2 of Absolute C++.

#ifndef STACK_H
#define STACK_H

//*******************************
// Exception class for popping
// an empty stack
//*******************************

class PopEmptyStackException
{
};

template<class T>
class Node
{
public:
    Node(T theData, Node<T>* theLink) :
        data(theData), link(theLink) {}
    Node<T>* getLink() const { return link; }
    const T getData() const { return data; }
    void setData(const T& theData) { data = theData; }
    void setLink(Node<T>* pointer) { link = pointer; }
private:
    T data;
    Node<T> *link;
};
```

```
template<class T>
class Stack
{
public:
    Stack();
    Stack(const Stack<T>& aStack);
    Stack<T>& operator =(const Stack<T>& rightSide);
    virtual ~Stack();
    void push(T stackFrame);
    T pop() throw (PopEmptyStackException);
    bool isEmpty() const;
private:
    Node<T> *top;
};

#endif
```

Turn In

- Make and submit a zip file(<your_full_name> _PS9.zip) which includes the following:
  - Source codes of Q1, Q2 and Q3
  - Run Q1, Q2 and Q3 and attach screenshots(in jpg format, not exceeding 300kb each) which show that your programs run.
  - At least 1 screenshot for each question.