T-201-GSKI, GAGNASKIPAN

VECTOR OF INTEGERS

Assignment grading. For full marks your solution needs to be accepted by Mooshak. Points may be deducted for

- not implementing the solution according to the specifications,
- redundant or repeated code.

If your solution is not accepted by Mooshak, it will receive a maximum grade of 7.

IntVector

You are working as an archaeologist in Pompeii when you discover an entry to what you believe to be an ancient bathhouse. As you walk in you discover what appears to be an ancient C++ class on the wall.

It appears to be a vector class for integers, but parts of it are missing. You decide to reconstruct the missing code.

Vectors can be thought of as arrays that can grow in length while your program is running. In C++, once your program creates an array, it cannot change the length of the array. Vectors serve the same purpose except that hey can change length while your program is still running. You can read more about vectors in Chapter 4.1.1 in Shaffer and Chapter 8.3 in Savitch.

In this assignment you will implement the class `IntVector` which is a vector of integers.

The template

You are given the class declaration of the class `IntVector` in *IntVector.h*. A partial class implementation is given in *IntVector.cpp*. A small main-program is given to test the class in *main.cpp*. Note that the testing is not complete. You should write your own tests to ensure that your implementation is correct!

The class declaration is as follows.

```
class IntVector
{
    private:
        // A pointer to an array that stores the vector's elements.
        int* array;
        // Contains the size of array (i.e. the maximum number of elements
        // that the vector can store using the current array).
        int  capacity;
        // The number of array positions currently holding values.
        int  count;
```

Last updated January 19, 2015.

```

    public:
        // Initializes an empty vector.
        IntVector();
        // Initializes a vector with 'size' copies of the element 'value'.
        // If 'size' is less than 0, the initalized vector is empty.
        IntVector(int size, int value = 0);
        // A copy constructor.
        IntVector(const IntVector& vec);
        // A destructor.
        ~IntVector();

        // Returns the element at position 'index'.
        // If index is out of range, the function throws
        // IndexOutOfRangeException.
        int  at(int index) const;
        // Sets the value at position 'index' to 'elem'.
        // If index is out of range, the function throws
        // IndexOutOfRangeException.
        void set_value_at(int index, int elem);

        // Returns the size of the array.
        int  size() const;
        // Returns true if and only if the array contains no elements.
        bool empty() const;

        // Appends elem to the vector.
        void push_back(int elem);
        // Removes the last element of the vector and returns it.
        // If the vector is empty, the function throws
        // EmptyException.
        int  pop_back();
        // Inserts 'elem' into the list at position 'index'. All elements to the
        // right of index are shifted one position to the right.
        // If index is out of range, the function throws
        // IndexOutOfRangeException.
        void insert(int index, int elem);
        // Removes the element at position 'index'. All elements to the
        // right of index are shifted one position to the left.
        // If index is out of range, the function throws
        // IndexOutOfRangeException.
        void remove_at(int index);
        // Removes all the elements from the list
        void clear();

        // Overloaded = operator.
        void operator=(const IntVector& vec);
        // Returns a reference to the element at position 'index'.
        // If index is out of range, the function throws
        // IndexOutOfRangeException.
        int& operator[] (int index);
};
```

Implementation

The file *IntVector.cpp* contains an implementation of the overloaded assignment operator and the copy constructor for the `IntVector` class. Note that the `<<` operator has also been overloaded for the class.

The details of functionality of the class are specified in the comments of the class declaration. In addition to that you should note the following.

- In the default constructor of the class you should create a dynamic array of size `INITAL_CAPACITY`.
- If you attempt to add an element to the vector and there is no more room, then a dynamic array with twice the capacity of the old dynamic array is created. The values of the old dynamic array are then copied to the new dynamic array (and the new value subsequently added).
- If the `insert` function is called with an index that is equal to the size of the vector, the inserted element is appended to the vector.
- The functions `at`, `set_value_at`, `insert`, `remove_at` and `operator[]` all throw an instance of `IndexOutOfRangeException` if the specified index is out of range.
- The function `pop_back` throws an instance of `EmptyException` if the vector is empty.
- You are free (and encouraged) to use private helper functions in your implementation to make your code less redundant and more readable.

School of Computer Science, Reykjavík University, Menntavegi 1, 101 Reykjavík