CSE 330 LABORATORY -- Week 2, Fall 2018

Instructor: Kerstin Voigt

This lab will have you do the hard labor of reproducing a C++ implementation of the Vector ADT. The implementation closely follows the one in Chapter 2 of Weiss, Data Structures and Algorithm Analysis in C++. However, the instructor made a few modifications:

- We will not use C++ exceptions at this time; instead, we will use assert statements from the <cassert> STL.
- We will write '<typename T>' instead of '<typename Object>'.

Exercise1: Implement the Vector class in a file Vector.cpp exactly as shown on the pages at the end of this document.

Exercise 2: Test your Vector implementation with the Median and MaxSubSum programs from the lecture notes. The provide programs use the STL <vector> library. Replace line '#include <vector>' with '#include "Vector.h' and compile and run these programs with the home-made Vector class. It should work …

Credit for this Lab: At about 3pm, before the **signup sheet** is circulated; make sure that you sign up before leaving. Submit your lab's work via **portal** (M on W at 11:59pm, W on F at 11:59pm).

The further you get today, the better shape your will be in for your upcoming first graded homework assignment.

For the portal submissions, please minimally mark your code files with a comment on top:

```
// firstname lastname
```

While the portal provides information that will allow me to link your submission to your name, your explicit names on your work will make the recording of earned credit more robust.

***

Place the code on the following pages into a file named

Vector.h

(copy code until you reach the last page).

```
#ifndef VECTOR_H
#define VECTOR_H

//#include <algorithm>  // for swap???
#include <cstdlib>  // KV trying ...
#include <iostream>
#include <cassert>   // KV prefers assert ...

//#include <stdexcept>
//#include "dsexceptions.h"

template <typename T>
class Vector
{
public:
    explicit Vector(int initSize = 0)
        : theSize{ initSize },
          theCapacity{ initSize +
                       SPARE CAPACITY }
    { data = new T[theCapacity]; }

    Vector(const Vector& rhs)
        : theSize{ rhs.theSize },
          theCapacity{ rhs.theCapacity },
          data{ nullptr }
    {
        data = new T[theCapacity];
        for (int k = 0; k < theSize; ++k)
            data[k] = rhs.data[k];
    }
```

```
Vector& operator= (const Vector& rhs)
{
    Vector copy = rhs;
    std::swap(*this, copy);
    return *this;
}

~Vector()
{
    delete[] data;
}

Vector(Vector&& rhs)
    : theSize{ rhs.theSize },
      theCapacity{ rhs.theCapacity },
      data{ rhs.data }
{
    rhs.data = nullptr;
    rhs.theSize = 0;
    rhs.theCapacity = 0;
}

Vector& operator= (Vector&& rhs)
{
    std::swap(theSize, rhs.theSize);
    std::swap(theCapacity, rhs.theCapacity);
    std::swap(data, rhs.data);

    return *this;
}

bool empty() const
{
    return size() == 0;
}
```

```
    int size() const
    {
        return theSize;
    }

    int capacity() const
    {
        return theCapacity;
    }

    T& operator[](int index)
    {
        /*
#ifndef NO CHECK
        if (index < 0 || index >= size())
            throw
ArrayIndexOutOfBoundsException{ };
#endif
*/
        assert(index >= 0 && index < theSize);
        return data[index];
    }
```

```
void resize(int newSize)
{
    if (newSize > theCapacity)
        reserve(newSize * 2);
    theSize = newSize;
}

void reserve(int newCapacity)
{
    if (newCapacity < theSize)
        return;

    T* newArray = new T[newCapacity];
    for (int k = 0; k < theSize; ++k)
        newArray[k] = std::move(data[k]);

    theCapacity = newCapacity;
    std::swap(data, newArray);
    delete[] newArray;
}

void push back(const T& x)
{
    if (theSize == theCapacity)
        reserve(2 * theCapacity + 1);
    data[theSize++] = x;
}

void push_back(T&& x)
{
    if (theSize == theCapacity)
        reserve(2 * theCapacity + 1);
    data[theSize++] = std::move(x);
}
```

```
void pop_back()
{
    assert(theSize >= 1);
    /*
    if (empty())
        throw UnderflowException{ };
        */
    --theSize;
}

const T& back() const
{
    /*if (empty())
        throw UnderflowException{ };
        */
    assert(theSize >= 1);
    return data[theSize - 1];
}

// Iterators (new concept)

typedef T* iterator;
typedef const T* const_iterator;

iterator begin()
{
    return &data[0];
}
const_iterator begin() const
{
    return &data[0];
}
```

```
    iterator end()
    {
        return &data[size()];
    }

    const_iterator end() const
    {
        return &data[size()];
    }

    static const int SPARE_CAPACITY = 2;

private:
    int theSize;
    int theCapacity;
    T* data;
};

#endif
```