CSE 330 LABORATORY -- Week 2, Spring 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>’.
- We will not (yet) implement “iterators” for Vector.

```
// File: Vector.h
#ifndef VECTOR_H
#define VECTOR_H

#include <algorithm>
#include <iostream>
#include <cassert>     // KV: instead of exceptions …

template <typename T>
class Vector
{
  public:
    Vector( int initsize = 0 )
      : theSize( initsize ),
        theCapacity( initsize + SPARE_CAPACITY )
      { objects = new T[ theCapacity ]; }

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

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

`
```

```
~Vector( )
  { delete [ ] objects; }


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

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

    return *this;
}

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

int size( ) const
  { return theSize; }

int capacity( ) const
  { return theCapacity; }

T & operator[]( int index )
{
    assert(index >= 0 && index < theSize);
    return objects[ index ];
}

const T & operator[]( int index ) const
{
       assert(index >= 0 && index < theSize);
       return objects[ 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(objects[k]);

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


void push_back( const T & x )
{
    if( theSize == theCapacity )
        reserve( 2 * theCapacity + 1 );

    objects[ theSize++ ] = x;
}

void push_back( T && x )
{
    if( theSize == theCapacity )
        reserve( 2 * theCapacity + 1 );

    objects[ theSize++ ] = std::move( x );
}

void pop_back( )
{
       assert(!empty());
    --theSize;
}

const T & back ( ) const
{
     assert(!empty());
    return objects[ theSize - 1 ];
}

  const T & front() const
  {
       assert(!empty());
       return objects[0];
  }


static const int SPARE_CAPACITY = 2;
```

```
  private:
    int theSize;
    int theCapacity;

    T * objects;
};

#endif
```

Test your Vector implementation with the following int main() function:

```
// File: VectorMain.cpp
#include <iostream>
#include "Vector.h"
using namespace std;

void print_vector(Vector<int> v)
{
	for (int i = 1; i < v.size(); i++)
		cout << v[i] << " ";
	cout << endl;
	return;
}

int main()
{
	Vector<int> v1;
	for (int i = 1; i <= 10; i++)
		v1.push_back(i);

	cout << "v1: ";
	print_vector(v1);

	Vector<int> v2(v1);

	cout << "v2(v1): ";
	print_vector(v2);

	Vector<int>  v3 = v2;

	cout << "v3=v2: ";
	print_vector(v3);

	Vector<int> v4(static_cast<Vector<int>&&>(v3));

	cout << "v4(&&v3): ";
	print_vector(v4);
	cout << "v3: ";
	print_vector(v3);
```

```
    Vector<int>v5 = static_cast<Vector<int>&&>(v2);

    cout << "v5=&&v2: ";
    print_vector(v5);
    cout << "v2: ";
    print_vector(v2);

    return 0;
}
```

More Testing: If there is time left and/or you would like to see your Vector.h implementation tested in the context of a more interesting task, use it to implement the "Maximum Contiguous Subsequence Sum Algorithm" in Weiss in pp. 61 (instead of the STL <vector>, use your own "Vector.h")

Credit for this Lab: (1) At about 3pm, before the signup sheet is circulated, **send an email** to student assistant Sarthak (at 006118706@coyote.csusb.edu) with subject line "CSE 330 LAB 2". In the body of the email, type your first and last name the output you get with running the linux command 'wc' for your Vector.h file. There will be three numbers; include all three in your email. **(2) After sending your email**, make sure to sign the **signup sheet.**

Whether or not you complete this Vector implementation during the lab, know that its completion will be foundation for the upcoming homework assignment. Thus, the farther you get, the better prepared you will be for your homework.