CSE 330 LABORATORY – WEEK 2
UPDATED VERSION, 10-10-2011

Prof. Kerstin Voigt[1]

Implement class Vector that provides the vector functionality needed by the test code VectorMain.cpp given below. When you do this, you should not use C++ vector class. Since Vector is a template container class, it should be implemented in one file: Vector.h. Complete Vector.h given below. A few points regarding capacity versus size:

- `capacity`: amount of memory allocated for the container in terms of length of T (element type).
- `size`: number of elements in the container.
- `capacity` is always >= `size`
- `reserve()`: increases capacity by allocating more memory.
- `resize()`: could increase or decrease size. When increasing size, if necessary (when `size == capacity`) increase capacity as well.
- In your implementation when `size == capacity` and `push_back()` is called, allocate 5 more memory locations first (increase capacity by 5), then insert the new element (increase size by 1).

Now complete the following: (see next page)

[1]The instructor may not be the original author of this lab exercise. Certain sets of lab exercises have evolved as a standard that has been developed, modified and adapted by many CSE 330 instructors over the years.

```
// Vector.h -- scaled down for lab2;

#ifndef VECTOR_H
#define VECTOR_H

using namespace std;

template <class T>
class Vector
{
public:

   Vector();
   Vector(unsigned int size);
   ~Vector();

   unsigned int capacity() const;         // capacity of vector (in elements)
   unsigned int size() const;             // the number of elements in vector
   bool empty() const;

   T & front();                           // reference to the first element
   T & back();                            // reference to the last element
   void push_back(const T & value);       // add a new element
   void pop_back();                       // remove the last element

   T & operator[](unsigned int index);    // return ref to indexed element

private:
   unsigned int my_size;
   unsigned int my_capacity;
   T * buffer;
};

// Your code goes here ...
template <class T>
Vector<T>::Vector()
{ ... }

...
```

```
template <class T>
void Vector<T>::push_back(const T & value)
{
   if (mySize < myCapacity)
   {
      cout << "... plain push_back ..." << endl;
      buffer[mySize] = x;
      mySize++;
    }
   else
   {
      cout << "... push_back with resize ..." << endl;
      T* newbuf = new T [myCapacity + 5];
      copy(buffer, buffer+mySize, newbuf);

      delete [] buffer;
      buffer = newbuf;

      buffer[mySize] = x;
      mySize++;
      myCapacity += 5;
   }
}

#endif
```

Test with the following program in `VectorMain.cpp`:

```
// VectorMain.cpp

#include <iostream>
#include "Vector.h"

using namespace std;

int main()
{
   Vector<int> v1(5);
   int next;

   for (int i = 0; i < 5; i++)
   {
```

```
      cout << "Integer: ";
      cin >> next;
      v1.push_back(next);
      cout << endl;
   }

   cout << endl << "v1: ";
   for (int i = 0; i < v1.size(); i++)
   {
       cout << v1[i] << " ";
   }
   cout << endl;

   if (v1.size() >= v1.capacity())
      cout << "The vector is full" << endl;
   else
      cout << "The vector has capacity left" << endl;

   for (int i = 1; i <= 3; i++)
   {
      cout << endl << "Integer: ";
      cin >> next;
      v1.push_back(next);
   }

   cout << endl;
   cout << "Vector with additional elements:" << endl;
   cout << endl << "v1: ";
   for (int i = 0; i < v1.size(); i++)
   {
      cout << v1[i] << " ";
   }
   cout << endl;
   return 0;
}
```

Bonus exercise: In your infix-to-postfix implementation, replace the <`stack`> library with your `Vector.h`, and use a `Vector`<`char`> in place of `stack`<`char`>. Make all necessary modifications so that the new version of your infix-to-postfix converter has the same input/output behavior as the original one.

Obtain Credit for this lab by handing in: (1) A hardcopy of your version of `Vector.h`. (2) A hardcopy of a typescript that demonstrate the successful compiling and running of `VectorMain.cpp`

You must have submitted items (1) and (2) **by the next lab session,October 12, 2011**. Generous partial credit is to be had. Lack of submission results in 0 credit for the lab. All submissions must be individual work.

You must have also signed the signup-sheet for this week's lab.