**CSE 330 Lab 4 Report**

Daniel Meyer

Data Structures

Fall 2017

**Status:** 100%

**Time Complexity:** O(n)

**Storage Complexity:** O(n)

**Source Code:** Pages 2-5

**Sample Run:** Page 6

/\*

Daniel Meyer

10-16-17

CSE 330

Fall 2017

Lab 4: Vector

Problem: Implement a class "Vactor" with the functionality of a STL vector class

but implementing it as a template class

Algorithm: Created various constuctors needed for the template, functions needed

to change the capacity and size of the buffer. FUrthermore, functions were implemented

to aquire a pointer to the beginning and end of the buffer as well as the elements at

the beggingin and end. STL vector functions were also created such as push\_back, pop\_back,

resize, and reserve.

Status: 100%

Time Complexity: O(n) Storage Complexity: O(n)

\*/

#ifndef VECTOR\_H

#define VECTOR\_H

// Vector.h

using namespace std;

template <class T>

class Vector

{

private:

unsigned int my\_size;

unsigned int my\_capacity;

T \*buffer;

public:

typedef T \*iterator;

Vector();

Vector(unsigned int size);

Vector(unsigned int size, const T &initial);

Vector(const Vector<T> &v); // copy constructor

~Vector();

unsigned int capacity() const; // return capacity of vector (in elements)

unsigned int size() const; // return the number of elements in the vector

bool empty() const;

iterator begin(); // return an iterator pointing to the first element

iterator end(); // return an iterator pointing to one past the last element

T& front(); // return a reference to the first element

T& back(); // return a reference to the last element

void push\_back(const T &value); // add a new element

void pop\_back(); // remove the last element

void reserve(unsigned int capacity); // adjust capacity

void resize(unsigned int size); // adjust size

T & operator[](unsigned int index); // return reference to numbered element

void operator=(const Vector<T> &v);

};

//Default Constructor for the Vector template class

template <class T>

Vector<T>::Vector()

{

my\_size = 0;

my\_capacity = 0;

buffer = 0;

}

//Constructor assigning the size and capacity of the Vector template to given value

template <class T>

Vector<T>::Vector(unsigned int size)

{

my\_size = size;

my\_capacity = size;

buffer = new T[size];

for (int i = 0; i < size; i++) {

buffer[i] = T();

}

}

//Constructor assigning size, capacity, and initializing buffer to given values

template <class T>

Vector<T>::Vector(unsigned int size, const T &initial)

{

my\_size = size;

my\_capacity = size;

buffer = new T[size];

for (int i = 0; i < size; i++) {

buffer[i] = initial;

}

}

//Copy Constructor for new Vector template class

template <class T>

Vector<T>::Vector(const Vector<T> &v)

{

buffer = new T[v.my\_capacity];

for (int i = 0; i < v.my\_size; i++)

buffer[i] = v.buffer[i];

my\_size = v.my\_size;

my\_capacity = v.my\_capacity;

}

//Deconstructor for new Vector template class

template <class T>

Vector<T>::~Vector()

{

delete[] buffer;

}

//Returns the capacity of the Vector template

template <class T>

unsigned int Vector<T>::capacity() const

{

return my\_capacity;

}

//Returns the size of the Vector template

template <class T>

unsigned int Vector<T>::size() const

{

return my\_size;

}

//Returns a T/F if the Vector template is empty

template <class T>

bool Vector<T>::empty() const

{

return (my\_size == 0);

}

//Returns an iterator pointing to the beginning of the buffer

template <class T>

typename Vector<T>::iterator Vector<T>::begin()

{

return buffer;

}

//Returns an iterator pointing to the end of the buffer

template <class T>

typename Vector<T>::iterator Vector<T>::end()

{

return buffer + my\_size;

}

//Returns a reference to the first element in the buffer

template <class T>

T& Vector<T>::front()

{

return buffer[0];

}

//Returns a reference to the last element in the buffer

template <class T>

T& Vector<T>::back()

{

return buffer[my\_size - 1];

}

//Adds new value to the buffer increases size while also adjusting the capacity if needed

template <class T>

void Vector<T>::push\_back(const T &value)

{

if (my\_size == my\_capacity) {

reserve(my\_capacity + 5);

}

buffer[my\_size++] = value;

}

//Removes last value from buffer by decrementing size

template <class T>

void Vector<T>::pop\_back()

{

my\_size--;

}

//Sets the Vector template capacity to given value

template <class T>

void Vector<T>::reserve(unsigned int capacity)

{

if (capacity < my\_capacity)

return;

my\_capacity = capacity;

T \*temp = new T[capacity];

for (int i = 0; i < my\_size; i++)

temp[i] = buffer[i];

delete[] buffer;

buffer = temp;

}

//Resizes the size of the Vector template and adjusts capacity if needed

template <class T>

void Vector<T>::resize(unsigned int size)

{

my\_size = size;

if (my\_capacity < my\_size)

{

reserve(my\_size + 5);

}

}

//Returns element at specified index in the buffer

template <class T>

T& Vector<T>::operator[](unsigned int index)

{

return buffer[index];

}

//Sets buffer to new buffer along with its contents as well as mupdates the size and capacity

template <class T>

void Vector<T>::operator=(const Vector<T> &v)

{

delete[] buffer;

buffer = new T[v.my\_capacity];

for (int i = 0; i < v.my\_size; i++)

buffer[i] = v.buffer[i];

my\_size = v.my\_size;

my\_capacity = v.my\_capacity;

}

#endif

**Sample Run**

Script started on Sat 21 Oct 2017 11:15:09 PM UTC

To run a command as administrator (user "root"), use "sudo <command>".

See "man sudo\_root" for details.

]0;ubuntu@ubuntu: ~/Desktop[01;32mubuntu@ubuntu[00m:[01;34m~/Desktop[00m$ g++ Vector.h

]0;ubuntu@ubuntu: ~/Desktop[01;32mubuntu@ubuntu[00m:[01;34m~/Desktop[00m$ g++ Vecot[K[Ktor.[K\_Test.cpp

]0;ubuntu@ubuntu: ~/Desktop[01;32mubuntu@ubuntu[00m:[01;34m~/Desktop[00m$ ./a.out

SUCCESS

]0;ubuntu@ubuntu: ~/Desktop[01;32mubuntu@ubuntu[00m:[01;34m~/Desktop[00m$ exit

Script done on Sat 21 Oct 2017 11:15:34 PM UTC