Eric Blasko

Data Structures CSE330

Winter 2018

Lab 4 – Vector

* **Status**

100% complete

* **Complexity Analaysis**

The following functions are O(1) as there functions are constant

* Vector(Vector<T> && v);
* ~Vector();
* Vector<T> & operator=(Vector<T> && v);
* unsigned int capacity() const;
* unsigned int size() const;
* bool empty() const;
* iterator begin();
* iterator end();
* T & front();
* T & back();
* void pop\_back();
* T & operator[](unsigned int index);

The following functions are O(n), were n is the size of the vector

* Vector(unsigned int size = 0, const T & initial = T());
* Vector(const Vector<T> & v);
* Vector<T> & operator=(const Vector<T> & v);
* void erase(iterator);

The following functions are ether best case O(1) or worst case O(n), were n is the size of the vector

* void resize(unsigned int size);
  + O(1) if vector is being changed to smaller size
  + O(n) if vector is being changed to larger size
* void reserve(unsigned int capacity);
  + O(1) if vector does not need capacity changed
  + O(n) if vector capacity needs to be changed
* void push\_back(const T & value);
  + O(1) if reserve function does not need to be called
  + O(n) if reserve function needs to be called
* **Source Code**

#ifndef VECTOR\_H

#define VECTOR\_H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Eric Blasko

\* Vector.h

\* 02/07/2018

\* This class replicates the functions of the vector class. It allows user to change

\* size of an array of elements as well as display, add, or remove elements from any

\* location of array. Templated class, so interface and implamincation are in same file

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<cassert>

using namespace std;

template <class T>

class Vector

{

public:

typedef T \* iterator;

Vector(unsigned int size = 0, const T & initial = T()); //default constructor

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

Vector(Vector<T> && v); //move constructor

~Vector(); //destructor

Vector<T> & operator=(const Vector<T> & v); //copy assignment

Vector<T> & operator=(Vector<T> && v); //move assignment

unsigned int capacity() const; //increase capacity

unsigned int size() const; //change size

bool empty() const; //check if empty

iterator begin(); //return iterator to first element

iterator end(); //return iterator to past last element

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

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

void push\_back(const T & value); //add element to end

void pop\_back(); //remove element from end

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

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

T & operator[](unsigned int index);

void erase(iterator); //erase elem iterator points to(then shift)

private:

unsigned int my\_size;

unsigned int my\_capacity;

T \* buffer;

};

//default constructor

template <class T>

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

{

my\_size = size;

my\_capacity = size;

if(size == 0)

buffer = nullptr;

else

{

buffer = new T[size];

assert(buffer != nullptr);

}

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

buffer[i] = initial;

}

//copy constructor

template <class T>

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

{

my\_size = v.my\_size;

my\_capacity = v.my\_capacity;

buffer = new T[my\_capacity];

assert(buffer != nullptr);

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

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

}

//move constructor

template <class T>

Vector<T>::Vector(Vector<T> && v) : my\_size(v.my\_size), my\_capacity(v.my\_capacity), buffer(v.buffer)

{

v.my\_size = 0;

v.my\_capacity = 0;

v.buffer = nullptr;

}

//destructor

template <class T>

Vector<T>::~Vector()

{

my\_size = 0;

my\_capacity = 0;

delete [] buffer;

}

//copy assignemnt

template <class T>

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

{

T \* temp = new T[v.my\_capacity];

assert(temp != nullptr);

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

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

delete [] buffer;

buffer = temp;

my\_size = v.my\_size;

my\_capacity = v.my\_capacity;

return \*this;

}

//move assingment

template <class T>

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

{

swap(my\_size, v.my\_size);

swap(my\_capacity, v.my\_capacity);

swap(buffer, v.buffer);

}

//increase to new capacity

template <class T>

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

{

if(capacity <= my\_capacity)

return;

T \* new\_buffer = new T[capacity];

assert(new\_buffer != nullptr);

my\_capacity = capacity;

int size;

for(size = 0; size < my\_size; size++)

new\_buffer[size] = buffer[size];

delete [] buffer;

buffer = new\_buffer;

}

//change to new size

template <class T>

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

{

if(size > my\_capacity)

reserve(size \* 2);

if(size > my\_size)

for(; my\_size < size; my\_size++)

buffer[my\_size] = T();

else my\_size = size;

}

//return capacity

template <class T>

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

{

return my\_capacity;

}

//return size

template <class T>

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

{

return my\_size;

}

//check if vector is empty

template <class T>

bool Vector<T>::empty() const

{

if(my\_size == 0)

return true;

return false;

}

//set iterator to start of vector

template <class T>

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

{

return iterator(buffer);

}

//set iterator to end of vector

template <class T>

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

{

return iterator(&buffer[my\_size]);

}

//return value at front of vector

template <class T>

T& Vector<T>::front()

{

return buffer[0];

}

//return value at end of vector

template <class T>

T& Vector<T>::back()

{

return buffer[size];

}

//add element to back. Increase capacity if needed and increase size by one

template <class T>

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

{

if(my\_size == 0)

reserve(10);

if(my\_size == my\_capacity)

reserve(my\_capacity \* 2);

my\_size += 1;

buffer[my\_size - 1] = value;

}

//remove last element and decrease size by one

template <class T>

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

{

buffer[my\_size - 1].~T();

my\_size -= 1;

}

//return valuse at index

template <class T>

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

{

return buffer[index];

}

//erase value at iterator position and shift remaining values left

template <class T>

void Vector<T>::erase(iterator it)

{

T \* temp = new T[my\_size - 1];

for(int i = 0; i < it; i ++)

temp[i] = buffer[i];

for(int i =it + 1; i < my\_size; i++)

temp[i] = buffer[i];

my\_size -= 1;

delete [] buffer;

buffer = temp;

}

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Eric Blasko

\* Vector\_test1.cpp

\* 02/07/2018

\* This program test the functionality of the templated Vector.h class as well as the String.h

\* class. Various instantances are created to test different constructor and copy and move

\* assignments

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <cassert>

#include "Vector.h"

#include "../lab3/String.h"

using namespace std;

//This fucntion prints the name of the instance and its contents

template<class T>

void print(const String & name, Vector<T> & v)

{

cout << name << ": ";

for(int i = 0; i < v.size(); i++)

cout << v[i] << " ";

cout << endl;

}

//Main function that test various constructors and member functions

int main()

{

cout << "\ndefault constructor\n";

Vector<int> v0(2);

v0[0] = 10;

v0[1] = 20;

print ("v0",v0);

cout << "\ncopy constructor v1(v0)\n";

Vector<int> v1(v0);

print("v0", v0);

print("vi", v1);

cout << "\nmove constructor v2(v1)\n";

Vector<int> v2(move(v1));

print("v1",v1);

print("v2",v2);

cout << "\ncopy assignment v1 = v2\n";

v1 = v2;

print("v1",v1);

print("v2",v2);

Vector<int> v3(3);

v3[0] = 100;

v3[1] = 200;

v3[2] = 300;

cout << endl;

print("v3",v3);

cout << "\n move assignment v2 = v3\n";

v2 = move(v3);

print("v2",v2);

print("v3",v3);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Eric Blasko

\* Vector\_test2.cpp

\* 02/07/2018

\* This program test the functionality of the templated class Vector.h. If any member

\* functions fail to perform as desired, assert will stop the program and display an

\* error

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <string>

#include <cassert>

#include "Vector.h"

using namespace std;

//Main function that test various instances of Vector.h and its member functions

//if member function fails to perform as desired, assert ends program with error

int main()

{

Vector<int> v;

v.reserve(2);

assert(v.capacity() == 2);

Vector<string> v1(2);

assert(v1.capacity() == 2);

assert(v1.size() == 2);

assert(v1[0] == "");

assert(v1[1] == "");

v1[0] = "hi";

assert(v1[0] == "hi");

Vector<int> v2(2,7);

assert(v2[1] == 7);

Vector<int> v10(v2);

assert(v10[1] == 7);

Vector<string> v3(2,"hello");

assert(v3.size() == 2);

assert(v3.capacity() == 2);

assert(v3[0] == "hello");

assert(v3[1] == "hello");

v3.resize(1);

assert(v3.size() == 1);

assert(v3[0] == "hello");

Vector<string> v4 = v3;

assert(v4.size() == 1);

assert(v4[0] == v3[0]);

v3[0] = "test";

assert(v4[0] != v3[0]);

assert(v4[0] == "hello");

v3.pop\_back();

assert(v3.size() == 0);

Vector<int> v5(7,9);

Vector<int>::iterator it = v5.begin();

while(it != v5.end())

{

assert(\*it == 9);

++it;

}

Vector<int> v6;

v6.push\_back(100);

assert(v6.size() == 1);

assert(v6[0] == 100);

v6.push\_back(101);

assert(v6.size() == 2);

assert(v6[0] == 100);

v6.push\_back(101);

cout << "SUCCESS\n";

}

* **Sample Runs**

**Test 1**

Script started on 2018-02-05 15:44:53-0800

^[]777;notify;Command completed;g++ -c Vector.h^G^[]0;005670557@csusb.edu@jb358-15:~/cse330/lab4^G^[]7;file://jb358-15.cse.csusb.edu/home/csusb.edu/005670557/cse330/lab4^G[005670557@csusb.edu@jb358-15 lab4]$ g++ Ve^H^[[K^H^[[K- ^H^[[Kc Vector\_test1.cpp^M

^[]777;notify;Command completed;g++ -c Vector\_test1.cpp^G^[]0;005670557@csusb.edu@jb358-15:~/cse330/lab4^G^[]7;file://jb358-15.cse.csusb.edu/home/csusb.edu/005670557/cse330/lab4^G[005670557@csusb.edu@jb358-15 lab4]$ g++ Vector\_test1.o ../lab3/String.o^M

^[]777;notify;Command completed;g++ Vector\_test1.o ../lab3/String.o^G^[]0;005670557@csusb.edu@jb358-15:~/cse330/lab4^G^[]7;file://jb358-15.cse.csusb.edu/home/csusb.edu/005670557/cse330/lab4^G[005670557@csusb.edu@jb358-15 lab4]$ ./a.out^M

^M

default constructor^M

v0: 10 20 ^M

^M

copy constructor v1(v0)^M

v0: 10 20 ^M

vi: 10 20 ^M

^M

move constructor v2(v1)^M

v1: ^M

v2: 10 20 ^M

^M

copy assignment v1 = v2^M

v1: 10 20 ^M

v2: 10 20 ^M

^M

v3: 100 200 300 ^M

^M

move assignment v2 = v3^M

v2: 100 200 300 ^M

v3: 10 20 ^M

^[]777;notify;Command completed;./a.out^G^[]0;005670557@csusb.edu@jb358-15:~/cse330/lab4^G^[]7;file://jb358-15.cse.csusb.edu/home/csusb.edu/005670557/cse330/lab4^G[005670557@csusb.edu@jb358-15 lab4]$ exit^M

Script done on 2018-02-05 15:45:31-0800

**Test 2**

Script started on 2018-02-05 15:45:41-0800

^[]777;notify;Command completed;./a.out^G^[]0;005670557@csusb.edu@jb358-15:~/cse330/lab4^G^[]7;file://jb358-15.cse.csusb.edu/home/csusb.edu/005670557/cse330/lab4^G[005670557@csusb.edu@jb358-15 lab4]$ g++ -c Vector\_s^H^[[Ktest2.cpp^M

^[]777;notify;Command completed;g++ -c Vector\_test2.cpp^G^[]0;005670557@csusb.edu@jb358-15:~/cse330/lab4^G^[]7;file://jb358-15.cse.csusb.edu/home/csusb.edu/005670557/cse330/lab4^G[005670557@csusb.edu@jb358-15 lab4]$ g++ Vector\_test2.o^M

^[]777;notify;Command completed;g++ Vector\_test2.o^G^[]0;005670557@csusb.edu@jb358-15:~/cse330/lab4^G^[]7;file://jb358-15.cse.csusb.edu/home/csusb.edu/005670557/cse330/lab4^G[005670557@csusb.edu@jb358-15 lab4]$ ./a.out^M

SUCCESS^M

^[]777;notify;Command completed;./a.out^G^[]0;005670557@csusb.edu@jb358-15:~/cse330/lab4^G^[]7;file://jb358-15.cse.csusb.edu/home/csusb.edu/005670557/cse330/lab4^G[005670557@csusb.edu@jb358-15 lab4]$ exit^M

Script done on 2018-02-05 15:46:13-0800