makefile

```
############################################################
# Program:
#     Lesson 00, Vector
#     Brother Helfrich, CS235
# Author:
#     <your name here>
# Summary:
#     <put a description here>
############################################################

############################################################
# The main rule
############################################################
a.out: vector.h lesson00.o
       g++ -o a.out lesson00.o
       tar -cf lesson00.tar vector.h lesson00.cpp makefile

############################################################
# The individual components
#      lesson00.o     : the driver program
############################################################
lesson00.o: vector.h lesson00.cpp
       g++ -c lesson00.cpp
```

Commented [HJ1]: NO HEADER

You need to take a minute and fill this out in every file. It is part of the assignment requirements.

vector.h

```
/******************
 * Vector.h
 * This file contains the classes for a Vector
 * and its iterator and the different functions
 * needed for its implementation.
 ******************/

#ifndef VECTOR_H
#define VECTOR_H

#include <cassert>
#include <iostream>
using namespace std;

//because we use this in the Vector class
template<class T>
class VectorIterator;

/********
 * the implementation of the Vector Class
 ********/
template <class T>
class Vector
{
  public:
   // default constructor
  Vector() : myCapacity(0), numItems(0), data(0x00000000) {}

   // non-default contsructor
   Vector(int capacity) throw (const char *);

   // copy constructor
   Vector(const Vector & rhs) throw (const char *) { *this = rhs; }

   // destructor
   ~Vector() { if (myCapacity) delete [] data; }

   // is the container empty
   bool empty() const { return numItems == 0; }

   //return how many items are in the Vector
   int size() const { return numItems; }
```

Commented [HJ2]: Not needed.

Commented [HJ3]: NAMESPACE

Do not include
    using namespace std;
in a header file. By so doing, you are forcing everyone using your library to also use the standard namespace. This may not be what they want to do.

```
    //how big is the Vector
    int capacity() const { return myCapacity; }

    //clears the Vector
    void clear() { numItems = 0; }

    // the push back function, allowing the user to add values to the Vector.
    // Also doubles the size and reallocates when the Vector gets full.
    void push_back(T newValue);

    // the square bracket operator overload
    const T operator [](int item) const;

    //overloaded assignment operator
    Vector<T> & operator = (const Vector<T>& rhs);

    // sets an iterator to the first item of data
    VectorIterator<T> begin() { return VectorIterator<T>(data); }

    // sets an iterator to the last item of data
    VectorIterator<T> end() {return VectorIterator<T>(data +numItems); }

  private:
    int myCapacity; //how big the Vector is
    int numItems;   //the number of Items in the Vector
    T * data;       // the storage unit of the Vector

};

//implementation of the VectorIterator class
template <class T>
class VectorIterator
{
  public:
    //default constructor
  VectorIterator() : p(0x00000000) {}

    //non-default constructor
  VectorIterator(T * p) : p(p) {}

    //copy contructor
    VectorIterator(const VectorIterator & rhs) { *this = rhs; }

    //assignent operator overloaded
    VectorIterator & operator = (const VectorIterator & rhs)
    {
       this->p = rhs.p;
       return *this;
    }

    //not equal operator
    bool operator != (const VectorIterator & rhs) const
    {
       return rhs.p != this->p;
    }
    //de reference operator
    T & operator * ()
    {
       return *p;
    }

    //increment.
    VectorIterator <T> & operator ++ ()
    {
       p++;
       return *this;
    }

    VectorIterator <T> operator++ (int postfix)
    {
       VectorIterator tmp(*this);
       p++;
       return tmp;
    }

  private:
    T * p;
```

Commented [HJ4]: Needs to throw

Commented [HJ5]: Not quite. The [] operator needs to return by-reference and not be a const. The () operator is defined like this. Both need to throw in case of an invalid index

Commented [HJ6]: Should throw

Commented [HJ7]: You need to precede class definitions with:
/*******************************
 *
 *******************************/

```
};


//implementation of the non default constructor
template <class T>
Vector <T> :: Vector (int capacity) throw (const char *)
{
   assert(capacity >= 0);

   if (capacity == 0)
   {
      this->myCapacity = this->numItems = 0;
      this->data = 0x00000000;
      return;
   }

   try
   {
      data = new T[capacity];
   }
   catch (std::bad_alloc)
   {
      throw "ERROR: Unable to allocate a new buffer for Vector";
   }


   this->myCapacity = capacity;
   this->numItems = 0;
}

//implementation of the push back function
template<class T>
void Vector <T> :: push_back(T newValue)
{
   T * newData;
   if (myCapacity == 0)
   {
      myCapacity += 1;
      data = new T[myCapacity];
   }

   if (myCapacity == numItems)
   {
      myCapacity *= 2;

      try
      {
         newData = new T[myCapacity];
      }

      catch(...)
      {
         cout << "Unable to allocate a buffer for Vector";
         myCapacity /= 2;
      }

      int i;
      for (i = 0; i < numItems; i++)
      {
         newData[i] = data[i];
      }
      newData[i] = '\0';

      delete [] data;
      data = newData;

   }

   data[numItems] = newValue;
   numItems++;
}

//implementation of the square bracket operator
template <class T>
const T Vector<T> :: operator [] (int item) const
{
   return this->data[item];
}
```

Commented [HJ8]: Full comment block please.

Commented [HJ9]: You should throw here, not display an error message.

Commented [HJ10]: No NULL character with a vector

Commented [HJ11]: Bounds chcking please.

```
//implementation of the assignment overator
template<class T>
Vector<T> & Vector<T> :: operator = (const Vector<T>& rhs)
{
   assert(rhs.myCapacity >= 0);

   if (rhs.myCapacity == 0)
   {
      this->myCapacity = this->numItems = 0;
      this->data = 0x00000000;
      return *this;
   }

   try
   {
      this->data = new T[rhs.myCapacity];
   }
   catch(std:: bad_alloc)
   {
      throw "ERROR: Unable to allocate buffer";
   }

   //  assert(rhs.numItems >= 0 && rhs.numItems <= rhs.myCapacity);
   this->myCapacity = rhs.myCapacity;
   this->numItems = rhs.numItems;
   for (int i = 0; i < numItems; i++)
   {
      this->data[i] = rhs.data[i];
   }
}

#endif
```

Commented [HJ12]: There should be only one copy of this copy code.

lesson00.cpp

```
/***********************************************************************
* Program:
*    Lesson 00, Vector
*    Brother Helfrich, CS 235
* Author:
*    Br. Helfrich
* Summary:
*    This is a driver program to exercise the Vector class.  When you
*    submit your program, this should not be changed in any way.  That being
*    said, you may need to modify this once or twice to get it to work.
************************************************************************/

#include <iostream>       // for CIN and COUT
#include <string>         // because testIterate() uses a Vector of string
#include "vector.h"       // your Vector class needs to be in vector.h
using namespace std;

// prototypes for our four test functions
void testSimple();
void testFill();
void testIterate();
void testCopy();
void testExtra();

// To get your program to compile, you might need to comment out a few
// of these. The idea is to help you avoid too many compile errors at once.
// I suggest first commenting out all of these tests, then try to use only
// TEST1.  Then, when TEST1 works, try TEST2 and so on.
#define TEST1   // for testSimple()
#define TEST2   // for testFill()
#define TEST3   // for testIterate()
#define TEST4   // for testCopy()
//#define TEST5   // for testExtra()

/***********************************************************************
 * MAIN
 * This is just a simple menu to launch a collection of tests
 ***********************************************************************/
int main()
{
   // menu
   cout << "Select the test you want to run:\n";
```

Commented [HJ13]: Good.

```
   cout << "\t1. Just create and destroy a Vector.\n";
   cout << "\t2. The above plus fill the Vector.\n";
   cout << "\t3. The above plus iterate through the Vector.\n";
   cout << "\t4. The above plus copy the Vector.\n";
   cout << "\t5. The extra credit test: constant and reverse iterators.\n";

   // select
   int choice;
   cout << "> ";
   cin  >> choice;
   switch (choice)
   {
      case 1:
         testSimple();
         cout << "Test 1 complete\n";
         break;
      case 2:
         testFill();
         cout << "Test 2 complete\n";
         break;
      case 3:
         testIterate();
         cout << "Test 3 complete\n";
         break;
      case 4:
         testCopy();
         cout << "Test 4 complete\n";
         break;
      case 5:
         testExtra();
         cout << "Test 5 complete\n";
         break;
      default:
         cout << "Unrecognized command, exiting...\n";
   }

   return 0;
}

/*****************************************
 * TEST SIMPLE
 * Very simple test for a vector: create and destroy
 *****************************************/
void testSimple()
{
#ifdef TEST1
   // Test1: bool Vector with default constructor
   cout << "Create a bool Vector using default constructor\n";
   Vector <bool> v1;
   cout << "\tSize:     " << v1.size()                  << endl;
   cout << "\tCapacity: " << v1.capacity()              << endl;
   cout << "\tEmpty?    " << (v1.empty() ? "Yes" : "No") << endl;

   {
      // Test2: double Vector with non-default constructor
      cout << "Create a double Vector using the non-default constructor\n";
      Vector <double> v2(10 /*capacity*/);
      cout << "\tSize:     " << v2.size()                  << endl;
      cout << "\tCapacity: " << v2.capacity()              << endl;
      cout << "\tEmpty?    " << (v2.empty() ? "Yes" : "No") << endl;
   }
#endif // TEST1
}

/*****************************************
 * TEST FILL
 * This will test the following:
 *   1. Instantiating a Vector object
 *   2. Filling the contents with values
 *   3. Destroying an object when finished
 *****************************************/
void testFill()
{
#ifdef TEST2
   // Test1: integer Vector with default constructor
   {
      // create
      cout << "Create an integer vector with the default constructor\n";
      Vector <int> v1(3);
```

```
      cout << "\tEnter numbers, type 0 when done\n";
      int number;
      do
      {
         cout << "\t> ";
         cin  >> number;
         if (number)
            v1.push_back(number);
      }
      while (number);
      cout << "\tSize:     " << v1.size()                  << endl;
      cout << "\tCapacity: " << v1.capacity()              << endl;
      cout << "\tEmpty?    " << (v1.empty() ? "Yes" : "No") << endl;
   }
   cout << "\tFirst vector deleted\n";

   // Test2: character Vector with non-default constructor
   {
      cout << "Create a character Vector with non-default constructor\n";
      Vector <char> v2(2);

      cout << "Insert user-provided characters in the Vector\n";
      cout << "\tEnter characters, type 'q' when done\n";
      char letter;
      do
      {
         cout << "\t> ";
         cin  >> letter;
         if (letter != 'q')
            v2.push_back(letter);
      }
      while (letter != 'q');
      cout << "\tSize:  " << v2.size() << endl;

      // clear the contents
      cout << "\tNow we will clear the contents\n";
      v2.clear();
      cout << "\tSize:     " << v2.size()                  << endl;
      cout << "\tCapacity: " << v2.capacity()              << endl;
      cout << "\tEmpty?    " << (v2.empty() ? "Yes" : "No") << endl;
   }
   cout << "\tSecond Vector deleted\n";
#endif // TEST2
}

/****************************************
 * TEST FILL
 * This will test the following:
 *   1. Instantiating a Vector object
 *   2. Filling the contents with values
 *   3. Displaying the values using an iterator
 *   4. Destroying an object when finished
 ****************************************/
void testIterate()
{
#ifdef TEST3
   // create a list
   cout << "Create a Vector of strings with the default constructor.\n";
   Vector <string> v;

   // fill the container with text
   cout << "\tEnter text, type \"quit\" when done\n";
   string text;
   do
   {
      cout << "\t> ";
      cin  >> text;
      if (text != "quit")
         v.push_back(text);
   }
   while (text != "quit");

   // display the contents of the Container
   cout << "Use the iterator to display the contents of the vector\n";
   VectorIterator <string> it;
   for (it = v.begin(); it != v.end(); ++it)
      cout << "\t" << *it << endl;
```

```
   // find a given item
   int index;
   cout << "Which item would you like to look up?\n";
   cout << "> ";
   cin  >> index;
   try
   {
      cout << "\t" << v[index] << endl;
   }
   catch (const char * s)
   {
      cout << "\t" << s << endl;
   }
#endif // TEST3
}

/*********************************************
 * TEST COPY
 * This will test the following:
 *    1. Instantiate a Vector object using non-default constructor
 *    2. Fill the contents with values
 *    3. Copy one Vector with the values of another
 *    4. Display the contents of the copied Vector with an iterator
 *********************************************/
void testCopy()
{
#ifdef TEST4
   // create a list
   cout << "Create a Vector of floats with the default constructor.\n";
   Vector <float> v1;

   // fill the vector with numbers
   cout << "\tEnter numbers, type 0.0 when done\n";
   float number;
   do
   {
      cout << "\t> ";
      cin  >> number;
      if (number != 0.0)
         v1.push_back(number);
   }
   while (number != 0.0);

   // copy the container
   cout << "Copy the contents of the Vector into a new Vector\n";
   Vector <float> v2(v1);

   // display the contents of the Vector
   cout << "Use the iterator to display the contents of the Vector\n";
   VectorIterator <float> it;
   cout.setf(ios::fixed | ios::showpoint);
   cout.precision(1);
   for (it = v2.begin(); it != v2.end(); ++it)
      cout << "\t" << *it << endl;
#endif // TEST4
}

/*********************************************************
 * TEST EXTRA
 * This will test the following for extra credit:
 *    1. Instantiate a Vector object using non-default constructor
 *    2. Fill the contents with values
 *    3. Iterate through the Vector backwards
 *    4. Iterate through the Vector with a constant iterator
 *    5. Iterate through the Vector backwards with a constant iterator
 *********************************************************/
void testExtra()
{
#ifdef TEST5
   // create a list
   cout << "Create a Vector of int with the non-default constructor.\n";
   Vector <int> v1(4);

   // fill the vector with numbers
   cout << "\tEnter four integers\n";
   for (int i = 0; i < 4; i++)
   {
      int number;
      cout << "\t> ";
```

```
      cin  >> number;
      v1.push_back(number);
   }

   // backwards non-constant iterator
   cout << "Move through the vector backards using a non-constant iterator\n";
   for (VectorIterator <int> it = v1.rbegin(); it != v1.rend(); --it)
      cout << "\t" << *it << endl;

   // copy the vector to a constant vector
   const Vector <int> v2 = v1;

   // forwards constant iterator
   cout << "Move through the vector forwards with a constant iterator\n";
   for (VectorConstIterator <int> it = v2.cbegin(); it != v2.cend(); ++it)
      cout << "\t" << *it << endl;

   // backwards constant iterator
   cout << "Move through the vector backwards with a constant iterator\n";
   for (VectorConstIterator <int> it = v2.crbegin(); it != v2.crend(); --it)
      cout << "\t" << *it << endl;
#endif // TEST5
}
```

Test Bed Results

```
Test bed did not pass

Lambertson_David_Rezeau.out:

Started program
   > Select the test you want to run:
   >    1. Just create and destroy a Vector.
   >    2. The above plus fill the Vector.
   >    3. The above plus iterate through the Vector.
   >    4. The above plus copy the Vector.
   >    5. The extra credit test: constant and reverse iterators.
   > > 1
   > Create a bool Vector using default constructor
   >    Size:     0
   >    Capacity: 0
   >    Empty?    Yes
   > Create a double Vector using the non-default constructor
   >    Size:     0
   >    Capacity: 10
   >    Empty?    Yes
   > Test 1 complete
Program terminated successfully

Started program
   > Select the test you want to run:
   >    1. Just create and destroy a Vector.
   >    2. The above plus fill the Vector.
   >    3. The above plus iterate through the Vector.
   >    4. The above plus copy the Vector.
   >    5. The extra credit test: constant and reverse iterators.
   > > 2
   > Create an integer vector with the default constructor
   >    Enter numbers, type 0 when done
   >    > 2
   >    > 4
   >    > 6
   >    > 8
   >    > 0
   >    Size:     4
   >    Capacity: 6
   >    Empty?    No
   >    First vector deleted
   > Create a character Vector with non-default constructor
   > Insert user-provided characters in the Vector
   >    Enter characters, type 'q' when done
   >    > a
   >    > b
   >    > c
   >    > d
   >    > e
   >    > f
   >    > g
```

```
   >    Size:  6
   >    Now we will clear the contents
   >    Size:    0
   >    Capacity: 8
   >    Empty?    Yes
   >    Second Vector deleted
   > Test 2 complete
Program terminated successfully

Started program
   > Select the test you want to run:
   >    1. Just create and destroy a Vector.
   >    2. The above plus fill the Vector.
   >    3. The above plus iterate through the Vector.
   >    4. The above plus copy the Vector.
   >    5. The extra credit test: constant and reverse iterators.
   > > 3
   > Create a Vector of strings with the default constructor.
   >    Enter text, type "quit" when done
   >    > CS124
   >    > CS165
   >    > CS235
   >    > CS246
   >    > CS364
   >    > CS499
   >    > quit
   > Use the iterator to display the contents of the vector
   >    CS124
   >    CS165
   >    CS235
   >    CS246
   >    CS364
   >    CS499
   > Which item would you like to look up?
   > > 2
   >    CS235
   > Test 3 complete
Program terminated successfully

Started program
   > Select the test you want to run:
   >    1. Just create and destroy a Vector.
   >    2. The above plus fill the Vector.
   >    3. The above plus iterate through the Vector.
   >    4. The above plus copy the Vector.
   >    5. The extra credit test: constant and reverse iterators.
   > > 4
   > Create a Vector of floats with the default constructor.
   >    Enter numbers, type 0.0 when done
   >    > 1.2
   >    > 2.3
   >    > 3.4
   >    > 4.5
   >    > 5.6
   >    > 6.7
   >    > 7.8
   >    > 8.9
   >    > 9.0
   >    > 0.0
   > Copy the contents of the Vector into a new Vector
   > Use the iterator to display the contents of the Vector
   >    1.2
   >    2.3
   >    3.4
   >    4.5
   >    5.6
   >    6.7
   >    7.8
   >    8.9
   >    9.0
   > Test 4 complete
Program terminated successfully

No Errors
```

lam10006@byui.edu

Grading Criteria

Criteria	Exceptional 100%	Good 90%	Acceptable 70%	Developing 50%	Missing 0%	Weight	Score
Vector interface	The interfaces are perfectly specified with respect to const, pass-by-reference, etc.	lesson00.cpp compiles without modification	All of the methods in Vector and VectorIterator match the problem definition	Both Vector and VectorIterator have many of the same interfaces as the problem definition	The public methods in the Vector class do not resemble the problem definition	20	16
Vector Implementation	The code is robust, efficient, and elegant	All the methods in the Vector and VectorIterator class work	All the methods in the Vector class work	Code exists for all the methods that "resembles" a working solution	None of the non-trivial Vector interfaces are implemented	20	18
Functionality	Passes testBed	Passes three testBed tests	Passes two testBed tests	Passes two testBed tests	Program fails to compile or does not pass any testBed tests	30	30
Style	Great variable names, no errors, great comments	No obvious style errors	A few minor style errors: non-standard spacing, poor variable names, missing comments, etc.	Overly generic variable names, misleading comments, or other gross style errors	No knowledge of the BYU-I code style guidelines were demonstrated	30	21
Extra Credit	Iterator decrement 10%	Constant iterator 10%				20	0
Total							85