Assignment 9

100 points

Purpose

The purpose of this assignment is to give you some experience in dynmaic memory allocation and all that that entails. It should also give you a bit of additional experience in operator overloading. A third purpose of the assignment is to work with templates.

Assignment

This program creates and implements a simplified template vector class. Much of the functionality of the STL vector class does not exist here.

A *driver program* is provided for this assignment to test your implementation. You don't have to write the tests.

Program

You will need to write a single class for this assignment, the `VectorN` class. You will need to implement several functions assoicated with this class.

As a template class, the class should be implemented entirely in a header file. The template class definition should be followed immediately by the implementations of the template class methods.

`class VectorN`

This class contains a dynamically allocated array of data values. Because the array is allocated dynamically, an integer value is also maintained inside the class to track the number of elements in the array.

constructors

The class should have several constructors. One constructor, the default constructor should take no arguments and initialize the vector to hold no data. Another constructor should take a single integer and initialize the vector to that size, with default data values. Another constructor should take two arguments, the first being an array of data to be copied into the vector and the second an integer count of how many elements are in the array.

Destructor

Any class that has dynamic memory better have a destructor.

Copy constructor

The class should also have a proper copy constructor.

`operator =`

The assignment operator should be properly overloaded.

`clear()`

This method should properly set the instance back to a vector of zero elements.

`size()`

Returns the size of the vector, i.e., the number of elements in the dynamically allocated array.

`operator []`

The indexing operator should be overloaded to provide accessor functions for the class. The provided index indicates which value should be accessed from the dynamically allocated array. For speed, no error checking needs to be done.

Don't forget that this operator needs to be overloaded twice, once for reading values out of the instance, and once for writing values to the instance.

`operator ==`

The equality operator should be overloaded to compare two `VectorN`s. The two vectors are considered equal only if they are componentwise equal. For example, (1, 2, 3) is equal to (1, 2, 3), but not to (4, 3, 2). All components must be equal. If the operands are of different sizes, then the vectors are automatically not equal, regardless of the component values.

`operator +`

The addition operator should be overloaded to add two `VectorN`s, producing another `VectorN`. The vectors are added together component by component. The resulting vector will be the same size as the smallest of the two operands.

`operator <<`

Although technically not a method of the class, the output operator should also be overloaded to print out the class, with the format as given below.

Output

A driver program, `vectortest.cc` is provided for this assignment. The purpose of a driver program is to test other pieces that you code. You do not need to write the driver program yourself. A copy of the driver program can also be found on `turing/hopper` at
`/home/turing/duffin/courses/cs689/code/vectortest.cc`

```
#include "vectorN.h"
#include <iostream>

using std::cout;
using std::endl;

#include <string>
using std::string;

int  main()
{
  int test = 1;

  cout << "\nTest " << test++ << ": Default constructor and printing\n" << endl;

  const VectorN<int>  v1;

  cout << "v1: " << v1 << endl;

  cout << "\nTest " << test++ << ": Array constructor and printing\n" << endl;

  const float ar2[] = {11.6, -2.3, 23.45};

  const VectorN<float> v2(ar2, 3);

  cout << "v2: " << v2 << endl;

  cout << "\nTest " << test++ << ": Clear and size\n" << endl;
  VectorN<float> v3(ar2, 3);

  cout << "The size of v3: " << v3 << " is " << v3.size() << endl;

  v3.clear();

  cout << "After clearing, the size of v3: " << v3 << " is ";
  cout << v3.size() << endl;

  cout << "\nTest " << test++ << ": Subscripting\n" << endl;

  const string ar3[] = {"First", "Second", "Third", "Fourth", "Fifth"};
  const VectorN<string> v4(ar3, 5);

  cout << "v4: " << v4 << endl;
  cout << "v4[0]: " << v4[0] << "  v4[1]: " << v4[1] << endl;
  cout << "v4[2]: " << v4[2] << "  v4[3]: " << v4[3] << endl;
  cout << "v4[4]: " << v4[4] << endl;

  VectorN<string> v5(ar3, 5);
  v5[0] = "Sixth";  v5[1] = "Seventh";  v5[2] = "Eighth";  v5[3] = "Ninth";
  cout << "v5: " << v5 << endl;
```

```
cout << "\nTest " << test++ << ": Copy constructor\n" << endl;

const VectorN<float> v6(ar2, 3);
const VectorN<float> v7 = v6;

cout << "v6: " << v6 << " size: " << v6.size() << endl;
cout << "v7: " << v7 << " size: " << v7.size() << endl;

VectorN<string> v8(ar3, 5);
VectorN<string> v9 = v8;

cout << "v8: " << v8 << " size: " << v8.size() << endl;
cout << "v9: " << v9 << " size: " << v9.size() << endl;

v8[1] = "Test";
cout << "Changing..." << endl;
cout << "v8: " << v8 << " size: " << v8.size() << endl;
cout << "v9: " << v9 << " size: " << v9.size() << endl;

cout << "\nTest " << test++ << ": Assignment operator\n" << endl;
VectorN<string> v10(ar3, 5);
VectorN<string> v11;

cout << "v10: " << v10 << endl;
cout << "v11: " << v11 << endl;

v11 = v10;

v10[0] = "Zeroth";

cout << "v10: " << v10 << endl;
cout << "v11: " << v11 << endl;

cout << endl;

// Chained assignment
VectorN<string> v12, v13;

cout << "v11: " << v11 << endl;
cout << "v12: " << v12 << endl;
cout << "v13: " << v13 << endl;

v13 = v12 = v11;
v11[0] = "Success?";

cout << "v11: " << v11 << endl;
cout << "v12: " << v12 << endl;
cout << "v13: " << v13 << endl;

cout << endl;

// Assignment to self

v13 = v13;

VectorN<float> v14(ar2, 3);

cout << "v13: " << v13 << endl;
```

```
  cout << "\nTest " << test++ << ": Equality\n" << endl;

  const int ar6[] = {2, 3, 3};
  const int ar7[] = {2, 3, -3};
  const int ar8[] = {2, 3, 3, 0};
  const int ar9[] = {2, 3, -3, 0};

  const VectorN<int> v18(ar6, 3), v19(ar7, 3);
  const VectorN<int> v20(ar8, 4), v21(ar9, 4);

  cout << v18 << " and " << v18 << " are ";

  if(v18 == v18)
    cout << "equal" << endl;

  else
    cout << "not equal" << endl;


  cout << v18 << " and " << v19 << " are ";

  if(v18 == v19)
    cout << "equal" << endl;

  else
    cout << "not equal" << endl;

  cout << v18 << " and " << v20 << " are ";

  if(v18 == v20)
    cout << "equal" << endl;

  else
    cout << "not equal" << endl;

  cout << v21 << " and " << v19 << " are ";

  if(v21 == v19)
    cout << "equal" << endl;

  else
    cout << "not equal" << endl;


  return 0;
}
```

Output from the correctly functioning driver program should look like the following:

```
turing% vectortest

Test 1: Default constructor and printing

v1: []

Test 2: Array constructor and printing

v2: [11.6, -2.3, 23.45]
```

```
Test 3: Clear and size

The size of v3: [11.6, -2.3, 23.45] is 3
After clearing, the size of v3: [] is 0

Test 4: Subscripting

v4: [First, Second, Third, Fourth, Fifth]
v4[0]: First  v4[1]: Second
v4[2]: Third  v4[3]: Fourth
v4[4]: Fifth
v5: [Sixth, Seventh, Eighth, Ninth, Fifth]

Test 5: Copy constructor

v6: [11.6, -2.3, 23.45] size: 3
v7: [11.6, -2.3, 23.45] size: 3
v8: [First, Second, Third, Fourth, Fifth] size: 5
v9: [First, Second, Third, Fourth, Fifth] size: 5
Changing...
v8: [First, Test, Third, Fourth, Fifth] size: 5
v9: [First, Second, Third, Fourth, Fifth] size: 5

Test 6: Assignment operator

v10: [First, Second, Third, Fourth, Fifth]
v11: []
v10: [Zeroth, Second, Third, Fourth, Fifth]
v11: [First, Second, Third, Fourth, Fifth]

v11: [First, Second, Third, Fourth, Fifth]
v12: []
v13: []
v11: [Success?, Second, Third, Fourth, Fifth]
v12: [First, Second, Third, Fourth, Fifth]
v13: [First, Second, Third, Fourth, Fifth]

v13: [First, Second, Third, Fourth, Fifth]

Test 7: Equality

[2, 3, 3] and [2, 3, 3] are equal
[2, 3, 3] and [2, 3, -3] are not equal
[2, 3, 3] and [2, 3, 3, 0] are not equal
[2, 3, -3, 0] and [2, 3, -3] are not equal

Test 8: Addition

v22: [-5, 2, 6, 14, 0, 0, -6, 8]
v23: [2, 9, -3, 7, 8, -1, 2]
v24 (v22+v23): [-3, 11, 3, 21, 8, -1, -4]
v22: [-5, 2, 6, 14, 0, 0, -6, 8]
v23: [2, 9, -3, 7, 8, -1, 2]
v24 (v23 + v22): [-3, 11, 3, 21, 8, -1, -4]

v25: [Holly, surf, face, light]
v26: [wood, board, book, house]
v25+v26: [Hollywood, surfboard, facebook, lighthouse]
```

Implementation Hints

- Start off at the beginning of the driver program and try to get it working piece by piece. Maintain a working program as you go.

  Start off by commenting out most of the driver program. Uncomment a single test at a time. Get the aspects of the assignment tested by that test working. When that test works, (and only when that test works), uncomment the next test and continue.

Other Points

- A `Makefile` is required. Same as always. Make sure it has appropriate rules for all the pieces involved.
- All functions of the class that can be implemented as methods should be implemented as methods.
- Make sure that you properly free up memory. It's very hard to test this in a driver program, but your code will be examined for this and graded accordingly.
- Submit your program using the electronic submission guidelines posted on the course web site.