

PONDER 01 : VECTOR

Due Saturday at 11:59 PM MST

The first programming assignment of this semester is to write a class with the same functionality as the standard template library (STL) Vector class. As you may recall, the std::vector class behaves much like an array except it grows to accommodate any number of elements. Your vector class will need to be generic. This means it must be able to handle any number of values each being any data-type.

Vector

Your vector will be a template class. It will need to be defined in its own header file (vector.h). The class name must be Vector. Your class will need to support the following operations:

* **Constructors**: Default constructor (setting the capacity to zero initially), a non-default constructor (taking a capacity value as a parameter,) and the copy constructor. Both the non-default constructor and the copy constructor may allocate memory. If there is insufficient memory to allocate a new buffer, then the following exception is thrown:  
  ERROR: Unable to allocate a new buffer for Vector
* **Destructor**: When finished, the class should delete all the allocated memory.
* **operator=**: Copy the data from one vector to another. If there is insufficient memory to allocate a new buffer, then the following exception is thrown:  
  ERROR: Unable to allocate a new buffer for Vector
* **operator[]**: Returns a value from within the container. The passed parameter is an integer, referring to the index in the array of the element to be retrieved. There will be two variants of this method: one that returns by-reference that is not constant, the second is a constant method that returns const by-reference. Unlike std::vector, our array index operator will perform proper bounds checking. In other words, if the passed index is out of bounds, then a constant c-string exception will be thrown: "ERROR: Invalid index"
* **empty()**: Test whether the vector is empty. This method takes no parameters and returns a Boolean value.
* **size()**: Return the vector size. This method takes no parameters and returns an integer value.
* **capacity()**: Return the number of elements that can be held in the currently allocated storage. This method takes no parameters and returns an integer value.
* **clear()**: Clear the contents. This method takes no parameters and returns nothing. Note that you do not need to free the allocated memory; just set the size member variable to zero.
* **push\_back()**: Adds an element to the end of the vector and returns nothing. If the vector is currently empty, then the capacity is set to one. Otherwise, if the vector is currently full, then the capacity is doubled. This follows a similar algorithm to the Expanding Array example in Chapter 4.1 of Procedural Programming in C++. If there is insufficient memory to allocate a new buffer, then the following exception is thrown:  
  ERROR: Unable to allocate a new buffer for Vector
* **begin()**: Return an iterator to the first element in the Vector. It takes no parameters and returns a VectorIterator.
* **end()**: Return an iterator referring to the past-the-end element in the vector. The past-the-end element is the theoretical element that would follow the last element in the vector. It does not point to any element so it must not be de-referenced.

You may also choose to have other private methods as needed. Though you may choose to store items in the vector any way you choose, you might want to consider using a dynamically allocated array. Make sure that robust error checking is done ensuring that the vector is always in a valid state.

As you may recall from CS 165, we usually put class definitions in .h files and the code in .cppfiles. This enables us to separate the interface (or contract the class is designed to fulfill) from the implementation (or *how* the contract is fulfilled). Unfortunately, this does not quite work that way with template classes. In this case, we need to put the entirety of the class definition and all the function implementations in the header file.

Iterator

Additionally, create an iterator class that will traverse the vector. Call this class VectorIterator.Note that Vector::begin() and Vector::end() will return a VectorIterator. This will be a class that allows the client to loop through all the elements in the vector with a standard FOR loop:

for (VectorIterator <int> it = v.begin(); it != v.end(); ++it)

cout << \*it << endl;

This class will need to support the following operators:

* **Constructors**: Default constructor (with a NULL pointer), a copy constructor, and any other non-default constructor you may find helpful
* **operator=**: The assignment operator where the left-hand-side will be a VectorIterator.
* **operator!=**: The not-equals operator that will return a bool where the left-hand-side will be a VectorIterator.
* **operator==**: The equals operator that will return a bool where the left-hand-side will be aVectorIterator.
* **operator\***: The de-reference operator returning a single item by reference of the Vectorclass.
* **operator++**: The increment operator, advancing to the next element in the vector. Both the prefix (++x) and the postfix (x++) increment operators need to be written.

For an example of how this works, please see [CS 165 - 4.5 Iterators](https://content.byui.edu/file/fcfa0eb8-c04e-4270-b79d-9134ffc9c2e5/1/4.5%20Iterators.pdf) and[/home/cs235/week01/examples/container.h](https://content.byui.edu/file/6fd48820-a332-4400-b185-cc19902526da/1/container.html). Note that you do not need to create a constant iterator or implement all the methods described in VectorIterator though you may want to for extra credit.

Driver Program

A driver program (/home/cs235/week01/week01.cpp) is provided. It will pound-include your header file (vector.h) and expect a template class Vector as well as the iterator class VectorIterator to be defined therein. This program will exercise your class, filling the vector with user input and displaying the results.

Additionally, a makefile (/home/cs235/week01/makefile) is provided, looking for the class driver program (week01.cpp) and your header file (vector.h) to be in the current directory. It will build an executable (a.out) and also create a TAR file for submission (week01.tar).

If your class is properly implemented, then testBed will pass with the file unedited. That being said, you will probably need to make a few modifications to the program as you work through your class. Notice the five #defines at the top of the file:

// 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()

If you add comments before each of the #define's except the first one, then only testSimple()will compile. This will enable you to focus on the simplest case without having to weed through hundreds of confusing compile errors. Next, remove the comment before Test2 and get it to work. When you are done, the file should look like the original, unless of course you also getTEST5 to work as well.

Common Mistakes

The most common mistakes students make with this assignment include the following:

* **Failing to make the Vector class a template**. This data structure will need to support all data-types.
* **Putting the implementation of the class in a .cpp file**. Since this vector will need to be a template, everything must be in a header file.
* **Forgetting files in the .tar file.** You may want to un-TAR your file (tar -xf [filename].tar) to make sure it has everything you need to compile and pass the tests.

Test Bed

The testBed for this assignment is:

testBed cs235/week01 week01.tar

You can also run testBed on the executable:

testBed cs235/week01 a.out

Of course, you will need to pass testBed to get full credit on the assignment.

Submitting

You will submit this assignment individually using the Linux submit command. Please:

1. Create a TAR file built from the makefile, which will contain three files:
   * makefile: Directly from /home/cs235/week01/makefile except with your edits on the comment block.
   * vector.h: Your class definition for Vector and VectorIterator.
   * week01.cpp: Unmodified from /home/cs235/week01/week01.cpp.
2. Run the program by hand a few times through all four test cases.
3. Verify your solution with testBed.
4. Submit your file using the submit command. The submit command will prompt you for your instructor, the class (cs235), and the assignment (week01). You submit your file with:

submit week01.tar

Your program will be graded according to the following rubric:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Exceptional 100% | Good 90% | Acceptable 70% | Developing 50% | Missing 0% |
| Vector Interface  30% | The interfaces are perfectly specified with respect to const, pass-by-reference, etc. | week01.cpp compiles without modification | All of the methods in Vector andVectorIteratormatch the problem definition | Both Vector andVectorIteratorhave many of the same interfaces as the problem definition | The public methods in the Vectorclass do not resemble the problem definition |
| Vector Implementation  30% | The code is robust, efficient, and elegant. No redundant or duplicate code | All the methods in the Vector andVectorIteratorclass work | All the methods in the Vector class work | Code exists for all the methods that "resembles" a working solution | None of the non-trivialVectorinterfaces are implemented |
| Functionality  30% | Passes the four VectortestBed tests | Passes three testBed tests | Passes two testBed tests | Passes one testBed test | Program fails to compile or does not pass any testBed tests |
| Style  10% | 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 |

Please make sure to fill out the program header in the makefile with the following information: the amount of coding time required to complete the assignment, and what was the most difficult part of the assignment.

In addition to the above rubric, 10% will be possible if you make your iterator bi-directional. This means it supports both the increment and decrement operator.