COMP322: Assignment 4- Winter 2013

Due at 11:30pm EST, 15th April 2013

1 Introduction

In this assignment, you will implement two things:

1. A new data type TwoWayVector which is similar to the vector class except it will allow for negative indexing as well.
2. An iterator to traverse this collection and make it compatible with the algorithms in the stl library. You will provide 2 versions of this: One to let someone traverse an array in a read only fashion, and one to allow write access.

For this assignment, you are not allowed to use vector or any other stl library class, since the point is you are trying to make a guess for how it is represented!

Part 1: TwoWayVector.cc

A TwoWayVector will represent the notion of an array, but it will do so in a way that hides most of the details of the array from the user of the type. It will be a container that can hold any type of data. To do this, you should make your type templates as outlined in the course notes as well as at `http://www.cplusplus.com/doc/tutorial/templates/`.

Define a class `TwoWayVector` which is templated and has the following functionality:

1. A constructor which initializes any necessary private properties
2. A destructor which frees any memory that was dynamically allocated (i.e. with the new operator) throughout the class
3. A method `push_back` which adds an element given as input to the end of the vector (and returns void)
4. A method `pop_back` which removes an element from the end of the vector (and returns the element that was removed)
5. A method `size` which returns an int representing how many elements are in the collection
6. The operator [ ] should be overloaded in the following manner to take as input an int `index`
   (a) If `index` is a positive number within the bounds of the array (between 0 and size()), you method should return the element at that position
   (b) If `index` is a negative number with absolute value less than or equal to `size()`, your method should return the element at that position but **COUNTING FROM THE END**. For example, array[-1] should return the LAST element of the array.

(c) If `index` is any other value, your method should throw an exception. For simplicity you may simply throw a string with an error message that includes the index. (You may choose to throw a different type of exception if you prefer)

(d) A method `begin` which returns a `TwoWayVectorIterator` (see part 2) to the start of the collection.

(e) A method `end` which returns a `TwoWayVectorIterator` (see part 2) to the element one past the end of the collection.

(f) A method `const_begin` which returns a `TwoWayVectorConstIterator` (see part 3) to the start of the collection.

(g) A method `const_end` which returns a `TwoWayVectorConstIterator` (see part 3) to the element one past the end of the collection.

Methodology: The way you will solve this problem is by internally maintaining three private properties:

1. `T* data` where `T` is the type being used for the template. This is a pointer to an array. In the constructor, you should initialize this pointer to store the address of an array with 10 values in it. When the method `push_back` is called, the data will be added to the array `data`. (Remember that you can use [ ] notation with pointers and arrays interchangeably.) In the case that there is no room in the array, because too many elements have been added already, you should allocate space for a larger array with a size double the current size, copy the elements from `data` into your new array, and finally store into `data` the new array's address. This must be done in such a way that any arbitrary size array can be created.

2. `int capacity` This property stores how large the array `data` is.

3. `int nextFree` This property stores the next free position of `data`.

Your solution must obey proper memory management and thus shouldn't create any memory leaks. You can test your TwoWayIterator as you are going by separately adding a main function although the main function is not required for this part of the assignment.

Part 2: Implementing an iterator (write access)

Now we will implement an iterator to traverse a `TwoWayVector` in a fashion that allows for writing to elements while iterating. The iterator will also be a templated type, since it will have different types depending on the sort of `TwoWayVector` it is iterating over.

Define a new type `TwoWayVectorIterator`. A `TwoWayVectorIterator` should have the following properties:

1. `TwoWayVector<T> * vector`. This property is a pointer to the vector that is being iterated over.

2. `int currentPosition`. This property represents where in the array the iterator is currently at.

With these 2 properties, the iterator maintains the state necessary to traverse the collection. An iterator that is at the beginning of the collection would have `currentPosition` set to be 0. An iterator that is at the last element in the collection would have `currentPosition` set to be `vector.size() - 1`, and an iterator with any other value is outside the range of the collection.

You should define the following methods on the iterator by overloading several operators:

1. Define a constructor which takes as input the address of a `TwoWayVector` as well as an initial position. It should set the 2 properties of the iterator accordingly.
2. Define `==` to check whether the `vector` is the same as well as if `currentPosition` is the same.
3. Define `!=` to compare two iterators (opposite of ==)
4. Define `++` to increment the position by one. You should do this for both pre and post fix notation. (i.e. ++x vs x++)
5. Define `=` to assign one iterator to another. This function should assign values to the iterator being assigned to AND return the iterator being assigned to in order to allow statements like (x = y = z)
6. Define `+` to take as input an int and add a value to the current position.
7. Define `-` to take as input an int and subtract a value from the current position.
8. Define `<` to take as input another iterator and check whether one iterator is less than the other. `<` is determined based only on the value of `currentPosition`
9. Define `*` operator to get the value that the iterator is currently "pointing" at. For this iterator, because it is possible to use the iterator to write values, you should return a reference (simply add a & in the method header after the return type)

The trick to getting this correct is to think of all the weird cases one might use these symbols with other types and try to match the usage. For example, if x is an int, the statements `x++;` and `++x;` do the exact same thing by themselves. However, `int y = x++;` and `int y = ++x;` do slightly different things (the first one stores x into y before incrementing). With equals, the most common case is `x = y`, but one could also write something like : `z = x = y` which means "Store into z the result of the expression `x = y`." The expression `x = y` does something (stores y into x) and also returns a value.

You can then write the `begin()` and the `end()` methods for the TwoWayVector class so that they return `TwoWayVectorIterator`s with properties such that they represent the beginning and end of the container respectively.

You can test your method by doing something like the following:

```
TwoWayVector<int> numbers;
numbers.push_back(3);
numbers.push_back(2);
for (TwoWayVectorIterator current = numbers.begin();
current != numbers.end();
current++)
{
    cout << *numbers;
}
```

You can also see that you'll be able to use your type with methods such as `find_if` that is part of the algorithms.h file.

Part 3: Implementing a vector with read only access

This last part is very short once you have implemented the last two parts. Now you will implement a type `TwoWayVectorConstIterator` which is a *read only* iterator. The point is to see that *read only* is determined not based on any sort of keyword, but rather based on what operators you are allowed to use for your iterator. The code for `TwoWayVectorConstIterator` should be exactly the same as `TwoWayVectorIterator` except the `*` operator should not return a reference to the data. It should return a copy of the data. You can do this by removing the & from the header.

Now that you've done this, you can finish the `TwoWayVector` class by adding the `const_begin` and `const_end` methods. You can test the difference by trying to use your iterator to change the value of your container. For example:

```
TwoWayVector<int> numbers;
numbers.push_back(3);
numbers.push_back(2);
for (TwoWayVectorIterator current = numbers.begin();
current != numbers.end();
current++)
{
    *numbers = *numbers * 2;
}
```

will work. But if you use the other type, it will not.

2 Submitting your assignment

What To Submit

```
TwoWayVector.cc
TwoWayVectorIterator.cc
TwoWayVectorConstIterator.cc
```

`Any other files needed for compiling (e.g.  header files`
`CompileCommand.txt` : In this file you should put the exact command you used to compile your program with g++. This will vary depending on exactly how you connected your files together and will let the TA run your code more easily.
`Confession.txt` (optional) In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your problem, it may lead the TA to give you more partial credit. On the other hand, it also may lead the TA to notice something that otherwise he or she would not.