Overview

In this lab you will add some details to the Vec<T> class we discussed in Monday’s lecture and also test it. Refer back to the lecture notes for a detailed discussion of the class. Before getting started, download the code for the class from

http://www.cs.rpi.edu/~stewart/cs2/labs/week10/Vec.h

This is a version of Vec that includes the functions written during lecture.

It will also be extremely helpful to look at the second version of the str class, which we completed during lecture on Monday. It is posted on-line at

http://www.cs.rpi.edu/~stewart/cs2/week09/str_ver2/str.h
http://www.cs.rpi.edu/~stewart/cs2/week09/str_ver2/str.cpp

Checkpoints

1. In lecture we have stressed the fact that classes having dynamic memory (member variables that are pointers to dynamically-allocated objects or arrays) should have an assignment operator and a copy constructor. When a class does not include these, the compiler provides “default” versions that just copy the values of the member variables. This is called a shallow copy and is the way the current version of Vec works. The purpose of this checkpoint is to demonstrate the significance of this problem. In the next checkpoint, you will fix it.

Write a short test main program that demonstrates why using the default assignment operator is a problem. You will need to create two Vec (make them of ints or of doubles) objects — call them a and b. Make b a “copy” of a, either by assignment or by using the copy constructor. Print out the contents of both a and b. (Note: the output stream operator is not defined for Vec, so you will need a loop to output the contents — or write your own function.) Then make changes to the contents of one of them. Finally, print out the contents of both (again).

Aside: note that the declaration
Vec<int> b = a;

where a is a Vec<int>, uses the copy constructor, not the assignment operator (despite the syntax).

To complete this checkpoint, compile and run your program. This program is likely to cause a “Debug Assertion” failure in Visual Studio. Show the output to one of the TAs AND explain briefly (a) why that particular output occurred and (b) why the “Debug Assertion” failure occurred.

2. Write the copy constructor and the assignment operator. Both have prototypes that are “commented out” in the class declaration. There is also a prototype for a private member function called copy that does the actual work of copying. This is called by the copy constructor, which is completely defined inside the class declaration. (Therefore, once copy is written, you do not need to do anything else for the copy constructor other than remove the comment symbol // before its declaration.) The assignment operator should also call the copy function after it checks for self-assignment.

This will be the first time you write a templated member function. Look carefully at Monday’s lecture notes and the examples of the create and push_back functions to help you with the syntax.

To complete this checkpoint, write the functions, and add any code that you need to the main program to ensure that both the assignment operator and the copy constructor are tested. Compile and run the program. Show the results to one of the TAs. You should not have the “Debug Assertion” failure anymore.

3. Write and demonstrate the erase member function. Look carefully at the erase member function from the str class. The function should take a single iterator as an argument, remove the entry corresponding to this iterator from the Vec, and return an iterator pointing to what was the next item in the Vec. As a result of a single call to erase, the size of the Vec should be one fewer.

Add code to your test main program to demonstrate that your erase function is working correctly. This should include demonstrating that the size of the Vec, as computed by the size member function, is correctly changed by erase.
4. Write and demonstrate an operator\texttt{==} for \texttt{Vec}'s. This should be a non-member function and have the prototype:

\begin{verbatim}
    template <class T>
    bool operator==( const Vec<T>& left, const Vec<T>& right );
\end{verbatim}

The operator return true if and only if the contents of the two \texttt{Vec}'s are exactly the same. Comparing each individual entry must use \texttt{operator==} on type \texttt{T}. The compiler will issue an error message if a type is used in \texttt{Vec} for which this operator does not exist.