Overview

In this lab you will implement and test several of the *cs2list* member functions. You will also do course evaluations at the end of the lab. This is the last lab!

Please download two files from the course web site:

http://www.cs.rpi.edu/~stewart/cs2/labs/week14/cs2list.h
http://www.cs.rpi.edu/~stewart/cs2/labs/week14/testcs2list.cpp

The first is the header file defining the *cs2list* class plus the node and iterator classes. The second is a main program to test this class. You will need to add substantially to both.

Thinking about writing linked-list code is best done visually. In all cases, draw a picture of the linked list before your code begins to work. Use this to figure out what you need to do and in what order to need to do it! Draw separate pictures for each special case you must handle.

Debugging linked list code can be challenging. Reasons for this include the difficulty of visualizing what’s happening and the fact that the impact of errors (incorrectly assigned pointers, usually!) often appears later in the code and in a different place from where the mistake was actually made. Three methods can be used in debugging, often in combination, to find and fix the errors. The first is writing functions to print the contents of a list. Calls to this function should be inserted in many places in the code and compared to the output you expect. The second is testing the functions on small examples and special cases. The third is using a debugger to step through the contents of a linked list, following the pointers. Try to apply these techniques as you work toward solutions to the following checkpoints.

Checkpoints

1. Write and test the *pop_front* member function. Start by re-examining the *push_front* member function and then outline carefully what you need to do. Here are several suggestions:

   - Use a local pointer variable (not a new node!) to point to the node that needs to be deleted.
• Check for the special case of deleting the only node in the linked list.
• Make sure you decrement the size counter.
• Do not explicitly delete the node until the very end of the function, after it has been removed from the linked list.

The code in the main program has a number of checks to make sure that you’ve done this correctly.

2. Write and test both the push_back and pop_back member functions. Use push_front and pop_front to guide you, both in terms of the logic and in terms of the format of the function definitions (in cs2list.h). Add tests to the main program.

3. Write the insert function. Consider the prototype that you need to write when you insert this function at the bottom of cs2list.h:

```cpp
template <class T>
cs2list<T> :: iterator
cs2list<T> :: insert( iterator itr, const T& v )
```

At first this looks odd. The return type is written `cs2list<T> :: iterator`, but inside the argument list we just use `iterator`. These refer to EXACTLY the same type. Why then is there a difference? The answer is “scope”: the return type is specified outside the scope of the cs2list<T> class and therefore the class scope operator `cs2list<T>::` needs to be used. The inside of the argument list is within the scope of the cs2list<T> class and so the scope operator is no longer needed.

Now, let’s look at the arguments themselves and use this to explain the exact purpose of the function. The first argument is an iterator that indicates a node in the list. The pointer to this node is stored in the iterator:

```cpp
itr.ptr_
```

and because cs2list<T> is a friend of the iterator class, the insert function has direct access to the value (using the syntax `itr.ptr_`). The second argument to the function is the value to be added to the list. The job of the function is to

(a) create a new node containing the value,
(b) insert the node before the node pointed to by \texttt{itr.ptr}, and
(c) return an iterator that contains the address of the new node as its \texttt{ptr} value.

Be sure that you manipulate the pointers correctly (draw a picture, Draw A Picture, DRAW A PICTURE). Be sure that you consider the special case that \texttt{itr.ptr} == \texttt{head}.

4. Write the function \texttt{destroy_list}. It is currently in the code as an empty function. This function should remove and delete each node in the list. It should also reset the member variables properly. Test your function by writing main function code that (a) calls \texttt{clear}, (b) prints out the size of the list, (c) adds new values to the list, and finally (d) prints its contents.