Fall 2006 EE380L Quiz2: NAME:

RULES: No internet searches. You may work on this examination for 2 hours (no longer). The time can be distributed over at most two contiguous intervals where each interval is at least 15 minutes long. When you elect to work on the examination, you must be in a private work environment with no outside assistance of any kind. Text books are permitted, as are all EE380L materials (including the EE380L web sites). You are not permitted to consult any "research papers" or published articles of any kind (only text books). You may use a compiler to write and test possible solutions. However, it is not necessary that your solution compile or run in order to receive a perfect score (be wary, as the compiler can be a time sink).

Addendum: You are permitted a grand total of four hours on this assignment. You can spend at most 2 hours actively working on your solution. In addition to the two hours of active work, you are permitted to "daydream" about the problem for at most an additional two hours. That means, if you are thinking about this problem on your ride home today, then that time spent in the car/bus will not count against the time you have available to spend sitting at your desk working on the exam. You can think about the problem while you're in the shower, lay awake at night thinking about it, etc. However, you must not abuse this license. After two hours of stewing over the problem, any additional time you spend thinking on the problem counts against the time allotted. Time spent in class today does not count against your four hours.

By attaching your name to this paper, you affirm the following statement:
I have fully complied with both the letter and the spirit of the academic honesty policies for this examination. I recognize that this examination is intended to be an evaluation of my ability to develop my own designs. Therefore, I recognize that searching for and/or reproducing designs, approaches or solutions developed by others (whether published or not) is in violation of the academic honesty rules. I attest that the following log accurately reflects the time I spent working on this examination:

Date	time started	time completed

Assume I have a library of generic functions, designed according to the conventions of the C++ standard library. These functions are invoked by passing two arguments, the arguments are iterators (or possibly pointers) to the beginning and end of a container, just like in the C++ standard library.

Some of the functions require random access iterators, others require only forward iterators. In this quiz, you will write a template class (or classes) and a template function (or functions) so that the functions can be used with any type of iterator.

Consider the case of a hypothetical function *doit* that will only work with random access iterators (because it uses one of the two iterator functions that are only defined for random access iterators, i.e,. pointer subtraction or pointer/integer addition).

```
template <typename iterator>
void doit(iterator b, iterator e) {
  // blah blah
  int len = (e - b);
  // blah blah
}
```

Your task is to make it possible for me to call this function using some forward iterator as follows (imagine that Slist is a singly-linked list with a forward iterator)

```
Slist s = // blah blah
doit(powerUp(s.begin()), powerUp(s.end()));
```

Obviously, since it is not possible to perform pointer subtraction in constant time using the Slist::iterator type, *doit* will probably have worse time complexity in this context than it would "normally" have. However, since it could not normally be used at all on singly-linked lists (would not compile), we're willing to pay the time complexity price.

Write the powerUp function. You can make the following assumptions:

- iterator_traits<T> works for all iterator types, pointer types and is available for you to use.
- For any iterator *p* it is always safe to perform ++p or to perform p = = q. However, *p may result in an error.
- if p points to the end of the container, then ++p may not change the value of p. If p points to the end of the container, then the result of zero or more invocations of ++p is guaranteed to NOT point at any of the locations inside the container (i.e., iterators are not permitted to wrap around and go back to the beginning).
- You only need to worry about five of the six fundamental operations on iterators/pointers (i.e., you don't need to worry about expressions like p < q). Recall the six fundamental operations are *p, ++p, --p, p = = q, p – q and p + k. You do not need to support decrement (--p), and you can assume that when addition is perform (p + k) that k is positive. You can assume that all of the

functions in the library are written to only use these five functions, and that negative integers are never added to iterators.
- Your powerUp function needs to work correctly for pointers as well as iterators.

Your powerUp function must not make any function slower than it is currently. In other words, if doit(p, q) compiles and runs, then doit(powerUp(p), powerUp(q)) must compile and run at least as fast.