Project 1 FAQ

1. **I'm thinking of implementing a private member function that returns a pointer to a nested class/struct. How do I do that?**

Consider this:

```
class M
{
   ...
   struct N
   {
      ...
   };
   N* f();
   ...
};

N* M::f()      // Error!  Compiler doesn't recognize N.
{
   ...
}

M::N* M::f()  // Oh, M's N!  This will compile.
{
   ...
}
```

There's a C++ language rule that says roughly that the return type part of the definition of a member function of a class M isn't in the scope of M, so a name from M (like N) needs to be qualified (like M::N). Oddly enough, the parameter list is in the scope of M, so if M declared a function void g(N* n), the implementation could begin void M::g(N* n) if you wish (or void M::g(M::N* n), of course).

2. Does this compile? If so, what is its output?

```
#include <iostream>
using namespace std;

void f(const int& x, int& y)
{
   y = x + 1;
}

int main()
{
   int k = 0;
   f(k, k);
   cout << k << endl;
   return 0;
}
```

Yes, it compiles, and the output is 1. The declaration `const int& x;` says that *through the name x*, the function *is not* allowed to modify the object `x` refers to, so an assignment like `x = 42` would not compile. The declaration `int& y;` says that *through the name y*, the function *is* allowed to modify the object y refers to. So in the call `f(k, k)`, the function can modify `k` if it does so through the name `y`.

3. Can I test for memory leaks?

That's a hard problem in general. If we limit it to checking whether we leak any linked list nodes, where each linked list node contains an object of the type stored in the Sequence, then we can do it this way: Declare the item type of the Sequence to be a special class we'll write, one that will keep track of all creations and deletions of objects of that special type. We can do this by instrumenting every constructor and the destructor of that special class. Then we run our tests and see if the number of destructions equals the number of constructions, as shown in some sample code (`memleak.cpp`).