Project 2 FAQ

Question:

Do I need to make the list circular? May I use a tail node?

Answer:

To the first part of the question, no. We wanted to leave you a fair amount of flexibility in how you implemented your lists, so whether or not it is circular is up to you. However, making it circular will make it much easier to solve the Josephus permutation.

The answer to this depends on whether or not your list is circular. If it is circular, then the head and tail nodes would always be pointing to one another. That's a little bit redundant, so if your list is circular, don't bother with a tail node. On the other hand, if you elected not to make the list circular, than you should use a tail node.

Also, if you do use a tail node, follow the same guidelines we gave you for the head node, in the programming notes section of the project description.

Question:

How do I implement reverse without physically reversing all of the links?

Answer:

Well, here's a hint: The only way a user can iterate a pointer is with the Next method. So, you can just keep track of when the list is reversed and adjust your logic accordingly. Also remember that when you reverse the list the push and pop operations should also be reversed. For example: PopFront becomes PopBack, when the list is reversed.

Question:

Visual studio throws an error saying that it can't find 'unistd.h'

Answer:

There's something wrong with preprocessor directives in random.h. If you are using visual studio make the following changes to them:

```
//#ifdef VISUAL_STUDIO
#include <process.h>
//#else
//#include <unistd.h>
```

//#endif

This will still throw a warning, but it should compile just fine under visual studio. When you test your code on black, make sure you uncomment the preprocessor directives.

Question:

Ok, let's say you declare a linked list but do not add any items to it. Do we want functions such as beginning and end to return null? For the find function if the element does not exist should that also return null?

Answer:

If the only thing in the list is Head (or tail) nodes, then you should return null from any method that returns a pointer into the list. Your linked list should never return a pointer from the head or tail nodes.

If find fails, than return null.

Question:

Does the copy constructor need to be a deep one, or can it just pass the addresses of the existing elements?

Answer:

The copy constructor must do a deep copy. No two linked lists should use the same nodes.
