Array-backed List - Implementation Details:

1. What is the running time of `insertFront()`?

```
[0] [1] [2] [3] [4]
C S 2 2 5
```

2. What is our resize strategy?

Resize Strategy #1:

```
[0] [1] [2] [3] [4]
C S 2 2 5
```

Resize Strategy #2:

```
[0] [1] [2] [3] [4]
C S 2 2 5
```

3. What is the running time of `get()`?

```
T* arr:     C 5 2 2 5
[0] [1] [2] [3] [4]
```

<table>
<thead>
<tr>
<th>Stack ADT</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function Name</td>
<td>Purpose</td>
</tr>
<tr>
<td>Insert/Remove at <code>front</code></td>
<td>Singly Linked List</td>
</tr>
<tr>
<td>Insert after a <code>given</code> element</td>
<td></td>
</tr>
<tr>
<td>Remove after a <code>given</code> element</td>
<td></td>
</tr>
<tr>
<td>Insert at <code>arbitrary</code> location</td>
<td></td>
</tr>
<tr>
<td>Remove at <code>arbitrary</code> location</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Queue ADT</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function Name</td>
<td>Purpose</td>
</tr>
<tr>
<td>Insert/Remove at <code>front</code></td>
<td>Singly Linked List</td>
</tr>
<tr>
<td>Insert after a <code>given</code> element</td>
<td></td>
</tr>
<tr>
<td>Remove after a <code>given</code> element</td>
<td></td>
</tr>
<tr>
<td>Insert at <code>arbitrary</code> location</td>
<td></td>
</tr>
<tr>
<td>Remove at <code>arbitrary</code> location</td>
<td></td>
</tr>
</tbody>
</table>
Stack and Queue Implementations

Stack.h

```cpp
#ifndef STACK_H
#define STACK_H

#include "List.h"

template <typename T>
class Stack {
public:
    void push(T & t); 
    T & pop();
    bool isEmpty();

private:
    List<T> list;
};
#endif
```

Stack.cpp

```cpp
#include "Stack.h"

template <typename T>
void Stack::push(T & t) {
    list_.add(t, 0);
}

template <typename T>
T & Stack::pop() {
    return list_.remove(0);
}

bool Stack::isEmpty() {
    return list_.isEmpty();
}
```

Implication of Design

1. Who manages the lifecycle of the data?
2. Is it possible to store a NULL as the data?
3. If the data is manipulated by user code while stored in our data structure, are the changes reflected within our data structure?
4. Speed

<table>
<thead>
<tr>
<th>Storage by Reference</th>
<th>Storage by Pointer</th>
<th>Storage by Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lifecycle management of data?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Possible to insert NULL?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>External data manipulation?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Speed</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Three designs for data storage in data structures:

1. T & data
2. T * data
3. T data

CS 225 – Things To Be Doing:

1. Programming Exam A starts Feb. 13 (next Tuesday)
2. MP2 due Feb. 12 (next Monday)
3. lab_inheritance due Sunday
4. Daily POTDs