List Implementation #2: ____________________

```
List.h
1 #pragma once
2 template <typename T>
3 class List {
4     public:
5         /* ... */
6         private:
7     private;
8 }
```

Implementation Details and Analysis:
What is the running time of `insertFront()`?

<table>
<thead>
<tr>
<th>C</th>
<th>S</th>
<th>T</th>
<th>2</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

What is our resize strategy?

---

Array Resize Strategy #2:

- total copies across all resizes: _________
- total number of insert operations: _________
- average (amortized) cost of copies per insert: _________

Running Time:

<table>
<thead>
<tr>
<th>Insert/Remove at <code>front</code></th>
<th>Singly Linked List</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert after a <code>given</code> element</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove after a <code>given</code> element</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert at arbitrary location</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove at arbitrary location</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A List implementation in `std`
- `std::vector` implements a list with dynamic growth
- `#include <vector>` to use it!
- Documentation widely available, including on CBTF exams
Stack ADT

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Queue ADT

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Stack and Queue Implementations

Stack.h

```cpp
#pragma once
#include <vector>

template <typename T>
class Stack {
public:
    void push(T & t);
    T & pop();
    bool isEmpty();
private:
    std::vector<T> list_;};
```

Stack.hpp

```cpp
template <typename T>
void Stack<T>::push(const T & t) {
    list_.push_back(t);
}

template <typename T>
const T & Stack<T>::pop() {
    const T & data = list_.back();
    list_.pop_back();
    return data;
}
```

Three designs for data storage in data structures:

1. T & data

2. T * data

3. T data

Implication of Design

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

CS 225 – Things To Be Doing:

1. Programming Exam A starts Feb. 14 (next Thursday)
2. MP2 due Feb. 11 (next Monday); MP3 released Tuesday
3. lab_inheritance due Sunday
4. Daily POTDs