List ADT

• Linked Memory Implementation (“Linked List”)
  • $O(1)$ insert/remove at front/back
  • $O(1)$ insert/remove after a given element
  • $O(n)$ lookup by index

• Array Implementation (“Array List”)
  • $O(1)$ insert/remove at front/back
  • $O(n)$ insert/remove at any other location
  • $O(1)$ lookup by index
Queue ADT

• [Order]:

• [Implementation]:

• [Runtime]:

CS 225 So Far...
CS 225 So Far...

Stack ADT

• [Order]:

• [Implementation]:

• [Runtime]:
What type of implementation is this Queue?

How is the data stored on this Queue?
What type of implementation is this Queue?

How is the data stored on this Queue?

```cpp
Queue<int> q;
q.enqueue(3);
q.enqueue(8);
q.enqueue(4);
q.dequeue();
q.enqueue(7);
q.dequeue();
q.dequeue();
q.enqueue(2);
q.enqueue(1);
q.enqueue(3);
q.enqueue(5);
q.dequeue();
q.enqueue(9);
```
#pragma once

template <typename T>
class Queue {
public:
  void enqueue(T e);
  T dequeue();
  bool isEmpty();
private:
  T *items_;
  unsigned capacity_;  // Original: capacity
  unsigned count_;     // Original: count
};

Queue<char> q;  // Original: Queue
q.enqueue(m);
q.enqueue(o);
q.enqueue(n);
...
q.enqueue(d);
q.enqueue(a);
q.enqueue(y);
q.enqueue(i);
q.enqueue(s);
q.dequeue();
q.enqueue(h);
q.enqueue(a);
Type of headache

Migraine

hypertension

high pressure

Segmentation fault
Implications of Design

1. ```cpp
   class ListNode {
   public:
   T & data;
   ListNode * next;
   ... 
   }
```

2. ```cpp
   class ListNode {
   public:
   T * data;   ...
   }
```

3. ```cpp
   class ListNode {
   public:
   T data;   ...
   }
# Implications of Design

<table>
<thead>
<tr>
<th></th>
<th>Storage by Reference</th>
<th>Storage by Pointer</th>
<th>Storage by Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Who manages the lifecycle of the data?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is it possible for the data structure to store NULL?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>If the data is manipulated by user code while in our data structure, is the change reflected in our data structure?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Speed</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data Lifecycle

Storage by reference:

1 Cube c;
2 myStack.push(c);

Storage by pointer:

1 Cube c;
2 myStack.push(&c);

Storage by value:

1 Cube c;
2 myStack.push(c);
Possible to store NULL?

Storage by reference:

```cpp
class ListNode {
    public:
       T & data;
       ListNode * next;
    ListNode(T & data) : data(data), next(NULL) { }
};
```

Storage by pointer:

```cpp
T ** arr;
```

Storage by value:

```cpp
T * arr;
```
Data Modifications

```java
1 Cube c(1);
2 myStack.push(c);
3 c.setLength(42);
4
5 Cube r = myStack.pop();
6 // What is r's length?
```
Iterators

Suppose we want to look through every element in our data structure:
Iterators encapsulated access to our data:
Iterators

Every class that implements an iterator has two pieces:

1. [Implementing Class]:
Iterators

Every class that implements an iterator has two pieces:

2. [Implementing Class’ Iterator]:
   • Must have the base class std::iterator
   • Must implement
     
     operator*

     operator++

     operator!=
```cpp
#include <list>
#include <string>
#include <iostream>

struct Animal {
    std::string name, food;
    bool big;
    Animal(std::string name = "blob", std::string food = "you", bool big = true) :
        name(name), food(food), big(big) { /* nothing */ }
};

int main() {
    Animal g("giraffe", "leaves", true), p("penguin", "fish", false), b("bear");
    std::vector<Animal> zoo;
    zoo.push_back(g);
    zoo.push_back(p);  // std::vector's insertAtEnd
    zoo.push_back(b);
    for ( std::vector<Animal>::iterator it = zoo.begin(); it != zoo.end(); it++ ) {
        std::cout << (*it).name << " " << (*it).food << std::endl;
    }
    return 0;
} 
```
```cpp
#include <list>
#include <string>
#include <iostream>

struct Animal {
    std::string name, food;
    bool big;
    Animal(std::string name = "blob", std::string food = "you", bool big = true) :
        name(name), food(food), big(big) { /* none */ }
};

int main() {
    Animal g("giraffe", "leaves", true), p("penguin", "fish", false), b("bear");
    std::vector<Animal> zoo;

    zoo.push_back(g);
    zoo.push_back(p);  // std::vector’s insertAtEnd
    zoo.push_back(b);

    for ( const Animal & animal : zoo ) {
        std::cout << animal.name << " " << animal.food << std::endl;
    }
    return 0;
}
```