CS 225
Data Structures

Feb. 12 – Iterators
Wade Fagen-Ulmschneider
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
CS 225 So Far...

Queue ADT

• FIFO: First in, first out – like a line/queue at a shop
• Implemented with a list, O(1) enqueue/dequeue

Stack ADT

• LIFO: Last in, first out – list a stack of papers
• Implemented with a list, O(1) push/pop
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?

```
#include <iostream>

int main() {
    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);
    return 0;
}
```
#ifndef QUEUE_H
#define QUEUE_H

template <class QE>
class Queue {
  public:
    Queue(); // ...etc...
    void enqueue(QE e);
    QE dequeue();
    bool isEmpty();

  private:
    QE *items_;
    unsigned capacity_;  
    unsigned count_;  
    unsigned entry_;  
    unsigned exit_;  
  }

};
#endif
Implications of Design

1. ```
struct ListNode {
    T & data;
    ListNode * next;
    ListNode(T & data) : data(data), next(NULL) { }
};
```
## 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</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the data?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is it possible for the data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>structure to store NULL?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>If the data is manipulated</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>by user code while in our</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data structure, is the change</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>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. `Sphere s;`
2. `myStack.push(s);`

Storage by pointer:

1. `Sphere s;`
2. `myStack.push(&s);`

Storage by value:

1. `Sphere s;`
2. `myStack.push(s);`
Possible to store NULL?

Storage by reference:

```c
struct ListNode {
    T & data;
    ListNode * next;
    ListNode(T & data) : data(data), next(NULL) { }
};
```

Storage by pointer:

```c
T ** arr;
```

Storage by value:

```c
T * arr;
```
### Data Modifications

```cpp
1  Sphere s(1);
2  myStack.push(s);
3
4  s.setRadius(42);
5
6  Sphere r = myStack.pop();
7  // What is r’s radius?
```
Speed
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!=`
#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::list<Animal> zoo;
    zoo.push_back(g);
    zoo.push_back(p); // std::list's insertAtEnd
    zoo.push_back(b);
    for ( std::list<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::list<Animal> zoo;
  zoo.push_back(g);
  zoo.push_back(p);  // std::list's insertAtEnd
  zoo.push_back(b);
  for ( const Animal & animal : zoo ) {
    std::cout << animal.name << " " << animal.food << std::endl;
  }
  return 0;
}
```