#12: Stacks, Queues, and Design

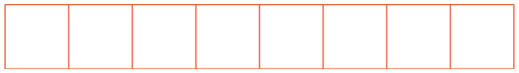
September 23, 2020 · *G Carl Evans*

Data Structures Review

- 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

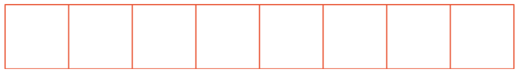
	Queue	Stack
Operations + Data Order:		
Implementation:		
Runtime:		

Example 1



```
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);
```

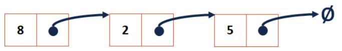
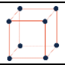
Example 2



```
Queue<char> q;
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');
```

Accessing Every Element in Our List / Queue / [Anything]

Suppose we want to look through every element in our data structure. What if we don't know what our data structure even looks like?

	Linked List
	Array
	Hypercube

Iterators

In C++, iterators provide an interface for client code access to data in a way that abstracts away the internals of the data structure.

An instance of an iterator is a current location in a pass through the data structure:

Type	Cur. Location	Current Data	Next
Linked List			
Array			
Hypercube			

The iterator minimally implements three member functions:

`operator*`, Returns the current data
`operator++`, Advance to the next data
`operator!=`, Determines if the iterator is at a different location

Implementing an Iterator

A class that implements an iterator must have two pieces:

1. [Implementing Class]:

2. [Implementing Class' Iterator]:
   A separate class (usually an internal public member class) that extends **`std::iterator`** and implements an iterator.

Using an Iterator

stlList.cpp

```
#include <vector>
#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;
}
```

Q: What does the above code do?

For-Each loop with Iterators

stlList-forEach.cpp

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

CS 225 – Things To Be Doing:

1. mp_lists released
2. lab_quacks starts today
3. Daily POTDs