CS 225

Data Structures

Sept. 27 – Queues and Iterators

Queue.h

```
 1 #ifndef QUEUE_H
 2 #define QUEUE_H
 3
 4 template <class QE>
 5 class Queue {
 6   public:
 7
 8
 9
10
11
12
13
14
15
16
17   private:
18
19
20 };
21
22 #endif
```

Queue.h

```
 1 #ifndef QUEUE_H
 2 #define QUEUE_H
 3
 4 template <class QE>
 5 class Queue {
 6   public:
 7     void enqueue(QE e);
 8     QE dequeue();
 9     bool isEmpty();
10
11   private:
12     struct QueueNode {
13       QE data;
14       QueueNode *next;
15       QueueNode(QE data) ... {}
16     }
17     QueueNode *entry_, *exit_;
18     int size_;
19
20 };
21
22 #endif
```

What type of implementation is this Queue?

How is the data stored on this Queue?

Which pointer is “entry” and which pointer is “exit”?

What is the running time of enqueue()?

What is the running time of dequeue()?

Queue.h

```
 1  #ifndef QUEUE_H
 2  #define QUEUE_H
 3
 4  template <class QE>
 5  class Queue {
 6    public:
 7      void enqueue(QE e);
 8      QE dequeue();
 9      bool isEmpty();
10
11    private:
12      QE *items_;
13      unsigned capacity_;
14      unsigned count_;
15
16
17
18  };
19
20  #endif
21
22
```

What type of implementation is this Queue?

How is the data stored on this Queue?

```
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);
```

Queue.h

```
 1 #ifndef QUEUE_H
 2 #define QUEUE_H
 3
 4 template <class QE>
 5 class Queue {
 6   public:
 7     Queue(); // …etc…
 8     void enqueue(QE e);
 9     QE dequeue();
10     bool isEmpty();
11
12   private:
13     QE *items_;
14     unsigned capacity_;
15     unsigned count_;
16     unsigned entry_;
17     unsigned exit_;
18
19 };
20
21 #endif
22
```

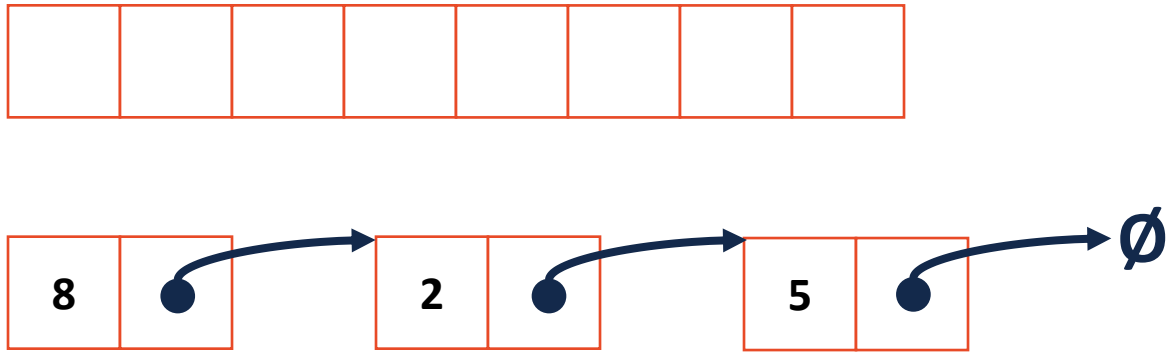

```
Queue<char> q;
q.enqueue(m);
q.enqueue(o);
q.enqueue(n);
…
q.enqueue(d);
q.enqueue(a);
q.enqueue(y);
q.enqueue(i);
q.enqueue(s);
q.dequeue();
q.enqueue(h);
q.enqueue(a);
```

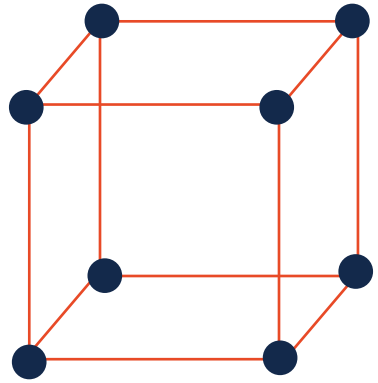
stlList.cpp

```
 1 #include <list>
 2 #include <string>
 3 #include <iostream>
 4
 5 struct Animal {
 6   std::string name, food;
 7   bool big;
 8   Animal(std::string name = "blob", std::string food = "you", bool big = true) :
 9     name(name), food(food), big(big) { /* none */ }
10 }
11
12 int main() {
13   Animal g("giraffe", "leaves", true), p("penguin", "fish", false), b("bear");
14   std::list<Animal> zoo;
15
16   zoo.push_back(g);
17   zoo.push_back(p);   // std::list's insertAtEnd
18   zoo.push_back(b);
19
20   for ( std::list<Animal>::iterator it = zoo.begin(); it != zoo.end(); it++ ) {
21     std::cout << (*it).name << " " << (*it).food << std::endl;
22   }
23
24   return 0;
25 }
```

Iterators

Iterators give client code access to traverse the data!

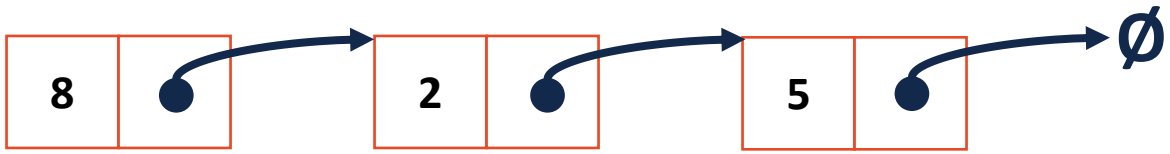


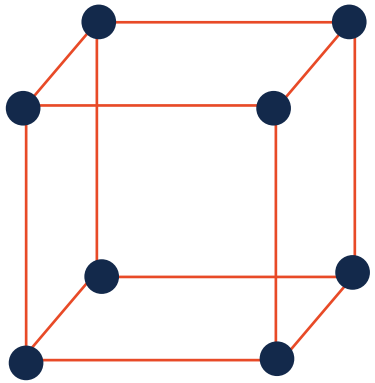


Operators:

Types of iterators:

Iterators encapsulated access to our data:





private var	++	*

CS 225 – Things To Be Doing

Exam 3 (Theory, C++) finishes today!

More Info: https://courses.engr.illinois.edu/cs225/fa2017/exams/

MP3: Available now!

Up to +7 extra for submission by Monday, Oct. 2!

Lab: lab_quacks start today!

Fun lab with one of my favorite debugging techniques!

POTD

Every Monday-Friday – *Worth +1 Extra Credit /problem (up to +40 total)*