



CS 225

Data Structures

February 11 – Lists and Iterators

G Carl Evans

Resize Strategy: +2 elements every time





Resize Strategy: +2 elements every time

Resize Strategy: x2 elements every time





Resize Strategy: x2 elements every time

Array Implementation

	Singly Linked List	Array
Insert/Remove at front		
Insert at given element		
Remove at given element		
Insert at arbitrary location		
Remove at arbitrary location		



Queue ADT

- [Order]:
- [Implementation]:
- [Runtime]:



Stack ADT

- [Order]:
- [Implementation]:
- [Runtime]:

Queue.h

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

What type of implementation is this Queue?

How is the data stored on this Queue?

Queue.h

```
1 #pragma once
2
3 template <typename T>
4 class Queue {
5     public:
6         void enqueue(T e);
7         T dequeue();
8         bool isEmpty();
9
10    private:
11        T *items_;
12        unsigned capacity_;
13        unsigned size_;
14 };
15
16
17
18
19
20
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 #pragma once
2
3 template <typename T>
4 class Queue {
5     public:
6         void enqueue(T e);
7         T dequeue();
8         bool isEmpty();
9
10    private:
11        T *items_;
12        unsigned capacity_;
13        unsigned size_;
14 };
15
16
17
18
19
20
21
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);`

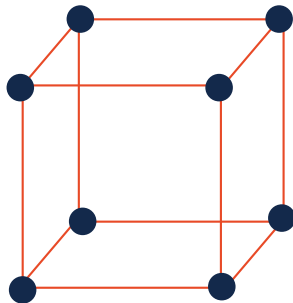
`q.dequeue();`

`q.enqueue(h);`

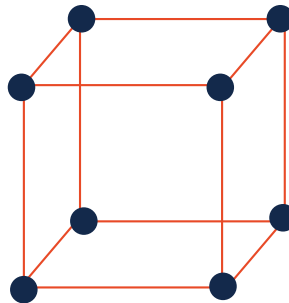
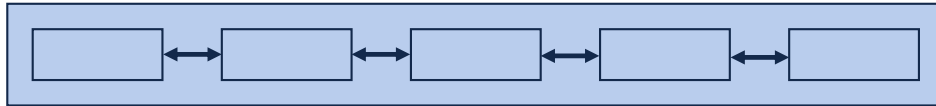
`q.enqueue(a);`

Iterators

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



Iterators encapsulated access to our data:



Cur. Location	Cur. Data	Next
<code>ListNode *</code>		
<code>index</code>		
<code>(x, y, z)</code>		



Iterators

Every class that implements an iterator has two pieces:

1. [Implementing Class]:



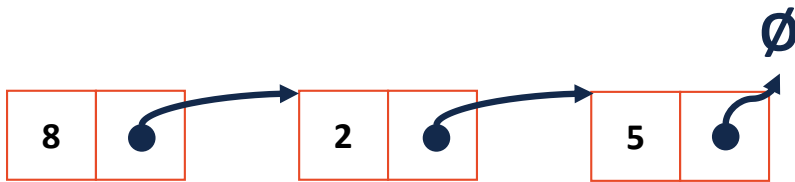
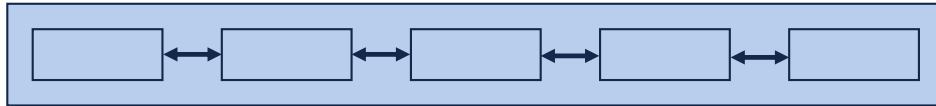
Iterators

Every class that implements an iterator has two pieces:

2. [Implementing Class' Iterator]:

- Must have the **base class: `std::iterator`**
- **`std::iterator`** requires us to minimally implement:

Iterators encapsulated access to our data:



::begin	::end

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) { /* nothing */ }
10 };
11
12 int main() {
13     Animal g("giraffe", "leaves", true), p("penguin", "fish", false), b("bear");
14     std::vector<Animal> zoo;
15
16     zoo.push_back(g);
17     zoo.push_back(p);    // std::vector's insertAtEnd
18     zoo.push_back(b);
19
20     for ( std::vector<Animal>::iterator it = zoo.begin(); it != zoo.end(); it++ ) {
21         std::cout << (*it).name << " " << (*it).food << std::endl;
22     }
23
24     return 0;
25 }
```

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) { /* nothing */ }
10 };
11
12 int main() {
13     Animal g("giraffe", "leaves", true), p("penguin", "fish", false), b("bear");
14     std::vector<Animal> zoo;
15
16     zoo.push_back(g);
17     zoo.push_back(p);    // std::vector's insertAtEnd
18     zoo.push_back(b);
19
20     for ( auto it = zoo.begin(); it != zoo.end(); it++ ) {
21         std::cout << (*it).name << " " << (*it).food << std::endl;
22     }
23
24     return 0;
25 }
```

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::vector<Animal> zoo;
15
16     zoo.push_back(g);
17     zoo.push_back(p);    // std::vector's insertAtEnd
18     zoo.push_back(b);
19
20     for ( const Animal & animal : zoo ) {
21         std::cout << animal.name << " " << animal.food << std::endl;
22     }
23
24     return 0;
25 }
```

For Each and Iterators

```
for ( const TYPE & variable : collection ) {  
    // ...  
}
```

```
14 std::vector<Animal> zoo;  
   ...  
20 for ( const Animal & animal : zoo ) {  
21     std::cout << animal.name << " " << animal.food << std::endl;  
22 }
```

For Each and Iterators

```
for ( const TYPE & variable : collection ) {  
    // ...  
}
```

```
14 std::vector<Animal> zoo;  
...  
20 for ( const Animal & animal : zoo ) {  
21     std::cout << animal.name << " " << animal.food << std::endl;  
22 }
```

```
... std::unordered_set<std::string, Animal> zoo;  
...  
20 for ( const Animal & animal : zoo ) {  
21     std::cout << animal.name << " " << animal.food << std::endl;  
22 }
```