Announcements

MP3 available, due 2/22, 11:59p. EC: 2/15, 11:59p.

MP 3.1 will be on Exam 1.

Exam 1: 2/19, 7-10p, in rooms tba. 75min exam, given 3hr.

Class cancelled 2/18.

Review session - 2/18, 12-2p, Siebel 1404.

MP2 solution party: Sat, 2/16, 10a, Siebel 0216.

Review session: Sun, 2/17, 5p, Siebel 0216.

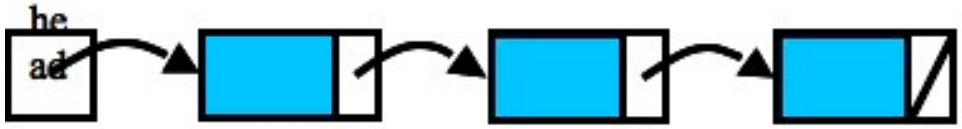
TODAY: Last little bit of C++

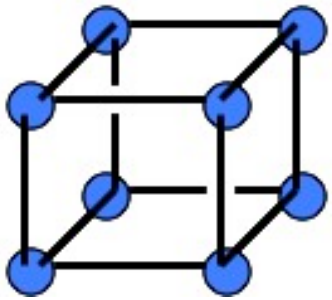
lots of magic:

```
#include <list>
#include <iostream>
#include <string>
using namespace std;

struct animal {
   string name;
   string food;
   bool big;
   animal(string n="blob", string f="you”, bool b=true):name(n),food(f),big(b) {}
};

int main() {

   animal g("giraffe","leaves"), p("penguin","fish”,false), b("bear");
   list<animal> zoo;

   zoo.push_back(g); zoo.push_back(p); zoo.push_back(b); //STL list insertAtEnd

   for(list<animal>::iterator it = zoo.begin(); it != zoo.end(); it++)
      cout << (*it).name << "  " << (*it).food << endl;

return 0;
}
```

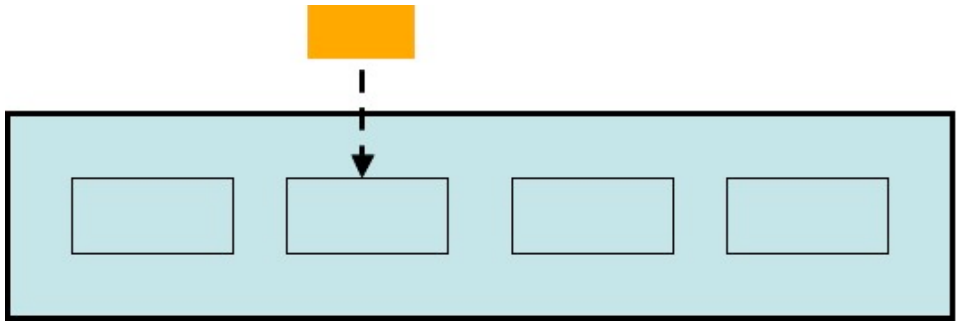
Suppose these familiar structures were encapsulated.

Iterators give us the access we need to traverse them anyway!





Iterators:



Objects of type “iterator” promise to have at least the following defined:

operator++

operator*

operator!=

operator==

operator=

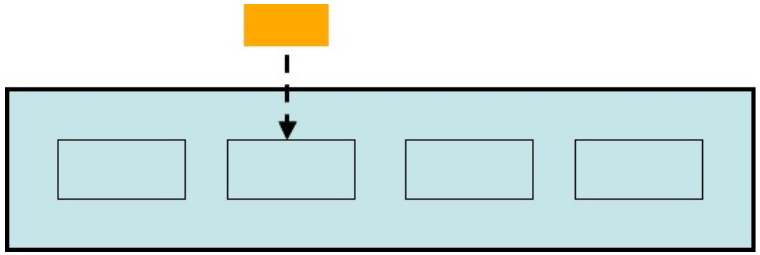
“Container classes” typically have a variety of iterators defined within:

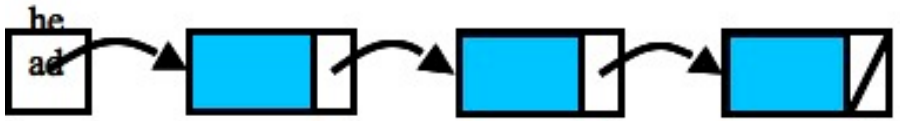
Forward

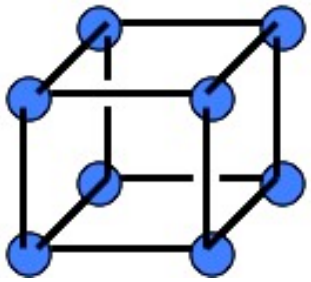
Reverse

Bidirectional

Iterators:







	pm	++	*
linked list			
array			
hypercube			

```
class human {
public:




private:



};
```

http://www.sgi.com/tech/stl/

Generic programming: (more magic)

```
#include <list>
#include <io
#include <st
using namesp

struct anima
   string na
   string fo
   bool big;
   animal(st                                                    }
};

int main() {

   animal g("giraffe","leaves"), p("penguin","fish",false), b("bear");
   list<animal> zoo;

   zoo.push_back(g); zoo.push_back(p); zoo.push_back(b); //STL list insertAtEnd

   for(list<animal>::iterator it = zoo.begin(); it != zoo.end(); it++)
      cout << (*it).name << "  " << (*it).food << endl;

return 0;
}
```

```
template<class Iter, class Formatter>
void print(Iter first, Iter second, Formatter printer) {
   while (!(first==second)) {
      printer(*first);
      first++;
   }
}
```

Generic programming: (more magic)

```
#include <list>
#include <io
#include <st
using namesp

struct anima
   string na
   string fo
   bool big;
   animal(st
};

int main() {

   animal g(
   list<anim

   zoo.push_

   for(list<
      cout <

return 0;
}
```

```
template<class Iter, class Formatter>
void print(Iter first, Iter second, Formatter printer) {
   while (!(first==second)) {
      printer(*first);
      first++;
   }
}
```

```
class printIfBig {
public:
   void operator()(animal a) {
      if (a.big) cout << a.name << endl;
   }
};
```

Generic programming: (more magic)

```
#include <list>
#include <io
#include <st
using namesp

struct anima
   string na
   string fo
   bool big;
   animal(st
};

int main() {

   animal g(
   list<anim

   zoo.push_

   for(list<
      cout <

return 0;
}
```

```
template<class Iter, class Formatter>
void print(Iter first, Iter second, Formatter printer) {
   while (!(first==second)) {
      printer(*first);
      first++;
   }
}
```

```
class printIfBig {
public:
   void operator()(animal a) {
      if (a.big) cout << a.name << endl;
   }
};
```

```
printIfBig myFun;
print<list<animal>::iterator,printIfBig>(zoo.begin(),zoo.end(),myFun);
```

Suppose these familiar structures were encapsulated.

Iterators give us the access we need to traverse them anyway!

And function objects give us the ability to change their data systematically.