Announcements

MP4 available, due 10/16, 11:59p. EC due 10/9, 11:59p.

```
#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
```

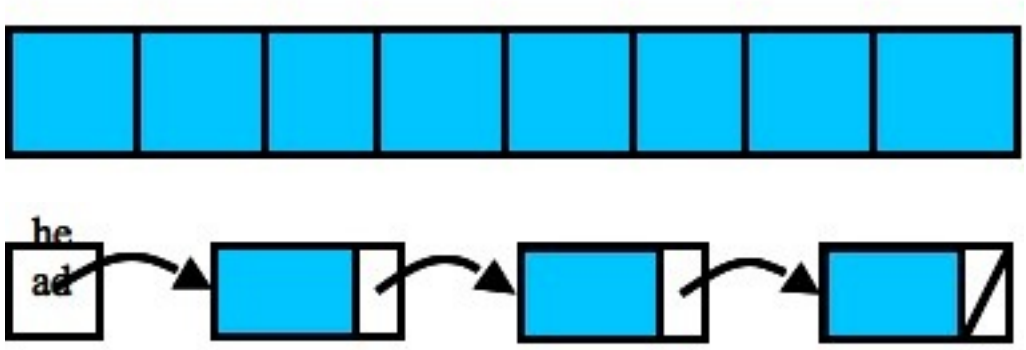
Our list:

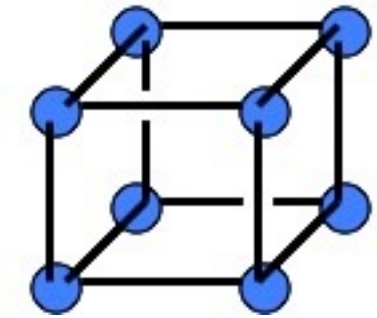
giraffe	leaves	TRUE
penguin	fish	FALSE
bear	you	TRUE

```
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 client the access it needs to traverse them anyway!





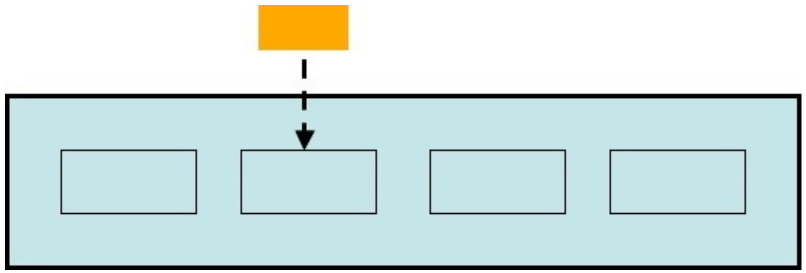
Objects of type “iterator” promise to have at least the following defined:

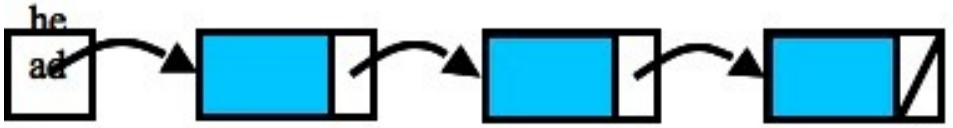
operator++
*operator**
operator!=
operator==
operator=

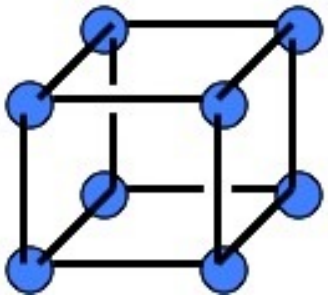
“Container classes” typically have a variety of iterators defined within:

Forward
Reverse
Bidirectional

Iterator class:







	pm	++	*
linked list			
array			
hypercube			

```
class apartmentBldg {
public:




private:



};
```

Where do these constructs live?

iterator class

begin()/end()

op++/op*

iter representation

std library documentation: http://www.sgi.com/tech/stl/

Generic programming: (more magic)

```
#include <list>
#include <ios
#include <sti
using namespa

struct animal
   string nam
   string foo
   bool big;
   animal(sti
};

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 <ios
#include <sti
using namespa

struct animal
   string nam
   string foo
   bool big;
   animal(sti
};

int main() {

   animal g('
   list<anima

   zoo.push_

   for(list<a
      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 <ios
#include <sti
using namespa

struct anima
   string nam
   string foo
   bool big;
   animal(sti
};

int main() {

   animal g('
   list<anima

   zoo.push_

   for(list<a
      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);
```