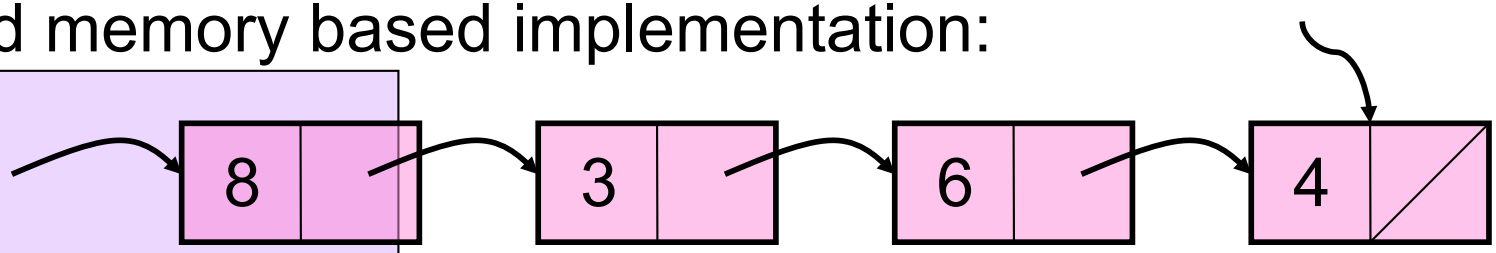
Announcements

MP3 available, due 2/24, 11:59p.

Exam3: 2/26-2/28

Queue—linked memory based implementation:

```
template<class SIT>
class Queue {
public:
    // ctors dtor
    bool empty() const;
    void enqueue(const SIT & e);
    SIT dequeue();
private:
    struct queueNode {
        SIT data;
        queueNode * next;
    };
    queueNode * entry;
    queueNode * exit;
    int size;
};
```



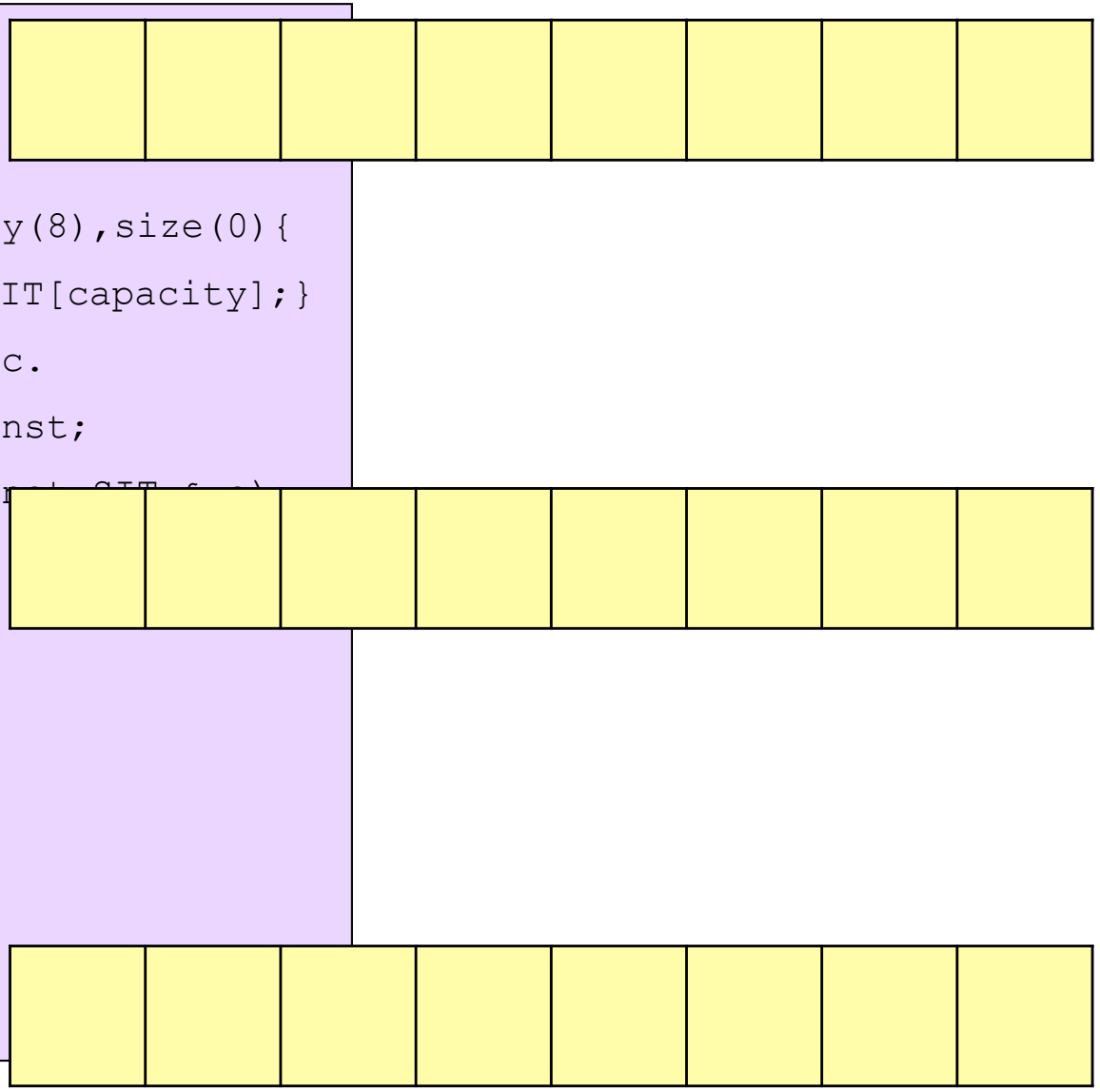
Which pointer is “entry” and which is “exit”?

What is running time of enqueue?

What is running time of dequeue?

Queue array based implementation:

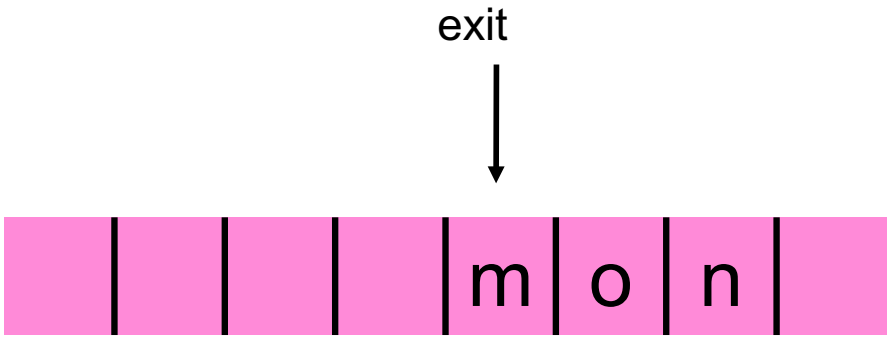
```
template<class SIT>
class Queue {
public:
    Queue():capacity(8),size(0){
        items=new SIT[capacity];}
    ~Queue(); // etc.
    bool empty() const;
    void enqueue(const SIT & e);
    SIT dequeue();
private:
    int capacity;
    int size;
    SIT * items;


};
```



enqueue(3);
enqueue(8);
enqueue(4);
dequeue();
enqueue(7);
dequeue();
dequeue();
enqueue(2);
enqueue(1);
enqueue(3);
enqueue(5);
dequeue();
enqueue(9);

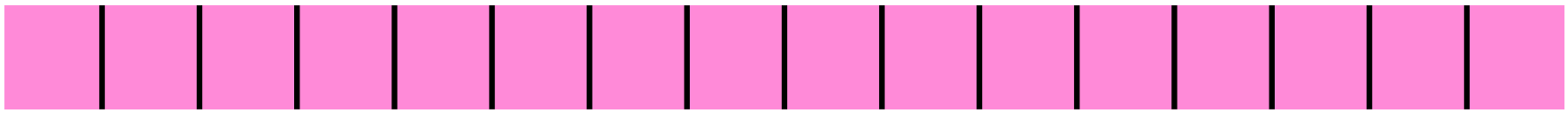
Queue array based implementation:

```
template<class SIT>
class Queue {
public:
    Queue(); ~Queue(); // etc.
    bool empty() const;
    void enqueue(const SIT & e);
    SIT dequeue();
private:
    int capacity;
    int size;
    SIT * items;
    int entry;
    int exit;
};
```



…

enqueue(m);	enqueue(y);
enqueue(o);	enqueue(i);
enqueue(n);	enqueue(s);
…	dequeue();
enqueue(d);	enqueue(h);
enqueue(a);	enqueue(a);

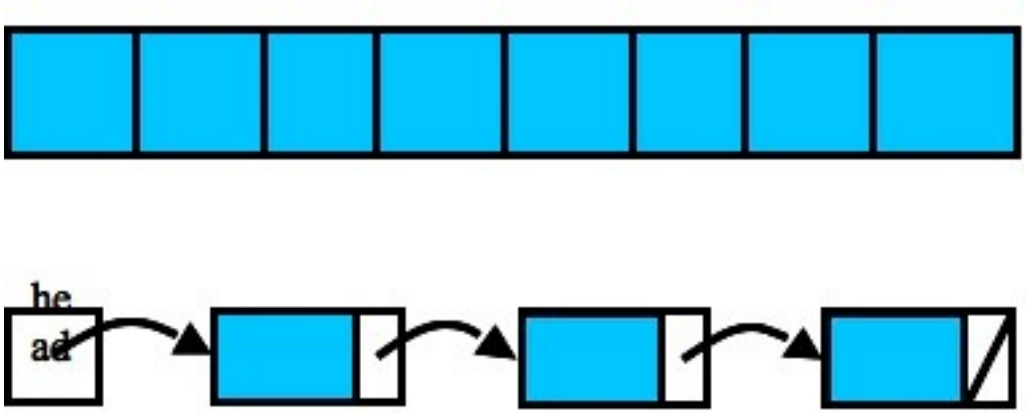


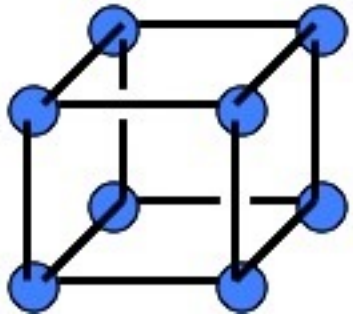
C++: A bit of Magic…

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





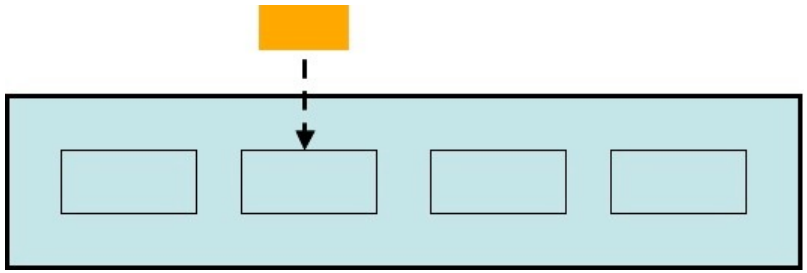
Objects of type “iterator” promise to have at least the following defined:

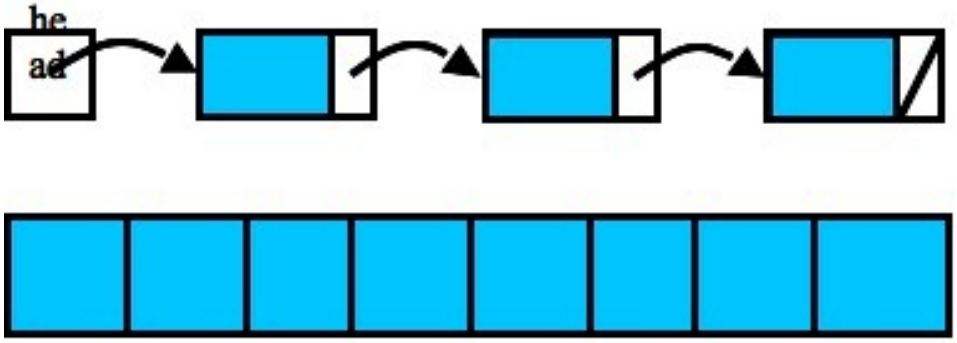
operator++
*operator**
operator!=
operator==
operator=

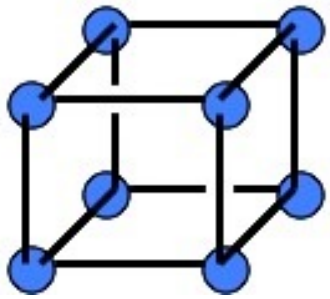
“Container classes” typically have a variety of iterators defined within:

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

struct animal {
   string name;
   string fo
   bool big;
   animal(sti
};

int main() {

   animal g("
   list<anima

   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 << " is " << (*it).food << endl;

return 0;
}
```

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

1. Declare an object of type `animal`:
2. Declare an object of type `printIfBig`:
3. Using your answers for 1 and 2, invoke a member function of the `printIfBig` class: