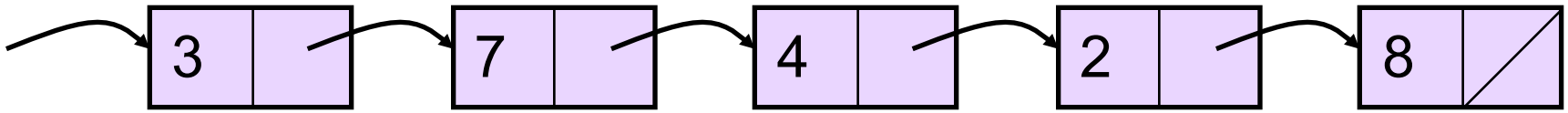
Announcements

```
struct listNode {
   LIT data;
   listNode * next;
   listNode(LIT ...
};
```

MP3 available, due 10/2, 11:59p. EC due 9/25, 11:59p.

Exam 1: 9/30, 7-10p in rooms TBA

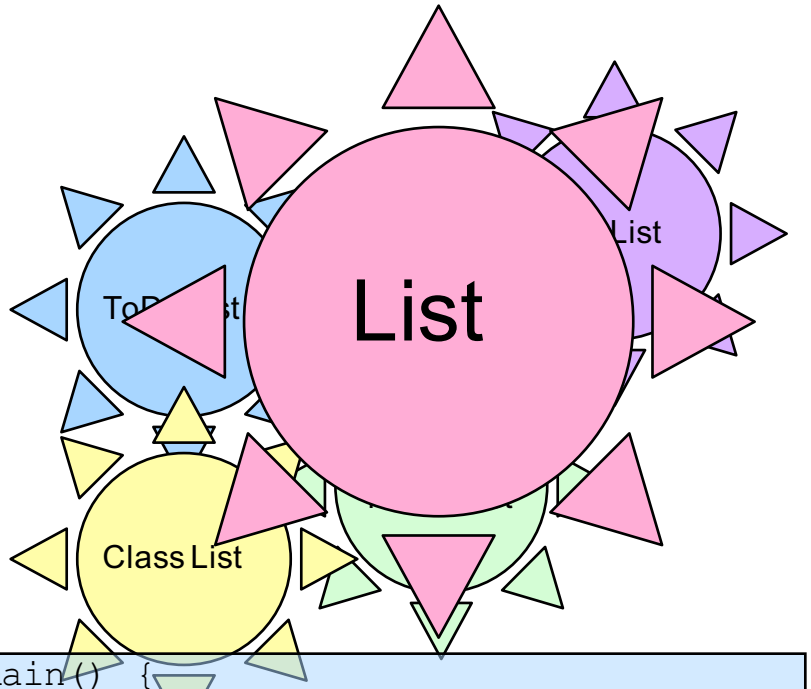



```
//returns pointer to node k steps forward from *curr
listNode * findKth(listNode *   curr, int k) {
   if (curr == NULL || k == 0) return curr
   else
      return findKth(curr->next, k-1);
}
```

Analysis:

Find kth in array:

Abstract Data Types (an example):




```
template<class LIT>
class List {
public:
    List();
    //~List();
    int getSize() const;
    void insert(int loc, LIT e);
    void remove(int loc);
    LIT const & getItem(int loc) const;
private:
    //my little secret
};
```

```
int main() {
    List<int> myList;
    myList.insert(1,4);
    myList.insert(1,6);
    myList.insert(1,8);
    myList.insert(3,0);
    myList.insert(4,myList.getItem(2));
    cout << myList.getSize() << endl;
    myList.remove(2);
    cout << myList.getItem(3) << endl;
    return 0;
}
```

ADT List, implementation 1:

```
template<class LIT>
class List {
public:
    List():size(0){}
    //~List();
    int getSize() const;
    void insert(int loc, LIT e);
    void remove(int loc);
    LIT const & getItem(int loc) const;
private:
    LIT items[8];
    int size;
};
```

0	1	2	3	4	5	6	7

```
template<class LIT>
int List<LIT>::getSize() const {
   return size;
}
template<class LIT>
void List<LIT>::insert(int loc, LIT e){
if ((size + 1) < 8) {
   LIT go = e;
   int it = loc-1;
   while (it < size+1){
     LIT temp = items[it];
     items[it] = go;
     go = temp;
     it ++;
   }
   size ++;
}}
template<class LIT>
void List<LIT>::remove(int loc) {
if (size > 0) {
   int it = loc-1;
   while (it < size){
     items[it] = items[it+1];
     it ++;
   }
   size --;
}}
template<class LIT>
LIT const & List<LIT>::getItem(int loc)
const {return items[loc -1];}
```

Don't look at this code!

Don't look at this code!

Implementing a list using an array:

0	1	2	3	4	5	6	7

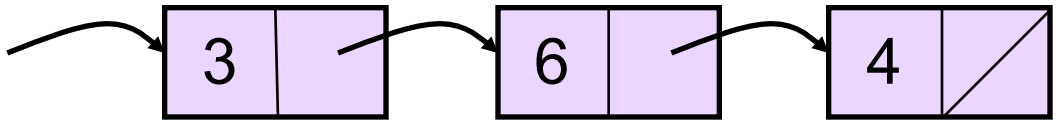
0	1	2	3	4	5	6	7

ADT List, implementation 2:

```
template<class LIT>
class List {
public:
    List():size(0),head(NULL){}
    ~List(); // also copy constructor, assignment op
    int getSize() const;
    void insert(int loc, LIT e);
    void remove(int loc);
    LIT const & getItem(int loc) const;
private:
    listNode * head;
    int size;
    listNode * Find(listNode * place, int k);
    struct listNode {
       LIT data;
       listNode * next;
       listNode(LIT newData)
    }
}
```

```
template<class LIT>
listNode * List<LIT>::Find(listNode * place, int k){
if ((k==0) || (place==NULL))
   return place;
else
   return Find(k-1, place->next);
}
```

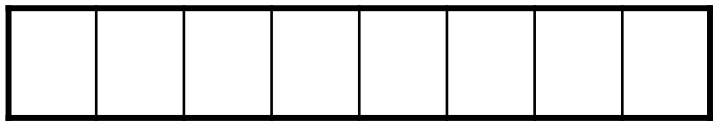
Insert new node in kth position:



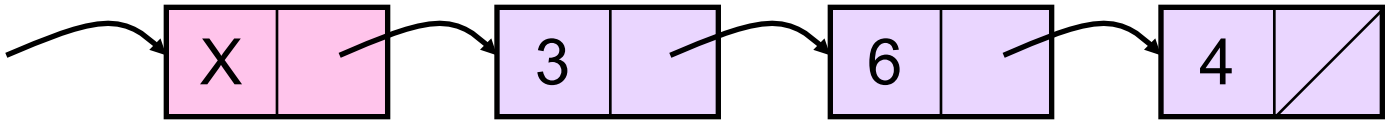

```
void List<LIT>::insert(int loc, LIT e) {


}
```

Analysis:

insert new kth in array:



Insert new node in kth position with sentinel:

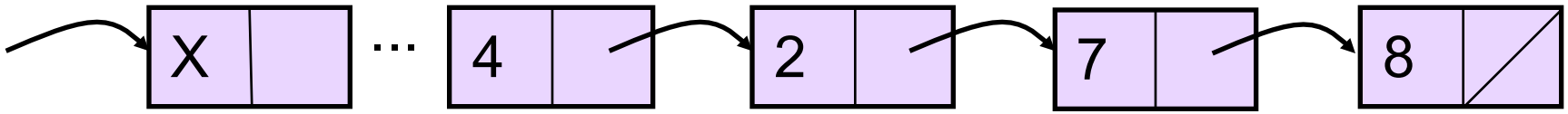



```
void List<LIT>::insert(int loc, LIT e) {
    listNode * curr = Find(head, loc-1);
    listNode * newN = new listNode(e);
    newN->next = curr ->next;
    curr->next = newN;
}
```

Wow, this is convenient! How do we make it happen?

```
template<class LIT>
List<LIT>::List(){



}
```

Remove node in fixed position (given a pointer to node you wish to remove):



Solution #1:

```
void List<LIT>::removeCurrent(listNode * curr) {

}
```