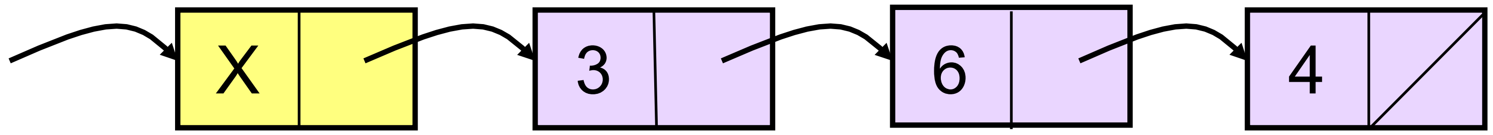
Announcements

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

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

Insert new node in kth position with sentinel:

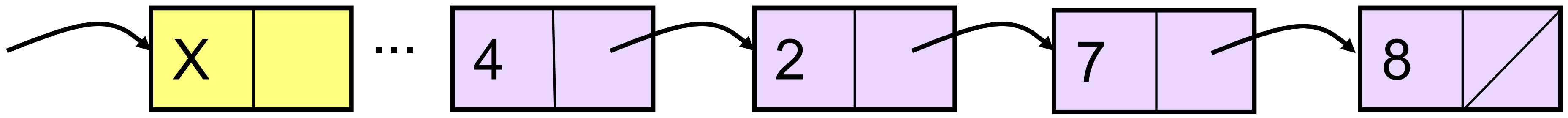


```
void List<LIT>::insert(int loc, LIT e) {
     listNode * curr = Find(head, loc-1);
     listNode * newN = new listNode(e);
     newN->next = curr ->next;
     curr->next = newN;
}
```

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

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

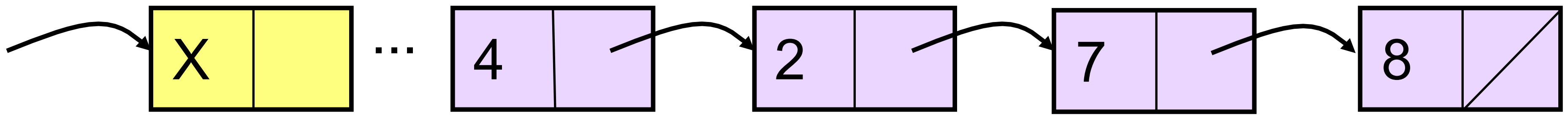
Remove node in fixed position (given a pointer to node you wish to remove):



Solution #1:

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



}
```

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



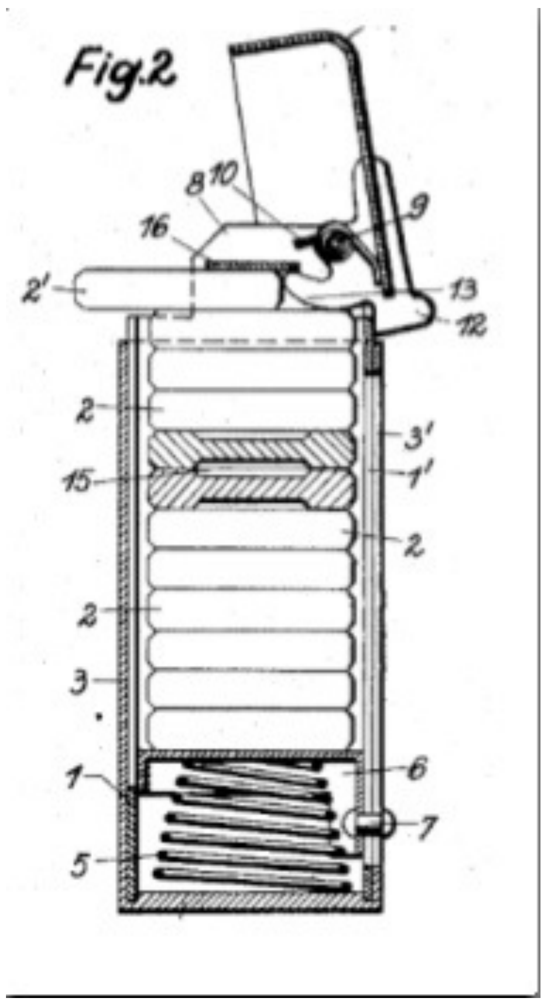
Constant time hack:

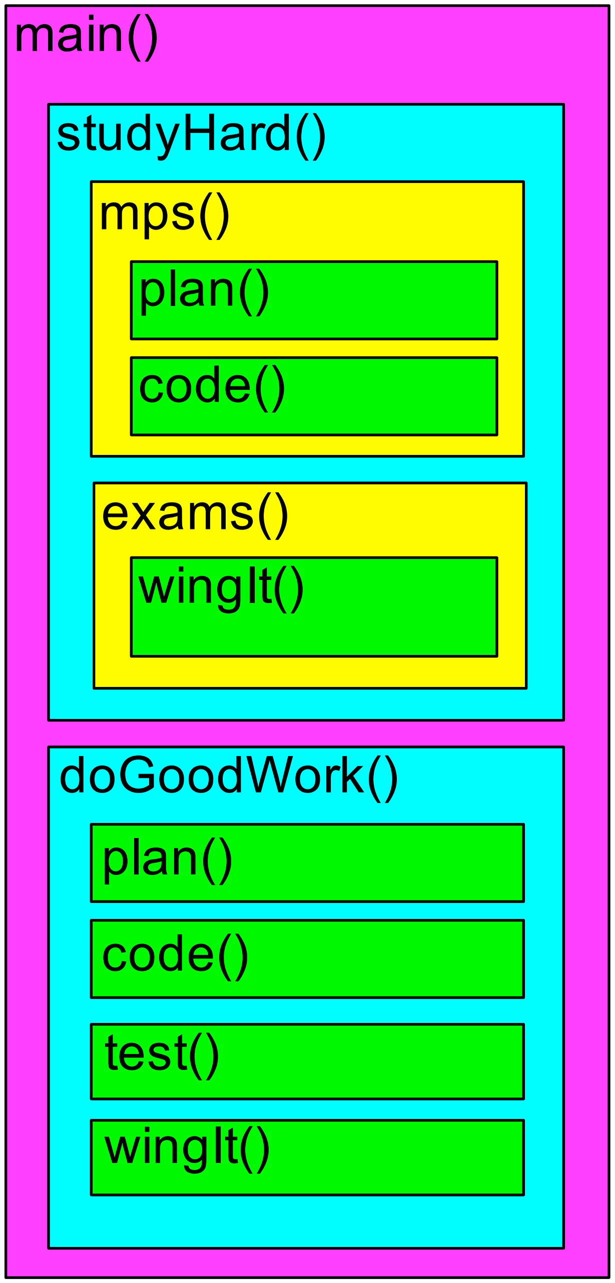
```
void List<LIT>::removeCurrent(listNode * curr) {







}
```

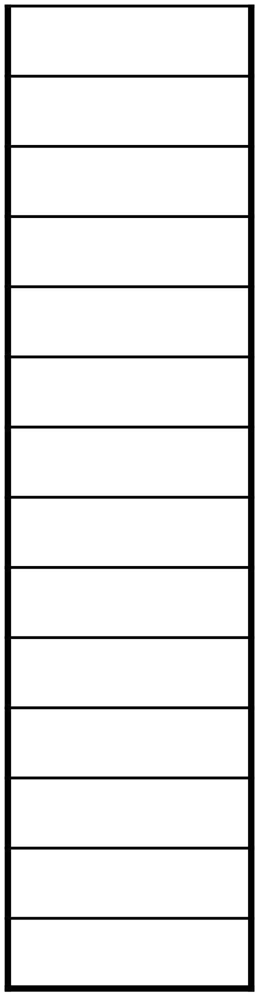
Summary – running times for List functions:

	SLL	Array
Insert/Remove at front:	O(1)	O(1)
Insert at given location:	O(1)	O(1)
Remove at given location:	O(1) hack	O(n) shift
Insert at arbitrary location:	O(1)	O(n) shift
Remove at arbitrary location:	O(n) find	O(n) shift

Stacks:







({ } ( ) [(( )) { (( )) ( ) }] ( ))

4 5 + 7 2 - * 3 - 6 /

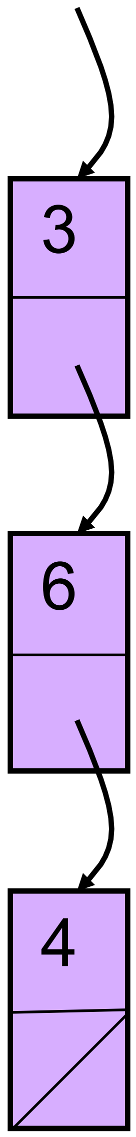
Stack ADT:

```
template<class SIT>
class Stack {
public:
    Stack();
    ~Stack(); // also copy
    constructor, assignment op
    bool empty() const;
    void push(const SIT & e);
    SIT pop();
private:

       ?

};
```

push(3)
push(8)
push(4)
pop()
pop()
push(6)
pop()
push(2)
pop()
pop()

Stack linked memory implementation:

```
template<class SIT>
class Stack {
public:
    Stack();
    ~Stack(); // etc.
    bool empty() const;
    void push(const SIT & e);
    SIT pop();
private:
    struct stackNode {
        SIT data;
        stackNode * next;
    };
    stackNode * top;
    int size;
};
```



```
template<class SIT>
SIT Stack<SIT>::pop(){







}
```

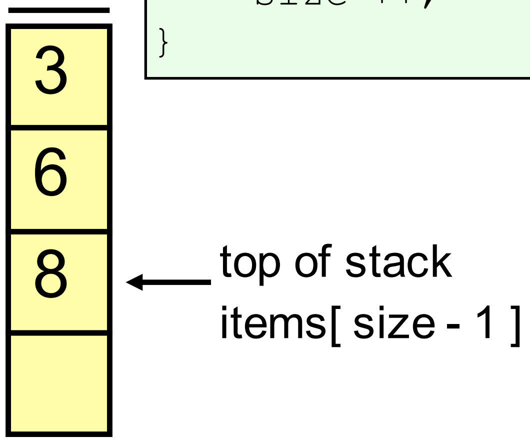
```
template<class SIT>
void Stack<SIT>::push(const SIT & d){
stackNode * newNode = new stackNode(d);
newNode->next = top;
top = newNode;
}
```

Stack array based implementation:

```
template<class SIT>
class Stack {
public:
    Stack();
    ~Stack(); // etc.
    bool empty() const;
    void push(const SIT & e);
    SIT pop();
private:
    int capacity;
    int size;
    SIT * items;
};
```

```
template<class SIT>
Stack<SIT>::Stack(){
    capacity = 4;
    size = 0;
    items = new SIT[capacity];
}
```

```
template<class SIT>
void Stack<SIT>::push(const SIT & e){
    if (size >= capacity) {
      // grow array somehow
    }
    items[size] = e;
    size ++;
}
```


3
6
8
top of stack
items[size - 1]