

Announcements

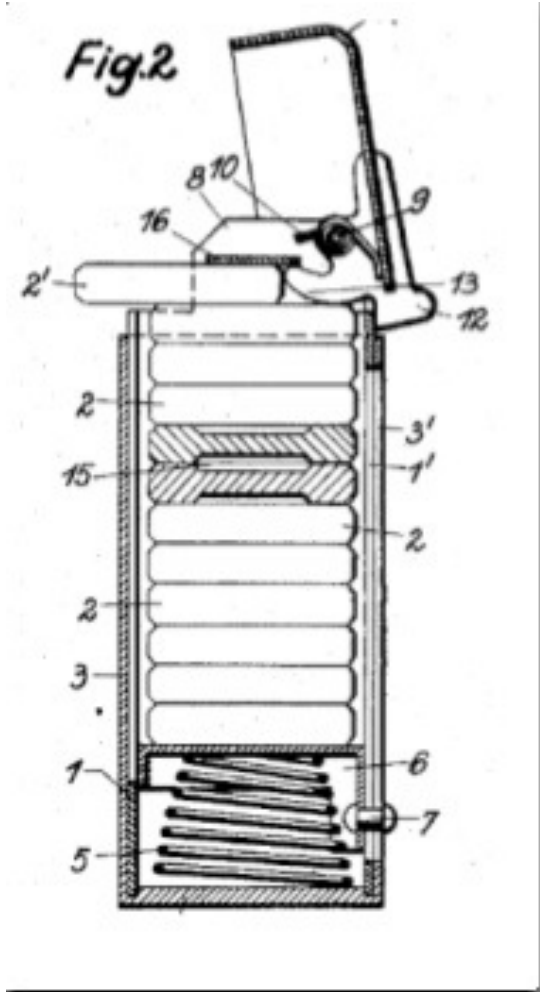
MP3 available, due 9/30, 11:59p.

Exam2: 9/25-9/28

Summary – running times for List functions:

	<u>SLL</u>	<u>Array</u>
Insert/Remove at front:	$O(1)$	$O(1)$
Insert at <i>given</i> location:	$O(1)$	$O(1)$
Remove at <i>given</i> location:	$O(1)$ hack	$O(n)$ shift
Insert at <i>arbitrary</i> location:	$O(1)$	$O(n)$ shift
Remove at <i>arbitrary</i> location:	$O(n)$ find	$O(n)$ shift

A yellow Bart Simpson head is attached to a red pushpin. The pushpin is inserted into a red base that has the words "PUSH HERE" embossed on it.



winglt()

[illegible]

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()

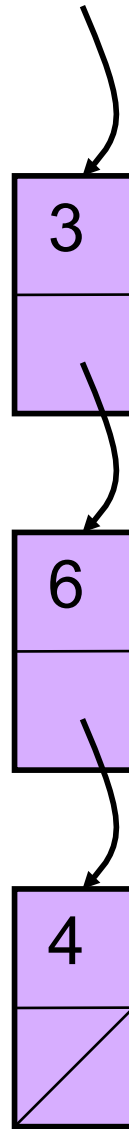
push(2)

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() {
```

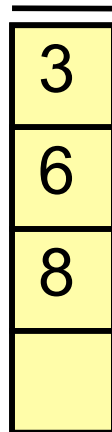
```
template<class SIT>
void Stack<SIT>::push(const SIT & d) {
    stackNode * newNode = new stackNode(d);
    newNode->next = top;
    top = newNode;
}
```

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];
}
```

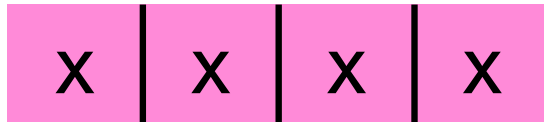
```
template<class SIT>
void Stack<SIT>::push(const SIT & e) {
    if (size >= capacity) {
        // grow array somehow
    }
    items[size] = e;
    size ++;
}
```



top of stack
items[size - 1]

Stack array based implementation: (what if array fills?)

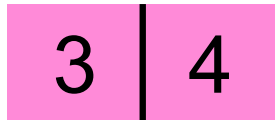
Analysis holds for array based implementations of Lists, Stacks, Queues, Heaps...



General Idea: upon an insert (push), if the array is full, create a larger space and copy the data into it.

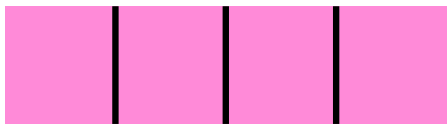
Main question: What's the resizing scheme? We examine 2.

Stack array based implementation: (what if array fills?)



How does this scheme do on a sequence of n pushes?

Stack array based implementation: (what if array fills?)



How does this scheme do on a sequence of n pushes?