CS180 - Linked Lists

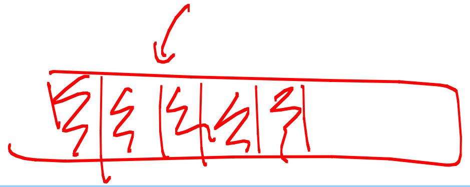
Note Title 9/16/2010

Announcements

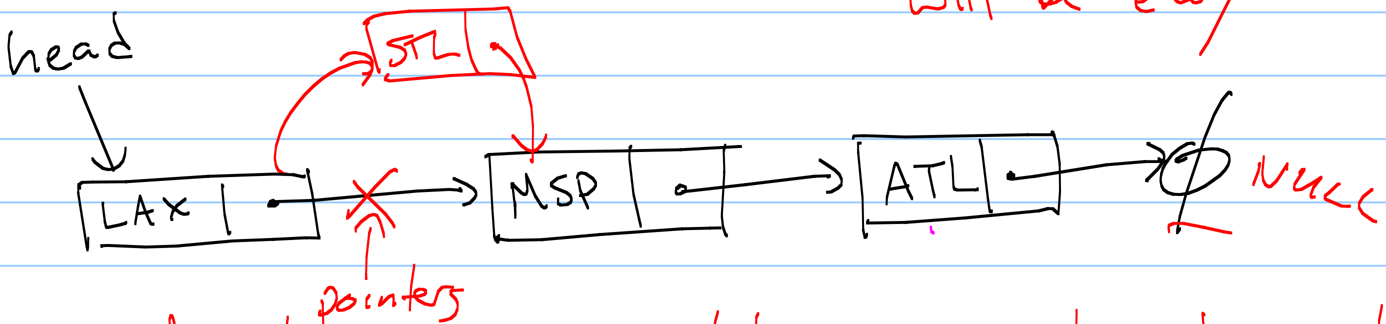
- Midterm will be in 1 week
- HW due Monday
- Thurs - in class review session
- Practice midterm - last problem was too hard
- Program 2 will be posted today or Monday due in ~2 weeks (stacks)

Linked List

Abstract picture

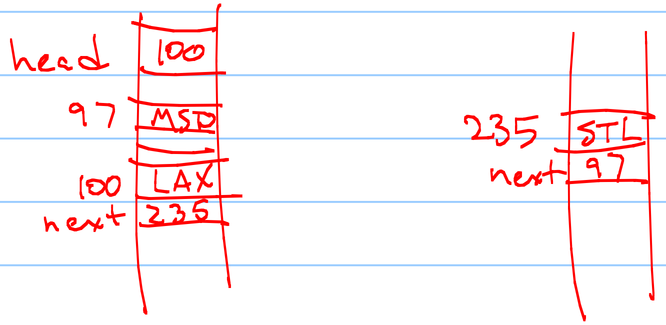


inserting + deleting will be easy



- each element has data + a pointer to next element

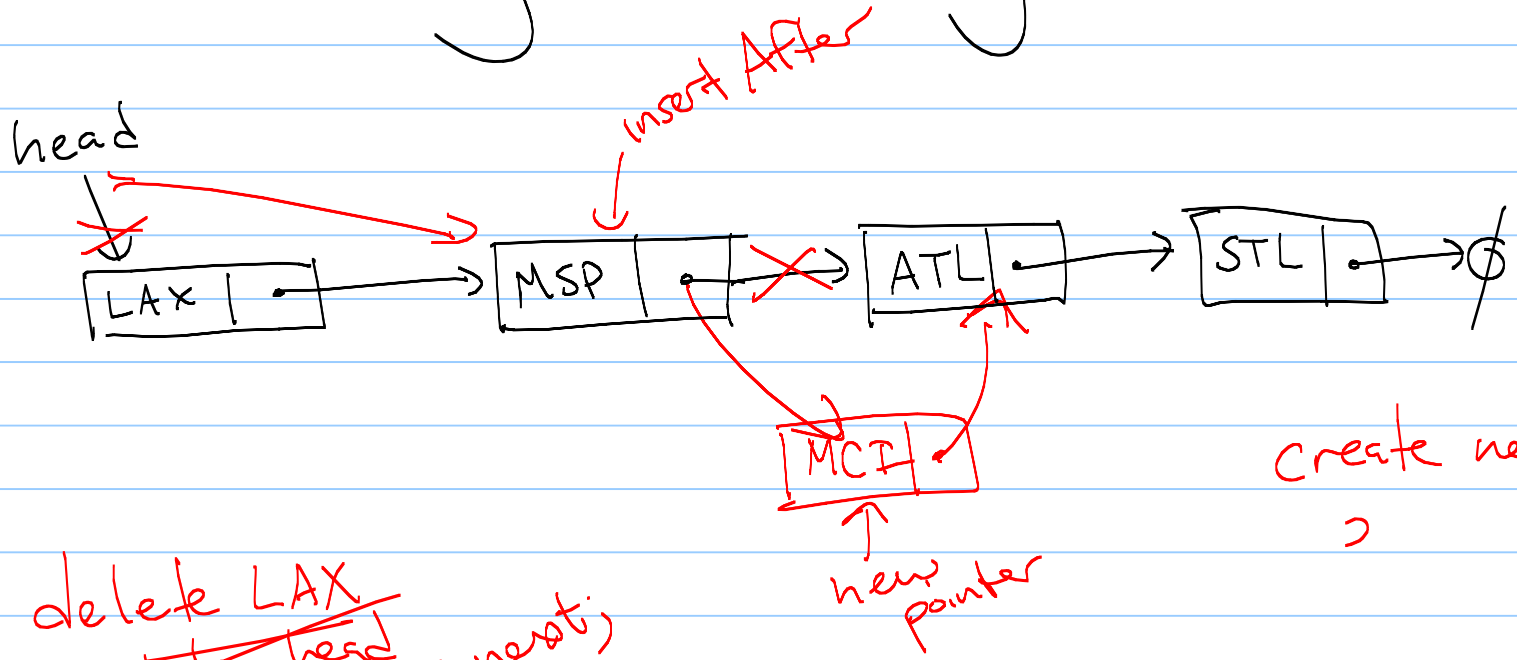
Reality:



Linked list ADT

- called a singly linked list
- ★ always need a pointer to the head of the list
- last entry points to a null pointer

Inserting + Deleting



create new element

delete LAX

~~delete head~~

temp = head;

head = head -> next;

delete temp

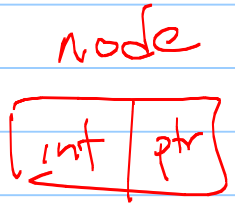
constant time O(1)

How to implement?

Each ~~element~~ node needs:

① Object (string, int, ...)

② Pointer to next ~~element~~ node



could use a class

The node structure

```
template <typename Object>

struct Node {
    Object element;          // value of this node
    Node* next;              // ptr to next node

    // constructor
    Node (const Object& e = Object(), Node* n=NULL) :
          element (e), next(n) {}

};
```

NodePtr

(don't really need private data)

Typedefs –

short cuts for things you will use a lot

at top of file

```
typedef Node* NodePtr ;
```

Now we can use NodePtr in our declarations

Why a struct & not a class?

- not really private data
  or functions

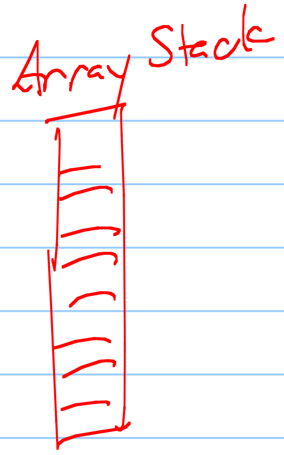
→ LIFO

Linked Stack

A version of a stack which uses an underlying linked list.

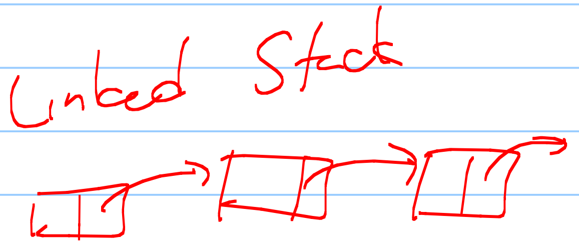
Advantage:

- no max capacity
- uses less memory (b/c if few elements, array would be empty)



Disadvantage:

- uses more memory (b/c we also have pointers)



Code:

Our node struct will be included as "protected" (instead of public/private).
Why?

want main to not have access, but want it to be inheritable

Private data:

head → ∅

```
Node* tp;
int   sz;
```

Functions (Easy Ones)

Constructor:

```
LinkedStack(): tp(NULL), sz(0) {}
```

size:

```
int size() const {return sz;}
```

empty

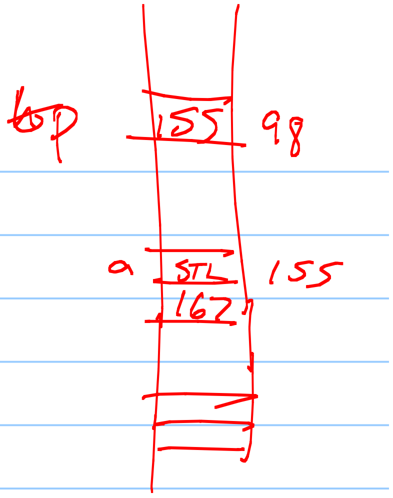
```
bool isEmpty() Const {return sz==0;}
                            {return tp==NULL;}
```

Top: (constant ref. version)

```
const Object& top () const {
    if isEmpty()
        throw error
    return tp->element;  ✓
}
```

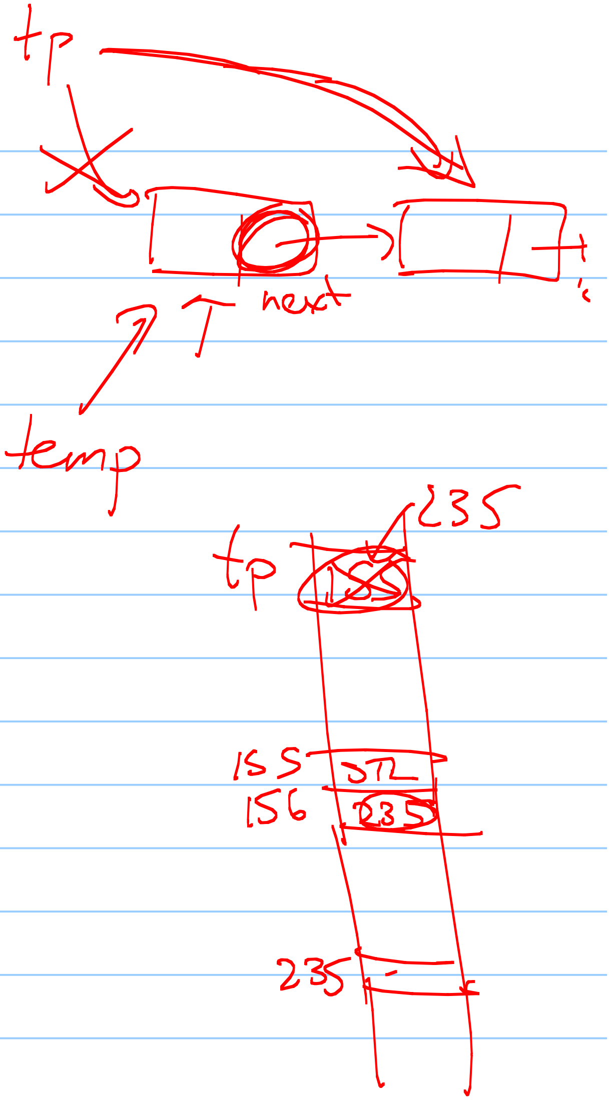
(*tp).element ✓

~~(*tp)~~ → 98



Pop:

```
void pop() {
  → if (isEmpty())
        throw error;
    Node* temp = tp;
    tp = tp->next;
    delete temp;
    sz--;
}
```


tp
next
temp
235
tp
155
155
372
156
235
235