CS 180 - Linked Lists

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

- Each element has data + a pointer to next element

Reality:

```
head 100 97 MSP 235 STL
100 LAX 235
next 97
```
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

head

LAX

MSP

ATL

STL

insert After

MCT

new pointer

Create new element

key = head->next
head = head->next
delete temp

constant time O(1)
How to implement?

Each element needs:

1. Object (string, int, ...)
2. Pointer to next element node

Could use a class
The node structure

```cpp
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) {}
};
```

(don't really need private data)
Type defs -

short cuts for things you will use a lot

at top of file

type def Node* NodePtr;

Now we can use NodePtr in our declarations
Why a struct & not a class?

- not really private data
  or functions
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? I want main to not have access, but want it to be inheritable.

Private data:

Node*
int Sz

head 6
Functions (Easy Ones)

Constructor:

```
tr::LinkStack() : 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

Top: 98
Pop:

void pop() {
    if (is_empty())
        throw error;

    Node* temp = tp;
    tp = tp->next;
    delete temp;
}

ST -->