A Stack Class Interface

```
class IntStack {
public:
  IntStack();          // default constructor
  // should have a copy constructor, too

  bool isEmpty();      // true if no items on stack
  void push(int item); // add item to top

  int pop();           // remove and return top item
  int top();           // just return the top item
private:
  . . .
};
```


Possible Implementations

- Many possible implementations
  - Array-based
  - Linked list
  - Or even, using already implemented Vector ADT
- As implementer, use other ADTs to make job easier
  - Don't reinvent the wheel for every problem
  - Often simplifies job to reuse pieces when possible
- We'll use stack of ints as an example
  - could have stack of any type of data item

Stack Via Vector (3)

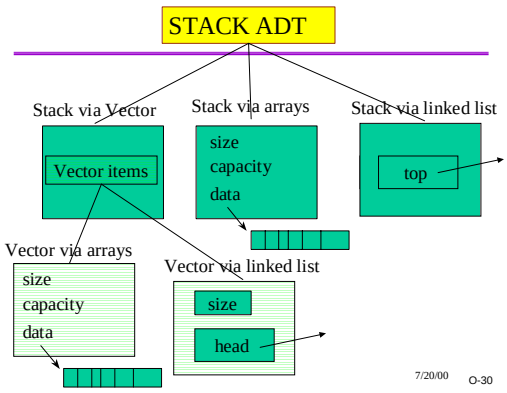
```
void IntStack::push(int item) {
  items.insert(0, item);
}

int IntStack::top() const {
  return items.retrieve(0);
}

int IntStack::pop() {
  return items.remove(0);
}
```


Stack Via Linked List

- Another implementation technique
- Main idea: keep a linked list, with private "top" pointer to the front of the list
- Add new data as a new link to the beginning of the linked list
- Pop/top: remove/return the beginning of the linked list
- Not the only way -- could have decided to make top be the end of the list
  - Important thing is to choose a way;document it; and stick with it.



Discussion

- Why learn three different ways to implement the same ADT?
- What are the pro's and con's of each way?
  - Programming effort?
  - Speed (efficiency) of execution?
  - Suitability to application?
  - Other factors?