Stack ADT

LIFO - Last in, First out.

• Three Operations:
  
  • **PUSH(value)** - add a new element to the "top" of the stack, with the given value.
  
  • **value = POP** - gives back the value of the most recently pushed element that has not yet been popped, and removes this element from the stack.
  
  • **value = TOP** - returns the value of the most recently pushed element that has not already been popped.
  
• No constraints on the type of the value.
A Stack of Integers

- PUSH adds a new block to the top of the pile.
- POP tells you the value of the top element, and removes this element.

![Stack Diagram]

- Note - the value of POP is undefined if the stack is empty. This is generally considered to be an error.
- Generally, this is due to a programming error, but not always. Every stack should define its behavior in this case ...
A Stack as a List

Given a Stack S, and a List L:

• **PUSH (S, v) <=>**
  \[\text{INSERT (v, 0, L)}\]

• **v = POP (S) <=>**
  \[v = \text{FINDKTH (0); DELETE (v, L)}\]

• **v = TOP (S) <=>**
  \[v = \text{FINDKTH (0)}\]

• It might also be handy to use ISEMPY(L).

Top can also be implemented by the other Stack ADT operations themselves:

\[v = \text{POP (S); PUSH (S, v)}\]
Implementing Stacks

- If we use a linked-list representation of list S, then all of these operations take \( O(1) \) time:
  - \textbf{PUSH} = \textbf{INSERT} (at start of List) = \( O(1) \)
  - \textbf{TOP} = \textbf{FINDKTH}(0) = \( O(1) \)
  - \textbf{POP} = \textbf{DELETE} (at start of List) = \( O(1) \)
- For arrays, however, \textbf{INSERT} at the start of the list is \( O(n) \). (in fact, it’s \( \Theta(n) \)). It’s going to be slow.
- But stacks \textbf{can be implemented efficiently with arrays}!
  - See Weiss, pages 68-70, for an example.

- Observations
  - We can append a new element to the end of an array very quickly - just put it there.
  - Finding the value at the end of the array is also easy, if we keep a pointer to it (or its index).
  - The List ADT doesn’t let us get at these features, because it is \textbf{Abstract}.

Given any list implementation where \textbf{INSERT} and \textbf{DELETE} at the end are \( O(1) \), we can implement a stack ADT in such a way that the \textbf{PUSH} and \textbf{POP}, and \textbf{TOP} operations are all \( O(1) \).
A Curiosity: List Reversal

Let L be a list. For the sake of illustration, let it contain the elements 1, 2, 3. To reverse the list, using a stack S, we can use the following algorithm:

1. Initialize S to be an empty stack.

2. Starting at the beginning of L, push each element in L onto stack S and then remove it from L.
   - push(1) 2-3
   - push(2) 3
   - push(3)

3. Pop the elements off of the stack S until S is empty, pasting each value onto the end of a list.
   - 3 = pop() 3
   - 2 = pop () 3-2
   - 1 = pop () 3-2-1
Rotation

• Sometimes you may want to rotate the stack, or part of the stack:

• ROTATION is shifting the elements of the list toward the front of the list (or toward the end) by one, with the element at the front being appended to the end (or vice versa).

  • We can perform this operation several times in succession

```
  top ------- 5
             4
             3
             2
             1

  Rotate (stack, 2)
```

```
  top ------- 3
             2
             1
             5
             4
```

```
  top ------- 5
             4
             3
             2
             1

  Rotate (top 3, 1)
```

```
  top ------- 4
             3
             5
             2
             1
```

• We’ll just look at rotating the whole stack for now.
Rotation of Arrays

• Let’s rotate this 10 element list by 5...

So it looks like this...

1. Reverse (Flip) the Left

2. Reverse (Flip) the Right

3. Reverse (Flip) All

• Steps 1 and 2 together are \( \Theta(N) \). Step 3 is also \( \Theta(N) \). Rotation is \( \Theta(N) \), where \( N \) is the number of elements rotated, not how far.
Rotation of Linked Lists

- It’s just a matter of manipulating the correct pointers...

- To rotate this list by n:

  new_tail = /* nth element of list */
  tail = /* last element of list */
  tail->next = List_p;
  List_p = new_tail->next;
  new_tail->next = NULL;

- Example: Rotate by 3: