Data Structures and Algorithms
CS-206

STACKS

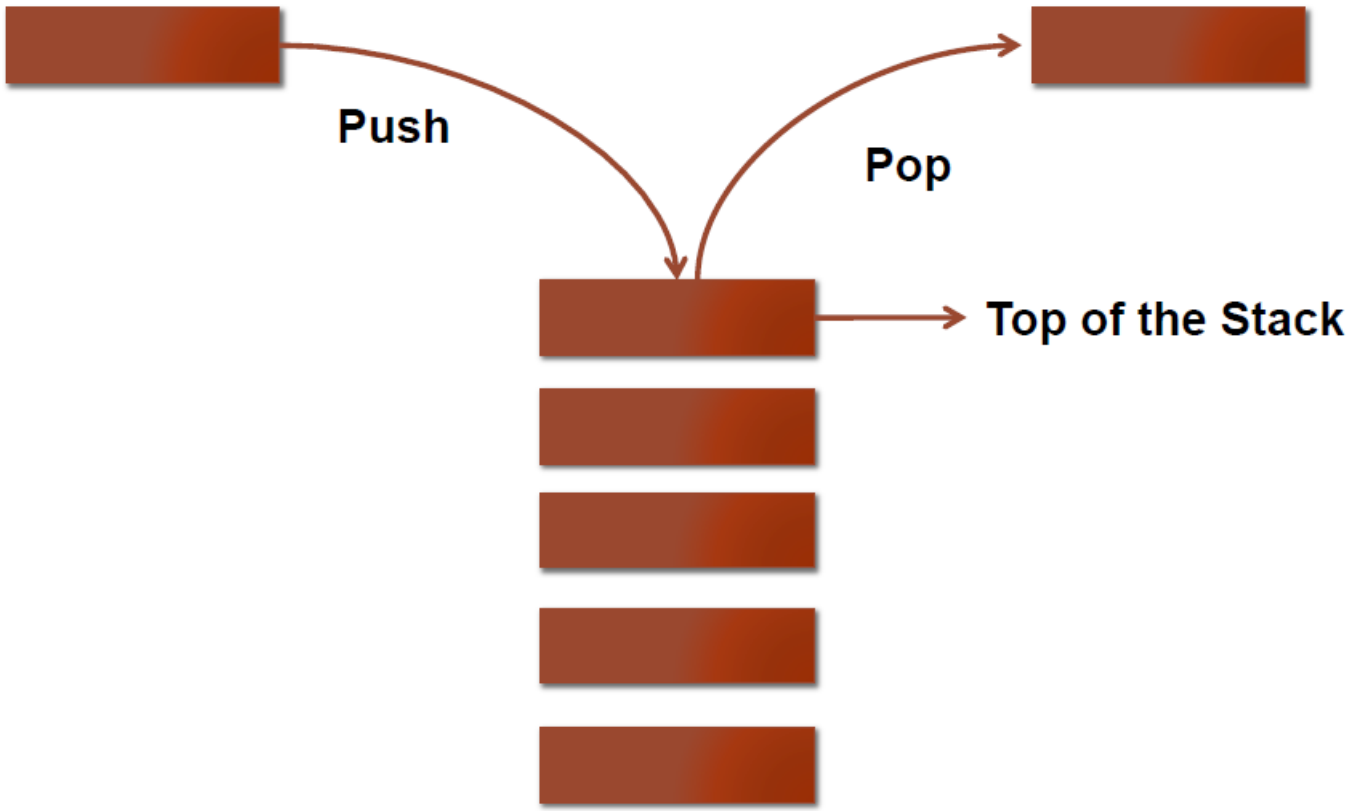
Instructor
Dr. Maria Anjum
Assistant Professor

Department of Computer Science
Lahore College for Women University

Stack

- Stack is a data structure that allows access to items in a last in first out (LIFO) style
- Main Stack operation:
  - push(object): insert an element to the stack
  - pop(): return the last inserted element and remove it
- Auxiliary stack operations:
  - top() / peek(): return the element on top of the stack (last inserted element)
  - size(): return the number of elements stored
  - isEmpty(): return a boolean value indicating elements are store or not in the stack

Stack Structure



Stack Function

Operation	output	stack
• push(8)	-	(8)
• push(3)	-	(3, 8)
• pop()	3	(8)
• push(2)	-	(2, 8)
• push(5)	-	(5, 2, 8)
• top()	5	(5, 2, 8)
• pop()	5	(2, 8)
• pop()	2	(8)
• pop()	8	()
• pop()	"error"	()
• push(9)	-	(9)
• push(1)	-	(1, 9)

Application of Stack

- Reversing data
- Page-visited history in a Web browser
- Undo sequence in a text editor
- Implementing recursion

Many other you may need to explore!

Typing and Correcting Chars

- What data structure would you use for this problem?
  - User types characters on the command line
  - Until she hits enter, the backspace key (<) can be used to "erase the previous character"

Sample

Action	Result
type h	h
type e	he
type l	hel
type o	helo
type <	hel
type l	hell
type w	hellw
type <	hell
type <	hel
type <	he
type <	h
type i	hi

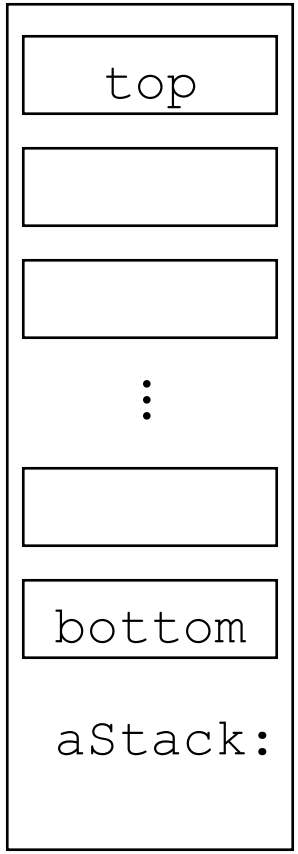
Analysis

- We need to store a sequence of characters
- The order of the characters in the sequence is significant
- Characters are added at the end of the sequence
- We only can remove the most recently entered character

- We need a data structure that is *Last in, first out*, or LIFO –
  a *stack*
  - Many examples in real life: stuff on top of your desk, trays in the cafeteria, discard pile in a card game, …

Stack Terminology

- *Top*: Uppermost element of stack,
  - first to be removed
- Bottom: Lowest element of stack,
  - last to be removed
- Elements are always inserted and removed from the top (LIFO)

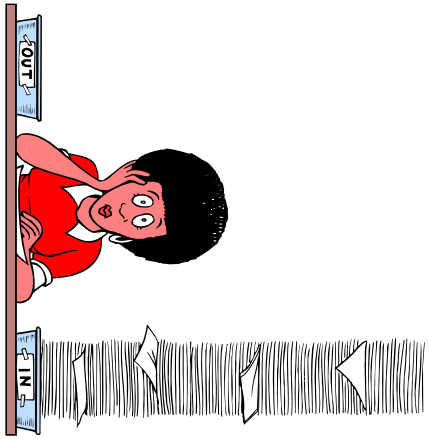


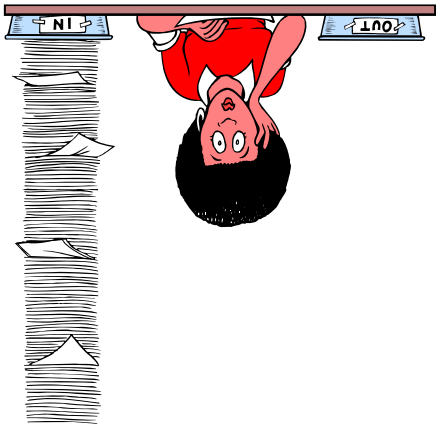
Stack Operations

- push(Object): Adds an element to top of stack, increasing stack height by one
- Object pop( ): Removes topmost element from stack and returns it, decreasing stack height by one
- Object top( ): Returns a copy of topmost element of stack, leaving stack unchanged
- No “direct access”
  - cannot index to a particular data item
- No convenient way to traverse the collection
  - Try it at home!

Picturing a Stack

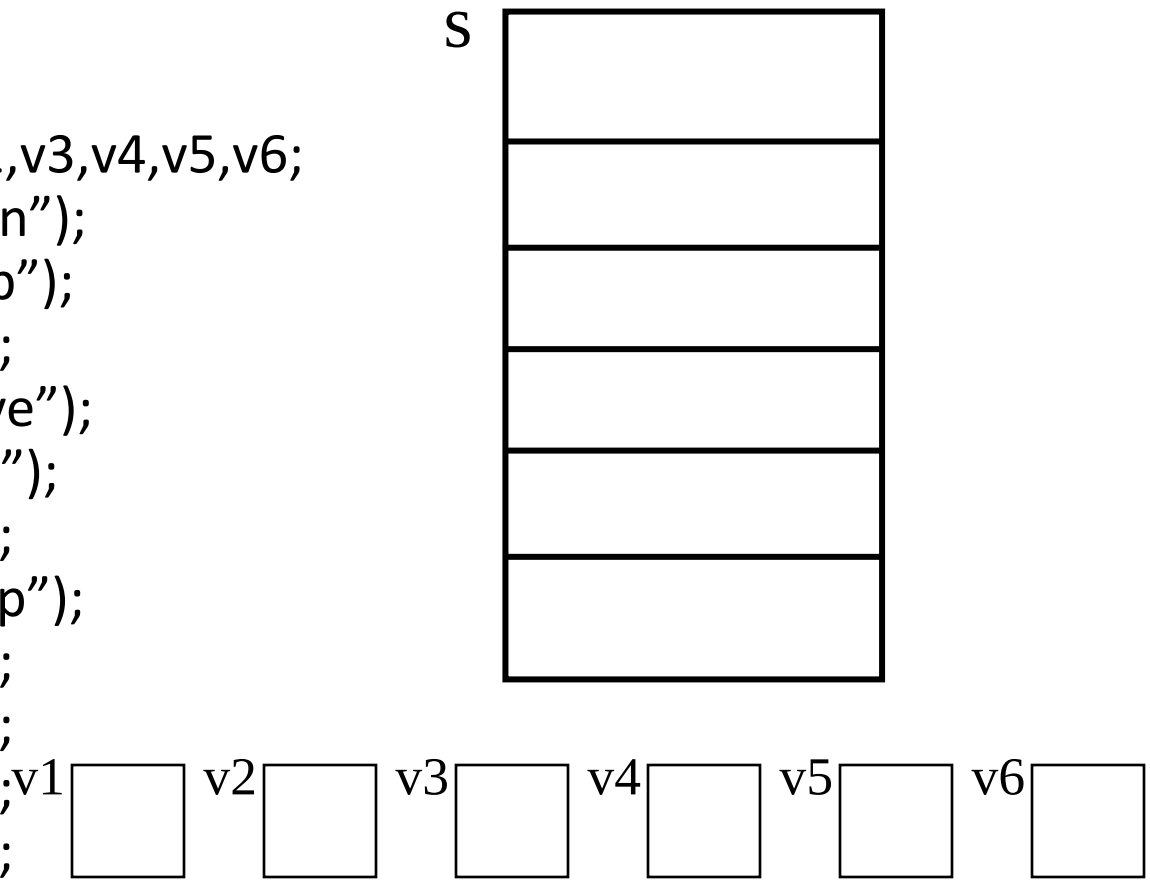
- Stack pictures are usually somewhat abstract
- "Top" of stack can be up, down, left, right – just label it.





What is the result of…

- Stack s;
- Object v1,v2,v3,v4,v5,v6;
- s.push(“Yawn”);
- s.push(“Burp”);
- v1 = s.pop( );
- s.push(“Wave”);
- s.push(“Hop”);
- v2 = s.pop( );
- s.push(“Jump”);
- v3 = s.pop( );
- v4 = s.pop( );
- v5 = s.pop( );
- v6 = s.pop( );



Stack Practice

- Show the changes to the stack in the following example:

  - Stack s;
  - Object obj;
  - s.push(“abc”);
  - s.push(“xyzzy”);
  - s.push(“secret”);
  - obj = s.pop( );
  - obj = s.top( );
  - s.push(“swordfish”);
  - s.push(“terces”);

Stack Implementations

- Several possible ways to implement
  - An array
  - A linked list
  - A vector
    - Useful thought problem: How would you do these?
  - push(Object) :: add(Object)
  - top( ) :: get(size( ) –1)
  - pop( ) :: remove(size( ) -1)
  - Precondition for top( ) and pop( ): stack not empty

Designing and Building a Stack class

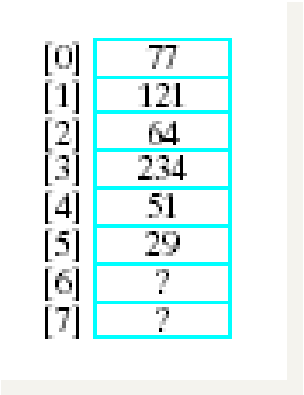
- The basic functions are:
  - Constructor: construct an empty stack
  - Empty(): Examines whether the stack is empty or not
  - Push(): Add a value at the top of the stack
  - Top(): Read the value at the top of the stack
  - Pop(): Remove the value at the top of the stack
  - Display(): Displays all the elements in the stack
  - Full(): Examines whether the stack is full

Selecting storage structures

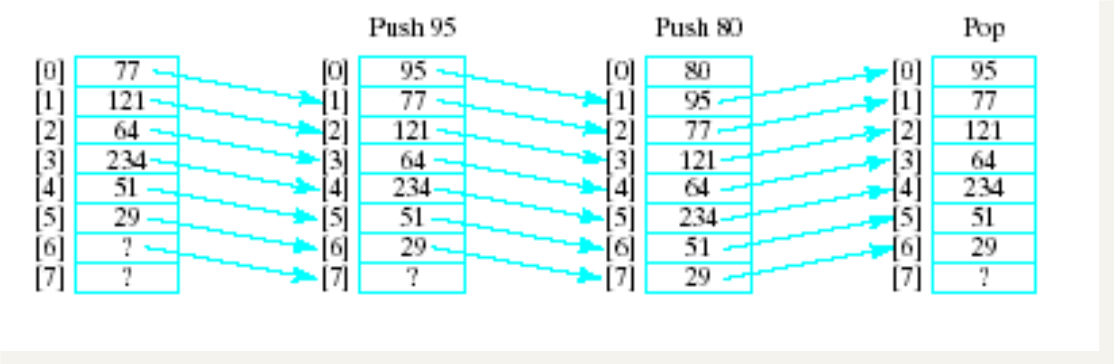
- Two choices
  - Select position 0 as top of the stack
  - Select position 0 as bottom of the stack

Select position 0 as top of the stack

- Model with an array
  - Let position 0 be top of stack

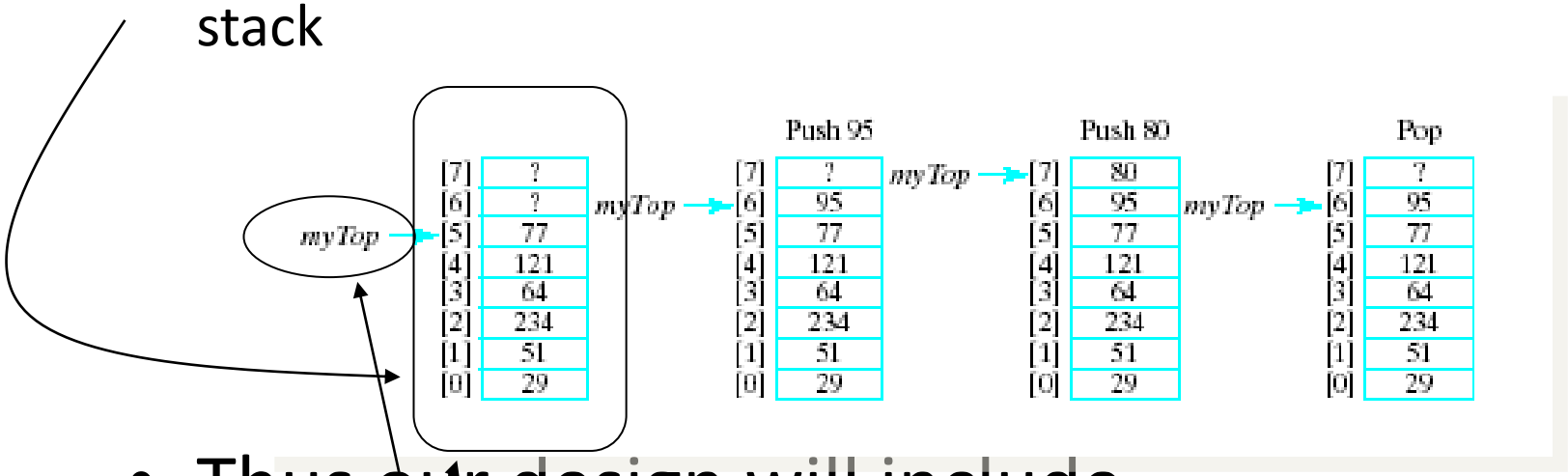


- Problem … consider pushing and popping
  - Requires much shifting



Select position 0 as bottom of the stack

- A better approach is to let position 0 be the <u>bottom</u> of the stack



- Thus our design will include
  - An <u>array</u> to hold the stack elements
  - An <u>integer</u> to indicate the top of the stack

Implementation of the Operations

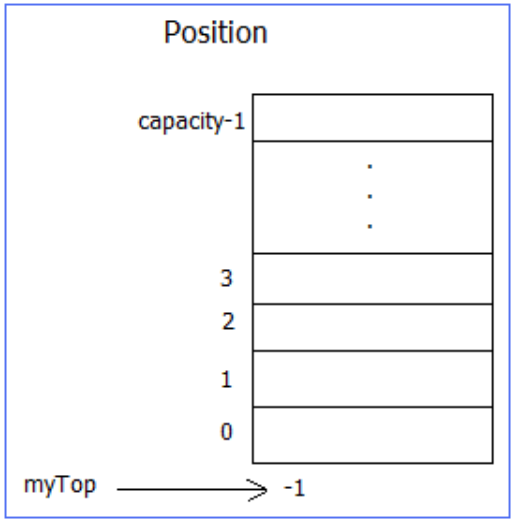
- Constructor:

  Create an array: (int) array[capacity]

  Set myTop = -1

- Empty():

  check if myTop == -1



Implementation of the Operations

- Push(int x):

```
if array is not FULL (myTop < capacity-1)
    myTop++
    store the value x in array[myTop]
else
    output “out of space”
```

Implementation of the Operations

- Top():

```
If the stack is not empty
        return the value in array[myTop]
else:
        output “no elements in the stack”
```

Implementation of the Operations

- Pop():

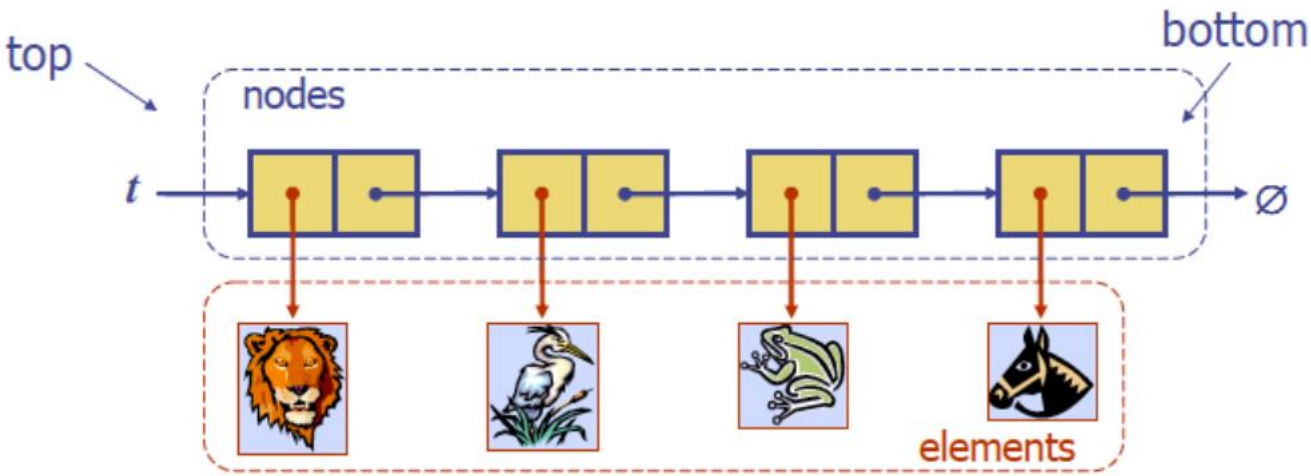
```
If the stack is not empty
        myTop -= 1
else:
        output “no elements in the stack”
```

Further Considerations

- What if static array initially allocated for stack is too small?
  - Terminate execution?
  - Replace with larger array!

- Creating a larger array
  - Allocate larger array
  - Use loop to copy elements into new array
  - Delete old array

Stack Implementation Using linked List



References and Credit

Data Structures and Algorithms in C++ Goodrich, Tamassia and Mount (Wiley, 2004)

University of Washington Data Structures and Algorithms

Dr. Bernard Chen University of Central Arkansas

University of Pennsylvania Data Structures and Algorithms