CRICOS PROVIDER 00123M

School of Computer Science

COMP SCI 1103/2103 Algorithm Design & Data Structure

Stacks

Abstract Data Types - revisited

- Recall the definition of ADT.
  - A data type consists of a collection of values together with a set of basic operations defined on those values.
  - A data type is called an abstract data type if the programmers who use the type do not have access to the implementation details.
- You should not need to know anything about the implementation in order to use that type
- ADT in C++
  - Built-in
  - User defined

List as an ADT

- A general list of form $A_0, A_1, \ldots, A_{n-1}$
- For any list except the empty list, we say that A_i follows A_{i-1} ($i<n$) and A_{i-1} precedes A_i

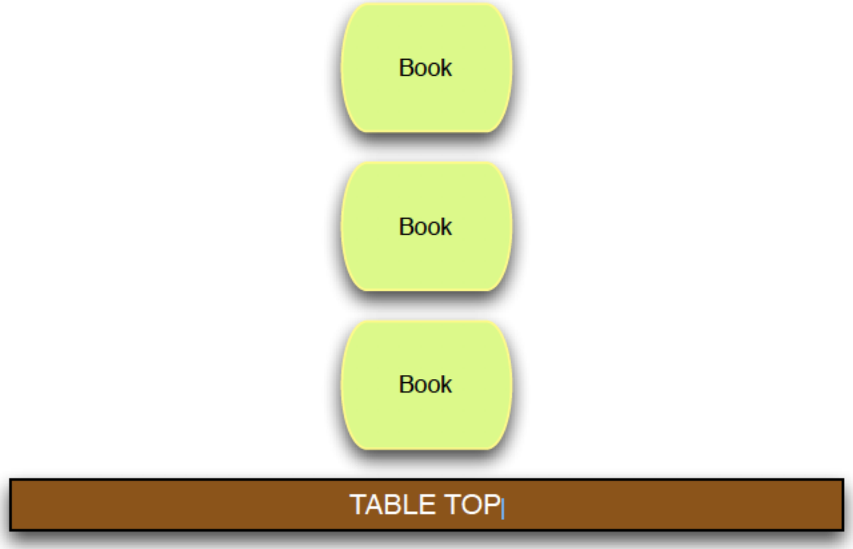
- Operations:
  - toString
  - isEmpty
  - search
  - insert
  - remove
  - getValue

Abstract Data Types we know

- We’re familiar with
  - strings, arrays, vectors
    - Efficient for accessing the elements by index (random access)
  - Linked lists
    - Dynamically grows and shrinks
- Both can be used for implementing a list
  - Differences in *time* complexity for different functions
  - Differences in *space* complexity
- Two common applications of lists:
  - stacks
  - queues

Stack

- A stack is a data structure that retrieves data in the opposite order to which it was stored.
- You only have access to the top of the stack.
  - Can only insert or delete from top
- This is called Last-In/First-Out (LIFO).
- Backseat of a taxi!
  Pile of Books.



Stack operations

- The operations associated with a stack are:
  - push - we put an element on top of the stack
  - pop - we take the top element off the stack and return it
  - isEmpty - we return true if the stack is empty, false otherwise

- Any preconditions for pop?

Stacks, using linked lists or arrays

- Stacks are very easy to implement with linked lists
  - push: add a node to the **top** of the list.
  - pop: remove the node at the **top**, returns the value and destroys the old node, updating **top** to point to the new top.
  - isEmpty: check to see if **top** points to NULL.

  - What do you think about the complexity of the operations?

Stacks, using linked lists or arrays

- Both implementations can guarantee O(1) complexity for the basic operations.
- When we know that the size of stack will not be too large, it is easier and more efficient to use arrays

Notes for stacks

- Black box
  - Inside, we may have a linked list or an array
  - While you know the whole chain is there, the functions that you use in this data structure **restrict** you to only accessing certain elements.
- This enforces the **LIFO** semantics of the data structure and this allows you to write your code knowing that this will be enforced.
- What would happen if your stack allowed random access?

Summary

- A stack is a data structure that retrieves data in the opposite order to which it was stored.
- You only have access to the top of the stack.
- This is called Last-In/First-Out (LIFO).

- The operations associated with a stack are:
  - push - we put an element on top of the stack
  - pop - we take the top element off the stack and return it
  - isEmpty - we return true if the stack is empty, false otherwise

THE UNIVERSITY
of ADELAIDE