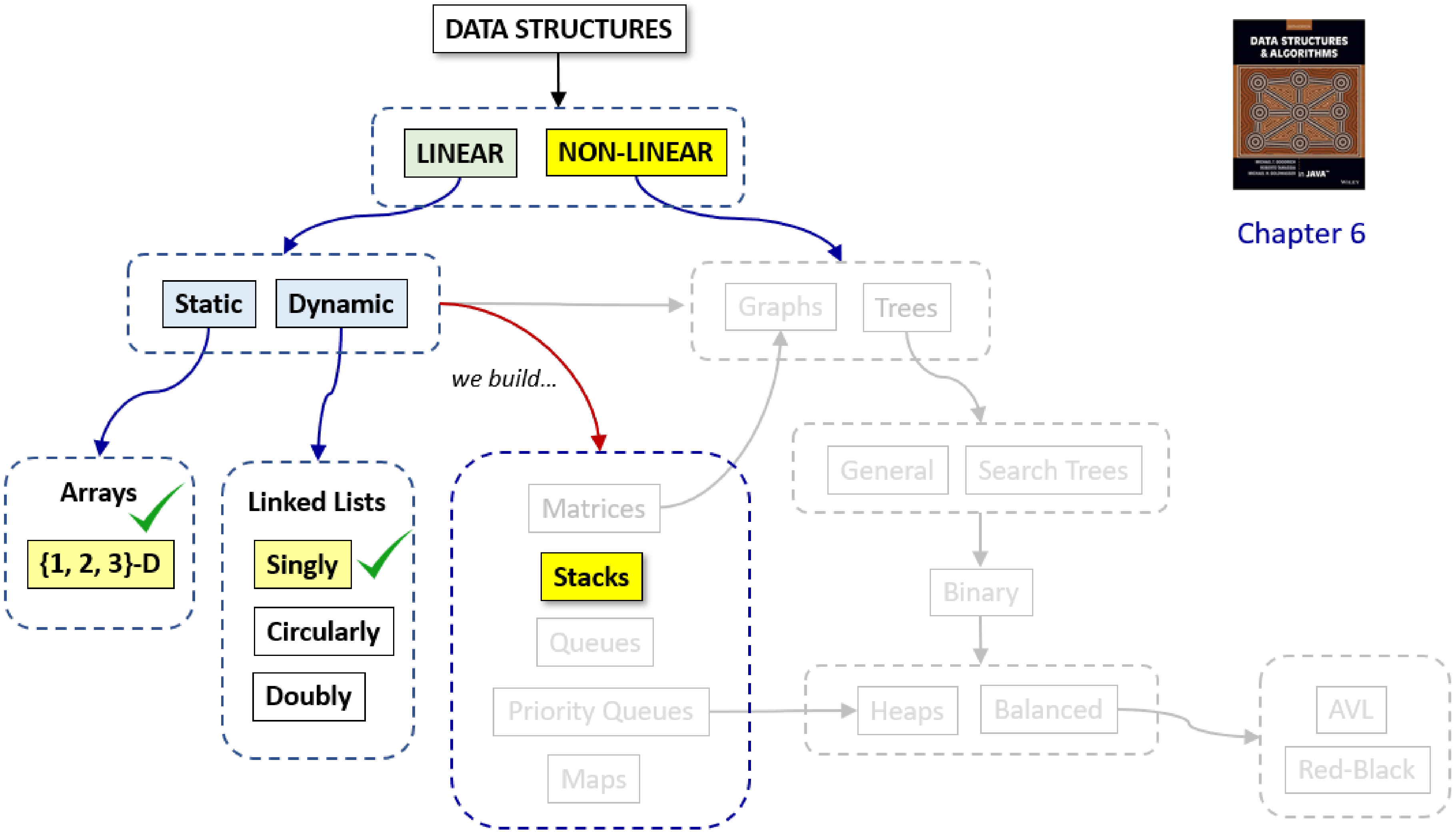
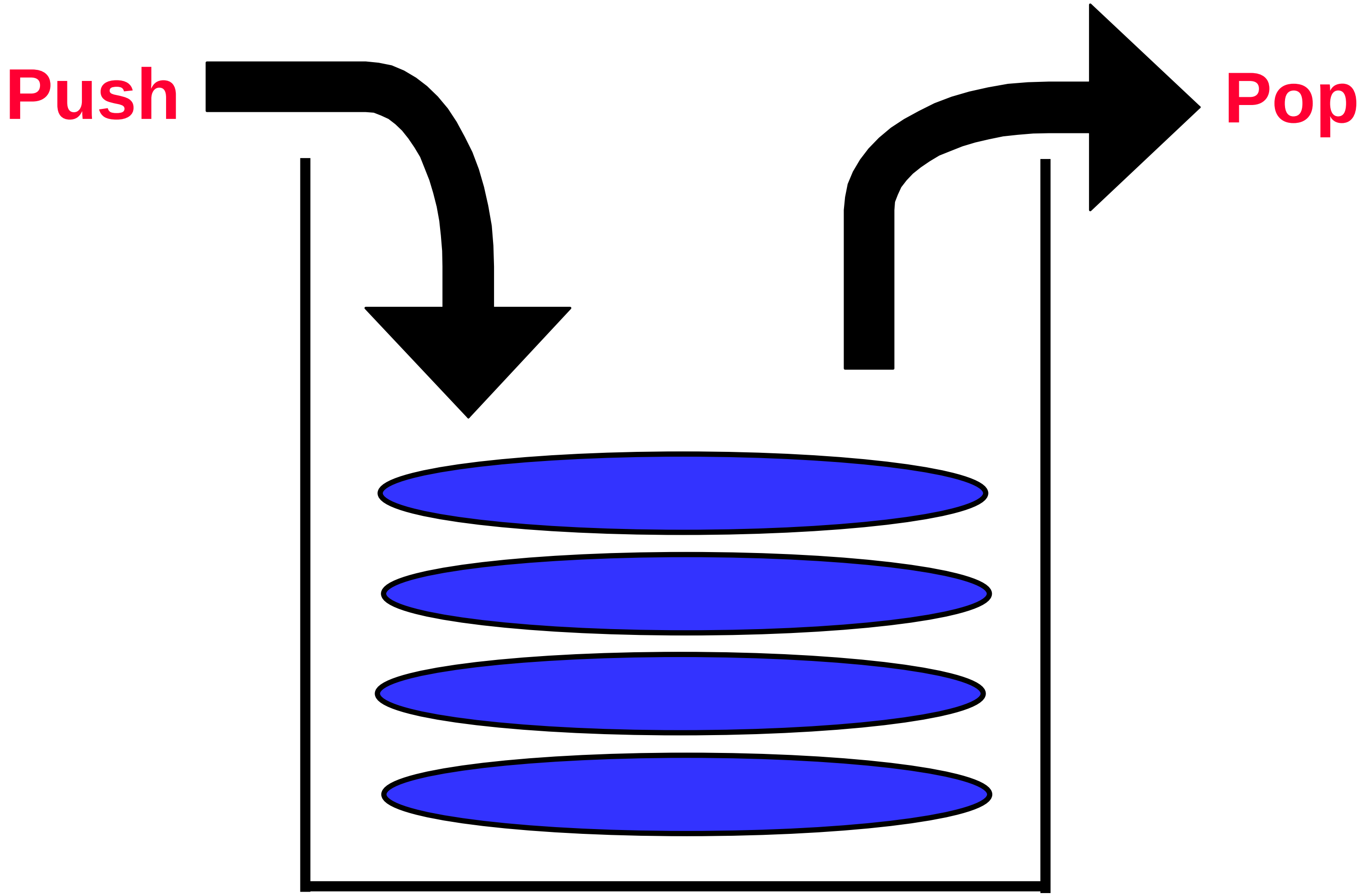
Stacks and Queues

UNIVERSITY OF CALGARY


DATA STRUCTURES
LINEAR
NON-LINEAR
Static
Dynamic
Graphs
Trees
we build...
Arrays
{1, 2, 3}-D
Linked Lists
Singly
Circularly
Doubly
Matrices
Stacks
Queues
Priority Queues
Maps
General
Search Trees
Binary
Heaps
Balanced
AVL
Red-Black
DATA STRUCTURES & ALGORITHMS

Chapter 6

The Stack



The Stack ADT

- The Stack ADT stores arbitrary objects
- Insertions and deletions follow the last-in first-out (LIFO) scheme
- Think of a spring-loaded plate dispenser
- Main stack operations:
  - push(object): inserts an element
  - object pop(): removes and returns the last inserted element

The Stack ADT

- Auxiliary stack operations:
  - object top(): returns the last inserted element without removing it
  - integer size(): returns the number of elements stored
  - boolean isEmpty(): indicates whether no elements are stored

Applications of Stacks

- **Direct applications**
  - Page-visited history in a Web browser
  - Undo sequence in a text editor
  - Activation Stack (recursive calls)
  - Chain of method calls in the Java Virtual Machine
  - (…)

Properties

Idea: a "Last In, First Out" (LIFO) data structure

Behaviors:

- **Push: Add to top of stack**
- **Pop: Remove from top of stack (and return that top value)**
- **Top: Return topmost item (but leave it on the stack)**
- **Is_Full: is it full?**
- **Is_Empty: is it empty?**
- **Initialize: empty stack**

The Stack as a Logical Data Structure

- **The stack is an idea**
- **It implies a set of logical behaviors**
- **It can be implemented various ways**
  - **Using a linked list or a tree or an array**
- **In this example, we'll focus on dynamic implementations using dynamic data...**

Array-based Stack

- A simple way of implementing the Stack ADT uses an array
- We add elements from left to right
- A variable keeps track of the index of the top element

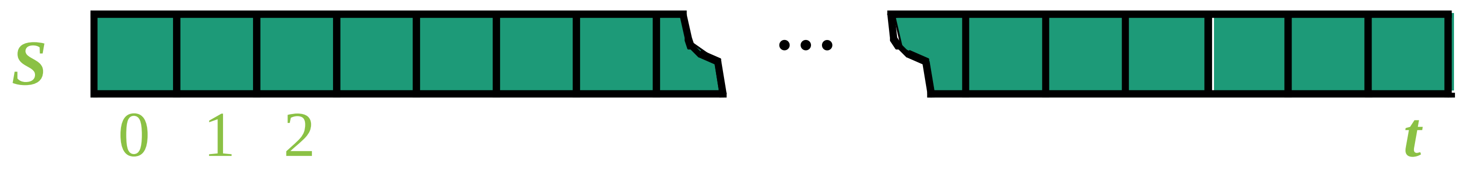
```
Algorithm size()
  return t + 1

Algorithm pop()
  if isEmpty() then
    throw EmptyStackException
   else
    t ← t − 1
    return S[t + 1]
```

Array-based Stack (cont.)

- The array storing the stack elements may become full
- A push operation will then throw a FullStackException
  - Limitation of the array-based implementation
  - Not intrinsic to the Stack ADT

Algorithm *push*(*o*)
  if $t = S.length - 1$ **then**
    throw *FullStackException*
  else
    $t \leftarrow t + 1$
    $S[t] \leftarrow o$

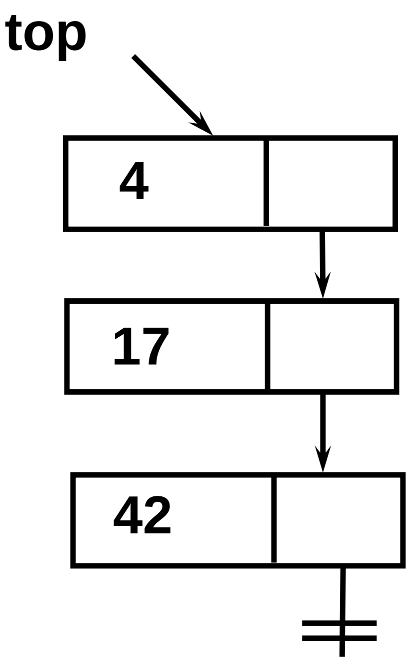


Performance and Limitations

- **Performance**
  - Let $\boldsymbol{n}$ be the number of elements in the stack
  - The space used is $\boldsymbol{O}(\boldsymbol{n})$
  - Each operation runs in time $\boldsymbol{O}(1)$
- **Limitations**
  - The maximum size of the stack must be defined a priori and cannot be changed
  - Trying to push a new element into a full stack causes an implementation-specific exception

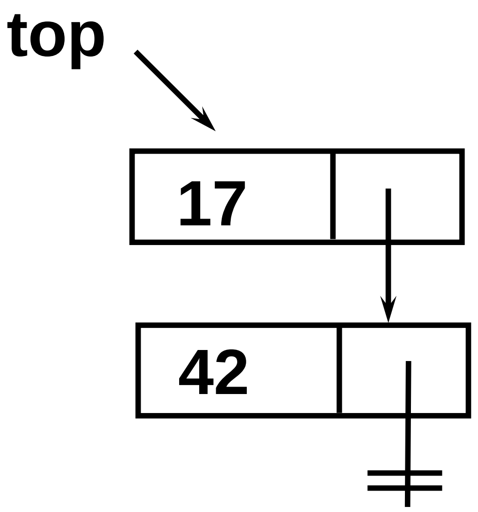
Stacks:
Dynamic Implementation

- **A singly linked list with restricted set of operations to change its state: only modified from one end**



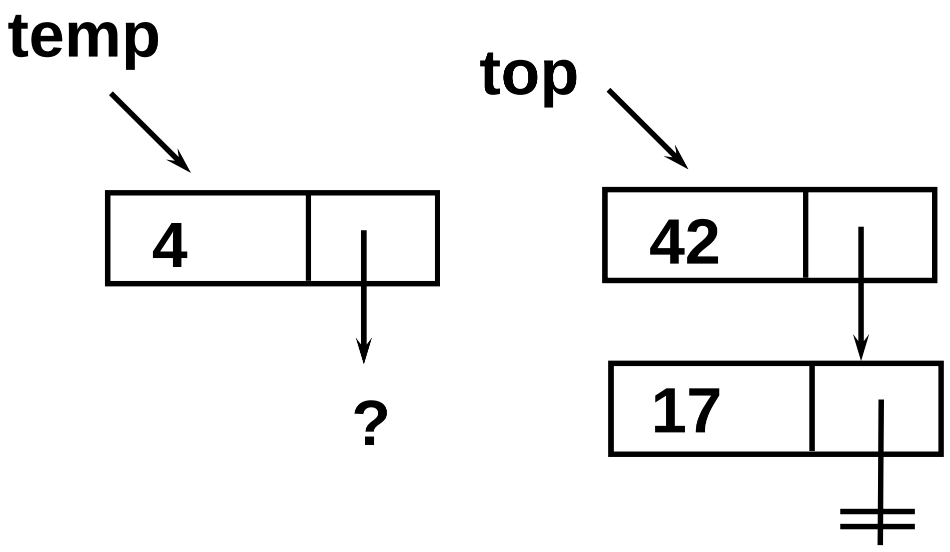
Push

- **Create new node**
- **Add it to the front**



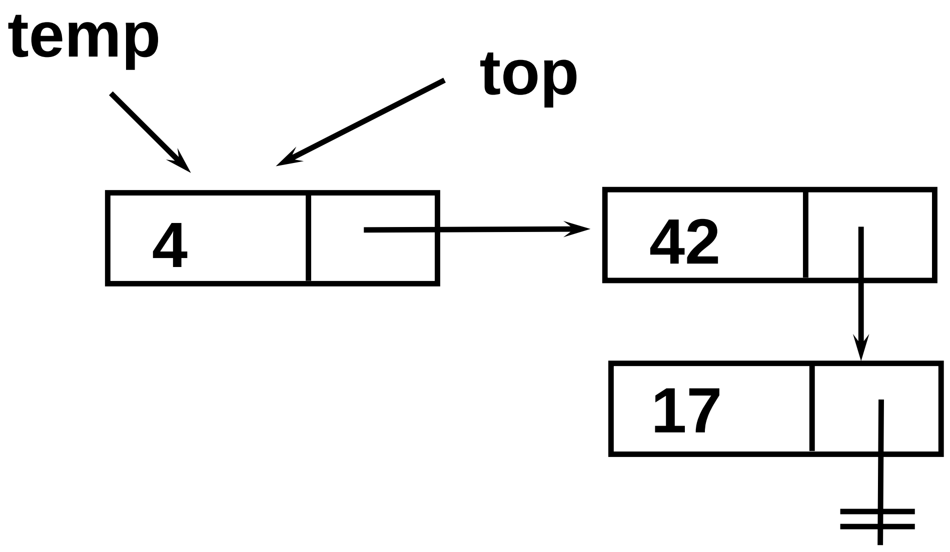
Push

- Create new node
- **Add it to the front**



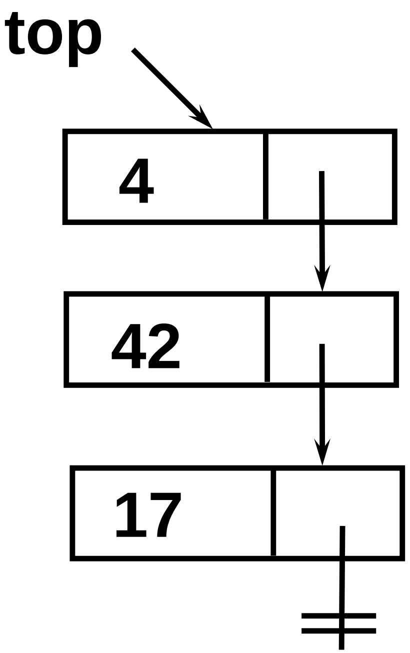
Push

- **Create new node**
- **Add it to the front**



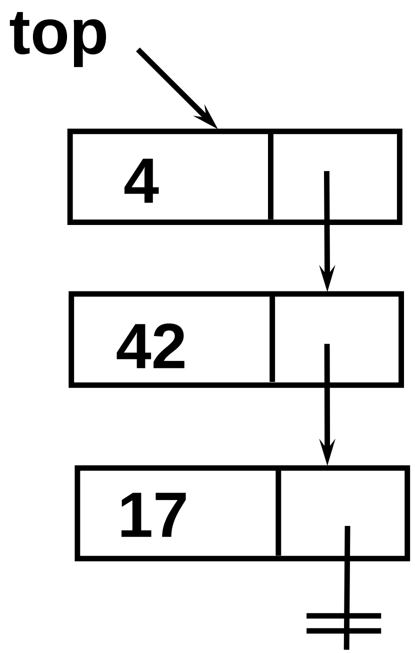
Push

- **Create new node**
- **Add it to the front**



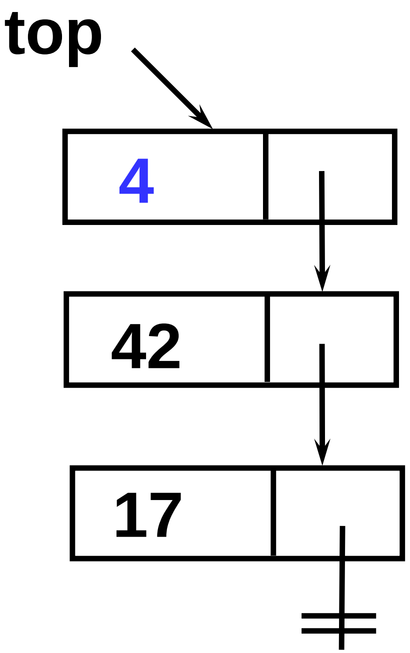
Pop

- **Capture the first value (to return)**
- **Remove the first node (move top to next)**



Pop

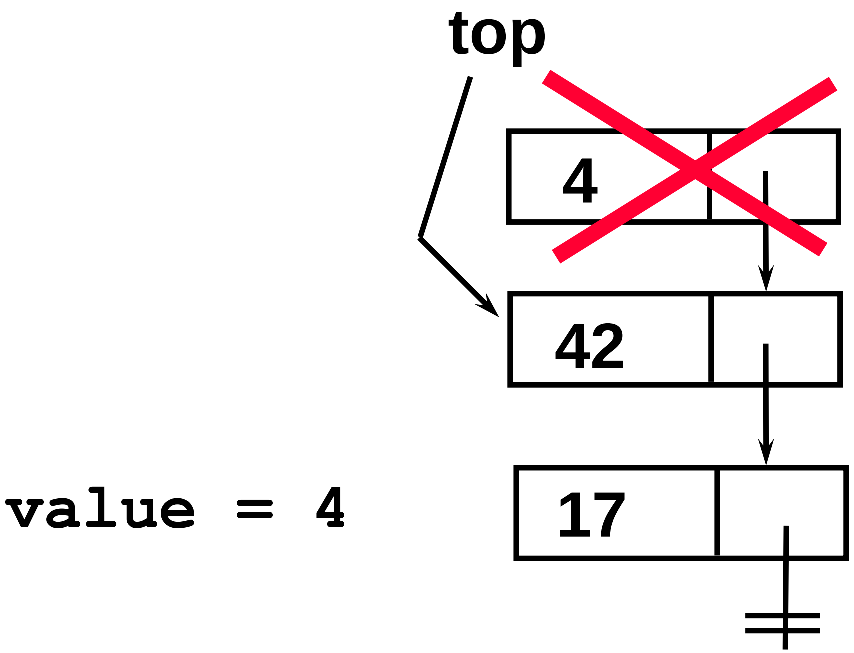
- Capture the first value (to return)
- **Remove the first node (move top to next)**




```
value = 4
```

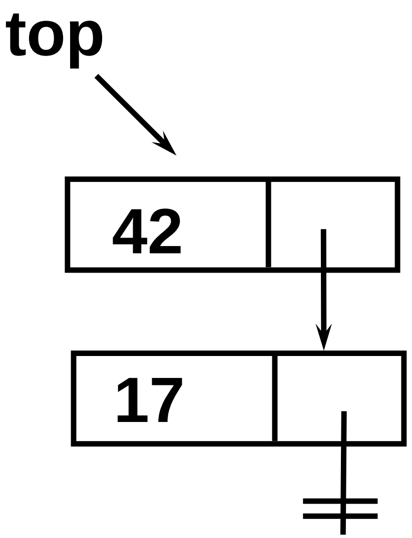
Pop

- **Capture the first value (to return)**
- **Remove the first node (move top to next)**



Pop

- **Capture the first value (to return)**
- **Remove the first node (move top to next)**



`value (4) is returned`

Summary: Stack

- **Allow us to model "last-in, first-out" (LIFO) behavior**
- **Can be implemented using different data types**
- **Behavior is important (and defines a Stack)**
  - **Push to the front**
  - **Pop from the front**
  - **(from the same end)**

Why Use an Arrayed Stack?

- No overhead as linked stacks
- <u>Major disadvantage</u>: amount of time to resize → O(n) algorithm

Why Use a Linked Stack?

- Push as many items as you want
- <u>Every operation at the end of the list</u> → pushing and popping are both O(1) algorithms