# Dynamic arrays

Christophe Rhodes

# Motivation

- constant-time access of (fixed) arrays
- extensibility of linked lists
- Java: `ArrayList`, C++ `std::vector`

*We can solve any problem [in Computer Science] by introducing an extra level of indirection. – David J. Wheeler*

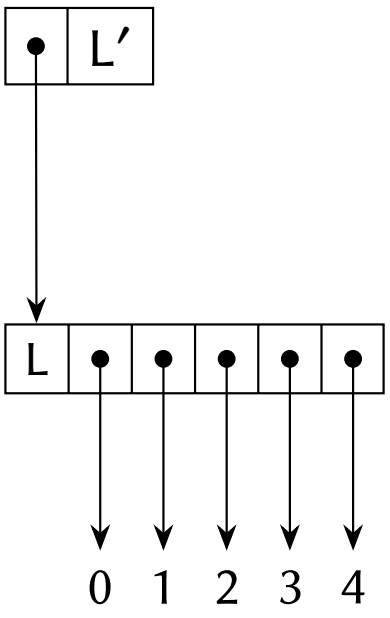
# Definition

A dynamic array is a finite sequential collection of data.
(removal of “fixed-size” from the definition of vector)

# Operations

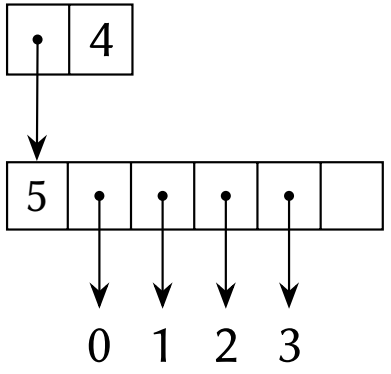
| | |
|---|---|
| length | return the current size of the dynamic array |
| select[k] | return the $k^{th}$ element of the dynamic array |
| store![o,k] | set the $k^{th}$ element of the array to o |
| push![o] | increase the length of the dynamic array by 1, and set the endmost element to o |
| pop! | return the endmost element, decreasing the size of the dynamic array by 1 |

# Implementation



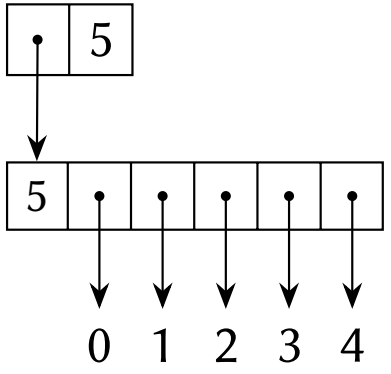
# Push!

**Require:** A :: dynamic array
  **function** PUSH!(A,k)
    **if** LENGTH(LEFT(A)) = RIGHT(A) **then**
      EXTEND(A)
    **end if**
    A[RIGHT(A)] ⟵ k
    RIGHT(A) ⟵ RIGHT(A) + 1
  **end function**



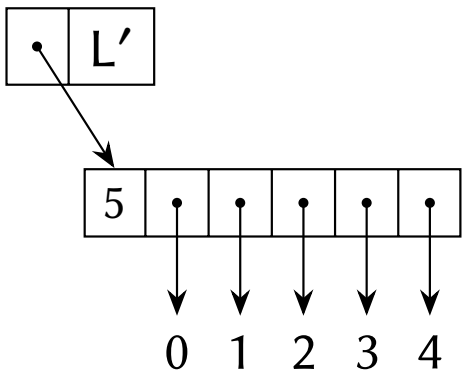
# Push!

**Require:** A :: dynamic array
  **function** PUSH!(A,k)
    **if** LENGTH(LEFT(A)) = RIGHT(A) **then**
      EXTEND(A)
    **end if**
    A[RIGHT(A)] ⟵ k
    RIGHT(A) ⟵ RIGHT(A) + 1
  **end function**



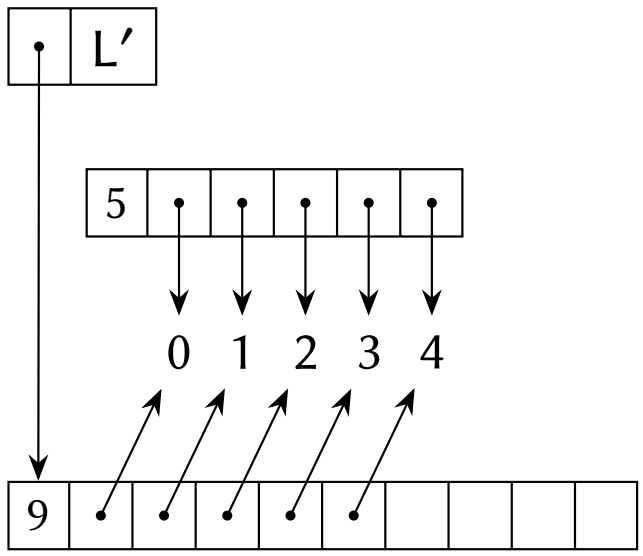
# Extend

**Require:** A :: dynamic array
  **function** EXTEND(A)
    newL ← NEWLENGTH(RIGHT(A))
    new ← **new** Vector(newL)
    **for** 0 ≤ i < LENGTH(A) **do**
      new[i] ← LEFT(A)[i]
    **end for**
    LEFT(A) ← new
  **end function**



# Extend

**Require:** A :: dynamic array
  **function** EXTEND(A)
    newL ⟵ NEWLENGTH(RIGHT(A))
    new ⟵ **new** Vector(newL)
    **for** 0 ≤ i < LENGTH(A) **do**
      new[i] ⟵ LEFT(A)[i]
    **end for**
    LEFT(A) ⟵ new
  **end function**




What should NEWLENGTH(n) be?

- return n + C (*e.g.* n + 10)?
- return C × n (*e.g.* 2 × n)?
- return $n^C$ (*e.g.* $n^2$)?

# Complexity analysis

## length, select, store!

- each is a pointer read (to get the storage array) and a $\Theta(1)$ array operation

$$\Longrightarrow \Theta(1)$$

## push!

Usual case:

- increment length
- store value in storage array

$$\Longrightarrow \Theta(1)$$

When extending storage array:

- as above plus...
- ... copy existing contents to new array

$$\Longrightarrow \Theta(N)$$

# Work

1. Reading
   - CLRS, section 17.4
2. Implement a dynamic array using a pair and an array (as shown in these slides). What will you do with the storage array when implementing POP!?