

# Lecture 8: ADT List

*CS2504/CS4092– Algorithms and Linear Data Structures*

Dr. Kieran T. Herley

Department of Computer Science  
University College Cork

2013/14

## Summary

*The concept of a list and ADT List. Definition and usage of ADT. Array-based implementation of ADT.*

# Lists in the Real World

**List** An ordered finite sequence of items

## Examples

- Class list: list of names
- To-do/shopping list: list of tasks/goods
- First  $n$  digits of  $\pi$ : list of digits
- Book: list of chapters; chapter: list of paragraphs; paragraph: list of sentences; sentence: list of words

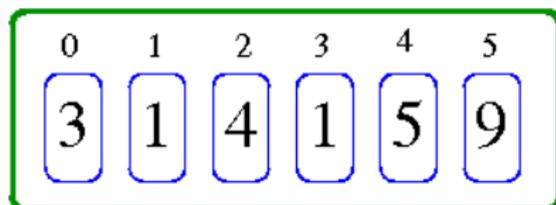
## Notes

- Order is significant–  $(3, 1, 4, 1) \neq (1, 4, 3, 1)$
- Duplicate items allowed

# ADT List

**ADT List** Container type; represents indexed sequence of elements of some type; supports operations to get/set specified elements as well as add and remove elements.

**Pic**



**Note** This is a simplified cousin of Java's array-list concept

## ADT List

A list is a container capable of holding an ordered arrangement of elements. The *index* of an element is the number of elements that precede it in the list.

**get(*inx*):** Return the element at specified index. Illegal if no such index exists<sup>1</sup>. *Input: int; Output: E.*

**set(*inx*, *newElt*):** Replace the element at specified index with *newElt*. Return the old element at that index. Illegal if no such index exists<sup>1</sup>. *Input: int, E; Output: E.*

## ADT List

A list is a container capable of holding an ordered arrangement of elements. The *index* of an element is the number of elements that precede it in the list.

**get(inx):** Return the element at specified index. Illegal if no such index exists<sup>1</sup>. *Input: int; Output: E.*

**set(inx, newElt):** Replace the element at specified index with newElt. Return the old element at that index. Illegal if no such index exists<sup>1</sup>. *Input: int, E; Output: E.*

**add(inx, newElt):** Add element newElt to the list at index inx. Illegal if inx is negative or greater than current list size<sup>1</sup>. *Input: int, E; Output: None.*

## ADT List

A list is a container capable of holding an ordered arrangement of elements. The *index* of an element is the number of elements that precede it in the list.

**get(*inx*):** Return the element at specified index. Illegal if no such index exists<sup>1</sup>. *Input: int; Output: E.*

**set(*inx*, *newElt*):** Replace the element at specified index with *newElt*. Return the old element at that index. Illegal if no such index exists<sup>1</sup>. *Input: int, E; Output: E.*

**add(*inx*, *newElt*):** Add element *newElt* to the list at index *inx*. Illegal if *inx* is negative or greater than current list size<sup>1</sup>. *Input: int, E; Output: None.*

**remove(*inx*):** Remove the element at the specified index from the list and return it. Illegal if no such index exists<sup>1</sup>. *Input: int; Output: E.*

ADT List also supports some other ops (incl. *size*, *isEmpty*), see handout for complete list.

## List Ops. Illustrated

Operation	Output	$L$
add(0, 7)	-	(7 <sub>0</sub> )
add(0, 4)	-	(4 <sub>0</sub> , 7 <sub>1</sub> )
get(1)	7	(4 <sub>0</sub> , 7 <sub>1</sub> )
add(2, 2)	-	(4 <sub>0</sub> , 7 <sub>1</sub> , 2 <sub>2</sub> )
get(3)	Error	(4 <sub>0</sub> , 7 <sub>1</sub> , 2 <sub>2</sub> )
remove(1)	7	(4 <sub>0</sub> , 2 <sub>1</sub> )
add(1, 5)	-	(4 <sub>0</sub> , 5 <sub>1</sub> , 2 <sub>2</sub> )
add(1, 3)	-	(4 <sub>0</sub> , 3 <sub>1</sub> , 5 <sub>2</sub> , 2 <sub>3</sub> )
add(4, 9)	-	(4 <sub>0</sub> , 3 <sub>1</sub> , 5 <sub>2</sub> , 2 <sub>3</sub> , 9 <sub>4</sub> )
get(2)	5	(4 <sub>0</sub> , 3 <sub>1</sub> , 5 <sub>2</sub> , 2 <sub>3</sub> , 9 <sub>4</sub> )
set(3, 8)	2	(4 <sub>0</sub> , 3 <sub>1</sub> , 5 <sub>2</sub> , 8 <sub>3</sub> , 9 <sub>4</sub> )

Note: subscripts indicate indices

# Simple List Manipulation

**Goal** Algorithm that traverses a list of numbers and deletes all the odd numbers.

## Algorithm

**Algorithm** RemoveOdds(L):

$i \leftarrow 0$

**while**  $i < L.size()$  **do**

**if**  $odd(L.get(i))$  **then**

$L.remove(i)$

**else**

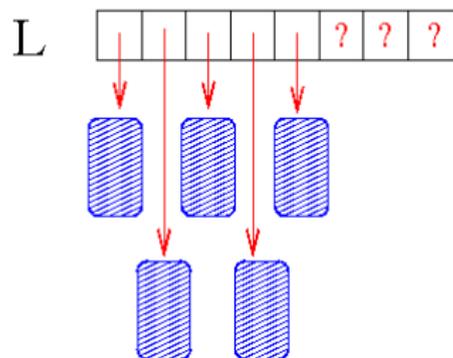
$i \leftarrow i + 1$

Why "else"?

# Implementing ADT List

Representation

$n = 5$



- L** left-justified array of elements
- n** the number of elements in the list

Implementation

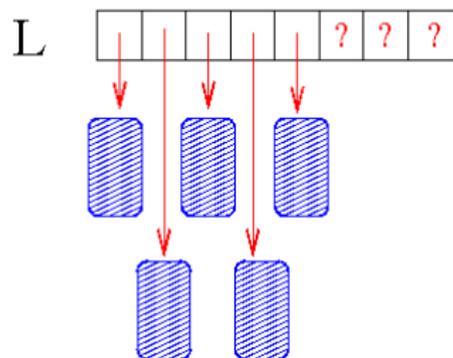
**Construction** create array;  
initialize  $n$  to zero

**get/set** probe/change specified  
index in array

# Implementing ADT List

Representation

$n = 5$



Implementation

**Construction** create array;  
initialize  $n$  to zero

**get/set** probe/change specified  
index in array

**add** make room for element by  
shifting others rightwards as  
necessary; insert new elt. in  
specified index; increment  $n$

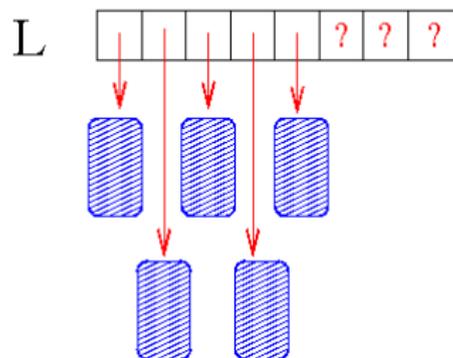
**L** left-justified array of elements

**n** the number of elements in the  
list

# Implementing ADT List

Representation

$n = 5$



- L** left-justified array of elements
- n** the number of elements in the list

Implementation

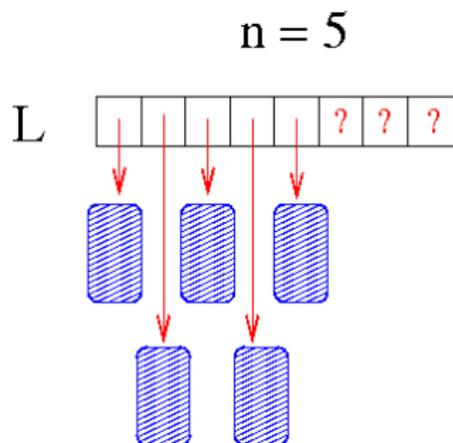
**Construction** create array;  
initialize  $n$  to zero

**get/set** probe/change specified  
index in array

**add** make room for element by  
shifting others rightwards as  
necessary; insert new elt. in  
specified index; increment  $n$

**remove** extract specified element;  
shift others leftwards as necessary  
to delete; decrement  $n$

# Simple Operations



**Algorithm** size:

```
return n;
```

**Algorithm** isEmpty:

```
return (n == 0)
```

**Algorithm** get(i):

```
return L[i]
```

**Algorithm** set(i, elt):

```
old ← L[i]
```

```
L[i] ← elt
```

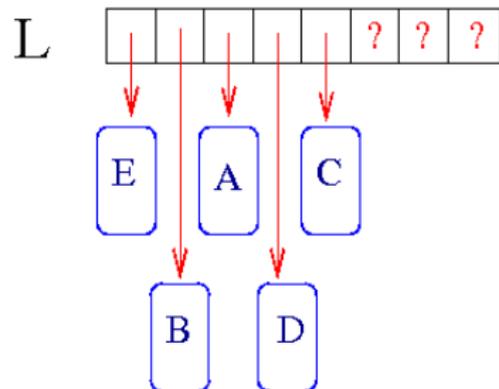
```
return old
```

Error checking (e.g. illegal indices)  
omitted

## Method remove

remove(1)

$n = 5$



**Algorithm** remove(i):

```
e ← L[i]
```

```
for j = i to n-2 do
```

```
    L[j] ← L[j+1]
```

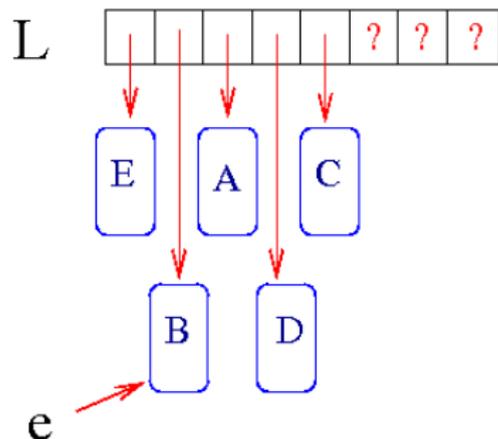
```
n ← n - 1
```

```
return e
```

## Method remove

remove(1)

$n = 5$



**Algorithm** remove( $i$ ):

$e \leftarrow L[i]$

**for**  $j = i$  **to**  $n-2$  **do**

$L[j] \leftarrow L[j+1]$

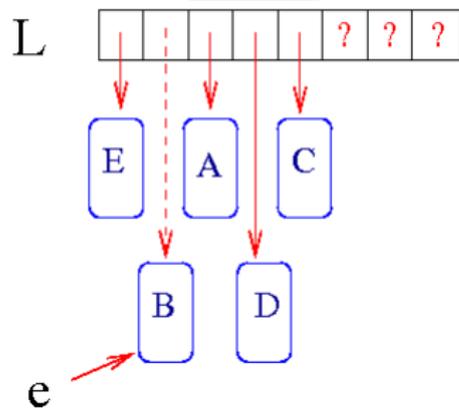
$n \leftarrow n - 1$

**return**  $e$

## Method remove

remove(1)

$n = 5$



**Algorithm** remove( $i$ ):

$e \leftarrow L[i]$

**for**  $j = i$  **to**  $n-2$  **do**

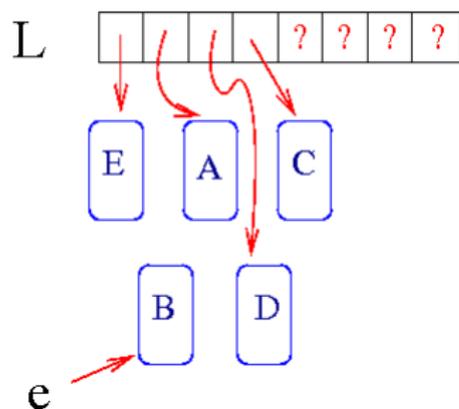
$L[j] \leftarrow L[j+1]$

$n \leftarrow n - 1$

**return**  $e$

## Method remove

remove(1)  
 $n = 4$



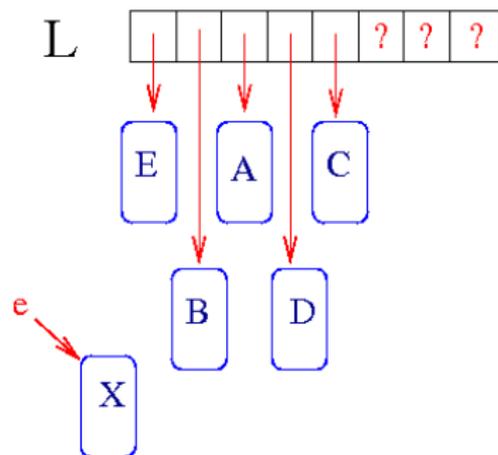
**Algorithm** remove(i):

```
e ← L[i]
for j = i to n-2 do
    L[j] ← L[j+1]
n ← n - 1
return e
```

## Method add

add(1, X):

$n = 5$



**Algorithm** add( $i, e$ ):

**for**  $j \leftarrow n-1, n-2, \dots, i$  **do**

$L[j+1] \leftarrow L[j]$

$L[i] \leftarrow e$

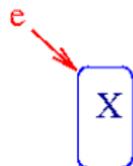
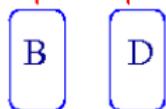
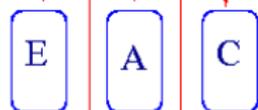
$n \leftarrow n + 1$

## Method add

add(1, X):

$n = 5$

shift



**Algorithm** add( $i, e$ ):

**for**  $j \leftarrow n-1, n-2, \dots, i$  **do**

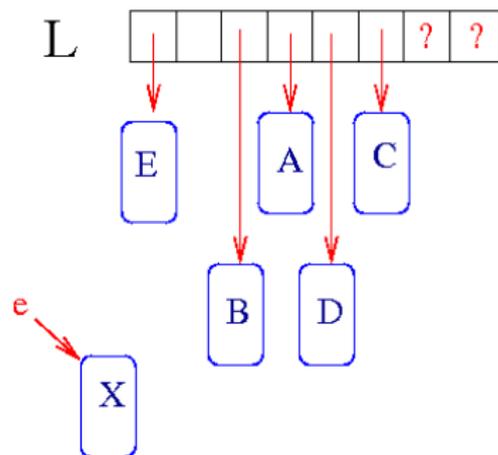
$L[j+1] \leftarrow L[j]$

$L[i] \leftarrow e$

$n \leftarrow n + 1$

## Method add

add(1, X):  
n = 5

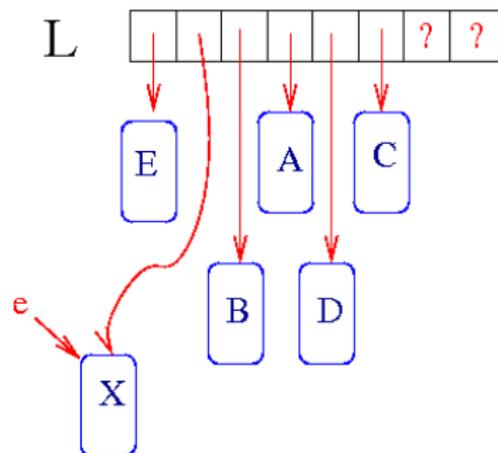


**Algorithm** add(i, e):

```
for j ← n-1, n-2, . . . , i do
    L[j+1] ← L[j]
L[i] ← e
n ← n + 1
```

## Method add

add(1, X):  
 $n = 6$



**Algorithm** add( $i, e$ ):

```
for  $j \leftarrow n-1, n-2, \dots, i$  do  
     $L[j+1] \leftarrow L[j]$   
 $L[i] \leftarrow e$   
 $n \leftarrow n + 1$ 
```

## List/ArrayBasedList (Simplified)

```
public interface List<EltType>
{
    . . .
    public EltType get(int inx);
    . . .
}
```

```
public class ArrayBasedList<EltType>
    implements List<EltType>
{
    public ArrayBasedList()
    {
        numElts = 0;
        capacity = INIT_CAP;
        elements = (EltType[])
            (new Object[capacity]);
    }
    . . .
    public EltType get(int inx)
    {
        . . .
        return elements[inx];
    }

    private static final int
    INIT_CAP = 10;
    private int capacity;
    protected EltType elements[];
    protected int numElts;
}
```

See web page for complete implementation.