ADT
Lists, Stacks, and Queues

Instructor: Ahmed Eldawy
Objectives

- Understand the importance of ADT
- Learn how to implement ADT in C++
- Recognize the difference between ADT definition and implementation
- Build an ADT for lists
Abstraction
Abstract Data Types

ADT

- Physical Memory Structure
- Algorithm Implementation

Operations

Application Programs
Abstraction in C++

Class

Private member variables and constants

Public methods

Private methods

Application Programs
Example: Rational Numbers

Class

Numerator
Denominator

GCD(x, y)
Normalize()

Create, +, -, *, /, print, …

Application Programs
ADT Design

- What is ADT design?
  - Defining the public interface
- Who designs an ADT?
  - You!
  - With your users
  - Sometimes, YOU are your own user
Lists

- List: A sequence of zero or more elements $A_1, A_2, \ldots, A_N$
- $N$: Size or length of the list
- $A_1$: First element
- $A_N$: Last element
- The order of items should be preserved
List ADT

- `initialize()`: Creates an empty list
- `push_back(x)`: Appends the item x to the end of the list
- `pop_back()`: Removes the last element
- `push_front(x)`: Prepends the item x at the beginning of the list
- `pop_front()`: Removes the first element
- `insert(x, i)`: Inserts item x at position i
- `erase(i)`: Deletes item at position i
- `find(x)`: Finds the position of the element with value x
- `size()`: Returns the number of elements
Array Implementation of List

List capacity

List size

Consecutive memory space

\[ A_1 \]
\[ A_2 \]
\[ \ldots \]
Initialize

List capacity

C=10

List size

N=0

initialize() {
    C=10 // Initial capacity
    N=0 // Initial size
    Allocate a memory space for C elements
}


push_back(x) 

push_back(x) {
    if (N==C) the Expand A
    N = N + 1
    A_N = x
}

List capacity
C

List size
N++

A_1
A_2
...
A_N
  x
push_front(x) {
    if (N == C) the Expand A
    Shift all elements $A_1$ to $A_N$ by one position
    $A_1 = x$
    $N = N + 1$
}
```c
insert(i, x) {
    if (N == C) the Expand A
    Shift all elements \( A_i \) to \( A_N \) by one position
    \( A_i = x \)
    \( N = N + 1 \)
}
```

<table>
<thead>
<tr>
<th>List capacity</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>List size</td>
<td>N++</td>
</tr>
</tbody>
</table>

| \( A_1 \)     |
| \( A_2 \)     |
| \( \ldots \)  |
| \( x \)       |
| \( \ldots \)  |
| \( A_N \)     |
erase(i) {
    Shift all elements $A_{i+1}$ to $A_N$
    by one position
    $N = N - 1$
}
pop_back()

pop_back() {
    N = N - 1
}

List capacity | C
List size     | N--

A_1
A_2
...
A_N
pop_front()

pop_front() {
    Shift all elements $A_1$ to $A_N$ by one position
    $N = N - 1$
}

List capacity

List size

![List diagram with elements $A_1$ to $A_N$.]
Linked-list Implementation

List size

Head $A_1$ → $\ldots$ → $A_N$

Tail

Null

List size $N$
Initialize

initialize() {
  N=0
  Tail = Head = Null
}

List size

N=0

Tail → Null
Head → Null
push_back(x)

push_back(x) {
    N = N + 1
    n = Allocate new node
    n.next = null
    n.value = x
    if (Head is null) {
        Head = Tail = n
    } else {
        Tail.next = n
        Tail = n
    }
}
push_front(x) { 
    N = N + 1
    n = Allocate new node
    n.next = Head
    n.value = x
    if (Head is null) {
        Head = Tail = n
    } else {
        Head = n
    }
}
pop_front(x)

pop_front(x) {
    if N=0 then raise exception
    N = N - 1
    old_node = Head
    Head = Head.next
    delete old_node
    // Are we done?
}

Head → A₁ → ... → AN → Null

List size: N--
# Array Vs Linked-list

<table>
<thead>
<tr>
<th>Operation</th>
<th>Array</th>
<th>Linked-list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>push_back</td>
<td></td>
<td></td>
</tr>
<tr>
<td>push_front</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pop_back</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pop_front</td>
<td></td>
<td></td>
</tr>
<tr>
<td>find</td>
<td></td>
<td></td>
</tr>
<tr>
<td>erase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clear</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Array Vs Linked-list

<table>
<thead>
<tr>
<th>Operation</th>
<th>Array</th>
<th>Linked-list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>push_back</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>push_front</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>pop_back</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>pop_front</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>find</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>erase</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>clear</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>