Vectors
Outline and Reading

- The Vector ADT (§5.1.1)
- Array-based implementation (§5.1.2)
The Vector ADT

The Vector ADT stores objects to which it provides direct access.

An element can be accessed, inserted or removed by specifying its rank (number of elements preceding it).

An exception is thrown if an incorrect rank is specified (e.g., a negative rank).

Main vector operations:

- `elemAtRank(int r)`: returns the element at rank `r` without removing it.
- `replaceAtRank(int r, Object o)`: replace the element at rank `r` with `o`.
- `insertAtRank(int r, Object o)`: insert a new element `o` to have rank `r`.
- `removeAtRank(int r)`: removes the element at rank `r`.

Additional operations `size()` and `isEmpty()`.
Applications of Vectors

- Direct applications
  - Sorted collection of objects (elementary database)

- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures
Array-based Vector

- Use an array $V$ of size $N$.
- A variable $n$ keeps track of the size of the vector (number of elements stored).
- Operation $\text{elemAtRank}(r)$ is implemented in $O(1)$ time by returning $V[r]$. 

![Diagram of an array-based vector with elements indexed from 0 to n-1]
Insertion

In operation $\text{insertAtRank}(r, o)$, we need to make room for the new element by shifting forward the $n - r$ elements $V[r], \ldots, V[n - 1]$

In the worst case ($r = 0$), this takes $O(n)$ time
Deletion

In operation `removeAtRank(r)`, we need to fill the hole left by the removed element by shifting backward the $n - r - 1$ elements $V[r + 1], \ldots, V[n - 1]$. In the worst case ($r = 0$), this takes $O(n)$ time.
Performance

In the array based implementation of a Vector

- The space used by the data structure is $O(n)$
- `size`, `isEmpty`, `elemAtRank` and `replaceAtRank` run in $O(1)$ time
- `insertAtRank` and `removeAtRank` run in $O(n)$ time

In an `insertAtRank` operation, when the array is full, instead of throwing an exception, we can replace the array with a larger one.
Lists and Sequences
Outline and Reading

- Singly linked list
- Position ADT and List ADT (§5.2.1)
- Doubly linked list (§ 5.2.3)
- Sequence ADT (§5.3.1)
- Implementations of the sequence ADT (§5.3.3)
A singly linked list is a concrete data structure consisting of a sequence of nodes.

Each node stores:
- element
- link to the next node
Stack with a Singly Linked List

- We can implement a stack with a singly linked list.
- The top element is stored at the first node of the list.
- The space used is $O(n)$ and each operation of the Stack ADT takes $O(1)$ time.
Queue with a Singly Linked List

- We can implement a queue with a singly linked list
  - The front element is stored at the first node
  - The rear element is stored at the last node
- The space used is $O(n)$ and each operation of the Queue ADT takes $O(1)$ time
Position ADT

The `Position` ADT models the notion of place within a data structure where a single object is stored.

A special `null` position refers to no object.

Positions provide a unified view of diverse ways of storing data, such as:
- a cell of an array
- a node of a linked list

**Member functions:**
- `Object& element()`: returns the element stored at this position
- `bool isNull()`: returns true if this is a null position
List ADT

- The List ADT models a sequence of positions storing arbitrary objects.
- It establishes a before/after relation between positions.
- Generic methods:
  - size(), isEmpty()
- Query methods:
  - isFirst(p), isLast(p)
- Accessor methods:
  - first(), last()
  - before(p), after(p)
- Update methods:
  - replaceElement(p, o), swapElements(p, q)
  - insertBefore(p, o), insertAfter(p, o)
  - insertFirst(o), insertLast(o)
  - remove(p)
Doubly Linked List

- A doubly linked list provides a natural implementation of the List ADT.
- Nodes implement Position and store:
  - element
  - link to the previous node
  - link to the next node
- Special trailer and header nodes
Insertion

We visualize operation `insertAfter(p, X)`, which returns position q.
Deletion

We visualize \texttt{remove}(p), where \( p = \text{last}() \)
Performance

In the implementation of the List ADT by means of a doubly linked list:

- The space used by a list with $n$ elements is $O(n)$
- The space used by each position of the list is $O(1)$
- All the operations of the List ADT run in $O(1)$ time
- Operation `element()` of the Position ADT runs in $O(1)$ time
Sequence ADT

- The Sequence ADT is the union of the Vector and List ADTs.
- Elements accessed by Rank, or Position.
- Generic methods: size(), isEmpty()
- Vector-based methods: elemAtRank(r), replaceAtRank(r, o), insertAtRank(r, o), removeAtRank(r)

- List-based methods: first(), last(), before(p), after(p), replaceElement(p, o), swapElements(p, q), insertBefore(p, o), insertAfter(p, o), insertFirst(o), insertLast(o), remove(p)

- Bridge methods: atRank(r), rankOf(p)
Applications of Sequences

The Sequence ADT is a basic, general-purpose, data structure for storing an ordered collection of elements

Direct applications:
- Generic replacement for stack, queue, vector, or list
- Small database (e.g., address book)

Indirect applications:
- Building block of more complex data structures
Array-based Implementation

- We use a circular array storing positions.
- A position object stores:
  - Element
  - Rank
- Indices $f$ and $l$ keep track of first and last positions.
## Sequence Implementations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Array</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>size, isEmpty</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>atRank, rankOf, elemAtRank</td>
<td>1</td>
<td>n</td>
</tr>
<tr>
<td>first, last, before, after</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>replaceElement, swapElements</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>replaceAtRank</td>
<td>1</td>
<td>n</td>
</tr>
<tr>
<td>insertAtRank, removeAtRank</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>insertFirst, insertLast</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>insertAfter, insertBefore</td>
<td>n</td>
<td>1</td>
</tr>
<tr>
<td>remove</td>
<td>n</td>
<td>1</td>
</tr>
</tbody>
</table>