Elementary Data Structures: Stacks & Queues
Abstract Data Type

• A well defined *set of operations* on a mathematical model

• Example: Set ADT definition

  Insert(x,S)  Intersection(S1,S2)
  Delete(x,S)  Union(S1,S2)
  Member(x,S)
Data Type vs. Data Structure

• *Data Structure* is the way an ADT is implemented.

• Must be distinguished from the ADT itself!
C++ Implementation of Stack

Stack.h is a link to either Stack_list.h or Stack_array.h
Stacks & Queues

- Data Sets with predefined removal policies
- Stack removes from the top
  - Last In First Out - LIFO
- Queue removes from the bottom
  - First In First Out - FIFO

Stack ▶ ▶ ▶

Queue ▶ ▶
Stack Operations

- **Push** *(x)*
  Insert the element x to the top of the stack

- **Pop** ()
  Remove element from the top

- **Top** ()
  Get the next element to be removed (do not remove)

- **IsEmpty** ()
  Check whether the stack is empty
Queue Operations

- **Enqueue**(x)
  Insert the element x to the tail of the queue
- **Dequeue**()
  Remove the first element from the head of the queue
- **Top**()
  Get the next element to be removed (do not remove)
- **IsEmpty**()
  Check whether the queue is empty
Notes

• Practical implementations usually include more operations
• Initialization/Destruction
• “Luxury” operations:
  - Queue::Size()
  - Queue::Append(q2)
  - Queue::Print()
Queue Implementation Using an Array:

A: 7 2 4 3 1 9

insert(x):
A[tail] ← x
tail ← (tail+1) mod N

Remove():
x ← A[head]
head ← (head+1) mod N
return x

IsEmpty():
if tail == head
return TRUE
else
return FALSE

IsFull():
if (tail+1) mod N == head
return TRUE
else
return FALSE
Stack Implementation Using a Linked List:

**Pop()**:  
if IsEmpty()  
    stop  
    tmp ← top  
    top ← top.next  
    tmp2 ← tmp.data  
    delete tmp  
    return tmp2

**Push(x)**:  
tmp ← new LinkedItem  
tmp.data ← x  
tmp.next ← top  
top ← tmp

**IsEmpty()**:  
if top == NIL  
    return TRUE  
else  
    return FALSE
Pseudo Code

• A schematic description of the algorithm which can be easily translated to a “real” programming language

• Algorithms in lectures, recitations and homework will be written in pseudo code

• More information can be found in CLRS book
Pseudo Code – Basic Operators

- “←” – assignment operator
- “for”, “while”, “repeat-until” – loop operators
- “if-else”, “case” – condition operators
- “return” – return from procedure
- Lots of other operators with their meaning obvious from their names, e.g.,
  - “swap”, “delete”, “new”, ’==‘ etc.
Pseudo Code - Conventions

- Indentation indicates block structure
- Variables are local to a given procedure
- $A[i]$ denotes the $i$-th element of the array $A$
- An empty pointer value is denoted by NIL
- Fields (and methods) in the composite objects are accessed by ".",
  - e.g. `Person.id`, `Person.FirstName`, `Person.FamilyName`, etc.
Two-Sided Queue

- `InsHead(x), InsTail(x)`
  Insert to the head/tail of the queue
- `RemHead(), RemTail()`
  Remove element from the head/tail
- `Head(), Tail()`
  Get the head/tail element (do not remove)
- `IsEmpty()`
  Check whether the queue is empty
# Two-Sided Queue Implementation

- Use a simple queue to make a two-sided queue

<table>
<thead>
<tr>
<th>InsHead(x):</th>
<th>InsTail(x):</th>
<th>RemHead():</th>
<th>RemTail():</th>
</tr>
</thead>
<tbody>
<tr>
<td>InsTail(x)</td>
<td>Q.Ins(x)</td>
<td>Q.Rem()</td>
<td>Flip()</td>
</tr>
<tr>
<td>Flip()</td>
<td></td>
<td></td>
<td>RemHead()</td>
</tr>
</tbody>
</table>

Flip()

\[
\text{for } k=1 \text{ to } Q.\text{Size()-1}
\]

\[
Q.\text{Ins(Q.Top())}
\]

\[
Q.\text{Rem()}
\]
Reminder: Mathematical Induction

• Motivation: Proving that a claim holds for all natural n
  – e.g., $1 + 2 + \ldots + n - 1 = \frac{n \cdot (n - 1)}{2}$

• Structure of proof:
  – A clear statement of the inductive claim
  – **Basis**: Proof that claim holds for all required base values, depends on claim
  – **Induction step**:
    • **Weak induction**: Proof that claim is true for $n=k+1$ assuming it holds for all $n=k$
    • **Strong induction**: Proof that claim is true for $n=k+1$ assuming it holds for all $n \leq k$
Reminder: Proof by Contradiction

- Lets assume that we would like to prove a claim, \textit{Claim1}

- Structure of proof:
  - Assume that \textit{Claim1} is incorrect
  - Show that this assumption leads to a contradiction
  - The contradiction should be either to assumptions made in the claim, or to well-known mathematical facts (e.g., \(0=1\))
Reminder: Pigeonhole Principle

**Theorem** If \((m+1)\) pigeons are put into \(m\) pigeonholes, then there is at least one pigeonhole with more than one pigeon

**Proof:**
- Let us denote by \(n_i\) the number of pigeons in pigeonhole \(i\).
- Assume by way of contradiction that all \(m\) pigeonholes contain less than or equal to 1 pigeon, namely, \(n_i \leq 1\) for all \(i = 1 \ldots m\).
- Let TOTAL be the number of pigeons. Then

\[
TOTAL = \sum_{i=1}^{m} n_i \leq \sum_{i=1}^{m} 1 = m.
\]

Contradicting the fact that \(TOTAL = m + 1\) by assumption.

QED
Problems

• Given two stacks, can you implement a queue?

• Given two queues, can you implement a stack?

• What about one stack? One queue?