Introduction to Computers and Programming

Prof. I. K. Lundqvist

Lecture 4
Mar 17 2004

Data Structures*

- Example: Sorting elements
  - Input: a set *S* of numbers
  - Output: elements of *S* in increasing order
  - Algorithm:
    1. Locate smallest item in *S*
    2. Output smallest item
    3. Delete smallest item from *S*
    4. GOTO 1, while $S \neq \varnothing$

Key to a good solution: **data structure for *S***

Topics for next 5 lectures

- Elementary data structures
  - Stacks and Queues
  - Linked lists
  - Graphs
  - Trees
- Today:
  - Stacks and Queues
  - FIFO vs. LIFO
  - Implementations using **arrays**
  - Expression Conversion

Stacks and Queues

- Dynamic sets in which the element removed from the set by the *Delete operation* is prespecified.
- STACK
  - Element deleted is: most recently inserted element
- QUEUE
  - Element deleted is: element that has been in the set the longest

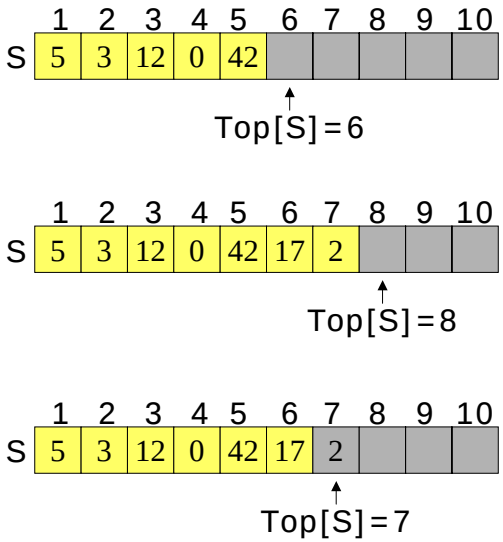
Stack

- **Stack**: A list with *insertion* and *deletion* both take place at **one** end: the top
  - Main operations
    - **Push**
      - New item added on top of stack, size of stack increases by one
    - **Pop**
      - Top-item removed from stack, decreasing stack size by one
  - Other operations
    - Initialize, Empty, Size, Top, Stack_Top, Display, ...
- Implements a **LIFO** policy
  - New **addition** makes older items inaccessible

Stack

- Example use of stacks:
- Implementations used:
  - Arrays or linked lists

Implementing Stack using Array



Empty(*S*)

```
if top[S]=1 then
  return true
else
  return false
```

Push (*S*, *x*)

```
if STACK-FULL(S) then
        error "overflow"
else
        S[top[S]] := x
        top[S] := top[S]+1
```

Pop (*S*)

```
if STACK-EMPTY(S) then
        error "underflow"
else
   top[S] := top[S]-1
   return S[top[S]]
```

my_stack.ads
my_stack.adb
test_stack.adb

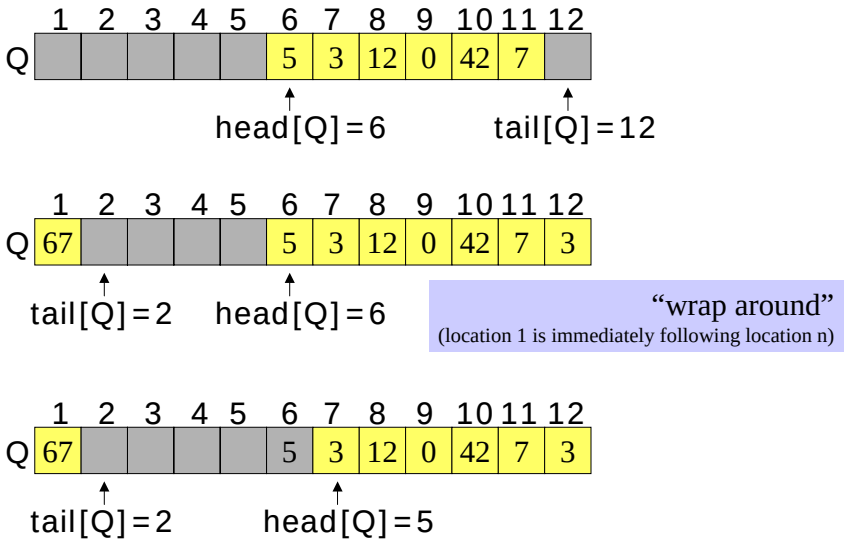
Performance and Limitations of My_Stack

- Performance
  - Let N be the number of elements in the stack
  - The space used is O(N)
  - Each operation used is O(1)
- Limitations
  - The maximum size of the stack must be defined a priori and cannot be changed
  - Trying to push a new element into a full stack causes an implementation-specific exception

Queue

- A list of elements with
  - Insertion: at end of list, tail
  - Deletion: at start of list, head
- Implements a **FIFO** policy
  - Queues are fair when someone has to wait
- Examples:

Implementing a Queue using an Array Q[1..12]



ENQUEUE (*Q*, *x*)

Q[*tail*[*Q*]] := *x*
if *tail*[*Q*] = *length*[*Q*] **then**
  tail[*Q*] := 1
else
  tail[*Q*] := *tail*[*Q*]+1

DEQUEUE (*Q*)

x := *Q*[*head*[*Q*]]
if *head*[*Q*] = *length*[*Q*] **then**
  head[*Q*] := 1
else
  head[*Q*] :=*head*[*Q*]+1
return *x*

Operations on Queues

- Create, Enqueue, Dequeue, Size, Is_Empty, Is_Full, Display

Exercise: Update my_queue to make it a circular queue

my_queue.ads
my_queue.adb
test_queue.adb

Examples Using Stacks

- Infix vs. postfix
  - How to evaluate postfix
  - How to evaluate infix
- Convert Infix to Postfix

my_expression_converter.ads
my_expression_converter.adb
converter_test.adb
[**use these for this weeks PSET**]

Infix vs. Postfix

Infix Expressions	Corresponding Postfix
5 + 3 + 4 + 1	5 3 + 4 + 1 +
(5 + 3) * 10	5 3 + 10 *
(5 + 3) * (10 – 4)	5 3 10 4 - *
5 * 3 / (7 – 8)	5 3 * 7 8 - /
(b * b – 4 * a * c) / (2 * a)	b b * 4 a * c * - 2 a * /

How to Evaluate Postfix

A program can evaluate postfix expressions by reading the expression from left to right

```
for I in 1 .. length loop
    If Is_Number(expr(I)) = true then
          push expr(I)
    If Is_Operator(expr(I)) then
          pop two numbers from the stack
          perform operation
          push result onto stack
end loop
--  result is on top of stack
```

How is a bad postfix expression indicated?

Evaluating infix expressions

- Need two stacks, one for numbers and one for operators

```
for I in 1 .. length loop
        If Is_Number(expr(I)) = true then
          push expr(I) onto operand_stack
        If Is_Operator(expr(I)) then
          push expr(I) onto operator_stack
        If expr(I) = ')' then
          pop 2 numbers from operand_stack
          pop an operator from the operator_stack
          perform operation
          push the result onto the operand_stack
end loop
-- The top of stack contains the result.
```

Infix to Postfix: Example

- Infix Expression

  3 + 5 * 6 – 7 * (8 + 5)

- Postfix Expression

  3 5 6 * + 7 8 5 + * –

Infix to Postfix

```
post_fix := ""
Create(Op_Stack)
for I in 1 .. Length loop
   If Is_Operand(expr(I)) = true then
         Append(post_fix, expr(I))

   If Is_Operator(expr(I)) = true then
         Process_Next_Operator(expr(I))
end loop
-- string post_fix has the result
```

Process_Next_Operator

```
Done := False
loop
     If Is_Empty(Op_Stack) or next_op is '(',
        push next_op onto Op_Stack
        set Done to True
     Elsif precedence(next_op) > precedence(top_operator)
        Push next_op onto Op_stack
        -- ensures higher precedence operators evaluated first
        Set Done to True
     Else
         Pop the operator_stack
         If operator popped is '('
                set Done to True
         Else
                append operator popped to post_fix string
     exit when Done = True
end loop
```