Data Structures and Algorithms

(CS210A)

Semester I – **2014-15**

Lecture 10:

- **Arithmetic expression evaluation: Complete algorithm using stack**
- **Two interesting problems**

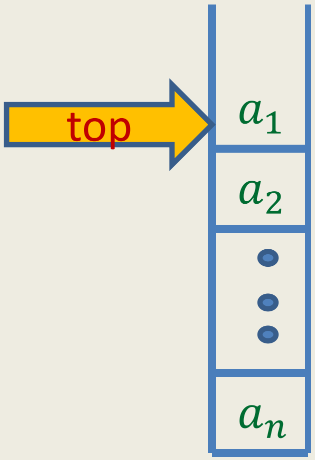
Quick Recap of last lecture

Stack: a new data structure

A <u>special kind</u> of list

where all operations (insertion, deletion, query) take place at <u>one end</u> only,

called the **top**.



Evaluation of an arithmetic expression

Question: How does a computer/calculator evaluate an arithmetic expression given in the form of a string of symbols ?

$$8 + 3 * 5 \wedge 2 - 9$$

Evaluation of an arithmetic expression

Question: How does a computer/calculator evaluate an arithmetic expression given in the form of a string of symbols?

operators

$$8 + 3 * 5 \wedge 2 - 9$$

operands

- What about expressions involving parentheses: 3+4*(5-6/(8+9^2)+33) ?
- What about associativity of the operators ?

Overview of our solution

1. **Focusing on a simpler version of the problem:**
    1. Expressions without parentheses
    2. Every operator is left associative

2. **Solving the simpler version**

3. **Transforming the solution of simpler version to generic**

Incorporating precedence of operators through priority number

Operator	Priority
+ , -	1
* , /	2
^	3

Insight into the problem

Let o_i : the operator at position *i* in the expression.

Aim: To determine an order in which to execute the operators.

8 **+** 3 ***** 5 **^** 2 **–** 9 ***** 67

Position of an operator **does** matter

Question: Under what conditions can we execute operator o_i immediately?

Answer: if

- priority(o_i) > priority(o_{i-1})
- priority(o_i) ≥ priority(o_{i+1})

Expression: $n_1 o_1\ n_2 o_2\ n_3\ o_3\ n_4\ o_4\ n_5\ o_5\ n_6\ o_6 \ldots$

We keep two stacks:

N-stack
for operands

O-stack
for operators

Expression: $n_1\, o_1\; n_2\, o_2\; n_3\; o_3\; n_4\; o_4\; n_5\; o_5\; n_6\; o_6\; \ldots$

We keep two stacks:

N-stack
for operands

O-stack
for operators

Expression: $n_1\, o_1\, n_2\, o_2\, n_3\, o_3\, n_4\, o_4 \quad n' \quad o_6 \ldots$

We keep two stacks:

n'
n_4
n_3
n_2
n_1

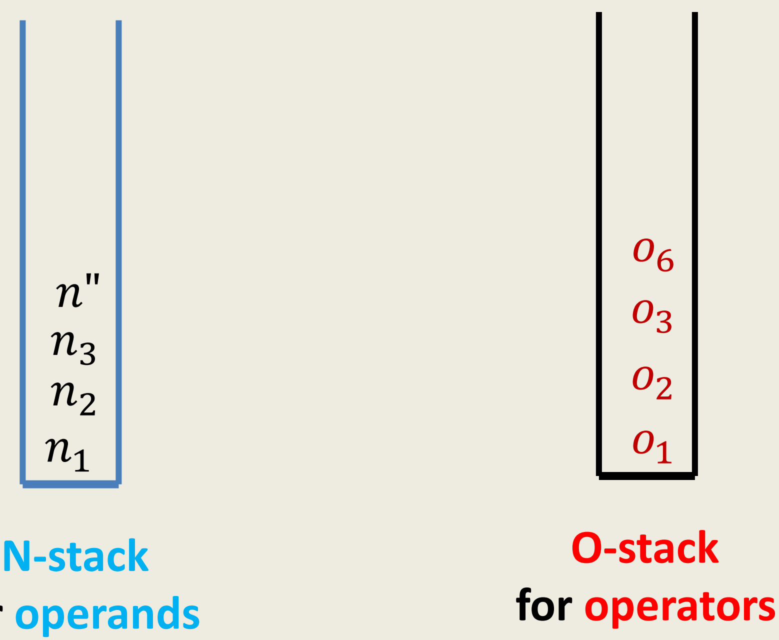
N-stack
for operands

o_4
o_3
o_2
o_1

O-stack
for operators

Expression: $n_1 o_1 \; n_2 o_2 \; n_3 \; o_3 \qquad n'' \qquad o_6$ …

We keep two stacks:



A **simple** algorithm

```
While ( ? ) do
{    x ← next_token();
     Two cases:
         x is number   :  push(x,N-stack);
         x is operator :
                          while( PRIORITY(TOP(O-stack)) >=   PRIORITY(x)    )
                          { o ← POP(O-stack);
                            Execute(o);
                          }
                          push(x,O-stack);
}
```

- **POP** two numbers from **N-stack** and apply operator **x** on them
- place the result back into **N-stack**

Next step

Transforming the solution to Solve the most general case

How to handle parentheses ?

$\boxed{3+4*}(5 - 6/2)$

Question: What do we do whenever we encounter (in the expression ?

Answer:

Evaluate the expression enclosed by this parenthesis

before any other operator currently present in the O-stack.

➔ So we must push (into the O-stack.

Observation 1: While (is the **current operator** encountered in the expression, it must have **higher priority** than every other operator in the stack

How to handle parentheses ?

$\boxed{3+4*}(5 - 6/2)$

Question: What needs to be done when (is at the top of the O-stack ?

Answer:

The (at the top of the stack should act as an *artificial bottom* of the O-stack .

➔ every other operator that follows (should be allowed to sit on the top of (in the stack .

Observation 2 : while (**is inside the stack**, it must have **less priority** than every other operator that follows.

A CONTRADICTION !!

Observation 1: While (is the **current operator** encountered in the expression,
it must have **higher priority** than every other operator in the stack

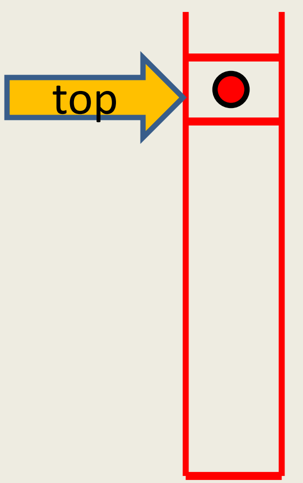
Take a pause for a few minutes to realize surprisingly that the contradicting requirements for the priority of (in fact hints at a suitable solution for handling (.

How to handle parentheses ?

Using two **types** of priorities of each operator ●.

InsideStack priority

The priority of an operator ● when it is inside the stack.

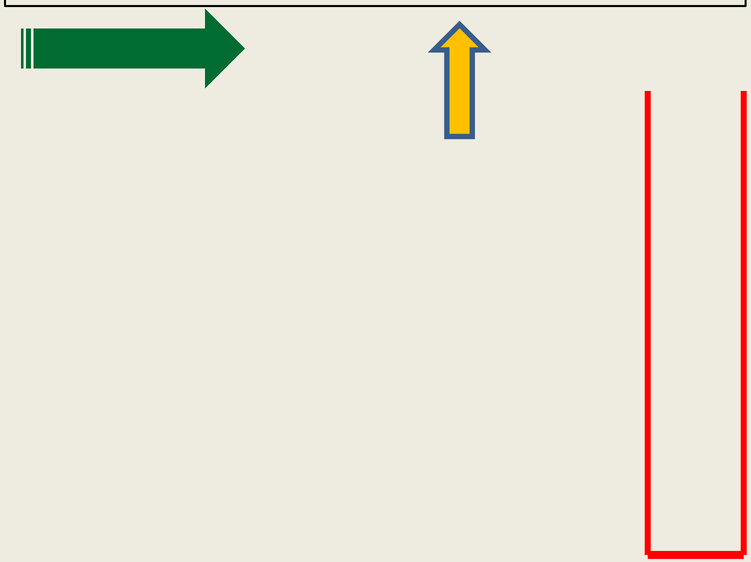


O-stack

OutsideStack priority

The priority of an operator ● when it is encountered in the expression.

$n_1 o_1\ n_2 o_2\ n_3\ o_3\ n_4$ ● $n_5\ o_5\ n_6\ o_6$



O-stack

How to handle parentheses ?

Using two **types** of priorities of each operator.

Operator	InsideStackPriority	OutsideStackPriority
+ , -	1	1
* , /	2	2
^	3	3
(	0	4

Does it take care of nested parentheses ? Check it yourself.

How to handle parentheses ?

$\boxed{3+4*}(5 - 6/2)$

Question: What needs to be done whenever we encounter) in the expression ?

Answer: Keep **popping** O-stack and evaluating the operators until we get its matching (.

The algorithm generalized to handle **parentheses**

```
While (    ?    ) do
  x ← next_token();
  Cases:
    x is number :      push(x,N-stack);
    x is )      :      while(    TOP(O-stack) <> (    )
                       {    o ← Pop(O-stack);
                            Execute(o);
                       }
                       Pop(O-stack);          //popping the matching (
    otherwise   :      while(InsideStackPriority(TOP(O-stack)) >= OutsideStackPriority(x))
                       {    o ← Pop(O-stack);
                            Execute(o);
                       }
                       Push(x,O-stack);
```

Practice exercise

Execute the algorithm on 3+4*((5+6*(3+4)))^2 and convince yourself through proper reasoning that the algorithm handles parentheses suitably.

How to handle associativity of operators ?

Associativity of arithmetic operators

Left associative operators : +, - , * , /

- a+b+c = (a+b)+c
- a-b-c = (a-b)-c
- a*b*c = (a*b)*c
- a/b/c = (a/b)/c

We have already handled left associativity in our algorithm.

Right associative operators: ^

- 2^3^2 = 2^(3^2) = 512.

How to handle right associativity ?

What we need is the following:

If ^ is **current operator** of the expression, and ^ is on **top of stack**, then ^ should be evaluated before ^.

How to incorporate it ? Play with the **priorities** ☺

How to handle associativity of operators ?

Using two **types** of priorities of each **right associative** operator.

Operator	InsideStackPriority	Outside-stack priority
+ , -	1	1
* , /	2	2
^	3	4
(	0	5

The **general** Algorithm

It is the same as the algorithm to handle parentheses :-)

```
While (  ?  ) do
  x ← next_token();
  Cases:
    x is number :     push(x,N-stack);
    x is )      :     while(    TOP(O-stack) <> (    )
                      {    o ← Pop(O-stack);
                           Execute(o);
                       }
                       Pop(O-stack);      //popping the matching (
    otherwise   :      while(InsideStackPriority(TOP(O-stack)) >= OutsideStackPriority(x))
                       {    o ← Pop(O-stack);
                            Execute(o);
                        }
                       Push(x,O-stack);
```

Homeworks

1. Execute the general algorithm on 3+4*((4+6)^2)/2 and convince yourself through proper reasoning that the algorithm handles nested parentheses suitably.
2. Execute the general algorithm on 3+4^2^2*3 and convince yourself through proper reasoning that the algorithm takes into account the right associativity of operator ^.
3. Our simple (as well as general) algorithm does not consider the case when the O-stack is empty. How can you take care of this small technical thing without changing the **while** loop of the algorithm ?

Hint: Introduce a new operator $ with both its priorities -1 and push it into **O-stack** before the **while** loop.

4. How to take care of the end of the expression ?

Hint: Introduce a new operator symbol # so that upon seeing #, we do very much like what we do on seeing **)**.

Proof of correctness of iterative algorithms

- Computing **sum of first n** positive integers.
- Computing **maximum-sum subarray**.
- **Local Minima** in an array.
- **Binary search**

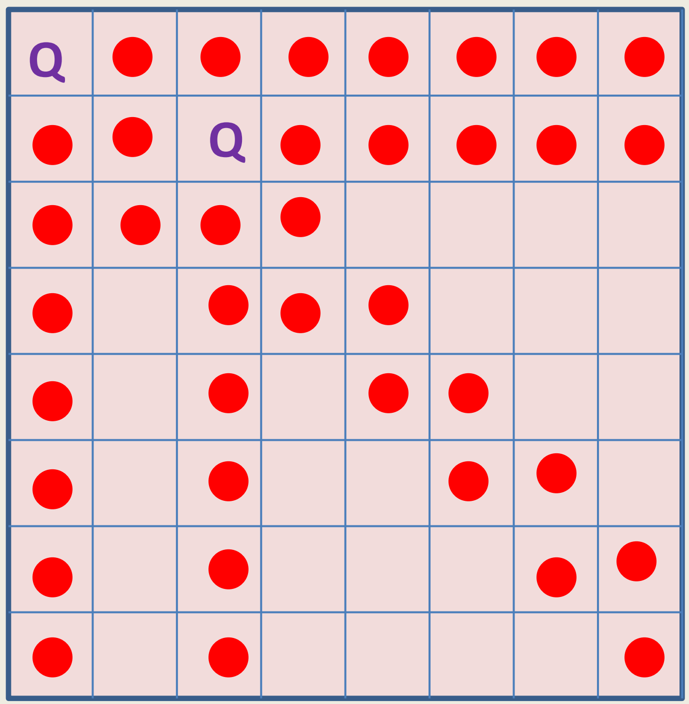
Fully internalize these proofs.

Two interesting problems

Applications of simple data structures

8 queen problem

Place 8 queens on a chess board so that no two of them attack each other.

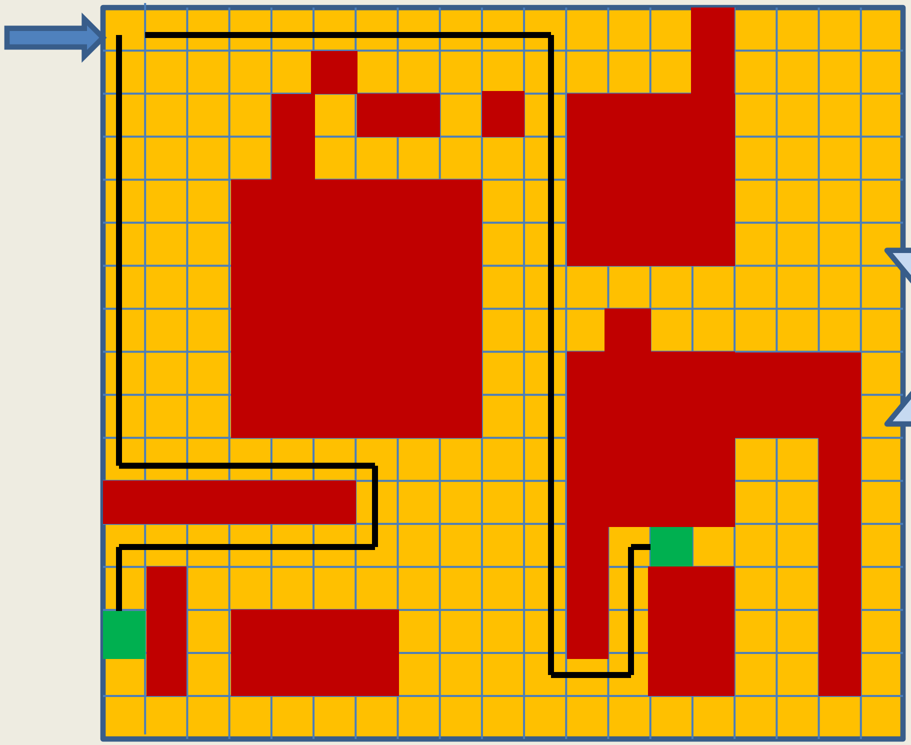


With this sketch/hint, try to design the complete algorithm using stack or otherwise.

Shortest route in a grid

From a cell in the grid, we can move to any of its neighboring cell in one step.

From top left corner, **find shortest route** to each green cell avoiding obstacles.



Ponder over this beautiful problem ☺