1.1. Transform a given arithmetic expression in its postfix form.

operators: + , − , * , / and parenthesis

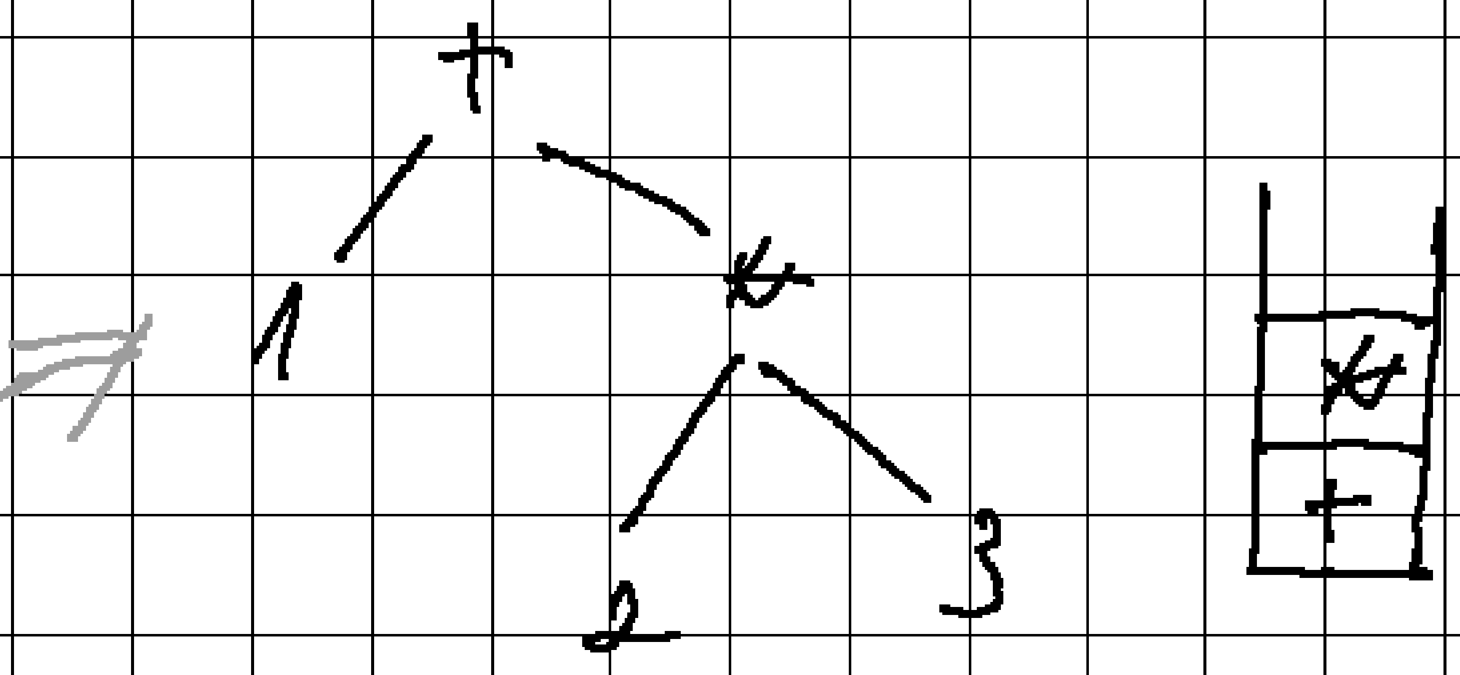
1.2. Evaluate an arith. expr. given in its postfix form

operation

1 + 2      1 2 +

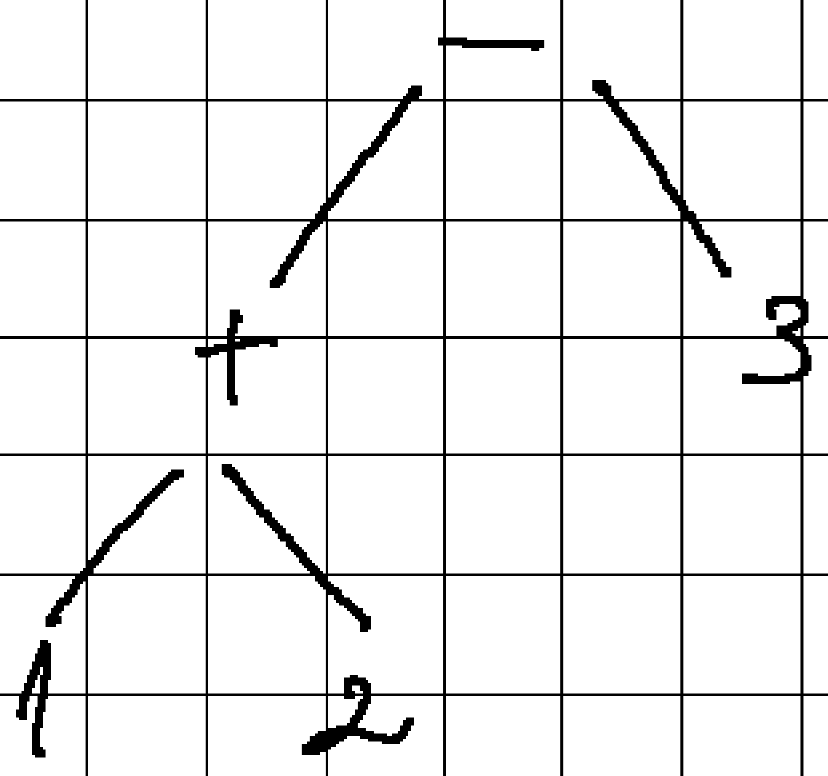
operands

| | |
|---|---|
| 1+2+3 | 12+3+ |
| 1+2−3 | 12+3− |
| 1+2*3 | 123*+ |
| 1*2+3 | 12*3+ |



1+2−3

| Curr. symb | postfix | Stack |
|---|---|---|
| 1 | 1 | |
| + | 1 | + |
| 2 | 1,2 | + |
| — | 1,2,+ | — |
| 3 | 1,2,+,3 | — |
| | 1,2,+,3,− | |

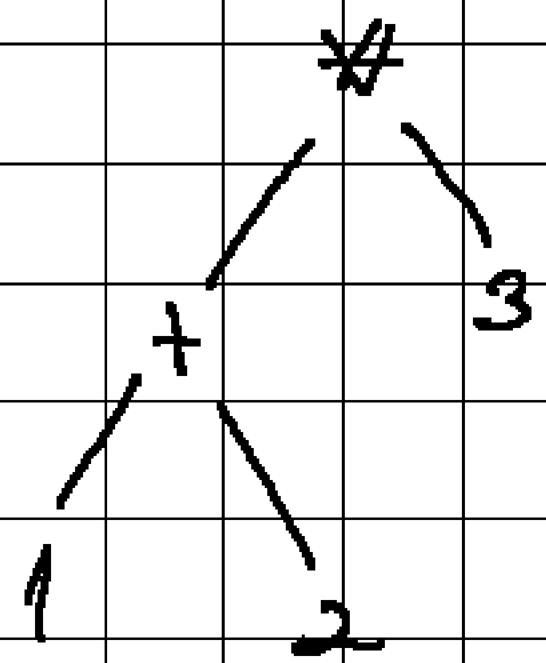


$1 * 2 + 3 * 4$

| Curr. Symb | postfix | stack |
|---|---|---|
| 1 | 1 | |
| * | 1 | * |
| 2 | 1,2 | * |
| + | 1,2,* | + |
| 3 | 1,2,*,3 | + |
| * | 1,2,*,3, | +,* |
| 4 | 1,2,*,3,4 | +,* |
| | 1,2,*,3,4,*,+ | |

$(1+2) * 3$

| Curr. Symb | postfix | stack |
|---|---|---|
| ( | | ( |
| 1 | 1 | ( |
| + | 1 | (,+ |
| 2 | 1,2 | (,+ |
| ) | 1,2,+ | |
| * | 1,2,+ | * |
| 3 | 1,2,+,3 | * |
| | 1,2,+,3,* | |



(1+2) * (3+4) - 5

| C. symb | postfix | stack |
|---|---|---|
| ( | | ( |
| 1 | 1 | ( |
| + | 1 | (, + |
| 2 | 1, 2 | (, + |
| ) | 1, 2, + | |
| * | 1, 2, + | * |
| ( | 1, 2, + | *, ( |
| 3 | 1, 2, +, 3 | *, ( |
| + | 1, 2, +, 3 | *, (, + |
| 4 | 1, 2, +, 3, 4 | *, (, + |
| ) | 1, 2, +, 3, 4, + | * |
| - | 1, 2, +, 3, 4, +, * | - |
| 5 | 1, 2, +, 3, 4, +, *, 5 | - |
| | 1, 2, +, 3, 4, +, *, 5, - | |

```
st:     Init (st)
        isEmpty (st)
        push (st, e)
        pop (st) => e
        top (st) => e
  isOperator (e)
  isOperand                         Compute (O1 , op , O2)
                                            (ex: 1    +    2 )
  priority (e1)
                 > >, =, <...
  priority (e2)

Function   transform ( expression)
        init (st)
        init (q)
        for each e in expression execute:
            if isOperand (e) then
                push (q, e)
            else if e == '(' then
                push (st, e)
            else if e == ')' then
                el = pop(st)
                while el != '(' execute
                    push (q, el)
            el = pop(st)
         endwhile
      else   // operators +,-,*,/
         if isEmpty(st) then
               push(st,e)
         else
            while not isEmpty(st) and
                  top(st) != '(' and
                  priority(top(st), priority(e))
               push(q, pop(st))
            push(st,e)
      endif
   endfor
   while not isEmpty(st) execute
      push(q, pop(st))
   endwhile
   transform ← q      // return q
endfunction
```

1, 2, 3, *, +

| C.S. | stack |
|---|---|
| 1 | 1 |
| 2 | 1,2 |
| 3 | 1,2,3 |
| * | 1,6 |
| + | 7 |

1, 2, *, 3, +

| C.S | stack |
|---|---|
| 1 | 1 |
| 2 | 1,2 |
| * | 2 |
| 3 | 2,3 |
| + | 5 |

```
Function evaluate(postfix):
    init(st)
    While not isEmpty(postfix) execute
        e ← pop(postfix)
        if isOperand(e) then
            push(st, e)
        else
            o1 ← pop(st)
            o2 ← pop(st)
            rez ← compute(o2, e, o1)
            push(st, rez)
        end if
    end while
```

3 2 * 1 −

| C.S | stack |
|---|---|
| 3 | 3 |
| 2 | 3,2 |
| * | 6 |
| 1 | 6,1 |
| − | 5 |

Given an array with distinct numbers. Compute the max. sum of K elem. from the array.

ex: 100, 50, 95, $\underline{200}$, $\underline{150}$

$n=5$, $k=2$

V1: find max k times $\Theta(k * n)$

V2: sort + sum the last K $(K \leq n)$

$\uparrow$ $\Theta(n \log n)$

partial sort $\Theta(k * n)$

V3: MAX heap

V4: MIN heap $O(n \log k)$

V3.1: $n$ * add (toate elem) max K
$O(n \log n + k \log n)$

V3.2: use buildHeap
$O(n + k \log n)$

init(h)
add(h, e)
remove(h) $\Rightarrow$ e
top(h)
buildHeap $O(n)$

100, 50, 70, 40, 30

k = 3 MiN heap