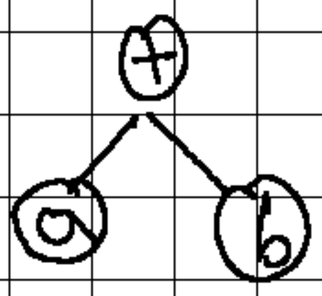
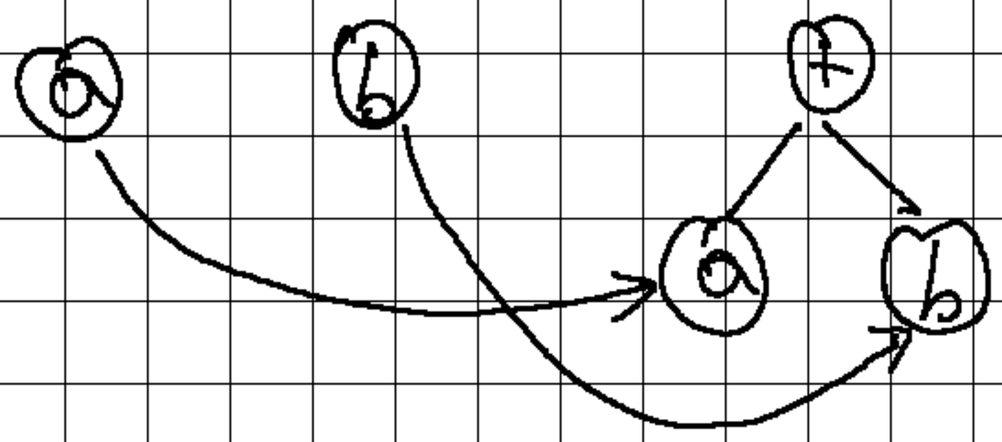
a+b



ab+



isOperand

isOperator

stack

init

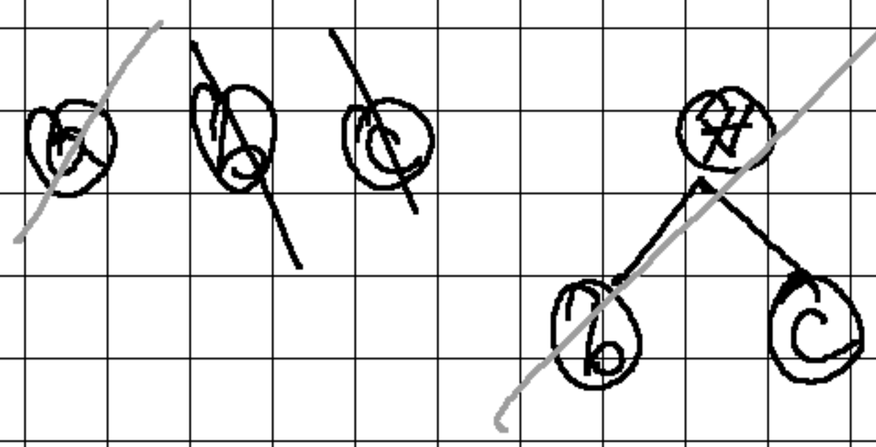
push

pop

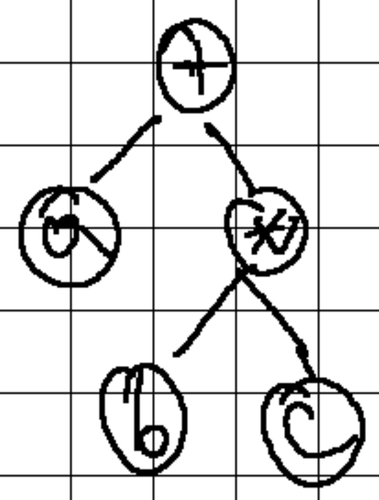
top

isEmpty

a+b*c

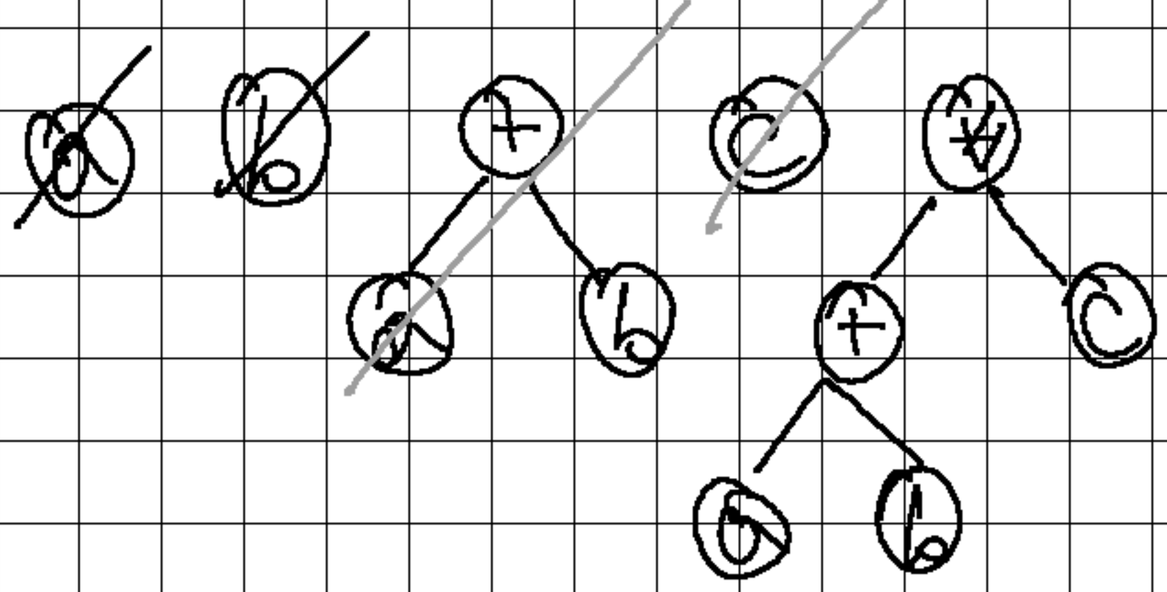


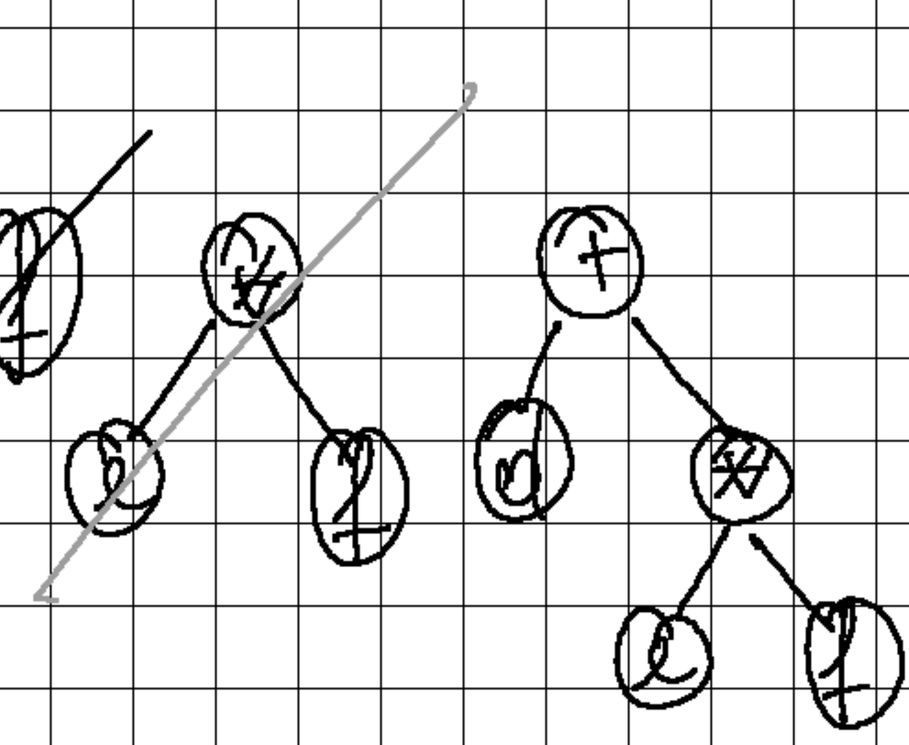
abc*+



(a+b)*c-(d+e*f)+g

ab+c*def*+-g+





Node:

l: TElem

left: ↑Node

right: ↑Node

BinTree

root: ↑Node

Subalg buildTree (postfE, tree)

PRE:
postfE is not empty
is correct

init(st)

for each e in postfE execute:

if isOperand(e) then

allocate(p), [p].left ← NIL

[p].e ← e [p].right ← NIL

st.push(p) // push(st, e)

else

allocate(p)

pop(st, p1)

pop(st, p2)

[p].left ← p2

[p].right ← p1

[p].e ← e

push(st, p)

endif

endfor

true.root ← pop(st)

bnd_subtrg. LFT

PRE-ORDER:

6, 10, 3, 4, 17, 25, 8, 11, 21, 28, 59, 33

IN-ORDER

3, 10, 4, 6, 25, 17, 11, 28, 21, 33, 59

POST-ORDER

3, 4, 10, 25, 28, 33, 59, 21, 11, 8, 17, 6

```
postorder(node)
    if  node is NILL
            return
    postorder(node.left)
     postorder(node.right)
    process
```

```
-Subalg traverse_inlevels (tree)
    init(S)
    if tree.root ≠ Nil then
        push(st, tree.root)
    endif
    while NOT isEmpty(S) ex.
        p ← pop(S)
        if [p].left ≠ Nil then
            push(s, [p].left)
        endif
        if [p].right ≠ Nil then
            push(s, [p].right)
        endif
    endwhile
endsubalg
```

```
Subalg   pfemales(tree):
    if  tree.root ≠ NIL then
            print([tree.root].k)
            pfemales_rec(tree.root)
    endif
endsubalg
```

```
Subalg   pfemales_rec(pNode)
  if   pNode ≠ NIL then
        pfemales_rec([pNode].left)
        if  pNode.left ≠ NIL then
              print([pNode.left].k)
              pfemales_rec([pNode].right)
  endif
endSubalg
```

```
Subalg ancestor_rec ( pNode , k);
    if pNode ≠ Nil then
        if k=0 then
            print ([pNode].e)
        else    //k>0
```

Subalg ancestors (tree , k):
    ancestor-rec (tree.root ,k)

Pagina

- init
- push-back
- pop-back
- back
- push-front
- pop-front
- front
- isEmpty

minLeft
maxRight

```
Subalg  print-TopView (tree)
        init (dq)      // < pNode, level, dist >
        init (topView) // < pNode, level >
        if tree.root ≠ Nil then:
            push-back (dq, <tree.root, 0, 0>)
            push-back (topView, <tree.root, 0>)
            minLeft ← 0
            maxRight ← 0
            while NOT isEmpty (dq) lx:
                < pNode, level, dist > ← pop-front (dq)
                if [pNode].left ≠ Nil then
                    push-back (dq, <[pNode].left, level+1, dist-1 >)
                    if dist-1 < minleft then
                        push-front (topView, <[pNode].left, level+1 >)
                        minLeft ← dist-1
                    endif
                endif
                if [pNode].right ≠ Nil then
                    push-front (dq, <[pNode].right, level+1, dist+1 >)
                    if dist+1 > maxRight then
                        push-back (topView, <pNode.right, level+1 >)
                        maxRight ← dist+1
```

```
    else if dist+1 == maxRight then
        <p, l> ← back (TopView)
        if l == level+1 then
            pop_back(topView)
            push_back(topView, <tNode].right, level+1>)
        endif
    endif
endwhile
```