Binary tree traversal

```
Subalg. onLevels(T)
    initEmpty (q)
    if not isEmpty (T) then
        enqueue(q, rootPos (T))
    endif
    while (not isEmpty(q)) do
        p := dequeue(q)
        @ process p^.info
        if (p^.left <> NIL) then
            enqueue( q, p^.left)
        endif
        if ( p^.right <> NIL) then
            dequeue( q, p^.right)
        endif
    endWhile
endOnLevels
```

Binary tree traversal

Pre-order:

root, left-subtree, right-subtree

1. Visit the node.
2. Traverse the left subtree.
3. Traverse the right subtree.

In-order:

left-subtree, root, right-subtree

1. Traverse the left subtree.
2. Visit the node.
3. Traverse the right subtree.

Post-order:

left-subtree, right-subtree, root

1. Traverse the left subtree.
2. Traverse the right subtree.
3. Visit the node.

Binary tree parsing

Preorder   ➜ use stack

  Subalg. preorder(p)

    if p<>NIL then

       @ process p^.info

       preorder(p^.left)

       preorder(p^.right)

    endif

  endPreorder

```
Subalg. iterativePreorder(T) .
    initEmpty(s)
    if (not isEmpty(T)) then
         push(s,rootPos(T))
    endif
    while not isEmpty(s) do
         p := pop(s)
          @process p^.info
         if p^.right <>NIL then    push(s, p^.right)    endif
         if p^.left   <>NIL then    push(s, p^.left)      endif
    endwhile
end_iterativePreorder
```

Subalg. iterativeInorder(T)

■

```
initEmpty (s)
p := rootPos(T)
while not ( isEmpty(s) and p=NIL ) do
        while (p <> NIL) do
                push(s,p)
                p := p^.left
        endwhile
        p := pop(s)
        @ process p^.info
        p:=p^.right
endWhile
```

end_iterativeInorder

Subalg. iterativeinorder(T)

```
    initEmpty (s)
    p := rootPos(T)
    while not ( isEmpty(s))
        and p=NIL ) do
        while (p <> NIL) do
                push(s,p)
                p := p^.left
        endwhile
        p := pop(s)
        @ process p^.info
        p:=p^.right
    endWhile

end_iterativeInorder
```

Subalg. iterativeinorder2(T)

```
▪   initEmpty (s)
    p := rootPos(T)
    while not ( isEmpty(s)) and
        p=NIL ) do
        if (p <> NIL) then
                push(s,p)
                p := p^.left
        else
                p := pop(s)
                @ process p^.info
                p:=p^.right
        endif
    endWhile

end_iterativeInorder2
```

```
Subalg. iterativePostorder(T)                    //use a flag: left subtree visited

      initEmpty(s)
      p:= rootPos(T)                             ▪
      while not ( isEmpty(s) and p=NIL) do
            while p<>NIL do                       //Traverse downwards to left
                  push(s,[p,0])
                  p:=p^.left
            endwhile
            [p,k]:=pop(s)
            if k=0 then                            //upwards from left
                  push(s,[p,1])
                  p:=p^.right
            else                                   //  upwards from right
                   @ process p^.info
                  p := NIL
            endif
      endwhile

End_iterativePostorder
```

Subalg. iterativePostorder2(T)   // version 2

```
initEmpty(s)
pNode = NIL                                 ▪
cNode = rootPos(T)
while not ( isEmpty(s) and cNode = NIL)
   if  parent(pNode,cNode) then     //Traverse downwards
        if (cNode^.left <>NIL) then          //More traversing to do
                push(s,cNode)
                pNode = cNode
                cNode = cNode^.left
        else if (cNode^.right <>NIL) then
                        push(s,cNode)
                        pNode = cNode
                        cNode = cNode^.right
                else      //cNode does not have descendants: go upward
                        @ process cNode^.info
                        pNode = cNode
                        cNode = peek(s)
                endif
        endif
```

Subalg. iterativePostorder2(T)   // version 2

```
initEmpty(s)                                   ▪
pNode = NIL
cNode = rootPos(T)
while not ( isEmpty(s) and cNode = NIL)
  if  parent(pNode,cNode) then     //Traverse downwards
        if (cNode^.left <>NIL) then         //More traversing to do
                push(s,cNode)
                pNode = cNode
                cNode = cNode^.left
        else if (cNode^.right <>NIL) then
                        push(s,cNode)
                        pNode = cNode
                        cNode = cNode^.right
              else        //cNode does not have descendants: go upward
                        @ process cNode^.info
                        pNode = cNode
                        cNode = peek(s)
              endif
        endif
```

```
    else   //cNode is parent of pNode; traverse upwards
        if ( cNode^.left = pNode) and (cNode^.right <>NIL) thens
                //upwards from left;
                //go right if possible; then downwards
                pNode = cNode
                cNode = cNode^.right
        else        //  upwards from right; process & go upward
                @process cNode
                pop(s)              //!!!
                pNode = cNode
                cNode = peek(s)
        endif
    endif
 endwhile


end_iterativePostOrder2
```

Remark:
Define peek to return NIL if stack is empty

Binary tree examples