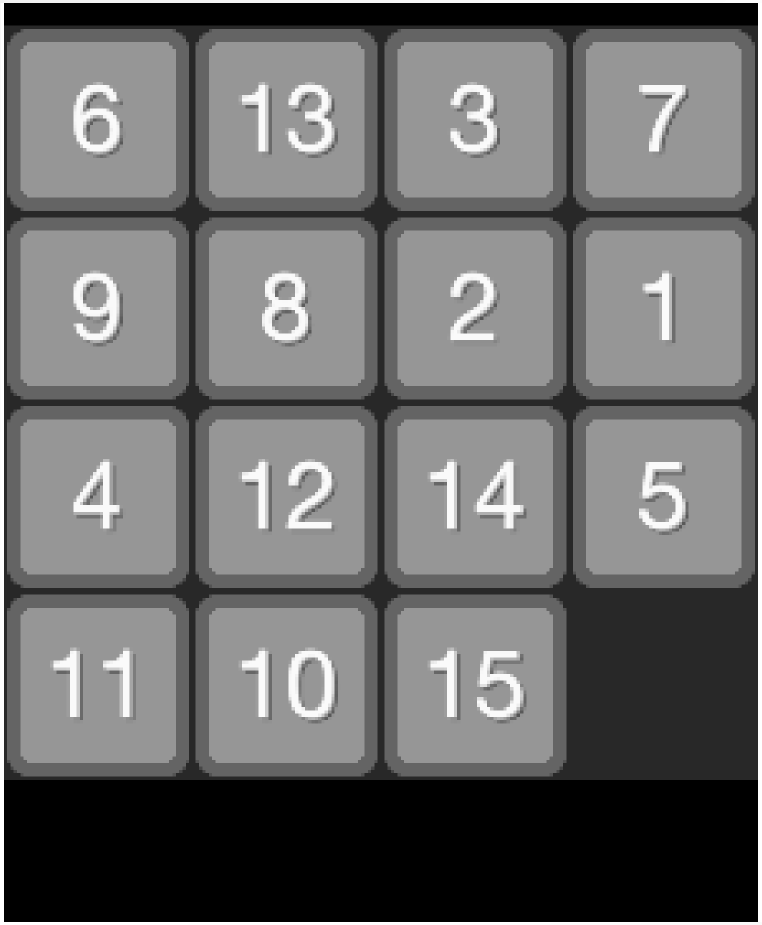
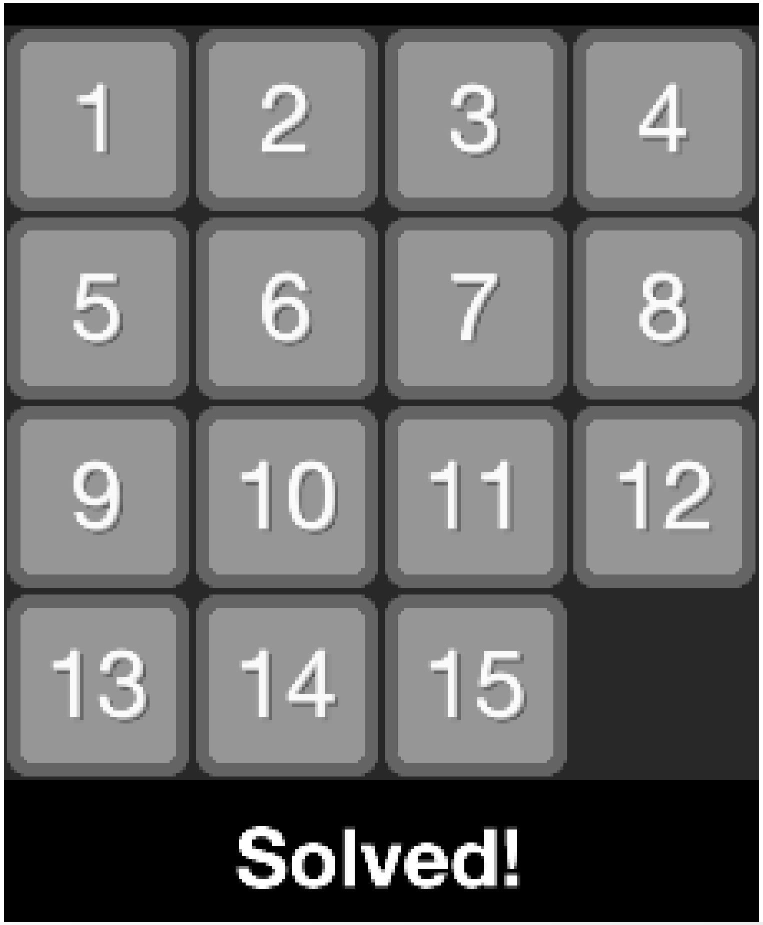
Game 15

- Fifteen puzzle: 4x4 tiles



6	13	3	7
9	8	2	1
4	12	14	5
11	10	15	



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Solved!

```
Subalg. searchSpace( initialConfig )
    reachedConfig ← { initialConfig }
    unExpandedConfig ← { initialConfig }

    while unExpandedConfig ≠ ϕ do
          config := extractOne (unExpandedConfig)
          @for any valid successor succ of config do
                    if succ ∉ reachedConfig then
                              if isFinal(succ) then
                                        // … !! process solution
                              endif
                              reachedConfig ← {succ}∪ reachedConfig
                              unExpandedConfig ← {succ} ∪ unExpandedConfig
                    endif
          endfor
    endWhile
End_searchSpace
```

Robot in a maze (1)

Consider a maze (rectangular shape) with occupied cells (X) and free cells (*). Consider a robot (R) in this maze, and a goal position in this maze.

(a) Verify if the robot can reach the goal position.

(b) Determine a path (if exists).

(c) Determine the shortest path (if exists).

X	*	X	X	*	*	*
*	X	*	*	X	*	*
G	*	*	*	*	*	*
*	X	*	S	*	*	X
*	X	*	*	*	*	X
*	X	*	*	X	*	*
*	X	*	X	*	*	*

Robot in a maze (2)

Consider a maze (rectangular shape) with occupied cells (X) and free cells (*). Consider a robot (R) in this maze

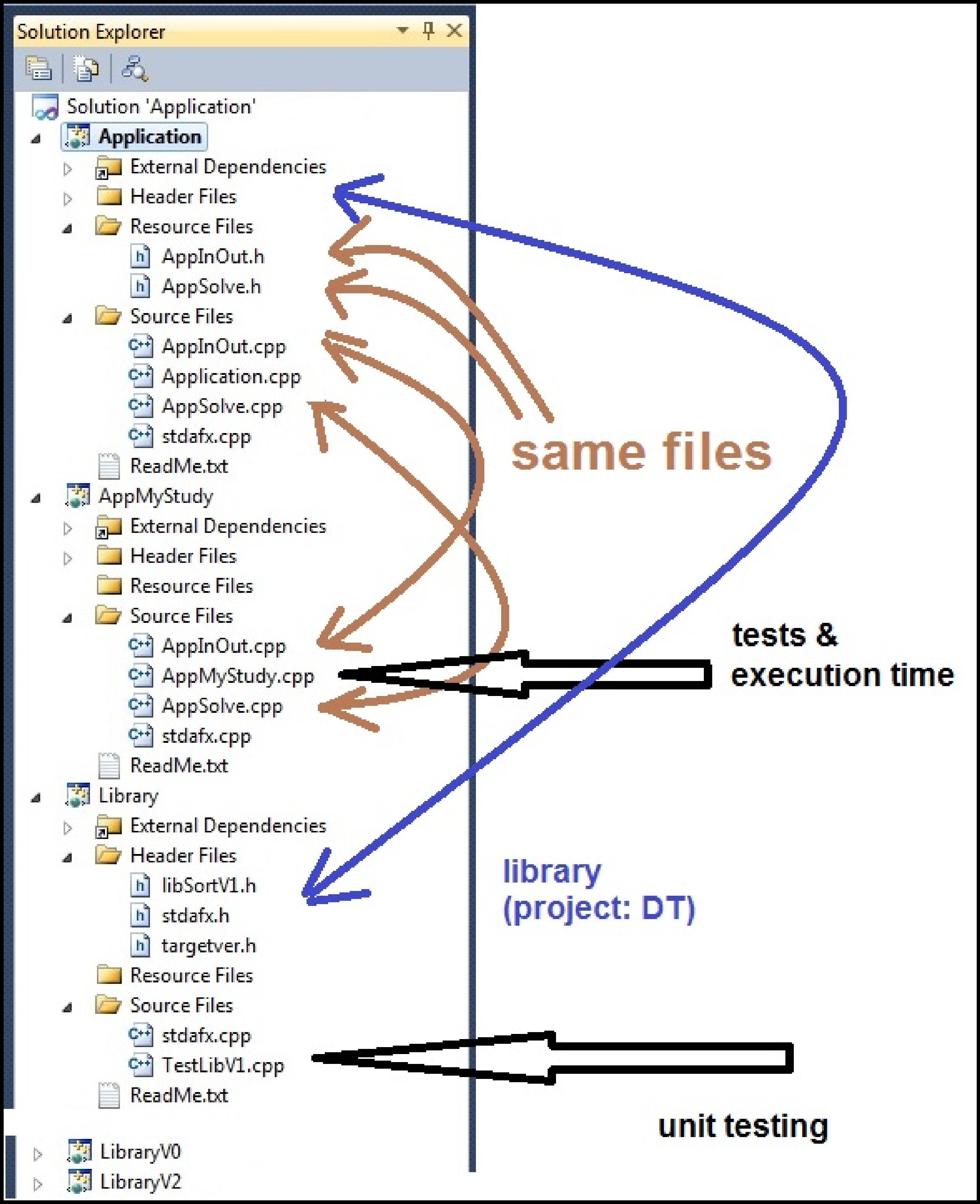
(a) Verify if the robot can get out of the maze (can reach any of the margins).

(b) Determine a path (if exists) to get out of the maze.

(c) Determine the shortest path (if exists) to get out of the maze.

X	X	X	X	X	X	X
X	S	*	*	*	*	*
X	X	X	*	X	X	X
*	*	*	*	X	X	X
X	X	X	X	X	X	X
X	X	X	X	X	X	X
X	X	X	X	X	X	X

Sketch: possible project structure



```
Solution Explorer
Solution 'Application'
  Application
    External Dependencies
    Header Files
    Resource Files
      AppInOut.h
      AppSolve.h
    Source Files
      AppInOut.cpp
      Application.cpp
      AppSolve.cpp
      stdafx.cpp
    ReadMe.txt
  AppMyStudy
    External Dependencies
    Header Files
    Resource Files
    Source Files
      AppInOut.cpp
      AppMyStudy.cpp
      AppSolve.cpp
      stdafx.cpp
    ReadMe.txt
  Library
    External Dependencies
    Header Files
      libSortV1.h
      stdafx.h
      targetver.h
    Resource Files
    Source Files
      stdafx.cpp
      TestLibV1.cpp
    ReadMe.txt
  LibraryV0
  LibraryV2
```

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
```

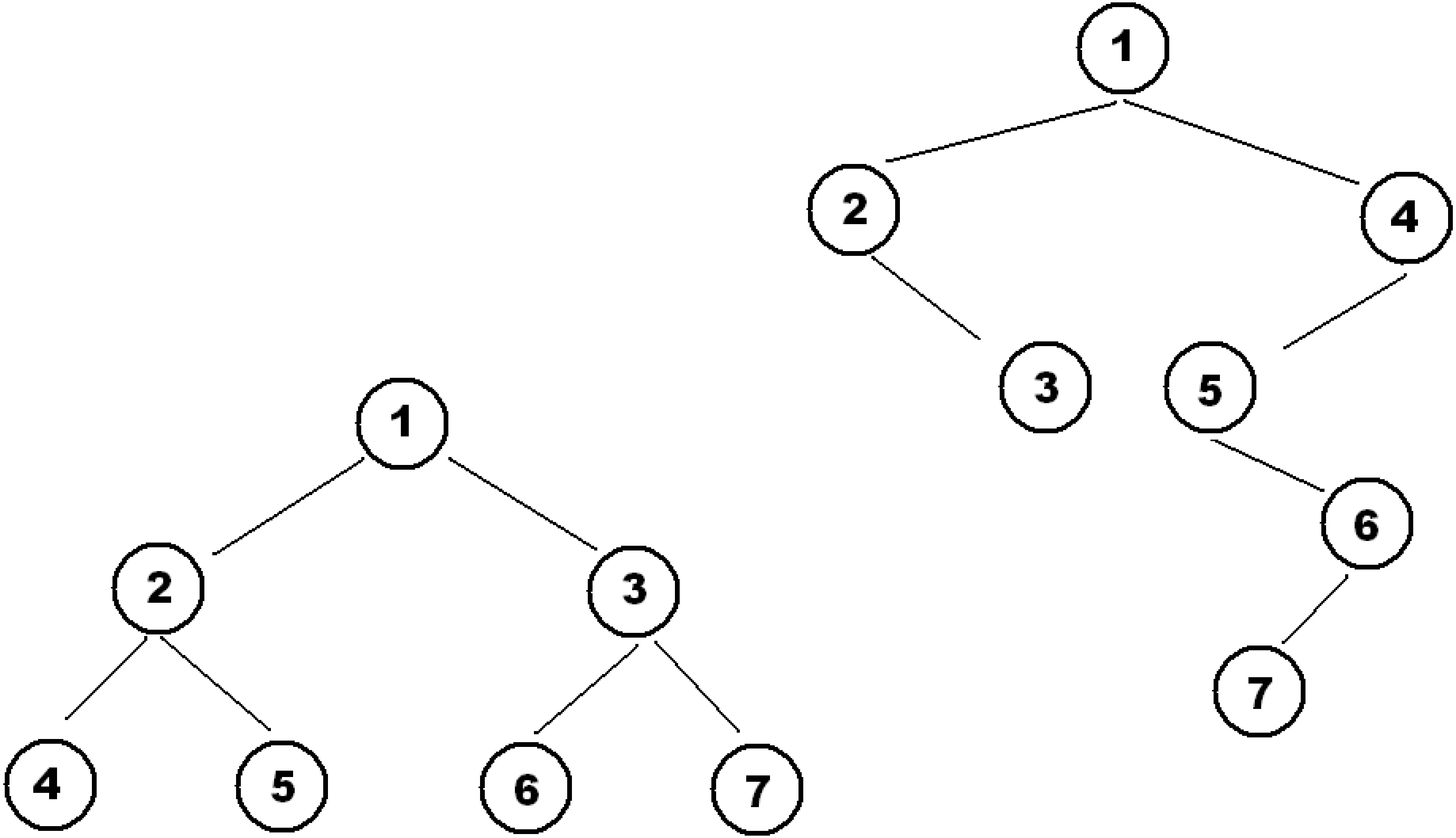
//continuation of iterativePostorder2(a) version 2

```
   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


1
2
4
3
5
6
7
1
2
3
4
5
6
7