PREORDER(T)

1.[Initialize]
If T = NULL
then Write(“*EMPTY TREE”)*
Return
else TOP 0
Call PUSH(S, TOP, T)
2.[Process each stacked branch address]
Repeat step 3 while TOP > 0
3.[Get stored address and branch left]
P POP(S, TOP)
Repeat while P ≠ NULL
Write(DATA(P))
If RPTR(P) ≠ NULL
then Call PUSH(S, TOP, RPTR(P))
P LPTR(P) (branch left)

RPREORDER(T)

1.[Process the root node]
If T ≠ NULL
then Write(DATA(T))
else Write(“*EMPTY TREE”)*
Return
2.[Process the left sub tree]
If LPTR(T) ≠ NULL
then Call RPREORDER(LPTR(T))
3.[Process the right sub tree]
If RPTR ≠ NULL
then Call RPREORDER(RPTR(T))
4.[Finished]
Return

POSTORDER(T)

1.[Initialize]
If T = NULL
then write(*“EMPTY TREE”*)
Return
else P □ T
TOP □ 0
2.[Traverse in postorder]
Repeat thru step 5 while true
3.[Descend left]
Repeat while P ‡ NULL
Call PUSH(S, TOP, P)
P □ LPTR(P)

POSTORDER(T) (CONT...)

4.[Process a node whose left and right sub trees have been traversed]
Repeat while S[TOP] < 0
P □ POP(S, TOP)
Write(DATA(P))
If TOP = 0
then Return
5.[Branch right and then work node from which we branched]
P □ RPTR(S[TOP])
S[TOP] □ – S[TOP]

RPOSTORDER(T)

1.[Check for empty tree]
If T = NULL
then Write(“*EMPTY TREE”)*
Return
2.[Process the left sub tree]
If LPTR(T) ‡ NULL
then Call RPOSTORDER(LPTR(T))
3.[Process the right sub tree]
If RPTR(T) ‡ NULL
then Call RPOSTORDER(RPTR(T))
4.[Process the root node]
Write(DATA(T))
5.[Finsihed]
Return