**ExtraExDS\_4Trees – Exercise 5**

**Specification**

spec: progressiveTree[NODE]

genres: tree, node, bool

operations:

parent: node tree 🡪 node

leftmost\_child: node tree 🡪 node

right\_sibling: node tree 🡪 node

label: node tree 🡪 label

create: label tree tree -> tree

root: tree -🡪 node

makenull: tree 🡪 tree

isProgressive: node tree 🡪 bool

transformTree: node tree -> tree

endspec

**Implementation**

node=record

element: label

leftmostchild: ^node

rightsibling: ^node

parent: ^node

endrecord

label: elementtype

class Tree

private root: ^node

public ^node parent(n: node)

public ^node left\_mostchild(n: node)

public ^node right\_sibling(n: node)

public label label(n: node)

public void create(leftTree, rightTree: ^node, l: label)

public ^node root()

public void makenull()

public bool isProgressive(n:node) {Returns a boolean if the tree is progressive (it is ordered)}

public void transformTree(n: node) {Transforms a non-progressive tree into a progressive one}

endclass

bool progressiveTree::isProgressive(n:node)

check: bool

child: ^node

if(n == null)

return true

endif

check:= false

child:= n^.leftchild {if there is no leftchild child:=NULL}

while child!=null

if (child^.label>child.^rightsibling^.label)

return false

else

check:= isProgressive(child) {Recursive call to the method}

if(check)

child:= child^rightsibling

else

return false

endif

endif

endwhile

endtrue

endmethod

void progressiveTree::transformTree(n:^node)

child:^node

aux\_list:list

if(n == null)

return

endif

child:=n^.leftchild

while(child != null)

transformTree(child)

list^.intert(child^.label)

child:= child^.rightchild

endwhile

aux\_list.bubblesort() {Method in the list to convert the list into a sorted list}

child:= n^.leftchild

while(child != null)

child^.label := aux\_list^.delete(aux\_list^.header)

child:= child^.rightchild

endwhile

endmethod