TREES: EXERCISE 5

Specification

```
spec: progressiveTree [NODE]
      genres: progressiveTree, node, list[nodes]
      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 -> boolean
            transformTree: node tree -> tree
endspec
```

Implementation

DATATYPES

```
node=record
      element: label
      leftmostchild: ^node
      rightsibling: ^node
endrecord
progressiveTree: ^node
label: elementtype
```

{Returns a boolean if the tree is progressive (it is ordered)}

```
bool progressiveTree::isProgressive(n:node)
      check: bool
      check:= false
      child: ^node
      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
      return true
endmethod
```

{Transforms a non-progressive tree into a progressive one}

```
void progressiveTree::transformTree(n:^node)
      child:^node
      child:=n^.leftchild
      aux_list:list
      while(child != null)
            transformTree(child)
            list.insert(child^.label)
            child:=child^.rightchild
      endwhile
      list.bubblesort()       {Converts the list into a sort list}
      child:=n^.leftchild
      while(child != null)
            child^.label:=list.delete(first)
            child:=child^.rightchild
      endwhile
endmethod
```