```
(*PROBLEM 6*)
(*Problem 6.1/6.2: A signature BTree that matches with binary tree structures
 that provide a representation for binary trees AND a signature ITEM that
 encapsulates whatever is needed to be known about any data type to create a
 structure satisfying the BTree signature.*)

signature ITEM =
sig
  type item
  val lt : item * item -> bool
  val printVal : item -> unit
end;

signature BTree =
sig
  structure Item : ITEM
  type item = Item.item
  type 'a tree
  val initTree : item tree
  val insert : item * item tree -> item tree
  val member : item * item tree -> bool
  val printTree : item tree -> unit
end;

(*Problem 6.3: A functor BTree that takes a structure satisfying the ITEM
 signature and produces a structure satisfying the BTree signature.*)

functor BTree (Item : ITEM) : BTree =
struct
    structure Item = Item
    type item = Item.item

    fun lt(p:Item.item,q:Item.item) = Item.lt(p,q)
    fun printVal(p:Item.item) = Item.printVal(p)

    datatype 'a tree = Empty | Node of 'a * 'a tree * 'a tree

    val initTree = Empty

    fun insert(i, Empty) = Node(i, Empty, Empty)
      | insert(i, Tree as Node(j, ltree, rtree)) =
        if Item.lt(i,j) then Node(j, insert(i, ltree), rtree)
        else
         if Item.lt(j,i) then Node(j, ltree, insert(i, rtree))
         else Tree

    fun member(i, Empty) = false
      | member(i, Node(j, ltree, rtree)) =
        if Item.lt(i,j) then member(i, ltree)
        else
         if Item.lt(j,i) then member(i, rtree)
         else true

```

```
54
55      fun printTree(Empty) = print("")
56       | printTree(Node(j, ltree, rtree)) =
57        (printTree(ltree);
58        Item.printVal(j);
59        printTree(rtree))
60
61    end;
62
63  (*Problem 6.3: ITEM structures for integers and strings and use with the BTree
64   functor *)
65
66  structure IntItem : ITEM =
67  struct
68   type item = int
69   val lt : (int * int ->bool) = op <;
70   fun printVal x =  print(Int.toString(x)^ "\n");
71  end;
72
73  structure StringItem : ITEM =
74  struct
75   type item = string
76   val lt : (string * string ->bool) = op <;
77   fun printVal x =  print(x^ "\n");
78  end;
79
80  structure IntTree = BTree(IntItem);
81  structure StrTree = BTree(StringItem);
82
83
84  Control.Print.printDepth := 100;
85  Control.Print.printLength := 100;
86
87  (*Problem 6.5 Creating and testing...*)
88  val intTree1 = IntTree.initTree;
89  val strTree1 = StrTree.initTree;
90
91  (*Test 1: Multiple insertions and print. The purpose of this test is to ensure
92   that the insert function correctly inserts integer elements into an integer
93   tree, while not inserting duplicates, and printing the result to test that the
94   print function correctly uses an in-order traversal method of printing elements
95  *)
96
97  print("\nInteger Tree Test1: Multiple insertions and print\n");
98  print("\nInserting 9\n");
99  val intTree2 = IntTree.insert(9,intTree1);
100 print("\nInserting 21\n");
101 val intTree3 = IntTree.insert(21,intTree2);
102 print("\nInserting 4\n");
103 val intTree4 = IntTree.insert(4,intTree3);
104 print("\nInserting 11\n");
105 val intTree5 = IntTree.insert(11,intTree4);
106 print("\nInserting 4 again\n");
107 val intTree4 = IntTree.insert(4,intTree3);
108 print("\nPrinting Tree:\n");
109 IntTree.printTree(intTree5);
```

```
110
111 (*Test 2: 4 is a member. The purpose of this test is to ensure that the member
112  function correctly returns true upon finding a queried item in a tree*)
113
114 print("\nInteger Tree Test2: 4 is a member?\n");
115 val test2 = IntTree.member(4,intTree5);
116
117 (*Test 3: 13 is not a member. The purpose of this test is to ensure that the
118  member function correctly returns false upon not finding a queried item in a
119  tree*)
120
121 print("\nInteger Tree Test3: 13 is a member?\n");
122 val test3 = IntTree.member(13,intTree5);
123
124 (*Test 4: Multiple insertions and print. The purpose of this test is to ensure
125  that the insert function correctly inserts string elements into an string
126  tree, while not inserting duplicates, and printing the result to test that the
127  print function correctly uses an in-order traversal method of printing elements
128 *)
129
130 print("\nString Tree Test4: Multiple insertions and print\n");
131 print("\nInserting Golf\n");
132 val strTree2 = StrTree.insert("Golf",strTree1);
133 print("\nInserting November\n");
134 val strTree3 = StrTree.insert("November",strTree2);
135 print("\nInserting Alpha\n");
136 val strTree4 = StrTree.insert("Alpha",strTree3);
137 print("\nInserting Zulu\n");
138 val strTree5 = StrTree.insert("Zulu",strTree4);
139 print("\nPrinting Tree:\n");
140 StrTree.printTree(strTree5);
141
142 (*Test 5: Zulu is a member. The purpose of this test is to ensure that the
143  member function correctly returns false upon not finding a queried item in a
144  tree*)
145
146 print("\nInteger Tree Test2: Zulu is a member?\n");
147 val test2 = StrTree.member("Zulu",strTree5);
148
149 (*Test 6: Bravo is not a member. The purpose of this test is to ensure that the
150  member function correctly returns false upon not finding a queried item in a
151  tree*)
152
153 print("\nInteger Tree Test3: Bravo is a member?\n");
154 val test3 = StrTree.member("Bravo",strTree5);
```

PDF document made with CodePrint using Prism