```
(*Problem 1.1: A type that is capable of representing a binary search tree
of any kind*)

datatype 'a tree = Empty | Node of 'a * ('a tree) * ('a tree);

(*Problem 1.2: A function to check if a given object is present in a binary
search tree*)

fun member(eq, ord, i, Empty) = false
 | member(eq, ord, i, Node(j, ltree, rtree)) =
  case eq(i, j) of
   true => true
   | false => if ord(i, j) then member(eq, ord, i, ltree)
    else member(eq, ord, i, rtree)
;

fun equality(x, y) =
 if (x = y)
  then true
 else
  false
;

fun intOrd(val1, val2) =
 if (val1 < val2)
  then true
 else
  false
;

fun strOrd(val1, val2) =
 case String.compare(val1, val2) of
  LESS => true
 |EQUAL => true
 |GREATER => false
;

(*Problem 1.3: A function to insert an element into a binary search tree*)

fun insert(eq, ord, i, Empty) = Node(i, Empty, Empty)
 | insert(eq, ord, i, tr as Node(j, ltree, rtree)) =
  case eq(i, j) of
   true => tr
  | false => if ord(i, j) then Node(j, insert(eq, ord, i,ltree), rtree)
   else Node(j, ltree, insert(eq, ord, i, rtree))
;

(*Problem 1.4: A function to insert an element into a binary search tree*)

fun printInt(x) =
 print(Int.toString(x)^ "\n")
;

```

```
fun printStr(x) =
 print(x^ "\n")
;

fun printTree(printType, Empty) = print("")
 | printTree(printType, Node(j, ltree, rtree)) =
  (printTree(printType, ltree);
  printType(j);
  printTree(printType, rtree))
;

Control.Print.printDepth := 100;
Control.Print.printLength := 100;

val intTree1 = Node(7, Node(5, Empty, Empty), Empty);

(*Test 1: 5 is a member*);
print("\nInteger Tree Test1: 5 is a member\n");
val test1 = member(equality, intOrd, 5, intTree1);

(*Test 2: 10 is not a member*);
print("\nInteger Tree Test2: 10 is not a member\n");
val test2 = member(equality, intOrd, 10, intTree1);

(*Test 3: Multiple insertions and print*);
print("\nInteger Tree Test3: Multiple insertions and print\n");
print("\nInserting 0\n");
val intTree2 = insert(equality, intOrd, 0, intTree1);
print("\nInserting 17\n");
val intTree3 = insert(equality, intOrd, 17, intTree2);
print("\nInserting 1\n");
val intTree4 = insert(equality, intOrd, 1, intTree3);
print("\nInserting 6\n");
val intTree5 = insert(equality, intOrd, 6, intTree4);
print("\nInteger Tree elements:\n");
printTree(printInt, intTree5);


val strTree1 = Node("Hotel", Empty, Node("Whiskey", Empty, Empty));

(*Test 4: Hotel is a member*);
print("\nString Tree Test4: Hotel is a member\n");
val test2 = member(equality, strOrd, "Hotel", strTree1);

(*Test 5: Alpha is not a member*);
print("\nString Tree Test5: Alpha is not a member\n");
val test3 = member(equality, strOrd, "Alpha", strTree1);
(*Test 6: Multiple insertions and print*);
print("\nString Tree Test6: Multiple insertions and print\n");
print("\nInserting Alpha\n");
val strTree2 = insert(equality, strOrd, "Alpha", strTree1);
print("\nInserting Foxtrot\n");
val strTree3 = insert(equality, strOrd, "Foxtrot", strTree2);
print("\nInserting Bravo\n");
val strTree4 = insert(equality, strOrd, "Bravo", strTree3);
print("\nInserting India\n");
```

```
val strTree5 = insert(equality, strOrd, "India", strTree4);
print("\nString Tree elements:\n");
printTree(printStr, strTree5);
```

PDF document made with CodePrint using Prism