```
1   -- in file binarysearchtree.adb
2   package body BinarySearchTree is
3
4      procedure InsertBinarySearchTree(Root: in out BinarySearchTreePoint;
       ARecord: BinarySearchTreeRecord) is
5         P, Q : BinarySearchTreePoint;
6         RecordKey : Akey := GetKey(ARecord);
7      begin
8         AllocateNode(Q, ARecord);
9         if Root = Null then  -- If null is passed in as Root, this is the first
          item in the tree. Create a head/root and attach to the left.
10           --HeadRecord := "zzzzzzzzzz"; --Save Info.Name of head so we can skip
             it while traversing the tree.
11           Put("Creating new tree starting with: ");
12           PrintFullRecord(Q.Info);
13           new_line;
14           Root := new Node;
15           Root.Rtag := true;
16           Root.Rlink := Root;
17           Root.Ltag := false;
18           Root.Info := HeadRecord;
19           InsertNode(Root,Q);
20        else  -- Tree is not empty.  Locate a match with existing node or
          position to insert new node.
21           P := Root;
22           Finder_Loop :
23           Loop  -- Search left and right for a match or insert in tree if not
             found.
24              if RecordKey < P.Info then  -- Search to left
25                 if P.Ltag then
26                    P := P.Llink;
27                 else
28                    InsertNode(P, Q);
29                    exit Finder_Loop;
30                 end if;
31              elsif RecordKey > P.Info then  -- Search to right.
32                 if P.Rtag then
33                    P := P.RLink;
34                 else-- Insert node as right subtree.
35                    InsertNode(P, Q);-- New node inserted.
36                    exit Finder_Loop;
37                 end if;
38              else  -- Implies that Akey matches P.Key.
39                    -- Customer with matching name exists, insert to left of
                      duplicate.
40                    -- Overloaded "<=" used in InsertNode handles this.
41                 InsertNode(P, Q);
42                 exit Finder_Loop;
43              end if;
44           end loop Finder_Loop;
45        end if;
46        New_Line;
47     end InsertBinarySearchTree;
48
49     procedure InsertNode(P, Q: in out BinarySearchTreePoint) is
50     begin
51        if GetKey(Q.Info) <= P.Info then
52           --Insert Q as left subtree of P
53           Put("Inserting "); PrintIdentityRecord(Q.Info);
54           Put(" as left child of "); PrintIdentityRecord(P.Info);
55           New_Line;
56           numNodes := numNodes + 1;
57           Q.Llink := P.Llink;
```

```
 58            Q.Ltag := P.Ltag;
 59            P.Llink := Q;
 60            P.Ltag := true;
 61            Q.Rlink := P;
 62            Q.Rtag := false;
 63            if Q.Ltag then
 64               InOrderPredecessor(Q).Rlink := Q;
 65            end if;
 66         else
 67            --Insert Q as right subtree of P
 68            Put("Inserting "); PrintIdentityRecord(Q.Info);
 69            Put(" as right child of "); PrintIdentityRecord(P.Info);
 70            New_Line;
 71            numNodes := numNodes + 1;
 72            Q.Rlink := P.Rlink;
 73            Q.Rtag := P.Rtag;
 74            P.Rlink := Q;
 75            P.Rtag := true;
 76            Q.Llink := P;
 77            Q.Ltag := false;
 78            if Q.Rtag then
 79               InOrderSuccessor(Q).Llink := Q;
 80            end if;
 81         end if;
 82      end InsertNode;
 83
 84      procedure AllocateNode(Q: out BinarySearchTreePoint; ARecord:
         BinarySearchTreeRecord) is
 85      begin  -- Allocates and places AKey in node pointed to by Q.
 86         Q := new Node;
 87         Q.Info := ARecord;
 88         Q.LLink := null;
 89         Q.RLink := null;
 90         Q.Ltag := false;
 91         Q.Rtag := false;
 92      end AllocateNode;
 93
 94      procedure FindCustomerIterative(Root: in BinarySearchTreePoint; RecordKey:
         in Akey; RecordPoint: out BinarySearchTreePoint) is
 95         P : BinarySearchTreePoint := Root;
 96      begin
 97         Finder_Loop :
 98         loop
 99            if RecordKey < P.Info and P.Ltag then
100               P := P.Llink;
101            elsif RecordKey > P.Info and P.Rtag then
102               P := P.Rlink;
103            else
104               --Either reached a different leaf node or found the customer in the
                  tree.
105               exit Finder_Loop;
106            end if;
107         end loop Finder_Loop;
108         if RecordKey = P.Info then
109            RecordPoint := P;
110            Put("Found customer "); PrintFullRecord(P.info); Put(" iteratively.");
111            New_Line;
112            RecordPoint := P;
113         else
114            RecordPoint := null;
115            Put("Could not find "); PrintKey(RecordKey); Put(" iteratively.");
116            New_Line;
117         end if;
```

```
118         return;
119      end FindCustomerIterative;
120
121      procedure FindCustomerRecursive(Root: in BinarySearchTreePoint; RecordKey:
         in AKey; RecordPoint: out BinarySearchTreePoint) is
122      begin
123         if RecordKey < Root.Info and Root.Ltag then
124            FindCustomerRecursive(Root.Llink, RecordKey, RecordPoint);
125         elsif RecordKey > Root.Info and Root.Rtag then
126            FindCustomerRecursive(Root.Rlink, RecordKey, RecordPoint);
127         elsif RecordKey = Root.Info then
128            RecordPoint := Root;
129            New_Line;
130            Put("Found customer "); PrintFullRecord(RecordPoint.Info); Put("
               recursively.");
131            New_Line;
132            return;
133         else
134            RecordPoint := null;
135            Put("Could not find "); PrintKey(RecordKey); Put(" recursively.");
136            New_Line(2);
137            return;
138         end if;
139      end FindCustomerRecursive;
140
141      procedure PreOrderTraversalIterative(TreePoint: in BinarySearchTreePoint) is
142         package nodeStack is new gstack(numNodes, BinarySearchTreePoint);
143         use nodeStack;
144         P, Q: BinarySearchTreePoint := TreePoint;
145         StartingInfo : BinarySearchTreeRecord := TreePoint.Info;
146         flag: Integer := 0;
147      begin
148         New_Line;
149         Put_Line("Starting pre order traversal iterative");
150         If GetKey(TreePoint.Info) = HeadRecord and TreePoint.Ltag then
151            P := TreePoint.Llink;
152         end if;
153         Traverse_Loop:
154         loop
155            if P /= null then
156               PrintFullRecord(P.Info);
157               New_Line;
158               nodeStack.push(P);
159               if P.Ltag then
160                  P := P.Llink;
161               else
162                  P := null;
163               end if;
164            else
165               if nodeStack.numItems = 0 then
166                  exit Traverse_Loop;
167               end if;
168               P := nodeStack.pop;
169               if P.Rtag then
170                  P := P.Rlink;
171               else
172                  P := null;
173               end if;
174            end if;
175         end loop Traverse_Loop;
176      end PreOrderTraversalIterative;
177
178      function PreOrderSuccessor(TreePoint: in BinarySearchTreePoint) return
```

```
      BinarySearchTreePoint is
179      P, Q: BinarySearchTreePoint;
180   begin
181      P := TreePoint;
182      if P.Ltag then
183         Q := P.Llink;
184      else
185         Q := P;
186         while Q.Rtag /= true loop
187            Q := Q.Rlink;
188         end loop;
189         Q := Q.Rlink;
190      end if;
191      return Q;
192   end PreOrderSuccessor;
193
194   procedure PostOrderTraversalIterative(TreePoint: in BinarySearchTreePoint) is
195      type TNode is
196         record
197            aNode : BinarySearchTreePoint;
198            Way: Integer;
199         end record;
200      package nodeStack is new gstack(numNodes, TNode);
201      MyNode : TNode;
202      P : BinarySearchTreePoint := TreePoint;
203   begin
204      New_Line;
205      Put("Starting post order traversal iterative from: ");
206      If GetKey(TreePoint.Info) = HeadRecord and TreePoint.Ltag then
207         P := TreePoint.Llink;
208      end if;
209      PrintFullRecord(P.Info);
210      New_Line;
211      Traverse_Loop:
212      loop
213         if P /= null then
214            MyNode.aNode := P;
215            MyNode.Way := 0;
216            nodeStack.push(MyNode);
217            if P.Ltag then
218               P := P.Llink;
219            else
220               P := null;
221            end if;
222         else
223            if nodeStack.numItems = 0 then
224               exit Traverse_Loop;
225            end if;
226            MyNode := nodeStack.pop;
227            P := MyNode.aNode;
228            if MyNode.Way = 0 then
229               MyNode.Way := 1;
230               nodeStack.push(MyNode);
231               if P.Rtag then
232                  P := P.Rlink;
233               else
234                  P := null;
235               end if;
236            else
237               Inner_Loop:
238               loop
239                  if P /= null then
240                     PrintFullRecord(P.Info);--Visit P
```

```
241                        new_line;
242                     end if;
243                     if nodeStack.numItems = 0 then
244                        exit Traverse_Loop;
245                     end if;
246                     MyNode := nodeStack.pop;
247                     P := MyNode.aNode;
248                     if MyNode.Way = 0 then
249                        MyNode.Way := 1;
250                        nodeStack.push(MyNode);
251                        if P.Rtag then
252                           P := P.Rlink;
253                        else
254                           P := null;
255                        end if;
256                        exit Inner_Loop;
257                     end if;
258                  end loop Inner_Loop;
259               end if;
260            end if;
261         end loop Traverse_Loop;
262      end PostOrderTraversalIterative;
263
264      procedure PostOrderTraversalRecursiveCaller(TreePoint: in
         BinarySearchTreePoint) is
265      begin
266         New_Line;
267         Put("Starting post oder traversal recursive with: ");
268         if TreePoint.Info = HeadRecord then -- Ignore head
269            PrintFullRecord(TreePoint.Llink.Info);
270         else
271            PrintFullRecord(TreePoint.Info);
272         end if;
273         New_Line;
274         PostOrderTraversalRecursive(TreePoint);
275      end PostOrderTraversalRecursiveCaller;
276
277      procedure PostOrderTraversalRecursive(TreePoint: in BinarySearchTreePoint) is
278         S : BinarySearchTreePoint := TreePoint;
279      begin
280         if S.Ltag then --Traverse the left subtree.
281            PostOrderTraversalRecursive(S.Llink);
282         end if;
283         if S.Rtag then --Traverse the right subtree.
284            if S.Rlink = S then
285               --S is Head Node
286               return;
287            end if;
288            PostOrderTraversalRecursive(S.Rlink);
289         end if;
290         PrintFullRecord(S.Info);--Visit the node (print its contents)
291         New_Line;
292         return;
293      end PostOrderTraversalRecursive;
294
295      procedure PreOrderTraversalRecursive(TreePoint: in BinarySearchTreePoint) is
296         S : BinarySearchTreePoint := TreePoint;
297      begin
298         if S.Info = HeadRecord then
299            S := S.Llink;
300         end if;
301         New_Line;
302         PrintFullRecord(S.Info);--Visit the node (print its contents)
```

```
303         if S.Ltag then --Traverse the left subtree.
304            PreOrderTraversalRecursive(S.Llink);
305         end if;
306         if S.Rtag then --Traverse the right subtree.
307            if S.Rlink = S then
308               --S is Head Node
309               return;
310            end if;
311            PreOrderTraversalRecursive(S.Rlink);
312         end if;
313         return;
314      end PreOrderTraversalRecursive;
315
316      procedure PreOrderTraversalRecursiveCaller(TreePoint: in
         BinarySearchTreePoint) is
317      begin
318         New_Line;
319         Put_Line("Starting pre order traversal recursive with: ");
320         PreOrderTraversalRecursive(TreePoint);
321      end PreOrderTraversalRecursiveCaller;
322
323      function InOrderSuccessor(TreePoint: in BinarySearchTreePoint) return
         BinarySearchTreePoint is
324         Q: BinarySearchTreePoint;
325      begin
326         Q := TreePoint.Rlink; --Look right
327         if TreePoint.Rtag = false then
328            return Q;
329         else
330            --Search left
331            while Q.Ltag loop
332               Q := Q.Llink;
333            end loop;
334         end if;
335         return Q;
336      end InOrderSuccessor;
337
338      function InOrderPredecessor(TreePoint: in BinarySearchTreePoint) return
         BinarySearchTreePoint is
339         Q: BinarySearchTreePoint;
340      begin
341         Q := TreePoint.Llink;
342         if TreePoint.Ltag then
343            while Q.Rtag loop
344               Q := Q.Rlink;
345            end loop;
346         end if;
347         return Q;
348      end InOrderPredecessor;
349
350      procedure InOrderTraversal(TreePoint: in BinarySearchTreePoint) is
351         P : BinarySearchTreePoint := TreePoint;
352         i : Integer := 0;
353      begin
354         New_Line;
355         Put("Starting inorder traversal from: ");
356         If GetKey(TreePoint.Info) = HeadRecord and TreePoint.Ltag then
357            P := TreePoint.Llink;
358         end if;
359         PrintFullRecord(P.Info);
360         New_Line;
361         while i < numNodes loop
362            P :=  InOrderSuccessor(P);
```

```
363            if GetKey(P.Info) = HeadRecord then --skip printing head and go to next
364               P := InOrderSuccessor(P);
365            end if;
366            PrintFullRecord(P.Info);
367            New_Line;
368            i := i + 1;
369         end loop;
370         New_Line;
371      end InOrderTraversal;
372
373      procedure TreeFromFile(filename: String; Root: in out BinarySearchTreePoint)
         is
374         f: File_Type;
375         Str: String(1..50);
376         Arecord : BinarySearchTreeRecord;
377      begin
378         Ada.Text_IO.Open(File => f, Mode => In_File, Name => filename);
379         New_Line;
380         Put_Line("Reading records from a file.");
381         while not End_Of_File(f) loop
382            Move(Get_Line(f),Str);
383            RecordFromString(Str, ARecord);
384            Put("Read "); PrintKey(GetKey(Arecord)); Put(" from file.");
385            New_Line;
386            InsertBinarySearchTree(Root, ARecord);
387         end loop;
388         Ada.Text_IO.Close(f);
389      end TreeFromFile;
390
391      procedure DeleteRandomNode(DeletePoint, Head: in BinarySearchTreePoint) is
392         Q: BinarySearchTreePoint := DeletePoint;
393         S: BinarySearchTreePoint := InOrderSuccessor(DeletePoint);
394         QParent: BinarySearchTreePoint := FindParent(Q, Head);
395         SParent: BinarySearchTreePoint := FindParent(S, Head);
396         Temp: BinarySearchTreeRecord;
397      begin
398
399         if not (Q.Rtag or else Q.Ltag) then
400            --Base Case: deleting a leaf.
401            if QParent.Llink = Q then
402               --Q is left from its parent
403               if Q.LLink.Rtag = false then
404                  Q.Llink.Rlink := QParent;
405               end if;
406               QParent.Ltag := false;
407               QParent.Llink := Q.Llink;
408            elsif QParent.Rlink = Q then
409               --Q is right from its parent
410               if Q.Rlink.Ltag = false then
411                  Q.Rlink.Llink := QParent;
412               end if;
413               QParent.Rtag := false;
414               QParent.Rlink := Q.Rlink;
415            end if;
416            Put_Line("Deleting found item and returning space to the heap.");
417            Free(Q);
418            numNodes := numNodes - 1;
419            return;
420         elsif Head.Llink = Q then
421            --Deleting root of tree.
422            Temp := S.Info; --save the record in the inorder successor to be
               swapped in
423            DeleteRandomNode(S, Head); --recursively delete Q's inorder successor
```

```
424            Put("Swapping record "); PrintFullRecord(Temp); Put(" into ");
               PrintFullRecord(Q.Info); Put("'s node");
425            New_Line;
426            Q.Info := Temp; --swap in the record from Q's inorder successor
427         else
428            --Deleting non-root with at least 1 child.
429            if S = Head then
430               --There is no inorder successor to replace the deleted node with,
                  so we will use the inorder predecessor instead.
431               S := InOrderPredecessor(Q);
432               Temp := S.Info;
433               DeleteRandomNode(S, Head);
434               Put("Swapping record "); PrintFullRecord(Temp); Put(" into ");
                  PrintFullRecord(Q.Info); Put("'s node");
435               New_Line;
436               Q.Info := Temp;
437            else
438               Temp := S.Info;
439               DeleteRandomNode(S, Head);
440               Put("Swapping record "); PrintFullRecord(Temp); Put(" into ");
                  PrintFullRecord(Q.Info); Put("'s node");
441               New_Line;
442               Q.Info := Temp;
443            end if;
444         end if;
445         return;
446      end DeleteRandomNode;
447
448      function FindParent(P, Head: in BinarySearchTreePoint) return
         BinarySearchTreePoint is
449         J, S: BinarySearchTreePoint := Head;
450         Q : BinarySearchTreePoint := P;
451      begin
452         Finder_Loop :
453         loop
454            if GetKey(P.Info) < J.Info and J.Ltag then
455               Q := J;
456               J := J.Llink;
457            elsif GetKey(P.Info) > J.Info and J.Rtag then
458               Q := J;
459               J := J.Rlink;
460            else
461               exit Finder_Loop; --Either reached a different leaf node or found
                  the customer in the tree.
462            end if;
463         end loop Finder_Loop;
464         return Q;
465      end FindParent;
466
467      procedure ReverseInOrderCaller(treePoint: in BinarySearchTreePoint) is
468      begin
469         Put("Starting reverse in order traversal from: ");
470         if GetKey(treePoint.Info) = HeadRecord then --skip displaying info of
         Head node
471            PrintFullRecord(TreePoint.Llink.Info);
472         else
473            PrintFullRecord(TreePoint.Info);
474         end if;
475         New_Line;
476         ReverseInOrder(treePoint);
477      end ReverseInOrderCaller;
478
479      procedure ReverseInOrder(treePoint: in BinarySearchTreePoint) is
```

```
480         S: BinarySearchTreePoint := treePoint;
481      begin
482         if GetKey(treePoint.Info) = HeadRecord then --Do not print head.
483            S := treePoint.Llink;
484         end if;
485         if S.Rtag then --Traverse the right subtree.
486            ReverseInOrder(S.Rlink);
487         end if;
488         PrintFullRecord(S.Info);--Visit the node (print its contents)
489         New_Line;
490         if S.Ltag then --Traverse the left subtree.
491            ReverseInOrder(S.Llink);
492         end if;
493         return;
494      end ReverseInOrder;
495
496      procedure GetHead(P: in out BinarySearchTreePoint) is
497      begin
498         while GetKey(P.Info) /= HeadRecord loop
499            P := InOrderSuccessor(P);
500         end loop;
501         return;
502      end GetHead;
503   end BinarySearchTree;
504
```