```
 1  -- in file binarysearchtree.ads
 2  with gstack;
 3  with Ada.Strings;
 4  use Ada.Strings;
 5  with Ada.Text_IO;
 6  use ada.Text_IO;
 7  with Ada.Strings.Fixed;
 8  use Ada.Strings.Fixed;
 9  with Unchecked_Deallocation;
10  generic
11     type Akey is private;
12     type BinarySearchTreeRecord is private;
13     with function "<" (TheKey: in Akey; ARecord: in BinarySearchTreeRecord)
       return Boolean;
14     with function ">" (TheKey: in Akey; ARecord: in BinarySearchTreeRecord)
       return Boolean;
15     with function "=" (TheKey: in Akey; ARecord: in BinarySearchTreeRecord)
       return Boolean;
16     with function "<=" (TheKey: in Akey; ARecord: in BinarySearchTreeRecord)
       return Boolean;
17     with procedure PrintFullRecord(ARecord: in BinarySearchTreeRecord);
18     with procedure PrintIdentityRecord(ARecord: in BinarySearchTreeRecord);
19     HeadKey : Akey;
20     HeadRecord : BinarySearchTreeRecord;
21     with function GetKey(ARecord: in BinarySearchTreeRecord) return Akey;
22     with procedure RecordFromString(Str1: in String; ARecord: in out
       BinarySearchTreeRecord);
23     with procedure PrintKey(TheKey: in Akey);
24  package BinarySearchTree is
25     --Track the number of nodes in tree. Used in some traversals.
26     numNodes : Natural := 0;
27
28     subtype String10 is String(1..10);  -- You may use an enumeration type if
       desired.
29
30     -- Points to a node in a binary search
       tree.
31     type BinarySearchTreePoint is limited private;
32
33     -- This procedure inserts a node (customer) into the tree in search tree
       using iteration. If a customer with
34     -- duplicate name already customer exist, the new customer should be
       inserted so they would
35     -- appear "after" the older customer when the tree is traversed in inorder.
36     -- The tree must be threaded in "inorder". The search to locate the position
       for the new
37     -- record must be iterative!
38     procedure InsertBinarySearchTree(Root: in out BinarySearchTreePoint;
       ARecord: BinarySearchTreeRecord); --pg 93, modify for threads
39
40     -- This procedure locates a customer using a binary search.  A pointer is
       returned to the
41     -- customer record if they exist, otherwise a Null pointer is returned (in
       CustomerPoint).
42     -- The search must be implemented iteratively.
43     procedure FindCustomerIterative(Root: in BinarySearchTreePoint; RecordKey:
       in Akey; RecordPoint: out BinarySearchTreePoint);
44
45     -- This procedure locates a customer using a binary search.  A pointer is
       returned to the
46     -- customer record if they exist, otherwise a Null pointer is returned (in
       CustomerPoint).
47     -- The search must be implemented recursively.
```

```
48      procedure FindCustomerRecursive(Root: in BinarySearchTreePoint; RecordKey:
        in AKey; RecordPoint: out BinarySearchTreePoint);
49
50      -- This function returns the address of the next node in "inorder" taking
        advantage of threads.
51      -- The user may enter the  tree at any random location.  This is sometimes
        called an iteration
52      -- function or iterater (no recursion).
53      function InOrderSuccessor(TreePoint: in BinarySearchTreePoint) return
        BinarySearchTreePoint;
54      function InOrderPredecessor(TreePoint: in BinarySearchTreePoint) return
        BinarySearchTreePoint;
55
56      --Traverse the BST in order using method InOrderSuccessor
57      procedure InOrderTraversal(TreePoint: in BinarySearchTreePoint);
58
59      -- Pre/Post order traversal of a tree using using a stack allocated
        explicitly by the programmer!
60      procedure PreOrderTraversalIterative(TreePoint: in BinarySearchTreePoint);
61      procedure PostOrderTraversalIterative(TreePoint: in BinarySearchTreePoint);
        --pg 85
62
63      --The steps never call for this, but the grading check 5 specifically
        mentions it?
64      -- See message at end of A Option in main.
65      procedure PreOrderTraversalRecursive(TreePoint: in BinarySearchTreePoint);
66      procedure PreOrderTraversalRecursiveCaller(TreePoint: in
        BinarySearchTreePoint);
67
68      -- B Option
69      --This procedure deletes a node by recursively replacing it with its inorder
        successor if one is available, otherwise it replaces it with its inorder
        predecessor.
70      --If the inorder successor/predecessor replacing the deleted node is not a
        leaf node, DeleteRandomNode is called recursively replacing nodes with their
        inorder successors until a leaf node is finally returned to the heap.
71      procedure DeleteRandomNode(DeletePoint, Head: in BinarySearchTreePoint);
72
73      --Returns the parent node of P, used in DeleteRandomNode
74      function FindParent(P, Head: in BinarySearchTreePoint) return
        BinarySearchTreePoint;
75
76      --Must be recursive.
77      procedure ReverseInOrder(treePoint: in BinarySearchTreePoint);
78      --Calls ReverseInOrder after writing starting info.
79      --Can't set this up in ReverseInOrder because it is recursive and will
        display the message multiple times.
80      procedure ReverseInOrderCaller(treePoint: in BinarySearchTreePoint);
81
82      -- A Option
83      procedure PostOrderTraversalRecursive(TreePoint: in BinarySearchTreePoint);
84      --Calls PostOrderTraversalRecursive after writing starting info.
85      procedure PostOrderTraversalRecursiveCaller(TreePoint: in
        BinarySearchTreePoint);
86
87      --Creates a node
88      procedure AllocateNode(Q: out BinarySearchTreePoint; ARecord:
        BinarySearchTreeRecord); --pg 93, modify for threads
89      --Inserts a node into the BST
90      procedure InsertNode(P, Q: in out BinarySearchTreePoint); --pg 93, modify
        for threads
91      --Reads from a file and calls overloaded generic method RecordFromString to
        create a record before inserting it into the tree.
```

```
 92      procedure TreeFromFile(filename: String; Root: in out BinarySearchTreePoint);
 93
 94      --Traverse the tree from any node to find and return the head node.
 95      procedure GetHead(P: in out BinarySearchTreePoint);
 96
 97   private
 98
 99      type Node;
100      type BinarySearchTreePoint is access Node;
101      type Node is
102         record
103            Llink, Rlink:  BinarySearchTreePoint;
104            Ltag, Rtag:  Boolean;  -- True indicates pointer to lower level, False
               a thread.
105            Info:  BinarySearchTreeRecord;
106         end record;
107      HeadName :String10;
108
109      --Return space to the heap
110      procedure Free is new Unchecked_Deallocation(Node, BinarySearchTreePoint);
111   end BinarySearchTree;
112
```