```
 1  --in file main.adb
 2  with BinarySearchTree;
 3  with Ada.Strings;
 4  use Ada.Strings;
 5  with Ada.Strings.Fixed;
 6  use Ada.Strings.Fixed;
 7  with Ada.Text_IO;
 8  use ada.Text_IO;
 9  procedure Main is
10     subtype String10 is String(1..10);
11     type Customer is
12        record
13           Name:  String10;
14           PhoneNumber: String10;
15        end record;
16     HeadKey : String10 := "zzzzzzzzzz";-- 'z' is the highest value ASCII
       character, so a String10 with all z's guarantees insertion to the left of
       Head node
17     HeadCustomer : Customer := Customer'(HeadKey,"867-5309  ");
18
19     function "<" (TheKey: in String10;  ARecord: in Customer) return Boolean is
20     begin
21        -- Is TheKey less than the key of ARecord?
22        return (TheKey < ARecord.Name);
23     end "<";
24
25     function ">" (TheKey: in String10;  ARecord: in Customer) return Boolean is
26     begin
27        --Is TheKey greater than the key of ARecord?
28        return (TheKey > ARecord.Name);
29     end ">";
30
31     function "=" (TheKey: in String10;  ARecord: in Customer) return Boolean is
32     begin
33        --Is TheKey equal to the key of ARecord?
34        return (TheKey = ARecord.Name);
35     end "=";
36     function "<=" (TheKey: in String10; ARecord: in Customer) return Boolean is
37     begin
38        return (TheKey <= ARecord.Name);
39     end "<=";
40
41     function ToString10(str: in String) return String10 is
42        str10: String10;
43     begin
44        Move(str,str10);
45        return str10;
46     end ToString10;
47
48     procedure CustomerFromString(Str1: in String; ARecord: in out Customer) is
49        i: Integer := 1;
50        custName : String10;
51        custPhone: String10;
52     begin
53        --Take in a String and output a record.
54        MLoop :
55        loop
56           i := i + 1;
57           if Str1(i..i) = "," then
58              exit MLoop;
59           end if;
60        end loop MLoop;
61        custName := ToString10(Str1(Str1'First..(i-1)));
```

```
62          custPhone := ToString10(Trim(Str1((i+1)..Str1'Last), Left));
63          ARecord := Customer'(custName,custPhone);
64          return;
65       end CustomerFromString;
66
67       procedure PrintFullCustomer(ARecord: in Customer) is
68       begin
69          Put(Trim(ARecord.Name, Right));
70          Put(", ");
71          Put(Trim(ARecord.PhoneNumber, Right));
72       end PrintFullCustomer;
73
74       procedure PrintCustomerName(ARecord: in Customer) is
75       begin
76          Put(Trim(ARecord.Name, Right));
77       end PrintCustomerName;
78
79       function GetName(ARecord: in Customer) return String10 is
80       begin
81          return ARecord.Name;
82       end GetName;
83
84       procedure PrintKey(Name: in String10) is
85       begin
86          Put(Trim(Name, Right));
87       end PrintKey;
88
89       package MySearchTree is new BinarySearchTree(String10, Customer, "<", ">",
         "=","<=", PrintFullCustomer, PrintCustomerName, HeadKey, HeadCustomer,
         GetName, CustomerFromString, PrintKey);
90       use MySearchTree;
91       Root: BinarySearchTreePoint;
92       FoundCustomer: BinarySearchTreePoint;
93       f : Ada.Text_IO.File_Type;
94    begin
95       Ada.Text_IO.Create(f, Ada.Text_IO.Out_File, "output.txt");
96       Ada.Text_IO.Put_Line("Redirecting all output to output.txt");
97       Ada.Text_IO.Set_Output(f);
98
99       --C OPTION TRANSACTIONS
100      New_Line;
101      Put_Line("C Option");
102      TreeFromFile("inputC1.txt", Root); --1
103      FindCustomerIterative(Root, ToString10("Ortiz"), FoundCustomer); --2
104      FindCustomerRecursive(Root, ToString10("Ortiz"), FoundCustomer); --3
105      FindCustomerIterative(Root, ToString10("Penton"), FoundCustomer); --4
106      FindCustomerRecursive(Root, ToString10("Penton"), FoundCustomer); --5
107      FindCustomerIterative(Root, ToString10("Ikerd"), FoundCustomer); --6.1
108      InOrderTraversal(FoundCustomer); --6.2
109      TreeFromFile("inputC2.txt", Root); --7
110      InOrderTraversal(Root); --8
111      PreOrderTraversalIterative(Root); --9
112      PostOrderTraversalIterative(Root); --10
113
114      --B OPTION TRANSACTIONS
115      New_Line;
116      Put_Line("B Option");
117      GetHead(Root);
118      FindCustomerIterative(Root, ToString10("Robson"), FoundCustomer); --7.1
119      DeleteRandomNode(FoundCustomer, Root); --7.2
120      InOrderTraversal(Root);
121      FindCustomerIterative(Root, ToString10("Moutafis"), FoundCustomer); --7.3
122      DeleteRandomNode(FoundCustomer, Root); --7.4
```

```
123      InOrderTraversal(Root);
124      FindCustomerIterative(Root, ToString10("Ikerd"), FoundCustomer); --7.5
125      DeleteRandomNode(FoundCustomer, Root); --7.6
126      TreeFromFile("inputB1.txt", Root); --8
127      InOrderTraversal(Root); --9
128      ReverseInOrderCaller(Root); --10
129      PreOrderTraversalIterative(Root); --11
130
131      --A
132      New_Line;
133      Put_Line("A Option");
134      PostOrderTraversalIterative(Root);--12
135      PostOrderTraversalRecursiveCaller(Root); --13
136
137      New_Line(2);
138      Put_Line("The lab steps do not ask for this, but there is a grading check
         for a recursive preorder traversal.");
139      Put_Line("All preorder traversals before this one use the iterative
         version.");
140      Put_Line("This is just a demonstration that I created a working recursive
         preorder as well.");
141      PreOrderTraversalRecursiveCaller(Root);
142
143      Ada.Text_IO.Close(f);
144   end Main;
145
```