COSC3319 Search Tree Spring 2018 Burris

Due: Thursday April 26 prior to the start of class! Late labs will not be accepted. Do no procrastinate. *All trees must be threaded using the inorder predecessor and inorder successor for left and right links respectively*!

"C" Option (WARNING: Only the FindCustomerRecursive method is recursive!):

You may use Ada, C++, assembly or Java. If you desire to use another language ask for permission. Implement the following package using constructs in the language of your choice. Please make all data structures private types in your implementation language. **I have recommended but not required "*limited private*" in Ada. This is a special feature of Ada developed specifically for high security situations encountered by the military, NSA, Homeland Security, and other security conscience agencies/applications. As an analogy, it allows you to provide a "key" to a user. The user has complete functional use of the key but the language translator enforces the abstraction the user may not make a copy of the key. This level of security in programming languages is very rare and not in any of the previously listed languages other than Ada.**

```
package BinarySearchTree is
    type String10 is new String(1..10);  // You may use an enumeration type if desired.

    -- Points to a node in a binary search tree.
    type BinarySearchTreePoint is limited private;  -- or type BinarySearchTreePoint is private;

    -- This procedure inserts a node (customer) into the tree in search tree order. If a customer with
    -- duplicate name already customer exist, the new customer should be inserted so they would
    -- appear "after" the older customer when the tree is traversed in inorder.
    -- The tree must be threaded in "inorder." The search to locate the position for the new
    -- record must be iterative!
    procedure InsertBinarySearchTree(Root:  in out BinarySearchTreePoint;
                                     custName: in String10; custPhone: String10 );

    -- This procedure locates a customer using a binary search.  A pointer is returned to the
    -- customer record if they exist, otherwise a Null pointer is returned (in CustomerPoint).
    -- The search must be implemented iteratively.
    procedure FindCustomerIterative(Root: in BinarySearchTreePoint;
                                    CustomerName:  in String10;
                                    CustomerPoint:  out BinarySearchTreePoint);

    -- This procedure locates a customer using a binary search.  A pointer is returned to the
    -- customer record if they exist, otherwise a Null pointer is returned (in CustomerPoint).
    -- The search must be implemented recursively.
    procedure FindCustomerRecursive(Root: in BinarySearchTreePoint;
                                    CustomerName:  in String10;
                                    CustomerPoint:  out BinarySearchTreePoint);


    -- This function returns the address of the next node in "inorder" taking advantage of threads.
    -- The user may enter the  tree at any random location.  This is sometimes called an iteration
    -- function or iterater (no recursion).
    function InOrderSuccessor(TreePoint: in BinarySearchTreePoint)
        return BinarySearchTreePoint;

    -- Access functions to return customer names and phone numbers.
    function CustomerName(TreePoint: in BinarySearchTreePoint) return String10;
    function CustomerPhone(TreePoint: in BinarySearchTreePoint) return String10;
```

```
        -- Preorder traversal of a tree using using a stack allocated explicitly by the programmer!
        procedure PreOrderTraversalIterative(TreePoint: in BinarySearchTreePoint);

private
        type Customer is
                record
                        Name:  String10;
                        PhoneNumber: String10;
                end record;

        type Node;
        type BinarySearchTreePoint is access Node;
        type Node is
                record
                        Llink, Rlink:  BinarySearchTreePoint;
                        Ltag, Rtag:  Boolean;  -- True indicates pointer to lower level, False a thread.
                        Info:  Customer;
                end record;
end BinarySearchTree;
```

You may amend the package specification if required but management would really prefer the actual tree remain limited private (the tree should only be accessible via member functions by users of the package). C++ and Java support the "private" concept but do not support the concept of "limited private." The runtime system assures the user data of type limited private may not be copied by a user of the system. This is a very important security feature for implementing high security software.

The implementation of the above tree constitutes an example of a abstract data type (ADT). **The tree search to locate the position in the tree where a new node is to be inserted must be iterative in procedure InsertBinarySearchTree(Root: in out BinarySearchTreePoint; custName: in String10; custPhone: String10);**. You may not use recursion to implement any method other than FindCustiomerRecursive and PostOrderRecursive..
Process the following transactions in the indicated order:

1) Insert the following data in the tree using InsertBinarySearchTree: Nkwantal, 295-1492; Idle, 291-1864; Green, 295-1601; Realzola, 293-6122; Easlon, 295-1882; Bolen, 291-7890; Hedreen, 294-8075; Bell, 584-3622.

2) Find Bolen and print the phone number using FindCustomerIterative.

3) Find Bolen and print the phone number using FindCustomerRecursive.

4) Find Penton and print the phone number using FindCustomerIterative.

5) Find Penton and print the phone number using FindCustomerRecursive.

6) Starting with Easlon, traverse the entire tree in inorder back to Easlon printing the name field of each node encountered. (you should be able to do this starting at any node, i.e., find Easlon using a binary search then traverse the tree from this point in inorder).
7) Insert Altayyar, 294-1568; Gammons, 294-1882; and Whitehead 295-6622.

8) Traverse the tree in inorder starting at the root using the method InOrderSuccessor. Print the name and phone number of each node as it is encountered.

9) Traverse the tree in pre-order using "procedure PreOrderTraversalIterative(TreePoint: in BinarySearchTreePoint);."

"B" Option:

In addition to the "C" option, add a method to delete a random item from the tree leaving a binary search tree, a method ReverseInOrder, and a method preorder as described below.

DeleteRandomNode(DeletePoint: in BinarySearchTreePoint): This procedure deletes a random node from the tree. The resulting tree is a binary search tree. Note that DeletePoint = Root, DeletePoint = P.LLink or DeletePoint = P.Rlink. Management would be impressed if you minimize the number of nodes that must be examined to determine which of the above is true. Your procedure should contain comments explaining your strategy. You may add additional parameters if desired.

The procedure ReverseInOrder(treePoint: in BinarySearchTreePoint); is defined as:

1) Traverse the right subtree.
2) Visit the node (print its contents)
3) Traverse the left subtree.

You must implement ReverseInOrder recursively.

Process the following transactions after processing the "C" option transactions in the specified order:

7) Delete: Bolen, Najar, and Green. (Note they may not be in the tree.) ***Locate the nodes to be deleted using a binary search***.

8) Insert: Novak, 294-1666; and Gonzales, 295-1882.

9) Traverse the tree in inorder starting at the root using the method InOrderSuccessor. Print the name and phone number of each node as it is encountered.

10) Traverse the tree using ReverseInOrder starting at the root.

11) Traverse the tree in preorder printing the name field using an iterative routine taking advantage of the threads.

Remember, a good programmer would not allow hemorrhaging of either blood or gray matter (memory) to occur in their programs.

"A" Option:

Process the "C" option operations followed by the "B" option operations. You need not explicitly do the "C" and "B" options.

12) Print the name field traversing the tree in PostorderIterative using an iterative procedure taking advantage of the threads.
13) Print the name field traversing the tree in PostorderRecursive using a recursive procedure.

You must use recursion or iteration to implement methods as specified by the "C" and "B" options. Make the package generic as indicated below. Use the following definitions in your main program and assume the sort field is the customer name, Name.

```
type String10 is String(1..10);
type Customer is
        record Name:  String10;  PhoneNumber: String10; end record;
```

Note the main program must supply overloads for ">", "<" and "=" for use by the BinarySearchTree package to sort by Name. Name will correspond to the generic parameter Akey and Customer will correspond to the generic parameter BinarySearchTreeRecord.

package MySearchTree is new BinarySearchTree(Name, Customer, <, >, =);

****

```
generic
        type Akey is private;
        type BinarySearchTreeRecord is private;
        -- These functions compare two nodes in the tree.
        with function "<"(TheKey:  in Akey;  ARecord: in BinarySearchTreeRecord)
                return Boolean;  -- Is TheKey less than the key of ARecord?
        with function ">"(TheKey:  in Akey;  ARecord: in BinarySearchTreeRecord)
                return Boolean;
        with function "="(TheKey:  in Akey;  ARecord: in BinarySearchTreeRecord)
                return Boolean;  --Is TheKey equal to the key of ARecord?

package BinarySearchTree is
        type BinarySearchTreePoint is limited private;
        -- Inserts the node in sorted order base on overloads of "<", and ">".

        -- Points to a node in a binary search tree.
        type BinarySearchTreePoint is limited private;

        -- This procedure inserts a node (customer) into the tree in search tree order. If a customer with
        -- duplicate name already customer exist, the new customer should be inserted so they would
        -- appear "after" the older customer when the tree is traversed in inorder.
        -- The tree must be threaded in "inorder."  The search to locate the position for the new
        -- record must be iterative!
        procedure InsertBinarySearchTree(Root:  in out BinarySearchTreePoint;
                                         custName: in String10; custPhone: String10 );

        -- This procedure locates a customer using a binary search.  A pointer is returned to the
        -- customer record if they exist, otherwise a Null pointer is returned (in CustomerPoint).
        -- The search must be implemented iteratively.
        procedure FindCustomerIterative(Root: in BinarySearchTreePoint;
                                        CustomerName:  in String10;
                                        CustomerPoint:  out BinarySearchTreePoint);
```

```
        -- This procedure locates a customer using a binary search.  A pointer is returned to the
        -- customer record if they exist, otherwise a Null pointer is returned (in CustomerPoint).
        -- The search must be implemented recursively.
        procedure FindCustomerRecursive(Root: in BinarySearchTreePoint;
                                        CustomerName:  in String10;
                                        CustomerPoint:  out BinarySearchTreePoint);


        -- This function returns the address of the next node in "inorder."  The user may enter the
        -- tree at any random location.  This is sometimes called an iteration function
        -- (no recursion).
        function InOrderSuccessor(TreePoint: in BinarySearchTreePoint)
                return BinarySearchTreePoint;

        -- Access functions to return customer names and phone numbers.
        function CustomerName(TreePoint: in BinarySearchTreePoint) return String10;
        function CustomerPhone(TreePoint: in BinarySearchTreePoint) return String10;


        -- Iterative procedure utilizing threads that prints the name fields of the tree in preorder
        -- from within the procedure as it traverses the nodes.
        procedure PreOrder(TreePoint: in out BinarySearchTreePoint);

        --Procedure to traverse the tree utilizing threads printing the name fields.
        --You may assume traversal will always start at the root.
        procedure PostOrderIterative(TreePoint: in out BinarySearchTreePoint);
        procedure PostOrderRecursive(TreePoint: in out BinarySearchTreePoint);


private

        type Node;
        type BinarySearchTreePoint is access Node;
        type Node is
                record
                        Llink, Rlink:  BinarySearchTreePoint;
                        Ltag, Rtag:  Boolean;  -- True indicates pointer to lower level, False a thread.
                        Info:  BinarySearchTreeRecord;
                end record;
end BinarySearchTree;
```

Specific Grading Checks all Options:

1) **InsertBinarySearchTree(--) is iterative.**
2) **FindCustomerIterative(--) is a binary search accomplished iteratively.**
3) **FindCustomerRecursive(--) is a binary search accomplished recursively.**
4) **InOrderSuccesor(--) is iterative.**
5) **PreOrder(--) is recursive.**
6) **PostOrderIterative(--) is iterative.**
7) **PostOrderRecursive(--) is recursive.**
8) **ReverseInOrder(--) is recursive.**
9) **PreOrderTraversalIterative(TreePoint: in BinarySearchTreePoint) uses a programmer defined stack with push and pop. See notes for algorithm.**