Christopher McDaniel

COSC 3319.01

Submitted: 25 November 2019

Grading Option: "**A**"

----- 'C' Option Transactions-----

Step 1:
New root node Moutafis inserted
Ikerd inserted
Gladwin inserted
Robson inserted
Dang inserted
Bird inserted
Harris inserted
Ortiz inserted

Step 2:
Ortiz 584-3622

Step 3:
Ortiz 584-3622

Step 4:
Penton not in tree.

Step 5:
Penton not in tree.

Step 6:
In Order traversal starting at Ikerd:

Name of Customer: Bird Phone Number: 291-7890

Name of Customer: Dang Phone Number: 295-1882

Name of Customer: Gladwin Phone Number: 295-1601

Name of Customer: Harris Phone Number: 294-8075

Name of Customer: Ikerd Phone Number: 291-1864

Name of Customer: Moutafis Phone Number: 295-1492

Name of Customer: Ortiz Phone Number: 584-3622

Name of Customer: Robson Phone Number: 293-6122

Step 7:
Avila inserted
Quijada inserted
Villatoro inserted

Step 8:
In Order traversal:

Name of Customer: Avila Phone Number: 294-1568

Name of Customer: Bird Phone Number: 291-7890

Name of Customer: Dang Phone Number: 295-1882

Name of Customer: Gladwin Phone Number: 295-1601

Name of Customer: Harris Phone Number: 294-8075

Name of Customer: Ikerd Phone Number: 291-1864

Name of Customer: Moutafis Phone Number: 295-1492

Name of Customer: Ortiz Phone Number: 584-3622

Name of Customer: Quijada Phone Number: 294-1882

Name of Customer: Robson Phone Number: 293-6122

Name of Customer: Villatoro Phone Number: 295-6622

Step 9:
Pre order traversal:

Name of Customer: Moutafis Phone Number: 295-1492

Name of Customer: Ikerd Phone Number: 291-1864

Name of Customer: Gladwin Phone Number: 295-1601

Name of Customer: Dang Phone Number: 295-1882

Name of Customer: Bird Phone Number: 291-7890

Name of Customer: Avila Phone Number: 294-1568

Name of Customer: Harris Phone Number: 294-8075

Name of Customer: Robson Phone Number: 293-6122

Name of Customer: Ortiz Phone Number: 584-3622

Name of Customer: Quijada Phone Number: 294-1882

Name of Customer: Villatoro Phone Number: 295-6622

Step 10:
Post Order Iterative traversal:

Name of Customer: Avila Phone Number: 294-1568

Name of Customer: Bird Phone Number: 291-7890

Name of Customer: Dang Phone Number: 295-1882

Name of Customer: Harris Phone Number: 294-8075

Name of Customer: Gladwin Phone Number: 295-1601

Name of Customer: Ikerd Phone Number: 291-1864

Name of Customer: Quijada Phone Number: 294-1882

Name of Customer: Ortiz Phone Number: 584-3622

Name of Customer: Villatoro Phone Number: 295-6622

Name of Customer: Robson Phone Number: 293-6122

Name of Customer: Moutafis Phone Number: 295-1492

----- 'B' Option Transactions-----

Step 7:
Robson deleted
Moutafis deleted
Ikerd not in tree.

Step 8:
Poudel inserted

----------BinarySearchTree.ads----------

```
generic
   type Akey is private;
   type BinarySearchTreeRecord is private;

   --These functions compare two nodes in the tree.
   with function "<"(TheKey:  in Akey;  ARecord: BinarySearchTreeRecord)
                  return Boolean;  --Is TheKey less than the key of ARecord?

   with function ">"(TheKey:  in Akey;  ARecord: BinarySearchTreeRecord)
                  return Boolean; --Is TheKey greater than the key of ARecord?

   with function "="(TheKey:  in Akey;  ARecord: in BinarySearchTreeRecord)
                  return Boolean;  --Is TheKey equal to the key of ARecord?

package BinarySearchTree is
   subtype string10 is string(1..10);

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

   --Procedure that initializes the tree.
   Procedure InitializeTree;

   --This procedure inserts a node (customer) into the tree in search tree order.
  procedure InsertBinarySearchTree(CustomerName: string10; CustomerPhone: string10);

   --This procedure locates a customer using a binary search.
   function FindCustomerIterative(CustomerName: string10) return BinarySearchTreePoint;

   --This procedure locates a customer using a binary search.
   function FindCustomerRecursive(CustomerName: string10) return BinarySearchTreePoint;

   --This is sometimes called an iteration function (no recursion).
   procedure InOrder(CustomerName: string10);

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

   --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 BinarySearchTreePoint);

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

   --Procedure that traverses the tree starting at the root.
   procedure ReverseInOrder (TreePoint: BinarySearchTreePoint);

   --Procedure that deletes a random node within the tree.
   procedure DeleteRandomNode(CustomerName: string10);
```

```
   --Function that returns CustomerName into the tree.
   function CustomerName(TreePoint : in BinarySearchTreePoint) return string10;

   --Function that returns CustomerPhone into the tree.
   function CustomerPhone(TreePoint : in BinarySearchTreePoint) return string10;

   --Function that finds the InOrder Successor in a threaded binary tree.
   function InOrderSuccessor(TreePoint: in BinarySearchTreePoint) return
BinarySearchTreePoint;

   --Function that finds the InOrder Predecessor in a threaded binary tree.
   function InOrderPredecessor(TreePoint: in BinarySearchTreePoint) return
BinarySearchTreePoint;

   function Root return BinarySearchTreePoint;

   --Displays customer information.
   procedure DisplayNameNumber(TreePoint: in BinarySearchTreePoint);

private
   type Node;
   type Customer is
      record
         CustomerName : string10;
         CustomerPhone : string10;
      end record;

   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; --Info : BinarySearchTreeRecord
      end record;

end BinarySearchTree;
```

----------GenericStack.ads----------

```
with Ada.Text_IO; use Ada.Text_IO;
with Unchecked_Conversion;

generic
   maxStack : natural := 5; --Max size of stack.
   type MyType is private; --Type that goes into stack.

package GenericStack is
   procedure setMax(newMax: in natural); --setMax procedure definition.
   procedure push(x: in MyType); --Push procedure defintion.
   function pop return MyType; --Pop function definition.
   function peek return MyType; --Peek function defintion.
   function isEmpty return Boolean; --Empty function defintion.
   function isFull return Boolean; --Full function defintion.
   function isOverflow return Boolean; --Overflow function defintion.
   function isUnderflow return Boolean; --Underflow function defintion.
end GenericStack;
```

----------BinarySearchTree.adb----------

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Unchecked_Deallocation;
with GenericStack;

package body BinarySearchTree is
   knt : integer := 1;
   top : BinarySearchTreePoint;
   ReqPoint : BinarySearchTreePoint;

   --Creates procedure to free dynamically allocated memory and place on available
   --storage list for later use.
   procedure Free is new Ada.Unchecked_Deallocation(Node, BinarySearchTreePoint);

   --Procedure that initializes the tree.
   procedure InitializeTree is
   begin
      top := new node;
      top.Llink := top;
      top.Rlink := top;
      top.Ltag := false;
      top.Rtag := true;
      ReqPoint := top.Llink;
   end InitializeTree;

   --This procedure inserts and allocates a node (customer) into the tree in
   --search tree order.
   --Page 94 of DS notes.
   procedure InsertBinarySearchTree(CustomerName: string10; CustomerPhone: string10) is
      P : BinarySearchTreePoint;
      Q : BinarySearchTreePoint;

   begin
      P := new node;
      P.Info.CustomerName := CustomerName;
      P.Info.CustomerPhone := CustomerPhone;

      --If empty insert as the root of the tree.
      if "=" (top.Llink, top) then
         put("New root node ");
         P.Llink := top;
         P.Rlink := top;
         P.Ltag := false;
         P.Rtag := false;
         top.Llink := P;
         top.Ltag := true;

         put(CustomerName);
         put(" inserted");
         new_line(1);

      else --Tree is not empty. Locate a match with existing node or position to insert new node.
         put(CustomerName);
         put(" inserted");
```

```
      new_line(1);
      Q := top.Llink;

      loop --Search left and right for a match or insert in tree if not found.
        if "<" (CustomerName, Q.Info.CustomerName) then  --Search to left.
          if Q.LTag /= false then
            Q := Q.Llink;
          else --Insert node as left subtree.
            P.Llink := Q.Llink;
            P.Ltag := Q.Ltag;
            Q.Llink := P;
            Q.Ltag := true;
            P.Rlink := Q;
            P.Rtag := false;
            exit; --New node inserted.
          end if;
        elsif ">" (CustomerName, Q.Info.CustomerName) then --Search to right.
          if Q.RTag /= false then
            Q := Q.Rlink;
          else --Insert node as right subtree.
            P.Rlink := Q.Rlink;
            P.Rtag  := Q.Rtag;
            Q.Rlink := P;
            Q.Rtag := true;
            P.Llink := Q;
            P.Ltag := false;
            exit; --New node inserted.
          end if;
          --Implies CustomerName matches Q.Info.CustomerName.
        elsif "=" (CustomerName, Q.Info.CustomerName) then --Search to right.
          if Q.RTag /= false then
            Q := Q.Rlink;
          else --Insert as right subtree of the equal node.
            P.Rlink := Q.Rlink;
            P.Rtag  := Q.Rtag;
            Q.Rlink := P;
            Q.Rtag := true;
            P.Llink := Q;
            P.Ltag := false;
            exit; --New node inserted.
          end if;
        end if;
      end loop;
    end if;
    knt := knt + 1;
end InsertBinarySearchTree;

--This function iteratively locates a customer using a binary search.
--Page 94 of DS Notes.
function FindCustomerIterative(CustomerName: string10) return BinarySearchTreePoint is
    P : BinarySearchTreePoint;
    Customer1 : Customer;

begin
```

```
  Customer1.CustomerName := CustomerName;
  Customer1.CustomerPhone := "          ";
  P := top.Llink;


  loop --Search left and right for a match
    if (CustomerName < P.Info.CustomerName) then --Search to left.
      if P.Ltag = true then
        P := P.Llink;
      else --Missing Customer.
        put(CustomerName);
        put_line("not in tree.");
        P := new Node;
        P.Info := Customer1;
        return P;
      end if;
    elsif (CustomerName > P.Info.CustomerName) then --Search to right
      if P.Rtag = true then
        P := P.Rlink;
      else --Missing Customer.
        put(CustomerName);
        put_line("not in tree.");
        P := new Node;
        P.Info := Customer1;
        return P;
      end if;
    elsif (CustomerName = P.Info.CustomerName) then --Implies match.
      return P;
    else --Missing Customer.
      put(CustomerName);
      put_line("not in tree.");
      P := new Node;
      P.Info := Customer1;
      return P;
    end if;
  end loop;
end FindCustomerIterative;

--This function locates a customer using a binary search.
--Page 82 of DS Notes
function FindCustomerRecursive(CustomerName: string10) return BinarySearchTreePoint is
  P : BinarySearchTreePoint;
  Customer1 : Customer;

begin
  Customer1.CustomerName := CustomerName;
  Customer1.CustomerPhone := "          ";

  if ReqPoint = top then
    ReqPoint := top.Llink;
  end if;

  loop --Search left and right for a match
    if (CustomerName < ReqPoint.Info.CustomerName) then --Search to left.
```

```
        if ReqPoint.Ltag = true then
          ReqPoint := ReqPoint.Llink;
          return FindCustomerRecursive(CustomerName);
        else --Missing Customer.
          put(CustomerName);
          put_line("not in tree.");
          P := new node;
          P.Info := Customer1;
          return P;
        end if;
      elsif (CustomerName > ReqPoint.Info.CustomerName) then --Search to right
        if ReqPoint.Rtag = true then
          ReqPoint := ReqPoint.Rlink;
          return FindCustomerRecursive(CustomerName);
        else --Missing Customer.
          put(CustomerName);
          put_line("not in tree.");
          P := new node;
          P.Info := Customer1;
          return P;
        end if;
      else
        return ReqPoint;
      end if;
    end loop;
end FindCustomerRecursive;


--This is sometimes called an iteration function (no recursion).
--Page 87 and 88 of DS notes.
procedure InOrder(CustomerName: string10) is
   Q : BinarySearchTreePoint;

begin
   Q := FindCustomerIterative(CustomerName);

   while Q.Ltag = true --Go left as far as possible.
   loop
      Q := Q.Llink;
   end loop;

   for I in 1..knt - 1
   loop
      DisplayNameNumber(Q);
      Q := InOrderSuccessor(Q);
      --Q := InOrderPredecessor(Q);
   end loop;
end InOrder;


--PreOrder traversal of a tree using a stack allocated
--explicitly by the programmer.
--Page 81 and Page 88 of DS Notes
procedure PreOrderTraversalIterative (TreePoint: in BinarySearchTreePoint) is
```

```
   P : BinarySearchTreePoint;

   package stack is new GenericStack(20, BinarySearchTreePoint);

begin
   P := TreePoint;

   loop
      if P /= top then --Visit
         DisplayNameNumber(P);
         stack.push(P);
      end if;

      --Thread
      if P.Ltag = true then
         exit when P.Llink = TreePoint;
         P := P.Llink; --Left
      else
         while P.Rtag = false
         loop
            P := P.Rlink;
         end loop;
         if P.Rlink = TreePoint then
            exit;
         end if;
         P := P.Rlink; --Right
      end if;
   end loop;
end PreOrderTraversalIterative;

--Iterative procedure utilizing threads that prints the name fields of the tree in
--preorder from within the procedure as it traverses the nodes.
--Page 88 of DS Notes
procedure PreOrder (TreePoint: in BinarySearchTreePoint) is
   T : BinarySearchTreePoint;

begin
   T := TreePoint;

   loop
      exit when T = top;
      put(T.Info.CustomerName);
      new_line(1);
      if T.Ltag = true then -- = "+"
         T := T.Llink;
      else
         while T.Rtag /= true -- <> "+"
         loop
            T := T.Rlink;
         end loop;
         T := T.Rlink;
      end if;
   end loop;
end PreOrder;
```

```
--Procedure to traverse the tree utilizing threads printing the name fields.
--You may assume traversal will always start at the root.
--Page 86 and page 88 in DS Notes.
procedure PostOrderTraversalIterative(TreePoint: in BinarySearchTreePoint) is
   P : BinarySearchTreePoint;

   type RecordStack is record
      Point: BinarySearchTreePoint;
      Number: natural range 0 .. 1;
   end record;

   ARecord: RecordStack;
   D : natural range 0 .. 1;
   I : natural := 0;

   package gStack is new GenericStack(5, RecordStack); --Stack

begin
   P := TreePoint;
   loop
      if P /= null then
         ARecord.Point := P; --A <= (P, 0)
         ARecord.Number := 0;
         gStack.push(ARecord); --Push onto stack.

         if P.Ltag = false then
            P := null;
         else
            P := P.Llink; --Left
         end if;
      else
         exit when gStack.isEmpty = true; --The algorithm terminates.
         ARecord := gStack.pop; --Pop from stack.
         P   := ARecord.Point;
         D   := ARecord.Number;

         if D = 0 then
            ARecord.Point := P; --A <= (P, 1)
            ARecord.Number := 1;
            gStack.push(ARecord); --Push onto stack.

            --Thread
            if P.Rtag = false then
               P := null;
            else
               P := P.Rlink; --Right
            end if;
         else
            loop
               DisplayNameNumber(P);
               I := I + 1;

               if I = knt - 1 then
```

```
            return;
          end if;

          exit when gStack.isEmpty = true; --The algorithm terminates.
          ARecord := gStack.pop; --Pop from stack.
          P := ARecord.Point; --A <= (P, D)
          D := ARecord.Number;

          if D = 0 then
            ARecord.Point := P; --A <= (P, 1)
            ARecord.Number := 1;
            gStack.push(ARecord); --Push onto stack.

            --Thread
            if P.Rtag = false then
              P := null;
            else
              P := P.Rlink; --Visit
            end if;
          end if;
          exit when D = 0;
        end loop;
      end if;
    end if;
  end loop;
end PostOrderTraversalIterative;

--Procedure to traverse the tree utilizing threads printing the name fields.
--You may assume traversal will always start at the root.
--Page 86 and page 88 of DS Notes
procedure PostOrderRecursive(TreePoint: in BinarySearchTreePoint) is
  type RecordStack is record
    Point : BinarySearchTreePoint;
    Number : natural range 0 .. 1;
  end record;

  package postStack is new GenericStack (5, RecordStack); --Stack
  I : natural := 0;
  PostOrderDone : Boolean := false;

  P : BinarySearchTreePoint;
  ARecord : RecordStack;
  D : natural range 0..1;

  --Procedure to loop until PostOrder stack is reset to 0.
  procedure Reset is
    Dispose : RecordStack;
  begin
    I := 0;

    if postStack.isEmpty = false then
      while postStack.isEmpty = false
      loop
        Dispose := postStack.pop; --Pop from stack.
```

```
        end loop;
      end if;
    end Reset;

begin
    P := TreePoint;
    loop
      if P /= null then
        ARecord.Point := P; --A <= (P, 0)
        ARecord.Number := 0;
        postStack.push(ARecord);  --Push onto stack.

        if P.Ltag = false then
          PostOrderRecursive(null);
          if PostOrderDone = true then
            Reset;
            return;
          end if;
        else
          PostOrderRecursive(P.Llink); --Left
          if PostOrderDone = true then
            Reset;
            return;
          end if;
        end if;
      else
        exit when postStack.isEmpty = true; --The algorithm terminates.
        ARecord := postStack.pop; --Pop from stack.
        P := ARecord.Point;
        D := ARecord.Number;

        if D = 0 then
          ARecord.Point := P; --A <= (P, 1)
          ARecord.Number := 1;
          postStack.push(ARecord);  --Push onto stack.

          --Thread
          if P.Rtag = false then
            PostOrderRecursive(null);
            if PostOrderDone = true then
              return;
            end if;
          else
            PostOrderRecursive(P.Rlink); --Right
            if PostOrderDone = true then
              return;
            end if;
          end if;
        else
          loop
            DisplayNameNumber(P);
            I := I + 1;

            if I = knt - 1 then
```

```
            PostOrderDone := true;
            Reset;
            return;
          end if;

          exit when postStack.isEmpty = true; --The algorithm terminates.
          ARecord := postStack.pop; --Pop from stack.
          P := ARecord.Point; --A <= (P, D)
          D := ARecord.Number;

          if D = 0 then
            ARecord.Point := P; --A <= (P, 1)
            ARecord.Number := 1;
            postStack.push(ARecord);  --Push onto stack.

            --Thread
            if P.Rtag = false then
              PostOrderRecursive(null);
              if PostOrderDone = true then
                return;
              end if;
            else
              PostOrderRecursive(P.Rlink); --Visit
              if PostOrderDone = true then
                return;
              end if;
            end if;
          end if;
          exit when D = 0;
        end loop;
      end if;
    end if;
  end loop;
end PostOrderRecursive;

--Procedure that traverses the tree starting at the root.
--Page 81 of DS notes, just do the algorithm in reverse.
procedure ReverseInOrder (TreePoint : BinarySearchTreePoint) is
  K : Integer := 1;
  T : BinarySearchTreePoint;

begin
  T := TreePoint;

  --Right
  if T.Rtag = true then
    ReverseInOrder(T.Rlink);
    if K > knt - 1 then
      return;
    end if;
  end if;
  --Visit
  DisplayNameNumber(T);
  K := K + 1;
```

```
   if K > knt - 2 then
      return;
   end if;

   --Left
   if T.Ltag = true then
      ReverseInOrder(T.Llink);
      if K > knt - 2 then
         return;
      end if;
   end if;
end ReverseInOrder;

--Procedure that deletes a random node within the tree.
--Page 95 DS notes
--Could not get to work properly.
procedure DeleteRandomNode(CustomerName: string10) is
   T : BinarySearchTreePoint;
   Q : BinarySearchTreePoint;
   S : BinarySearchTreePoint;
   R : BinarySearchTreePoint;

begin
   --Find head node.
   Q := FindCustomerIterative(CustomerName);
   R := top.Llink;

   --Blank phone number implies it was not found.
   if Q.Info.CustomerPhone = "          " then
      return;
   end if;
   --Begin deletion.
   T := Q;
   S := Q;

   if S /= Root then
      while S.Llink /= Q and S.Rlink /= Q
      loop
         if S = top then
            S := top.Llink; --Head node.
         else
            S := InOrderSuccessor(S); --Find successor in InOrder.
         end if;
      end loop;
   else
      S := top;
   end if;

   if T.Rtag = false then --Found the successor in InOrder.
      if Q = T.Llink then  --Search to the left looking for null Llink.
         if Q = top then
            Q := top.Llink; --Head node.
         end if;
      end if;
```

```
    else
      if T.Ltag = false then --Easy delete if Llink = Null.
        if Q = T.Rlink then
          if Q = top then
            Q := top.Llink;
          end if;
        end if;
      end if;
    end if;

    if S = top then
      top.Llink := Q;
    else --Assumes first node in the left or right subtree have been deleted.
      if T = S.Llink then
        S.Llink := Q; --Left.
        S.Ltag := T.Ltag;
      else
        S.Rlink := Q; --Right.
        S.Rtag := T.Rtag;
      end if;
    end if;

    Free(T); --Prevent memory hemorrhaging.
    knt := knt - 1; --Decrement the number of nodes.

    put(CustomerName);
    put("deleted");
    new_line(1);
  end DeleteRandomNode;

  --Function that returns CustomerName into the tree.
  function CustomerName(TreePoint: in BinarySearchTreePoint) return string10 is
  begin
    return TreePoint.Info.CustomerName;
  end CustomerName;

  --Function that returns CustomerPhone into the tree.
  function CustomerPhone(TreePoint: in BinarySearchTreePoint) return string10 is
  begin
    return TreePoint.Info.CustomerPhone;
  end CustomerPhone;

  --Function that finds the InOrder Successor in a threaded binary tree.
  --Page 87 in DS Notes.
  function InOrderSuccessor(TreePoint: in BinarySearchTreePoint) return
BinarySearchTreePoint is
    Q : BinarySearchTreePoint;

  begin
    Q := TreePoint.Rlink; --Looks right.

    if Q = top then
      while Q.Ltag = true
      loop
```

```
        Q := Q.Llink;
      end loop;
    end if;

    --If it's a thread.
    if TreePoint.Rtag = false then
      return Q; --TreePoint points to the InOrder successor.
    else --Search to left.
      while Q.Ltag = true
      loop
        Q := Q.Llink;
      end loop;
      return Q;
    end if;
  end InOrderSuccessor;

  --Function that finds the InOrder Predecessor in a threaded binary tree.
  --Page 88 in DS Notes.
  function InOrderPredecessor(TreePoint: in BinarySearchTreePoint) return
BinarySearchTreePoint is
    Q : BinarySearchTreePoint;

  begin
    Q := Treepoint.Llink; --Looks left.

    if Q = top then
      while Q.Rtag = true
      loop
        Q := Q.Rlink;
      end loop;
    end if;

    --If it's a thread.
    if TreePoint.Ltag = false then
      return Q; --TreePoint points to the InOrder Predecessor.
    else --Search to Right.
      while Q.Rtag = true
      loop
        Q := Q.Rlink;
      end loop;
      return Q;
    end if;
  end InOrderPredecessor;

  function Root return BinarySearchTreePoint is
  begin
    return top.Llink;
  end Root;

  --Displays customer information.
  procedure DisplayNameNumber(TreePoint: in BinarySearchTreePoint) is
  begin
    new_line(1);
    put("Name of Customer: ");
```

```
      put(TreePoint.Info.CustomerName);
      put("   Phone Number: ");
      put(TreePoint.Info.CustomerPhone);
      new_line(1);
   end DisplayNameNumber;

begin
   InitializeTree;
end BinarySearchTree;
```

----------GenericStack.adb----------

```
package body GenericStack is
   max : natural := maxStack;
   stack : array(1..max) of MyType;
   top : natural range 0..max := 0; --Initialize top of the stack to 0.
   overflow : Boolean := false;
   underflow : Boolean := false;

   procedure setMax(newMax: in natural) is
   begin
      max := newMax;
   end setMax;

   --Page 14 of DS Notes.
   procedure push(x: in MyType) is
   begin
      if (top < max) then
         underflow := false;
         top := top + 1;
         stack(top) := x; --Insert item x into stack.
      else
         overflow := true; --Error condition.
         new_line(1);
         put_line("Overflow has occured, the stack is full");
      end if;
   end push;

   --Page 14 of DS Notes.
   function pop return MyType is
      blank : MyType;
   begin
      if (top > 0) then
         overflow := false;
         top := top - 1;
         return stack(top + 1); --Pop the stack
      else
         underflow := true; --Desired condition.
         new_line(1);
         put_line("Underflow has occured, the stack is empty");
         return blank;
      end if;
   end pop;

   function peek return MyType is
      blank : MyType;
   begin
      if isEmpty then
         return blank;
      else
         return stack(top);
      end if;
   end peek;

   function isEmpty return Boolean is
```

```
  begin
    if (top = 0) then
      return true;
    else
      return false;
    end if;
  end isEmpty;

  function isFull return Boolean is
  begin
    if (top = max) then
      return true;
    else
      return false;
    end if;
  end isFull;

  function isOverflow return Boolean is
  begin
    if (overflow) then
      return true;
    else
      return false;
    end if;
  end isOverflow;

  function isUnderflow return Boolean is
  begin
    if (underflow) then
      return true;
    else
      return false;
    end if;
  end isUnderflow;

end GenericStack;
```

----------Main.adb----------

```
with Ada.Text_IO; use Ada.Text_IO;
with BinarySearchTree;

procedure Main is
   type string10 is new string(1..10);
   type Customer is
      record
         Name : string10;
         Phone : string10;
      end record;

   --Is TheKey less than the key of ARecord?
   function "<"(TheKey: in string10; ARecord: Customer) return Boolean is
   begin
      if TheKey < ARecord.Name then
         return true;
      else
         return false;
      end if;
   end "<";
   --Is TheKey greater than the key of ARecord?
   function ">"(TheKey: in string10; ARecord: Customer) return Boolean is
   begin
      if TheKey > ARecord.Name then
         return true;
      else
         return false;
      end if;
   end ">";
   --Is TheKey equal to the key of ARecord?
   function "="(TheKey: in string10; ARecord: in Customer) return Boolean is
   begin
      if TheKey = ARecord.Name then
         return true;
      else
         return false;
      end if;
   end "=";

   package BSTree is new BinarySearchTree(string10, Customer, "<", ">", "=");
begin
   --Start of C Option
   put_line("----- 'C' Option Transactions-----");
   new_line(2);

   --Step 1
   put_line("Step 1:");
   BSTree.InsertBinarySearchTree("Moutafis  ", "295-1492  ");
   BSTree.InsertBinarySearchTree("Ikerd     ", "291-1864  ");
   BSTree.InsertBinarySearchTree("Gladwin   ", "295-1601  ");
   BSTree.InsertBinarySearchTree("Robson    ", "293-6122  ");
   BSTree.InsertBinarySearchTree("Dang      ", "295-1882  ");
   BSTree.InsertBinarySearchTree("Bird      ", "291-7890  ");
```

```
BSTree.InsertBinarySearchTree("Harris    ", "294-8075  ");
BSTree.InsertBinarySearchTree("Ortiz     ", "584-3622  ");
new_line(1);

--Step 2
put_line("Step 2:");
put("Ortiz     ");
put_line(BSTree.CustomerPhone(BSTree.FindCustomerIterative("Ortiz     ")));
new_line(1);

--Step 3
put_line("Step 3:");
put("Ortiz     ");
put_line(BSTree.CustomerPhone(BSTree.FindCustomerRecursive("Ortiz     ")));
new_line(1);

--Step 4
put_line("Step 4:");
put_line(BSTree.CustomerPhone(BSTree.FindCustomerIterative("Penton    ")));
new_line(1);

--Step 5
put_line("Step 5:");
put_line(BSTree.CustomerPhone(BSTree.FindCustomerRecursive("Penton    ")));
new_line(1);

--Step 6
put_line("Step 6:");
put_line("In Order traversal starting at Ikerd: ");
BSTree.InOrder("Ikerd     ");
new_line(1);

--Step 7
put_line("Step 7:");
BSTree.InsertBinarySearchTree("Avila     ", "294-1568  ");
BSTree.InsertBinarySearchTree("Quijada   ", "294-1882  ");
BSTree.InsertBinarySearchTree("Villatoro ", "295-6622  ");
new_line(1);

--Step 8
put_line("Step 8:");
put_line("In Order traversal: ");
BSTree.InOrder(BSTree.CustomerName(BSTree.Root));
new_line(1);

--Step 9
put_line("Step 9:");
put_line("Pre order traversal: ");
BSTree.PreOrderTraversalIterative(BSTree.Root);
new_line(2);

--Step 10
put_line("Step 10:");
put_line("Post Order Iterative traversal: ");
```

```
   BSTree.PostOrderTraversalIterative(BSTree.Root);
   new_line(1);

   --Start of B Option
   put_line("----- 'B' Option Transactions-----");
   new_line(2);

   --Step 7
   --Delete would not work properly...
   put_line("Step 7:");
   BSTree.DeleteRandomNode("Robson    ");
   BSTree.DeleteRandomNode("Moutafis  ");
   BSTree.DeleteRandomNode("Ikerd     ");
   new_line(1);

   --Step 8
   put_line("Step 8:");
   BSTree.InsertBinarySearchTree("Poudel    ", "294-1666  ");
   BSTree.InsertBinarySearchTree("Spell     ", "295-1882  ");
   new_line(1);

   --Step 9
   put_line("Step 9:");
   put_line("In Order traversal: ");
   BSTree.InOrder(BSTree.CustomerName(BSTree.Root));
   new_line(1);

   --Step 10
   put_line("Step 10:");
   put_line("Reverse In Order traversal: ");
   BSTree.ReverseInOrder(BSTree.Root);
   new_line(1);

   --Step 11
   put_line("Step 11:");
   put_line("Pre order traversal utilizing threads: ");
   BSTree.PreOrder(BSTree.Root);
   new_line(2);

   --Start of A option
   put_line("----- 'A' Option Transactions-----");
   new_line(2);

   --Step 13
   put_line("Step 13:");
   put_line("Post Order Recursive traversal: ");
   BSTree.PostOrderRecursive(BSTree.Root);
   new_line(1);

   put_line("End of Program!");
   new_line(1);
end Main;
```