```
(*********************************************************)
(* Alejo Alamillo                COSC 055                  *)
(* Spring 1991                   Assignment: TreeMod       *)
(*********************************************************)


(*************************************************************)
(*  Several procedures for binary tree processing are        *)
(*  contained in the module below.                           *)
(*  This is NOT a complete module ready to link up with a    *)
(*  driver program.   PRB 10/90    *)
(*************************************************************)
PROGRAM TreeMod (Input,Output);

  TYPE  DataType = Real;
   NodePointer = ^NodeRec;
   NodeRec= RECORD
    Data: DataType;
    Level: 0..Maxint;
    LeftLink,
    RightLink: NodePointer;
  END;

  VAR   Head: NodePointer;
   SEED : Integer;
   DataIn: DataType;

(*****************************************************)
(*  Generates a random number ( 0 <= R < 1 )         *)
(*  SEED  must be initialized ONCE before using    *)
(*****************************************************)
FUNCTION Random (VAR SEED : Integer): Real;
  CONST Modulus = 65536;
   Multiplier = 25173;
   Increment = 13849;

  BEGIN
  SEED :=((Multiplier*SEED ) + Increment) MOD Modulus;
  Random:= SEED /Modulus;
  END;
(*****************************************************)
(* Disposes of the nodes of an existing, unneeded  *)
(* Tree.  Recursively called in postorder.   *)
(*****************************************************)
PROCEDURE DisposeTree (VAR AquiNode:NodePointer);

BEGIN
WITH AquiNode^ DO
  BEGIN
   IF LeftLink<> nil THEN
   DisposeTree (LeftLink);
    IF RightLInk<> nil THEN
DisposeTree (RightLink);
    Dispose (AquiNode);
  END;
END;
(*****************************************************)
(*  Recursively searches for node to insert DataIn *)
(*  Inserts data DataIn into a tree in order  *)
(*****************************************************)
PROCEDURE AddNode (VAR Aqui: NodePointer; DataIn: DataType;
                                  AquiLevel:  Integer);
BEGIN
IF Aqui = nil THEN(* Place is found *)
```

```
  BEGIN
  New(Aqui);
  Aqui^.Data:= DataIn;
  Aqui^.Level:= AquiLevel;
  Aqui^.LeftLink:= nil;
  Aqui^.RightLink:= nil;
  END
ELSE(* Search farther *)
  IF (DataIn < Aqui^.Data) THEN
    AddNode (Aqui^.LeftLink, DataIn, AquiLevel+1)
  ELSE
    AddNode (Aqui^.RightLink, DataIn,AquiLevel+1); (* Duplicate keys   *)
END;   (* are inserted in  *)
  (* original order   *)
(*************************************)
(* *)
(*************************************)
  PROCEDURE FormTree(VAR Head:NodePointer);
    CONST NumberofNodes = 15;
    VAR I: Integer;
   DataIn: DataType;
  (*************************************)
  (*  Currently randomly generated*)
  (*************************************)
  PROCEDURE GetData(VAR DataIn: DataType);
    BEGIN
    DataIn:=100*Random(SEED )+1;
    END;

BEGIN (* FormTree *)
Head:= nil;
FOR I:= 1 TO NumberofNodes DO
  BEGIN
  GetData(DataIn);
  AddNode (Head, DataIn, 0);
  END;
END;
(*********************************************)
(*  SHOWTREERecursively displays a tree    *)
(*   in L-R order.   *)
(*********************************************)
PROCEDURE SHOWTREE  (AquiNode: NodePointer);

BEGIN
WITH AquiNode^ DO
  BEGIN (* Reversed for rotated display *)
  IF RightLink<> nil THEN
    SHOWTREE  (RightLink);
  WRITELN('   ',Trunc(Data):3*(1+Level));
  IF LeftLink<>nil THEN
    SHOWTREE  (LeftLink);
  END;
END;

            (*************  MAIN *************)

BEGIN
 (* Initialize *)
 (* Describe *)

 Write('Please enter SEED  for the Random function: ');
  Readln(SEED );  WRITELN;

  FormTree(Head);
```

```
  WRITELN;  WRITELN;  WRITELN;
  SHOWTREE  (Head);
  DisposeTree (Head);
END.
```