```
unit ArbolesBinarios;

interface

Uses
Tipos, Dialogs, QueuesPointer, StackPointer, SysUtils, Variants;

Const
MIN = 1;
MAX = 2000;   // Tamaño maximo del arbol
Nulo= Nil;  // Posicion NO valida de un Nodo

Type
PosicionArbol = ^NodoArbol;

NodoArbol = Object
  Datos: TipoElemento;
  HI, HD: PosicionArbol;
End;

Arbol = Object
  Private
    Raiz: PosicionArbol;
    Q_Items: LongInt;
    TDatoDeLaClave: TipoDatosClave;
    Size: LongInt;
    Function ContarNodos(P: PosicionArbol): LongInt;
  Public
    Function Crear(avTipoClave: TipoDatosClave; alSize: LongInt): Resultado;
    Function EsVacio(): Boolean; // Sinonimo de arbol vacio
    Function EsLleno(): Boolean;
    Function RamaNula(P:PosicionArbol): Boolean; // Controla si un apuntador es nil
    Function Recuperar(P:PosicionArbol): TipoElemento;
    Function CargarArbol(): Resultado;
    Function PreOrden(): String;
    Function InOrden(): String;
    Function PostOrden(): String;
    Function Anchura(): String;
    Function PreOrdenITE(): String;
    Function Altura(): Integer;
    Function BuscarPreOrden(X:TipoElemento): PosicionArbol;
    Function Nivel(Q:PosicionArbol): LongInt;
    Function HijoIzquierdo(P:PosicionArbol): PosicionArbol;
    Function HijoDerecho(P:PosicionArbol): PosicionArbol;
    Function Padre(Hijo:PosicionArbol): PosicionArbol;
    Function CrearNodo(X:TipoElemento): PosicionArbol;
    // Propiedades del arbol
    Function DatoDeLaClave: TipoDatosClave;
    Function CantidadNodos(): LongInt;
    Function Root(): PosicionArbol;
    Function SizeTree(): LongInt;
    Function MaxSizeTree(): LongInt;
    // Propiedades de Asignacion al Arbol
    Procedure SetRoot(R:PosicionArbol);
    Procedure ConectarHI(P:PosicionArbol; Q:PosicionArbol);
    Procedure ConectarHD(P:PosicionArbol; Q:PosicionArbol);
  End;


implementation

// Crea el arbol vacio
Function Arbol.Crear(avTipoClave: TipoDatosClave; alSize: LongInt): Resultado;
Begin
  if alSize < Min then Crear:= CError;
  if alSize > Max then Crear:= CError;
  if (alSize >= Min) And (alSize <= Max) then Begin
    raiz := Nulo;
    q_items := 0;
    TDatoDeLaClave := avTipoClave;
    Size := alSize;
    Crear := OK;
```

```
  End;
End;

// Control de arbol vacio
Function Arbol.EsVacio(): Boolean; // Sinonimo de arbol vacio
Begin
  EsVacio := (raiz = Nulo);
End;

// Control de arbol lleno
Function Arbol.EsLleno(): Boolean;
Begin
  EsLleno := (q_items = Size);
End;

// Control de rama nula
Function Arbol.RamaNula(P:PosicionArbol): Boolean; // Controla si un apuntador es nil
Begin
  RamaNula := (P = Nulo);
End;

// carga un arbol nodo a nodo
Function Arbol.CargarArbol(): Resultado;
Var X: TipoElemento;
// Procedimiento que carga el arbol en preorden desde cero
  Procedure Load(Var P: PosicionArbol);
  Begin
    X.Clave := InputBox('Ingresar un Caracter', 'Ingresar Datos', '.');
    If X.Clave = '.' Then P := Nulo
    Else Begin
      New(P);
      P^.Datos := X;
      Inc(Q_Items);
      Load(P^.HI);
      Load(P^.HD);
    End;
  End;
// Inicio de la Funcion
Begin
  X.Valor2 := NIL;
  CargarArbol := CError;
  TDatoDeLaClave := Cadena;
  Crear(TDatoDeLaClave, Size);
  Load(raiz);
  CargarArbol := OK;
End;

// recupera la posicion P
Function Arbol.Recuperar(P:PosicionArbol): TipoElemento;
Var X: TipoElemento;
Begin
  Recuperar := X.TipoElementoVacio;
  If Not RamaNula(P) Then
  Begin
    Recuperar := P^.Datos;
  End;
End;

// Recorrido Pre-Orden Recursivo
Function Arbol.PreOrden(): String;
Var S: String;
  // Proceso que lee en preorden
  Procedure PreOrd(P: PosicionArbol);
  Begin
    If RamaNula(P) Then S := S + '.'
    Else Begin
      S := S + P^.Datos.ArmarString;
      PreOrd(P^.HI);
      PreOrd(P^.HD);
    End;
  End;
// Inicio de la funcion
```

```
Begin
  S := '';
  PreOrd(Raiz);
  PreOrden := S;
End;

// Recorrido IN-Orden Recursivo
Function Arbol.InOrden(): String;
Var S: String;
  // Proceso que lee en preorden
  Procedure InOrd(P: PosicionArbol);
  Begin
    If RamaNula(P) Then S := S + '.'
    Else Begin
      InOrd(P^.HI);
      S := S + P^.Datos.ArmarString;
      InOrd(P^.HD);
    End;
  End;
// Inicio de la funcion
Begin
  S := '';
  InOrd(Raiz);
  InOrden := S;
End;

// Recorrido Post-Orden Recursivo
Function Arbol.PostOrden(): String;
Var S: String;
  // Proceso que lee en preorden
  Procedure PostOrd(P: PosicionArbol);
  Begin
    If RamaNula(P) Then S := S + '.'
    Else Begin
      PostOrd(P^.HI);
      PostOrd(P^.HD);
      S := S + P^.Datos.ArmarString;
    End;
  End;
// Inicio de la funcion
Begin
  S := '';
  PostOrd(Raiz);
  PostOrden := S;
End;

// Recorre el arbol por niveles
Function Arbol.Anchura(): String;
Var S: String;
C: Cola;
Q: PosicionArbol;
X: TipoElemento;
Begin
  S := '';
  X.Clave := '';
  X.Valor2 := NIL;
  If Not EsVacio() Then Begin
    C.Crear(Cadena, Size);
    X.Valor2 := Raiz;
    C.Encolar(X);
    While Not C.EsVacia() Do Begin
      X := C.Recuperar();
      C.DesEncolar;
      Q := X.Valor2;
      S := S + Q^.Datos.ArmarString;
      // Si no es nulo encolo el hijo izq.
      If Not RamaNula(Q^.HI) Then Begin
        X.Valor2 := Q^.HI;
        C.Encolar(X);
      End;
      // Si no es nulo encolo el hijo der.
      If Not RamaNula(Q^.HD) Then Begin
```

```
        X.Valor2 := Q^.HD;
        C.Encolar(X);
      End;
    End;
  End;
  Anchura := S;
End;

// Realiza el recorrido pre-orden en forma iterativa
Function Arbol.PreOrdenITE(): String;
Var S: String;
P: Pila;
Q: PosicionArbol;
X: TipoElemento;
Begin
  S := '';
  P.Crear(Cadena, Size);
  X.Clave := '';
  X.Valor2 := NIL;
  Q := Raiz;
  While Not(P.EsVacia) OR Not(RamaNula(Q)) Do Begin // Ciclo del Loop de llamada por derecha
    While Not(RamaNula(Q)) Do begin // Simula la llamada recursiva por Izquierda
      S := S + Q^.Datos.ArmarString;
      X.Valor2 := Q;
      P.Apilar(X);
      Q := Q^.HI ;
    End;
    // Corto las llamada por izquierda. Tengo que desapilar e ir por derecha
    S := S + '.';
    X := P.Recuperar();
    Q := X.Valor2;
    P.DesApilar;
    Q := Q^.HD ;
  End;
  S := S + '.';
  PreOrdenITE := S;
End;

// Funcion Que Busca un Elemento en un Arbol Desordenado
Function Arbol.BuscarPreOrden(X:TipoElemento):PosicionArbol;
Var Pos: PosicionArbol;
  Procedure Buscar(P: PosicionArbol);
  Begin
  If Not RamaNula(P) Then Begin
    If X.Clave = P^.Datos.Clave Then Pos := P
    Else Begin
      Buscar(P^.HI);
      Buscar(P^.HD);
    End;
    End;
  End;
// Codigo de la funcion principal
Begin
  if X.TipoDatoClave(X.Clave) <> TDatoDeLaClave  then Begin
    BuscarPreOrden := Nulo;
    Exit;
  End;
  // Ahora lo Busca
  Pos := Nulo;
  Buscar(Raiz);
  BuscarPreOrden := Pos;
End;

// Sacamos la altura del Arbol usando el recorrido pre-orden
Function Arbol.Altura(): LongInt;
Var H: Integer;
  // Proceso que resuelve la altura. “C” cuenta los pasos desde la raiz a cada nodo
  Procedure Alt(P: PosicionArbol; C: Integer);
  Begin
    If RamaNula(P) Then Begin
      If C > H Then H := C; // cada vez que llega a la hoja pregunta si la cantidad de pasos fue ma
    End
```

```
    Else Begin
      Alt(P^.HI, C + 1);
      Alt(P^.HD, C + 1);
    End;
  End;
// Inicio de la funcion
Begin
  H := 0;
  Alt(Raiz, 0);
  Altura := H;
End;

// saco el Nivel de un Nodo recibiendo la posicion
Function Arbol.Nivel(Q:PosicionArbol): LongInt;
Var N: LongInt;
B: Boolean;
  Procedure Niv(P: PosicionArbol; C: LongInt);
  Begin
    If RamaNula(P) Then
    Else Begin
      If P = Q Then N := C;
      Niv(P^.HI, C + 1);
      Niv(P^.HD, C + 1);
    End;
  End;
// Codigo de la funcion principal
Begin
  N := 0;
  B := False;
  Niv(Raiz, 0);
  Nivel := N;
End;

// Retorna la posicion del padre de un nodo o NULO
Function Arbol.Padre(Hijo:PosicionArbol): PosicionArbol;
Var Pad: PosicionArbol;
  Procedure BuscaPadre(P: PosicionArbol);
  Begin
  If Not RamaNula(P) Then Begin
    If Not RamaNula(P^.HI) Then Begin
      If P^.HI = Hijo Then Pad := P
    End;
    If Not RamaNula(P^.HD) Then Begin
      If P^.HD = Hijo Then Pad := P
    End;
    BuscaPadre(P^.HI);
    BuscaPadre(P^.HD);
  End;
  End;
// codigo de la funcion principal
Begin
  Pad := Nulo;
  BuscaPadre(Raiz);
  Padre := Pad;
End;

// Retorna el Hijo Izquierdo de un Nodo
Function Arbol.HijoIzquierdo(P:PosicionArbol): PosicionArbol;
Begin
  HijoIzquierdo := Nulo;
  If Not RamaNula(P) Then HijoIzquierdo := P^.HI;
End;

// Retorna el Hijo Derecho de un Nodo
Function Arbol.HijoDerecho(P:PosicionArbol): PosicionArbol;
Begin
  HijoDerecho := Nulo;
  If Not RamaNula(P) Then HijoDerecho := P^.HD;
End;

// crea un nodo del arbol
Function Arbol.CrearNodo(X:TipoElemento): PosicionArbol;
```

```
Var P: PosicionArbol;
Begin
  New(P);
  P^.Datos := X;
  P^.HI := Nulo;
  P^.HD := Nulo;
  CrearNodo := P;
End;

// Retorno como propiedad la cantidad de nodos del arbol
Function Arbol.CantidadNodos(): LongInt;
Begin
  CantidadNodos := Q_Items;
End;

// Retorno la raiz del arbol
Function Arbol.Root(): PosicionArbol;
Begin
  Root := raiz;
End;

Function Arbol.DatoDeLaClave: TipoDatosClave;
Begin
  DatoDeLaClave := TDatoDeLaClave;
End;

Function Arbol.SizeTree(): LongInt;
Begin
  SizeTree := Size;
End;

Function Arbol.MaxSizeTree(): LongInt;
Begin
  MaxSizeTree := MAX;
End;

// Cuenta los Nodos a Partir de Una Posicion
Function Arbol.ContarNodos(P: PosicionArbol): LongInt;
Var C: LongInt;
  // Procedure que cuenta
  Procedure Cuenta(Q: PosicionArbol);
  Begin
    if Q <> Nulo then Begin
      Inc(C);
      Cuenta(Q^.HI);
      Cuenta(Q^.HD);
    End;
  End;
  // Cuerpo de la Funcion Principal
Begin
  C := 0;
  Cuenta(P);
  ContarNodos := C;
End;

// Propiedades de Asignacion al Arbol
Procedure Arbol.SetRoot(R:PosicionArbol);
Begin
  Raiz := R;
  Q_Items := ContarNodos(R);
End;

// Conecta Hijo Izquierdo (P-->Q)
Procedure Arbol.ConectarHI(P:PosicionArbol; Q:PosicionArbol);
Begin
  P^.HI := Q;
  If Q <> Nulo Then Q_Items := Q_Items + ContarNodos(Q);
End;

// Conecta Hijo Derecho (P-->Q)
Procedure Arbol.ConectarHD(P:PosicionArbol; Q:PosicionArbol);
Begin
```

```
  P^.HD := Q;
  If Q <> Nulo Then Q_Items := Q_Items + ContarNodos(Q);
End;

End.
```