```
unit ArbolesBinariosBusqueda;

interface

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

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

Type
PosicionArbol = ^NodoArbol;

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

ArbolBB = 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 PreOrden(): String;
    Function InOrden(): String;
    Function PostOrden(): String;
    Function Anchura(): String;
    Function PreOrdenITE(): String;
    Function Altura(): Integer;
    Function Nivel(Q:PosicionArbol): LongInt;
    Function HijoIzquierdo(P:PosicionArbol): PosicionArbol;
    Function HijoDerecho(P:PosicionArbol): PosicionArbol;
    Function Padre(Hijo:PosicionArbol): PosicionArbol;
    // Busqueda Binaria
    Function CrearNodo(X:TipoElemento): PosicionArbol;
    Function Insertar(X:TipoElemento):Resultado;
    Function Eliminar(X:TipoElemento):Resultado;
    Function BusquedaBinaria(X:TipoElemento):PosicionArbol;
    // Cargar las claves al Azar
    Function LlenarClavesRandom(alSize: LongInt; RangoDesde, RangoHasta: LongInt): Resultado;
   // Propiedades del arbol
    Function CantidadNodos(): LongInt;
    Function Root(): PosicionArbol;
    Function DatoDeLaClave: TipoDatosClave;
    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 ArbolBB.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 ArbolBB.EsVacio(): Boolean; // Sinonimo de arbol vacio
Begin
  EsVacio := (raiz = Nulo);
End;

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

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

// Recupero el elemento de la posicion P
Function ArbolBB.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 ArbolBB.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 ArbolBB.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 ArbolBB.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 ArbolBB.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 ArbolBB.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;

// Sacamos la altura del Arbol usando el recorrido pre-orden
Function ArbolBB.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 mayor
  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 ArbolBB.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 ArbolBB.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 ArbolBB.HijoIzquierdo(P:PosicionArbol): PosicionArbol;
Begin
  HijoIzquierdo := Nulo;
  If Not RamaNula(P) Then HijoIzquierdo := P^.HI;
End;

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

//-------------------------------------------------------------
// Rutinas de Arboles Binarios de Busqueda SIN BALANCEO
//-------------------------------------------------------------
Function ArbolBB.CrearNodo(X:TipoElemento): PosicionArbol;
Var P: PosicionArbol;
Begin
  New(P);
  P^.Datos := X;
  P^.HI := Nulo;
  P^.HD := Nulo;
  CrearNodo := P;
End;

// Inserta un Nuevo Nodo con la clave en un arbol binario de busqueda
// Las claves duplicadas van en el subarbol derecho
Function ArbolBB.Insertar(X:TipoElemento): Resultado;
Var P, Q: PosicionArbol;
Begin
  if X.TipoDatoClave(X.Clave) <> TDatoDeLaClave  then Begin
    Insertar := ClaveIncompatible;
    Exit;
  End;
  // Ahora lo Inserto
  If EsLleno() Then Insertar := LLena
  Else Begin
    If EsVacio() Then Begin // Inserta la Raiz. Primer Nodo
      Q := CrearNodo(X);
      Raiz := Q;
      Inc(Q_Items);
    End
    Else Begin
      Q := Raiz;
      While Not(RamaNula(Q)) Do Begin // Busca la Hoja donde insertar
        P := Q;
        If X.Clave < Q^.Datos.Clave Then Q := Q^.HI
        Else Q := Q^.HD;
      End;
      // Crea el Nodo Nuevo
      Q := CrearNodo(X);
      Inc(Q_Items);
      // Conecta el Nodo Nueva con la Hoja por el Hijo que corresponda segun sea menor o mayor
      If X.Clave < P^.Datos.Clave Then P^.HI := Q
      Else P^.HD := Q;
    End;
    Insertar := OK;
  End;
End;

// Busca la clave y si la encuentra la elimina del arbol binario
```

```
// de busqueda conectando los subarboles
Function ArbolBB.Eliminar(X:TipoElemento): Resultado;
Var Q: PosicionArbol;
B: Boolean;
  // Proceso recursivo que elimina la clave del arbol
  Procedure Elimina(Var P: PosicionArbol);
    // Proceso que resuelve el problema de la eliminacion logica
    Procedure Buscar_Clave_Reemplazar(Var R: PosicionArbol);
    begin
      If Not (RamaNula(R^.HI)) Then Buscar_Clave_Reemplazar(R^.HI)
      Else Begin
        Q^.Datos := R^.Datos;  //Borrado logico
        Q := R;
        R := Q^.HD;
      End;
    End;
  // Inicio de Eliminar
  Begin
    If Not RamaNula(P) Then
    // Busca la clave por izquierda
    If X.Clave < P^.Datos.Clave Then Elimina(P^.HI)
    Else // Busca la clave por derecha
      If X.Clave > P^.Datos.Clave Then Elimina(P^.HD)
      Else Begin // aca encontre la clave
        Q := P; // Asigna P para hacer el Dispose
        // Pregunta si solo tiene hijo derecho
        If RamaNula(Q^.HI) Then P := Q^.HD
        Else // Pregunta si solo tiene hijo izquierdo
          If RamaNula(Q^.HD) Then P := Q^.HI
            Else // Tiene 2 hijos. Busca la clave para reemplazar
              Buscar_Clave_Reemplazar(Q^.HD);
        Dispose(Q); // borrado Fisico
        Dec(Q_Items);
        B := True; // Marco que el borrado fue posible
      End;
  End;
// Inicio de la Funcion
Begin
  if X.TipoDatoClave(X.Clave) <> TDatoDeLaClave  then Begin
    Eliminar := ClaveIncompatible;
    Exit;
  End;
  // Ahora lo Elimino
  If EsVacio() Then Eliminar := Vacia
  Else Begin
    Eliminar := CError;
    B := False;
    Elimina(raiz);
    If B = True Then Eliminar := OK;
  End;
End;

// Busca en Forma Binaria la Clave en Funcion de uno de los Campos del TipoElemento
Function ArbolBB.BusquedaBinaria(X:TipoElemento):PosicionArbol;
Var Q: PosicionArbol;
    Encontre: Boolean;
Begin
  BusquedaBinaria := Nulo;
  if X.TipoDatoClave(X.Clave) <> TDatoDeLaClave  then Begin
    Exit;
  End;
  // Ahora lo Busco
  Encontre := False;
  Q := Raiz;
  While Not(RamaNula(Q)) And (Not(Encontre)) Do Begin // Busca a izquierda y Derecha
    If X.Clave < Q^.Datos.Clave Then Q := Q^.HI
    Else
      If X.Clave > Q^.Datos.Clave Then Q := Q^.HD
      Else Encontre := True;
  End;
  If Encontre Then BusquedaBinaria := Q;
End;
```

```
// Llena las claves de forma random
Function ArbolBB.LlenarClavesRandom(alSize: LongInt; RangoDesde, RangoHasta: LongInt): Resultado;
Var X: TipoElemento;
Begin
  TDatoDeLaClave := Numero;
  // Creo el arbol y controlo
  If Crear(TDatoDeLaClave, alSize) <> OK Then Begin
    LlenarClavesRandom := CError;
    Exit;
  End;
  // Cargo hasta que se llene
  X.Inicializar(TDatoDeLaClave, '');
  Randomize;
  While Not EsLleno() Do Begin
    X.Clave := (RangoDesde + Random(RangoHasta));
    Insertar(X);
  End;
  LlenarClavesRandom := OK;
End;

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

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

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

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

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

// Cuenta los Nodos a Partir de Una Posicion
Function ArbolBB.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 ArbolBB.SetRoot(R:PosicionArbol);
Begin
  Raiz := R;
  Q_Items := ContarNodos(R);
```

```
End;

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

End.
```