```
unit ArbolesBinariosAVL;

interface

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

Const
MIN = 1;
MAX = 10000;  // Tamaño maximo del arbol en cantidad de nodos
Nulo= Nil;   // Posicion NO valida del arbol

Type
PosicionArbol = ^NodoArbol;

NodoArbol = Object
  Datos: TipoElemento;
  HI, HD: PosicionArbol;
  FE: -1..1; // Para el Balanceo
End;

ArbolAVL = 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 BusquedaBinaria(X:TipoElemento):PosicionArbol;
    // Busqueda Binaria Balanceado
    Function Insertar(X:TipoElemento):Resultado;
    Function Eliminar(X:TipoElemento):Resultado;
    // Cargar al Azar las claves
    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 ArbolAVL.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 ArbolAVL.EsVacio(): Boolean; // Sinonimo de arbol vacio
Begin
  EsVacio := (raiz = Nulo);
End;

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

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

// recupera un TipoElemento de la Posicion P
Function ArbolAVL.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 ArbolAVL.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 ArbolAVL.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 ArbolAVL.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 ArbolAVL.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 ArbolAVL.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 ArbolAVL.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 ArbolAVL.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 ArbolAVL.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 ArbolAVL.HijoIzquierdo(P:PosicionArbol): PosicionArbol;
Begin
  HijoIzquierdo := Nulo;
  If Not RamaNula(P) Then HijoIzquierdo := P^.HI;
End;

// Retorna el Hijo Derecho de un Nodo
Function ArbolAVL.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 ArbolAVL.CrearNodo(X:TipoElemento): PosicionArbol;
Var P: PosicionArbol;
Begin
  New(P);
  P^.Datos := X;
  P^.HI := Nulo;
  P^.HD := Nulo;
  P^.FE := 0;
  CrearNodo := P;
End;

// Busca en Forma Binaria la Clave en Funcion de uno de los Campos del TipoElemento
Function ArbolAVL.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;

//-------------------------------------------------------------
// RUTINAS DE BALANCEO
//-------------------------------------------------------------
// Rotacion por desbalanceo Izquierdo Simple
Procedure R_II(Var N: PosicionArbol; N1: PosicionArbol);
Begin
  // Rotacion
  N^.hi := N1^.hd;
  N1^.hd:= N;
  // Acomoda los factores de equilibrio
  If N1^.fe = -1 then Begin
    N1^.fe := 0;
```

```
    N^.fe := 0;
  End
  Else Begin
    N^.fe := -1;
    N1^.fe := 1;
  End;
  // retorna la nueva raiz
  N := N1;
End;

// Rotacion por desbalanceo Derecho Simple
Procedure R_DD(Var N: PosicionArbol; N1: PosicionArbol);
Begin
  // Rotacion
  N^.hd := N1^.hi;
  N1^.hi:= N;
  // Acomodo FE
  If N1^.fe = 1 then Begin
    N1^.fe := 0;
    N^.fe := 0;
  End
  Else Begin
    N^.fe := 1;
    N1^.fe := -1;
  End;
  // Raiz nueva
  N := N1;
End;

// Rotacion por desbalanceo Doble Izquierdo-Derecho
Procedure R_ID(Var N: PosicionArbol; N1: PosicionArbol);
Var N2: PosicionArbol;
Begin
  N2:= N1^.hd;
  // 1er. Rotacion
  N^.hi := N2^.hd;
  N2^.hd:= N;
  // Segunda rotacion
  N1^.hd:= N2^.hi;
  N2^.hi:= N1;
  // Acomoda FE
  If N2^.fe = -1 then N^.fe := 1
  Else N^.fe := 0;
  If N2^.fe = 1 Then N1^.fe := -1
  Else N1^.fe := 0;
  // FE de N2 siempre es cero
  N2^.fe := 0;
  // retorna nueva raiz
  N := N2;
End;

// Rotacion por desbalanceo Doble Derecho-Izquierdo
Procedure R_DI(Var N: PosicionArbol; N1: PosicionArbol);
Var N2: PosicionArbol;
Begin
  N2:= N1^.hi;
  // 1er. rotacion
  N^.hd := N2^.hi;
  N2^.hi:= N;
  // 2 da. rotacion
  N1^.hi:= N2^.hd;
  N2^.hd:= N1;
  // Acomoda el FE
  If N2^.fe = -1 then N1^.fe := 1
  Else N1^.fe := 0;
  If N2^.fe = 1 Then N^.fe := -1
  Else N^.fe := 0;
  // FE de N2
  N2^.fe := 0;
  // retorna raiz nueva
  N := N2;
End;
```

```
// Funcion que inserta un Nodo en el arbol binario balanceado
Function ArbolAVL.Insertar(X: TipoElemento): Resultado;
Var bh: Boolean;
Inserto: Boolean;
  // Proceso recursivo que inserta la clave
  Procedure Inserta(Var R: PosicionArbol; Var bh: Boolean);
  Var N1: PosicionArbol;
  Begin
    If RamaNula(R) Then Begin
      R := CrearNodo(X);
      bh := True;
      Inserto := True;
    End
    Else
    If X.Clave < R^.Datos.Clave Then Begin
      Inserta(R^.hi, bh);
      If bh Then
        Case R^.fe of
        1: Begin
            R^.fe := 0;
            bh := False;
          End;
        0: R^.Fe := -1;
        -1: Begin
            N1:= R^.hi;
            If N1^.fe <= 0 Then R_II(R, N1)
            Else R_ID(R, N1);
            bh:= False;
          End;
        End;
    End
    Else
      If ((X.Clave > R^.Datos.Clave) OR (X.Clave = R^.Datos.Clave)) Then Begin
        Inserta(R^.hd, bh);
        If bh Then
          Case R^.fe Of
          -1: Begin
              R^.fe := 0;
              bh := False;
            End;
          0: R^.fe := 1;
          1: Begin
              N1:= R^.hd;
              If N1^.fe >= 0 Then R_DD(R, N1)
              Else R_DI(R, N1);
              bh:= False;
            End;
          End;
      End;
  End;
// Codigo de la funcion principal
Begin
  Inserto := False;
  if X.TipoDatoClave(X.Clave) <> TDatoDeLaClave  then Begin
    Insertar := ClaveIncompatible;
    Exit;
  End;
  If EsLLeno() Then Insertar := Llena
  Else Begin
    Insertar := CError;
    bh := False;
    Inserta(raiz, bh);
    If Inserto = True Then Begin
      Inc(Q_Items);
      Insertar := OK;
    End;
  End;
End;

{ --- Al Borrar un Nodo por la Izquierda entonces crece la derecha --- }
Procedure Equilibrar_I(Var N:PosicionArbol; Var bh: Boolean);
```

```
Var N1: PosicionArbol;
Begin
  Case N^.fe Of
    -1: N^.fe:= 0;
    0: Begin
        N^.fe:= 1;
        bh:= False;
      End;
    1: Begin
      N1:= N^.hd;
      If N1^.fe >= 0 Then Begin
        If N1^.fe = 0 Then bh:= false;
        R_DD(N, N1);
      End
      Else R_DI(N, N1);
    End;
  End;
End;

{ --- Al Borrar un Nodo por la Derecha entonces crece la izquierda --- }
Procedure Equilibrar_D(Var N:PosicionArbol; Var bh: Boolean);
Var N1: PosicionArbol;
Begin
  Case N^.fe Of
    1: N^.fe:= 0;
    0: Begin
        N^.fe:= -1;
        bh:= False;
      End;
    -1: Begin
        N1:= N^.hi;
        If N1^.fe <= 0 Then Begin
          If N1^.fe = 0 Then bh:= false;
          R_II(N, N1);
        End
        Else R_ID(N, N1);
      End;
  End;
End;

// Funcion que elimina una clave en el arbol binario y lo deja equilibrado
Function ArbolAVL.Eliminar(X: Tipoelemento): Resultado;
Var bh: Boolean;
  Procedure Elimina(Var R: PosicionArbol; Var bh: Boolean);
  Var Q: PosicionArbol;
    { Busca el Nodo a Reemplazar Todo por la Izquierda}
    Procedure B_N_R(Var P: PosicionArbol; Var bh: Boolean);
    Begin
      If P^.hi <> Nulo Then Begin
        B_N_R(P^.hi, bh);
        If bh = True Then Equilibrar_I(P, bh);
      End
      Else Begin
        Q^.Datos := P^.Datos;
        Q := P;
        P := P^.hd;
        bh:= True;
      End;
    End;
  { Aqui Comienza el procedure Eliminar }
  Begin
    If Not(RamaNula(R)) Then
      If X.Clave < R^.Datos.Clave Then Begin
        Elimina(R^.hi, bh);
        If bh Then Equilibrar_I(R, bh);
      End
      Else
        If X.Clave > R^.Datos.Clave Then Begin
          Elimina(R^.hd, bh);
          If bh Then Equilibrar_D(R, bh);
        End
        Else Begin
```

```
          Q := R;
          If Q^.hd = Nulo Then Begin
            R := Q^.hi;
            bh:= True;
          End
          Else
            If Q^.hi = Nulo Then Begin
              R := Q^.hd;
              bh:= True;
            End
            Else Begin
              B_N_R(Q^.hd, bh);
              If bh Then Equilibrar_D(R, bh);
            End;
          Dispose(Q); // Borra de la memoria el Nodo Ocupado
          Dec(Q_Items);
        End;
  End;
// Codigo de la funcion principal
Begin
  if X.TipoDatoClave(X.Clave) <> TDatoDeLaClave  then Begin
    Eliminar := ClaveIncompatible;
    Exit;
  End;
  // ahora lo elimina
  If EsVacio() Then Eliminar := Vacia
  Else Begin
    Eliminar := CError;
    Elimina(raiz, bh);
    Eliminar := OK;
  End;
End;

// Llena las claves de forma random
Function ArbolAVL.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;
  // Ahora lo lleno Random
  X.Inicializar(TDatoDeLaClave, '');
  Randomize;
  // Cargo hasta que se llene
  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 ArbolAVL.CantidadNodos(): LongInt;
Begin
  CantidadNodos := Q_Items;
End;

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

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

Function ArbolAVL.SizeTree(): LongInt;
```

```
Begin
  SizeTree := Size;
End;

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

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

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

End.
```