```
Unit BTreeMejorado;

Interface

Uses
Tipos, SysUtils;

Const
Orden = 5;                // Orden del arbol B
MinClaves = Orden div 2;  // cantidad minima de claves de la pagina
MaxClaves = Orden - 1;    // cantidad maxima de claves de la pagina
Nulo = Nil;               // Posicion NO valida de una pagina
MAX = 2000;              // Cantidad Maxima de claves a almacenar
MIN = 1;

Type
PosicionArbolBmejorado = ^Pagina;

Indice = 0..MaxClaves;

Pagina = Object
  Cantidad : Indice;
  Claves : Array[1..MaxClaves] of TipoElemento;
  Ramas : Array[Indice] of PosicionArbolBmejorado;
End;

ArbolBmejorado = Object
  Private
    Raiz: PosicionArbolBmejorado;
    Q_Claves : LongInt;
    Q_Paginas: LongInt;
    TDatoDeLaClave: TipoDatosClave;
    Size : LongInt;
  Public
    Function Crear(avTipoClave: TipoDatosClave; alSize: LongInt): Resultado;
    Function EsVacio(): Boolean;
    Function EsLLeno(): Boolean;
    Function RamaNula(P: PosicionArbolBmejorado): Boolean;
    Function Insertar(X:TipoElemento): Resultado;
    Function Eliminar(X:TipoElemento): Resultado;
    Function BuscarClave(X:TipoElemento; Var k: indice): PosicionArbolBmejorado;
    Function Recuperar(P: PosicionArbolBmejorado; k:Indice): TipoElemento;
    Function LLenarClavesRandom(alSize: LongInt; RangoDesde, RangoHasta: LongInt): Resultado;
    Function InOrden(): String;
    Function Root(): PosicionArbolBmejorado;
    Function CantidadPaginas(): LongInt;
    Function CantidadClaves(): LongInt;
    Function Altura (): LongInt;
    Function CantidadClavesPagina(P: PosicionArbolBmejorado): Integer;
    Function HijoDeLaRama(P: PosicionArbolBmejorado; K:indice): PosicionArbolBmejorado;
    Function DatoDeLaClave: TipoDatosClave;
    Function SizeTree(): LongInt;
    Function MaxSizeTree(): LongInt;
  End;

Var
  Inserte: Boolean; // Indica si efectivamente creamos claves
  Elimine: Boolean; // Indica si efectivamente eliminamos claves
  NewPage: Integer; // Usada para Contar las paginas nuevas creadas
  OldPage: Integer; // Usada para Contar cuantas paginas se eliminaron


Implementation

// Crea el arbol vacio
Function ArbolBmejorado.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_Claves := 0;
```

```
    Q_Paginas:= 0;
    TDatoDeLaClave := avTipoClave;
    Size := alSize;
    Crear := OK;
  End;
End;

// Control de arbol vacio
Function ArbolBmejorado.EsVacio(): Boolean;
Begin
  EsVacio := (Raiz = Nulo);
End;

// Control de arbol lleno
Function ArbolBmejorado.EsLLeno(): Boolean;
Begin
  EsLLeno := (Q_Claves = Size);
End;

// Control de rama nula
Function ArbolBmejorado.RamaNula(P: PosicionArbolBmejorado): Boolean;
Begin
  RamaNula := (P = Nulo);
End;

{----------------------------------------------------------------------
recibe: x = clave nueva; P = posicion de la pagina
retorna: encontre = true si lo encontro; k = posicion del indice de la clave dentro
de la pagina
----------------------------------------------------------------------}
Procedure BuscarNodo(x: TipoElemento; P:PosicionArbolBmejorado; Var Encontre: Boolean; Var K:Indice
);
Begin
  If X.Clave < P^.claves[1].Clave Then begin
    Encontre := False;
    k := 0;
  End
  Else Begin
    k := P^.Cantidad;
    While (X.Clave < P^.claves[k].Clave) And (K > 1) Do
      k := K - 1;
    Encontre := (X.Clave = P^.claves[k].Clave);
  End;
End;

{----------------------------------------------------------------------
recibe: x = clave nueva; B = raiz del arbol B
retorna: encontre = true si lo encontro; P = posicion de la pagina encotrada;
k = posicion del indice de la clave dentro de la pagina
----------------------------------------------------------------------}
Function ArbolBmejorado.BuscarClave(X:TipoElemento; Var k: indice): PosicionArbolBmejorado;
Var Encontre: Boolean;
// Proceso interno que busca la clave
  Procedure Buscar(Q: PosicionArbolBmejorado; Var Encontre: Boolean);
  Begin
    If RamaNula(Q) Then Encontre := False
    Else begin
      BuscarNodo(X, Q, Encontre, k);
      If Encontre Then BuscarClave := Q
      Else Buscar(Q^.Ramas[k], Encontre);
    End;
  End;
// Cuerpo de la Funcion principal
Begin
  Encontre := False;
  BuscarClave := Nulo;
  K := 0;
  Buscar(Raiz, Encontre);
End;

{----------------------------------------------------------------------
recibe: x = clave nueva; pd = posicion del rama derecha; p = posicion de la pagina;
```

```
k = indice de la clave
-------------------------------------------------------------------------}
Procedure MeterHoja(x: TipoElemento; pd, p: PosicionArbolBmejorado; k: indice);
Var i: integer;
Begin
  // Se realiza el corrimiento a la derecha para generar el Hueco
  // donde poner la clave
  For i:= P^.cantidad Downto (K + 1) Do Begin
    P^.claves[i + 1] := P^.claves[i];
    P^.ramas[i + 1] := P^.ramas[i];
  End;
  // Se pone la clave nueva en la posicion K+1 y se apunta a la pagina derecha
  P^.claves[k + 1] := x;
  P^.ramas [k + 1] := pd;
  P^.cantidad := P^.cantidad + 1;
  Inserte := True;
End;


{-------------------------------------------------------------------------
recibe: P = posicion de pagina; k = indice
recibe: PP = posicion de pagina padre; kp = indice pagina padre
recibe: X = clave nueva a insertar
-------------------------------------------------------------------------}
Procedure MoverDerechaInsert(PP, P: PosicionArbolBmejorado; kp, k: indice; X: TipoElemento);
Var J: indice;
Begin
  // Genero el hueco en la rama derecha.
  // Corro todas las claves un lugar a la derecha
  For J:= PP^.Ramas[kp + 1]^.Cantidad Downto 1 Do Begin
    PP^.ramas[kp + 1]^.claves[J + 1] := PP^.ramas[kp + 1]^.claves[J];
    PP^.ramas[kp + 1]^.ramas [J + 1] := PP^.ramas[kp + 1]^.ramas[J];
  End;
  // Ahora en el primer lugar de la rama derecha pongo la clave que baja del padre
  PP^.ramas[kp + 1]^.Claves[1] := PP^.Claves[kp + 1];
  PP^.ramas[kp + 1]^.Ramas [1] := Nulo;
  PP^.ramas[kp + 1]^.Ramas [0] := Nulo;
  Inc(PP^.ramas[kp + 1]^.Cantidad);
  // Si la clave es la mayor entonces esta es la que sube al padre
  If k = P^.Cantidad Then Begin
    // Ahora paso la clave nueva (X) al padre
    PP^.Claves[kp + 1] := X;
  End
  Else Begin
    // Ahora paso la clave de mas a la derecha de la rama izquierda al padre
    PP^.Claves[kp + 1] := P^.Claves[P^.Cantidad];
    // Ahora meto la clave nueva donde corresponde en la rama izquierda
    For J := P^.Cantidad  DownTo K + 2 Do Begin
      P^.Claves[J] := P^.Claves[J - 1];
      P^.Ramas [J] := P^.Ramas [J - 1];
    End;
    // Ahora inserto la clave nueva en la pagina <P>
    P^.Claves[K + 1] := X;
    P^.Ramas [K + 1] := Nulo;
    P^.Ramas [K] := Nulo;
  End;
End;


{-------------------------------------------------------------------------
recibe: P = posicion de pagina; k = indice
recibe: PP = posicion de pagina padre; kp = indice pagina padre
recibe: X = clave nueva a insertar
-------------------------------------------------------------------------}
Procedure MoverIzquierdaInsert(PP, P: PosicionArbolBmejorado; kp, k: indice; X:TipoElemento);
Var J: indice;
Begin
    // Pongo en la rama izquieda la clave que baja del padre
    PP^.Ramas[kp - 1]^.Claves[PP^.Ramas[kp - 1]^.Cantidad + 1] := PP^.Claves[kp];
    PP^.Ramas[kp - 1]^.Ramas [PP^.Ramas[kp - 1]^.Cantidad + 1] := Nulo;
    Inc(PP^.Ramas[kp - 1]^.Cantidad);
    // Me fijo si la clave a insert es la mas chica de todas va al padre directamente
```

```
    If k = 0 Then Begin
      PP^.Claves[kp] := X;
    End
    Else Begin
      // Paso la clave menor de la rama derecha Arriba (al padre)
      PP^.Claves[kp] := P^.Claves[1];
      // Ahora corro todas las clave un lugar a la izquierda
      For J:= 1 To P^.Cantidad - 1 Do Begin
        P^.claves[J] := P^.claves[J + 1];
        P^.ramas[J]  := P^.ramas [J + 1];
      End;
      Dec(P^.Cantidad);
      // Ahora debo generar el Hueco donde va la nueva clave
      For J:= P^.Cantidad Downto K Do Begin
        P^.claves[J + 1] := P^.claves[J];
        P^.ramas [J + 1] := P^.ramas [J];
      End;
      // Ahora asigno la clave nueva en la posicion K
      P^.Claves[K]  := X;
      P^.ramas [K-1]:= Nulo;
      P^.ramas [K]  := Nulo;
      Inc(P^.Cantidad);
    End;
End;


{------------------------------------------------------------------------
recibe: PP = Padre de P; p = posicion de la pagina
k = indice de la clave;  KP = indice del padre por donde bajo; X = Clave a insertar
retorna: Dividir = Señal que marca si se debe o NO dividir la pagina
------------------------------------------------------------------------}
Procedure ReestablecerAntesDividir(PP:PosicionArbolBmejorado; P: PosicionArbolBmejorado; X:TipoElem
ento; KP: indice; K:indice; Var Dividir: Boolean);
Begin
    Dividir := True;
    // Se fija que este en una hoja antes que nada
    If (P^.Ramas[K] = Nulo) And (P <> PP) Then Begin
      // si KP = 0 se fija si la hoja de la derecha tiene lugar
      If KP = 0 Then Begin
        If PP^.Ramas[1].Cantidad < MaxClaves Then Begin
          MoverDerechaInsert(PP, P, KP, K, X);
          Dividir := False;
        End
      End
      Else Begin
        // Si KP = cantidad de claves se fija en la hoja de la Izquierda
        If KP = PP^.Cantidad Then begin
          If PP^.Ramas[KP - 1].Cantidad < MaxClaves Then Begin
            MoverIzquierdaInsert(PP, P, KP, K, X);
            Dividir := False;
          End
        end
        Else Begin
          // Sino se fija a ambos lados si puede transferir una clave para evitar la division del n
odo
          If PP^.Ramas[KP].Cantidad < MaxClaves Then Begin
            MoverDerechaInsert(PP, P, KP, K, X);
            Dividir := False;
          End
          Else begin
            if KP = 0 then Begin
              Dividir := True;
              Exit;
            End;
            If PP^.Ramas[KP - 1].Cantidad < MaxClaves Then Begin
              MoverIzquierdaInsert(PP, P, KP, K, X);
              Dividir := False;
            End
          end;
        End;
      End;
    End;
```

```
End;


{------------------------------------------------------------------------
recibe: x = clave nueva; pd = posicion del rama derecha; p = posicion de la pagina
k = indice de la clave
retorna: cm = clave de la mediana a subir; pcm = posicion de la rama derecha de la
clave mediana
------------------------------------------------------------------------}
Procedure DividirNodo(X:TipoElemento; pd,P: PosicionArbolBmejorado; k:indice; Var cm:TipoElemento;
Var pcm: PosicionArbolBmejorado);
Var i, pm: indice;
Begin
    {Se saca si la clave va a la izq o a la derecha}
    If k <= MinClaves Then pm := MinClaves
    Else pm := MinClaves + 1;
    {se crea el nuevo nodo y se pasan las claves de mas a la derecha}
    New(pcm);
    For i:= pm + 1 To MaxClaves do Begin
      pcm^.claves[i - pm] := P^.claves[i];
      pcm^.Ramas [i - pm] := P^.ramas[i];
    End;
    {Se saca la cantidad de claves que quedan en cada nodo}
    pcm^.cantidad := MaxClaves - pm;
    P^.cantidad := pm;
    {Se inserta la clave y su rama derecha}
    If k <= MinClaves Then MeterHoja(x, pd, P, k)
    Else MeterHoja(x, pd, pcm, k - pm);
    {Se actualizan los valores}
    cm := P^.claves[P^.cantidad];
    pcm^.Ramas[0] := P^.ramas[P^.cantidad];
    P^.cantidad := P^.cantidad - 1;
    // Marco que genere una pagina nueva
    NewPage := NewPage + 1;
End;

{------------------------------------------------------------------------
recibe: x = clave nueva; p = posicion de la pagina
retorna: empujaarriba = true o false segun corresponda; cm = clave mediana;
pd = puntero rama derecha
Profesor: Carlos Rodriguez
Lic. en Sistemas de Información
Programación 2
------------------------------------------------------------------------}
Procedure Empujar(X:TipoElemento; P:PosicionArbolBmejorado; PP:PosicionArbolBmejorado; KP: indice;
Var empujaarriba: Boolean; Var cm: TipoElemento; Var pd: PosicionArbolBmejorado);
Var k: indice;
    esta, divide: boolean;
Begin
  {llegue a una rama vacia}
  If P = Nulo Then begin
    empujaarriba := True;
    cm := X;
    pd := Nulo;
  End
  Else Begin
    BuscarNodo(X, P, Esta, k);
    {Clave repetida}
    If Esta Then Begin
      Exit;
    End;
    {llamo recursivamante con la rama K que retorna el buscarnodo}
    Empujar(X, P^.ramas[k], P, k, empujaarriba, cm, pd);
    {si hay que empujar hacia arriba}
    If empujaarriba Then
      {si el nodo no esta lleno solo pongo la hoja y salgo}
      If P^.cantidad < MaxClaves Then Begin
        empujaarriba := False;
        MeterHoja(cm, pd, P, k);
      End
      { si el nodo esta lleno debe dividirlo en 2 paginas nuevas}
      Else Begin
```

```
        // se fija si puede evitar la division
        ReestablecerAntesDividir(PP, P, X, kp, k, divide);
        // Si hay que dividir
        If divide = True Then Begin
          empujaarriba := True;
          DividirNodo(cm, pd, P, k, cm, pd);
        End
        Else Begin
          empujaarriba := False;
          Inserte := True;
        End;
      End;
  End;
End;

//---------------------------------------------------------------------------
//recibe: x = clave nueva; B = Arbol B
//---------------------------------------------------------------------------
Function ArbolBmejorado.Insertar(X:TipoElemento): Resultado;
  // Proceso interno que hace la insercion
  Procedure Inserta(Var P: PosicionArbolBmejorado);
  Var empujaarriba: Boolean;
      x1: TipoElemento;
      pd, Q: PosicionArbolBmejorado;
  Begin
    { llamo a empujar arriba}
    Empujar(X, P, P, 0, empujaarriba, x1, pd);
    { si se empujo entonces creo una nueva raiz}
    If empujaarriba Then begin
      New(Q);
      Q^.cantidad := 1;
      Q^.claves[1]:= x1; {pongo la clave mediana}
      Q^.ramas [0]:= P; {apunto a la raiz actual}
      Q^.Ramas [1]:= pd; {apunto al nuevo nodo que resulto de la division}
      P := Q; {retorno la nueva raiz}
      Inserte := True;
      NewPage := NewPage + 1;
    End;
  End;
// Cuerpo de la Funcion
Begin
  if X.TipoDatoClave (X.Clave) <> TDatoDeLaClave  then Begin
    Insertar := ClaveIncompatible;
    Exit;
  End;
  // Ahora intenta insertalo
  Insertar := CError;
  NewPage := 0;
  Inserte := False;
  If EsLLeno() Then Insertar := Llena
  Else Begin
    Inserta(raiz);
    If Inserte Then Begin
      Inc(Q_Claves);
      Q_Paginas := Q_Paginas + NewPage;
    End;
    Insertar := OK;
  End;
End;


{---------------------------------------------------------------------------
recibe: P = posicion de pagina; k = indice
---------------------------------------------------------------------------}
Procedure MoverDerecha(P: PosicionArbolBmejorado; k: indice);
Var J: indice;
Begin
  {Genera un hueco mandando claves a la derecha}
  For J:= P^.ramas[k]^.cantidad Downto 1 Do Begin
    P^.ramas[k]^.claves[J + 1] := P^.ramas[k]^.claves[J];
    P^.ramas[k]^.ramas [J + 1] := P^.ramas[k]^.ramas[J];
  End;
```

```
  {inserta la clave K del nodo padre}
  P^.ramas[k]^.cantidad := P^.ramas[k]^.cantidad + 1;
  P^.ramas[k]^.ramas[1] := P^.ramas[k]^.ramas[0];
  P^.ramas[k]^.claves[1] := P^.claves[k];
  {Asciende la clave mayor del hermano izquierdo padre P}
  P^.claves[k] := P^.Ramas[k - 1]^.claves[P^.Ramas[k - 1]^.cantidad];
  P^.ramas [k]^.ramas[0] := P^.Ramas[k - 1]^.ramas[P^.Ramas[k - 1]^.cantidad];
  P^.Ramas[k - 1]^.cantidad := P^.Ramas[k - 1]^.cantidad - 1;
End;

{------------------------------------------------------------------------
recibe: P = posicion de pagina; k = indice
------------------------------------------------------------------------}
Procedure MoverIzquierda(P: PosicionArbolBmejorado; k: indice);
Var J: indice;
Begin
  {Desciende la clave k(1) del nodo padre (P) al hijo izquierdo y la
  inserta en la posicion mas alta, asi se reestablece el minimo de claves}
  P^.Ramas[k - 1]^.cantidad := P^.Ramas[k - 1]^.cantidad + 1;
  P^.Ramas[k - 1]^.claves[P^.Ramas[k - 1]^.cantidad] := P^.claves[k];
  P^.Ramas[k - 1]^.ramas[P^.Ramas[k - 1]^.cantidad] := P^.ramas[k]^.ramas[0];
  {Sube la clave 1 del hermano derecho}
  P^.claves[k] := P^.ramas[k]^.claves[1];
  P^.ramas[k]^.ramas[0] := P^.ramas[k]^.ramas[1];
  P^.ramas[k]^.cantidad := P^.ramas[k]^.cantidad - 1;
  {se corren las claves y las ramas un lugar a la izquierda}
  For J:= 1 To P^.ramas[k]^.cantidad Do Begin
    P^.ramas[k]^.claves[J] := P^.ramas[k]^.claves[J + 1];
    P^.ramas[k]^.ramas[J] := P^.ramas[k]^.ramas[J + 1];
  End ;
End;


{------------------------------------------------------------------------
recibe: P = posicion de pagina; k = indice
------------------------------------------------------------------------}
Procedure Combina(P: PosicionArbolBmejorado; k: indice);
Var J: indice;
    Q: PosicionArbolBmejorado;
Begin
  {Posicion a borrar}
  Q := P^.ramas[K];
  { Baja la clave mediana del nodo padre}
  P^.ramas[k - 1]^.cantidad := P^.ramas[k - 1]^.cantidad + 1;
  P^.ramas[k - 1]^.claves[P^.ramas[k - 1]^.cantidad] := P^.claves[k];
  P^.ramas[k - 1]^.ramas[P^.ramas[k - 1]^.cantidad] := Q^.ramas[0];
  { se pasan al final del nodo izquierdo el resto de las claves del nodo derecho a eliminar}
  For J:= 1 To Q^.cantidad Do Begin
    P^.ramas[k - 1]^.cantidad := P^.ramas[k - 1]^.cantidad + 1;
    P^.ramas[k - 1]^.claves[P^.ramas[k - 1]^.cantidad] := Q^.claves[j];
    P^.ramas[k - 1]^.ramas [P^.ramas[k - 1]^.cantidad] := Q^.ramas[j];
  End;
  { Se reconstruyen las claves y ramas del nodo padre ya que una clave descendio}
  For J:= K To P^.Cantidad - 1 Do Begin
    P^.claves[J] := P^.claves[j + 1];
    P^.ramas[J] := P^.ramas[j + 1];
  End;
  P^.cantidad := P^.cantidad - 1;
  { Libero el Nodo Q}
  Dispose(Q);
  // Marco que tengo una pagina menos
  OldPage := OldPage + 1;
End;

{------------------------------------------------------------------------
recibe: P = posicion de pagina; k = indice
------------------------------------------------------------------------}
Procedure Restablecer(P: PosicionArbolBmejorado; k: indice);
Begin
  {Si K > 0 tiene hermano izquierdo}
  If k > 0 Then Begin
    // Si K es la ultima clave la unica que queda es mirar la rama de la izquierda
```

```
    If k = P^.Cantidad Then Begin
      If P^.ramas[k - 1]^.Cantidad > MinClaves Then MoverDerecha(P, k)
      Else Combina(P, k); {debo combinar 2 nodos en uno}
    End
    Else Begin
      // K > 0 y K < cantidad de claves entonces tengo rama izquierda y derecha valida
      If P^.ramas[k - 1]^.Cantidad > MinClaves Then MoverDerecha(P, k)
      Else Begin
        If P^.ramas[k + 1]^.Cantidad > MinClaves Then MoverIzquierda(P, (k + 1))
        Else Combina(P, k); {debo combinar 2 nodos en uno}
      End;
    End;
  End
  Else Begin {Solo tengo hermano derecho}
    If P^.ramas[1]^.cantidad > MinClaves Then MoverIzquierda(P, 1)
    Else Combina(P, 1);
  End;
End;

{------------------------------------------------------------------------
recibe: P = posicion de pagina; k = indice
------------------------------------------------------------------------}
Procedure Quitar(P: PosicionArbolBmejorado; k: indice);
Var J: indice;
Begin
  // realiza los corrimientos de claves a la izquierza
  // aplastando la posicion "k"
  For J:= k + 1 To P^.cantidad Do Begin
    P^.claves[j - 1] := P^.claves[j];
    P^.ramas [j - 1] := P^.ramas[j];
  End;
  P^.cantidad := P^.cantidad - 1;
  Elimine := True;
End;

{------------------------------------------------------------------------
recibe: P = posicion de pagina; k = indice
------------------------------------------------------------------------}
Procedure Sucesor(P: PosicionArbolBmejorado; k: indice);
Var Q: PosicionArbolBmejorado;
Begin
  Q := P^.ramas[k];
  { va a la hoja mas a la izquierda}
  While Q^.ramas[0] <> Nulo Do
    Q := Q^.ramas[0];
  { reemplaza la clave K por la 1 del mas a la izquierda}
  P^.claves[k] := Q^.claves[1];
End;

{------------------------------------------------------------------------
recibe: x = Clave a buscar; P = posicion del arbol (raiz)
retorna: encontre = true o false segun corresponda
------------------------------------------------------------------------}
Procedure EliminarRegistro(X:TipoElemento; P: PosicionArbolBmejorado; Var encontre: Boolean);
var k: indice;
Begin
  If P = Nulo Then encontre := False {la clave no esta en el Arbol B}
  Else Begin
    BuscarNodo(X, P, encontre, k);
    If encontre Then
      If P^.ramas[k - 1] = Nulo Then
        Quitar(P, k) {es un nodo hoja solo se quita la clave}
      Else Begin
        // No es nodo hoja, se necesita la clave sucesora y se la busca
        // de la pagina a la derecha, luego la mas a la izquierda
        Sucesor(P, k);
        EliminarRegistro(P^.claves[k], P^.ramas[k], encontre);
        If not encontre Then Begin
          Exit;
        End;
    End
    Else
```

```
      EliminarRegistro(X, P^.ramas[k], encontre);
      // Controla si debe o no reestablecer
      If P^.ramas[k] <> Nulo Then
        If P^.ramas[k]^.Cantidad < MinClaves Then
          Restablecer(P, k);
  End;
End;

{------------------------------------------------------------------------
recibe: x = Clave a buscar; B = Arbol B; Campo Donde esta la clave
------------------------------------------------------------------------}
Function ArbolBmejorado.Eliminar(X:TipoElemento): Resultado;
// Proceso interno que elimina
  Procedure Elimina(Var P: PosicionArbolBmejorado);
  Var encontre: Boolean;
      Q: PosicionArbolBmejorado;
  Begin
    EliminarRegistro(X, P, encontre);
    If not encontre Then
    Else
      If P^.cantidad = 0 Then begin
        Q := P;
        P := P^.ramas[0];
        Dispose(Q);
        Elimine := True;
        OldPage := OldPage + 1;
      End;
  End;
// Cuerpo de la Funcion principal
Begin
  Eliminar := CError;
  Elimine := False;
  OldPage := 0;
  If EsVacio() Then Eliminar := Vacia
  Else Begin
    Elimina(Raiz);
    If Elimine Then Begin
      Dec(Q_Claves);
      Q_Paginas := Q_Paginas - OldPage;
    End;
    Eliminar := OK;
  End;
End;

{------------------------------------------------------------------------
Recibe: B = arbol B, P = Apuntador de la Pagina,
k Subindice donde esta la clave dentro de la pagina
------------------------------------------------------------------------}
Function ArbolBmejorado.Recuperar(P: PosicionArbolBmejorado; k:Indice):TipoElemento;
Var X: TipoElemento;
Begin
  Recuperar := X.TipoElementoVacio;
  If Not EsVacio() Then Begin
    If Not RamaNula(P) Then Begin
      Recuperar := P^.claves[k];
    End;
  End;
End;

{------------------------------------------------------------------------
Recibe: B = Arbol B
Retorno todos los items del Arbol en un solo String
------------------------------------------------------------------------}
Function ArbolBmejorado.InOrden(): String;
Var S, SS: String;
// Proceso Interno que recorre el arbol
  Procedure IOB(P: PosicionArbolBmejorado);
  Var I: Integer;
  Begin
    If Not RamaNula(P) Then Begin
      IOB(P^.Ramas[0]);
      For I := 1 To P^.cantidad Do Begin
```

```
        S := P^.claves[I].ArmarString;
        SS := SS + S + cCRLF;
        IOB(P^.Ramas[I]);
      End;
    End;
  End;
// Cuerpo de la Funcion
Begin
  SS := '';
  IOB(Raiz);
  InOrden := SS;
End;

// Llena el arbol con claves random
Function ArbolBmejorado.LLenarClavesRandom(alSize: LongInt; RangoDesde, RangoHasta: LongInt): Resul
tado;
Var X: TipoElemento;
I: LongInt;
Begin
  Try
    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;
    I := 0;
    While Not EsLLeno() Do Begin
      X.Clave := RangoDesde + Random(RangoHasta);
      Inc(I);
      Insertar(X);
    End;
    LLenarClavesRandom := OK;
  except
    LLenarClavesRandom := CError;
  End;
End;

// retorna como propiedad la raiz del arbol
Function ArbolBmejorado.Root(): PosicionArbolBmejorado;
Begin
  Root := raiz;
End;

// Retorna como propiedad la cantidad de paginas del arbol B
Function ArbolBmejorado.CantidadPaginas(): LongInt;
Begin
  CantidadPaginas := Q_Paginas;
End;

// Retorna como propiedad la cantidad de claves del arbol B
Function ArbolBmejorado.CantidadClaves(): LongInt;
Begin
  CantidadClaves := Q_Claves;
End;

// Retorna la Altura del Arbol “B”
Function ArbolBmejorado.Altura(): LongInt;
Var H: LongInt;
  // Proceso Interno que saca la altura
  Procedure Alt(P: PosicionArbolBmejorado; C: Integer);
  Begin
    If Not RamaNula(P) Then Begin
      P := P^.Ramas[0];
      Alt(P, C+1);
    End
    Else Begin
      If C > H Then H := C;
    End;
```

```
  End;
// Cuerpo de la Funcion
Begin
  H := 0;
  Alt(Raiz, 0);
  Altura := H;
End;

Function ArbolBmejorado.CantidadClavesPagina(P: PosicionArbolBmejorado): Integer;
Begin
  CantidadClavesPagina := 0;
  if not Ramanula(P) then Begin
    CantidadClavesPagina := P^.Cantidad;
  End;
End;

Function ArbolBmejorado.HijoDeLaRama(P: PosicionArbolBmejorado; K:indice): PosicionArbolBmejorado;
Begin
  HijoDeLaRama := Nulo;
  if not ramanula(P) then begin
    HijoDeLaRama := P^.Ramas[K];
  end;
End;

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

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

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

End.
```