```
Unit BMas;

{Objetivo: Primitivas de manejo de un arbol B+ en un
archivo. Los elementos del
arbol pueden ser de cualquier tipo: tipos predefinidos de
Pascal, registros de
datos completos o registros de indice a registros de datos
(clave de
identificacion de registro de datos con referencia a
registro). En cualquier
caso se especifica cual es al clave de identificacion de
los elementos en
un tipo exclusivo y en una funcion primitiva (de interfaz)
que la devuelve.

Autor: Lic. Arturo Servetto

Nota: los comentarios estan escritos obviando acentos y
otros signos diacriticos
para evitar problemas de compatibilidad.}

Interface
Const
     TN=16;  {Tamano de nodo (debe ser un valor que sea
submultiplo o multiplo
     del tamano de los registros fisicos de los
dispositivos de almacenamiento
     persistente o de las unidades de asignacion de
espacio a archivos del
     sistema de archivo del sistema operativo a utilizar:
2^n*512, con n desde
     -5 para nodos de 16 bytes)}

Type
     TElemento=Word;  {tipo de elementos a almacenar en el
arbol (en este caso
     numeros naturales entre 0 y 65535)}
     TClave=Word;  {tipo de clave de identificacion de
elementos (en este caso la
     clave es el mismo elemento, por tratarse de elementos
simples)}
     TRefB=Byte; {tipo de referencias a nodos sucesores}
```

```
Const
     CI=(TN-2-SizeOf(TRefB)) div
(SizeOf(TCLave)+SizeOf(TRefB)); {canti-
     dad maxima de elementos en nodo interno}
     MitadDerCI=CI div 2;
     MitadIzqCI=MitadDerCI+CI mod 2;
     CH=(TN-2-SizeOf(TRefB)) div SizeOf(TElemento);
{cantidad maxima de e
     lementos en nodo hoja}
     MitadDerCH=(CH+1) div 2;
     MitadIzqCH=MitadDerCH+(CH+1) mod 2;
     CR=(TN-2-SizeOf(TRefB)) mod SizeOf(TElemento);
{cantidad de bytes de
     relleno}

      {Las mitades de cantidades de elementos cuando se
parte un nodo puden no
     ser iguales, si CI o CH no son pares: un nodo puede
tener que recibir un
     elemento mas que el otro (se escoge cargar mas al
izquierdo)}

Type
     {$PackRecords 1}  {directiva al compilador (solo para
Free Pascal) para
     que almacene los campos de registros direccionando de
a byte}
     TNodoB=Record {nodo de arbol B}
            cont: Byte;  {contador de elementos contenidos
efectivamente}
            Case tipo: Char of  {tipo de nodo ('i'nterno u
'h'oja)}
            'i': (ei: Array[1..CI] of TClave;  {elementos
de nodo interno}
                  suc: Array[0..CI] of TRefB);
{referencias a nodos sucesores}
            'h': (eh: Array[1..CH] of TElemento;
                  sigH: TrefB;
                  relleno: Array[1..CR] of Byte);  {relleno
para completar el
                  tamano de nodo objetivo (debe activarse
cuando CR>0)}
            end;
```

```
     TArbolB=Record  {arbol B+ persistente - se agrupan
todos los componentes,
     incluso un buffer para mantener la raiz siempre en
memoria, para minimizar
     el acoplamiento de subprogramas}
              nodos: File of TNodoB; {archivo de nodos}
              libres: File of TRefB;  {archivo de
referencias a nodos libres}
              raiz, ultHoja: TNodoB;  {nodo raiz (para
tenerlo siempre en memoria)
              y ultima hoja}
              iUltElem: Byte  {indice del ultimo elemento
devuelto}
              end;

     TRes=(ok, duplicado, inexistente, sustituto);
{resultados posibles de una
     insercion, busqueda o supresion - sustituto reemplaza
a un elemento
     inexistente en las busquedas (busquedas aproximadas)}

Procedure Crear(var arbol: TArbolB);  {crea un arbol B
vacio: inicializa una raiz
vacia y la almacena en un archivo nuevo, creando tambien
un archivo de
referencias a nodos libres vacio}

Procedure Abrir(var arbol: TArbolB);  {abre el archivo de
almacenamiento del
arbol y el de referencias a nodos libres correspondiente,
y lee la raiz}

Procedure Cerrar(var arbol: TArbolB);  {cierra los archivos
componentes del
arbol}

Function Clave(elem: TElemento): TClave;  {devuelve la
clave de un elemento (se
debe implementar segun el tipo TElemento)}

Procedure Insertar(var arbol: TArbolB; elem: TElemento;
var result: TRes);
```

```
{inserta un elemento al arbol, excepto que ya exista uno
con la misma clave de
identificacion, en cuyo caso result vuelve con valor
"duplicado"}

Procedure Buscar(var arbol: TArbolB; claveE: TClave; var
elem: TElemento;
                 var result: TRes);
{si result=sustituto en elem vuelve el siguiente que
encuentre, o el ultimo del
arbol, en el caso extraordinario en que la clave del
elemento a buscar fuere
mayor a la del ultimo elemento del arbol, con
result=inexistente}

Procedure Proximo(var arbol: TArbolB; var elem: TElemento;
var result: TRes);
{si result=ok en elem vuelve el siguiente que encuentre a
partir del ultimo
obtenido por Buscar o por Proximo; si no hay mas elementos
devuelve
result=inexistente}

Procedure Suprimir(var arbol: TArbolB; claveElem: TClave;
var result: TRes);
{elimina el elmento identificado por clveElem del arbol o
devuelve
result=inexistente en caso de que no exista un elemento
con esa clave}

Procedure Visualizar(var arbol: TArbolB);  {muestra en
pantalla el contenido del
nodo raiz por defecto, y permite solicitar se muestre el
contenido de otros
nodos especificando numeros relativos en el archivo de
nodos - se debe adaptar
segun el tipo de elementos - solo muestra claves de
elementos (no los elementos
completos)}

Procedure Exportar(var arbol: TArbolB);  {exporta el arbol
en preorden a un
archivo de texto}
```