```
unit ConjuntosAVL;

interface
  Uses Tipos, ArbolesBinariosAVL, stdctrls, SysUtils, Variants;

Type
  Conjunto = Object
    Private
      Items: ArbolAVL;
      TDatoDeLaClave: TipoDatosClave;
      Function RetornarPosFisica(PosLogica: Integer): PosicionArbol;
    Public
      // Comportamiento del objeto (Operaciones)
      Function Crear(avTipoClave: TipoDatosClave; alSize: LongInt): Resultado;
      Function EsVacio(): Boolean;
      Function EsLLeno(): Boolean;
      Function Agregar(X:TipoElemento): Resultado;
      Function Borrar(X:TipoElemento):Resultado;
      Function BorrarPosicion(PLogica: Integer):Resultado;
      Function Recuperar(PLogica: Integer): TipoElemento;
      Function RetornarClaves(): String;
      // operaciones tipicas de conjuntos
      Function Interseccion(Var C: Conjunto): Conjunto;
      Function Union(Var C: Conjunto): Conjunto;
      Function Diferencia(Var C: Conjunto): Conjunto;
      Function Pertenece(X:TipoElemento): Boolean;
      // operaciones generales
      Function LlenarClavesRandom(alSize: LongInt; RangoDesde, RangoHasta: LongInt): Resultado;
      Function CantidadElementos: LongInt;
      Function DatoDeLaClave: TipoDatosClave;
      Function SizeSet(): LongInt;
      Function MaxSizeSet(): LongInt;
      Procedure Sort();
  End;

// Escribir la implementación del Objeto CONJUNTO
implementation

// crea el conjunto vacio
Function Conjunto.Crear(avTipoClave: TipoDatosClave; alSize: LongInt): Resultado;
Begin
  Crear := items.Crear(avTipoClave, alSize);
  TDatoDeLaClave := avTipoClave;
End;

// control de conjunto vacio
Function conjunto.EsVacio(): Boolean;
Begin
  EsVacio := items.EsVacio()
End;

// control de conjunto lleno
Function conjunto.EsLLeno() : Boolean;
Begin
  EsLLeno := items.EsLLeno();
End;

// Agrega un elemento al conjunto
Function conjunto.Agregar(X:TipoElemento): Resultado;
Begin
  // Controlo el tipo de dato de la clave
  if X.TipoDatoClave(X.Clave) <> TDatoDeLaClave then Begin
    Agregar := ClaveIncompatible;
    Exit;
  End;
  // No permite elemento repetidos
  if Items.BusquedaBinaria (X) <> Nulo then Begin
    Agregar := ClaveDuplicada;
    Exit;
  End;
  // Ahora los agrego
  Agregar := items.Insertar(X);
```

```
End;

// Busca un elemento dentro del conjunto en función de un dato
// Borra el elemento si lo encuentra
Function conjunto.Borrar(X:TipoElemento): Resultado;
Begin
  // Controlo el tipo de dato de la clave
  if X.TipoDatoClave(X.Clave) <> TDatoDeLaClave then Begin
    Borrar := ClaveIncompatible;
    Exit;
  End;
  // Ahora lo mando a Borrar
  Borrar := items.Eliminar(X);
End;

// retorna recorriendo en orden el arbol AVL la posicion fisica
// respecto a la posicion logica
Function conjunto.RetornarPosFisica(PosLogica: Integer): PosicionArbol;
Var Q: PosicionArbol;
    I: Integer;
    // Proceso interno
    Procedure IO(P: PosicionArbol);
    Begin
      If Not items.ramanula(P) Then Begin
        IO(items.hijoizquierdo(P));
        Inc(I);
        If I = PosLogica Then Begin
          Q := P;
          Exit;
        End;
        IO(items.hijoderecho(P));
      End;
    End;
// Cuerpo principal
Begin
  Q := Nulo;
  I := 0;
  If PosLogica > items.cantidadnodos Then RetornarPosFisica := Nulo
  Else IO(items.root);
  RetornarPosFisica := Q;
End;

// retorna por referencia en X el elemento del conjunto
Function conjunto.Recuperar(PLogica: Integer): TipoElemento;
Var Q: PosicionArbol;
    X: TipoElemento;
Begin
  Recuperar := X.TipoElementoVacio;
  Q := RetornarPosFisica(PLogica);
  If Q <> Nulo Then Recuperar := items.recuperar(Q);
End;

// Ordena el conjunto. Ordena los strings sencible a Mayusculas y Minusculas
Procedure conjunto.Sort();
Begin
End;

// Retorna un string con todos los elementos del conjunto
Function conjunto.RetornarClaves():String;
Begin
  RetornarClaves := items.inorden()
End;

// Cantidad de elementos del conjunto
Function conjunto.CantidadElementos: LongInt;
Begin
  CantidadElementos := items.CantidadNodos;
End;

// Llena el conjunto de 0 a <RangoHasta> el atributo DI de la lista
Function conjunto.LlenarClavesRandom(alSize: LongInt; RangoDesde, RangoHasta: LongInt): Resultado;
Var X: TipoElemento;
```

```
Begin
  Randomize;
  TDatoDeLaClave := Numero;
  If Items.Crear(TDatoDeLaClave, alSize) <> OK Then Begin
    LlenarClavesRandom := CError;
    Exit;
  End;
  // Ahora lo lleno random
  X.Inicializar(TDatoDeLaClave, '');
  while Not Items.EsLleno Do Begin
    X.Clave := RangoDesde + Random(RangoHasta);
    if Items.BusquedaBinaria(X) = Nulo then Begin
      Items.Insertar(X);
    End;
  End;
  LlenarClavesRandom := OK;
End;

// Borrar por posición lógica dentro del conjunto
Function conjunto.BorrarPosicion(PLogica: Integer):Resultado;
Var Q: PosicionArbol;
    X: TipoElemento;
Begin
  BorrarPosicion := CError;
  Q := RetornarPosFisica(PLogica);
  If Q <> Nulo Then Begin
    X := items.recuperar(Q);
    If items.BusquedaBinaria(X) = Q Then Begin
      BorrarPosicion := items.Eliminar(X);
      Exit;
    End;
  End;
End;

//-----------------------------------------------------------------
// Funciones típicas de un conjunto
//-----------------------------------------------------------------
// Retorna True si el elemento pertenece al menos 1 vez al conjunto
Function conjunto.Pertenece(X:TipoElemento): Boolean;
Begin
  Pertenece := False;
  If items.BusquedaBinaria(X) <> Nulo Then Pertenece := True;
End;

// Retorna un conjunto INTERSECCION del mismo con <C>
Function conjunto.Interseccion(Var C: Conjunto): Conjunto;
Var CI: Conjunto;
    X : TipoElemento;
    // Proceso interno que resuelve la interseccion
    Procedure IO(P:PosicionArbol);
    Begin
      If Not items.RamaNula(P) Then Begin
        IO(items.HijoIzquierdo(P));
        X := items.Recuperar(P);
        If C.Pertenece(X) = True Then CI.Agregar(X);
        IO(items.hijoderecho(P));
      End;
    End;
// De la funcion Principal
Begin
  CI.Crear(TDatoDeLaClave, SizeSet());
  // Controlo que si uno de los 2 conjuntos es vació no hay intersección
  If items.EsVacio Or C.EsVacio Then Interseccion := CI
  Else IO(items.Root);
  // Retorno el conjunto intersección
  Interseccion := CI;
End;

// Retorna un conjunto UNION del mismo con <C>
Function conjunto.Union(Var C: Conjunto): Conjunto;
Var CU: Conjunto;
    X : TipoElemento;
```

```
    I, Q : Integer;
    // Proceso interno
    Procedure IO(P: PosicionArbol);
    Begin
      If Not items.RamaNula(P) Then Begin
        IO(items.hijoizquierdo(P));
        X := items.recuperar(P);
        CU.Agregar(X);
        IO(items.hijoderecho(P));
      End;
    End;
// cuerpo de la funcion principal
Begin
  CU.Crear(TDatoDeLaClave, (SizeSet() + C.SizeSet));
  // Paso Todos los elementos de conjunto
  IO(items.Root);
  // Agrego todos los elementos del conjunto C incluso lo repetidos
  For I:= 1 To C.cantidadelementos Do Begin
    X := C.recuperar(I);
    CU.Agregar(X);
  End;
  // Retorno el conjunto Unión
  Union := CU;
End;

// Retorna un conjunto con la diferencia del mismo sobre <C>
Function conjunto.Diferencia(Var C: Conjunto): Conjunto;
Var CD: Conjunto;
    X : TipoElemento;
    // Proceso interno que resuelve la interseccion
    Procedure IO(P:PosicionArbol);
    Begin
      If Not items.RamaNula(P) Then Begin
        IO(items.HijoIzquierdo(P));
        X := items.Recuperar(P);
        If Not C.Pertenece(X) Then CD.Agregar(X);
        IO(items.hijoderecho(P));
      End;
    End;
// Funcion principal
Begin
  CD.Crear(TDatoDeLaClave, SizeSet());
  // Controlo que si alguno de los conjuntos es vacio
  If items.EsVacio Then Diferencia := CD
  Else IO(items.root);
  // Retorno el conjunto Diferencia
  Diferencia := CD;
End;

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

Function Conjunto.SizeSet(): LongInt;
Begin
  SizeSet := items.SizeTree;
End;

Function Conjunto.MaxSizeSet(): LongInt;
Begin
  MaxSizeSet := MAX;
End;

End.
```