```
unit TadHash;

interface

uses
  Tipos, ListArray, Variants, SysUtils;

Const
  MinTable = 0;     // Posicion Minima de la Tabla
  MaxTable = 2000;    // Posicion Maxima de la Tabla
  MaxSizeLC= 100;    // Tamaño Maximo Lista de Colisiones
  PosNula = -1;     // Posicion Invalida

Type
  PosicionTabla = LongInt;

  TipoRegistroTabla = Object
    Clave : TipoElemento;
    Ocupado : Boolean;
    ListaColision: Lista;
  End;

  TablaHash = Object
    Private
      Tabla: Array of TipoRegistroTabla;
      Q_Ocupados: Integer;
      Q_Claves:   Integer;
      Q_ClavesLC: Integer;
      TDatoDeLaClave: TipoDatosClave;
      TFuncionHash: TipoFuncionesHash;
      Size: LongInt;
    Public
      Function Crear(avTipoClave: TipoDatosClave; avTipoFuncionHash: TipoFuncionesHash; alSize: Lon
gInt; alNroPrimo: LongInt): Resultado;
      Function EsVacia(): Boolean;
      Function EsLLena(): Boolean;
      Function Insertar(X:TipoElemento): Resultado;
      Function Eliminar(X:TipoElemento): Resultado;
      Function Buscar(X:TipoElemento; Var PL: Variant): PosicionTabla;
      Function Recuperar(P: PosicionTabla; PL: Variant): TipoElemento;
      Function RetornarClaves(): String;
      Function LLenarClavesRandom(alSize, alNroPrimo: LongInt; RangoDesde, RangoHasta: LongInt): Re
sultado;
      Function CantidadClaves(): LongInt;
      Function CantidadOcupados(): LongInt;
      Function CantidadClavesZO(): LongInt;
      Function PrimerPosicionOcupada(): PosicionTabla;
      Function ProximaPosicionOcupada(P: PosicionTabla): PosicionTabla;
      Function RetornarLC(P: PosicionTabla): Lista;
      Function DatoDeLaClave: TipoDatosClave;
      Function FuncionHash: TipoFuncionesHash;
      Function TableSize(): LongInt;
      Function MaxTableSize(): LongInt;
      Function NroPrimo(): LongInt;
  End;


// variables de Instancia de la Libreria
Var
  NPrimo: LongInt;  // Usada para la Funcion Interna
  NSize: LongInt;   // Usada para la funcion interna


implementation

// Marca todas las posiciones como vacias. Por cada posición crea la lista de
// colisiones vacia
Function TablaHash.Crear(avTipoClave: TipoDatosClave; avTipoFuncionHash: TipoFuncionesHash; alSize:
 LongInt; alNroPrimo: LongInt): Resultado;
Var I: LongInt;
Begin
  if alSize < (MinTable + 1) then Crear:= CError;
```

```
  if alSize > MaxTable then Crear:= CError;
  if (alSize >= (Min + 1)) And (alSize <= MaxTable) then Begin
    SetLength(Tabla, (alSize + 1));
    For I:= MinTable To alSize Do Begin
      Tabla[I].Ocupado := False;
      Tabla[I].ListaColision.Crear(avTipoClave, MaxSizeLC);
    End;
    Q_Ocupados := 0;
    Q_Claves := 0;
    Q_ClavesLC := 0;
    TDatoDeLaClave := avTipoClave;
    TFuncionHash := avTipoFuncionHash;
    Size := alSize;
    NPrimo := alNroPrimo;
    NSize := alSize;
    Crear := OK;
  End;
End;

// Control de tabla vacia
Function TablaHash.EsVacia(): Boolean;
Begin
  EsVacia := (Q_Ocupados = 0);
End;

// Control de Tabla Llena
Function TablaHash.EsLLena(): Boolean;
Begin
  EsLlena := (Q_Ocupados = Size);
End;

// Esta es la funcion de transformación HASH
Function FuncionTransformacion(X: TipoElemento; TFH: TipoFuncionesHash):PosicionTabla;
Var S, S1, S2: String;
    P: LongInt;
    D: Int64;
Begin
  FuncionTransformacion := PosNula;

  // Funcion Hash x modulo
  if TFH = Modulo then Begin
    FuncionTransformacion := (X.Clave Mod NPrimo);
  End;

  // Funcion Hash x plegamiento
  if TFH = Plegamiento then Begin
    if Length(VarToStr(X.Clave)) < Length(IntToStr(NSize)) Then Begin
      if X.Clave > NSize then FuncionTransformacion := (X.Clave Mod NSize)
      Else FuncionTransformacion := X.Clave;
    End
    Else Begin
      S := VarToStr(X.Clave);
      S1 := S.Substring(0, (Length(S) Div 2));
      S2 := S.Substring((Length(S) Div 2), Length(S));
      P := StrToInt(S1) + StrToInt(S2);
      if P > NSize then Begin
        P := (P Mod NSize);
      end;
      FuncionTransformacion := P;
    End;
  End;

  // Funcion Hash x Mitad del Cuadrado
  If TFH = MitadDelCuadrado then Begin
    D := (X.Clave * X.Clave);
    S := VarToStr(D);
    if Length(S) <= Length(IntToStr(NSize)) Then Begin
      if D < NSize then FuncionTransformacion := D
      Else FuncionTransformacion := (D Mod NSize);
    End
    Else Begin
      S := S.Substring(0, Length(IntToStr(NSize)));
```

```
      P := StrToInt(S);
      if P > NSize then Begin
        S := S.Substring(0, Length(IntToStr(NSize)) - 1);
        P := StrToInt(S);
      End;
      FuncionTransformacion := P;
    End;
  End;
End;


// La funcion insertar primero ubica la posicion y se fija si esta libre
// En caso de estar ocupada lo agrega en la lista de colisiones
Function TablaHash.Insertar(X:TipoElemento): Resultado;
Var P:PosicionTabla;
    Q:Variant;
Begin
  // Verifica la clave compatible
  if X.TipoDatoClave (X.Clave) <> TDatoDeLaClave  then Begin
    Insertar := ClaveIncompatible;
    Exit;
  End;
  // Ahora Controla que la Tabla NO este LLena
  if EsLlena() then begin
    Insertar := Llena;
    Exit;
End;
  // Ahora Controla que la Clave Ya No Exista
  if Buscar(X, Q) <> PosNula then Begin
    Insertar := ClaveDuplicada;
    Exit;
  End;
  // Ahora la inserta
  P := FuncionTransformacion(X, TFuncionHash);
  If P = PosNula Then Insertar := CError
  Else Begin
    If Tabla[P].Ocupado = False Then Begin
      Tabla[P].Clave := X;
      Tabla[P].Ocupado := True;
      Inc(Q_Ocupados);
    End
    Else Begin
      If Tabla[P].ListaColision.Agregar(X) <> OK Then Begin
        Insertar := CError;
        Exit;
      End
      Else Begin
        Inc(Q_ClavesLC);
      End;
    End;
    Inc(Q_Claves);
    Insertar := OK;
  End;
End;

// Primero busca si la clave existe.
// Luego si existe la elimina controlando si existe o NO lista de colisiones
Function TablaHash.Eliminar(X:TipoElemento): Resultado;
Var P: PosicionTabla;
    Q: PosicionLista;
Begin
  Eliminar := CError;
  // Llamo al FH
  P := FuncionTransformacion(X, TFuncionHash);
  If P <> PosNula Then Begin
    If Tabla[P].Ocupado = True Then
      If X.Clave = Tabla[P].Clave.Clave Then
        // Si tiene claves en colision debo tomar la primer clave
        // de la lista y pasarla a la tabla hash
        If Not Tabla[P].ListaColision.EsVacia Then begin
          X := Tabla[P].ListaColision.Recuperar(Tabla[P].ListaColision.Comienzo);
          Tabla[P].Clave := X;
```

```
          Tabla[P].ListaColision.Eliminar(Tabla[P].ListaColision.Comienzo);
          Dec(Q_Claves);
          Dec(Q_ClavesLC);
          Eliminar := OK;
        End
        Else Begin
          // La clave esta en la tabla y no hay colisiones
          Tabla[P].Ocupado := False;
          Dec(Q_Ocupados);
          Dec(Q_Claves);
          Eliminar := OK;
        End
      Else Begin
        // La clave esta en la lista de colision la borro de la lista simplemente
        Q := Tabla[P].ListaColision.Buscar(X);
        If Q <> Nulo Then Begin
          Tabla[P].ListaColision.Eliminar(Q);
          Dec(Q_Claves);
          Dec(Q_ClavesLC);
          Eliminar := OK;
        End;
      End;
  End;
End;

// Busca la clave segun la posicion que retorna la funcion hash
// Si no esta la busca en la lista de colisiones
Function TablaHash.Buscar(X:TipoElemento; Var PL:Variant): PosicionTabla;
Var P: PosicionTabla;
Begin
  Buscar := PosNula;
  PL := Nulo;
  P := FuncionTransformacion(X, TFuncionHash);
  // Posicion valida de la tabla
  If P <> PosNula Then Begin
    If Tabla[P].Ocupado = True Then
      If X.Clave = Tabla[P].Clave.Clave Then Buscar := P
      Else Begin
        // Busca si esta en la lista de colisiones
        If Not Tabla[P].ListaColision.EsVacia() Then begin
          PL := Tabla[P].ListaColision.Buscar(X);
          If PL <> Nulo Then Buscar := P;
        End;
      End;
  End;
End;

// recupera la clave completa de la tabla o lista de colisiones
Function TablaHash.Recuperar(P: PosicionTabla; PL: Variant): TipoElemento;
Var X: TipoElemento;
Begin
  Recuperar := X.TipoElementoVacio;
  If P <> PosNula Then Begin
    If PL <> Nulo Then
      // Toma la clave de las listas de colisiones
      Recuperar := Tabla[P].ListaColision.Recuperar(PL)
    Else
      // La clave esta directamente en la tabla
      Recuperar := Tabla[P].Clave;
  End;
End;

// retorno toda la tabla como un string para ponerlo directamente
// en memo, con su lista de colisiones tambien
Function TablaHash.RetornarClaves(): String;
Var X: TipoElemento;
    I: Integer;
    S: String;
    SS:String;
    Q: PosicionLista;
Begin
  SS := '';
```

```
  For I := MinTable To Size Do Begin
    If Tabla[I].Ocupado = True Then Begin
      X := Tabla[I].Clave;
      S := X.ArmarString;
      SS:= SS + S + cCRLF;
      If Not Tabla[I].ListaColision.EsVacia() Then Begin
        Q := Tabla[I].ListaColision.Comienzo;
        While Q <> Nulo Do Begin
          X := Tabla[I].ListaColision.Recuperar(Q);
          S := X.ArmarString;
          SS:= SS + cTab + 'LC: ' + S + cCRLF;
          Q := Tabla[I].ListaColision.Siguiente(Q);
        End;
      End;
    //SS:= SS + cCR;
    End;
  End; // del for
  RetornarClaves := SS;
End;

// LLena la tabla con claves Random
Function TablaHash.LLenarClavesRandom (alSize, alNroPrimo: LongInt; RangoDesde, RangoHasta: LongInt
): Resultado;
Var X: TipoElemento;
    I: LongInt;
Begin
  TDatoDeLaClave := Numero;
  If Crear(TDatoDeLaClave, Modulo, alSize, alNroPrimo) <> OK Then Begin
    LLenarClavesRandom := CError;
    Exit;
  End;
  // Ahora la llena random
  X.Inicializar(TDatoDeLaClave,'');
  Randomize;
  //while Not EsLlena() do Begin
  For I:= MinTable To alSize Do Begin
    X.Clave := RangoDesde + Random(RangoHasta);
    Insertar(X);
  End;
  LLenarClavesRandom := OK;
End;

// Propiedad que retorna la cantidad de claves de la tabla
// Incluye todas la claves
Function TablaHash.CantidadClaves(): LongInt;
Begin
  CantidadClaves := Q_Claves;
End;

// Propiedad que retorna la cantidad de posiciones de la tabla ocupadas
Function TablaHash.CantidadOcupados(): LongInt;
Begin
  CantidadOcupados := Q_Ocupados;
End;

// Propiedad que retorna la cantidad de claves de la lista de Colisiones
Function TablaHash.CantidadClavesZO(): LongInt;
Begin
  CantidadClavesZO := Q_ClavesLC;
End;

// retorna la proxima posicion ocupada de la tabla a partir de una posicion
Function TablaHash.ProximaPosicionOcupada(P: PosicionTabla): PosicionTabla;
Var Q: PosicionTabla;
Begin
  ProximaPosicionOcupada := PosNula;

  if P <> PosNula then Begin
    for Q := (P + 1) to Size do Begin
      if Tabla[Q].Ocupado = true then Begin
        ProximaPosicionOcupada := Q;
        Exit;
```

```
      End;
    End;
  End;
End;

// retorna la primer posicion ocupada de la tabla comenzando desde el inicio
Function TablaHash.PrimerPosicionOcupada(): PosicionTabla;
Var Q:PosicionTabla;
Begin
  PrimerPosicionOcupada := PosNula;
  for Q := MinTable to Size do Begin
    if Tabla[Q].Ocupado = true then Begin
      PrimerPosicionOcupada := Q;
      Exit;
    End;
  End;
End;

// Retorna la lista de colisiones de una posicion
Function TablaHash.RetornarLC(P: PosicionTabla): Lista;
Begin
  RetornarLC.Crear(TDatoDeLaClave, MaxSizeLC);
  if P <> PosNula then Begin
    if Tabla[P].Ocupado then RetornarLC := Tabla[P].ListaColision;
  End;
End;

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

Function TablaHash.FuncionHash: TipoFuncionesHash;
Begin
  FuncionHash := TFuncionHash;
End;

Function TablaHash.TableSize(): LongInt;
Begin
  TableSize := Size;
End;

Function TablaHash.MaxTableSize(): LongInt;
Begin
  MaxTableSize := MaxTable;
End;

Function TablaHash.NroPrimo(): LongInt;
Begin
  NroPrimo := NPrimo;
End;

end.
```