

```
unit TadHashZO;
```

```
interface
```

```
uses
```

```
Tipos, ListArray, Variants, SysUtils;
```

```
Const
```

```
MinTable = 0;    // Minima posicion de la tabla
MaxTable = 2000; // Maxima posicion de la tabla
MaxSizeLC= 10;   // Tamaño Maximo Lista de Colisiones
PosNula = -1;    // Posicion NO valida de la tabla
```

```
Type
```

```
PosicionTabla = LongInt;
```

```
TipoRegistroTabla = Object
```

```
Clave : TipoElemento;
```

```
Ocupado : Boolean;
```

```
End;
```

```
TablaHash = Object
```

```
Private
```

```
Tabla: Array of TipoRegistroTabla;
```

```
ZO: Array of TipoRegistroTabla;
```

```
Q_Ocupados: Integer;
```

```
Q_Claves: Integer;
```

```
Q_ClavesZO: Integer;
```

```
TDatoDeLaClave: TipoDatosClave;
```

```
TFuncionHash: TipoFuncionesHash;
```

```
Size: LongInt;
```

```
// funciones internas del objeto
```

```
Function EliminarZO(X:TipoElemento): Boolean;
```

```
Function BuscarZO(X:TipoElemento): PosicionTabla;
```

```
Public
```

```
Function Crear(avTipoClave: TipoDatosClave; avTipoFuncionHash: TipoFuncionesHash; alSize: LongInt; alNroPrimo: LongInt): Resultado;
```

```
Function EsVacía(): Boolean;
```

```
Function EsLLena(): Boolean;
```

```
Function Insertar(X:TipoElemento): Resultado;
```

```
Function Eliminar(X:TipoElemento): Resultado;
```

```
Function Buscar(X:TipoElemento; Var MarcaZO:Variant): PosicionTabla;
```

```
Function Recuperar(P: PosicionTabla; MarcaZO: Variant): TipoElemento;
```

```
Function RetornarClaves(): String;
```

```
Function LLenarClavesRandom(alSize, alNroPrimo: LongInt; RangoDesde, RangoHasta: LongInt): Resultado;
```

```
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 hash
```

```
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: Integer;
```

```

Begin
  if alSize <= MinTable then Crear:= CError;
  if alSize > MaxTable then Crear:= CError;
  if (alSize >= MinTable) And (alSize <= MaxTable) then Begin
    SetLength(Tabla, alSize);
    SetLength(ZO, alSize);
    For I:= MinTable To alSize Do Begin
      Tabla[I].Ocupado := False;
      Tabla[I].Clave.Valor2 := NIL;
    End;
    for I := MinTable To (alSize div 2) Do Begin
      ZO[I].Ocupado := False;
      ZO[I].Clave.Valor2 := NIL;
    End;
    Q_Ocupados := 0;
    Q_Claves := 0;
    Q_ClavesZO:= 0;
    TdatoDeLaClave := avTipoClave;
    TFuncionHash := avTipoFuncionHash;
    Size := alSize;
    NPrimo := alNroPrimo;
    NSize := alSize;
    Crear := OK;
  End;
End;

// Tabla vacia sin claves
Function TablaHash.EsVacía(): Boolean;
Begin
  EsVacía := (Q_Ocupados = 0);
End;

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);
  End;

```

```
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 secuencialmente en la ZO o en el primer libre
Function TablaHash.Insertar(X:TipoElemento): Resultado;
Var P, Q: PosicionTabla;
    K: Variant;
    Encontre: Boolean;
Begin
    // Verifica la clave compatible
    if X.TipoDatoClave (X.Clave) <> TDatoDeLaClave then Begin
        Insertar := ClaveIncompatible;
        Exit;
    End;
    // Verifico que no este llena
    If EsLLena() then Begin
        Insertar := Llena;
        Exit;
    End;
    // Ahora Controla que la Clave Ya No Exista
    if Buscar(X, K) <> PosNula then Begin
        Insertar := ClaveDuplicada;
        Exit;
    End;
    // Tomo la posicion de la tabla donde insertar
    P := FuncionTransformacion(X, TFuncionHash);
    // Si es NULO entonces la funcion Hash retorno error
    If P = PosNula Then Insertar := CError
    Else Begin
        // Controla si puede poner la clave en la tabla
        If Tabla[P].Ocupado = False Then Begin
            Tabla[P].Clave := X;
            Tabla[P].Ocupado := True;
            Inc(Q_Ocupados);
        End
        Else Begin
            // Busco un lugar libre en la zona de Overflow
            Q := MinTable;
            Encontre := False;
            While (Q <= (Size Div 2)) And (Not(Encontre)) Do Begin
                If ZO[Q].Ocupado = True Then
                    Inc(Q)
                Else
                    Encontre := True;
            End;
            // Si Hay un Lugar Libre en la ZO la inserta
            If Encontre Then Begin
                ZO[Q].Clave := X;
                ZO[Q].Ocupado := True;
                Inc(Q_ClavesZO);
            End
            Else Begin
                // Tabla ZO llena no hay lugar para poner la clave
                Insertar := Llena;
                Exit;
            End;
        End;
    End;
End;
```

```

    Inc(Q_Claves);
    Insertar := OK;
End;
End;

// Busco y Elimina una Clave de la ZO - Funcion Interna del Objeto
Function TablaHash.EliminarZO(X:TipoElemento): Boolean;
Var Q: PosicionTabla;
    Encontre: Boolean;
Begin
    EliminarZO:= False;
    // Busco la Clave en la ZO para ver si existe y eliminarla
    Q := MinTabla;
    Encontre := False;
    While (Q <= (Size Div 2)) And (Not(Encontre)) Do Begin
        If X.Clave = ZO[Q].Clave.Clave Then
            Encontre := True
        Else
            Inc(Q);
    End;
    // Verifico si la encontro
    If Encontre Then Begin
        ZO[Q].Ocupado := False;
        ZO[Q].Clave.Valor2 := NIL;
        Dec(Q_ClavesZO);
        EliminarZO := True;
    End;
End;

// Primero busca si la clave existe.
// Luego si existe la elimina controlando si existe en la tabla o en la ZO
Function TablaHash.Eliminar(X:TipoElemento): Resultado;
Var P: PosicionTabla;
    Q: PosicionTabla;
    Encontre: Boolean;
Begin
    Eliminar := CError;
    // saca la Posicion de la Tabla
    P := FuncionTransformacion(X, TFuncionHash);
    // Si la FH encontro una posicion
    If P <> PosNula Then Begin
        If Tabla[P].Ocupado = True Then Begin
            // Si la clave esta en la tabla libero la posicion
            If X.Clave = Tabla[P].Clave.Clave Then Begin
                Tabla[P].Ocupado := False;
                Tabla[P].Clave.Valor2 := NIL;
                Dec(Q_Ocupados);
                Dec(Q_Claves);
                Eliminar := OK;
            End
        Else Begin
            // Busco si esta en la ZO
            If EliminarZO(X) Then Begin
                Dec(Q_Claves);
                Eliminar := OK;
            End;
        End;
    End
    Else Begin
        // Busco si esta en la ZO
        If EliminarZO(X) Then Begin
            Dec(Q_Claves);
            Eliminar := OK;
        End;
    End;
End;
End;

// Busco una Clave en la ZO - Funcion Interna del Objeto

```

```

Function TablaHash.BuscarZO(X:TipoElemento): PosicionTabla;
Var Q: PosicionTabla;
    Encontre: Boolean;
Begin
    BuscarZO:= PosNula;
    // Busco la Clave en la ZO para ver si existe y eliminarla
    Q := MinTable;
    Encontre := False;
    While (Q <= (Size Div 2)) And (Not(Encontre)) Do Begin
        If ZO[Q].Ocupado = True Then Begin
            If X.Clave = ZO[Q].Clave.Clave Then
                Encontre := True
            Else
                Inc(Q);
        End
        Else
            Inc(Q);
    End;
    // Verifico si la encuentro
    If Encontre Then BuscarZO := Q;
End;

// Busca la clave segun la posicion que retorna la funcion hash
// Si no esta la busca en la ZO y pone la Marca en True para Saber que tabla esta
Function TablaHash.Buscar(X:TipoElemento; Var MarcaZO:Variant): PosicionTabla;
Var P, Q: PosicionTabla;
Begin
    Buscar := PosNula;
    MarcaZO:= False;
    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
            // La Busca en la ZO
            P := BuscarZO(X);
            If P <> PosNula Then Begin
                Buscar := P;
                MarcaZO := True;
            End;
        End
    Else Begin
        // La Busca en la ZO
        P := BuscarZO(X);
        If P <> PosNula Then Begin
            Buscar := P;
            MarcaZO := True;
        End;
    End;
End;
End;

// recupera la clave completa de la tabla o ZO
Function TablaHash.Recuperar(P: PosicionTabla; MarcaZO: Variant): TipoElemento;
Var X: TipoElemento;
Begin
    Recuperar := X.TipoElementoVacio;
    If P <> PosNula Then Begin
        If MarcaZO Then
            // Toma la clave de la ZO
            Recuperar := ZO[P].Clave
        Else
            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;
Begin
    SS := '';
    // recorro la Tabla
    For I := MinTable To Size Do Begin
        If Tabla[I].Ocupado = True Then Begin
            X := Tabla[I].Clave;
            S := X.ArmarmString;
            SS:= SS + S + cCRLF;
        End;
    End;

    SS:= SS + cCRLF;
    SS:= SS + 'Claves ZO: ' + cCRLF;

    // Ahora Recorro la ZO
    For I := MinTable To (Size Div 2) Do Begin
        If ZO[I].Ocupado = True Then Begin
            X := ZO[I].Clave;
            S := X.ArmarmString;
            SS:= SS + S + cCRLF;
        End;
    End;
    // Retorno las claves concatenadas
    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;
    // La llena random
    X.Inicializar(TdatoDeLaClave, '');
    Randomize;
    For I:= MinTable To Size 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 en la zona de overflow
Function TablaHash.CantidadClavesZO(): LongInt;
Begin
    CantidadClavesZO := Q_ClavesZO;
End;

```

```

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

// Proxima posicion ocupada de la tabla desde <P>
Function TablaHash.ProximaPosicionOcupada(P: PosicionTabla): PosicionTabla;
Var I: PosicionTabla;
Begin
    ProximaPosicionOcupada := PosNula;
    if P = PosNula then Exit;
    for I := (P + 1) to Size do Begin
        if Tabla[I].Ocupado then Begin
            ProximaPosicionOcupada := I;
            Exit;
        End;
    End;
End;

// Retorna la zona de overflow como una lista
Function TablaHash.RetornarLC(P: PosicionTabla): Lista;
Var I: PosicionTabla;
Begin
    RetornarLC.Crear(TDatoDeLaClave, (Size Div 2));
    for I := MinTable to (Size Div 2) do Begin
        if ZO[I].Ocupado then Begin
            RetornarLC.Agregar(ZO[I].Clave);
        End;
    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.

```