```
Unit ListCursor;

Interface
  Uses Tipos, stdctrls, SysUtils, Variants;

Const
  MIN = 1;  // Inicio del Cursor
  MAX = 2000; // Fin del cursor
  NULO= 0; // Indica posicion invalida de la lista

Type
  PosicionLista = Integer;

  Nodo_Lista = Object
    Datos: Tipoelemento;
    Prox, Ante: PosicionLista;
  End;

  Lista = Object
    Private
      Cursor: Array Of Nodo_Lista;
      Inicio, Final, Libre: PosicionLista;
      Q_Items: LongInt;
      TDatoDeLaClave: TipoDatosClave;
      Size: LongInt;
      // Comportamiento privado
      Procedure Intercambio (P,Q: PosicionLista);
    Public
      // Comportamiento del objeto (Operaciones del TAO)
      Function Crear(avTipoClave: TipoDatosClave; alSize: LongInt): Resultado;
      Function EsVacia(): Boolean;
      Function EsLLena(): Boolean;
      Function Agregar(X:TipoElemento): Resultado;
      Function Insertar(X:TipoElemento; P:PosicionLista): Resultado;
      Function Eliminar(P:PosicionLista): Resultado;
      Function Buscar(X:TipoElemento):PosicionLista;
      Function Siguiente(P:PosicionLista): PosicionLista;
      Function Anterior(P:PosicionLista): PosicionLista;
      Function Ordinal(PLogica: Integer): PosicionLista;
      Function Recuperar(P:PosicionLista): TipoElemento;
      Function Actualizar(X:TipoElemento; P:PosicionLista): Resultado;
      Function ValidarPosicion(P:PosicionLista): Boolean;
      Function RetornarClaves(): String;
      Function LlenarClavesRandom(alSize: LongInt; RangoDesde, RangoHasta: LongInt): Resultado;
      Function Comienzo: PosicionLista;
      Function Fin: PosicionLista;
      Function CantidadElementos: LongInt;
      Function DatoDeLaClave: TipoDatosClave;
      Function SizeList(): LongInt;
      Function MaxSizeList(): LongInt;
      Procedure Sort(Ascendente: Boolean);
  End;


Implementation

// Crea la lista vacia
Function Lista.Crear(avTipoClave: TipoDatosClave; alSize: LongInt): Resultado;
Var Q: PosicionLista;
Begin
  if alSize < Min then Crear:= CError;
  if alSize > Max then Crear:= CError;
  if (alSize >= Min) And (alSize <= Max) then Begin
    SetLength(Cursor, (alSize + 1));  // La posicion 0 no la usamos pero hay que tenerla en cuenta
    For Q := MIN To (alSize - 1) Do // Encadenamientos de Libres
      Cursor[Q].Prox := Q + 1;
    Cursor[alSize].Prox := NULO;
    Libre := MIN;
    Inicio := NULO;
    Final := NULO;
    Q_items := 0;
    TDatoDeLaClave := avTipoClave;
```

```
    Size := alSize;
    Crear := OK;
  End;
End;

// Control de lista vacia
Function Lista.EsVacia(): Boolean;
Begin
  EsVacia := (Inicio = NULO);
End;

// Control de lista llena
Function Lista.EsLlena(): Boolean;
Begin
  EsLlena := (Libre = NULO);
End;

// Proximo de P o NULO
Function Lista.Siguiente (P:PosicionLista): PosicionLista;
Var Q: PosicionLista;
Begin
  If EsVacia() Or (P = NULO) Then Q := NULO
  Else Q := Cursor[P].Prox;
  Siguiente := Q;
End;

// Anterior de P o NULO
Function Lista.Anterior (P:PosicionLista): PosicionLista;
  Var Q: PosicionLista;
  Begin
  If EsVacia() Or (P = NULO) Then Q := NULO
  Else Q := Cursor[P].Ante;
  Anterior := Q;
End;

// Agrega elementos al final de la lista. Despues del ultimo.
Function Lista.Agregar(X:TipoElemento): Resultado;
Var Q: PosicionLista;
Begin
  Agregar := CError;
  // Controla que el Tipo de Dato de la Clave sea Homogeneo a la Lista
  if X.TipoDatoClave(X.Clave) <> TDatoDeLaClave then Begin
    Agregar := ClaveIncompatible;
    Exit;
  End;
  // Ahora lo agrega
  If Not(Esllena()) Then Begin
    Q := Libre; // Tomo el primer libre disponible
    Libre := Cursor[Libre].Prox; // Encadeno el resto de los libres
    Cursor[Q].Datos := X;    // Q^.Datos
    Cursor[Q].Prox := NULO;
    Cursor[Q].Ante := Final;
    If EsVacia() Then Inicio := Q // Controlo si es el primer elemento de la lista
    Else Cursor[Final].Prox := Q;
    Final := Q;
    Inc(Q_items);
    Agregar := OK;
  End
  Else
    Agregar := Llena;
End;

// Inserta un elemento en cualquier lugar de la lista.
Function Lista.Insertar(X:TipoElemento; P:PosicionLista): Resultado;
Var Q: PosicionLista;
Begin
  Insertar := CError;
  // Controla que el Tipo de Dato de la Clave sea Homogeneo a la Lista
  if X.TipoDatoClave(X.Clave) <> TDatoDeLaClave then Begin
    Insertar := ClaveIncompatible;
    Exit;
  End;
```

```
  // Ahora lo Inserta
  If Esllena() Then Insertar := LLena
  Else Begin
    If ValidarPosicion(P) Then Begin
      Q := Libre; // Tomo el primer libre disponible
      Libre := Cursor[Libre].Prox; // Encadeno el resto de los libres
      Cursor[Q].Datos := X;
      Cursor[Q].Prox := P;
      Cursor[Q].Ante := Cursor[P].Ante;
      Cursor[P].Ante := Q;
      If P = Inicio Then Inicio := Q
      Else Cursor[Cursor[Q].Ante].Prox := Q;
      Inc(Q_items);
      Insertar := OK;
    End
    Else
      Insertar := PosicionInvalida;
  End;
End;

// Elimina un elemento de cualquier posicion de la lista.
Function Lista.Eliminar (P:PosicionLista): Resultado;
Var Q: PosicionLista;
Begin
  Eliminar := CError;
  If EsVacia() Then Eliminar := Vacia
  Else Begin
    If ValidarPosicion(P) Then Begin
      Q := P;
      If (P = Inicio) And (P = Final) Then Crear(TDatoDeLaClave, Size) // Unico de la lista. Se cre
a vacia
      Else Begin
        If (P = Inicio) Then Begin // Se elimina el primer elemento. Cambia el inicio
          Inicio := Cursor[Inicio].Prox;
          cursor[Inicio].Ante := NULO;
        End
        Else If (P = Final) Then Begin // Se elimina el ultimo elemento. Cambia el Final
              Final := cursor[Final].Ante;
              Cursor[Final].Prox := NULO;
            End
            Else Begin // Se elimina en cualquier otro lugar que no es ni Inicio, ni Final
              Cursor[Cursor[P].Ante].Prox := Cursor[P].Prox;
              Cursor[Cursor[P].Prox].Ante := Cursor[P].Ante;
            End;
        Cursor[Q].Prox := Libre; // retorno a Libres el Nodo Eliminado
        Libre := Q;
      End;
      Dec(Q_Items);
      Eliminar := OK;
    End
  Else
    Eliminar := PosicionInvalida;
  End;
End;

// Busca una clave en la lista desde inicio a fin.
// Retorna la posicion de la primer ocurrencia o NULO
Function Lista.Buscar(X:TipoElemento): PosicionLista;
Var Q: PosicionLista;
    Encontre: Boolean;
Begin
  // Asumimos que no existe
  Buscar := NULO;
  // Controla que el Tipo de Dato de la Clave sea Homogeneo a la Lista
  if X.TipoDatoClave(X.Clave) <> TDatoDeLaClave then Begin
    Exit;
  End;
  // comienzo a Buscar desde inicio a fin
  Q := Inicio;
  Encontre := False;
  While (Q <> NULO) And (Not(Encontre)) Do Begin
    If Cursor[Q].Datos.Clave <> X.Clave Then Q := Cursor[Q].Prox
```

```
    Else Encontre := True;
  End;
  // Si la encontro retorno Q = Posicion del Primer elemento encontrado.
  If Encontre Then Buscar := Q;
End;

// Recupera el elemento de la posicion P
Function Lista.Recuperar(P:PosicionLista): TipoElemento;
Var X: TipoElemento;
Begin
  Recuperar := X.TipoElementoVacio;
  If (P <> NULO) AND ValidarPosicion(P) Then Begin
    Recuperar := Cursor[P].Datos;
  End
End;

// Actualiza La posicion P. Sobreescribe todo el elemento sin importar su contenido
Function Lista.Actualizar (X:TipoElemento; P:PosicionLista): Resultado;
Begin
  Actualizar := CError;
  // Controla que el Tipo de Dato de la Clave sea Homogeneo a la Lista
  if X.TipoDatoClave(X.Clave) <> TDatoDeLaClave then Begin
    Actualizar := ClaveIncompatible;
    Exit;
  End;
  // ahora Actualiza
  If EsVacia() Then Actualizar := Vacia
  Else Begin
    If (P <> NULO) And ValidarPosicion(P) Then Begin
      Cursor[P].Datos := X;
      Actualizar := OK;
    End
  Else
    Actualizar := PosicionInvalida;
  End;
End;

// Retorna la posicion fisica de la lista a partir de la posicion ordinal
// de inicio a final
Function Lista.Ordinal(PLogica: Integer): PosicionLista;
Var Q: PosicionLista;
    I: Integer;
Begin
  Ordinal := Nulo;
  If Not EsVacia() Then Begin
    I := 1;
    Q := Inicio;
    While (I < PLogica) And (Q <> Nulo) Do Begin
      Inc(I);
      Q := cursor[Q].Prox;
    End;
    If Q <> Nulo Then Ordinal := Q;
  End;
End;

// Valida que la posicion P pertenezca a esta lista
Function Lista.ValidarPosicion(P:PosicionLista): Boolean;
Var Q: PosicionLista;
    Encontre: Boolean;
Begin
  ValidarPosicion := False;
  If Not EsVacia() Then Begin
    Q := Inicio;
    Encontre := False;
    While (Q <> Nulo) And Not(Encontre) Do Begin
      If Q = P Then Encontre := True
      Else Q := cursor[Q].Prox;
    End;
    // Si encontro la posicion retorna verdadero
    If Encontre Then ValidarPosicion := True;
  End;
End;
```

```
// Intercambia el TE de 2 posiciones
Procedure Lista.Intercambio (P,Q: PosicionLista);
Var X1, X2: TipoElemento;
Begin
  X1 := Cursor[P].Datos;
  X2 := Cursor[Q].Datos;
  Cursor[P].Datos := X2;
  Cursor[Q].Datos := X1;
End;

// Ordena la lista. Ordena los strings sencible a Mayusculas y Minusculas
Procedure Lista.Sort(Ascendente: Boolean);
Var P, Q: PosicionLista;
    X1, X2: TipoElemento;
Begin
  // Ordeno por metodo de burbuja
  P := Inicio;
  While P <> Nulo Do Begin
    Q := Inicio;
    While Q <> Nulo Do begin
      X1 := Cursor[Q].Datos;
      If Siguiente(Q) <> Nulo Then Begin
        X2 := Cursor[Cursor[Q].Prox].Datos;
        if Ascendente then if X1.Clave > X2.Clave then Intercambio(Q, Cursor[Q].Prox);
        if Not Ascendente then if X1.Clave < X2.Clave then Intercambio(Q, Cursor[Q].Prox);
      End;
      Q := Cursor[Q].Prox;
    End;
    P := Cursor[P].Prox;
  End;
End;


// Retorna toda la lista en un string para luego mostrarla en un memo
// Directamente
Function Lista.RetornarClaves(): String;
Var Q: PosicionLista;
    X: TipoElemento;
    S: String;
    SS:String;
Begin
  SS := '';
  Q := Inicio;
  While Q <> Nulo Do Begin
    X := Recuperar(Q);
    S:= X.ArmarString; // Concatena separado por TAB los atributos de X
    SS := SS + S + cCRLF; // Retorno que carro + fin de linea
    Q := Siguiente(Q);
  End;
  RetornarClaves := SS;
End;

// Llena la lista de 0 a <RangoHasta> el atributo DI de la lista
Function Lista.LlenarClavesRandom(alSize: LongInt; RangoDesde, RangoHasta: LongInt): Resultado;
Var X: TipoElemento;
Begin
  TDatoDeLaClave := Numero;
  // Siempre la Crea vacia como Numero
  If Crear(TDatoDeLaClave, alSize) <> OK Then Begin
    LlenarClavesRandom := CError;
    Exit;
  End;
  // Ahora la llana Random
  X.Inicializar(TDatoDeLaClave, '');
  Randomize;
  While Not EsLLena Do Begin
    X.Clave := RangoDesde + Random(RangoHasta);
    Agregar(X);
  End;
  LlenarClavesRandom := OK;
End;
```

```
Function Lista.Comienzo: PosicionLista;
Begin
  Comienzo := Inicio;
End;

Function Lista.Fin: PosicionLista;
Begin
  Fin := Final;
End;

Function Lista.CantidadElementos: LongInt;
Begin
  CantidadElementos := Q_Items;
End;

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

Function Lista.SizeList(): LongInt;
Begin
  SizeList := Size;
End;

Function Lista.MaxSizeList(): LongInt;
Begin
  MaxSizeList := MAX;
End;

End.
```