```
unit ListArray;

interface
  Uses Tipos, stdctrls, SysUtils, Variants;

Const
  MIN = 1;  // Inicio de la Lista
  MAX = 2000; // Cota de tamaño maximo
  Nulo= 0; // Indica posicion invalida de la lista

Type
  PosicionLista = LongInt;

  Lista = Object
    Private
      Elementos: Array Of TipoElemento;
      Inicio, Final: PosicionLista;   // Bandera de inicio y fin de la lista
      Q_Items: Integer;               // Lleva la cantidad de elementos de la lista
      TDatoDeLaClave: TipoDatosClave; // Tipo de Dato de la Clave recibida en el crear lista vacia
      Size: LongInt;                  // Tamaño de la Lista
      // Comportamiento privado
      Procedure Intercambio (P,Q: PosicionLista);   // intercambia el tipoelemento de 2 posiciones
de la lista
    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;


// Escribir la implementación del Objeto LISTA
implementation

// Crea la lista vacia
Function Lista.Crear(avTipoClave: TipoDatosClave; alSize: LongInt): Resultado;
Begin
  if alSize < Min then Crear:= CError;
  if alSize > Max then Crear:= CError;
  if (alSize >= Min) And (alSize <= Max) then Begin
    Inicio := Nulo;
    Final := Nulo;
    Q_Items := 0;
    TDatoDeLaClave := avTipoClave;
    Size := alSize;
    SetLength(Elementos, (alSize+1));
    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 := (Final = Size);
End;


// Agrega un Items al Final de Lista
Function Lista.Agregar(X:TipoElemento): Resultado;
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 proceso a intentar agregarlo
  If EsLlena Then
    Agregar := LLena
  Else Begin
    Final := Final + 1;
    Elementos[Final] := X;
    Inc(Q_Items);
    If EsVacia Then Inicio := Final;
    Agregar := OK;
  End;
End;


// Inserta un Item entre el Inicio y el Final de 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 Inserto
  If EsLLena Then
    Insertar := LLena
  Else Begin
    If ValidarPosicion(P) Then Begin
      For Q := Final DownTo P Do
        Elementos[Q+1] := elementos[Q]; // Genera el hueco dentro de la lista
      // Ahora pego el dato en la posicion recibida (P)
      Elementos[P] := X;
      Inc(Final);
      Inc(Q_Items);
      Insertar := OK;
    End
    Else
      Insertar := PosicionInvalida;
  End;
End;


// Retorno la posicion fisica que se corresponde con la logica
// En este caso particular coinciden ambas
Function Lista.Ordinal(PLogica: Integer): PosicionLista;
Begin
  Ordinal := Nulo;
  if (PLogica > 0) And (PLogica <= Q_Items) then Ordinal := PLogica;
End;
```

```
// Elimina un item de la lista. Cualquier posicion Valida
Function Lista.Eliminar(P:PosicionLista): Resultado;
Var Q: PosicionLista;
Begin
  Eliminar := CError;
  If EsVacia Then
    Eliminar := Vacia
  Else Begin
    If ValidarPosicion(P) Then Begin
      For Q := P To (Final - 1) Do
        Elementos[Q] := elementos[Q + 1]; // Aplasta el item a borrar
      // Actualizo el final y la cantidad
      Dec(Final);
      Dec(Q_Items);
      If Final < Inicio Then Crear(TDatoDeLaClave, Size); //Si borrar el unico que hay se crea vaci
a
      Eliminar := OK;
    End
  Else
    Eliminar := PosicionInvalida;
  End;
End;


// Busca un elemento dentro de la lista en función de un dato
// Retorna la posición del primer elemento que coincide o NULO según corresponda
Function Lista.Buscar(X:TipoElemento): PosicionLista;
Var Q: PosicionLista;
    Encontre: Boolean;
Begin
  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;

  // Arranca por el Inicio y con Q recorre la lista
  Encontre:= False;
  Q := Inicio;
  While (Q <> Nulo) And (Q <= Final) And Not(Encontre) Do Begin
    If Elementos[Q].Clave = X.Clave Then Encontre := True
    Else Q := Q + 1;
  End;

  // Controla si lo encontro, si es asi retorno la posicion Q
  If Encontre Then Buscar := Q;
End;


// retorna el siguiente de una Posicion siempre que sea valida
// SINO retorna NULO
Function Lista.Siguiente(P:PosicionLista): PosicionLista;
Begin
  Siguiente := Nulo;
  If ValidarPosicion(P) And (P < Final) Then Siguiente := P + 1;
End;


// retorna el anterior de una Posicion siempre que sea valida
// SINO retorna NULO
Function Lista.Anterior(P:PosicionLista): PosicionLista;
Begin
  Anterior := Nulo;
  If ValidarPosicion(P) And (P > Inicio) Then Anterior := P - 1;
End;


// retorna el <tipoelemento> de la posicion <P> o bien
// retorna un <tipoelemento> vacio
```

```
Function Lista.Recuperar(P:PosicionLista): TipoElemento;
Var X: TipoElemento;
Begin
  Recuperar := X.TipoElementoVacio;
  If ValidarPosicion(P) Then Begin
    Recuperar := Elementos[P];
  End;
End;


// Cambia un Item dentro de la lista
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 valida la posicion
  If ValidarPosicion(P) Then Begin
    Elementos[P] := X;
    Actualizar := OK;
  End;
End;


// Valida una posicion de la lista
Function Lista.ValidarPosicion(P:PosicionLista): Boolean;
Begin
  ValidarPosicion := False;
  If Not EsVacia and (P >= Inicio) And (P <= Final) Then ValidarPosicion := True;
End;


// Intercambia los datos entre 2 posiciones de la misma lista
Procedure Lista.Intercambio (P,Q: PosicionLista);
Var X1, X2: TipoElemento;
Begin
  X1 := Elementos[P];
  X2 := Elementos[Q];
  Elementos[P] := X2;
  Elementos[Q] := 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 clave por metodo de burbuja
  P := Inicio;
  While (P <> Nulo) Do Begin
    Q := Inicio;
    while (Q <> Nulo) Do Begin
      X1 := Elementos[Q];
      If Siguiente(Q) <> Nulo Then Begin
        X2 := Elementos[Q+1];
        // Si es Ascendente
        if Ascendente then
          if X1.Clave > X2.Clave Then Intercambio(Q, (Q+1));
        // Si es Descendente
        if Not Ascendente then
          if X1.Clave < X2.Clave Then Intercambio(Q, (Q+1));
      End;
      Q := Siguiente(Q);
    End;
    P := Siguiente(P);
  End;
End;
```

```
// Retorna un string con todos los elementos de lista
// Cada campo de cada item separado por Tabuladores
// Cada Item separado por Retorno de Carro + Final de Linea
Function Lista.RetornarClaves(): String;
Var Q: PosicionLista;
    X: TipoElemento;
    S, SS: String;
Begin
  SS:= '';
  Q := Inicio;
  While Q <> Nulo Do Begin
    X := Recuperar(Q);
    S := X.ArmarString;
    SS := SS + S + cCRLF;
    Q := Siguiente(Q);
  End;
  RetornarClaves := SS;
End;


// retorno el puntero (o posicion) del comienzo de la lista
Function Lista.Comienzo: PosicionLista;
Begin
  Comienzo := Inicio;
End;


// retorno el puntero (o posicion) del final de la lista
Function Lista.Fin: PosicionLista;
Begin
  Fin := Final;
End;


// retorno la cantidad de elementos de la lista
Function Lista.CantidadElementos: LongInt;
Begin
  CantidadElementos := Q_Items;
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
  LlenarClavesRandom := CError;
  TDatoDeLaClave := Numero;
  // Siempre la Crea vacia como Numero
  If Crear(TDatoDeLaClave, alSize) <> OK Then Begin
    LlenarClavesRandom := CError;
    Exit;
  End;
  X.Inicializar(Numero, '');
  Randomize;
  While Not EsLLena Do Begin
    X.Clave := RangoDesde + Random(RangoHasta);
    Agregar(X);
  End;
  LlenarClavesRandom := OK;
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.
```