TAD **TListaEnteros**;
{Este TAD representa a una lista simple de números enteros sin orden, agrega elementos al final de la lista}

**Interface**
Tipo exportado **listaE;**

Procedure **crearL**(var L:listaE);
{se crea una lista vacia}

Function **vaciaL**(L:listaE):boolean;
{devuelve True si la lista está vacía o llegó al final, False si tiene elementos}

Function **longitudL**(L:listaE):integer;
{Devuelve la cantidad de elementos que hay en la lista}

Procedure **agregarL**(var L:listaE; num:Integer);
{Agregar el elemento num al final de la lista L}

Procedure **IniciarRecorridoL**(Var L:listaE);
{Prepara a la lista para iniciar el recorrido, se posiciona en el primer elemento, la lista debe tener elementos}

Procedure **actualL**(var L:ListaE; var num:Integer);
{Devuelve en num el elemento actual de la lista L,el elemento num sigue estando en la lista, la lista debe tener al menos un elemento}

Procedure **siguienteL**(var L: listaE);
{Avanza al siguiente elemento de la lista L, la lista L debe tener al menos un elemento}

Procedure **asignarL**(var L1:listaE; L2:listaE);
{Devuelve en L1 la lista L2}

Function **existeL**(L: listaE, num:integer): boolean;
{Devuelve True si el elemento num existe en la lista L, retorna False en caso contrario}

Procedure **eliminarL** (var L: listaE, num:integer);
{Elimina de la lista L el elemento num, num debe existir en la lista}

**Implementation**

```
type Lista = ^Nodo_Lista;
      Nodo_Lista = record
                     Elem: Integer;
                     Siguiente:Lista;
                  end;
       listaE =record
                Primero,Ultimo, actual:Lista;
                CantElementos: integer;
            end;

Procedure crearL(var L:listaE);
begin
    L.Primero:=nil;
    L.Ultimo:=nil;
    L.actual := nil;
```

```
    L.CantElementos:=0;
end;



Function vaciaL(L:listaE):boolean;
begin
    vaciaL:=(L.Actual=nil);
end;


Function longitudL(L:listaE):integer;
Begin
  longitudL:=L.CantElementos;
End;


Procedure agregarL(var L:listaE; num:Integer);
Var pAux: Lista;
begin
    new(pAux);
    pAux^.Elem:=num;
    pAux^.Siguiente:=nil;
    if (L.Primero=nil) then
      L.Primero:=pAux
    else
       L.Ultimo^.Siguiente:=pAux;
    L.Ultimo:=pAux;
    L.CantElementos:= C.CantElementos +1;
end;


Procedure IniciarRecorridoL(Var L:listaE);
Begin
L.Acctual := L.Primero;
end;


Procedure actualL(var L:ListaE; var num:Integer);
begin
    actual:= L.actual^.Elmen;
end;


Procedure  siguienteL(var L: listaE);
  begin
       L.Actual:=L.Actual^.Sigueinte;
  end;



Procedure  asignarL(var L1:listaE; L2:listaE);
Begin
          L1:=L2;
End;
```

```
Function existeL(L: listaE, num:integer): boolean;
Var aux: Lista;
Begin
        Aux:= L.primero;
        While (aux <> nil) and (aux^.elemen <> num) do begin
            Aux:= aux^.siguiente;
        End;
        existeL := (aux <> nil);
end;

Procedure eliminarL (var L: listaE, num:integer);
Var aux: lista;
     ant : lista;
Begin

   aux:= L.primero;
   While (aux <> nil) and (aux^.elemen <> num) do begin
        ant:= aux
        aux:= aux^.siguiente;
   End;

   If aux <> nil Then
     begin
          If (aux = L.primero) then
               Primero:= aux^.siguiente;
          Else
               If (aux = L.Ultimo) then
                    Ultimo := ant;
               else
                    ant^.Siguiente := aux^.Siguiente;
      end;
     dispose (aux);
end;
End:
```