

INTRODUCCIÓN Y CONCEPTOS FUNDAMENTALES

Tema 1

Tutor: Enrique Carmona
Asignatura: Estructuras de Datos y Algoritmos

INTRODUCCIÓN Y CONCEPTOS FUNDAMENTALES

Definiciones:

- **Tipo de Datos:** Conjunto de elementos.
- **Tipo de Datos Abstracto (TDA):** es un conjunto de elementos definidos de forma única mediante un tipo y un conjunto de operaciones sobre dichos elementos.
- **Estructura de Datos:** Implementación física de un TDA.

Análisis del coste de un TDA:

- **Almacenamiento:** tipo de datos y cantidad de ellos que maneja.
- **Coste de las operaciones básicas:** que se realizan sobre el conjunto de datos.

Pasos Básicos:

1. Análisis de Datos y operaciones.
2. Elección del TDA.
3. Elección de la implementación.

Tipos de Datos Abstractos Básicos

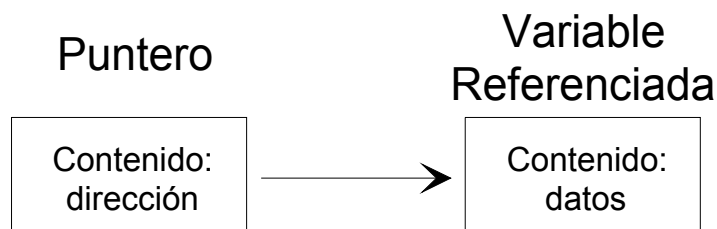
- **El TDA Entero:** conjunto de los números enteros definido por las matemáticas, $\{-1, -2, \dots, -\infty\} \cup \{0, 1, 2, \dots, \infty\}$, y como operaciones la suma, la resta, la multiplicación y la división.
- **El TDA Real:** conjunto de los números reales definido por las matemáticas y como operaciones la suma, la resta, la multiplicación y la división.
- **El TDA Booleano:** conjunto de valores booleanos, $\{\text{True}, \text{False}\}$, y como operaciones las definidas por el Algebra de Boole (AND, OR, NOT).
- **El TDA Carácter:** conjunto de caracteres definido por un alfabeto dado y como operaciones todos los operadores relacionales ($<$, $>$, $=$, \leq , \geq , \neq , $\langle \rangle$).
- **EL TDA Básico definido por el usuario**

Tipos de Datos Abstractos Compuestos Básicos

- **El TDA Conjunto:** colección de elementos sobre los que se definen las operaciones de unión, intersección y diferencia de conjuntos.
- **El TDA Arreglo:** colección homogénea de datos de longitud fija tal que cada una de sus componentes puede ser accedida individualmente mediante uno (arreglo unidimensional) o varios (arreglo multidimensional) índices, que serán de tipo ordinal, y que indican la posición de la componente dentro de la colección.
- **El TDA Registro:** es un tipo de datos heterogéneo compuesto por un número fijo de componentes denominados campos a los que se accede mediante un selector de campo (expresión formada por el nombre y un operador de selección, como por ejemplo: ‘.’)
- **EL TDA Compuesto definido por el usuario.** Ejemplo:
El TDA vector: aquel que tiene como tipo el conjunto de vectores definidos por las matemáticas, (X_1, X_2, \dots, X_n) , con $X_i \in C$, y sobre el que se define las operaciones, Obtener_elemento, Asignar_elemento, Sumar, Restar, Producto_vectorial, Producto_escalar y Transposición.

PUNTEROS

Un puntero es un tipo de datos simple cuyo valor es la *dirección* de una variable de otro tipo, denominada *variable referenciada*.



- Las variables referenciadas son variables dinámicas: se crean en tiempo de ejecución.
- La memoria requerida por las variables dinámicas se reserva (como memoria ocupada) durante la ejecución del programa y sólo cuando es necesaria.
- En Modula2, la declaración de un tipo puntero y de una variable puntero, se hace respectivamente:

```
TYPE Tipo_puntero = POINTER TO Tipo_referenciada  
  
VAR variable_puntero : Tipo_puntero
```

Manejo de punteros:

- La variable referenciada es accedida con el puntero mediante el operador apuntador. En Modula2: “^”.
- Hay que reservar la memoria necesaria para la variable referenciada por el puntero y asignar el valor del puntero a la dirección de memoria que ha reservado. En Modula2:

<code>Allocate(variable_puntero, SIZE(Tipo_referenciada))</code>

- Cuando la variable, apuntada por el puntero, deja de ser útil, es necesario liberar la memoria utilizada por ella. En Modula2:

<code>Deallocate(variable_puntero, SIZE(Tipo_referenciada))</code>

- Con los punteros se pueden realizar dos operaciones: la asignación y la comparación.

Ejemplo de operaciones con punteros:

Declaración de tipo puntero \longrightarrow TYPE
Tipo_puntero_a_caracter = POINTER TO CHAR

Declaración de variables puntero \longrightarrow VAR Ptr_car1, Ptr_car2 : Tipo_puntero_a_caracter

Ptr_car1 Ptr_car2
[?] [?]

Asignación de memoria \longrightarrow Allocate(Ptr_car1, SIZE(CHAR));
Allocate(Ptr_car2, SIZE(CHAR));

Comparación de punteros

Ptr_car1 = Ptr_car2

FALSO

Ptr_car1 \longrightarrow [?]
Ptr_car2 \longrightarrow [?]

Asignación de valores a las variables referenciadas \longrightarrow

Ptr_car1^ := 'C'; Ptr_car2^ := 'F';

Comparación de punteros

Ptr_car1 = Ptr_car2

FALSO

Ptr_car1 \longrightarrow [C]
Ptr_car2 \longrightarrow [F]

Asignación de valores entre variables referenciadas \longrightarrow

Ptr_car1^ := Ptr_car2^;

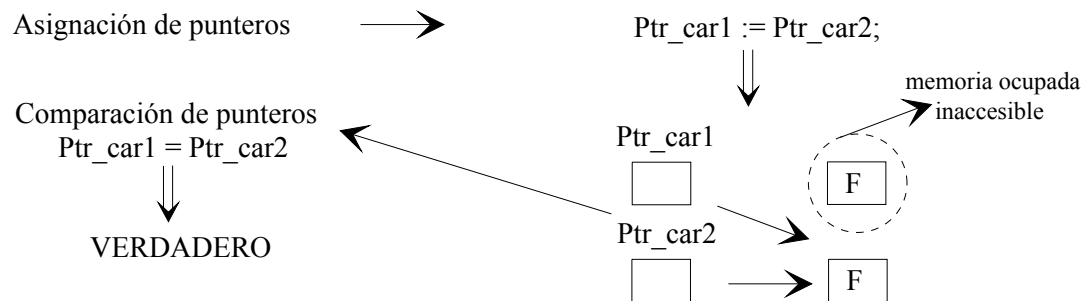
Comparación de punteros

Ptr_car1 = Ptr_car2

FALSO

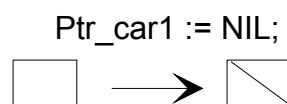
Ptr_car1 \longrightarrow [F]
Ptr_car2 \longrightarrow [F]

Pérdida de memoria por mala gestión de punteros:



Es importante señalar cuando un puntero no señala a ninguna variable referenciada. Modula2 dispone de una palabra clave, NIL, que representa una constante.

Esta asignación tiene que hacerla explícita el programador porque en el momento de declarar una variable de tipo puntero su valor no es NIL.



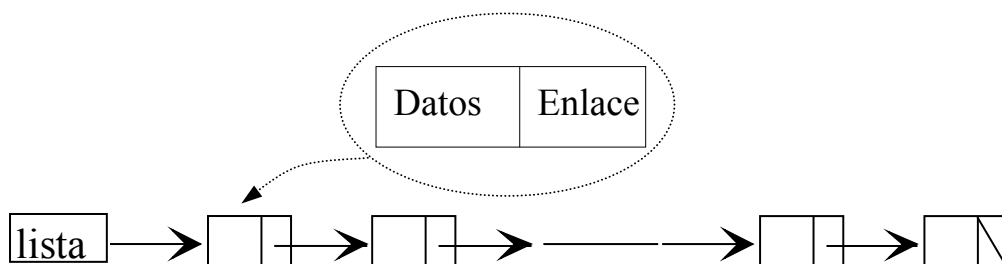
El TDA puntero: Es un tipo de datos simple cuyos valores son direcciones de memoria, y sus operadores asociados son la asignación y la comparación de punteros.

LISTAS ENLAZADAS

Las listas enlazadas son estructuras de datos dinámicas que se construyen con nodos.

Un nodo es un registro con dos campos:

- Datos (o data): contiene las componentes. Puede estar formado por cualquier tipo estructurado.
- Enlace (o next): puntero para señalar a otro nodo.



Ejemplo de lista enlazada

```

TYPE Ptr_Nodo = POINTER TO Nodo;
   Nodo = RECORD
       datos : Tipo_datos
       enlace : Ptr_Nodo;
   END;

VAR lista : Ptr_Nodo;

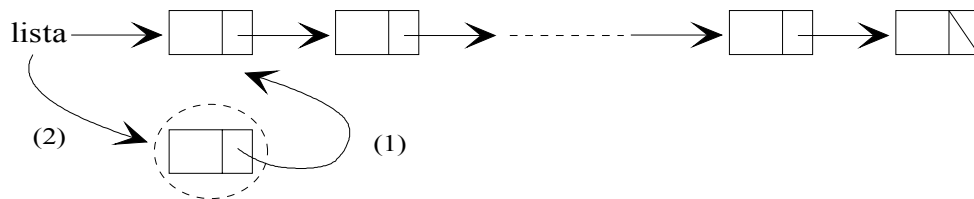
```

El TDA lista enlazada: es una colección de nodos ordenada según su posición, tal que cada uno de ellos es accedido mediante el campo enlace del nodo anterior.

IMPLEMENTACIÓN

DINÁMICA DE LISTAS ENLAZADAS:

Inserción por la cabeza (escribir en la lista por la cabeza):

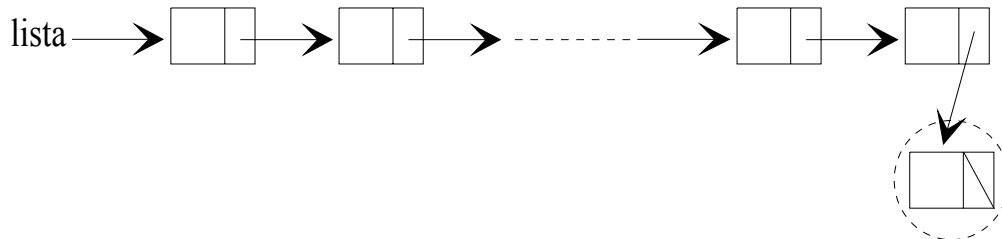


Nota: Es importante conservar las prioridades de asignación (1) y (2) para no perder información.

```
PROCEDURE Insertar_cabeza (VAR lista : Ptr_Nodo; nuevo_dato:
Tipo_datos);
VAR
  Nuevo_nodo : Ptr_Nodo; (*Puntero al nuevo nodo*)
BEGIN
  Allocate(Nuevo_nodo, SIZE(Nodo));
  Nuevo_nodo^.datos := nuevo_dato;
  Nuevo_nodo^.enlace := lista;
  lista := Nuevo_nodo
END InsertarCabeza;
```

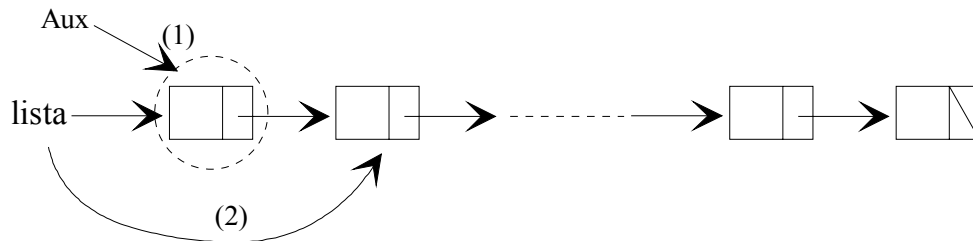
Insertar por el final (escribir en la lista por el final):

Para insertar por el final primero hay que llegar hasta él y luego hacer la inserción.



```
PROCEDURE Insertar_final (VAR lista : Ptr_Nodo; nuevo_dato: Tipo_datos);
VAR
    Nuevo_nodo, Actual : Ptr_Nodo;
BEGIN
    IF lista=nil THEN Insertar_cabeza(lista, nuevo_dato) ELSE
        Allocate(Nuevo_nodo, SIZE(Nodo)); (*Solicitar memoria para el nodo*)
        Nuevo_nodo^.datos := nuevo_dato; (*asignar valores al nodo*)
        Nuevo_nodo^.enlace := NIL; (*Por ser el último nodo*)
        Actual := lista;
        WHILE Actual^.enlace <> NIL DO (*Recorrer la lista (se supone que la lista no está vacía)*)
            Actual := Actual^.enlace
        END;
        Actual^.enlace := Nuevo_nodo
    END
END Insertar_final;
```

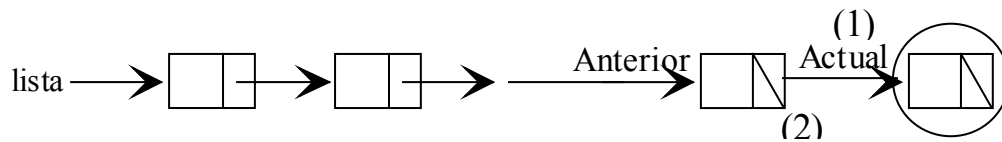
Suprimir por la cabeza (leer de la lista por la cabeza):



```
PROCEDURE Suprimir_cabeza (VAR lista : Ptr_Nodo; VAR dato:
Tipo_datos);
VAR
    Aux : Ptr_Nodo;
BEGIN
    Aux := lista; (* necesario para liberar la memoria del nodo leído*)
    dato := lista^.datos;
    lista := lista^.enlace;
    Deallocate(Aux, SIZE(Nodo))
END Suprimir_cabeza;
```

Nota: Esta función presupone que existe al menos un elemento en la lista.

Suprimir por el final (leer de la lista por el final):



```
PROCEDURE Suprimir_final (VAR lista : Ptr_Nodo; VAR dato:
Tipo_datos);
VAR
  Anterior, Actual : Ptr_Nodo;
BEGIN
  Anterior:= lista;
  Actual:=lista;
  WHILE Actual ^.enlace <> NIL DO (*Recorre la lista*)
    Anterior:= Actual;
    Actual:= Actual ^.enlace
  END
  dato := Actual ^.datos;
  Anterior ^.enlace:=NIL;
  IF Anterior=Actual THEN (* si sólo hay un nodo *)
    Lista:=NIL
  END
  Deallocate(Actual, SIZE(Nodo))
END Suprimir_final;
```

Nota: Esta función presupone que existe al menos un elemento en la lista.

Inserción según un criterio de orden

Suponer que la lista se utiliza para almacenar números enteros (Tipo_datos = INTEGER) y que la lista está ordenada en orden creciente.

Inserción según el criterio: “delante del actual”

```
PROCEDURE Insertar_orden_delante (VAR lista: Ptr_Nodo; dato :
Tipo_datos);
VAR
  Encontrado: BOOLEAN;
  Actual,Nuevo_nodo,Anterior: Ptr_Nodo;
BEGIN
  IF (lista=NIL)OR(dato<lista^.datos) THEN
    (* Insercion al principio *)
    Insertar_cabeza(lista,dato)
  ELSE
    ALLOCATE(Nuevo_nodo,SIZE(Nodo));
    Nuevo_nodo^.datos:=dato;
    (*Busca el punto de insercion*)
    Anterior:=lista;
    Actual := lista^.enlace;
    Encontrado := FALSE;
    WHILE (Actual<>NIL)AND(NOT Encontrado) DO
      IF dato>Actual^.datos THEN
        Anterior:=Actual;
        Actual:=Actual^.enlace
      ELSE
        Encontrado:=TRUE;
      END;
    END;
    (*Insertar el nuevo nodo*)
    Nuevo_nodo^.enlace := Actual;
    Anterior^.enlace := Nuevo_nodo
  END
END Insertar_orden_delante;
```

Insertión según el criterio: “detrás del actual”

```
PROCEDURE Insertar_orden_detras (VAR lista: Ptr_Nodo; dato :
Tipo_datos);
VAR
    Encontrado: BOOLEAN;
    Actual, Nuevo_nodo, Anterior: Ptr_Nodo;
BEGIN
    IF (lista=NIL)OR(dato<lista^.datos) THEN
        (* Insercion al principio *)
        Insertar_cabeza(lista,dato)
    ELSE
        ALLOCATE(Nuevo_nodo,SIZE(Nodo));
        Anterior:=lista;
        Actual := lista^.enlace;
        Encontrado := FALSE;
        WHILE (Actual<>NIL)AND(NOT Encontrado) DO      (*Ins. en
medio*)
            IF dato>Actual^.datos THEN
                Anterior:=Actual;
                Actual:=Actual^.enlace
            ELSE
                Nuevo_nodo^:=Actual^;
                Actual^.enlace:=Nuevo_nodo;
                Actual^.datos:=dato;
                Encontrado:=TRUE;
            END;
        END;
        IF NOT Encontrado THEN  (*Insercion al final*)
            Nuevo_nodo^.datos:=dato;
            Nuevo_nodo^.enlace:=NIL;
            Anterior^.enlace:=Nuevo_nodo
        END
    END
END Insertar_orden_detras;
```

Imprimir una lista enlazada

Imprimir en orden:

```
PROCEDURE Imprimir_lista (lista: Ptr_Nodo);  
VAR  
  Aux : Ptr_Nodo;  
BEGIN  
  Aux := lista;  
  WHILE Aux <> NIL DO  
    WriteInt(Aux^.datos,7);  
    Aux := Aux^.enlace;  
  END;  
END Imprimir_lista;
```

Imprimir en orden inverso:

```
PROCEDURE Imprimir_contrario (lista: Ptr_Nodo);  
BEGIN  
  IF lista <> NIL THEN  
    Imprimir_contrario(lista^.enlace);  
    WriteInt(lista^.datos,6);  
  END;  
END Imprimir_contrario;
```


LISTAS

El TDA lista (o Secuencia): es una colección homogénea de datos, ordenados según su posición en ella, tal que cada elemento tiene un predecesor (excepto el primero) y un sucesor (excepto el último), y los operadores asociados sobre ellas son:

- **Insertar:** inserta un elemento, x , en una posición, p , de la lista, pasando los elementos de la posición p y siguientes a la posición inmediatamente posterior.
- **Localizar:** localiza la posición p en la que se encuentra un elemento dado, x .
- **Recuperar:** encuentra el elemento x que esta en la posición p .
- **Suprimir:** elimina de la lista el elemento de la posición p .
- **Suprimir_dato:** elimina de la lista todas las apariciones del elemento x .
- **Anula:** ocasiona que la lista se vacíe.
- **Primero (Fin):** proporciona el primer (último) elemento de la lista.
- **Imprimir:** imprime todos los elementos de la lista en su orden.

IMPLEMENTACIÓN DINÁMICA DE LISTAS (SEGÚN EL CONCEPTO DE IMPLEMENTACIÓN DINÁMICA DE LISTAS ENLAZADAS)

Insertar:

```
PROCEDURE Insertar (VAR L: Ptr_Nodo; x : Tipo_datos;p: INTEGER);
(*inserta un elemento, x, en una posicion p de L, pasando los elementos
de la posicion p y siguientes a la posicion inmediatamente posterior*)
VAR Anterior, Actual,Nuevo_nodo: Ptr_Nodo;
    cont: INTEGER;
BEGIN
    IF p=1 THEN (*Inserta en la primera posicion*)
        ALLOCATE(Nuevo_nodo,SIZE(Nodo));
        Nuevo_nodo^.datos:=x;
        Nuevo_nodo^.enlace:=L;
        L:=Nuevo_nodo
    ELSE (*localiza la posicion p de insercion*)
        Anterior:=NIL;
        Actual:=L;
        cont:=1;
        WHILE (Actual#NIL)&(cont#p) DO
            Anterior:=Actual;
            Actual:=Actual^.enlace;
            cont:=cont+1
        END;
        IF p<cont+1 THEN(*  $p \leq [(numero\ de\ elementos\ de\ L)+1]$  *)
            ALLOCATE(Nuevo_nodo,SIZE(Nodo));
            Nuevo_nodo^.datos:=x;
            Anterior^.enlace:=Nuevo_nodo;
            Nuevo_nodo^.enlace:=Actual; (* Hay una "pseudoerrata" en el libro de texto *)
        END
    END
END Insertar;
```

Localizar:

```
PROCEDURE Localizar (L: Ptr_Nodo; x : Tipo_datos):INTEGER;
(*localiza la posicion en la que se encuentra un elemento dado x,
si no lo encuentra devuelve 0*)
VAR
  Actual,Anterior: Ptr_Nodo;
  Encontrado: BOOLEAN;
  p: INTEGER;
BEGIN
  Anterior:=NIL;
  Actual:=L;
  p:=0;

  Encontrado:=FALSE;
  WHILE NOT Encontrado & (Actual#NIL) DO
    IF Actual^.datos=x THEN
      Encontrado:=TRUE;
      p:=p+1
    ELSE
      Anterior:=Actual;
      Actual:=Actual^.enlace;
      p:=p+1;
    END;
  END;
  IF NOT Encontrado THEN
    (*Elemento no encontrado*)
    p:=0;
  ELSE
    Actual:=Actual^.enlace;
  END;
  RETURN p
END Localizar;
```

Recuperar:

```
PROCEDURE Recuperar (L: Ptr_Nodo; VAR x : Tipo_datos;p: INTEGER;
VAR encontrado:BOOLEAN);
(*encuentra el elemento x que esta en la posicion p, si la posicion p es
mayor que el numero de elementos de L, devuelve encontrado a FALSE*)
VAR
    Actual: Ptr_Nodo;
    cont:INTEGER;
BEGIN
    encontrado:=FALSE;
    Actual:=L;
    cont:=1;
    WHILE (Actual#NIL)&(cont<p) DO
        Actual:=Actual^.enlace;
        cont:=cont+1
    END;
    IF (cont=p)&(Actual#NIL) THEN
        x:=Actual^.datos;
        encontrado:=TRUE;
    END
END Recuperar;
```

Suprimir:

```
PROCEDURE Suprimir (VAR L: Ptr_Nodo; p: INTEGER);
(*elimina de L el elemento de la posicion p*)
VAR
  Anterior,Actual: Ptr_Nodo;
  cont:INTEGER;

BEGIN
  Anterior:=NIL;
  Actual:=L;
  cont:=1;
  (*Localizar la posicion p*)
  WHILE (Actual^.enlace#NIL)&(cont<p) DO
    Anterior:=Actual;
    Actual:=Actual^.enlace;
    cont:=cont+1
  END;
  IF cont=p THEN
    (*Eliminar*)
    IF cont=1 THEN (*eliminar el primer elemento*)
      L:=Actual^.enlace
    ELSIF Actual^.enlace=NIL THEN (*eliminar el ultimo*)
      Anterior^.enlace:=NIL
    ELSE (*intermedio*)
      Anterior^.enlace:= Actual^.enlace;
    END;
    DEALLOCATE(Actual,SIZE(Nodo))
  END
END Suprimir;
```

Suprimir_dato:

```
PROCEDURE Suprimir_dato (VAR L: Ptr_Nodo; x: Tipo_datos);
(*elimina de L todas las apariciones del elemento x*)
VAR
  Anterior,Actual,Aux: Ptr_Nodo;
  Encontrado: BOOLEAN;
BEGIN
  Anterior:=NIL;
  Actual:=L;
  WHILE Actual#NIL DO (*busqueda de todas las apariciones*)
    Encontrado:=FALSE;
    WHILE NOT Encontrado & (Actual#NIL) DO
      IF Actual^.datos=x THEN
        Encontrado:=TRUE
      ELSE
        Anterior:=Actual;
        Actual:=Actual^.enlace;
      END;
    END;
  END;
  IF Encontrado THEN (*Eliminar*)
    IF Actual=L THEN (*eliminar el primer elemento*)
      L:=Actual^.enlace;
      Aux:=L;
    ELSIF (Actual^.enlace=NIL)&(Anterior#NIL) THEN (*eliminar el
ultimo*)
      Anterior^.enlace:=NIL;
      Aux:=NIL;
    ELSE (*intermedio*)
      Anterior^.enlace:= Actual^.enlace;
      Aux:=Actual^.enlace;
    END;
  END;
  DEALLOCATE(Actual,SIZE(Nodo));
  Actual:=Aux;
END;
END Suprimir_dato;
```

Anula:

```
PROCEDURE Anula (VAR L: Ptr_Nodo);
(*ocasiona que L se vacie*)
VAR
    Actual: Ptr_Nodo;
BEGIN
    IF L#NIL THEN
        Actual:=L;
        WHILE Actual^.enlace#NIL DO
            L:=Actual^.enlace;
            DEALLOCATE(Actual,SIZE(Nodo));
            Actual:=L;
        END;
        DEALLOCATE(L,SIZE(Nodo));
        L:=NIL;
    END;
END Anula;
```

Primero:

```
PROCEDURE Primero (L: Ptr_Nodo): Tipo_datos;
(*proporciona el primer elemento de L*)
BEGIN
    IF L#NIL THEN
        RETURN L^.datos
    END;
END Primero;
```

Fin:

```
PROCEDURE Fin (L: Ptr_Nodo): Tipo_datos;  
(*proporciona el ultimo elemento de L*)  
VAR  
    Anterior, Actual: Ptr_Nodo;  
BEGIN  
    Anterior:=NIL;  
    Actual:=L;  
    WHILE Actual#NIL DO  
        Anterior:=Actual;  
        Actual:=Anterior^.enlace  
    END;  
    IF Anterior#NIL THEN  
        RETURN Anterior^.datos  
    END;  
END Fin;
```

Imprimir:

```
PROCEDURE Imprimir (L: Ptr_Nodo);  
(*imprime todos los elementos de la lista en su orden*)  
VAR  
    Actual : Ptr_Nodo;  
BEGIN  
    Actual := L;  
    WHILE Actual # NIL DO  
        WriteInt(Actual^.datos,6);  
        Actual := Actual^.enlace;  
    END;  
END Imprimir;
```