

## Resumen CADP (Hecho por Franja Morada)

En Pascal, existen **dos** formas principales de pasar argumentos a procedimientos y funciones: **por valor y por referencia**.

### Parámetros por valor:

Se crea una copia del valor del argumento original.

Cualquier modificación realizada dentro del procedimiento o función se realiza en la copia, no en el valor original.

Se utiliza cuando no se desea que el procedimiento o función modifique el valor original de la variable.

### Parámetros por referencia:

Se pasa la dirección de memoria de la variable original al procedimiento o función.

Cualquier modificación realizada dentro del procedimiento o función se realiza directamente en el valor original de la variable.

Se utiliza cuando se desea que el procedimiento o función modifique el valor original de la variable. En Pascal, los módulos, conocidos como subprogramas, se dividen en dos categorías: **procedimientos y funciones**. Ambos tipos de módulos permiten agrupar y reutilizar código para mejorar la organización y legibilidad de los programas, pero se diferencian:

### Procedimientos:

**Proceso:** devuelven desde 0 a más valores

**Objetivo:** ejecutan instrucciones específicas

**Estructura:** se define usando la palabra "procedure", seguida del nombre del procedimiento y una lista de parámetros entre paréntesis (opcional). Dentro del bloque se declaran las variables local (si se requiere) y se escriben las instrucciones que se ejecutan al llamar al procedimiento.

**Ejemplo:** un procedimiento para calcular el área de un rectángulo: e imprimir el resultado:

```
procedure calcularAreaRectangulo(base: real; altura: real);
```

```
var
```

```
    area: real;
```

```
begin
```

```
    area := base * altura;
```

```
    writeln('El area del rectángulo es: ', area);
```

**end;**

**Invocación/Llamada:** se invocan desde otro lugar utilizando su nombre seguido de la lista de argumentos entre paréntesis en el mismo orden que se creó. Los argumentos pueden ser pasados por valor o por referencia para proporcionar valores a las variables del procedimiento según se desee.

Seguido el **ejemplo** anterior:

**calcularAreaRectangulo(base,altura);**

**Funciones:**

**Función:** devuelve un valor único (tiene que ser tipo de dato Ordinal/Simple).

**Objetivo:** ejecutan instrucciones específicas y devuelven el resultado a la unidad que la llamó.

**Estructura:** se define usando la palabra "function", seguida del nombre de la función, una lista de parámetros entre paréntesis y un tipo de dato que retorna. Dentro del bloque se declaran las variables locales y se escriben las instrucciones que llevan a calcular un valor de retorno.

**Ejemplo:** Una función para calcular el cuadrado de un número (num<sup>2</sup>).

**function cuadrado(numero: real): real;**

**begin**

    cuadrado := numero \* numero;

**end;**

**Invocación/ Llamada:** se invocan desde otro lugar utilizando su nombre seguido de la lista de argumentos entre paréntesis. Los argumentos pueden ser pasados por valor o por referencia para proporcionar valores a las variables de la función. El valor de retorno se almacena en una variable o se utiliza en una expresión.

Seguido el **ejemplo** anterior, hay 2 opciones a elegir para este caso:

1) **Imprimir el valor:**

    writeln('El cuadrado del número ', num, ' es: ', cuadrado(num) );

2) **Utilizar una variable auxiliar** para almacenar el valor (debe ser del mismo tipo que devuelve la función):

    aux:= cuadrado(num);

Este módulo es para leer un tipo de **dato entero**, puede usarse con cualquier tipo de **dato simple**:

0	<b>procedure leerNum</b> (var unNum: integer);	/	<i>unNum es el parámetro donde se almacena el dato</i>
1		/	
0	<b>begin</b>		
2			
0	write('Número: ');	/	Escribe en pantalla el mensaje entre comillas
3		/	
0	readln(unNum);	/	Almacena en el parámetro unNum el valor ingresado
4		/	
0	<b>end;</b>		
5			

También puede aplicarse a tipos de **datos compuestos** como **registros**:

0	<b>procedure leerAuto</b> (var au: autos);	/	<i>au es el parámetro donde se almacena el dato</i>
1		/	
0	<b>begin</b>		
2			
0	write('Marca de auto: ');	/	Escribe en pantalla el mensaje entre comillas
3		/	
0	readln(au.marca);	/	Almacena en el parámetro au, campo marca, el dato
4		/	ingresado
0	write('Patente de auto: ');	/	Escribe en pantalla el mensaje
5		/	
0	readln(au.patente);	/	Almacena en el parámetro au, campo patente, el
6		/	dato ingresado
0	write('Titular: ');	/	Escribe en pantalla el mensaje
7		/	
0	readln(au.titular);	/	Almacena en el parámetro au, campo titular, el dato
8		/	ingresado
0	<b>end;</b>		
9			

Al usar registros, suele haber **cortes de control**. Por ejemplo: si se tiene que leer autos hasta que se ingresa la marca "ZZZ" que no debe procesarse, debemos preguntar primero por el campo de registro marca (au.marca) para no procesar los demás datos del auto en el módulo . En caso de que sea distinto del corte ("ZZZ") ingresamos los siguientes campos que le corresponden al auto.

**Recordar** que esto se hace para hacer más eficiente el programa y posteriormente en la carga de cualquier estructura (lista o vector de registros), a cargar, debe considerarse nuevamente el corte de control.

Considerando lo anterior, el módulo quedaría:

0	<b>procedure leerAuto</b> (var au: autos);	/	au es el parámetro donde se almacenacarga el dato
1		/	
0	<b>begin</b>		
2			
0	write('Marca de auto: ');	/	Escribe en pantalla el mensaje entre comillas
3		/	
0	readln(au.marca);	/	Almacena en el parámetro au, campo marca, el dato
4		/	ingresado
0	if ( au.marca <> 'ZZZ')then begin	/	Consulta si el dato ingresado es diferente al
5		/	corte de control
0	write('Patente de auto: ');	/	Escribe en pantalla el mensaje
6		/	
0	readln(au.patente);	/	Almacena en el parámetro au, campo patente, el
7		/	dato ingresado
0	write('Titular: ');	/	Escribe en pantalla el mensaje
8		/	
0	readln(au.titular);	/	Almacena en el parámetro au, campo titular, el dato
9		/	ingresado
1	end;	/	Finaliza el IF
0		/	
1	<b>end;</b>	/	Finaliza el proceso
1		/	

Teniendo datos cargados podemos implementar **comparaciones** cómo: calcular máximos, mínimos o modificar contadores. Estas variables auxiliares (max, min, cant) deben inicializarse antes de empezar a llamar los procesos:

0	<b>procedure encontrarMaximo</b> (unNum:	/	unNum es el número actual que se utilizará para
1	integer; unaMarca: string; var max:	/	comparar con el máximo, unaMarca es la marca
	integer; var marcaMax: string);		actual. max y marcaMax son los valores más grande
			almacenados.
0	<b>begin</b>		
2			

0	if(unNum > max)then begin	/	Comprueba si el valor actual es mayor al maximo
3		/	actual
0	max:= unNum;	/	Si lo es, actualiza valor del máximo
4		/	
0	marcaMax:= unaMarca;	/	Actualiza marca del valor máximo
5		/	
0	end;		
6			

También aplicable a **mínimos**:

0	<b>procedure encontrarMinimo</b> (unNum:	/	<i>unNum es el número actual que se utilizará para</i>
1	<i>integer; unaMarca: string; var min: integer;</i>	/	<i>comparar con el minimo, unaMarca es la marca</i>
	<i>var marcaMin: string);</i>		<i>actual. min y marcaMin son los valores más chicos</i>
			<i>almacenados.</i>
0	<b>begin</b>		
2			
0	if(unNum < min)then begin	/	Comprueba si el valor actual es menor al mínimo
3		/	actual
0	min:= unNum;	/	Si lo es, actualiza valor del mínimo
4		/	
0	marcaMin:= unaMarca;	/	Actualiza marca del valor mínimo
5		/	
0	end;		
6			

**Varios ejercicios piden encontrar dos de los anteriores:**

0	<b>procedure encontrarMaximos</b> (unNum:	/	<i>unNum es el número actual que se utilizará para</i>
1	<i>integer; unaMarca: string; var max, max2:</i>	/	<i>comparar con el máximo, unaMarca es la marca</i>
	<i>integer; var marcaMax, marcaMax2:</i>		<i>actual. max y marcaMax son los valores más grande</i>
	<i>string);</i>		<i>almacenados. max2 y marcaMax2 son los segundos</i>
			<i>valores mas grandes almacenados.</i>
0	<b>begin</b>		
2			
0	if(unNum > max)then begin	/	Comprueba si el valor actual es mayor al maximo mas
3		/	grande
0	max2:= max;	/	Si lo es, actualiza, maximo2 toma valor del mas
4		/	grande anterior
0	marcaMax2:= marcaMax;	/	Actualiza, marcaMax2 toma marca del mas grande
5		/	anterior

0 6	max:= unNum;	/	Actualiza valor, max toma el valor actual
0 7	marcaMax:= unaMarca;	/	Actualiza marca, marcaMax toma la marca mas grande actual
0 8	<b>end else</b>	/	Si el actual no es mas grande que el max mas grande
0 9	if(unNum > max2)then begin	/	Verifico si es mas grande que el segundo mas grande
1 0	max2:= unNum;	/	Si lo es, max2 toma valor actual
1 1	marcaMax2:= unaMarca;	/	marcaMax2 toma marca del segundo valor mas grande
1 2	end;		
1 3	end;		

0 1	<b>procedure encontrarMinimos</b> (unNum: integer; unaMarca: string; var min, min2: integer; var marcaMin, marcaMin2: string);	/	<i>unNum es el número actual que se utilizará para comparar con el minimo, unaMarca es la marca actual. min y marcaMin son los valores más chicos almacenados. min2 y marcaMin2 son los segundo valores más chicos almacenados.</i>
0 2	<b>begin</b>		
0 3	if(unNum < min)then begin	/	Comprueba si el valor actual es menor al minimo mas chico
0 4	min2:= min;	/	Si lo es, actualiza, min2 toma valor del mas chico anterior
0 5	marcaMin2:= marcaMin;	/	Actualiza, marcaMin2 toma marca del mas chico anterior
0 6	min:= unNum;	/	Actualiza valor, min toma el valor actual
0 7	marcaMin:= unaMarca;	/	Actualiza marca, marcaMin toma la marca mas chica actual
0 8	<b>end else</b>	/	Si el actual no es mas chico que el minimo mas chico
0 9	if(unNum < min2)then begin	/	Verifico si es mas chico que el segundo mas chico
1 0	min2:= unNum;	/	Si lo es, min2 toma valor actual

1 1	marcaMin2:= unaMarca;	/	marcaMin2 toma marca del segundo valor mas chico
1 2	end;		
1 3	end;		

Modificar contadores:

0 1	<b>procedure verificarSiEs4</b> (unNum: integer;var cont4s: integer);	/	<i>unNum es el número actual que se verifica, cont4s será el parametro que contara si el número cumple la condicion</i>
0 2	<b>begin</b>		
0 3	if(unNum = 4)then	/	Comprueba si el valor actual es igual a 4
0 4	cont4s:= cont4s +1;	/	Si lo es, suma al valor actual de cont4s, 1.
0 5	<b>end;</b>		

Además de estas validaciones hay funciones que suelen utilizarse para **cálculos, de sectores geométricos, promedios, porcentaje, incluso también verificaciones:**

0 1	<b>function</b> <b>calcularPerimetroCuadrado</b> (unaBase, unaAltura: integer): integer;	/	<i>unaBase seria valor de la base actual, unaAltura valor de la altura actual. Devuelve un valor integer.</i>
0 2	<b>begin</b>		
0 3	calcularPerimetroCuadrado:= (unaBase * unaAltura) * 2;	/	Asigna valor de retorno el cálculo de la base x altura x 2.
0 4	<b>end;</b>		

0 1	<b>function calcularPromedio</b> (cantTotal, cantCumple: integer): real;	/	<i>cantTotal seria valor de elementos en total, cantCumple valor de elementos que cumplan un criterio. Devuelve un valor real.</i>
0 2	<b>begin</b>		

0	calcularPromedio:= cantCumple /		/	Asigna valor de retorno el cálculo de los que cumplen
3	cantTotal;		/	dividido el total.
0	<b>end;</b>			
4				

0	<b>function esMultiploDe5</b> (unNum: integer):		/	<i>unNum es un valor que se verificará. Devuelve un</i>
1	boolean;		/	<i>valor bool</i>
0	<b>begin</b>			
2				
0	if(unNum MOD 5 = 0)then		/	Verifica si el valor actual dividido 5 deja resto 0
3			/	
0	esMultiploDe5:= true			Si lo hace, retorna true
4				
0	else			
5				
0	esMultiploDe5:= false;			Sino, retorna false
6				
0	<b>end;</b>			
7				

La misma función puede escribirse más directa:

0	<b>function esMultiploDe5</b> (unNum: integer):		/	<i>unNum es un valor que se verificará. Devuelve un</i>
1	boolean;		/	<i>valor bool</i>
0	<b>begin</b>			
2				
0	esMultiploDe5:= unNum MOD 5 = 0;		/	Retorna true/false si esa comparación se cumple o
3			/	no.
0	<b>end;</b>			
7				

**Recordar** que las funciones solo devuelven tipos de datos ordinales, si quisiera devolver un registro/ vector/ lista o cualquier tipo de estructura compuesta no se podrá, en ese caso debe utilizarse un procedimiento.

El siguiente módulo carga un **vector completo** hasta que el índice llegue al valor de su límite (dimensión física):

0	<b>procedure cargarVector</b> (var v: vector);		/	<i>v es un vector declarado</i>
1			/	



0 2	<b>var</b>			
0 3	dato, i: integer;	/	/	Declaro i como índice y dato para ingresar valores
0 4	<b>begin</b>			
0 5	for i:= 1 to dimF do begin	/	/	Repito dimF veces (dimF es cantidad max del vector)
0 6	leerNum(dato);	/	/	Utilizo leerNum para pedir e ingresar un valor en dato
0 7	v[i]:= dato;	/	/	Almaceno el dato en el vector en la posición i
0 8	<b>end;</b>			

**El siguiente módulo carga un vector parcialmente (hasta que corte al ingresar un dato específico):**

0 1	<b>procedure cargarVector</b> (var v: vector; var dimL: integer);	/	/	<i>v es un vector declarado. dimL es la cantidad de datos cargados en el vector</i>
0 2	<b>var</b>			
0 3	dato: integer;	/	/	Declaro dato para ingresar valores
0 4	<b>begin</b>			
0 5	dimL:= 0;	/	/	Inicializo dim lógica en 0, puede hacerse fuera
0 6	leerNum(dato);	/	/	Utilizo leerNum para pedir e ingresar un valor en dato
0 7	while(dato <> -1)AND(dimL < dimF)do begin	/	/	Mientras el valor ingresado no sea -1 y la dimL no llegue a dimF
0 8	dimL:= dimL +1;	/	/	Incrementa dim logica
0 9	v[dimL]:= dato;	/	/	Almacena en vector, en posición valor de dim lógica el dato
1 0	if(dimL < dimF)then	/	/	Verifica si dim lógica sigue siendo menor a la física
1 1	leerNum(dato);	/	/	Si lo es, ingresó un nuevo valor.
1 2	<b>end;</b>			

1 3	<b>end;</b>			
--------	-------------	--	--	--

El siguiente proceso imprime un **vector de registros**, el cual, podría adaptarse a cualquier tipo de dato que almacena el vector. En este caso, imprime el vector completo:

0 1	<b>procedure</b> imprimirVector(v: vector);	/	<i>v es un vector declarado y cargado</i>
1 2	<b>var</b>	/	
0 3	i: integer;	/	Declaro i como índice
0 4	<b>begin</b>	/	
0 5	for i:= 1 to dimF do begin	/	Repito dimF veces (dimF es la cantidad física del vector)
0 6	write('Marca: ',v[i].marca,' patente: ',v[i].patente,' Nombre de titular: ',v[i].titular);	/	Imprime el mensaje concatenado con valores almacenados en el registro dentro del vector en la posición i actual
0 7	end;		
0 8	<b>end;</b>		

En caso de que el vector **no está cargado completamente**, necesitaremos utilizar el valor de su **dimensión lógica**:

0 1	<b>procedure</b> imprimirVector(v: vector; dimL: integer);	/	<i>v es un vector declarado y cargado. dimL la cantidad de elementos cargados</i>
0 2	<b>var</b>		
0 3	i: integer;	/	Declaro i como índice
0 4	<b>begin</b>	/	
0 5	for i:= 1 to dimL do begin	/	Repito dimL veces (dimL es cantidad de elementos del vector)

0	write('Marca: 'v[i].marca,' patente:	/	Imprime el mensaje concatenado con valores
6	'v[i].patente,' Nombre de titular:	/	almacenados en el registro dentro del vector en la
	'v[i].titular);		posición i actual
0	end;		
7			
0	<b>end;</b>		
8			

Además de los procesos anteriores existen métodos para **ordenar** el vector por cierto criterio. El siguiente es el **método por Selección**, que recorre el vector completo (y en cada ejecución verifica desde su siguiente posición hasta el final hasta encontrar donde debería quedar según el ordenamiento).

En este caso es ordenamiento de forma **ascendente** (de menor a mayor):

0	<b>procedure ordenarPorSeleccion</b> (var v:	/	<i>v es un vector declarado y cargado</i>
1	vector);	/	
0	<b>var</b>		
2			
0	i, j, pos: integer;	/	Declaro i, j como índices pos para auxiliar
3		/	
0	aux: elementoVector;	/	Declaro aux para guardar valores momentaneamente
4		/	
0	<b>begin</b>		
5			
0	for i:= 1 to dimF-1 do begin	/	Repito dimF-1 veces (dimF es cantidad max del
6		/	vector)
0	pos:= i;	/	Pos guarda posición actual del vector
7		/	
0	for j:= i+1 to dimF do	/	Recorre desde la siguiente posición (de la pos
8		/	actual) hasta el final del vector
0	if(v[j] < v[pos]) then	/	Verificando si la actual(j) es menor a la actual
9		/	original(i)
1	pos:= j;	/	Si lo es, guarda en pos el índice del elemento más
0		/	chico
1	aux:= v[pos];	/	Terminado el for, aux guarda el valor del valor mas
1		/	chico del vector
7			
1	v[pos]:= v[i];	/	En el lugar de pos, intercambia valor con i (elemento
2		/	original)
1	v[i]:= aux;	/	En el lugar de i, intercambio valor con pos (valor mas
3		/	chico)

1 4	end;			
1 5	end;			

## Listas:

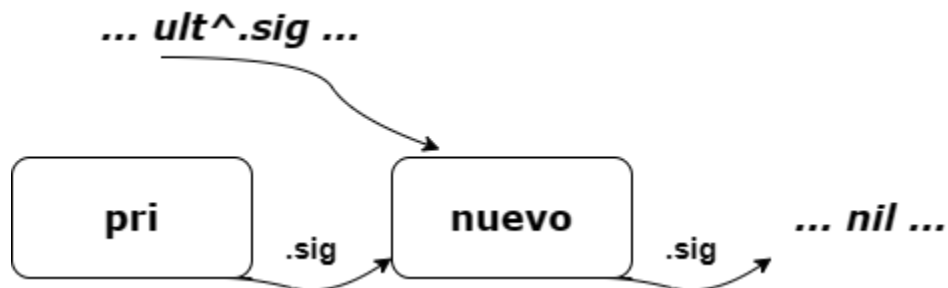
Formas de **agregar dato** a una lista:

0 1	<b>procedure</b> insertarAdelante( <i>var pri: lista;</i> <i>num: integer;</i>	/ /	<i>pri es la lista principal y num el dato a cargar en el nodo</i>
0 2	<b>var</b> nuevo: lista	/ /	Creo un nodo auxiliar para agregar a la lista principal
0 3	<b>begin</b>		
0 4	new(nuevo);	/ /	Creo el nodo nuevo
0 5	nuevo^.dato:= num;	/ /	El dato / elemento del nodo tomará el valor del dato que ingresa
0 6	nuevo^.sig:= pri;	/ /	El nodo apunta a la lista principal (se posiciona a la izquierda)
0 7	pri:= nuevo;	/ /	La lista principal a hora se pisa con el nuevo nodo para agregar este mismo a la lista
0 8	<b>end;</b>		

- Si la lista principal ingresa vacía, el primer nodo se posiciona y queda apuntando a vacío (se convierte en el primer nodo de la lista)



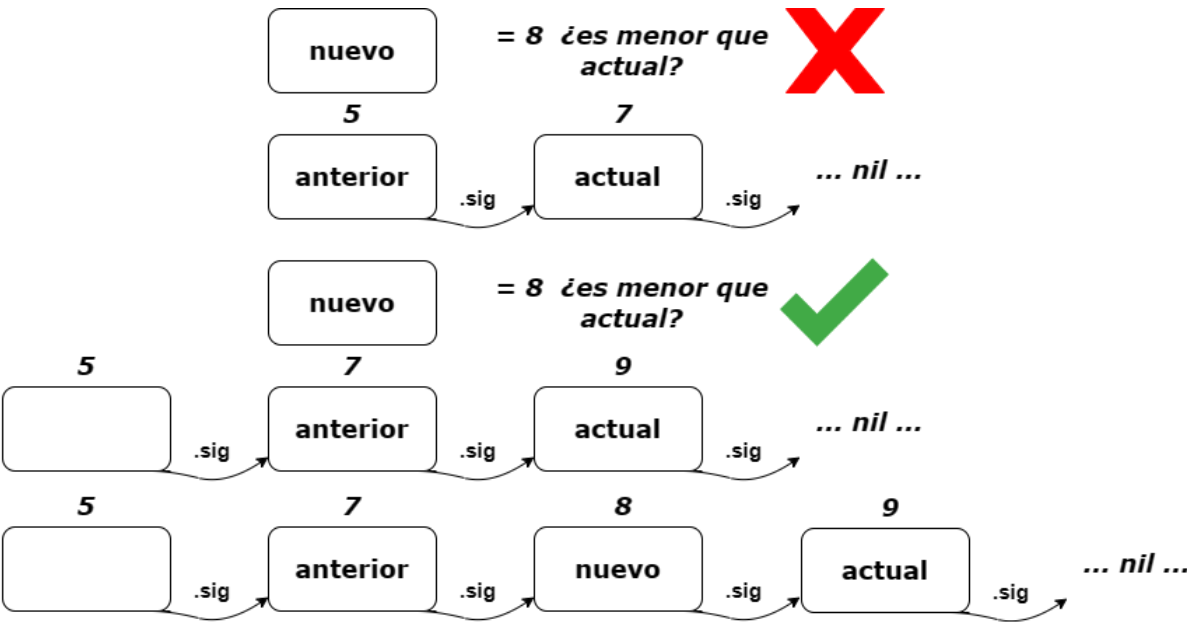
0	<b>procedure insertarAtras</b> (var pri,	/	<i>pri es la lista principal, ult es el punter a nuestro último</i>
1	ult: lista; num: integer);	/	<i>nodo de la lista y num el dato a cargar en el nodo</i>
0	<b>var</b>		
2			
0	nuevo: lista ;	/	Creo un nodo auxiliar para agregar a la lista principal
3		/	
0	<b>begin</b>		
4			
0	new(nuevo);	/	Creo el nodo nuevo (actualmente nil)
5		/	
0	nuevo^.dato:= num;	/	El dato / elemento del nodo tomará el valor del dato que ingresa
6		/	
0	nuevo^.sig:= nil;	/	El nodo apunta a vacío (se convierte en el último al final de la lista)
7		/	
0	If (pri <> nil) <b>then</b>	/	Evaluó si la lista principal tiene algo
8		/	
0	ult^.sig:= nuevo;	/	Si SI tiene algo, mi puntero de la última posición apuntará al nuevo
9		/	nodo
1	<b>else</b>		
0			
1	pri:= nuevo;	/	Si NO tiene nada, se carga en la lista principal el nuevo nodo
1		/	
1	ult:= nuevo;	/	Mi puntero ult toma los valores del último nodo que se agrega a la
2		/	lista.
1	<b>end;</b>		
3			



0	<b>procedure insertarOrdenado</b> (var	/	<i>pri es la lista principal y num el dato a cargar en el</i>
1	<i>pri: lista; num: integer);</i>	/	<i>nodo</i>
0	<b>var</b>		
2			
0	nuevo, anterior, actual: lista;	/	Creo tres nodos auxiliares para ordenar la lista principal
3		/	
0	<b>begin</b>		
4			
0	new(nuevo);	/	Creo el nodo nuevo
5		/	
0	nuevo^.dato:= num;	/	El dato / elemento del nodo tomará el valor del dato que ingresa
6		/	
0	actual:= pri;	/	El nodo actual adquiere valor del nodo entrante
7		/	
0	anterior:= pri;	/	El nodo anterior adquiere valor del nodo entrante
8		/	
0	while (actual <> nil) and (act^.dato <	/	Utilizando actual como nodo para recorrer la lista principal y
9	num) <b>do begin</b>	/	verificando si el dato ingresado es mayor al del nodo actual
1	anterior:= actual;	/	El nodo anterior adquiere valores de actual para no perder el
0		/	nodo y seguir avanzando
1	actual:= actual^.sig;	/	El nodo actual adquiere valor del siguiente nodo de la lista para
1		/	seguir verificando datos de la lista
1	<b>end;</b>		
2			
1	If (anterior = actual) <b>then begin</b>	/	Caso que la lista está vacía (la posición buscada es la primera o
3		/	no hay nada cargado)
1	nuevo^.sig:= nil;	/	El nuevo nodo apunta a vacío
4		/	
1	pri:= nuevo; <b>end</b>	/	Inserto el nodo nuevo en la lista
5		/	
1	else	/	Sino es el primer nodo, significa que salió porque el nodo
6		/	siguiente contiene el dato buscado (es mayor al del nodo actual)
1	anterior^.sig:= nuevo;	/	El nodo anterior apuntará al nuevo nodo
7		/	
1	nuevo^.sig:= actual;	/	Saliendo del if el nodo nuevo apuntará al actual (nodo con dato
8		/	mayor)

1	end;			
9				

- Recorre la lista hasta encontrar un elemento mayor, cuando lo haga, se posiciona a su izquierda

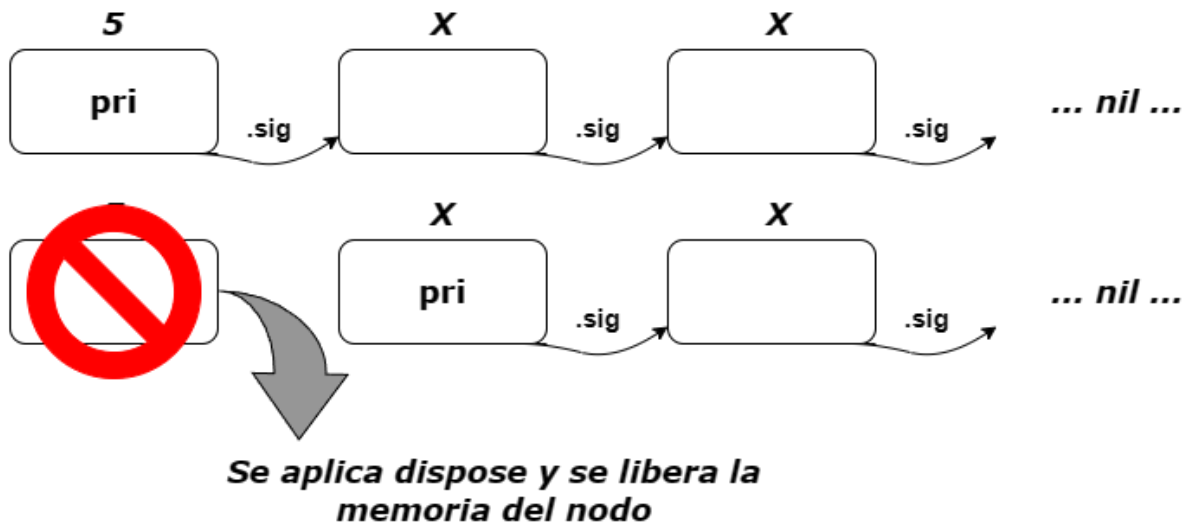


0	<b>procedure eliminarElemento</b> (var	/	pri es la lista principal y num es el dato a buscar y
1	pri: lista; num: integer);	/	eliminar
0	<b>var</b>		
2			
0	anterior, actual: lista;	/	Creo dos nodos auxiliares para ordenar la lista principal
3		/	
0	<b>begin</b>		
4			
0	actual:= pri;	/	Le doy a actual los valores de la lista principal
5		/	
0	while (actual <> nil) and (actual^.dato <>	/	Mientras no termine de recorrer la lista o haya encontrado el
6	num) <b>do begin</b>	/	dato a eliminar
0	anterior:= actual;	/	El nodo anterior toma el nodo de actual para no perder el enlace
7		/	

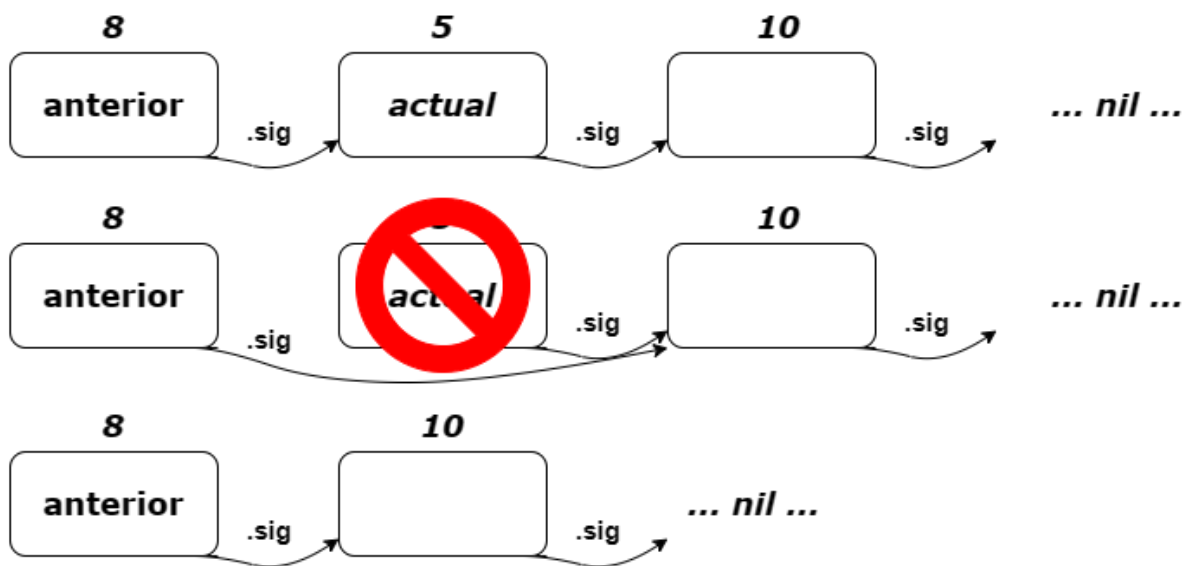
0 8	actual:= actual^.sig;	/	/	El nodo anterior adquiere valor del nodo entrante
0 9	<b>end;</b>			
1 0	if (actual <> nil) <b>then begin</b>	/	/	Si actual tiene un dato no vacío, significa que no termine de recorrer la lista, por lo tanto encontré el dato deseado
1 1	if (actual = pri) <b>then</b>	/	/	Si actual es igual a pri significa que el dato del nodo es el que buscaba
1 2	pri:= pri^.sig	/	/	El primer nodo de pri tomará valores del siguiente nodo (sus datos y direccionamiento)
1 3	<b>else</b>	/	/	Si no es el primer nodo significa que lo encuentre en otra posición
1 4	anterior^.sig:= actual^.sig;	/	/	Hago que el nodo anterior apunte al nodo siguiente del actual
1 5	<b>end;</b>	/	/	
1 6	dispose(actual);	/	/	Elimino la posición del nodo 'actual'
1 7	<b>end;</b>			

**Caso encuentre en el primer nodo de la lista - Buscando el nodo que contenga el valor**





**Caso no está en el primer nodo - Buscando el nodo que contenga el valor 5**



0	<b>procedure</b>	/	<i>pri es la lista principal y num es el dato a buscar y</i>
1	<b>eliminarElementoRep</b> (var pri:	/	<i>eliminar</i>
	lista; num: integer);		
0	<b>var</b>		
2			
0	anterior, actual: lista ;	/	Creó dos nodos auxiliares para ordenar la lista principal
3		/	
0	<b>begin</b>		
4			
0	actual:= pri;	/	Le doy a actual los valores de la lista principal
5		/	
0	anterior:= pri;	/	Anterior copia valores de la lista principal
6		/	
0	while (actual <> nil) <b>do begin</b>	/	Mientras no termine de recorrer la lista
7		/	
0	if(actual^.dato <> num) <b>then begin</b>	/	Verifico si no es un dato que debo eliminar
8		/	
0	anterior:= actual;	/	El nodo anterior toma el nodo de actual para no perder el enlace
9		/	
1	actual:= actual^.sig;	/	El nodo anterior adquiere valor del nodo entrante
0		/	
1	<b>end</b>		
1			
1	<b>else begin</b>	/	Si si debo eliminarlo
2		/	
1	if (actual = pri) <b>then begin</b>	/	Confirmo si es el primer nodo de la lista
3		/	
1	pri:= pri^.sig;	/	Asigné a pri el nodo siguiente
4		/	
1	<b>else</b>	/	Significa que no es el primer nodo
5		/	
1	anterior^.sig:= actual^.sig;	/	Hago que el nodo anterior apunte al nodo siguiente del actual
6		/	
1	dispose(actual)	/	Libero la memoria en donde se encuentra actual
7		/	
1	actual:= anterior;	/	Asigné a actual el nodo anterior para seguir recorriendo
8		/	
1	<b>end;</b>	/	
9		/	

