Repaso- Merge de Listas

Merge de Listas

Combinación de varias listas ordenadas por un criterio específico para generar una nueva lista ordenada por el mismo criterio.

Enunciado de ejemplo

Se disponen de 3 listas con información de las ventas realizadas por c/u de las 3 sucursales de una cadena de supermercados. De cada venta se conoce: **código de producto y cantidad vendida**. Se dispone además de una lista con los precios unitarios de cada producto. **Todas las listas se encuentran ordenadas por código de producto.**

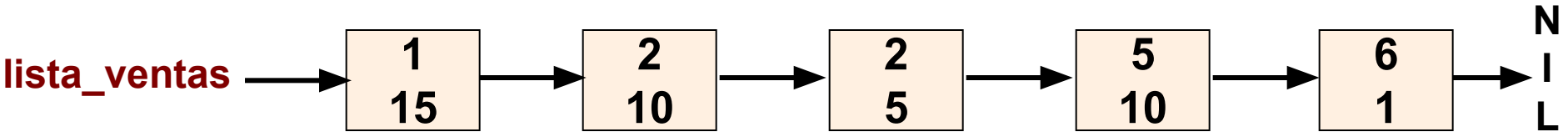
Realizar un modulo que procese los datos y genere una **nueva lista** ordenada por código de producto que contenga para cada producto vendido, la cantidad total vendida y la ganancia total obtenida .

Las listas deben recorrerse una única vez

Precondiciones

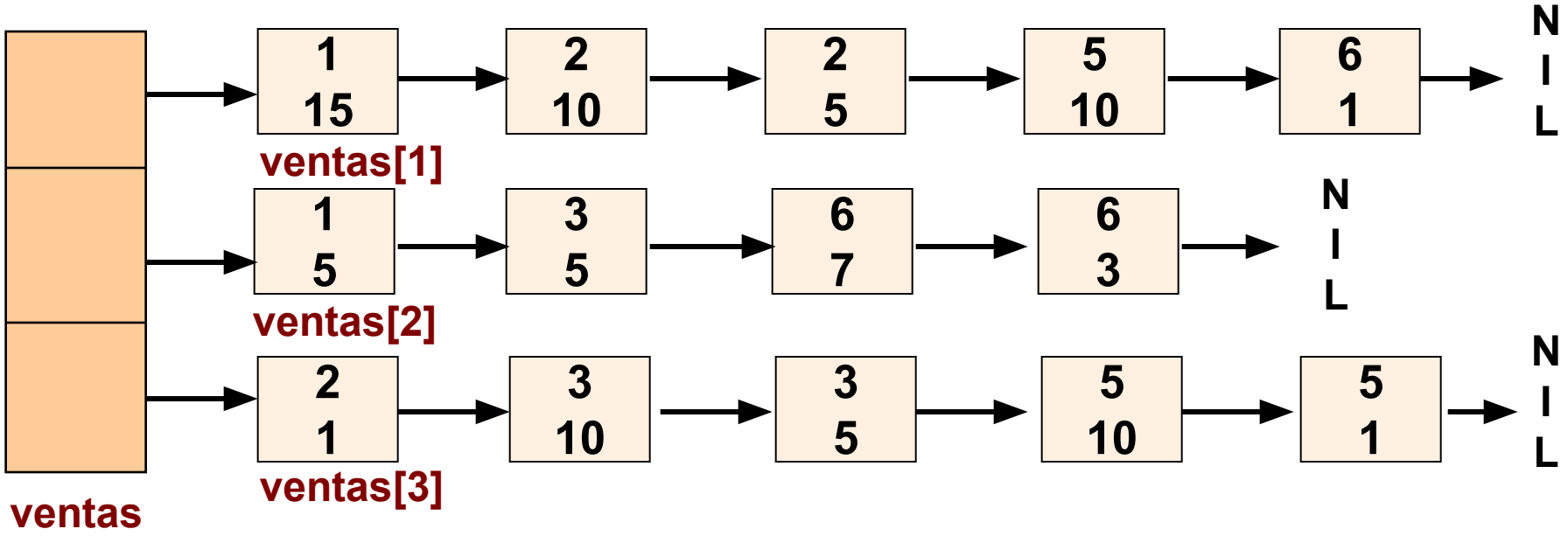
- ✔ Todas las lista que se disponen se encuentran ordenadas por código de producto.
- ✔ Todos los productos vendidos se encuentran en la lista de precios unitarios.
- ✔ Puede ocurrir que no se registren ventas de algunos de los productos de los cuales se dispone su precio unitario.

Estructuras de Datos utilizadas



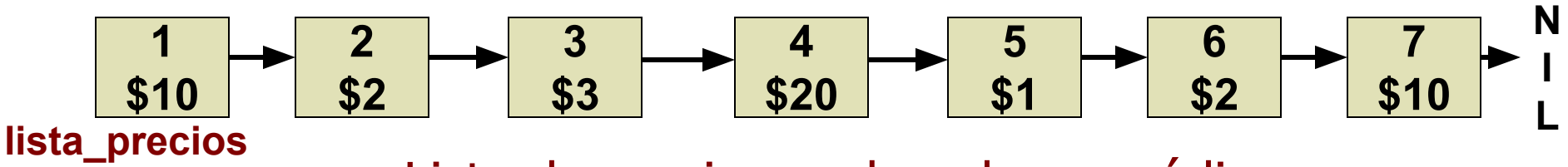
Lista con las ventas de cada sucursal, ordenada por códigos

Estructuras de Datos utilizadas

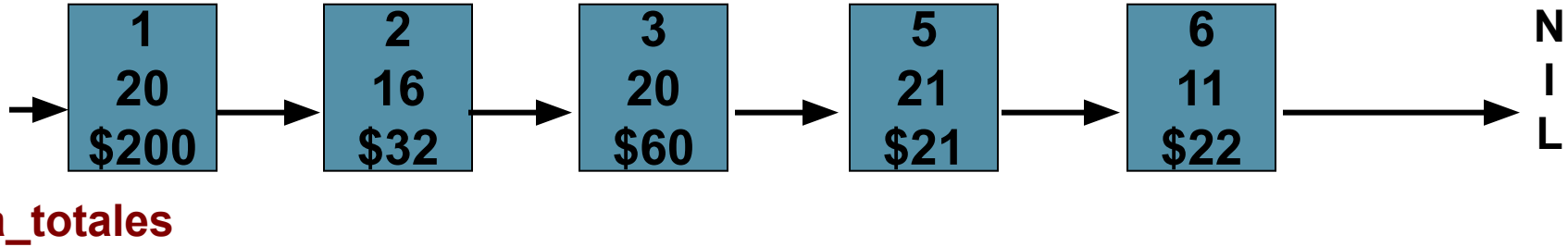


Vector de listas.

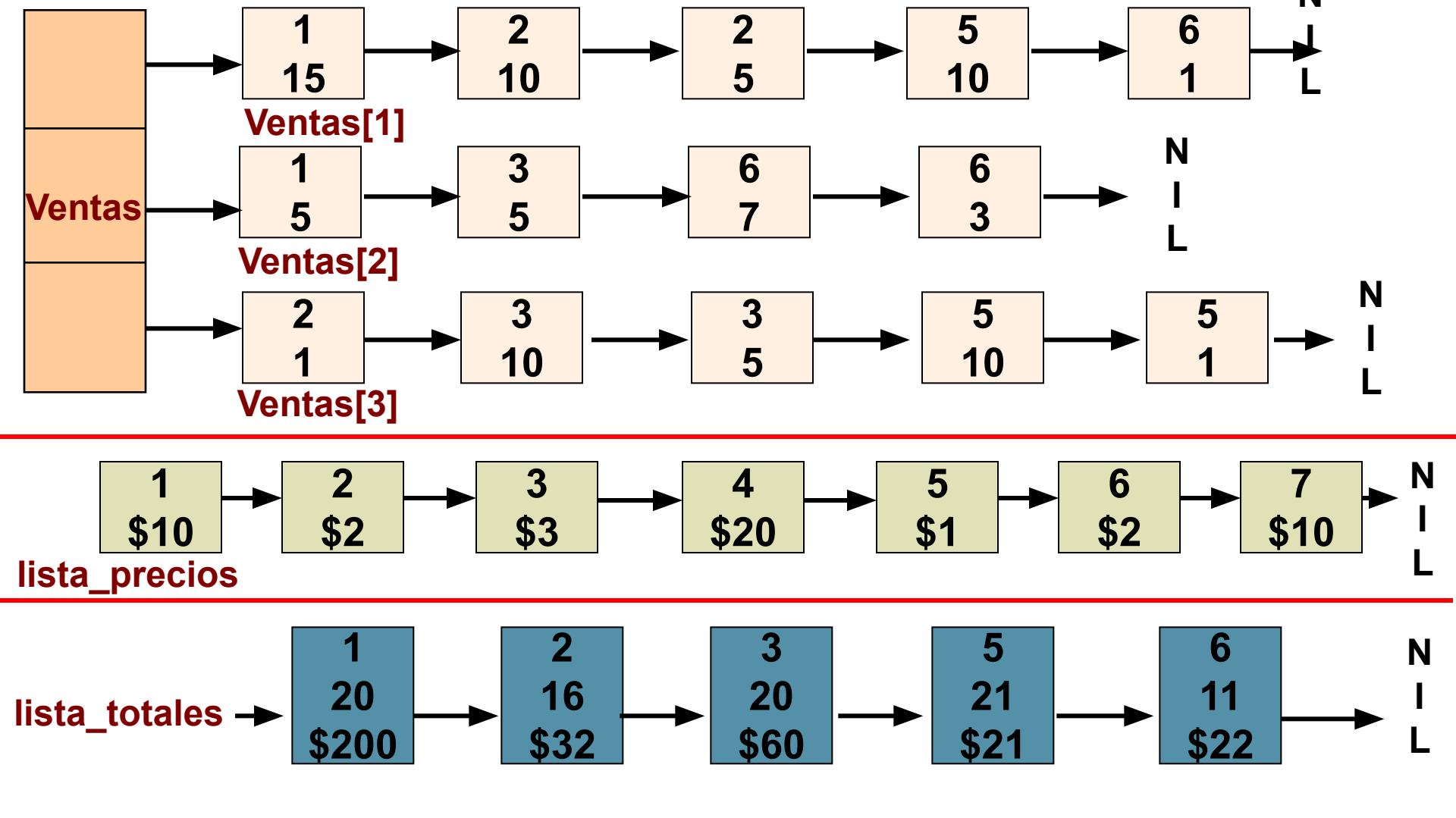
Estructuras de Datos utilizadas



Lista de precios, ordenada por código



Lista de Totales, ordenada por código


Ventas
1
15
2
10
2
5
5
10
6
1
NIL
Ventas[1]
1
5
3
5
6
7
6
3
NIL
Ventas[2]
2
1
3
10
3
5
5
10
5
1
NIL
Ventas[3]
1
$10
2
$2
3
$3
4
$20
5
$1
6
$2
7
$10
NIL
lista_precios
lista_totales
1
20
$200
2
16
$32
3
20
$60
5
21
$21
6
11
$22
NIL

Solución propuesta

```
Program merge;

const
        cant_suc = 3;
        valor_alto = 9999;

type
       venta = record
         cod_pro: integer;
         cant_vend: integer;
       end;

       lista_ventas = ^nodo_ven;
       nodo_ven = record
         ven: venta;
         sig: lista_ventas;
       end;

       ventas = array[1..cant_suc] of lista_ventas;
```

```
precio = record
  cod_pro: integer;
  pre_uni: real;
end;

lista_precios = ^nodo_precio;

nodo_precio = record
        pre: precio;
        sig:lis_precios;
end;

total = record
  cod_pro:integer;
  cant_tot:integer;
  monto_tot:real;
end;

lista_totales = ^nodo_tot;
nodo_tot = record
       tot:total;
       sig: lis_totales;
end;
```

Solución propuesta

```
{Proceso que totaliza las ventas de las sucursales generando una nueva lista con los resultados}

procedure totalizar (ven: ventas; pre: lista_precios; var totales: lista_totales);
var min: venta;
    precio: real;
    prod_actual, total_prod : integer;
    ult : lista_totales;
begin
      totales := nil;                          {Mientras las listas no se vacíen}
      buscar_min(ven, min);
      while ( min.cod_prod <> valor_alto) do begin
        prod_actual := min.cod_prod;
        total_prod := 0;
        while (prod_actual = min.cod_prod) do begin
                  total_prod := total_prod + min.cant_vend;
                  buscar_min(ven,min);
        end;                                   {Mientras el mínimo no cambia del actual, acumulo las ventas del mismo producto}
    {Cuando el mínimo cambia guardo el producto del que obtuve el total}
        buscar_precio(pre, prod_act, precio);
        insertar_atras(totales, ult, prod_act, total_prod, precio);
      end;
end;
```

Solución propuesta

```
{Proceso que busca dentro de las listas de las sucursales el producto con código menor}

procedure buscar_min(var ven: ventas; var min: venta);
var i,aux: integer;

begin
      min.cod_pro := valor_alto; {Cargo un valor alto para poder comparar después}
      for i := 1 to cant_suc do begin
        if (ven[i] <> nil) then
           if (ven[i]^.ven.cod_pro < min.cod_pro)
                then  begin
                 min := ven[i]^.ven;
                 aux := i; {Guardo en que lista encontré el mínimo}
           end;
      end;

      if (min.cod_pro <> valor_alto) {Si alguna lista tenia elementos}
        then  ven[aux] := ven[aux]^.sig; {Descarto el producto que es mi minimo esta vez}
end;
```

Solución propuesta

```
{Proceso que busca el precio de un producto el cual seguro está en la lista}

procedure buscar_precio(var l: lista_precios;  prod: integer; var precio: real);

begin
      while ( l^.pre.cod_pro <> prod) do {No uso l <> nil porque el producto seguro existe}
            l := l^.sig;
      precio := l^.pre.pre_uni;
end;
```

Solución propuesta

```
{Proceso que inserta en la lista resultante el total obtenido para un producto}

procedure insertar_atras (var pri,ult: lista_totales; prod, cant: integer; precio: real);
var nue: lista_totales;

begin
      new (nue);    {Creamos y cargamos el nuevo nodo}
      nue^.tot.cod_pro := prod;
      nue^.tot.cant_tot := cant;
      nue^.tot.monto_tot :=  cant  * precio;
      nue^sig := nil;
      if (pri = nil) then
        pri := nue; {Lista vacia}
      else
        ult^.sig := nue;     {Lista no vacia}

      ult := nue;  {El ultimo es el que acabo de agregar}
end;
```

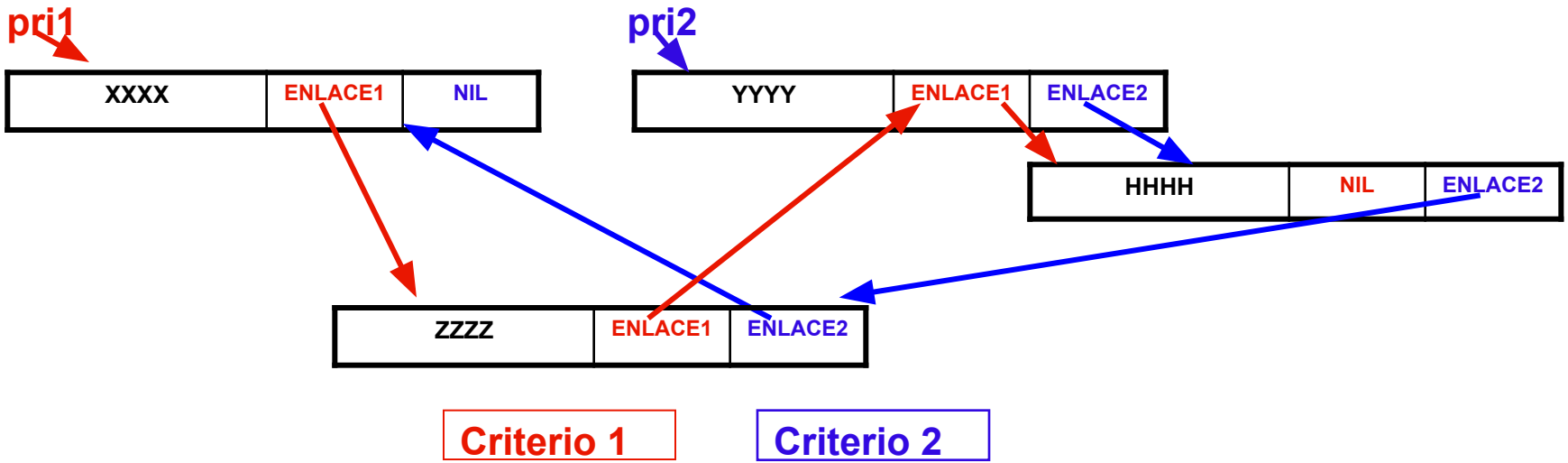
Solución propuesta

```
{Variables del programa principal}

Var
ven: ventas;
pre: lista_precios;
totales: lista_totales

{Comienzo del programa principal}
Begin
      cargar_sucursales( ven);    {No se implementa porque se dispone}
      cargar_precios (pre);       {No se implementa porque se dispone}
      totalizar (ven,pre,totales)
End.
```

Listas Dobles

Listas cuyos nodos están enlazados a otros dos nodos por criterios diferentes.

Repaso- Listas Dobles



Nos permite recorrer la información con dos órdenes diferentes y tener dos vistas diferentes de la misma información.

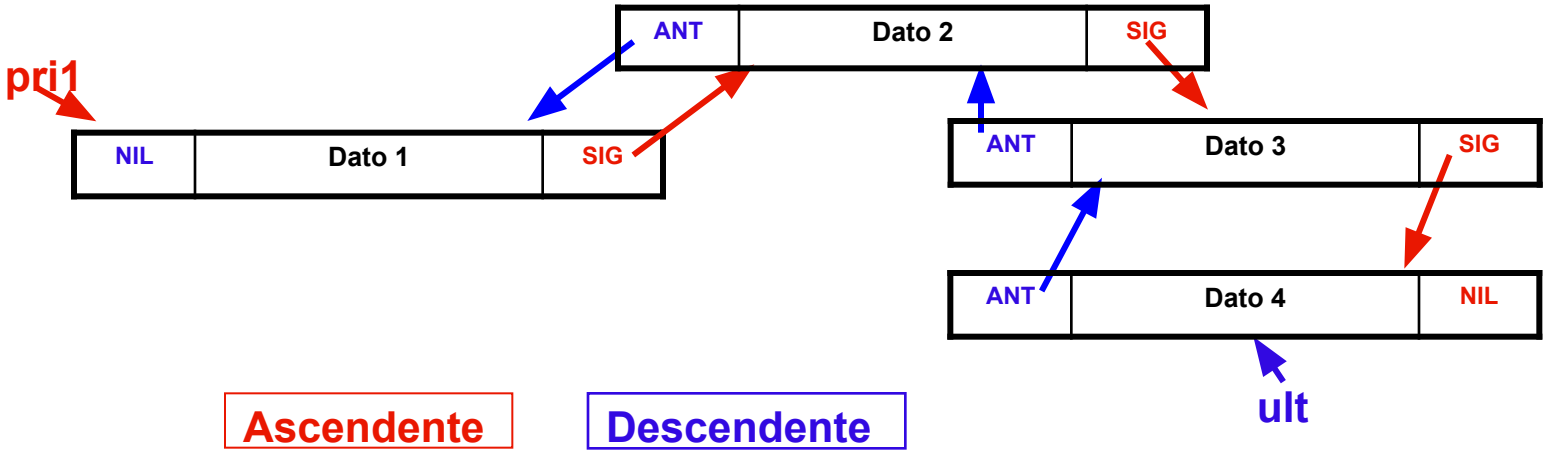
Definición Listas Dobles

```
TYPE   lista = ^nodo;

       nodo = record
     dato: tipo_dato;
     sig_o1: lista;
     sig_o2: lista;
       end;

       lista_doble= record
         pri_o1: lista;
         pri_o2: lista;
       end;

VAR
     ld: lista_doble;
```

Caso particular

Supongamos que cada nodo tiene un enlace al elemento siguiente y otro enlace al anterior

Podemos obtener una lista que se puede recorrer de forma ascendente y descendente.



Enunciado de ejemplo

Se leen caracteres pertenecientes a una palabra, almacenar los mismos en un lista doble para luego determinar si la misma es palíndromo. La lectura finaliza cuando se ingresa el carácter blanco.

Analizando el problema

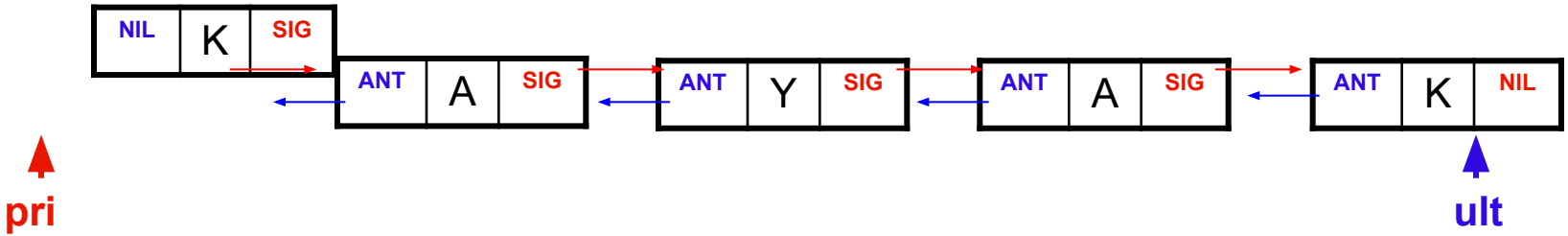
¿Qué deberíamos analizar para saber si una palabra es un palíndromo?

K A Y A K 

E R R E 

S A P O S 

Analizando el problema

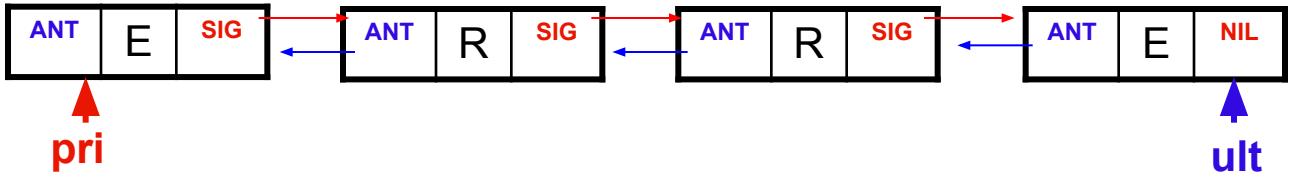


pri^.dato = ult^.dato

¿Qué condición deberíamos agregar para detener el recorrido?

l.pri <> l.ult

Analizando el problema

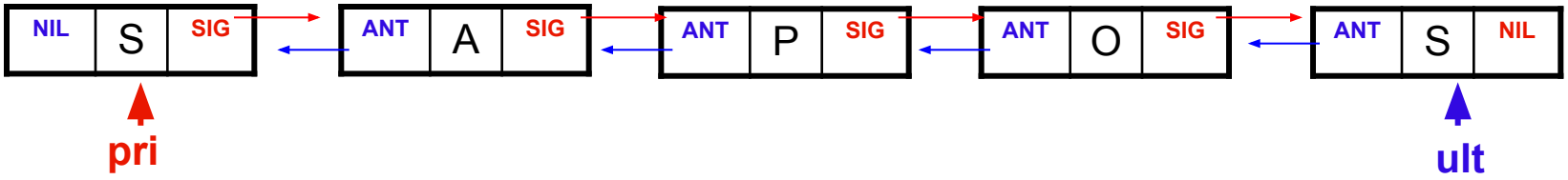


pri^.dato = ult^.dato

¿Qué condición deberíamos agregar para detener el recorrido?

l.pri^.ant <> l.ult

Analizando el problema



l.pri <> l.ult

pri^.dato = ult^.dato

l.pri^.ant <> l.ult

La palabra NO es un palíndromo

Solución propuesta

```
program ejercicio1;
Type

    lista = ^nodo;

    nodo = record
            info: char;
            sig: lista;
            ant: lista;
     end;

    lista_doble = record
             pri: lista;
             ult: lista;
      end;
```

Solución propuesta

```
procedure agregar (var l: lista_doble; c: char);
var
  nue: lista_doble;
begin
  new(nue);
  nue^.info := c;
  nue^.sig  := nil;
  nue^.ant  := nil;
  if (l.pri = nil) then {Lista vacia}
    l.pri := nue;
 else
 begin                {Lista No Vacia}
    l.ult^.sig := nue;
    nue^.ant   := l.ult;
 end;
  l.ult := nue;

end;
```

```
procedure crearlista (var l: lista_doble);
var
  car: char;
begin
  l.pri:= nil;
  l.ult:= nil;
  read(car);
  while (car <> ' ') do begin
    agregar(l, car);
    read(car);
  end;
end;
```

Solución propuesta

```
function palindromo (l: lista_doble): boolean;
var
  ok: boolean;
begin
  ok:= true;
  while (ok) and (l.pri <> l.ult) and (l.pri^.ant <> l.ult ) do
     if (l.pri^.info <> l.ult^.info) then
        ok:= false
     else begin
        l.pri:= l.pri^.sig;
        l.ult:= l.ult^.ant;
    end;
    palindromo := ok;
end;
```

Solución propuesta

```
{Variables del Programa Principal}
var
  ld : lista_doble;
{Comienzo del programa principal}

begin
  crearlista(ld);
  if (palindromo(ld)) then
    writeln('********** Es una palabra palindromo **********')
  else
    writeln('********** No es una palabra palindromo **********');
end.
```

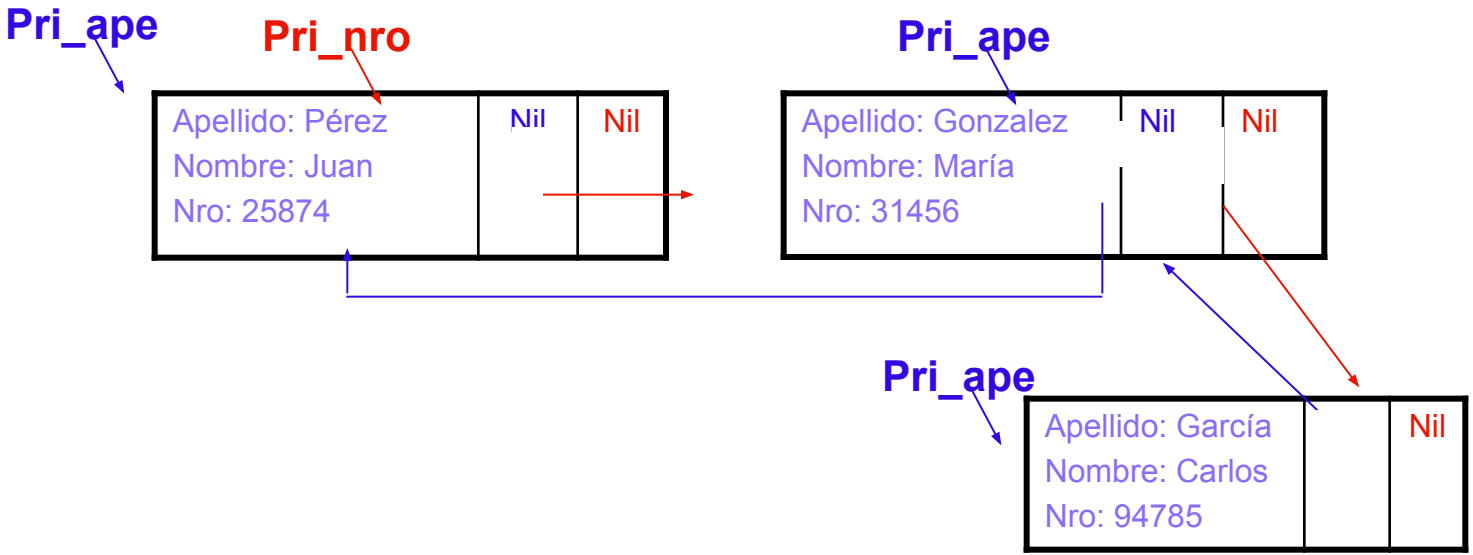
Enunciado de ejemplo

Se lee la información de los alumnos de Facultad. De cada alumno se lee el Número de Alumno, DNI, Apellido y Nombre. La lectura finaliza cuando llega un alumno con Número de Alumno 0. Esta información debe almacenarse en una lista ordenada por dos criterios diferentes:

- Por Apellido
- Por Número de Alumno

Analizando el problema

Apellido: Pérez Nombre: Juan Nro: 25874	Apellido: Gonzalez Nombre: María Nro: 31456	Apellido: García Nombre: Carlos Nro: 94785	Apellido: Nombre: Nro: 0



Solución propuesta

```
program ejercicio2;

TYPE
        cadena   = string[30];

        alumnos  = ^nodo;

        alumno   = record
             numero  : integer;
             nro       : integer;
             apellido: cadena;
             nombre  : cadena;
        end;
```

```
nodo =record
                dato: alumno;
                sig_num: alumnos;
                sig_ape: alumnos;
        end;

        lista_doble = record
              pri_ape: alumnos;
              pri_num: alumnos;
        end;
```

Solución propuesta

```
{Proceso que va leyendo la información de los alumnos y los agrega a la lista}
procedure crearlista (var l: lista_doble);
var
  alu: alumno;
Begin
  {Inicializo la lista}
  l.pri_num:= nil;
  l.pri_ape:= nil;
  leer_alumno(alu);
  while (alu.nro <> 0) do begin
      agregar(l, alu);
      leer_alumno(alu);
  end;
end;

{Variables del programa principal}
var
  la : lista_doble;

Begin   {Comienzo del Programa principal}
  crearlista(la);
end.
```

Solución propuesta

```
{Proceso que lee la información de un alumno}

procedure leer_alumno (var a: alumno);
begin
  writeln;
  writeln('---- Ingrese los datos del Alumno---');
  writeln;
  write('Ingrese DNI( Ingrese 0 para terminar): ');
  readln(a.nro);
  if (a.dni <> 0) then begin
     write('Ingrese el Numero de alumno: ');
     readln(a.numero);
     write('Ingrese el Apellido del alumno: ');
     readln(a.apellido);
     write('Ingrese el Nombre del alumno: ');
     readln(a.nombre);
  end;
end;
```

Solución propuesta

```
{Proceso que agrega un alumno a la lista por ambos órdenes}

procedure agregar (var l: lista_doble; a:alumno);
var
  nue: alumnos;
begin
  new(nue);
  nue^.dato    := a;
  nue^.sig_ape := nil;
  nue^.sig_num := nil;

  {Agrego por los dos órdenes}

  agregar_x_apellido(l, nue);
  agregar_x_numero  (l, nue);
end;
```

Solución propuesta

```
{Proceso que agrega un alumno a la lista respetando el orden por apellido}

procedure agregar_x_apellido (var l: lista_doble; nue: alumnos);
var
  ant, act: alumnos;
begin
  act:= l.pri_ape;
  while (act <> nil) and (nue^.dato.apellido>act^.dato.apellido) do begin
    ant:= act;
    act:= act^.sig_ape;
  end;
  if (act = l.pri_ape) then begin {agrega primero en la lista}
    nue^.sig_ape:= l.pri_ape;
    l.pri_ape := nue;
  end
  else begin  {agregar en el medio o al final de la lista}
    ant^.sig_ape := nue;
    nue^.sig_ape := act;
  end;
end;
```

Solución propuesta

```
{Proceso que agrega un alumno a la lista respetando el orden por número de alumno}

procedure agregar_x_numero (var l: lista_doble; nue: alumnos);
var
  ant, act: alumnos;
begin
  act:= l.pri_num;
  while (act <> nil) and (nue^.dato.numero > act^.dato.numero) do begin
    ant:= act;
    act:= act^.sig_num;
  end;
  if (act = l.pri_num) then begin {agrega primero en la lista}
    nue^.sig_num:= l.pri_num;
    l.pri_num := nue;
  end
  else begin                    {agregar en el medio o al final de la lista}
    ant^.sig_num := nue;
    nue^.sig_num := act;
  end;
end;
```