



# Oinarrizko Programazioa

## ***4. Oinarrizko Datu-egiturak***

***Erakusleak eta lista dinamikoak***



# Edukiak

1. Sarrera
2. Programazioko oinarrizko kontzeptuak
3. Programen beheranzko diseinua.  
Azpiprogramak: funtzioak eta prozedurak
4. **Oinarrizko datu-egiturak**
5. Programazio-lengoaiei erabilera
6. Aplikazio-adibideak



# Oinarrizko datu-egiturak

1. Sarrera
2. Bektoreak
3. Matrizeak
4. Erregistroak
5. Datu-egitura mistoak
6. Listak
  - Lista estatikoak
  - **Lista dinamikoak**

# Kokapena eta motibazioa

5
---

Z

Nagore	945123456
--------	-----------

L1.Izena

L1.Telefonoa

L1

```
type Integer...; --mota-definizioa
```

```
Z : Integer; --aldagai-erazagupena
```

```
Z := 5; --esleipena
```

---

```
type Lagun is --mota-definizioa
```

```
record
```

```
    Izena : String (1 .. 6);
```

```
    Telefonoa : String (1 .. 9);
```

```
end record;
```

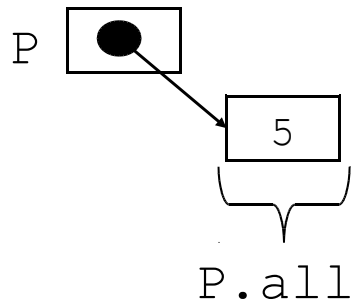
```
L1, L2: Lagun; --aldagai-erazagupenak
```

```
L1.Izena := "Nagore"
```

```
L1.Telefonoa := "945123456"
```

*Aldagai bat  
erazagutzeak  
memorian  
leku bat  
hartzea esan  
nahi du,  
aldagai horrek  
aurrerantzean  
izango dituen  
balioak  
gordetzeko.*

# Adako access motak: erakusleak




```
type Int_Erak is access Integer; --mota-definizioa
P : Int_Erak; --erakuslea (access motako aldagaia):
               -- P = null

P := new Integer; --memoria dinamikoan toki-hartzea
                  -- P /= null

P.all := 5; --balio-esleipena, aldagai izengabea

-- edo, gauza bera dena (laburrago idatziz):
P := new Integer'(5);
```

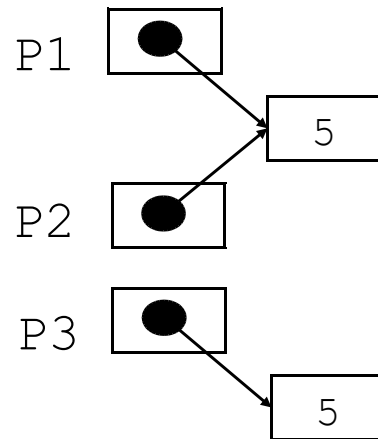
- Erakusle batek memoria-helbide bat gordetzen du.
- Erazagutu berritan, `null` balioa du (hau da, "ez du ezer erakusten").
- Memoria dinamikoan leku bat hartzeko, `new` egin behar da (erakusleak erakusten duen motarako aproposa izango da memoria-leku hori).



# Adako `access` motak: objektuen sorkuntza: `new`

- Memoria dinamikoan leku bat hartzeko, `new` egin behar da.
- Bi zeregin betetzen ditu `new` eragileak:
  - `Integer` motako (aurreko adibidearekin jarraituz) **objektu berri bat** errepresentatzeko memoria-posizio bat erreserbatzen du
  - Objektu hori errepresentatzeko erabiliko den **memoria-posizioaren helbidea** itzultzen du. Horixe da erakusleari asignatuko zaion balioa.

# Adako access motak: helbidea eta erakutsitako balioa



Kontuz!!!

$P1 = P2 \neq P3$

$P1.all = P2.all = P3.all = 5$

- Mota bateko erakusleek beti mota bereko objektuak erakusten dituzte:

```
type Lagun_Erak is access Lagun;
```

```
Laguna1, Laguna2, Laguna3: Lagun_Erak;
```

- Kasu honetan, Laguna1, Laguna2 **eta** Laguna3 erakusleek Lagun motako erregistroak erakutsiko dituzte.

# Adako access motak: atzipena edo erreferentziatzea

```
type Lagun_Erak is access Lagun;  
Laguna1, Laguna2, Laguna3 : Lagun_Erak;
```

Hasieratu Laguna1 aldagaia Nagoreren datuekin:

```
Laguna1 := new Lagun;  
Laguna1.all.Izena = "Nagore"; -- Laguna1.Izena  
Laguna1.all.Telefonia = "945123456"; --Laguna1.Telefonia
```

Erregistroen eta  
array-en kasuan,  
all ez idatzita ere  
balio du

Forma laburragoak gauza bera egiteko:

```
- Laguna1 := new Lagun'("Nagore", "945123456");
```

- Edo, pixka bat esplizituago egin nahi izanez gero:

```
Laguna1 := new Lagun'(Izena => "Nagore",  
                      Telefonia => "945123456");
```





# Adako access motak: esleipenak

```
type String_Erak is access String;  
S1, S2 : String_Erak;
```

```
S1 := new String'("Kaixo");  
S2 := new String'("Agur "); --luzera berdineko string-ak
```

```
S1.all := S2.all; -- S2-k erakutsitako balioa esleitzen dio  
               -- S1-ek erakutsitako aldagaiari
```

1

```
S1 := S2; -- S1-ek S2-ren balioa hartzen du: orain, biek  
          -- aldagai bera erakusten dute
```

2

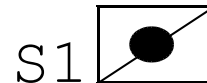
Ariketa: Egin bi egoera horiek adierazten dituzten irudiak.

# Adako `access` motak: `null` balioa

```
type String_Erak is access String;  
S1, S2 : String_Erak;
```

Zer balio dute S1 eta S2-k?

`null`, ez dute ezer erakusten!



Eta, orduan, `S1.all` edo `S2.all` egiten saiatzen bagara, zer gertatuko da?

`Constraint_Error`



# Ariketa

```
type Int_Erak is access Integer; --mota-definizioa  
A, B, C, D : Int_Erak; --erakusleak
```

```
A := new Integer'(3);  
B := new Integer'(3);  
C := A;  
D := new Integer;  
D.all := A.all;  
C.all := 1;
```

Ariketa: Irudikatu esleipen horien ondorengo egoera



# Ariketa

`A = B`

`A = C`

`A = D`


`A.all = B.all`

`A.all = C.all`

`A.all = D.all`


`B.all = D.all`

Ariketa: Aurreko ariketan sortutako egoerari erreparatuz, zer balio dute aurreko baldintza horiek?



# Datu-egitura dinamikoak. Motibazioa.

- Problema: *Irakurri zeroz bukatzen den oso-sekuentzia bat eta idatzi ordenatuta.*
- Orain arte ikusitakoaren arabera: array bat (edo lista estatiko bat) behar dugu, zenbakiak irakurri ahala gordetzen joateko.
- Array bidezko datu-egiturak **estatikoak** dira.
- Zein tamaina eman array horri edo lista estatiko horri?
  - 100?: Txikiegi gerta daiteke!
  - 1.000.000?: Memoria-posizio asko hartu bai, baina, seguruenik, gehienak ez dira erabiliko!
  - Ez dago soluzio onik, sekuentziaren tamaina aldeaz aurretik ez baldin badakigu.



# Datu-egitura dinamikoak. Motibazioa (II).

- Beste arazo bat:
  - **Lista estatikoetan**, txertaketek eta ezabaketek elementuak eskuinerantz edo ezkerrerantz desplazatu beharra ekartzen dute.
  - Eragiketa horiek kostu handikoak dira, ez dira eraginkorrak.
  - Aplikazio batean, txertaketak eta ezabaketak maiz egin beharra badago, ez da datuen errepresentazio egokiena.



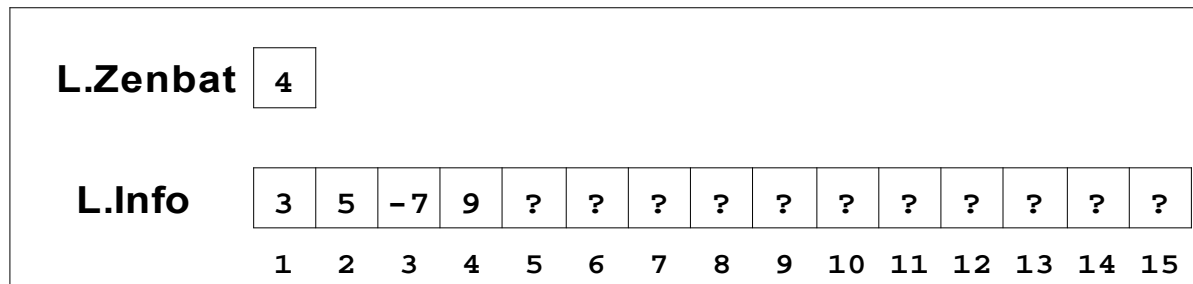
# Datu-egitura estatikoak eta dinamikoak

- Datu-egitura estatikoak (array bidezkoak):
  - Konpilatu aurretik ezarri behar da zenbat elementu izango dituen (jakin behar da zenbat memoria hartuko duen)
  - Exekuzio-garaian aldagaiek tamaina finkoa dute eta ezin da aldatu
- Datu-egitura dinamikoak (erakusle bidezkoak):
  - Exekuzio-garaian memoriako posizio gehiago har ditzakete, behar izanez gero (=> behar dena baino ez dugu erabiliko)
  - Elementu berriak txertatzeko edo ezabatzeko, ez da beharrezkoa gainontzeko elementuak mugitzen ibiltzea

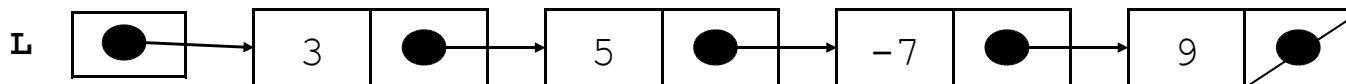
# Datu-egitura estatikoak eta dinamikoak (II)

- Datu-egitura estatikoak (array bidezkoak):

$L = \langle 3, 5, -7, 9 \rangle$



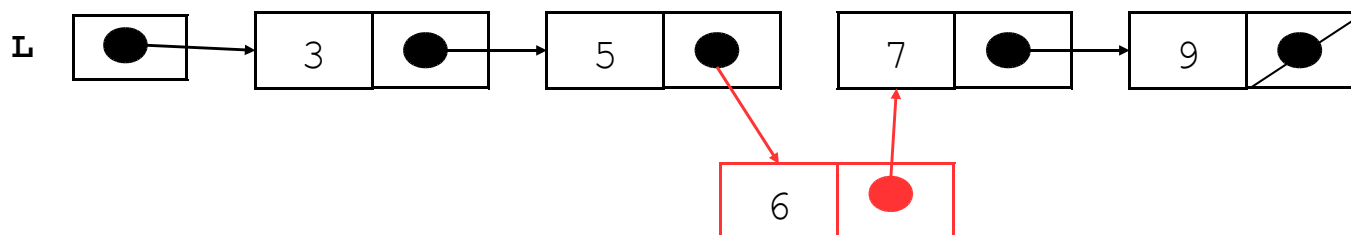
- Datu-egitura dinamikoak (erakusle bidezkoak):



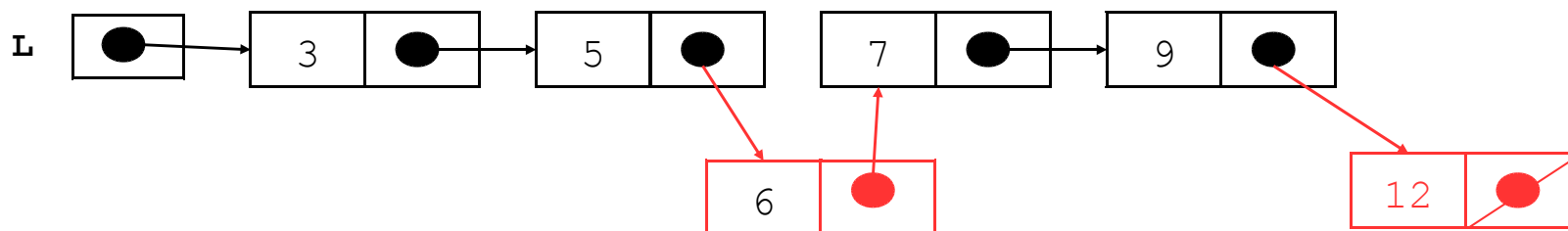


# Datu-egitura dinamikoak

- Datu-egitura dinamikoek erraztasunak ematen dituzte:
  - osagai bat edozein tokitan txertatzeko (hasieran, bukaeran, tartean):



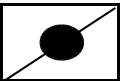
- Datu-egituraren beraren tamaina aldatzeko (aurrekotik ondorioztatzen dena, noski):



# Datu-egitura dinamikoak (II)

- Erazagupenak: **aurredefinizioa**

```
type Nodo;  
type Lista is access Nodo;  
type Nodo is record  
    Info : Integer;  
    Hurrengoa : Lista;  
end record;  
L : Lista := null;
```

L  hasierako egoera, goiko aldagai-erazagupenaren ondorengoa

L  listan osagaiak gehitu eta gero

Info Hurrengoa    Info Hurrengoa    ...    Info Hurrengoa

- lista nodoz osatuta dago: lista estekatua esaten zaio
- nodo bakoitza erregistro bat da, bi eremurekin: info eta hurrengoa
  - info: listaren osagaia
  - hurrengoa: listaren hurrengo osagaia erakusten duen erakuslea
- lista bera lehen nodoaren erakusle batek errepresentatzen du: L
- lista hutsa: lista errepresentatzen duen erakusleak (L) null balio du
- listaren amaiera: azken nodoaren hurrengoa eremuak null balio du

# Ariketa.

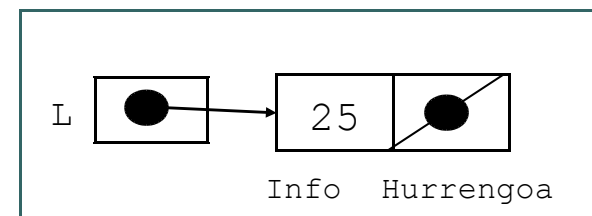
## Gehitu lehen osagaia.

- Aurreko transparentziako erazagupena emanda:
  - Listari gehitu lehen osagaia, 25 balioarekin.

```
L := new Nodo;
```

```
L.Info := 25;
```

```
L.Hurrengoa := null; --ez da ezinbestekoa
```



- edo, laburrago:

```
L := new Nodo'(25, null);
```

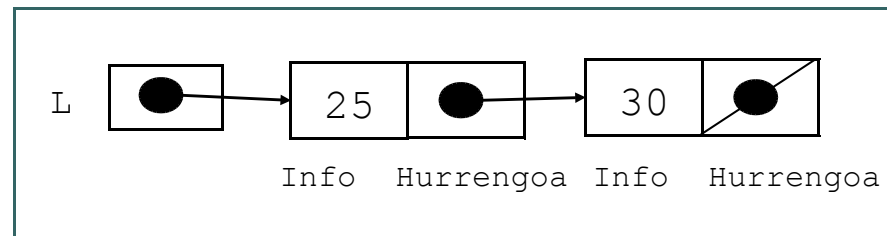
- edo:

```
L := new Nodo'(Info => 25, Hurrengoa => null);
```

# Ariketa.

Gehitu bigarren osagaia.

- Orain, gehitu bigarren osagaia, 30 balioarekin.



```
L.Hurrengoa := new Nodo;  
L.Hurrengoa.Info := 30;  
L.Hurrengoa.Hurrengoa := null;
```

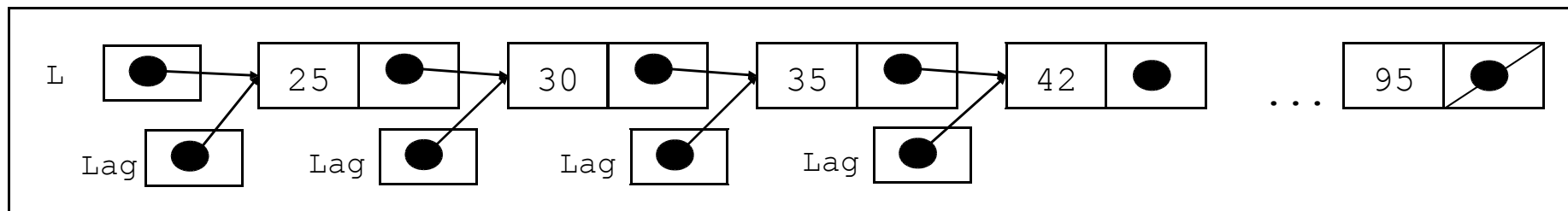
# Ariketa

Lista dinamikoko elementu guztiak inprimatzea.

- Korritu lista, eta inprimatu zenbaki guztiak,

`Lag : Lista;`

erazagupena eginda dagoela emanik.



```
Lag := L; -- erakusle lagungarria
while Lag /= null loop
    Ada.Integer_Text_IO.Put (Lag.Info);
    Lag := Lag.Hurrengoa; --hurrengora pasatu
end loop;
```

# Ariketa

## Elementu baten bilaketa lista ez-ordenatuan.

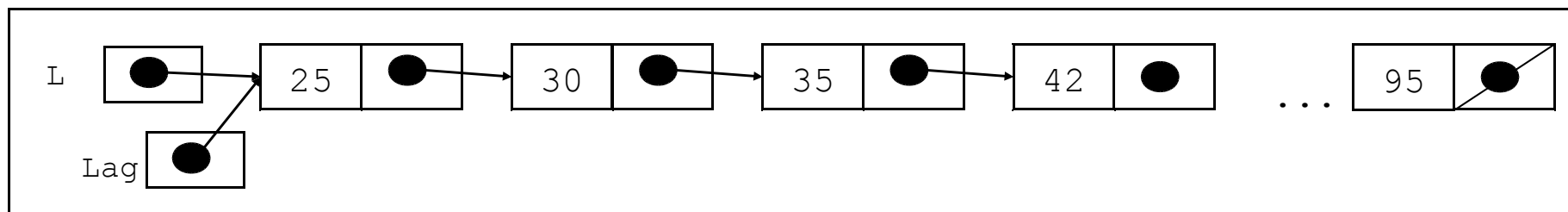
- Korritu lista, emandako zenbaki baten bila: adibidez, egiaztatu 42 zenbakia listan dagoela,

```
Aurkitua : Boolean := False;
```

```
Lag : Lista;
```

erazagupena eginda dagoela emanik.

Oharra: Ez suposatu L lista ordenatuta dagoenik.



```
Lag := L; -- erakusle lagungarria
```

```
while (Lag /= null) and (Aurkitua = False) loop
```

```
  if Lag.Info = 42 then
```

```
    Aurkitua := True;
```

```
  else
```

```
    Lag := Lag.Hurrengoa; --hurrengora pasatu
```

```
  end if;
```

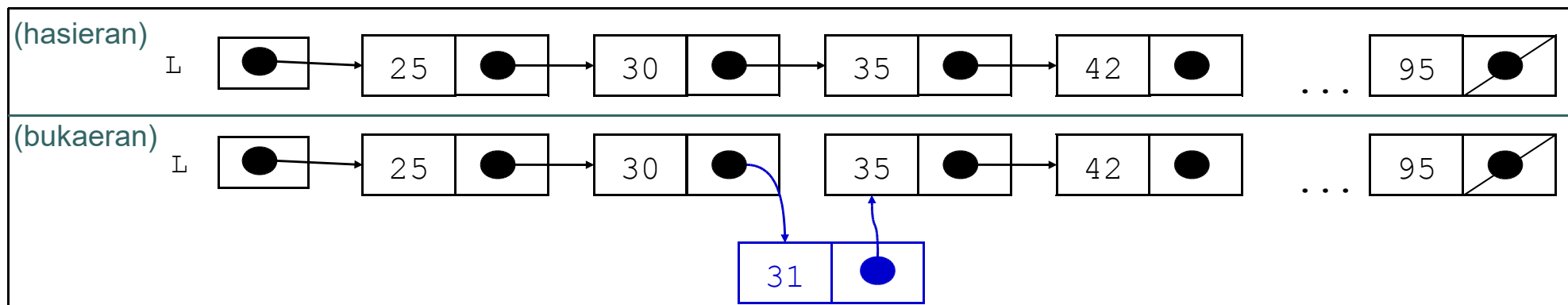
```
end loop;
```

# Ariketa

## Elementu baten txertaketa lista ordenatuan.

- Txertatu listan  $N$  zenbakia, dagokion tokian (lista txikienetik handienara dago ordenatuta).

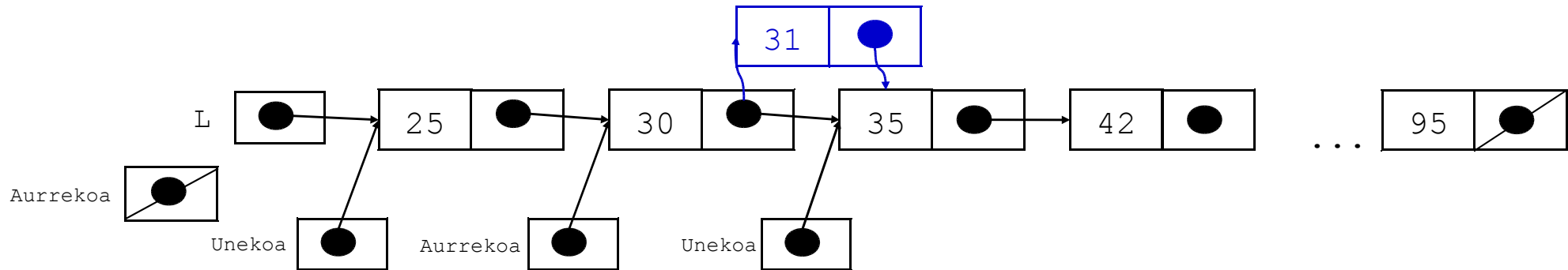
Adibidea,  $N=31$ :



- Ariketa honetan, kontuan izan behar dira honako kasu hauek:
  - lista hutsik egon daiteke
  - $N$  izan daiteke listaren lehen osagaia baino txikiagoa (eta, orduan, lehen tokian txertatu beharko da)
  - $N$  izan daiteke listaren azken osagaia baino handiagoa (eta, orduan, azken tokian txertatu beharko da)
  - $N$  txertatu beharreko tokia tarteko edozein izan daiteke
- Idea: Lista motako bi aldagai lagungarri erabiliko ditugu: Aurrekoa eta Unekoa

# Soluzioa

Elementu baten txertaketa lista ordenatuan.



```
Aurrekoa := null;  
Unekoa := L;  
while (Unekoa /= null) and then (Unekoa.Info < N) loop  
    Aurrekoa := Unekoa;  
    Unekoa := Unekoa.Hurrengoa;  
end loop;  
if Aurrekoa = null then -- hasieran txertatu  
    L := new Nodo'(N, L);  
else -- Aurrekoa /= null  
    Aurrekoa.Hurrengoa := new Nodo'(N, Unekoa);  
end if;
```





# Oso\_Listak:

## Inplementazio dinamikoa.

### Erazagupenak

```
package Lista_Dinamikoak is
  type Lista is private;

  procedure Sortu_Hutsa (L : out Lista) ;
  -- Post: L, lista hutsa

  procedure Txertatu_Hasieran (L : in out Lista; X : in Integer);
  -- Aurre: L, oso-lista
  --       X, zenbaki osoa
  -- Post: L listaren hasieran X dago, eta ondoren hasierako L lista

  function Dago (L : in Lista; X : in Integer) return Boolean;
  -- Sarrera: L, oso-lista
  --       X, zenbaki osoa
  -- Irteera: True, L-n X baldin badago
  --       False, bestela

  -- ...

  procedure Idatzi (L : in Lista);
  -- Sarrera: L, zenbaki-lista
  -- Eragina: L-ko zenbakiak idatzi dira irteera estandarrean

private
  type Nodo;
  type Lista is access Nodo;
  type Nodo is record
    Info : Integer;
    Hurrengoa : Lista;
  end record;
end Lista_Dinamikoak;
```



# Oso\_Listak:

## Inplementazio dinamikoa.

### Eragiketen implementazioa

```
with Ada.Text_Io, Ada.Integer_Text_Io;

package body Lista_Dinamikoak is

  procedure Sortu_Hutsa (L : out Lista) is
  begin
    L := null;
  end Sortu_Hutsa;

  procedure Txertatu_Hasieran (L : in out Lista ;
                               X : in Integer) is
  begin
    L := new Nodo'(X, L);
  end Txertatu_Hasieran;

  function Dago (L : in Lista; X : in Integer) return Boolean is
    Aurkitua : Boolean := False;
    Lag : Lista := L;
  begin
    while (Lag /= null) and (Aurkitua = False) loop
      if Lag.Info = X then
        Aurkitua := True;
      else
        Lag := Lag.Hurrengoa; --hurrengora pasatu
      end if;
    end loop;
    return Aurkitua;
  end Dago;

  -- ...
```



# Oso\_Listak:

## Inplementazio dinamikoa.

### Eragiketen inplementazioa (II)

```
procedure Idatzi (L : in Lista) is
  Lag : Lista := L;
begin
  Ada.Text_IO.Put ("<");
  while Lag /= null loop
    Ada.Integer_Text_IO.Put (Lag.Info, 5) ;
    Lag := Lag.Hurrengoa; --hurrengora pasatu
  end loop;
  Ada.Text_IO.Put_Line (" >");
end Idatzi;

end Lista_Dinamikoak;
```