MONDRAGON UNIBERTSITATEA

GOI ESKOLA POLITEKNIKOA

ESCUELA POLITÉCNICA SUPERIOR

Programazio Aurreratua:

Kontzeptu aurreratuak

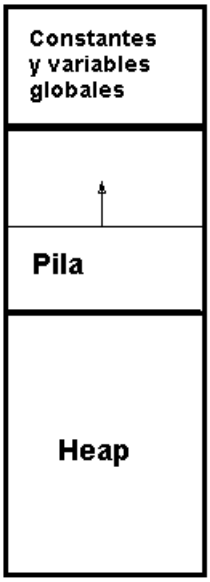
Informatika Saila

Dpto. Informática

Indizea

1 Memoria

Programa batek erabiltzen duen memoria, datuen eta aldagaien atalari dagokionean hiru talde funtzional ezberdinetan banatzen da.



- Konstanteak eta aldagai globalak:
  - Programaren bizi iraupen osoan existitzen duten aldagaiak
  - printf batean erabiltzen diren karaktere kate moduko konstanteak.

- Pila: funtzio bakoitzaren aldagai lokalak dauden memoriako atala da.

  - Funtzio bat exekutatzen hasten den bakoitzean, pilan aldagai lokalak (eta parametroak) gordetzeko adina espazio erreserbatzen da.
  - Funtzio honek bere exekuzioa amaitzen duenean, aldagai lokalentzat pilako atal honetan erreserbatuta zeukan espazioa liberatzen da. ***Aldagai lokal hauek existitzea uzten dute***.

- Heap: guk nahi dugun bezala erreserbatu eta liberatu ahal izango dugu memoriako atal hau. ***Memoria dinamiko*** bezala ere ezagutzen da.
  - malloc eta free funtzioak arduratuko dira espazioa erreserbatu eta libratzen hurrenez hurren.

- malloc funtzioa erabilita behin memoria erreserbatzen denean, memoriako zona hori ez da liberatuko zona horrentzat free funtzioa exekutatzen ez den bitartean.

```
void *malloc(int bloke_tamaina_bytetan); /*NULL emango du errorea baldin
                                           badago */
void  free(void*  blokera_erakuslea);
```

Memoria dinamikoaren erreserba egiteko arrazoia:

·Aldagai bat deklaratzean, array bat adibidez, ez dakigu zer luzera izango duen. Askotan behar baino espazio gehiago erreserbatzen da, edo gutxiegi agian.

Hori dela eta, memoria dinamikoaren erreserba egiten da.

Adibidea:

```
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
   int i;
   int *j;

   printf("j(erakuslea) zenbaki osoaren tamaina:%d\n", sizeof(j));
//ondoren, memorian int motako 4 daturentzat lekua erreserbatuko da
   for (i=0;i<4;i++) {
      j=(int *)malloc(sizeof(int));
//malloc funtzioak j-rentzat erreserbatu den memoriara erakusle bat emango digu
      printf("\n j=%u",j); //bueltan.Helbide bat emango digu, alegia.
   }

   System("PAUSE");
   return 0;
}
```

2 Egiturak

- Egiturek, elkarrekin zerikusia duten mota desberdinetako aldagaiak biltzeko mekanismoa daukate. Adibidez:

```
struct data {
   int eguna;
   int hilabetea;
   char hilab_izena[10];
   int urtea;
};
```

- Goiko adibidean, aldagai batzuentzat patroi bat definitzen ari gara. Mota honetako aldagaiak honela deklaratzen dira:

```
Struct data data1, urtebetetzea;
```

Baina ondorengo moduan ere lor genezake gauza bera:

```
struct data {
   int eguna;
   int hilabetea;
   char hilab_izena[10];
   int urtea;
} data1, urtebetetzea; // mota definizioa
```

- Egitura batek, beste egitura bat eduki dezake baldin eta bigarrena beste mota batekoa baldin bada. Adibidez:

```
#include <stdio.h>
      #define LUZERA 30

void main(void) {
   struct pertsona {
      char izena[LUZERA];
      int adina;
   };
   struct lantegia {
      char izena[LUZERA];
      char lantegi_mota[LUZERA];
   };
   struct langilea {
      struct pertsona kurrela;
      struct lantegia enpresa;
   };
   struct langilea mutila;
   char str[2];

   printf("\n Langilearen datuak. \n Sartu izena :  ");
   gets(mutila.kurrela.izena);
   printf("Adina : ");
   scanf("%d",&mutila.kurrela.adina);
   fflush(stdin);
   printf("\n Sartu enpresaren izena : ");
   gets(mutila.enpresa.izena);
   printf("Enpresaren iharduera: ");
   gets(mutila.enpresa.lantegi_mota);
   printf("Langilearen Informazioa : ");
   printf("\t Izena : %s\n",mutila.kurrela.izena);
   printf("\t Adina: %d\n",mutila.kurrela.adina);
   printf("\t Enpresaren izena : %s\n",mutila.enpresa.izena);
   printf("\t Enpresaren iharduera:%s\n",mutila.enpresa.lantegi_mota);

   gets(str);
}
```

- Egituren erabilera. Bere ataletara sarrera:

  Adibide1:

```
struct data data1, urtebetetzea;
data1.eguna=29;
if (data1.hilabetea==2) strcpy(data1. hilab_izena,”Otsaila”);
```

  Adibide2:

```
#include <stdio.h>

void main(void) {
     struct bat  {
          char kar;
          int zbkia;
     };
     struct bat a;
     char str[2];

     a.zbkia=2;
     a.kar='A';
     printf("\nEgituraren elementuak: %d, %c\n", a.zbkia,a.kar);

     gets(str);
}
```

- Funtzioei egiturak argumentu bezala pasatzea.

  Adibidea:

```
#include <stdio.h>
#define LUZERA 30
struct langileak {
   char izena[LUZERA];
   int num;
};
struct langileak  izen_berria (void);
void datuak_ikusi (struct langileak *kurrela);

int main(void) {
   char str[2];
   struct langileak lan_1;

   lan_1=izen_berria();
   printf("\n Langileen izenak: \n");
   datuak_ikusi (&lan_1);
   gets(str);
}
struct langileak izen_berria(void) {
   struct langileak kurrela;
   char str[2];

   printf("\n1.Langilearen datuak. \n Sartu izena: ");
   gets(kurrela.izena);
   printf("\n Sartu bere zenbakia: ");
   gets(str);
   sscanf(str,"%d",&kurrela.num);
   return(kurrela);
}
```

```
void datuak_ikusi(struct langileak *kurrela) {
   printf("\nLangilea:\n");
   printf("\tIzena : %s\n",kurrela->izena);
   printf("\tLangile kodea : %d\n",kurrela->num);
}
```

- Egiturentzako erakusleak:

```
struct data *pf, jaiotza;
pf=&jaiotza;
if(pf->hilabetea==2)  strcpy(pf->hilab_izena,"Otsaila");
```

pf, erakusle motako aldagai bat da, data egitura motako datuak apuntatzen dituen erakusle bat.Ondoren, jaiotza egiturak daukan helbidea pasatzen diogu aldagai horri, pf erakusle motako aldagaiari. Bukatzeko, jaiotza egituran idatzita dagoena pf erakuslea erabiliaz nola irakurri dezakegun idatzi da.

- Autoerreferentziatzea. Egitura batek ezin du bere mota berdina den datu mota bat eduki bere ataletako batean, baina eduki dezakeena zera da: bere datu mota berdina apuntatuko duen erakusle bat.
  Adibidea:

```
struct nodo {
  int elementua;
  struct nodo *pHurrengoa;
}
```

- **typedef**: honekin, moten izen berriak definitu genitzake.

```
typedef struct nodo {
    int elementua;
    struct nodo *pHurrengoa;
} NODO, *PNODO
```

  – Kode zati honekin, konpilatzaileari **NODO** *struct nodo*-ren sinonimo bezala hartu dezan eta **PNODO** *struct nodo* * -ren sinonimo bezala hartu dezan esanez gabiltza.

  – #define komandoaren antzekoa da baina indartsuagoa.

- **Unioak**: Suposa dezagun, aldagai batean batzutan zenbaki oso bat, bestetan double bat eta beste batzuetan karaktereak apuntatzen dituen erakusle bat eduki nahi dugula. Unioak, hau egiteko mekanismoa ematen digute modu erraz batean. Adibidea:

```
union bat {
   int i;
   double d;
   char * pc;
}
```

  - union bat motako aldagai guztiek, ez dauzkate hiru atal (bata int bestea double eta bestea char *), baizik eta momentu bakoitzean hauetako bakarra edukiko du
  - Izan bedi ondorengo definizioa eta kode zatia.

```
union bat x;
x.i=3;
x.d=8.23;  /*x.i-an idatzi den 3-a galdu da */
```

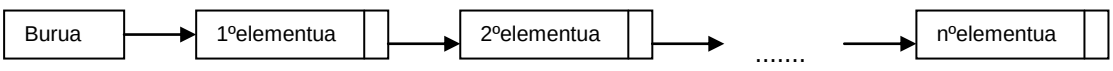
x.i-rekin, bere edukira baina zenbaki oso moduan sartu nahi dugula adierazten dugu.

  - Unio batek, berak bakarrik ez dauka aplikagarritasun handiegirik, ez baitakigu nola interpretatu bere edukia. Soluzio bat, egitura batean sartzea litzateke.

```
struct  zerbait {
    union bat u;
    char mota; //* 'i','d 'edo 'p'balioa hartuko du u atalaren arabera
}
```

3 Zerrenda Kateatuak

Zerrenda kateatuak erakusle bidez erlazionatuta dauden egiturak dira.

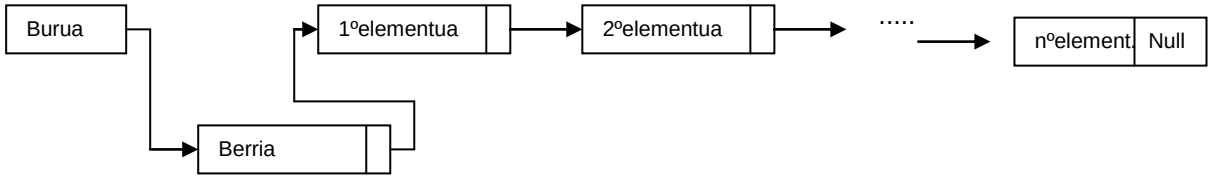


Listako elementu bakoitzak, hurrengo elementua apuntatzen duen erakuslea dauka eta hori dela eta elementu guztiak kateaturik daudela esaten da. Zerrenda guztietan, burua eta kola edo buztana daudela esaten da. Buruak, zerrendako lehenengo elementuaren helbidea gordeko du eta buztanak azken elementura iritsi garela

adieraziko digu. Azkenera iritsi garela adierazteko NULL erabiltzen da, eta honek zerrendan beste elementurik ez dagoela adieraziko digu.

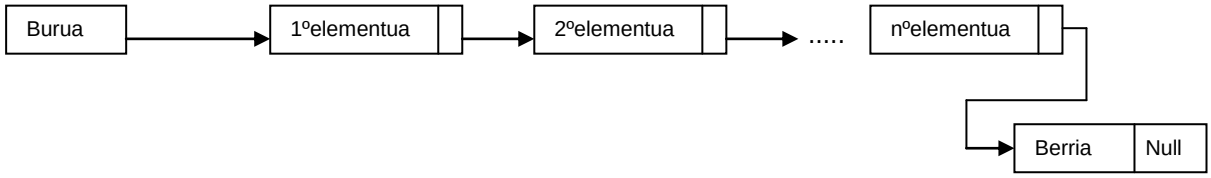
3.1 Zerrenda Kateatuak sortzeko moduak

3.1.1 Buruan sartu elementu berri bat



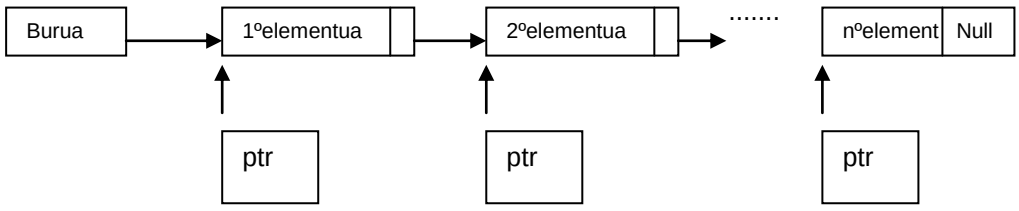

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define LUZERA 30
#define BAI 1
#define EZ 0

void main (void) {
   struct langileak {
      char izena[LUZERA];
      int num;
      struct langileak  *hurrengo_ptr;
   };
   struct langileak *burua_ptr;
   struct langileak *berria_ptr;
   int elementu_berriak=BAI;
   char c,str[4];

   burua_ptr=(struct langileak *)NULL;
   c='s';
   while (elementu_berriak==BAI) {
      berria_ptr=malloc(sizeof(struct langileak));
      if (burua_ptr==NULL) {
         printf("Sartu langile berriaren izena:\n");
         gets(berria_ptr->izena);
         burua_ptr=berria_ptr;
         burua_ptr->hurrengo_ptr=NULL;
      }
      Else {
         printf("Sartu langile berriaren izena:\n");
         gets(berria_ptr->izena);
         berria_ptr->hurrengo_ptr=burua_ptr;
         burua_ptr=berria_ptr;
      }
      printf("Elementu gehiago? (B/E)");
      gets(str);
      sscanf(str,"%c",&c);
      if((c=='E')||(c=='e')) {
           elementu_berriak=EZ;
      }
   }
}
```

3.1.2 Buztanean elementu berri bat sartu



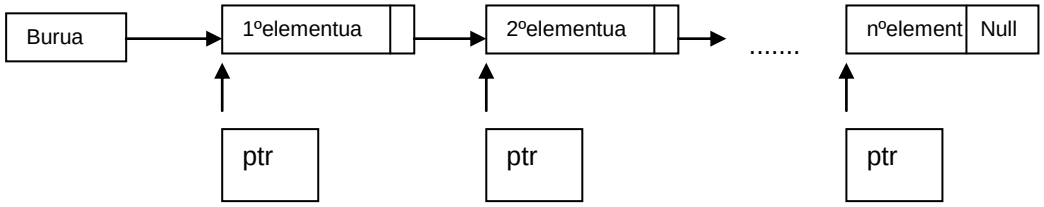

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define LUZERA 30
#define BAI 1
#define EZ 0

int main (void) {
   struct langileak {
      char izena[LUZERA];
      int num;
      struct langileak *hurrengo_ptr;
   };
   struct langileak *burua_ptr, *buztana_ptr, *berria_ptr;
   int elementu_berriak=BAI;
   char c,str[4];

   burua_ptr=NULL;
   c='s';
   while (elementu_berriak==BAI) {
      berria_ptr=malloc(sizeof(struct langileak));
      berria_ptr->hurrengo_ptr=NULL;
      printf("Sartu langile berriaren izena:\n");
      gets(berria_ptr->izena);
      if (burua_ptr==NULL) {
         burua_ptr=berria_ptr;

      }
      Else {
         buztana_ptr->hurrengo_ptr=berria_ptr;
      }
      buztana_ptr=berria_ptr;
      printf("Elementu gehiago? (“B/E)");
      gets(str);
      sscanf(str,"%c",&c);
      if((c=='E')||(c=='e')) {
         elementu_berriak=EZ;
      }
   }
}
```

3.1.3 Zerrenda osoan zehar ibiltzea



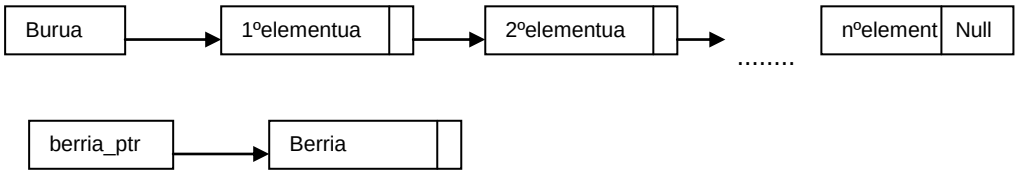
```
    ptr=burua_ptr;
    while(ptr!=NULL) {
       printf(ptr->izena);
       ptr=ptr->hurrengo_ptr;
   }
```

3.1.4 Zerrendan elementuak bilatu




```
ptr=burua_ptr;
bilatua=EZ;

while ((ptr!=NULL)&&(bilatua==EZ)) {
   bilatua= bilatu(elementua,ptr)   ;
   ptr=ptr->hurrengo_ptr;
}
```

3.1.5 Zerrendan elementuak sartu



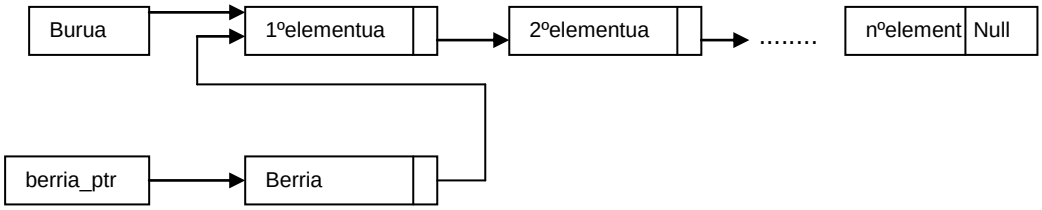
Zerrendan elementuak sartzeko 4 modu desberdin ikusiko ditugu:

- Buruan sartu elementu berria.
- Buztanean sartu elementu berria.
- Elementu berria, beste elementu baten atzean sartu.
- Elementu berria, beste elementu baten aurrean sartu.

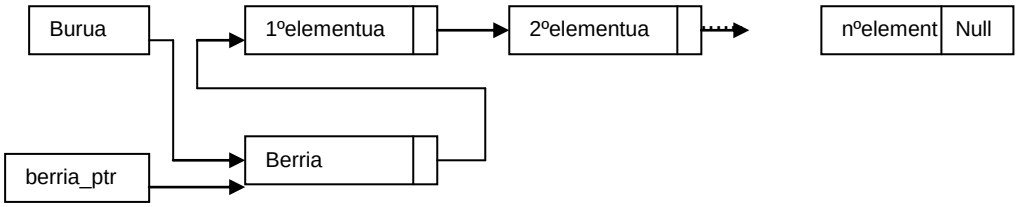
3.1.5.1 Buruan sartu elementu berria

Bi pausu egin beharko dira:

· Elementu berria, zerrendako lehen elementuarekin lotu.



· Buruko erakusleak elementu berriaren erakuslea hartuko du balio bezala.



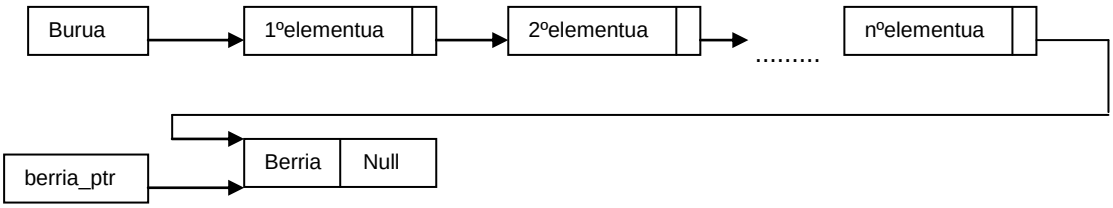

```
berria_ptr->hurrengo_ptr=burua_ptr;
burua_ptr=berria_ptr;
```

3.1.5.2 Buztanean sartu elementu berria

Ondorengo operazioak egin behar ditugu:

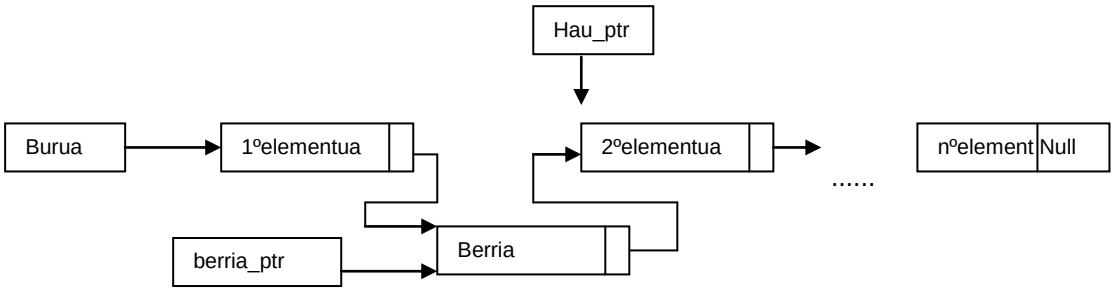
· Zerrendako azken elementua aurkitu

· Elementu berria, azkena aurkitu dugun elementuarekin lotu



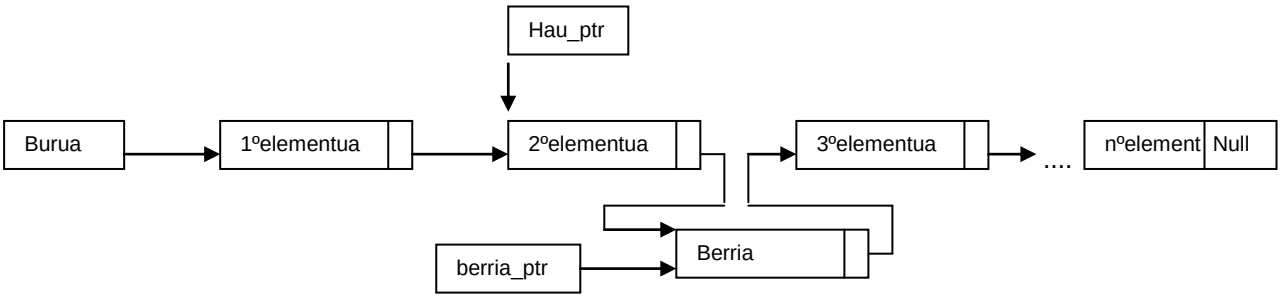

```
ptr=berria_ptr;
while(ptr->hurrengo_ptr!=NULL) {
    ptr=ptr->hurrengo_ptr;
}
ptr-> hurrengo _ptr=berria_ptr;
```

3.1.5.3 Elementu berria, beste elementu baten aurrean sartu




```
ptr=burua_ptr;
while (( ptr!=NULL)&&(ptr->hurrengo_ptr)!=Hau_ptr) {
   ptr=ptr->hurrengo_ptr;
}

if((ptr-> hurrengo_ptr)==Hau_ptr) {
   berria_ptr-> hurrengo_ptr=Hau_ptr;
   ptr-> hurrengo_ptr=berria_ptr;
}
```

3.1.5.4 Elementu berria, beste elementu baten atzean sartu

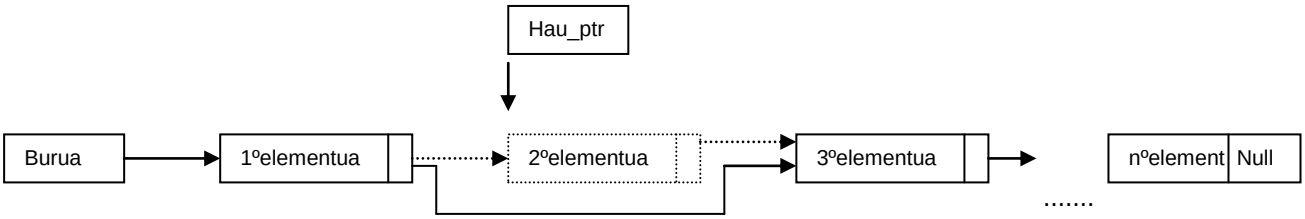



```
ptr=burua_ptr;
while (( ptr!=NULL) && ptr !=Hau_ptr) {
   ptr=ptr-> hurrengo_ptr;
}

if(ptr==Hau_ptr) {
   berria_ptr-> hurrengo_ptr=Hau_ptr-> hurrengo_ptr;
   ptr-> hurrengo_ptr=berria_ptr;
}
```

3.1.6 Zerrendatik elementuak kendu

Erreserbatutako memoria liberatzeko, **free** funtzioa erabiltzen da. Funtzio honek, liberatu behar den memoria apuntatzen duen erakuslea behar du parametro bezala.

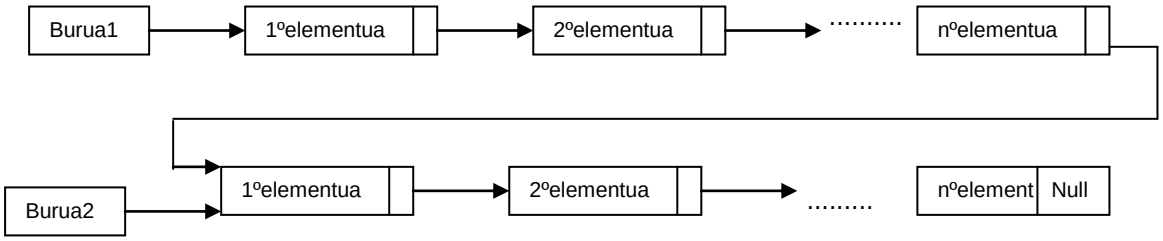
Hori dela eta, zerrendako elementu bat kendu aurretik, lehenengo egin beharreko lana elementu hori zerrendan topatzea da.




```
ptr=burua_ptr;
while(( ptr!=NULL)&&(ptr->hurrengo_ptr)!=Hau_ptr) {
   ptr=ptr-> hurrengo_ptr;
}
if((ptr-> hurrengo_ptr)==Hau_ptr) {
      ptr-> hurrengo_ptr=Hau_ptr-> hurrengo_ptr;
      free (Hau_ptr);
}
```

3.1.7 Bi zerrenda lotzea

Kasu honetan, lehenengo zerrendako azken elementua aurkituko da eta ondoren, azken elementu honek daukan erakuslea, bigarren zerrendako lehenengo elementura apuntatzen jarriko dugu.




```
ptr=burua1_ptr;
while ( ptr-> hurrengo_ptr!=NULL) {
   ptr=ptr-> hurrengo_ptr;
}
ptr-> hurrengo_ptr=burua2_ptr;
```