eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

BILBOKO INGENIARIEN GOI ESKOLA TEKNIKOA

KONPUTAGAILUEN PROGRAMAZIOA
TURBO PASCAL BITARTEZ

IV

EGILEA: Jesus-Mari Romo Uriarte

(hirugarren zirriborroa) 2000-9-20

HITZAURREA

Esku artean duzun liburu honek konputagailuen programazioaren oinarriak aurkezten ditu, helburu nagusi horrekin batera informatikaren hastapeneko kontzeptuak ere lantzen ditu.

Liburu osoan zehar ariketa praktikoei garrantzia handia ematen zaie, batez ere laugarren kapitulutik aurrera, hots, programazioaren gaia hasten denetik aurrera. Esan daiteke kontzeptu bakoitzeko programa bat idatzi dela, programen erabilpena errazago izan dadin kapituluka bildu dira eta kapitulu bakoitzaren amaieran euren identifikazio-taula tartekatu egin da. Programok diskete batean banatzen ditugu horrela eskatuz gero, iturburu-programak direnez exekutatu ahal izateko konpiladoreren bat beharko du ikasleak (guk Borland etxeko Turbo Pascal 7.0 konpiladorea erabili dugu).

Liburuak 14 kapitulu ditu, baina bigarren zirriborro hau amaitzen dugun une honetan lehen hamabiak idatzirik ditugu, gainerakoak hurrengo baterako utzi ditugularik.

14 kapituluen kontzeptuak jasotzeko ikasleak 12 kreditu beharko lituzke, euretatik 4 kreditu gutxienez konputagailuaren aurrean era praktiko batean kurtsatuko lituzke. 12 kreditu horiek bi ikasturtetan banaturik egonez gero, hona hemen proposatzen dugun jokabidea:

Ikasturtea	***Kredituak***	***Kapituluak***
1	6	1, 2, 3, 4, 5, 6, 8, 9, 10, 11
2	6	7, 12, 13, 14

Bilboko Ingeniarien Goi Eskola Teknikoan, Industri Ingeniarien titulazioan erabiltzen da liburu hau eta bertan 9 kreditu ditugu une honetan konputagailuei buruzko kontzeptu guztiak eman ahal izateko. Gauzak horrela, egun prestaturik ditugun lehen hamabi kapituluak jorratzeko denborarik ez dugulako zazpigarrena, unitateak lantzen dituena alegia, sakontasunik gabe gainbegiratzen dugu.

JM Romo Uriarte

NON ZER

11. ATALA: RECORD ETA SET DATU-MOTAK

AURKIBIDEA

11.1 SARRERA

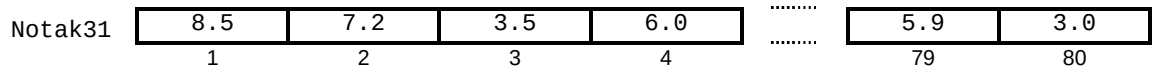
Datu-mota egituratuekin jarraituz kapitulu honetan Record eta Set datu-motak ikasiko ditugu. Record datu-motak Set datu-motak baino garrantzi gehiago duela programazioan zalantzarik ez dago, horregatik datu-mota bakoitzari dagokion orrialde eta adibide kopurua ez da berdina (ikus aurkibidea, non irakurleak datu-mota biren artean oreka falta berehala nabarituko duen).

Hamargarren kapituluak arrayaren kontzeptua gureganatzeko balio izan digu, eta array datu-motarako honako definizio hau eman genuen: datu-mota estatikoa den arrayak dituen elementuak *mota berekoak dira* eta indize-zerrenda baten bitartez identifikatzen dira, arrayaren elementu kopurua finitua da eta memoriaren gelasketan biltegitzean hurrenez hurren kokatzen dira.

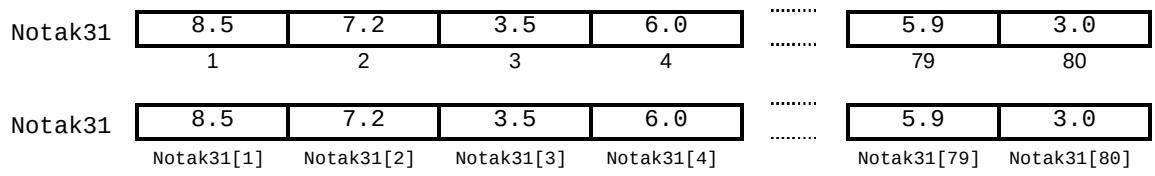
Arrayaren definizioan, une honetan, elementu guztiak mota berekoak izango direla nabarmendu dugu. Hots, array batek ezin dezakeela adibidez kopuruak (Integer, Real, ...) eta esaldiak (String) aldi berean gorde, arrayaren oinarrizko datu mota bat izango delako. Horregatik array datu-mota bat deklaratzean, besteak beste, elementuei dagokien datu-mota bat eta bakarra explizitoki jarriko da. Adibidez, demagun `DM_Notak` datu-motako arrayetan zenbaki errealak gordeko direla eta gehienez 80 zenbaki izango direla, hona hemen `DM_Notak` datu-mota egituratuari dagokion erazagupena:

```
TYPE
   DM_Notak = ARRAY[1..80] OF Real ;
VAR
   Notak31 : DM_Notak ;
```

Ikusten denez arrayen datu-mota den `DM_Notak` deklaratzean arrayaren oinarrizko elementuen datu-mota zein izango den finkatu egingo dela, kontura gaitezen arraya finitua izango dela adierazten duten bi indize aukeratu beharra dagoelako. `DM_Notak` datu-mota aintzat harturik, aldagaien erazagupenaren atalean `Notak31` arraya programan erabiliko dela deklaratzen da, `Notak31` aldagai horretan balioak gorde ondoren euren eskema eginez gero honako zerbait izango genuke:



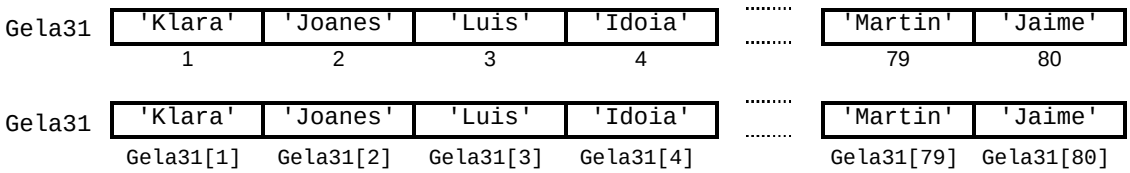
Ikusten denez `Notak31` aldagai egituratuak zenbaki errealen zerrenda bat biltegitzeko ahalmena du, ikusten da halaber egitura homogenoak direla eta arrayaren partaide bakoitzak berezko identifikadorea duela (indizearen araberako identifikadorea hain zuzen):



Baina kontutan izan dezagun notak ikasleei dagozkieela eta euren izenen zerrenda beste bigarren array baten bitartez gauzatu beharko dela. Honelaxe:

```
TYPE
   DM_Izena = String[39] OF DM_Izena ;
   DM_Taldea = ARRAY[1..80] OF DM_Izena ;
VAR
   Gela31 : DM_Taldea ;
```

Izenen zerrenda bat memorian biltegitzeko `Gela31` arraya deklaratzen dugu zein `DM_Taldea` datu-motakoa den. Bigarren array honen edukia honako zerbait izan daiteke:



Jarraian ematen den `ArrayParaleloakLantzen` izeneko programan `Notak31` eta `Gela31` arrayak definitu ondoren balioak gordetzen dira bertan, kontsideratuz arrayak paraleloak direla, hots, `Indz` indize bera daukaten bi elementuak ikasle jakin baten informazio osagarriak direla (`Gela31[Indz]` karaktere-katea `Indz`-garren ikaslearen izena litzateke, eta `Notak31[Indz]` kopurua `Indz`-garren ikasle beraren nota). Programa amaitu aurretik arrayen edukiak pantailaratzen dira.

`ArrayParaleloakLantzen` programan ikasle baten informazioa lantzeko `Notak31` eta `Gela31` arrayen posizio bera daukaten bi elementuak prozesatu behar dira uneoro. Hau da, array biek daukaten informazioa ordenez antolaturik dago, beste modu batez esanda `Notak31` eta `Gela31` arrayak paraleloak dira (horregatik luzera logikoa zehazten duen aldagaia bakarra izango da).

Hau da `ArrayParaleloakLantzen` izeneko programaren kodea:

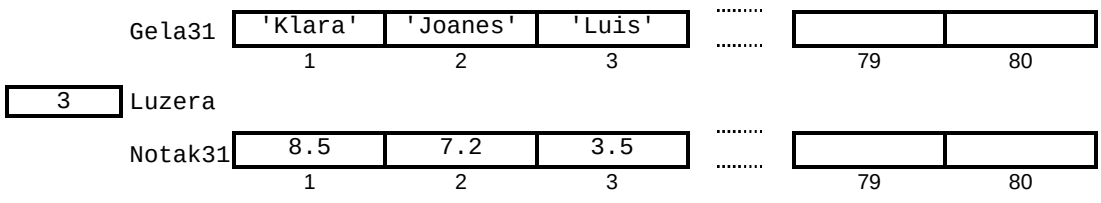
```
PROGRAM ArrayParaleloakLantzen ;                { \TP70\11\RECORD01.PAS }
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 80 ;
TYPE
  DM_Izena = String[39] ;
  DM_Taldea = ARRAY[BEHEMUGA..GOIMUGA] OF DM_Izena ;
  DM_Notak = ARRAY[BEHEMUGA..GOIMUGA] OF Real ;

PROCEDURE ArrayBiakBete (VAR Gela : DM_Taldea;
                         VAR Notak : DM_Notak;
                         VAR LuzLog : Byte) ;
VAR
  Indize : Byte ;
  Erantz : Char ;
BEGIN
   Erantz := 'B' ;
   Indize := 1 ;
   WHILE (Indize <= GOIMUGA) AND (Erantz = 'B') DO
   BEGIN
      Write (Indize, '. ikaslearen izena eman: ') ;
      ReadLn (Gela[Indize]) ;
      Write (Indize, '. ikaslearen nota eman:  ') ;
      ReadLn (Notak[Indize]) ;
      REPEAT
         Write ('Ikasle gehiagorik? (B/E): ') ;
         ReadLn (Erantz) ;
         Erantz := UpCase (Erantz) ;
      UNTIL (Erantz = 'B') OR (Erantz = 'E') ;
      IF Erantz = 'B' THEN
        Indize := Indize + 1 ;
   END ;

   IF Indize > GOIMUGA THEN
     LuzLog := GOIMUGA
   ELSE
     LuzLog := Indize ;
END;
```

```
PROCEDURE ArrayBiakIkus (CONST Gela : DM_Taldea;
                         CONST Notak : DM_Notak;
                         LuzLog : Byte) ;
VAR
  Indize : Byte ;
BEGIN
   FOR Indize:=BEHEMUGA TO LuzLog DO
   BEGIN
      Write (Indize, '. ikaslea: ') ;
      Write (Gela[Indize]:15, '   ') ;
      WriteLn (Notak[Indize]:10:1) ;
   END ;
END;

VAR
  Gela31 : DM_Taldea ;
  Notak31 : DM_Notak ;
  Luzera : Byte ;
BEGIN
   WriteLn ('Izenak eta notak memorian gorde:') ;
   ArrayBiakBete (Gela31, Notak31, Luzera) ;
   WriteLn ;
   WriteLn ('Hona hemen ikasle bakoitzeko gordetako informazioa:') ;
   ArrayBiakIkus (Gela31, Notak31, Luzera) ;
END.
```

Hauxe litzateke `ArrayParaleloakLantzen` programaren irteera posible bat:

```
Izenak eta notak memorian gorde:
1. ikaslearen izena eman: Klara
1. ikaslearen nota eman:  8.5
Ikasle gehiagorik? (B/E): b
2. ikaslearen izena eman: Joanes
2. ikaslearen nota eman:  7.2
Ikasle gehiagorik? (B/E): b
3. ikaslearen izena eman: Luis
3. ikaslearen nota eman:  3.5
Ikasle gehiagorik? (B/E): e

Hona hemen ikasle bakoitzeko gordetako informazioa:
1. ikaslea:           Klara          8.5
2. ikaslea:          Joanes          7.2
3. ikaslea:            Luis          3.5
_
```

Eskematikoki:



Array paraleloen prozesaketa erraza eta zehatza dela onar daiteke eta programaren exekuzio-adibidea azterturik aplikazioaren operadorea ez da konturatzen barneko antolaketa zein den. Baina soluzio honek ez du “ikasle” kontzeptua bere osotasunean aintzat hartzen, izan ere ikasle jakin bati dagokion informazioa bi egituretan barreiaturik dago. Esan dugun bezala ikasle bat `Gela31[Indz]` eta `Notak31[Indz]` bikotearen bitartez osatzen da.

Ikasle baten datuak erlazionaturik daudenez, eta askotan osotasunean tratatu behar direnez (ikasle baten informazioa zerrendara gehitu, ikasle baten informazioa zerrendatik kendu, zerrenda irizpide baten arabera ordenatu, ...), datu-mota ezberdineko informazioak biltegitzen duen egituraren bat lengoaiak eskaintzea komenigarria litzateke.

11.2 RECORD DATU-MOTA

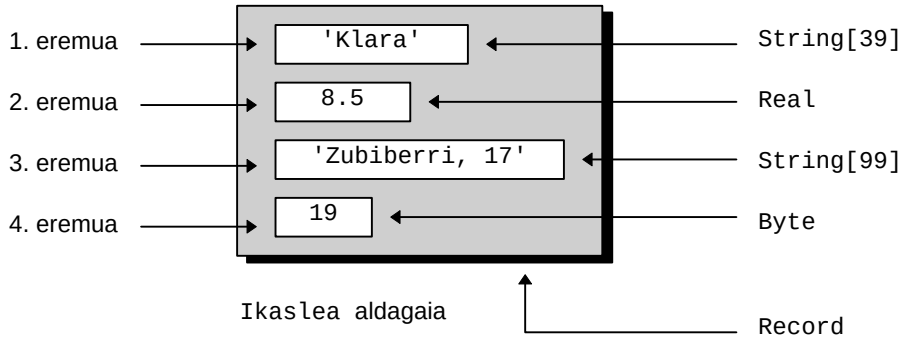
Informazio heterogeneoak egituratzeko Pascal lengoaiak Record datu-mota dauka aurredefiniturik, ikus dezagun zertan den Record edo *erregistro* datu-mota hau.

11.2.1 Definizioa

Record edo erregistro datu-motaren definizioa eman dezagun. Datu-egitura estatikoa[1] den erregistroak dituen elementuak *eremuak* deitzen dira eta datu-mota ezberdinak izan daitezke elkarrekiko, erregistroaren elementu kopurua finitua da eta eremu bakoitzaren atzipena identifikadore baten bitartez lortzen da.

Demagun aurreko adibideko ikaslearen kontzeptuari (izena eta nota propietateak dituen kontzeptuari) beste bi ezaugarri gehitzen dizkiogula adina eta helbidea, ondorioz array paraleloekin lan egitean lau array beharko genituzke, bat propietate bakoitzeko. Array paraleloen bide hori antzua delakoan baztertuko dugu eta informazio guztia bilduko duen array bakar bat antolatuko dugu, baina horretarako ezer baino lehen lau propietateak elkar gordetzen dituen egitura behar dugu. Argi esanda “ikasle” kontzeptua behar dugu definitu, eta ondoren “ikastaldea” aurrekoan oinarrituko den array lineal bat litzateke.

Aipaturiko lau propietateek datu-mota bera balute, “ikasle” kontzeptua gauzatzeko orduan soluzio bat eurekin array bat osatzea litzateke (lau osagai dituen array lineala). Baina tamalez behar dugun ”ikasle”-aren ezaugarriak homogeneoak ez dira, izan ere `Ikaslea` deituko genukeen aldagai bakarrak honela irudi daiteke:



`Ikaslea` aldagaia erregistro bat izango delako `Record` datu-mota berriaren hitz erreserbatua beharko dugu erabili, eta horrekin batera erregistroaren osagaiak diren eremuak definituko ditugu. Hona hemen, adibide horretarako Pascal lengoaiak darabilen sintaxia, ikusten denez eremu bakoitzeko programadoreak izen[2] bat aukeratzen du eta bere datu-mota zehaztu egiten da:

[1] Datu-egitura dinamikoak erakusle kontzeptuarekin elkarturik daude eta 13. kapituluan aztertzeko parada izango dugu.

[2] Izen hori hautatzeko Turbo Pascal lengoaiak identifikadoreetarako jartzen dituen baldintzak bete behar dira, gogoratu **4.2.2.2 Erabiltzailearen identifikadoreak** izenburuko puntua.

```
TYPE
   DM_Ikaslea = RECORD
                  Izena : String[39] ;
                  Nota : Real ;
                  Helbidea : String[99] ;
                  Adina : Byte ;
                END ;
VAR
   Ikaslea : DM_Ikaslea ;
```

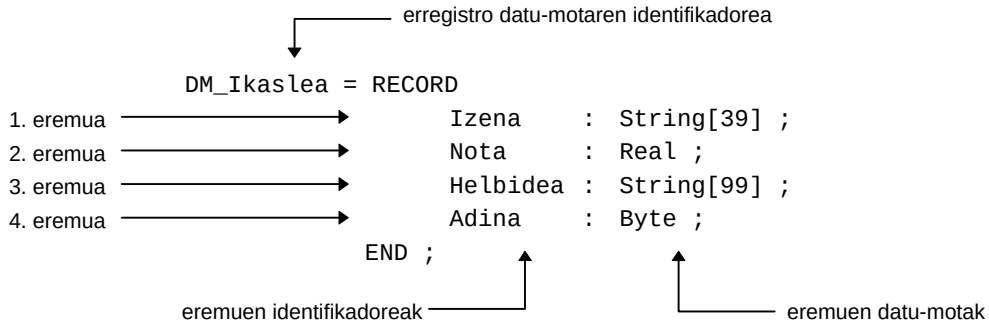
Behin `DM_Ikaslea` datu-mota berria erazagutu ondoren `Ikaslea` izeneko erregistro aldagai bat deklara daiteke, non pertsona bakar bati dagokion informazioa gorde daitekeen. Informazioaren antolaketa hierarkiakoa denez, ikasle multzo baten “gela” kontzeptua ere erraz gauzatuko litzateke, horretarako `DM_Ikaslea` datu-motan oinarritzen den array bat asma daiteke (**11.2.5 Erregistroen arrayak** puntuan ikusi eta ikasiko dugu).

11.2.2 Eremuak

Erregistro baten irudia eta definizioa ikusi ondoren erregistroaren osagaiak diren eremuei buruz hitz egin dezagun, lehenik sintaxia aipatuz eta ondoren eremuen helburua zein den argituz eta eremuen informazioak memorian nola biltegitzen diren azalduz.

11.2.2.1 Eremuak zehazteko sintaxia

Eremuak nola zehazten diren ikasteko `Record` datu-mota bat definitzeko behar dugun sintxaia aztertuko dugu. Aurreko `DM_Ikaslea` datu-motaren adibidearekin jarraituz horra sintxiaren ezaugarririk aipagarrienak:



- Definitu nahi den erregistro datu-motaren identifikadorea programadoreak berak aukeratzen du. Etiketa hori datu-mota bati dagokiola argi eta nabarmen gera dadin markaren bat gehitzea gomendatzen da, esaterako adibidean ikaslearen eredua den erregistro datu-motari `DM_` karaktereak jarri dizkiogu `DM_Ikaslea` geratuz.

- Definitu nahi den datu-mota erregistro bat izango dela adierazteko `Record` hitz erreserbatua idatzi behar da, eta bere aurrean `=` sinboloa jarri behar da (ez ahaztu `TYPE` bloke barnean aurkitzen garela).

- Eremuak bat baino gehiago izatea normalena delako, eremu guztiak elkartu egin behar dira. Hau da, erregistroaren mugak zehaztu behar zaizkio konpiladoreari, horretarako `RECORD` eta `END` hitz erreserbatuak erabiltzen dira, ez da `BEGIN`-ik jartzen.

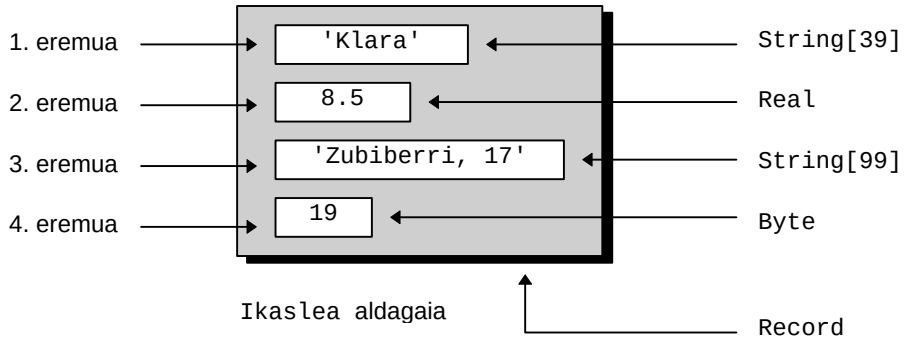
- `RECORD` eta `END` hitz erreserbatuen artean eremuen zerrenda editatzen da, eremu bakoitzeko bi kontzeptu adieraziz: eremuaren identifikadorea eta eremuari dagokion datu-mota.

- Eremu bakoitzak bere identifikadorea du ere eta, lehen bezala, programadoreak berak aukeratzen ditu. Eremuen etiketak ez dira datu-motak, aldagaiak ere ez, ezerekin konparatzekotan arrayen indizeekin[3] alderatuko genituzke.

- Eremu bakoitzak informazioren bat gordeko duenez datu-motaren bat izango du loturik, hori adierazteko baliagarria den datu-mota idazten da `:` eta `;` sinboloen artean.

11.2.2.2 Eremuen helburua

Demagun `DM_Ikaslea` deituriko datu-motak ondoko lau eremuak biltzen dituen erregistro bat dela, eta `Ikaslea` izeneko aldagaian irudiko informazioa gorde nahi dela:



Programaren hasieran `DM_Ikaslea` datu-mota definituko denez ezaguna egingo zaigu zalantzarik gabe, non dago `ErregistroaDatuzBetetzen` programaren ekarpen berria?

```
PROGRAM ErregistroaDatuzBetetzen ;                { \TP70\11\RECORD02.PAS }
TYPE
  DM_Ikaslea = RECORD
                  Izena : String[39] ;
                  Nota : Real ;
                  Helbidea : String[99] ;
                  Adina : Byte ;
               END ;
VAR
  Ikaslea : DM_Ikaslea ;
BEGIN
   Write ('Ikaslearen izena eman:    ') ;      { Datuak biltegitzen }
   ReadLn (Ikaslea.Izena) ;

   Write ('Ikaslearen nota eman:     ') ;
   ReadLn (Ikaslea.Nota) ;

   Write ('Ikaslearen helbidea eman: ') ;
   ReadLn (Ikaslea.Helbidea) ;

   Write ('Ikaslearen urteak eman:   ') ;
   ReadLn (Ikaslea.Adina) ;
```

[3] Izan ere, eremu baten identifikadorearen zeregina (array linealen indizearen bezala) erregistroaren osagai horren atzipena ahalbidetzea da.

```
{ Datuak memoriatik hartuz pantailaratzen }
   WriteLn ;
   WriteLn ;
   WriteLn ('Hona hemen ikaslearen datuak:') ;

   WriteLn ('Izena---------> ', Ikaslea.Izena) ;
   WriteLn ('Nota----------> ', Ikaslea.Nota:0:1) ;
   WriteLn ('Helbidea------> ', Ikaslea.Helbidea) ;
   WriteLn ('Adina---------> ', Ikaslea.Adina) ;
END.
```

`ErregistroaDatuzBetetzen` programaren bitartez `Ikaslea` izeneko aldagai batean bererai dagozkion informazio guztiak gordetzen dira. Informazioak mota ezberdinekoak direlako `Ikaslea` aldagaia erregistro bat izango da (lau eremuko erregistroa hain zuzen). Eremuen identifikadoreak eta euren datu-motak ezagunak dira programadorearentzat, eta horien bitartez `Ikaslea` izeneko aldagaiak bi string zenbaki erreal bat eta kopuru oso bat gordetzeko ahalmena duela kontura gaitezke. Baina nola?

Eremu batean informazioa gordetzeko kanpotik barnera abiatuko gara. Hots, lehenik erregistro aldagaia identifikatu beharko da (izan ere `ErregistroaDatuzBetetzen` programan `Ikaslea` izeneko aldagai bakarraren ordez bi erregistro egon zitezkeen, adibidez `Ikasle_1` eta `Ikasle_2` izeneko aldagaiak); horregatik eremu batean daturen bat gordetzeko unean honela egiten da, non lehenik erregistroa zehazten den:

```
ReadLn (Ikaslea          ) ;
```

`Ikaslea` erregistroa identifikatu ondoren barnerago joko dugu, lau eremuetatik irakurri nahi den eremua zehaztu egingo dugu (esate baterako lehendabiziko eremuari dagokion `Izena` identifikadorea jarriz). Baina, eta hauxe litzateke berrikuntzarik aipagarriena, erregistro eta eremuaren identifikadore horien artean banatzaile bat jarri beharra dago. Pascal lengoaiak identifikadoreen arteko banaketa burutzeko puntu bat darabil:

```
ReadLn (Ikaslea.Izena) ;
```

Laburtuz, egituratuak diren aldagaiek euren elementuen erreferentziak egiteko bideak definiturik dituzte, adibidez string edo array baten oinarrizko osagairen bat zehazteko `[indizea]` ezarri behar zaio aldagai egituratuari, baina erregistroak homogeneoak ez direla bestelako mekanismoaz horniturik daude (puntu notazioarena hain zuzen ere).

11.2.2.3 Eremuen biltegitzea memorian

Erregistro datu-motaren eremuak elkar ondoan kokatzen dira memorian. Honezkero erregistro jakin baten eremu bakoitzari dagokion helbidea aurrekoaren jarraian datorrena izango dela froga daiteke. Ikus `ErregistroarenEremuakMemorian` programa non erregistro aldagai bat definitzen den. `ErregistroarenEremuakMemorian` programa honek `Ikus` izeneko unitate bat darabil eta bertan `IntToHex()` izeneko funtzioa garatu izan da.

`ErregistroarenEremuakMemorian` programa egikaritzean memoriaren helbideak erakusten dira (pila non hasten den, datu-segmentuaren hasiera eta kode-segmentuarena), gero `Langile` aldagaiaren ezaugarriak datoz (Identifikadorea, helbidea eta tamaina), eta azkenik `Langile` aldagaiaren zazpi eremuen informazioak.

Hona hemen `ErregistroarenEremuakMemorian` programa:

```
PROGRAM ErregistroarenEremuakMemorian ;                    { \TP70\11\RECORD03.PAS }
USES
    Crt, Ikus ;      (* IKUS.TPU  unitatea darabil *)
TYPE
  DM_Langile = RECORD
                  Izena     : String[39] ;
                  Soldata   : Real ;
                  Helbidea  : String[99] ;
                  Adina     : Byte ;
                  Kategoria : Char ;
                  Gizonezko : Boolean ;
                  EgoeraZib : Char ;
               END ;
VAR
  Langile : DM_Langile ;
BEGIN             (* Programa Nagusia *)
   ClrScr ;

   WriteLn ('ERREGISTRO BATEN EREMUEN MEMORI HELBIDEAK') ;
   WriteLn ('========================================') ;
   WriteLn ;
   WriteLn ('PILARI dagokion segmentuaren hasiera:  ',IntToHex (sSeg)) ;
   WriteLn ('DATUEI dagokien segmentuaren hasiera:  ',IntToHex (dSeg)) ;
   WriteLn ('KODEARI dagokion segmentuaren hasiera: ',IntToHex (cSeg)) ;
   WriteLn ;
   WriteLn ;
   WriteLn ('ALDAGAIA':29, '       HELBIDEA', '   TAMAINA') ;
   WriteLn ('-----------------------------------------':60) ;

   WriteLn ('Langile ----> ':36, IntToHex (Seg (Langile)),':',
            IntToHex (Ofs (Langile)), '   (', SizeOf (Langile), ')') ;
   WriteLn ;
   WriteLn ;

   WriteLn ('Langile.EgoeraZib ----> ':36, IntToHex (Seg (Langile.EgoeraZib)),':',
            IntToHex (Ofs (Langile.EgoeraZib)), '   (', SizeOf (Langile.EgoeraZib), ')') ;

   WriteLn ('Langile.Gizonezko ----> ':36, IntToHex (Seg (Langile.Gizonezko)),':',
            IntToHex (Ofs (Langile.Gizonezko)), '   (', SizeOf (Langile.Gizonezko), ')') ;

   WriteLn ('Langile.Kategoria ----> ':36, IntToHex (Seg (Langile.Kategoria)),':',
            IntToHex (Ofs (Langile.Kategoria)), '   (', SizeOf (Langile.Kategoria), ')') ;

   WriteLn ('Langile.Adina ----> ':36, IntToHex (Seg (Langile.Adina)),':',
            IntToHex (Ofs (Langile.Adina)), '   (', SizeOf (Langile.Adina), ')') ;

   WriteLn ('Langile.Helbidea ----> ':36, IntToHex (Seg (Langile.Helbidea)),':',
            IntToHex (Ofs (Langile.Helbidea)), '   (', SizeOf (Langile.Helbidea), ')') ;

   WriteLn ('Langile.Soldata ----> ':36, IntToHex (Seg (Langile.Soldata)),':',
            IntToHex (Ofs (Langile.Soldata)), '   (', SizeOf (Langile.Soldata), ')') ;

   WriteLn ('Langile.Izena ----> ':36, IntToHex (Seg (Langile.Izena)),':',
            IntToHex (Ofs (Langile.Izena)), '   (', SizeOf (Langile.Izena), ')') ;
END.
```

`ErregistroarenEremuakMemorian` programa egikaritu eta bere irteera aztertzean `Langile` erregistroa `5A39` helbidea duen segmentuan aurkituko direla ikusten da (ordenadore batetik bestera helbide fisiko hori ezberdina izan daiteke). Zehatzago, `Langile` aldagaia `5A39:005E` helbidean hasten da (`Langile` erregistroa lehen elementua, lehen eremua, `5A39:005E` helbide horretan hasten da), eta gainerako beste elementuak, eremuak, jarraian aurkitzen dira.

`Langile` aldagaia, bere osotasunean harturik, `5A39:005E` helbidean hasten dela esan dugu eta dagokion tamaina 150 byte dira. Erregistro horren tamaina bere eremuen tamainen batura denez honela kalkulatuko litzateke: 150 = 40 + 6 + 100 + 1 + 1 + 1 + 1

`ErregistroarenEremuakMemorian` programa exekutatu ondoren, ordenadore zehatz batean irteera hau dela ziurtatzen badigute, helbide horien interpretazioa egin dezagun, jakinik helbideak bi zatitan ematen direla (segmentua eta desplazamendua):

```
ERREGISTRO BATEN EREMUEN MEMORI HELBIDEAK
=========================================

PILARI dagokion segmentuaren hasiera:  5A70
DATUEI dagokien segmentuaren hasiera:  5A39
KODEARI dagokion segmentuaren hasiera: 5947


                    ALDAGAIA       HELBIDEA   TAMAINA
              ---------------------------------------------
                     Langile ----> 5A39:005E   (150)


           Langile.EgoeraZib ----> 5A39:00F3   (1)
           Langile.Gizonezko ----> 5A39:00F2   (1)
           Langile.Kategoria ----> 5A39:00F1   (1)
               Langile.Adina ----> 5A39:00F0   (1)
            Langile.Helbidea ----> 5A39:008C   (100)
             Langile.Soldata ----> 5A39:0086   (6)
               Langile.Izena ----> 5A39:005E   (40)
_
```

Irteeran ikus daitekeenez lehenago definiturik dauden eremuek dituzten helbideak gerorago definituak dauden eremeun helbideak baino baxuagoak dira. Horrela, `Izena` eremuaren helbidea `5A39:005E` helbidea baxuagoa da `Soldata` eremuaren `5A39:0086` helbidearekin konparatzen badugu.

Irteeraren azterketatik ateratzen den beste ondorio bat eremu guztiak datu-segmentuan aurkitzen direla, zazpi eremuei dagozkien helbideak `5A39`-z hasten baitira. Izan ere eremuok programa nagusiko aldagai baten osagaiak dira eta programa nagusiko aldagai guztiak datu-segmentuan kokatzen dira.

Azter dezagun orain eremuen desplazamenduak datu-segmentu barruan. Lehen eremua `Langile.Izena` izanik `5A39:005E` helbidean hasten da, memoria 40 byte hartuz (39 karaktereen 39 byteak gehi 0 posizioaren bytea). Hori dela eta, `Langile.Izena` eremuaren azken memoria posizioa `5A39:0085` izango da:

```
    005E / 005F / 0060 - 006F / 0070 - 007F / 0080 / 0081 / 0082 / 0083 / 0084 / 0085
40 byte = 1 + 1    +      16      +      16     + 1  +   1  +  1  +   1  +  1  +   1
```

`Langile.Soldata` eremua `5A39:0086` posizioan hasten da eta zenbaki erreal bat delako 6 byte izango ditu, horregatik bere azken memoria posizioa `5A39:008B` izango da:

```
          0086 / 0087 / 0088 / 0089 / 008A / 008B
6 byte =   1   +   1   +   1   +   1   +   1   +   1
```

`Langile.Helbidea` izeneko eremua string bat da ere, baina 99 karakteretakoa. Honek 100 byte hartuko ditu memorian (`5A39:008C` posiziotik `5A39:009F` posiziora):

```
008C - 008F      4 byte
0090 - 009F     16 byte
00A0 - 00AF     16 byte
00B0 - 00BF     16 byte
00C0 - 00CF     16 byte
00D0 - 00DF     16 byte
00E0 - 00EF     16 byte
                ---------
                100 byte
```

Azken lau eremuek byte bana hartzen dute memorian, euren oinarrizko datu-motak `Char`, `Boolean` eta `Byte` baitira. Bakoitzaren kokamen zehatza irteeraren pantailaraketan ikus daiteke:

```
Langile.Adina       5A39:00F0    1 byte
Langile.Kategoria   5A39:00F1    1 byte
Langile.Gizonezko   5A39:00F2    1 byte
Langile.EgoeraZib   5A39:00F3    1 byte
```

11.2.3 Eragiketak erregistroekin

Hurrengo bi puntuetan erregistroek onartzen duten eragiketa bakarra eta erregistroak azpiprogrametan aipatuko ditugu.

11.2.3.1 Erregistroa eragigai bezala

Erregistroek onartzen duten eragiketa bakarra esleipena da. Hau da, oinarritzat datu-mota bera duten bi erregistroen arteko asignazioa egin daiteke:

```
PROGRAM ErregistroenEsleipena ;                    { \TP70\11\RECORD04.PAS }
TYPE
  DM_Ikaslea = RECORD
                 Izena    : String[39] ;
                 Helbidea : String[99] ;
                 Adina    : Byte ;
               END ;
VAR
  Ikaslea, Pertsona : DM_Ikaslea ;
BEGIN
   Write ('Ikaslearen izena eman:    ') ;      { Datuak "Ikaslea"n gordetzen }
   ReadLn (Ikaslea.Izena) ;

   Write ('Ikaslearen helbidea eman: ') ;
   ReadLn (Ikaslea.Helbidea) ;

   Write ('Ikaslearen urteak eman:   ') ;
   ReadLn (Ikaslea.Adina) ;

   { Irakurriko datuak beste erregistro bati esleitu }

   Pertsona := Ikaslea ;

   WriteLn ;
   WriteLn ;
   WriteLn ('Hona hemen "Pertsona" aldagaiaren datuak:') ;

   WriteLn ('Pertsona.Izena---------> ', Pertsona.Izena) ;
   WriteLn ('Pertsona.Helbidea------> ', Pertsona.Helbidea) ;
   WriteLn ('Pertsona.Adina---------> ', Pertsona.Adina) ;
END.
```

`ErregistroenEsleipena` programan erregistro baten datuak teklatuz irakurri ondoren beste erregistro bati transferi daitezke. Teklatuaren bitarteko irakurketak eremuka egin beharko dira, baina erregistro biren arteko asignazioa eremuka egiten ibili ordez orokorki egitea posible da.

Demagun `ErregistroenEsleipena` programan aldaketa txiki hau idazten dugula, non `Ikaslea` eta `Pertsona` erregistro aldagaiak datu-mota berdintsua dutela; berdintsua baina ez datu-mota bera. Zer gertatuko litzateke erregistroen arteko esleipena burutzean?.

```
PROGRAM ErregistroenEsleipenOkerra ;                { \TP70\11\RECORD05.PAS }
TYPE
  DM_Ikasle1 = RECORD
                  Izena    : String[39] ;
                  Helbidea : String[99] ;
                  Adina    : Byte ;
               END ;
  DM_Ikasle2 = RECORD
                  Izena    : String[39] ;
                  Helbidea : String[99] ;
                  Adina    : Byte ;
               END ;
VAR
  Ikaslea  : DM_Ikasle1 ;
  Pertsona : DM_Ikasle2 ;
BEGIN
   Write ('Ikaslearen izena eman:    ') ;      { Datuak "Ikaslea"n biltegitzen }
   ReadLn (Ikaslea.Izena) ;
```

`ErregistroenEsleipenOkerra` programa konpilatzean errorea detektatuko da, `Ikaslea` eta `Pertsona` erregistroen arteko esleipena ezinezkoa baita, honelako mezu aski ezaguna erakutsiz: `Error 26: Type mismatch.`

Esan den bezala erregistroek onartzen duten eragiketa bakarra asignazioa da. Hori dela eta, ezin daiteke erregistroen arteko konparaziorik egin. Horregatik erregistro bik informazio berdina gordetzen dutela frogatzeko eremuka lan egingo da (ikus hurrengo puntuan, helburu horrekin, garatu den funtzio boolearra).

11.2.3.2 Erregistroa parametro bezala

Seigarren kapituluan azaldu zen **6.3.3 Parametroen erabilpena Turbo Pascal lengoaian** izeneko puntua eta bere menpekoak (**6.3.3.1 Baliozko parametroa**, **6.3.3.2 Aldagai-parametroa** eta **6.3.3.3 Konstante-parametroa**) gogoratzea komeniko litzateke.

Programa baten moduluen arteko komunikazioa parametroen bitartez lortzen da, hots azpiprograma batek informazioa jasotzeko eta itzultzeko parametroak beharrezkoak dira. Parametroen pasatze moduak funtsean bi dira eta euren taula osatzeko menpeko moduluak, azpiprogramak, zer jasotzen duen kontutan izan behar da.

Menpeko moduluak definiturik daukan parametro formalak, modulu deitzailearen datuarengandik, zer jasotzen duen horrek parametro pasatze moduak desberdintzen ditu, hau da, parametroak balioak ala erreferentziak[4] izan daitezke, zein jarraian luzatzen den taularen zutabeak bereizteko irizpide nagusia den. Lehenengo zutabean baliozko parametroa darabilen parametro pasatze modua azaltzen da lerroetan ondoko bost ezaugarriak kontsideratuz: uneko argumentua, ikur markatzailea, komunikazio mota, parametro formalaren izaera eta parametro errealaren babes maila. Baina parametroa beste aldagai baten erreferentzia edo helbidea denean parametro pasatze modu bi bereizten dira aldagai-parametroa eta konstante-parametroa, azken honek dituen propietateak aproposak dira datu-mota egituratuak azpiprogrametan erabiltzeko.

[4] Memori posizioen helbideak.

Ikus jarraian ematen den taula:

	Baliozko parametroak	Erreferentziazko parametroak	
		Aldagai-parametroak	Konstante-parametroak
Deia	Espresioa	Aldagaia	Espresioa
Markatzailea	~~VAR~~	VAR	CONST
Komunikazioa	Sarrera	Sarrera / Irteera Irteera	Sarrera
Param. formala	Balio bat	Helbide bat	Helbide bat
Param. erreala	Babesturik	Babestu gabe	Babesturik

Seigarren kapituluan esan genuen erreferentziaz pasatutako parametro baten eta bere uneko parametroaren memori posizioak berdinak direla, **6.4.2 Aldagai-parametroak eta memori helbideak** izeneko puntuan `ParametroFormalenHelbideakVAR_A` programaz froga daitekeenez. Baina **6.4.3 Konstante-parametroak eta memori helbideak** puntuan `CONST` bitartez pasatutako argumentuak azaldu genituen `ParametroFormalenHelbideakCONST_A` izeneko programaren exekuzioa azalduz. Bertan `AzpirrutinaCONST()` deituriko prozedura baten irteera erakutsi zen, honelako zerbait azpimarratuz: `Oso1`, `Oso2` eta `Oso3` parametroak[5] erreferentziakoak izan beharrean baliozkoak dirudite euren helbideak pilan aurkitzen direlako. Esan genuen ere, datu-mota egituratuak ikastean argiago ulertzen dela `CONST` markaren bitartez pasatutako parametroak pilaren segmentuan ez daudela, hona hemen programa nagusitik `Azpirrutina()` prozedurara hiru arrayen informazioa bidaltzen duen programa:

```
PROGRAM ErregistroaParametroBezala ;            { \TP70\11\RECORD06.PAS }
USES
    Crt, Ikus ;      (* IKUS.TPU  unitatea darabil *)
TYPE
  DM_Langile = RECORD
                   Izena : String[39] ;
                   Adina : Byte ;
               END ;


PROCEDURE Azpirrutina (Erreg1 : DM_Langile;
                       VAR Erreg2 : DM_Langile;
                       CONST Erreg3 : DM_Langile) ;
BEGIN
   WriteLn ; WriteLn ;
   WriteLn ('ALDAGAIA':29, '       HELBIDEA', '   TAMAINA') ;
   WriteLn ('-------------------------------------------------':60) ;

   WriteLn ('Erreg3 ---(CONST)---->   ':36, IntToHex (Seg (Erreg3)),':',
            IntToHex (Ofs (Erreg3)), '    (', SizeOf (Erreg3), ')') ;
   WriteLn ('Erreg2 ---(VAR)------>   ':36, IntToHex (Seg (Erreg2)),':',
            IntToHex (Ofs (Erreg2)), '    (', SizeOf (Erreg2), ')') ;
   WriteLn ('Erreg1 ---(balioz)--->   ':36, IntToHex (Seg (Erreg1)),':',
            IntToHex (Ofs (Erreg1)), '    (', SizeOf (Erreg1), ')') ;

END ;
```

[5] `AzpirrutinaCONST()` izena duen prozedurara erreferentziaz pasatutako hiru zenbaki oso, gogoratu azpirrutinaren goiburukoa: `PROCEDURE AzpirrutinaCONST (CONST Oso1, Oso2, Oso3 : INTEGER) ;`

```
VAR
  Langile1, Langile2, Langile3 : DM_Langile ;

BEGIN              (* Programa Nagusia *)
   ClrScr ;

   WriteLn ('PARAMETRO ERREALEN ETA FORMALEN MEMORI HELBIDEAK') ;
   WriteLn ('==============================================') ;
   WriteLn ;
   WriteLn ('PILARI dagokion segmentuaren hasiera:  ',IntToHex (sSeg)) ;
   WriteLn ('DATUEI dagokien segmentuaren hasiera:  ',IntToHex (dSeg)) ;
   WriteLn ('KODEARI dagokion segmentuaren hasiera: ',IntToHex (cSeg)) ;
   WriteLn ;
   WriteLn ;
   WriteLn ('ALDAGAIA':29, '       HELBIDEA', '   TAMAINA') ;
   WriteLn ('-----------------------------------------':60) ;

   WriteLn ('Langile3 --->   ':36, IntToHex (Seg (Langile3)),':',
            IntToHex (Ofs (Langile3)), '    (', SizeOf (Langile3), ')') ;
   WriteLn ('Langile2 --->   ':36, IntToHex (Seg (Langile2)),':',
            IntToHex (Ofs (Langile2)), '    (', SizeOf (Langile2), ')') ;
   WriteLn ('Langile1 --->   ':36, IntToHex (Seg (Langile1)),':',
            IntToHex (Ofs (Langile1)), '    (', SizeOf (Langile1), ')') ;

   Azpirrutina (Langile1, Langile2, Langile3) ;

END.
```

`ErregistroaParametroBezala` programaman bi eremuko erregistroak erazagutzen dira, programa nagusiko hiru aldagaiak (`Langile1`, `Langile2` eta `Langile3`) datuen segmentuan sortzen dira eta `Azpirrutina()` prozedurara bidaltzean bakoitzak bere pasatze modua du:

- `Langile1` aldagaia baliozko parametroa da eta `Erreg1` dagokio
- `Langile2` arraya aldagai-parametro bat da eta `Erreg2` dagokio
- `Langile3` konstante-parametroari `Erreg3` dagokio azpirrutinan

`ErregistroaParametroBezala` programa egikaritu eta bere irteera azter dezagun, ikus daitekeenez, programa exekutatu den une eta ordenadore horretan pilaren segmentua `5A68` helbidean hasten da, datuak `5A29` helbidetik aurrera kokatzen dira eta programaren lehen sententzia `5949` helbidean dago:

```
PARAMETRO ERREALEN ETA FORMALEN MEMORI HELBIDEAK
================================================

PILARI dagokion segmentuaren hasiera:  5A68
DATUEI dagokien segmentuaren hasiera:  5A32
KODEARI dagokion segmentuaren hasiera: 5949


                     ALDAGAIA       HELBIDEA   TAMAINA
                  ----------------------------------------
                      Langile3 --->   5A32:00B2    (41)
                      Langile2 --->   5A32:0088    (41)
                      Langile1 --->   5A32:005E    (41)


                     ALDAGAIA       HELBIDEA   TAMAINA
              -----------------------------------------------
               Erreg3 ---(CONST)---->   5A32:00B2    (41)
               Erreg2 ---(VAR)------>   5A32:0088    (41)
               Erreg1 ---(balioz)--->   5A68:3DC4    (41)
_
```

`Langile1`, `Langile2` eta `Langile3` erregistroak programa nagusiko aldagaiak direlako datuen segmentuan sortzen dira. Bestalde, prozeduraren `Erreg2` eta `Erreg3`

aldagaiak erreferentziak dira eta ez dute pilan tokirik hartzen, baina `Erreg1` erregistroa berriz pilan aurkitzen da eta `Langile1` aldagaiaren kopia bat da. Ondorioz, baliozko parametro bat pasatzean parametro errealaren kopia bat egiten da pilan, eta datu-mota egituratuen kasuan prozesu horrek baliabide asko hartzen ditu[6] (nahiz eta adibideko erregistroak oso handiak ez izan), hori dela eta datu-mota egituratu bat azpiprograma batera bidaltzean erreferentziaz pasatuko da.

Hau guztia arrayetan aipatu izan zen ere, egin dezagun oraintxe laburpen bat.

Array edo/eta erregistro bat beti erreferentziaz pasatuko dela agindutzat hartuko dugu, hurrengo urratsa izango litzateke erreferentziazko parametroek dituzten bi moduetatik zein aukeratzea kasu bakoitzeko. Zalantza hau argitzeko aurreko taulari begirada arin bat ematea aski da, hots, array edo/eta erregistro baten portaera azpirrutinaren barnean datuak eskaintzearena denean konstante-parametro bat izango da (parametro erreala aldatua izatea ez baitzaigu interesatzen), baina array edo/eta erregistro batek balioak azpiprograman barnean hartzen dituenean modulu deitzaileak horren berri jaso behar duenez arraya edo/eta erregistroa aldagai-parametroa izango da.

Esandakoaren aplikazioa `ErregistroBiBerdinakOteDiren` adibide-programan ikusten da, non `ErregistroaBete()` prozedura eta `ErregistroakKonparatu()` funtzioa erabiltzen diren:

```
PROGRAM ErregistroBiBerdinakOteDiren ;              { \TP70\11\RECORD07.PAS }
TYPE
  DM_Ikaslea = RECORD
                  Izena : String[39] ;
                  Adina : Byte ;
               END ;

PROCEDURE ErregistroaBete (VAR Erreg : DM_Ikaslea) ;
BEGIN
   Write ('Ikaslearen izena eman:    ') ;      { Datuak biltegitzen }
   ReadLn (Erreg.Izena) ;

   Write ('Ikaslearen urteak eman:   ') ;
   ReadLn (Erreg.Adina) ;
END ;


FUNCTION ErregistroakKonparatu (CONST Erreg1, Erreg2 : DM_Ikaslea) : Boolean ;
BEGIN
   ErregistroakKonparatu := (Erreg1.Izena = Erreg2.Izena) AND
                            (Erreg1.Adina = Erreg2.Adina) ;
END ;


VAR                                              (* Programa Nagusia *)
  Ikasle1, Ikasle2 : DM_Ikaslea ;

BEGIN
   WriteLn ('"Ikasle1" izeneko erregistroa betetzen: ') ;
   ErregistroaBete (Ikasle1) ;

   WriteLn ('"Ikasle2" izeneko erregistroa betetzen: ') ;
   ErregistroaBete (Ikasle2) ;

   { Erregistroen arteko konparaketa burutu, hau da Ikasle1 erregistroaren
     eremuek gordetzen dutena eta Ikasle2-ren eremuen edukia berdinak diren }

   IF ErregistroakKonparatu(Ikasle1, Ikasle2) THEN
     WriteLn ('"Ikasle1" eta "Ikasle2" erregistroen edukiak berdinak dira')
   ELSE
     WriteLn ('"Ikasle1" eta "Ikasle2" erregistroak desberdinak dira') ;
END.
```

[6] Kopia egiteko denbora eta memoria behar direlako.

Esandakoarekin bat etorriz aurreko programan aurkitzen diren `ErregistroaBete()` prozeduraren eta `ErregistroakKonparatu()` funtzioaren parametroek dituzten pasatze moduak hauek dira:

`ErregistroaBete()`

`Erreg` prozedura honen bitartez `Erreg` erregistroak balioak hartzen ditu teklatuz string bat eta zenbaki oso bat irakurtzen baitira, teklatuz emandako datu horiek programa nagusiko `Ikasle1` eta `Ikasle2` erregistroek har ditzaten parametroaren pasatze modua aldagai-parametroarena izango da.

Azaldutako egoerak galdera bat egiteko aukera ematen digu, hona hemen: zer gertatuko litzateke `Erreg` parametroari aurretik `CONST` jarriko bagenio konstante-parametro bihurtuz?.

`CONST` bitartez markaturiko konstante-parametroa ezin denez errutina barruan aldatu `ReadLn(Erreg.Izena)` eta `ReadLn(Erreg.Adina)` sententziak ezin izango lirateke konpilatu, ezaguna den errore hau agertuko litzaiguke: `Error 122: Invalid variable reference.`

`ErregistroakKonparatu()`

`Erreg1` erregistroen arteko konparaketa lortzen duen `ErregistoakKonparatu()` funtzioaren lehen parametro hau egituratua da, eta portaeraz sarrerakoa da. Egituratua delako `Erreg1` erregistroa erreferentziaz pasatuko da (`VAR` edo `CONST`), eta sarrerakoa denez konstante-parametroa izango dela deduzitu ahal dugu.

Lehenago egin dugun galdera kontrako zentzuan egin dezagun, hots, zer gertatuko litzateke `Erreg1` parametroari aurretik `VAR` jarriko balitzaio aldagai-parametro bihurtuz?.

Ezer ere ez, `ErregistokKonparatu()` funtzioaren zeregina egokitasun eta zuzentasun osoz beteko litzateke. Baina kontuz, arrisku bat egon badago `Erreg1` parametro hori `VAR` bitartez markatzean, aldagai-parametroa izango litzatekeenez errutina barruan aldatzea zilegi litzaiguke eta nahi gabe modulu nagusiaren `Ikasle1` parametro erreala errutina barnean alda zitekeen.

`Erreg2` azaldu dugun `Erreg1` parametroa bezala.

11.2.4 Eragiketak erregistroen eremuekin

Erregistro baten eremuei aplikatuko zaizkien eragiketak eremu horri dagokion datu-motak onartzen dituen eragiketak izango dira.

Hau da, string bat eta zenbaki erreal bat biltzen dituen `Ikaslea` aldagaia erregistro bat izanik bere eremuei ezin zaie `Div` zatiketa osoa aplikatu, bai string eta bai zenbaki errealen datu-motek hori galarazten dutelako. Beste adibide bat jarriz gero, demagun `Ikaslea` aldagaiaren aurreneko eremua karaktere-katea dela, bere bigarren posizioan gordeta dagoen karakterea ezin izango da konstante batez biderkatu (karaktereekin eragiketa aritmetikoak debekatuta baitaude), ezta `AND` bezalako eragile logikoa aplikatu (karaktereekin eragiketa logikoak egiterik ez dago).

Hona hemen esandakoa Pascal lengoaiaz kodeturik:

```
TYPE
   DM_Ikaslea = RECORD
                    Izena : String[39] ;
                    Nota  : Real ;
                END;
VAR
   Ikaslea : DM_Ikaslea ;
```

Honezkero, jarraian ematen diren lerroetan erroreak daude:

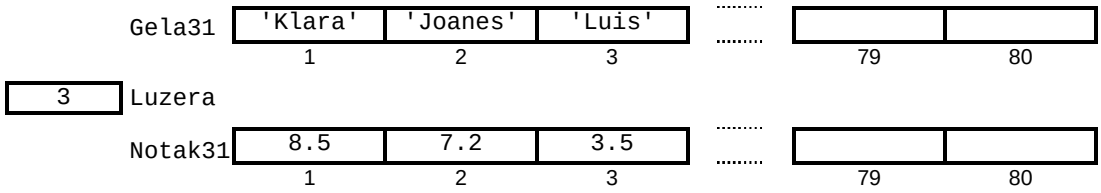
```
ZbkOsoa := Ikaslea.Nota Div 2 ;
ZbkOsoa := Ikaslea.Izena[2] * 5 ;
EmaitzaBoolear := Ikaslea.Izena[2] AND Amaiturik ;
```

Erregistroen eremuekin egin daitezkeen operazioen adibide bat **11.2.3.2 Erregistroa parametro bezala** puntuan ikusi izan da, non `ErregistroBiBerdinakOteDiren` adibide-programan `ErregistoakKonparatu()` funtzioaren barnean bi erregistroetako eremuen arteko konparaketak burutzen diren:

```
FUNCTION ErregistroakKonparatu (CONST Erreg1, Erreg2 : DM_Ikaslea) : Boolean ;
BEGIN
   ErregistroakKonparatu := (Erreg1.Izena = Erreg2.Izena) AND
                            (Erreg1.Adina = Erreg2.Adina) ;
END ;
```

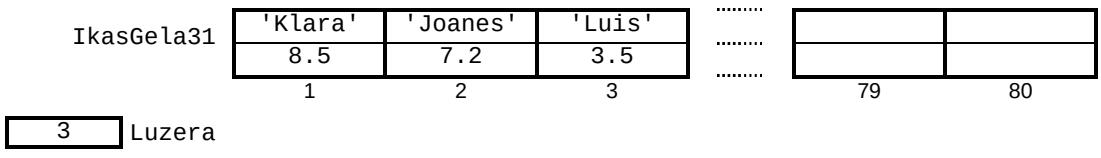
11.2.5 Erregistroen arrayak

Kapitulu honen hasieran `ArrayParaleloakLantzen` programaren kodea erakutsi genuen, zeinean ikasleen izenak eta notak bi array ezberdin baina erlazionatuetan gordetzen ziren. Array bat `Gela31` identifikadorez zehazten zen eta ikasleen izenak biltegitzeko erabiltzen zen, besteak berriz `Notak31` identifikadorea zuen eta paraleloki betetzen zen datuz. Hauxe litzateke `ArrayParaleloakLantzen` programa egikaritu ondoren array biek izan dezaketeen informazioa:



Array paraleloen prozesaketa erraza eta zehatza izanik soluzio honek ez du “ikasle” kontzeptua bere osotasunean aintzat hartzen, izan ere ikasle jakin bati dagokion informazioa bi egituretan barreiaturik dago. Esan genuen bezala ikasle bat adierazteko `Gela31[Indz]` eta `Notak31[Indz]` bikotea erabili baita.

Askoz argiago geratuko litzateke ikasle baten ezaugarri biak erregistro batean bilduko bagenitu, eta ondoren, erregistro horretan oinarritzen delarik ikastaldearen ideia ematen digun arraya definitu. Eskematikoki:



`ArrayParaleloakLantzen` programatik abiatuta erregistroetan oinarritzen den arrayaren programari `ErregistroenArraya` deituko diogu. Hona hemen bere kodea:

```
PROGRAM ErregistroenArraya ;                { \TP70\11\RECORD08.PAS }
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 80 ;
TYPE
  DM_Ikaslea = RECORD
                  Izena : String[39] ;
                  Nota  : Real ;
               END ;
  DM_Taldea = ARRAY[BEHEMUGA..GOIMUGA] OF DM_Ikaslea ;


PROCEDURE ArrayaBete (VAR IkasGela : DM_Taldea; VAR LuzLog : Byte) ;
VAR
  Indize : Byte ;
  Erantz : Char ;
BEGIN
   Erantz := 'B' ;
   Indize := 1 ;
   WHILE (Indize <= GOIMUGA) AND (Erantz = 'B') DO
   BEGIN
      Write (Indize, '. ikaslearen izena eman: ') ;
      ReadLn (IkasGela[Indize].Izena) ;
      Write (Indize, '. ikaslearen nota eman:  ') ;
      ReadLn (IkasGela[Indize].Nota) ;
      REPEAT
         Write ('Ikasle gehiagorik? (B/E): ') ;
         ReadLn (Erantz) ;
         Erantz := UpCase (Erantz) ;
      UNTIL (Erantz = 'B') OR (Erantz = 'E') ;
      IF Erantz = 'B' THEN
        Indize := Indize + 1 ;
   END ;

   IF Indize > GOIMUGA THEN
     LuzLog := GOIMUGA
   ELSE
     LuzLog := Indize ;
END;


PROCEDURE ArrayaIkus (CONST IkasGela : DM_Taldea; LuzLog : Byte) ;
VAR
  Indize : Byte ;
BEGIN
   FOR Indize:=1 TO LuzLog DO
   BEGIN
      Write (Indize, '. ikaslea: ') ;
      Write (IkasGela[Indize].Izena:15, '   ') ;
      WriteLn (IkasGela[Indize].Nota:10:1) ;
   END ;
END;

VAR
  IkasGela31 : DM_Taldea ;
  Luzera : Byte ;
BEGIN
   WriteLn ('Izenak eta notak memorian gorde:') ;
   ArrayaBete (IkasGela31, Luzera) ;
   WriteLn ;
   WriteLn ('Hona hemen ikasle bakoitzeko gordetako informazioa:') ;
   ArrayaIkus (IkasGela31, Luzera) ;
END.
```

Hauxe da aurreko eskemari dagokion `ErregistroenArraya` programaren exekuzioa:

```
Izenak eta notak memorian gorde:
1. ikaslearen izena eman: Klara
1. ikaslearen nota eman:  8.5
Ikasle gehiagorik? (B/E): b
2. ikaslearen izena eman: Joanes
2. ikaslearen nota eman:  7.2
Ikasle gehiagorik? (B/E): b
3. ikaslearen izena eman: Luis
3. ikaslearen nota eman:  3.5
Ikasle gehiagorik? (B/E): e

Hona hemen ikasle bakoitzeko gordetako informazioa:
1. ikaslea:           Klara          8.5
2. ikaslea:          Joanes          7.2
3. ikaslea:            Luis          3.5
_
```

`ErregistroenArraya` programan dauden berrikuntzak `IkasGela[Indize].Izena` eta `IkasGela[Indize].Nota` bezalako erreferentziak dira. Horiek ulertzeko arau orokor bat dago programazio lengoaia gehienetan, “egitura konplexu bat dugunean kanporago dauden egiturak ezkerrago idazten dira”. Beraz, `IkasGela[Indize].Izena[1]` erreferentzia honela banatzen da:

`IkasGela`[Indize].Izena[1] — Ezkerreko muturrean dagoen `IkasGela` identifikadoreak kanpoko azken geruza egitura izango da, etiketa horrek array bat adierazten duela dakigu. Ondorioz, programa barruan `IkasGela` identifikadorea aurkituko dugunean array osoaz ari garela ez ahaztu.

Array batek dituen osagaien erreferentzia egiteko `[]` sinboloak behar direla gogoratzen dugu, eta osagai horiek arrayaren “barnean” daudenez `[]` sinboloak bere eskuinean idatziko dira.

`IkasGela[Indize]`.Izena[1] — Ezkerreko muturretik hurrengo maila `IkasGela[Indize]` identifikadoreak zehazten du, eta honek arrayaren `Indize`-garren elementuaz ari garela adierazten du.

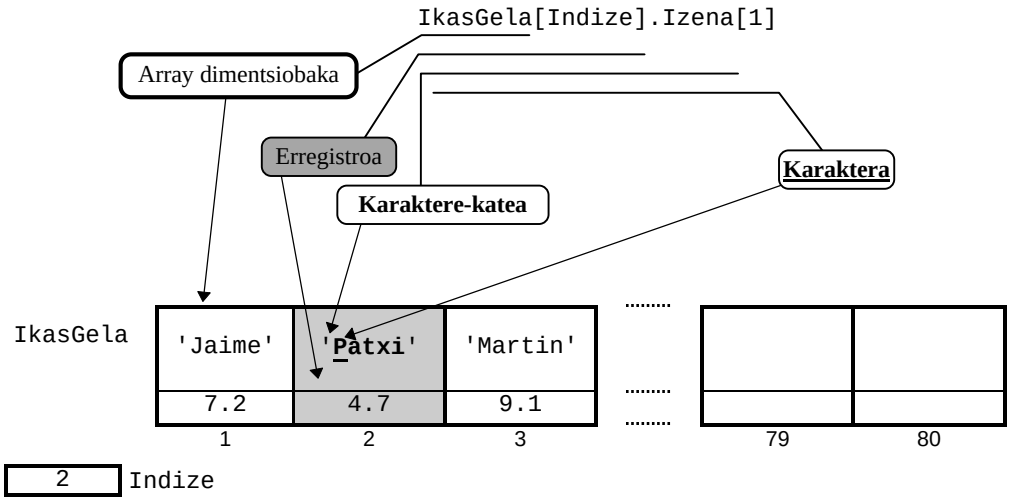
Ariketa honetan `IkasGela[Indize]` identifikadoreak datu-mota egituratua den erregistro bat dela dakigu, ikusi izan den bezala erregistro bat bere eremuekin konektatzeko puntu eragilea erabiltzen da.

`IkasGela[Indize].Izena`[1] — Egituraren hirugarren geruza `IkasGela[Indize].Izena` litzateke, zein karaktere-kate bat adierazten duen.

Karaktere-kateek bere barnean karaktereak dituzteelako egituratuak kontsidera daitezke. Horregatik ikasgelako ikasle baten izenaren aurreneko letra zehazteko `IkasGela[Indize].Izena` erreferentziari eskuinetik `[1]` sinboloak gehituko zaizkio.

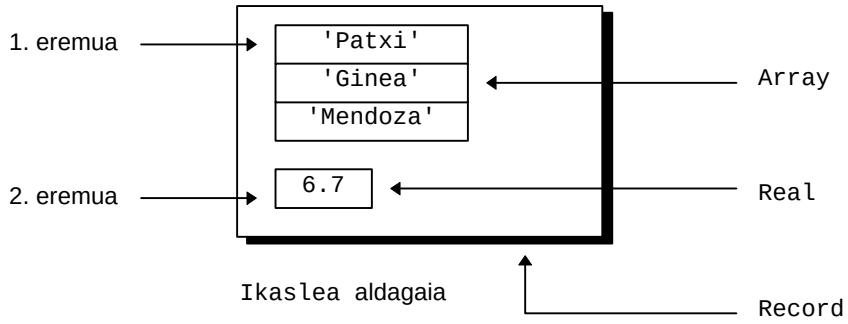
`IkasGela[Indize].Izena[1]` — Egitura osoaren azken geruza ikasle baten izenaren letra bat litzateke, adibidez `IkasGela[Indize].Izena[1]` bitartez adieraziko litzatekeena (ikasgelako ikasle baten izenaren aurreneko letra).

Dena bilduz:



Ikusten denez kanpoko mailak ezkerrean daude eta barnekoak eskuinean, maila batetik bestera igarotzeko erabili behar diren konektoreak datu-motaren araberakoak dira `[]` sinboloak array eta karaktere-kateetan eta puntu operadorea erregistroetan.

Datu-mota egituratu baten mailen arteko ordenak zerikusia handia du. Esate baterako erregistroen array bat izan beharrean, array bat erregistroan barneraturik bagenu honelako zerbait izango genuke (ikus `ArrayenErregistroa` izeneko programa):

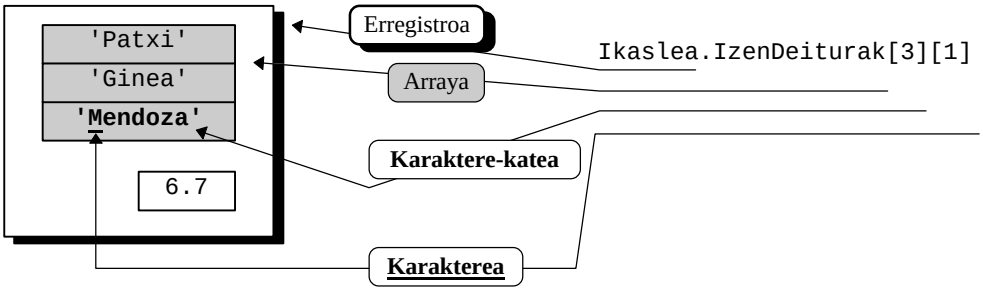



```
CONST
  BEHEMUGA = 1 ;
  GOIMUGA  = 3 ;
TYPE
  DM_Kateak = ARRAY[BEHEMUGA..GOIMUGA] OF String[9] ;
  DM_Ikasle = RECORD
                  IzenDeiturak : DM_Kateak ;
                  Nota  : Real ;
              END ;
VAR
  Ikaslea : DM_Ikasle ;
```

Eta kasu honetan bigarren abizenaren aurreneko karakterea lortzeko behar dugun segida honelako ordenean eman beharra dago: Erregistro aldagaia / Lehen eremua / Arrayaren hirugarren osagaia / Katearen lehendabiziko karakterea.

```
Ikaslea.IzenDeiturak[3][1]
```

`Ikaslea.IzenDeiturak[3][1]` erreferentzia eskematizatuz jarraian agertzen den irudia izan dezakegu. Ikusten denez `Ikaslea.IzenDeiturak[3][1]` erreferentziak, datu horiekin, `M` karakterea adierazten du:



Hau litzateke `Ikaslea` aldagaia informazioz betetzeko `ArrayenErregistroa` izeneko programa:

```
PROGRAM ArrayenErregistroa ;                { \TP70\11\RECORD09.PAS }
CONST
  BEHEMUGA = 1 ;
  GOIMUGA  = 3 ;
TYPE
  DM_Kateak = ARRAY[BEHEMUGA..GOIMUGA] OF String[9] ;
  DM_Ikasle = RECORD
                 IzenDeiturak : DM_Kateak ;
                 Nota  : Real ;
              END ;


PROCEDURE ErregistroaBete (VAR Ikaslea : DM_Ikasle) ;
BEGIN
   Write ('Ikaslearen izena eman:            ') ;
   ReadLn (Ikaslea.IzenDeiturak[1]) ;
   Write ('Ikaslearen lehen abizena eman:    ') ;
   ReadLn (Ikaslea.IzenDeiturak[2]) ;
   Write ('Ikaslearen bigarren abizena eman: ') ;
   ReadLn (Ikaslea.IzenDeiturak[3]) ;
   Write ('Ikaslearen nota eman:  ') ;
   ReadLn (Ikaslea.Nota) ;
END;


PROCEDURE ErregistroaIkus (CONST Ikaslea : DM_Ikasle) ;
VAR
  Indize : Byte ;
BEGIN
   Write ('Ikaslea -----> ') ;
   FOR Indize:=BEHEMUGA TO GOIMUGA DO
   BEGIN
      Write (Ikaslea.IzenDeiturak[Indize]:12, '   ') ;
   END ;
   WriteLn (Ikaslea.Nota:10:1) ;
END;

VAR
  Ikaslea : DM_Ikasle ;
BEGIN
   WriteLn ('Izen deiturak eta nota memorian gorde:') ;
   ErregistroaBete (Ikaslea) ;
   WriteLn ;
   WriteLn ('Hona hemen ikasleari dagokion informazioa:') ;
   ErregistroaIkus (Ikaslea) ;
END.
```

11.2.5.1 Adibidea

Asko dira erregistroen arrayekin plantea daitezkeen ariketak eta guztietan hamargarren kapituluko **10.4 ARRAY DIMENTSIOBAKARREN GAINEKO ERAGIKETAK** izeneko puntuan esandakoa aplikatu behar izaten da, esate baterako jarraian ematen den adibidean bilaketak eta ibilerak burutzen dira. Adibidearen zailtasunik aipagarriena datu-motak dira, izan ere "ikasle" kontzeptua errepresentatzeko erregistro hierarkiko[7] bat erabiltzen da (erregistro kabiatuak **11.2.7 Erregistro hierarkikoak** izeneko puntuan astiroago eta sakonago ikusiko dira).

Hona hemen ariketaren enuntziatua:

Ikasgela bateko zerrenda antolatu nahi da, ikasgela osatzen duten ikasle bakoitzeko ondoko informazioa gordetzen da:

`izena` (20 karakteredun katea)
`deiturak` (20 karakteredun 2 kateen arraya)
`azpitaldea` (karakterea)
`teoriako nota` (zenbaki erreala)
`laborategiko nota` (zenbaki erreala)

Programa menu baten bitartez gidatuko da eta hiru eginkizun bete ahal izango dira: ikasle baten datuak ikasgelako zerrendan gorde, ikasle bat bilatu ondoren bere informazioa pantailaratu eta zerrenda osoko informazioa erakutsi. Aukerok taula bezala jarriz:

1	*Ikasle baten datuak zerrendan GORDE*
2	*Ikasle baten datuak zerrendan BILATU*
3	*Zerrendako datuak ERAKUTS*
0	*IRTEN*

1. AUKERA

Aukera honen helburua ikasle berri baten informazioa zerrendan jasotzea da. Nahiz eta oso erraza izan ez da konprobatzen ikaslea benetan berria den, hau da, izen-abizen bereko ikaslerik lehendik erregistraturik ote dagoen. Arrayan informazioa gordetzean ez da inolako ordenik zaintzen, baina arrayaren luzera logikoak ez dezala gainditu luzera fisikoa kontrolatzen da.

2. AUKERA

Aukera honen bitartez ikasle baten izen-deiturak teklatuz ematen dira eta ikasgelako zerrendan bilatu ondoren bere datuak pantailaratzen dira. Arrayan inolako ordenik ez dagoelako bilaketa sekuentziala erabiltzen da eginkizun honetarako.

3. AUKERA

Aukera honi dagokion algoritmoa array baten ibilera litzateke.

[7] Erregistroen artean hierarkiak antola daitezke, hots, erregistro baten eremuren bat beste erregistro bat denenan. Erregistro bat beste erregistro baten barnean kabiatzen denean eremuen arteko konektorea, nola ez, puntu eragilea da: lehen mailako eremuen atzipenak lortzeko aldagaia eta eremua zehaztuko dira puntu batez konektaturik (aldagaia.eremua), baina bigarren mailako eremuen atzipenak gauzatzeko puntu bi beharko dira (aldagaia.eremua.eremua).

```
PROGRAM Erregistroen_Adibidea;                          { \TP70\11\IKASLE.PAS }
USES
  Crt;
CONST
  MAX=75;
TYPE
  KateLabur = STRING[10];
  KateLuze = STRING[20];
  DM_ident = RECORD
               izena : KateLabur;
               deiturak : ARRAY[1..2] OF KateLuze;
             END;
  DM_notak = RECORD
               teoria, laborategia : REAL;
             END;
  DM_ikasle = RECORD
                nor : DM_ident;
                azpitalde : CHAR;
                notak : DM_notak;
              END;
  DM_gela = ARRAY [1..MAX] OF DM_ikasle;



FUNCTION menua : CHAR;
VAR
   hautapen : CHAR;
BEGIN
    ClrScr;
    Writeln ('======================================');
    Writeln ('1  Ikasle baten datuak zerrendan gorde');
    Writeln ('2  Ikasle baten datuak zerrendan bilatu');
    Writeln ('3  Zerrendako datuak erakutsi');
    Writeln ('======================================');
    Write ('Irteteko 0. Zure aukera:  ');
    Readln (hautapen);
    menua := hautapen;
END;



PROCEDURE IkasleBatenDatuakSartu (VAR g:DM_gela; VAR luzera:BYTE);
VAR
   itxoin : CHAR;
BEGIN
    IF luzera = MAX THEN
                      BEGIN
                          Writeln ('Gela beterik dago!');
                          itxoin := Readkey;
                      END
    ELSE
      BEGIN
          ClrScr;
          luzera := luzera + 1;
          Write (luzera,'-garren ikaslearen izena: ');
          Readln (g[luzera].nor.izena);
          Write (luzera,'-garren ikaslearen 1. abizena: ');
          Readln (g[luzera].nor.deiturak[1]);
          Write (luzera,'-garren ikaslearen 2. abizena: ');
          Readln (g[luzera].nor.deiturak[2]);
          Write (luzera,'-garren ikaslearen azpitaldea: ');
          Readln (g[luzera].azpitalde);
          Write (luzera,'-garren ikaslearen teoriako nota: ');
          Readln (g[luzera].notak.teoria);
          Write (luzera,'-garren ikaslearen laborategiko nota: ');
          Readln (g[luzera].notak.laborategia);
      END;
END;
```



```
FUNCTION IkasleBatBilatu (CONST ikasgela : DM_gela;
                               zenbat : BYTE;
                               izenBat : KateLabur;
                               abi1, abi2 : KateLuze) : BYTE;
VAR
  aurkitua : BOOLEAN;
  kont : BYTE;
BEGIN
    IkasleBatBilatu:=0;
    kont := 1;
    aurkitua := FALSE;
    WHILE (NOT aurkitua) AND (kont<=zenbat) DO
    BEGIN
       IF (ikasgela[kont].nor.izena = izenBat) AND
          (ikasgela[kont].nor.deiturak[1] = abi1) AND
          (ikasgela[kont].nor.deiturak[2] = abi2)
       THEN
           BEGIN
                aurkitua := TRUE;
                IkasleBatBilatu := kont;
           END
       ELSE kont := kont + 1;
    END;
END;



PROCEDURE IkaslearenDatuakIkusi (CONST g:DM_gela; zein:BYTE);
VAR
  k : BYTE;
  itxoin : CHAR;
BEGIN
     Writeln;
     Writeln ('Hona hemen ', zein,'-garren ikaslearen datuak: ');
     Write ('Izen-deiturak:  ', g[zein].nor.izena);
     Write ('  ', g[zein].nor.deiturak[1]);
     Writeln ('  ', g[zein].nor.deiturak[2]);
     Writeln ('Praktikatarako azpitaldea: ', g[zein].azpitalde);
     Write ('Teoriako nota: ', g[zein].notak.teoria:0:2);
     Writeln ('   Laborategiko nota: ', g[zein].notak.laborategia:0:2);
     Writeln;
     Writeln ('Aurrera jarraitzeko tekla bat sakatu!');
     itxoin := ReadKey;
END;



PROCEDURE IkasleGuztienDatuakIkusi (CONST gela:DM_gela; luzera:BYTE);
VAR
  k : BYTE;
BEGIN
    FOR k:=1 TO luzera DO
        IkaslearenDatuakIkusi (gela, k)
END;



             (* Programa Nagusiko aldagaiak *)
VAR
   gela : DM_gela;
   luz, non : BYTE;
   aukera, geldi : CHAR;
   izenBat : KateLabur;
   abizen1, abizen2 : KateLuze;
```

```
BEGIN
   ClrScr;
   luz := 0;                        (* hasieran zerrenda hutsik dago *)
   REPEAT
         aukera := menua;
         CASE aukera OF
              '1' : IkasleBatenDatuakSartu (gela, luz);
              '2' : BEGIN
                      Write ('Bilatu behar den ikaslearen izena: ');
                      Readln (izenBat);
                      Write ('Bilatu behar den ikaslearen 1. abizena: ');
                      Readln (abizen1);
                      Write ('Bilatu behar den ikaslearen 2. abizena: ');
                      Readln (abizen2);
                      non := IkasleBatBilatu (gela, luz, izenBat, abizen1, abizen2);
                      IF 0 = non THEN
                                 BEGIN
                                   Write (izenBat,' ', abizen1,' ');
                                   Writeln (abizen2,' ikaslea ez dago');
                                   Writeln ('Tekla bat sakatu!');
                                   geldi := ReadKey;
                                 END
                      ELSE
                         BEGIN
                            IkaslearenDatuakIkusi (gela, non);
                         END;
                    END;
              '3' : IkasleGuztienDatuakIkusi (gela, luz);
         END;
   UNTIL aukera='0';
END.
```

11.2.5.2 Adibidea

Erregistroen arrayen bigarren adibidea konplexuagoa da, erregistro kabiatuak[8] erabiltzen dira (ikus **11.2.7 Erregistro hierarkikoak** izeneko puntua), gainera kontutan izan datu-motak definitzean ez da `DM_` markarik jarri eta horrek programa ulertzeko zailtasunak gehitzen ditu. Hona hemen ariketaren enuntziatua:

Biblioteka baten funtzionamendua simulatuko duen programa idatzi nahi da. Biblioteka osatzen duten liburuen multzoa array dimentsiobakar baten bidez modelizatuko da, array-aren elementuek biltzen duten informazioa honako hau da:

`izenburua` (50 karakteredun katea)
`egilea` (30 karakteredun katea)
`kodea` (10 karakteredun katea)
`itzultzeko eguna` (zenbaki osoa)

Azken eremuak, hots itzultzeko eguna, erabiltzaileak liburua noiz itzuli behar duen adierazten du. Zenbaki oso bat da, eta erabiltzaileak urtearen zein egun arte liburuaz balia daitekeen azaltzen du. Libururen bat prestaturik gabe bibliotekan dagoenean bere itzul-eguna 0 izango da.

[8] Erregistro bat beste erregistro baten barnean kabiatzen denean eremuen arteko konektoreak, esan dugun bezala, puntu eragilea da. Eremu baten erreferentzian agertuko diren puntuen kopurua eremuaren mailaren araberakoa izango da, esate baterako bigarren mailako eremuen atzipenak gauzatzeko puntu bi beharko dira (aldagaia.eremua.eremua). Horren erreferentzia luzeak ekiditeko `WITH` klausula erabil daiteke.

Utzitako liburuen kontrolerako beste array lineal bat dago, array honek izango duen informazioa jarraian zehazten da:

```
kodea  (10 karakteredun katea)
erabiltzaile  (holako eremuak dituen erregistroa:
              izena  (30 karakteredun katea)
              helbidea  (50 karakteredun katea)
              telefonoa  (10 karakteredun katea))
itzultzeko eguna  (zenbaki osoa)
```

Liburu jakin baterako, itzultzeko eguna eta kodea eremuak berberak izango dira array bietan (bibliotekan eta utzitako liburuen array-etan).

Bibliotekaren kudeaketa menu batez gidatuko da. Menuaren aukerak sei hauek dira:

1	*Liburu berri bat bibliotekan SARTU*
2	*Liburu bat bibliotekatik KENDU*
3	*Liburu bat prestaturik UTZI*
4	*Utzitako liburua ITZULI*
5	*Bibliotekaren KONTSULTA*
6	*Utzitako liburuak ZERRENDATU*
0	*IRTEN*

1. AUKERA

Aukera honen helburua, liburu berri baten fitxa egitea eta bibliotekan errregistratzea da. Liburu berriari dagokion kodea irakurriko da, eta benetan berria dela konprobatu ondoren gainerako datuak irakurriko dira eta bibliotekaren array-a gaurkoratuko da. Bibliotekaren array hau liburuen kodeei arabera ordenaturik mantentzen da.

2. AUKERA

Aurrekoaren aurkakoa litzateke. Bibliotekatik eliminatu nahi den liburuaren kodea irakurri eta gero, esistitzen dela eta utzita ez dagoela egiaztatu ondoren bibliotekaren array-atik kendu.

3. AUKERA

Kontsultatu nahi den liburua, bere kodez identifikatu ondoren esistitzen dela eta utzi gabe dagoela konprobatu behar da. Baiezkoan, gainerako informazioa irakurri eta utzitako liburuen array-a gaurkoratuko da. Era berean, liburu hori utzita dagoela adieraziz bibliotekaren arraya ere gaurkoratuko da.

4. AUKERA

Hirugarren aukeraren aurkako operazioa. Itzuli nahi den liburuaren kodea irakurri, eta benetan utzita dagoela egiaztatu ondoren array biak gaurkoratu.

5. AUKERA

Bibliotekaren array-aren elementu guztiak pantailaratzea litzateke.

6. AUKERA

Utzitako liburuen array-a ez dago ordenaturik, baina bere datuak pantailaratu aurretik ordenatu beharra dago. Aukera honen hasieran utzitako liburuen arrayaren ordenaziorako irizpidea eskatuko da (onartzen diren irizpideak erabiltzailearen izena eta itzultzeko eguna izango dira). Nolako ordenazioa nahi den ezaguturik, utzitako liburuen array-a ordenatu eta pantailaratu.

```
PROGRAM Biblioteka_Kudeaketa;                          { \TP70\11\LIBURU.PAS }
USES
    Crt;
CONST
    MAX=5;
    M=4;
TYPE
    kate50=STRING[50];
    kate30=STRING[30];
    kate10=STRING[10];
    liburu=RECORD
             izenburu:kate50;
             egile:kate30;
             kode:kate10;
             itzul_eguna:INTEGER;
           END;
    biblioteka=ARRAY[1..MAX] OF liburu;
    identifikazio=RECORD
                     izena:kate30;
                     helbide:kate50;
                     telefono:kate10;
                  END;
    erabilia=RECORD
                kode:kate10;
                erabiltzaile:identifikazio;
                itzul_eguna:INTEGER;
             END;
    utzitakoak=ARRAY[1..M] OF erabilia;



FUNCTION AukeraHautatu:CHAR;
VAR
  karak:CHAR;
BEGIN
   REPEAT
      WRITELN('==============Menua=============');
      WRITELN('1   Liburu baten ALTA ematea');
      WRITELN('2   Liburu baten BAJA ematea');
      WRITELN('3   Liburu bat prestaturik HARTU');
      WRITELN('4   Prestaturiko liburua ITZULI');
      WRITELN('5   Bibliotekaren edukia IKUSI');
      WRITELN('6   Utzitako liburuak LISTATU');
      WRITELN('0   IRTEN');
      WRITE('Zure aukera:  ');
      READLN(karak);
   UNTIL (karak>='0') AND (karak<='6');
   AukeraHautatu:=karak;
END;



FUNCTION OrdenatzekoIrizpideaAukeratu:CHAR;
VAR
  karak:CHAR;
BEGIN
   REPEAT
      WRITELN('===========Ordenatu============');
      WRITELN('1   Erabiltzailearen IZENARI buruz');
      WRITELN('2   Liburuaren itzul EGUNARI buruz');
      WRITE('Zure aukera:  ');
      READLN(karak);
   UNTIL (karak='1') OR (karak='2');
   OrdenatzekoIrizpideaAukeratu:=karak;
END;
```

```
FUNCTION LiburuaBibliotekanBilatu (CONST B:biblioteka;
                                   luzB:INTEGER;
                                   kode:kate10): INTEGER;
VAR
   kont:INTEGER;
   aurkitua:BOOLEAN;
BEGIN
    kont:=1;
    aurkitua:=FALSE;
    WHILE (kont<=luzB) AND (NOT aurkitua) DO
    BEGIN
        IF B[kont].kode=kode THEN
                                BEGIN
                                  aurkitua:=TRUE;
                                  LiburuaBibliotekanBilatu:=kont;
                                END;
        kont:=kont+1;
    END;
    IF aurkitua=FALSE THEN LiburuaBibliotekanBilatu:=0;
END;



FUNCTION UtzitakoLiburuenZerrendanBilatu (CONST U:utzitakoak;
                                          luzU:INTEGER;
                                          kode:kate10): INTEGER;
VAR
   kont:INTEGER;
   aurkitua:BOOLEAN;
BEGIN
    kont:=1;
    aurkitua:=FALSE;
    WHILE (kont<=luzU) AND (NOT aurkitua) DO
    BEGIN
        IF U[kont].kode=kode THEN
                                BEGIN
                                  aurkitua:=TRUE;
                                  UtzitakoLiburuenZerrendanBilatu:=kont;
                                END;
        kont:=kont+1;
    END;
    IF aurkitua=FALSE THEN UtzitakoLiburuenZerrendanBilatu:=0;
END;



PROCEDURE LiburuaBibliotekanSartu (VAR B:biblioteka;
                                   VAR luzB:INTEGER;
                                   CONST alea:liburu);
VAR
  i,kont:INTEGER;
BEGIN
     kont:=1;
     WHILE (kont<=luzB) AND (B[kont].kode<alea.kode) DO
             kont:=kont+1;
     IF kont=luzB+1 THEN B[kont]:=alea
     ELSE
         BEGIN
              FOR i:=luzB DOWNTO kont DO
                    B[i+1]:=B[i];
              B[i]:=alea;
         END;
     luzB:=luzB+1;
END;
```

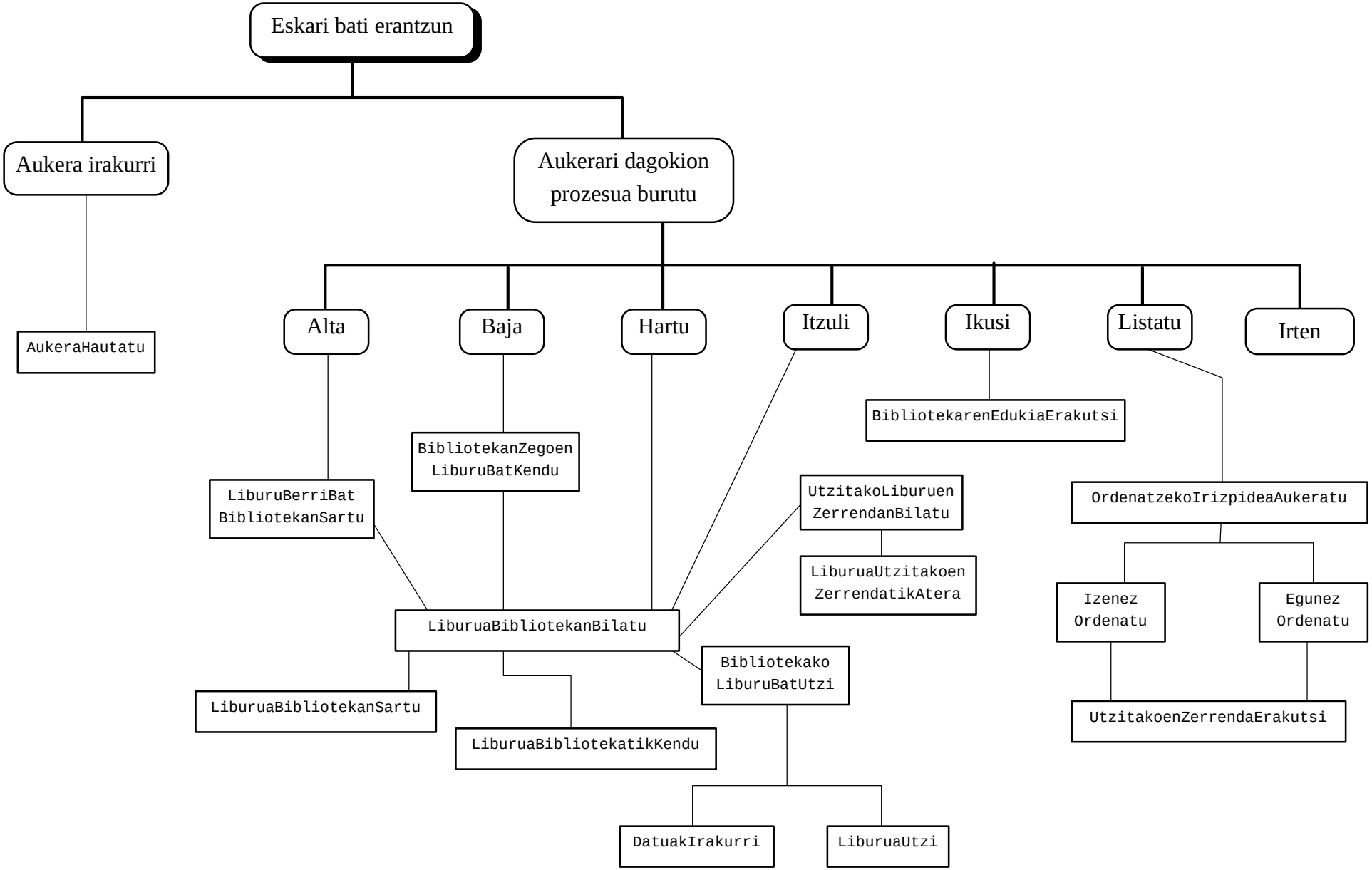
```
PROCEDURE LiburuaBibliotekatikKendu (VAR B:biblioteka;
                                     VAR luzB:INTEGER;
                                     indize:INTEGER);
VAR
  i:INTEGER;
BEGIN
     FOR i:=indize TO luzB-1 DO
        B[i]:=B[i+1];
     luzB:=luzB-1;
END;



PROCEDURE LiburuaUtzitakoenZerrendatikAtera (VAR U:utzitakoak;
                                             VAR luzU:INTEGER;
                                             indize:INTEGER);
VAR
  i:INTEGER;
BEGIN
     FOR i:=indize TO luzU-1 DO
        U[i]:=U[i+1];
     luzU:=luzU-1;
END;



PROCEDURE BibliotekarenEdukiaErakutsi (CONST B:biblioteka; luzB:INTEGER);
VAR
   i:INTEGER;
   karaktere:CHAR;
BEGIN
   FOR i:=1 TO luzB DO
   BEGIN
       clrscr;
       WRITELN(i, '. elementua');
       WRITELN;
       WITH B[i] DO
       BEGIN
            WRITELN('Izenburua: ', izenburu);
            WRITELN('Egilea:    ', egile);
            WRITELN('Kodea:     ', kode);
            WRITELN('Eguna:     ', itzul_eguna);
       END;
       karaktere:=READKEY;
   END;
END;



PROCEDURE UtzitakoenZerrendaErakutsi (CONST U:utzitakoak; luzU:INTEGER);
VAR
   i:INTEGER;
   karaktere:CHAR;
BEGIN
   FOR i:=1 TO luzU DO
   BEGIN
       clrscr;
       WRITELN(i, '. elementua');
       WRITELN;
       WITH U[i] DO
       BEGIN
            WRITELN('Kodea:       ', kode);
            WRITELN('Itzul eguna: ', itzul_eguna);
            WRITELN('Izena:       ', erabiltzaile.izena);
            WRITELN('Helbidea:    ', erabiltzaile.helbide);
            WRITELN('Telefonoa:   ', erabiltzaile.telefono);
       END;
       karaktere:=READKEY;
   END;
END;
```

```
PROCEDURE DatuakIrakurri (VAR mailegua:erabilia);
BEGIN
  WITH mailegua DO
  BEGIN
       WRITE('Norentzat:  ');
       READLN(erabiltzaile.izena);
       WRITE('Bizilekua:  ');
       READLN(erabiltzaile.helbide);
       WRITE('Telefonoa:  ');
       READLN(erabiltzaile.telefono);
       WRITE('Azken eguna:  ');
       READLN(itzul_eguna);
  END;
END;



PROCEDURE LiburuaUtzi (VAR B:biblioteka; indize:INTEGER;
                       VAR U:utzitakoak; VAR luzU:INTEGER;
                       mailegua:erabilia);
BEGIN
     B[indize].itzul_eguna:=mailegua.itzul_eguna;
     luzU:=luzU+1;
     U[luzU]:=mailegua;
END;



PROCEDURE IzenezOrdenatu (VAR U:utzitakoak; luzU:INTEGER);
VAR
   k,j,pos:INTEGER;           (* Aukeraketaren metodoa *)
   min:erabilia;
BEGIN
  FOR k:=1 TO luzU-1 DO       (* Iterazio bakoitzeko, izenik *)
  BEGIN                       (* txikiena AUKERATU egiten da *)
    min := U[k];
    pos := k;
    FOR j:=k+1 TO luzU DO
      IF min.erabiltzaile.izena > U[j].erabiltzaile.izena THEN
      BEGIN
        min:=U[j];
        pos:=j;
      END;

    U[pos]:=U[k];     (* Trukatu, tokiz aldatu *)
    U[k]:=min;
  END;
END;



PROCEDURE EgunezOrdenatu (VAR U:utzitakoak; luzU:INTEGER);
VAR
   k,j:INTEGER;        (* Bilaketaren metodoa *)
   lag:erabilia;
BEGIN
  FOR k:=2 TO luzU DO         (* Iterazio bakoitzeko, uneko egunari *)
  BEGIN                       (* dagokion posizioa bilatu egiten da *)
    lag := U[k];
    j := k-1;
    WHILE (lag.itzul_eguna < U[j].itzul_eguna) AND (j >= 1) DO
      BEGIN
        U[j+1]:=U[j];     (* Guztiak tokiz aldatu ondoren ... *)
        j:=j-1;
      END;
    U[j+1]:=lag;           (* ... tartekatu *)
  END;
END;
```

```
PROCEDURE LiburuBerriBatBibliotekanSartu (VAR B:biblioteka; VAR luzB:INTEGER);
VAR
  libKode:kate10;
  indize:INTEGER;
  alea:liburu;
BEGIN
     WRITE('Kodea:  ');
     READLN(libKode);
     indize:=LiburuaBibliotekanBilatu(B,luzB,libKode);
     IF indize<>0 THEN WRITELN('Errorea, liburu hori badago')
     ELSE
         BEGIN
              WITH alea DO
              BEGIN
                   kode:=libKode;
                   WRITE('Izenburua:  ');
                   READLN(izenburu);
                   WRITE('Egilea:     ');
                   READLN(egile);
                   itzul_eguna:=0;
              END;
              LiburuaBibliotekanSartu(B,luzB,alea);
         END;
END;



PROCEDURE BibliotekanZegoenLiburuBatKendu(VAR B:biblioteka;
                                          VAR luzB:INTEGER;
                                          libKode:kate10);
VAR
  indize:INTEGER;
BEGIN
     indize:=LiburuaBibliotekanBilatu(B,luzB,libKode);
     IF indize=0 THEN WRITELN('Errore, ez dago')
     ELSE
         BEGIN
              IF B[indize].itzul_eguna<>0 THEN WRITELN('Utzita. Ezin da kendu')
              ELSE LiburuaBibliotekatikKendu(B,luzB,indize);
         END;
END;



PROCEDURE BibliotekakoLiburuBatUtzi (VAR B:biblioteka; luzB:INTEGER;
                                     VAR U:utzitakoak; VAR luzU:INTEGER;
                                     libKode:kate10; indize:INTEGER);
VAR
   mailegua:erabilia;
BEGIN
   IF indize=0 THEN WRITELN('Errore, liburua ez dago')
   ELSE
     BEGIN
        IF B[indize].itzul_eguna<>0 THEN
          BEGIN
               WRITE('Utzita. Itzuliko den eguna: ');
               WRITELN(B[indize].itzul_eguna);
          END
        ELSE
          BEGIN
               IF luzU>=M THEN WRITELN('Mailegu larregi')
               ELSE
                  BEGIN
                       DatuakIrakurri(mailegua);
                       mailegua.kode:=libKode;
                       LiburuaUtzi(B,indize,U,luzU,mailegua);
                  END;
          END;
     END;
END;
```

```
VAR                                  (* Programa Nagusiko aldagaiak *)
   aukera,irizpide:CHAR;
   B:biblioteka;
   luzB:INTEGER;
   U:utzitakoak;
   luzU:INTEGER;
   libKode:kate10;
   indize:INTEGER;


BEGIN
     clrscr;
     luzB:=0;
     luzU:=0;

     REPEAT
         aukera:=AukeraHautatu;
         CASE aukera OF
           '1':BEGIN
                IF luzB>=MAX THEN WRITELN('biblioteka beterik')
                ELSE LiburuBerriBatBibliotekanSartu(B,luzB);
               END;
           '2':BEGIN
                WRITE('Kodea:  ');
                READLN(libKode);
                BibliotekanZegoenLiburuBatKendu(B,luzB,libKode);
               END;
           '3':BEGIN
                 WRITE('Kodea:  ');
                 READLN(libKode);
                 indize:=LiburuaBibliotekanBilatu(B,luzB,libKode);
                 BibliotekakoLiburuBatUtzi(B,luzB,U,luzU,libKode,indize);
               END;
           '4':BEGIN
                 WRITE('Kodea:  ');
                 READLN(libKode);
                 indize:=LiburuaBibliotekanBilatu(B,luzB,libKode);
                 IF indize=0 THEN WRITELN('Errore, liburua ez dago')
                 ELSE
                 BEGIN
                   B[indize].itzul_eguna:=0;
                   indize:=UtzitakoLiburuenZerrendanBilatu(U,luzU,libKode);
                   IF indize=0 THEN WRITELN('Liburu ez dago utzita')
                   ELSE LiburuaUtzitakoenZerrendatikAtera(U,luzU,indize);
                 END
               END;
           '5':BEGIN
                 BibliotekarenEdukiaErakutsi(B,luzB);
               END;
           '6':BEGIN
                 irizpide:=OrdenatzekoIrizpideaAukeratu;
                 CASE irizpide OF
                   '1': IzenezOrdenatu(U,luzU);
                   '2': EgunezOrdenatu(U,luzU);
                 END;
                 UtzitakoenZerrendaErakutsi(U,luzU);
               END;
         END;
     UNTIL aukera='0';
END.
```

`Biblioteka_Kudeaketa` programa hau hobeto ulertzeko eskema bat erakusten da hurrengo orrialdean.



11.2.6 Erregistro baten hasieraketa

Hamargarren kapituluan arrayak azaldu zirenean aldagaiak definitzean hasieraketa egin daitekeela[9] esan genuen.

Aldagai eta konstanteen arteko diferentzia laugarren kapituluan ikusi zen lehen aldiz, labur esanda konstanteak duen balioa ezin daiteke aldatu programa barruan. Ordutik hona konstanteak deklaratzeko `CONST` hitz erreserbatuaren atalean konstante bakoitzak gordeko duen balioa zehaztuz egin dugu, eta, aldagaiak berriz `VAR` blokean erazagutu egin ditugu baina balioak eman ordez aldagai bakoitzari dagokion datu-mota idatziz[10].

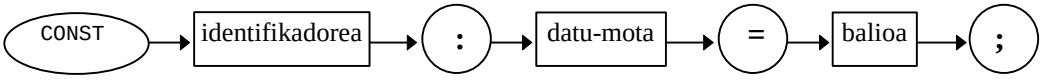
Baina Turbo Pascal lengoaian aldagaiak `CONST` bloke barruan deklara daitezke ere, guk ez dugu aukera hori erabili duen abantaila bakarra aldagai ez-egituratuekin interesgarria ez delako. Hots, `CONST` blokean deklaraturiko aldagaiari bertan egiten zaio hasieraketa:

```
CONST
   Adina : Integer = 27 ;         { Integer datu-motako aldagaia }
   Soldata : Real = 4.3 ;         { Real datu-motako aldagaia }
   Ezkondua : Boolean = FALSE ;   { Boolean datu-motako aldagaia }
   Kategoria : Char = 'B' ;       { Char datu-motako aldagaia }
```

Nahiz eta lau identifikadore horiek `CONST` hitz erreserbatuari dagokion atalean definitu izan, ez dira konstanteak aldagaiak baizik, jarraian ematen den programan ikus daitekeenez:

```
PROGRAM AldagaienHasieraketa ;                  { \TP70\11\HASIERAK.PAS }
CONST
   Adina : Integer = 27 ;
   Soldata : Real = 4.3 ;
   Ezkondua : Boolean = FALSE ;
   Kategoria : Char = 'B' ;
BEGIN
   WriteLn ('Balioak hasieran:') ;
   Write ('Adina:', Adina, '     Soldata:', Soldata:0:2, '     ') ;
   WriteLn ('Ezkondua:', Ezkondua:5, '     Kategoria:', Kategoria) ;

   Adina := 28 ;
   Soldata := 4.5 ;
   Ezkondua := TRUE ;
   Kategoria := 'A' ;

   WriteLn ('Balioak amaieran:') ;
   Write ('Adina:', Adina, '     Soldata:', Soldata:0:2, '     ') ;
   WriteLn ('Ezkondua:', Ezkondua:5, '     Kategoria:', Kategoria) ;
END.
```

Konstanteen blokean deklaraturik egon arren `Adina`, `Soldata`, `Ezkondua` eta `Kategoria` aldagaiak dira, horregatik `AldagaienHasieraketa` programan balio berriak har ditzakete. Hauxe dira hasieraketa integraturik duen aldagai baten deklarazioaren sintaxia, eta `AldagaienHasieraketa` programaren irteera:




```
Balioak hasieran:
Adina:27     Soldata:4.30     Ezkondua:FALSE     Kategoria:B
Balioak amaieran:
Adina:28     Soldata:4.50     Ezkondua: TRUE     Kategoria:A
_
```

[9] Posibilitate hau Turbo Pascal lengoaiaren berezitasuna da eta ez da Pascal estandarrean onartzen.
[10] Aldagaiek izaten duten hasieraketa sententzien atalean gertatzen da (`BEGIN` eta `END` artean).

Aurrerago esandakoarekin jarraituz, erregistro baten hasieraketa egiteko bi bide urratu ahal ditugu. Batetik edozein aldagai bezala sententzien atalean balioa esleituz, eta bestela erregistro aldagaia `CONST` blokean deklaratuz:

```
PROGRAM ErregistroenHasieraketa ;                { \TP70\11\RECORD10.PAS }
CONST
   AZTERKOPURU = 4 ;
TYPE
   DM_Azterk = ARRAY[1..AZTERKOPURU] OF Real ;
   DM_Ikasle = RECORD
                   Izena : String[19] ;
                   Adina : Byte ;
                   Notak : DM_Azterk ;
               END ;
CONST
   IkasleCON : DM_Ikasle = ( Izena : 'Patxi Garzia' ;
                             Adina : 19 ;
                             Notak : (2.1, 7.2, 6.8, 3.4)
                           ) ;
VAR
   IkasleVAR : DM_Ikasle ;
   Kont : Byte ;
BEGIN
   WriteLn ('IkasleCON aldagaia hasieratuta dago') ;

   WriteLn ('IkasleVAR aldagaia hasieratzen ari da') ;
   IkasleVAR.Izena := 'Laura Andion' ;
   IkasleVAR.Adina := 20 ;
   Randomize ;
   FOR Kont:=1 TO AZTERKOPURU DO
      IkasleVAR.Notak[kont] := Random * 10 ;

   WriteLn ('IkasleCON aldagaiaren balioak:') ;
   Write (IkasleCON.Izena :20) ;
   Write (IkasleCON.Adina :10) ;
   FOR Kont:=1 TO AZTERKOPURU DO
      Write (IkasleCON.Notak[Kont]:10:1) ;
   WriteLn ;

   WriteLn ('IkasleVAR aldagaiaren balioak:') ;
   Write (IkasleVAR.Izena :20) ;
   Write (IkasleVAR.Adina :10) ;
   FOR Kont:=1 TO AZTERKOPURU DO
      Write (IkasleVAR.Notak[Kont]:10:1) ;
   WriteLn ;
END.
```

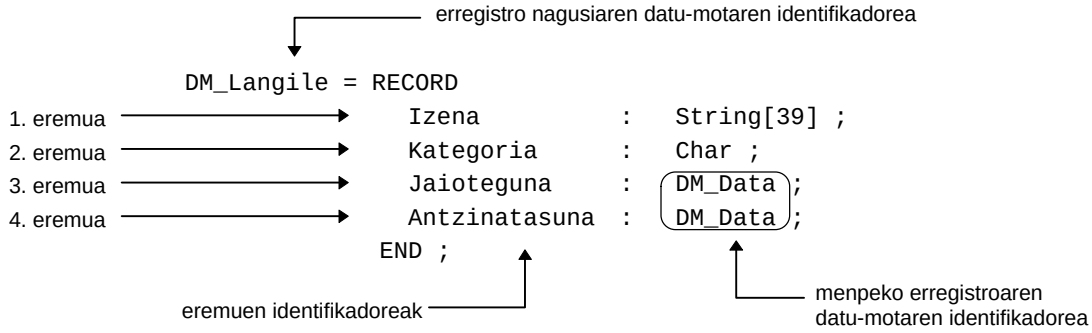
`ErregistroenHasieraketa` programan konstanteen deklarazioaren bloke bi daude, batean `AZTERKOPURU` benetako konstantea erazagutzen da eta bigarrenean `IkasleCON` aldagaia. `IkasleCON` erregistro aldagaiaren hasieraketa egiteko lehenago erakutsi den sintaxia jarraitu behar da, non balioak emateko parentesiz mugaturiko bloke bat idazten den eta eremuen balioak eman ondoren eremuak puntu eta komaz banatzen dira (arrayari dagozkion balioak nola ematen diren gogoratzeko **10.2.2 Array dimentsiobakarra den aldagai baten hasieraketa** puntua berrirakurri).

`ErregistroenHasieraketa` programaren exekuzioan `IkasleVAR` aldagaiaren notak `Random` funtzioaren araberako balioak ematen zaizkio, eta `ErregistroenHasieraketa`-ren irteera posible bat hauxe izan daiteke:

```
IkasleCON aldagaia hasieratuta dago
IkasleVAR aldagaia hasieratzen ari da
IkasleCON aldagaiaren balioak:
        Patxi Garzia        19       2.1       7.2       6.8       3.4
IkasleVAR aldagaiaren balioak:
        Laura Andion        20       8.5       7.3       4.1       6.6
_
```

11.2.7 Erregistro hierarkikoak

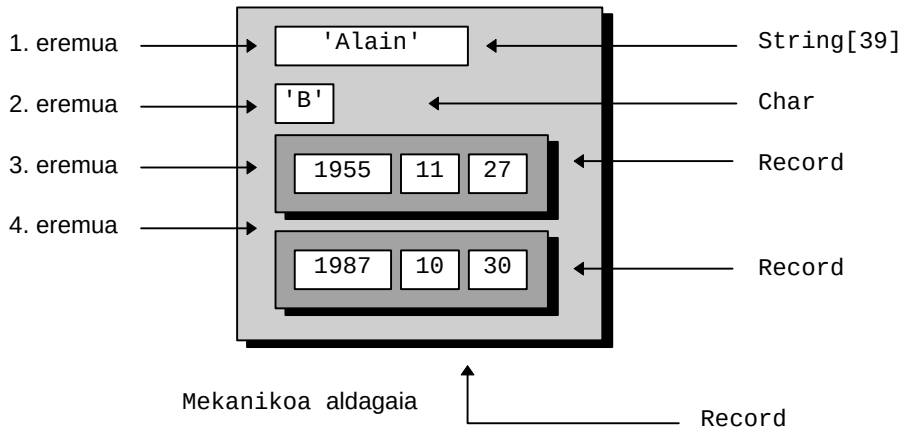
Erregistro bat definitzean bere eremuen datu-motak zehaztu behar izaten da, eta lehenago ikusi dugun bezala erregistro batek bere barnean beste erregistro bat izan dezake. Hau da, erregistro nagusiaren eremuren bati dagokion datu-mota `Record` dela. Adibidez langile baten adierazpide logikoa gauzatzeko bere jaioteguna eta antzinatasuna gordetzea nahi badugu data kontzeptua interesgarria izan daiteke:



Antolaketa hau burutu ahal izateko konpiladoreak exigitzen duen gauza bakarra `DM_Langile` datu-mota definitu baino lehen `DM_Data` datu-mota ezaguna izatea da, hots, `DM_Langile` datu-mota definitu ordurako `DM_Data` datu-mota definiturik egongo da:

```
TYPE
   DM_Data = RECORD
                Urtea : Integer ;
                Hilea, Eguna : Byte ;
             END ;
   DM_Langile = RECORD
                Izena : String[39] ;
                Kategoria : Char ;
                Jaioteguna, Antzinatasuna : DM_Data ;
              END ;
VAR
   Mekanikoa : DM_Langile ;
```

`DM_Langile` eta `DM_Data` datu-moten erlazio hierarkikoa hobeto ulertzeko `Mekanikoa` aldagaiari dagokion eskema egin dezagun:



`Mekanikoa` aldagaiarekin lan egiteko erregistro bat dela kontutan izango dugu, beraz eremuetan balioak gordetzeko aldagaiaren eta eremuen identifikadoreak puntu eragilearen

bitartez konektatuko ditugu. Esandako honek balio digu lehendabiziko mailaketarako (`Izena` eta `Kategoria` eremuetarako) baina ez bigarren mailan aurkitzen diren eremuetarako, izan ere langilea jaio zen hilebetea adierazteko ezinezkoak dira `Mekanikoa.Jaoiteguna` eta `Mekanikoa.Hilea` erreferentziak, zergatik?.

`Mekanikoa.Jaoiteguna` erabiltzean ez dugu langilearen zenbaki osoa den hilebetea errepresentatzen, erreferentzia horri dagokion datu-mota erregistro bat delako. Eta erregistro bat izanik onartzen duen eragiketa bakarra esleipena denez, programa konpilatzean agertuko zaigun errorerik normalena `Mekanikoa.Jaoiteguna` erregistroari aplikaturiko eragiketekin zerikusia duena izango litzateke.

`Mekanikoa.Hilea` idaztean ez dugu langilearen hilebetea errepresentatzen, konpilazio-errorea agertuko litzateke `Mekanikoa` aldagaiari ezagutzen zaizkion lau eremuetatik ez dago `Hilea` deitzen denik. Ondorioz konpiladoreak mezu egokia den hau agertuko liguke: `Error 44: Field identifier expected.`

Beraz, `Mekanikoa` aldagaiarekin lan egiteko lehendabiziko mailaketarako (`Izena`, `Kategoria`, `Jaioteguna` eta `Antzinatasuna` eremuetarako) puntu bakarra jarriko dugu; baina bigarren mailan aurkitzen diren `Urtea`, `Hilea` eta `Eguna` eremuetarako beste puntu bat idatzi beharra dago. Ikus `ErregistroKabiatuak` programa:

```
PROGRAM ErregistroKabiatuak ;                    { \TP70\11\RECORD11.PAS }
TYPE
   DM_Data = RECORD
               Urtea : Integer ;
               Hilea : Byte ;
               Eguna : Byte ;
             END ;
   DM_Langile = RECORD
                   Izena         : String[39] ;
                   Kategoria     : Char ;
                   Jaioteguna    : DM_Data ;
                   Antzinatasuna : DM_Data ;
                END ;


PROCEDURE ErregistroNagusiaBete (VAR ErregNag : DM_Langile) ;
BEGIN
   Write ('Langilearen izena eman:        ') ;
   ReadLn (ErregNag.Izena) ;

   Write ('Langilearen kategoria eman:    ') ;
   ReadLn (ErregNag.Kategoria) ;

   Write ('Langilearen jaio urtea eman:   ') ;
   ReadLn (ErregNag.Jaioteguna.Urtea) ;
   Write ('Langilearen jaio hilea eman:   ') ;
   ReadLn (ErregNag.Jaioteguna.Hilea) ;
   Write ('Langilearen jaio eguna eman:   ') ;
   ReadLn (ErregNag.Jaioteguna.Eguna) ;

   Write ('Enpresan sartu zen urtea eman: ') ;
   ReadLn (ErregNag.Antzinatasuna.Urtea) ;
   Write ('Enpresan sartu zen hilea eman: ') ;
   ReadLn (ErregNag.Antzinatasuna.Hilea) ;
   Write ('Enpresan sartu zen eguna eman: ') ;
   ReadLn (ErregNag.Antzinatasuna.Eguna) ;
END;
```

```
PROCEDURE ErregistroNagusiaIkus (CONST ErregNag : DM_Langile) ;
BEGIN
   Write (ErregNag.Izena) ;
   Write (ErregNag.Kategoria :10) ;
   WriteLn ;
   Write (ErregNag.Jaioteguna.Urtea) ;
   Write ('-', ErregNag.Jaioteguna.Hilea) ;
   Write ('-', ErregNag.Jaioteguna.Eguna) ;
   WriteLn ;
   Write (ErregNag.Antzinatasuna.Urtea) ;
   Write ('-', ErregNag.Antzinatasuna.Hilea) ;
   Write ('-', ErregNag.Antzinatasuna.Eguna) ;
   WriteLn ;
END;


VAR
   Mekanikoa : DM_Langile ;

BEGIN
   WriteLn ('Mekanikoa aldagaia datuz betetzen') ;
   WriteLn ('--------------------------------') ;
   ErregistroNagusiaBete (Mekanikoa) ;
   WriteLn;
   WriteLn ('Mekanikoa aldagaiaren edukia') ;
   WriteLn ('----------------------------') ;
   ErregistroNagusiaIkus (Mekanikoa) ;
END.
```

Programaren `ErregistroNagusiaBete` eta `ErregistroNagusiaIkus` prozedurak aztertzen baditugu parametroaren identifikadorea den `ErregNag` etiketa etengabe agertzen zaigu erregistro horrekin lan egitean. Errepikapen horiek aspergarriak direlakoan, eta erregistro kabiatuen barneen dauden eremuetara iristeko erreferentziak luzeegiak gertatzen direlako Pascal lengoaiak `With` sententzia eskaintzen du. `With` sententzia honela formulatzen da: *"hurrengo sententzian (sententzia bakuna ala konposatua[11]) erregistroaren[12] eremuak zuzenki atzitu ahal dira"*.

Honezkero `ErregistroNagusiaIkus` prozedura honela idatz zitekeen, non `ErregNag` identifikadorea `WITH` sententziaren hasieran jarri ondoren ez da gehiagorik agertzen:

```
PROCEDURE ErregistroNagusiaIkus (CONST ErregNag : DM_Langile) ;
BEGIN
   WITH ErregNag DO
   BEGIN
      Write (Izena) ;
      Write (Kategoria :10) ;
      WriteLn ;
      Write (Jaioteguna.Urtea) ;
      Write ('-', Jaioteguna.Hilea) ;
      Write ('-', Jaioteguna.Eguna) ;
      WriteLn ;
      Write (Antzinatasuna.Urtea) ;
      Write ('-', Antzinatasuna.Hilea) ;
      Write ('-', Antzinatasuna.Eguna) ;
      WriteLn ;
   END;
END;
```

[11] `With` sententziak `Begin` eta `End` mugatzaileak onartzen baititu.
[12] `With` sententziaren hasieran zehazten da zein erregistroz ariko garen.

11.2.7.1 Adibidea

`WITH` sententzia, nagusiki, erregistro hierarkikoekin erabiltzen da; eta `WITH` sententzia bat beste `WITH` sententzia baten barnean kabia daiteke. `ErregistroKabiatuak` programa aldatuz `WITH` kabiatuak idatziko ditugu.

Hona hemen `WithKabiatuak` deituko dugun programa:

```
PROGRAM WithKabiatuak ;                          { \TP70\11\RECORD12.PAS }
TYPE
   DM_Data = RECORD
               Urtea : Integer ;
               Hilea : Byte ;
               Eguna : Byte ;
             END ;
   DM_Langile = RECORD
                   Izena         : String[39] ;
                   Kategoria     : Char ;
                   Jaioteguna    : DM_Data ;
                   Antzinatasuna : DM_Data ;
                END ;

PROCEDURE ErregistroNagusiaBete (VAR ErregNag : DM_Langile) ;
BEGIN
  WITH ErregNag DO
  BEGIN
     Write ('Langilearen izena eman:        ') ;
     ReadLn (Izena) ;
     Write ('Langilearen kategoria eman:    ') ;
     ReadLn (Kategoria) ;
     WITH Jaioteguna DO
     BEGIN
        Write ('Langilearen jaio urtea eman:   ') ;
        ReadLn (Urtea) ;
        Write ('Langilearen jaio hilea eman:   ') ;
        ReadLn (Hilea) ;
        Write ('Langilearen jaio eguna eman:   ') ;
        ReadLn (Eguna) ;
     END;
     WITH Antzinatasuna DO
     BEGIN
        Write ('Enpresan sartu zen urtea eman: ') ;
        ReadLn (Urtea) ;
        Write ('Enpresan sartu zen hilea eman: ') ;
        ReadLn (Hilea) ;
        Write ('Enpresan sartu zen eguna eman: ') ;
        ReadLn (Eguna) ;
     END;                  { WITH Antzinatasuna DO }
  END;                               { WITH ErregNag DO }
END;



PROCEDURE ErregistroNagusiaIkus (CONST ErregNag : DM_Langile) ;
BEGIN
   WITH ErregNag, Jaioteguna DO
   BEGIN
      Write (Izena) ;
      Write (Kategoria :10) ;
      WriteLn ;
      Write (Urtea, '-') ;
      Write (Hilea, '-') ;
      Write (Eguna) ;
      WriteLn ;
   END;
```

2. mailarako konektorea

```
WITH ErregNag.Antzinatasuna DO
   BEGIN
      Write (Urtea, '-') ;
      Write (Hilea, '-') ;
      Write (Eguna) ;
      WriteLn ;
   END;
END;



VAR
   Mekanikoa : DM_Langile ;

BEGIN
   WriteLn ('Mekanikoa aldagaia datuz betetzen') ;
   WriteLn ('--------------------------------') ;
   ErregistroNagusiaBete (Mekanikoa) ;
   WriteLn;
   WriteLn ('Mekanikoa aldagaiaren edukia') ;
   WriteLn ('----------------------------') ;
   ErregistroNagusiaIkus (Mekanikoa) ;
END.
```

Aurreko programan `ErregistroNagusiaBete` eta `ErregistroNagusiaIkus` izeneko bi prozedura erabiltzen dira eta bakoitzean `WITH` sententziak agertzen dira, baina era ezberdinez idatzi dira beharrekoak diren kabiaketak:

```
WITH ErregNag DO
   WITH Jaioteguna DO
   BEGIN
      Write (Izena) ;
      Write (Kategoria :10) ;
      WriteLn ;
      Write (Urtea, '-') ;
      Write (Hilea, '-') ;
      Write (Eguna) ;
      WriteLn ;
   END;
```

```
WITH ErregNag, Jaioteguna DO
   BEGIN
      Write (Izena) ;
      Write (Kategoria :10) ;
      WriteLn ;
      Write (Urtea, '-') ;
      Write (Hilea, '-') ;
      Write (Eguna) ;
      WriteLn ;
   END;
```

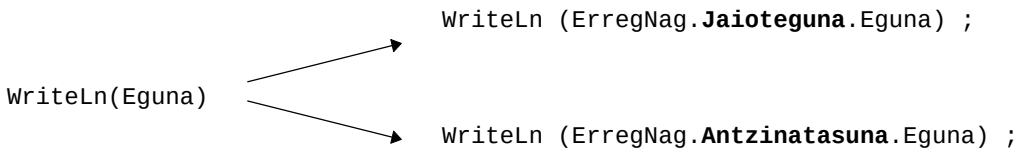
Ezkerreko kabiaketa `ErregistroNagusiaBete` prozeduran erabili da eta bere ulermena ez du inolako zailtasunik suposatzen. Eskuinekoa berriz trinkoagoa da eta erregistroak diren bi etiketak agertzen dira, `ErregistroNagusiaIkus` prozeduran `ErregNag` erregistro-aldagaia erabiltzen da (lehendabiziko etiketa), eta bigarren mailaketarako bidea emateko erregistroa den eremuaren `Jaioteguna` identifikadorea jartzen da ostean.

11.2.7.2 Adibidea

Baina kontura gaitezen `WithKabiatuak` deituriko programan zalantzako egoera bat gertatzen zaigula honelako `WITH` sententzia formulatzean:

```
WITH ErregNag, Jaioteguna, Antzinatasuna DO
   BEGIN
      WriteLn ('Eguna  ---> ', Eguna) ;
END;
```

Hots, `WITH` horren barnean `Eguna`, `Hilea` eta `Urtea` identifikadoreen erreferentziak egitean, adibidez `WriteLn(Eguna)` bezelako sententzia idaztean, ez da argi geratzen zertan ari garen. Izan ere, nola ulertu `WriteLn(Eguna)` sententzia?



Ezaguna da erregistro baten eremuak identifikatzeko etiketak ezin direla maila jakin batean errepikatu, erregistro eremuen etiketak bikoiztu egiten direnean (maila ezberdinetan daudenean posible da) erreferentzi trinkoak errazten dituen `WITH` sententziak zalantzak eragiten ditu. Arau bezala kontutan izan `WITH` bateko identifikadore zerrendan *eskuinerago dauden etiketak lehentasuna dutela ezkerrago daudenen kalterako*.

Ikus `WithKabiatuakZalantzako` programa non `ErregistroNagusiaIkus` prozedura honela idatzi den:

```
PROGRAM WithKabiatuakZalantzazko ;              { \TP70\11\RECORD13.PAS }
TYPE
   DM_Data = RECORD
               Urtea : Integer ;
               Hilea : Byte ;
               Eguna : Byte ;
             END ;
   DM_Langile = RECORD
                   Izena         : String[39] ;
                   Kategoria     : Char ;
                   Jaioteguna    : DM_Data ;
                   Antzinatasuna : DM_Data ;
                END ;


PROCEDURE ErregistroNagusiaIkus (CONST ErregNag : DM_Langile) ;
BEGIN
   WITH ErregNag, Jaioteguna, Antzinatasuna DO
   BEGIN
      Write (Izena) ;
      Write (Kategoria :10) ;
      WriteLn ;                        { Jaioteguna erakutsi nahian }
      Write (Urtea, '-') ;
      Write (Hilea, '-') ;
      Write (Eguna) ;
      WriteLn ;                        { Antzinatasuna erakutsi nahian }
      Write (Urtea, '-') ;
      Write (Hilea, '-') ;
      Write (Eguna) ;
      WriteLn ;
   END;
END;
```

`WithKabiatuakZalantzako` programa egikarituz:

```
Mekanikoa aldagaia datuz betetzen
---------------------------------
Langilearen kategoria eman:    Pedro
Langilearen kategoria eman:    M
Langilearen jaio urtea eman:   59
Langilearen jaio hilea eman:   11
Langilearen jaio eguna eman:   11
Enpresan sartu zen urtea eman: 80
Enpresan sartu zen hilea eman: 12
Enpresan sartu zen eguna eman: 12
```

```
Mekanikoa aldagaiaren edukia
----------------------------
Pedro         M
80-12-12
80-12-12
_
```

Programaren irteeran ikus daitekeenez, ezinezkoa da `ErregNag` aldagaiari dagokion `Jaioteguna` eremuaren datuak kanporatu adibidearen `WITH-DO` horren bitartez, eta ez da nahikoa `Jaioteguna` eta `Aintzinatasuna` eremuen zerrendan orden erlatiboa trukatu. Halako egoeretan lehentasuna ez duten eremuen erreferentzia osoak egin behar izaten dira:

```
PROCEDURE ErregistroNagusiaIkus (CONST ErregNag : DM_Langile) ;
BEGIN
   WITH ErregNag, Jaioteguna, Antzinatasuna DO
   BEGIN
      Write (Izena) ;
      Write (Kategoria :10) ;
      WriteLn ;
      Write (Jaioteguna.Urtea, '-') ;
      Write (Jaioteguna.Hilea, '-') ;
      Write (Jaioteguna.Eguna) ;
      WriteLn ;
      Write (Urtea, '-') ;       { Antzinatasuna.   ez da derrigorrezkoa }
      Write (Hilea, '-') ;       { Antzinatasuna.   ez da derrigorrezkoa }
      Write (Eguna) ;            { Antzinatasuna.   ez da derrigorrezkoa }
      WriteLn ;
   END;
END;
```

11.2.8 Erregistro aldakorrak

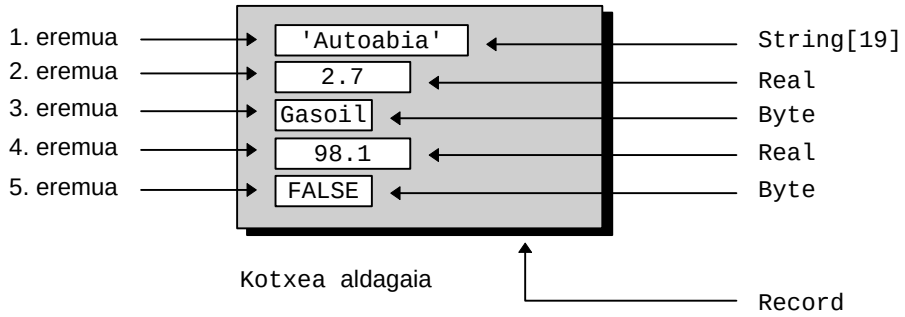
Orain arte ikusi ditugun erregistro datu-motek erregistro finkoak definitzen zituzten, orain arteko erregistro datu-mota batetik ondoriotzen ziren aldagaiek eremu kopuru berbera zuten. Esate baterako:

```
TYPE
   DM_Erregaia = (Gasolina, Gasoil) ;
   DM_Kotxea = RECORD
                  Marka     : String[19] ;
                  Prezioa   : Real ;
                  Modeloa   : DM_Erregaia ;
                  Oktanoak  : Real ;
                  Furgoneta : Boolean ;
               END ;
VAR
   Kotxea : DM_Kotxea ;
```

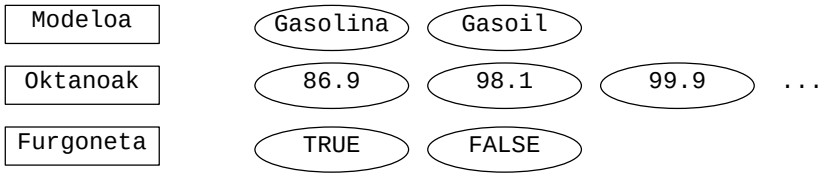
11.2.2.3 Eremuen biltegitzea memorian izenburuko puntuan ikusi izan dugun bezala `Kotxea` aldagaiak 34 byte hartuko lituzke:

```
Marka     : String[19] ;        20 byte
Prezioa   : Real ;               6 byte
Modeloa   : DM_Erregaia ;        1 byte
Oktanoak  : Real ;               6 byte
Furgoneta : Boolean ;            1 byte
                                 ---------
                                 34 byte
```

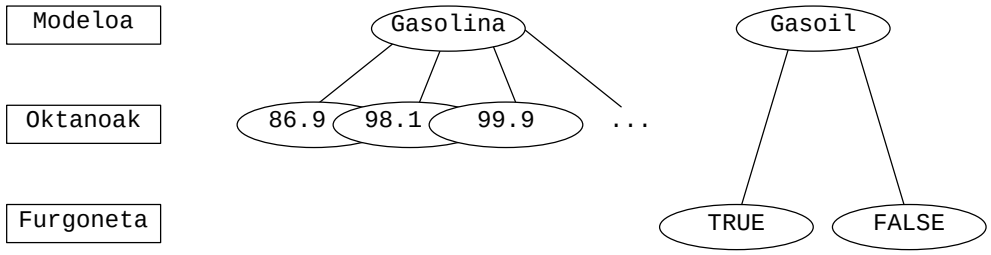
Zenbait balio jaso ondoren bere eskema honako hau delarik:



Non `Kotxea` aldagaian gordetako informazioaren arabera, besteak beste hau dakigu: `Gasoil`-eko modeloa dela baina ez da furgoneta. Eremuen arteko akoplamendurik ez dagoenean, eremu bakoitzak informazio zehatz eta banatu bat ematen duenean, goian erakutsi dugun bezalako erregistro “normalak” erabiltzen dira. Azken hiru eremuak aintzat harturik euren balio posibleak honako hauek izan daitezke:

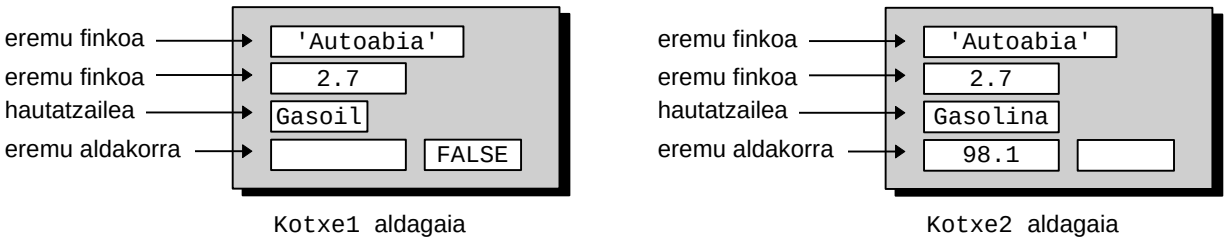


Baina demagun aldez aurretik dakigula, ariketa batean, gasolinako furgonetarik ez dela esistitzen eta bestalde gasoileko ibilgailu baterako bere oktanajeak garrantzirik ez duela. Beste modu batez esanik `Modeloa`, `Oktanoak`, eta `Furgoneta` azken hiru eremuak elkar erlazionaturik daudela, zuhaitz honek adierazten duen bezala:



Holako egoeretan “normalak” deitu ditugun baino trinkoagoak diren erregistro *aldakorrak* erabil daitezke, ondorioz erregistro datu-mota bereko aldagaiek egitura ezberdina izan dezakete eremu baten balioaren arabera. Eremu honi *hautatzaile* esaten zaio eta adibidearen kasuan `Modeloa` izango litzateke. `Modeloa`-k `Gasolina` balio duenean aldagaia `Oktanoak` eremua izango du, baina `Modeloa`-k `Gasoil` balio duenean aldagaiak `Furgoneta` eremua izango du.

Hasierako eremuak (`Marka` eta `Prezioa`) edozein erregistrorako amankomunak direnez, datu-mota beratik ondoriotzen diren honako egiturako aldagaiak egon daitezke:



Eremuen jatorria kontsideratuz `Kotxe1` eta atuz `Kotxe2` aldagaiek memorian hartuko luketen byte kopurua 28 eta 33 lirateke:

	Kotxe1		Kotxe2
`Marka : String[19] ;`	20 byte	`Marka : String[19] ;`	20 byte
`Prezioa : Real ;`	6 byte	`Prezioa : Real ;`	6 byte
`Modeloa : DM_Erregaia ;`	1 byte	`Modeloa : DM_Erregaia ;`	1 byte
`Furgoneta : Boolean ;`	1 byte	`Oktanoak : Real ;`	6 byte
	28 byte		33 byte

Non `Modeloa` erregistro hautatzailearen balioaren arabera bi eremu aldakorretik batek balioa duen. Memorian ez da `Oktanoak` eta `Furgoneta` erregistro aldakor bientzat tokirik erreserbatzen, erregistro aldakor zabalenak beharko duen byte kopurua hartzen da (gure adibidean 6 byte) eta memoriaren posizio horietan gordetzen da informazioa (balio boolear bat `Kotxe1` aldagaiaren kasuan eta zenbaki erreal bar `Kotxe2` erregistro aldagairen kasuan).

Beraz, `Modeloa` erregistro hautatzailearen balioaren arabera bi eremu aldakorretik batek balioa du. `Kotxe1` eta `Kotxe2` erregistro aldagaiei dagokien datu-mota bakarra definitzeko Pascal lengoaiak sintaxi berezia darabil:

```
TYPE
   DM_Erregaia = (Gasolina, Gasoil) ;
   DM_Kotxea = RECORD
                  Marka     : String[19] ;
                  Prezioa   : Real ;
                  CASE Modeloa : DM_Erregaia OF
                    Gasolina : (Oktanoak : Real) ;
                    Gasoil   : (Furgoneta : Boolean) ;
                END ;
VAR
   Kotxe1, Kotxe2 : DM_Kotxea ;
```

Jarraian ikus daitekeen `ErregistroAldakorra0` programan `DM_Kotxea` eta `DM_Autoa` datu-motaka erazagutu ondoren memorian hartzen duten byte kopurua aztertzen da.

```
PROGRAM ErregistroAldakorrak0 ;                { \TP70\11\RECORD014.PAS }
TYPE
  DM_Erregaia = (Gasolina, Gasoil) ;

  DM_Kotxea = RECORD
                 Marka     : String[19] ;
                 Prezioa   : Real ;
                 Modeloa   : DM_Erregaia ;
                 Oktanoak  : Real ;
                 Furgoneta : Boolean ;
              END ;

  DM_Autoa =  RECORD
                 Marka   : String[19] ;
                 Prezioa : Real ;
                 CASE Modeloa : DM_Erregaia OF
                     Gasolina : (Oktanoak  : Real) ;
                     Gasoil   : (Furgoneta : Boolean) ;
              END ;
BEGIN
   WriteLn ('DM_Kotxea datu-motaren byte kopurua: ', SizeOf (DM_Kotxea)) ;
   WriteLn ('DM_Autoa datu-motaren byte kopurua:  ', SizeOf (DM_Autoa)) ;
END.
```

```
DM_Kotxea datu-motaren byte kopurua: 34
DM_Autoa datu-motaren byte kopurua:  33
_
```

`ErregistroAldakorra0` programaren kodeari eta dagokion exekuzioari so eginez gauza bi azpimarratuko genituzke.

Batetik, esan bezala eremu aldakorrak dituen `DM_Autoa` datu-mota trinkoagoa da eremu isolatuak dituen `DM_Kotxea` datu-mota baino. Adibide-programa honetan datu-mota biren arteko diferentzia ez da handia, baina erregistroen arrayak ditugunean diferentzi txiki hori arrayaren osagaien kopuruagatik biderkatu beharko litzateke eta zenbait kasutan erregistro aldakorren erabilpena justifikaturik egongo litzateke (esate baterako erregistro aldakorrekin lan egiten duen duen `ErregistroAldakorrenArrayak` programa exekutatu).

Bestetik, eremu aldakorrak definitzeko sintaxiak arau jakinak ditu Pascal lengoaiak. Erregistroaren zati finkoa hasieran egongo da eta zati aldakorra bakarra izango da eta erregistroaren bukaeran kokatuko da, zati aldakorra `CASE-OF` baten bitartez definitzen da eta eremu hautatzailea datu-mota ordinal bat izan beharko da (`Integer` familiakoa, `Char`, `Boolean` edo datu-mota enumeratua).

11.2.8.1 Adibidea

Dakigunez erregistro datu-motaren eremuak elkar ondoan kokatzen dira memorian. Honezkero erregistro jakin baten eremu bakoitzari dagokion helbidea aurrekoaren jarraian datorrena izango dela frogatuta daukagu `ErregistroarenEremuakMemorian` programaren bitartez.

`ErregistroarenEremuakMemorian` programa berridatziz erregistro aldakor baten helbideak erakuts ditzan, jarraian ematen den `ErregistroAldakorrenHelbideak` izeneko programa lor genezake. Non berezitasunik nabarmenena `Oktanoak` eremuak duen helbidea eta `Furgoneta` eremuari dagokiona helbide berbera den:

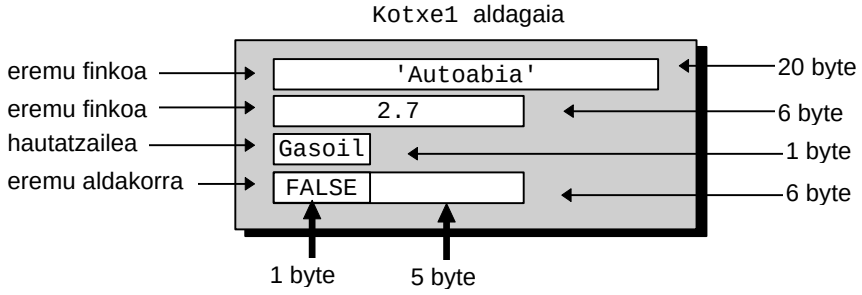
```
PROGRAM ErregistroAldakorrenHelbideak ;          { \TP70\11\RECORD016.PAS }
USES
    Crt, Ikus ;      (* IKUS.TPU  unitatea darabil *)
TYPE
  DM_Erregaia = (Gasolina, Gasoil) ;
  DM_Autoa = RECORD
                Marka : String[19] ;
                Prezioa : Real ;
                CASE Modeloa : DM_Erregaia OF
                    Gasolina : (Oktanoak : Real) ;
                    Gasoil   : (Furgoneta : Boolean) ;
             END ;
VAR
  Autoa : DM_Autoa ;
BEGIN              (* Programa Nagusia *)
   ClrScr ;

   WriteLn ('ERREGISTRO BATEN EREMUEN MEMORI HELBIDEAK') ;
   WriteLn ('========================================') ;
   WriteLn ;
   WriteLn ('PILARI dagokion segmentuaren hasiera:  ',IntToHex (sSeg)) ;
   WriteLn ('DATUEI dagokien segmentuaren hasiera:  ',IntToHex (dSeg)) ;
   WriteLn ('KODEARI dagokion segmentuaren hasiera: ',IntToHex (cSeg)) ;
   WriteLn ;
   WriteLn ;
   WriteLn ('ALDAGAIA':29, '       HELBIDEA', '   TAMAINA') ;
   WriteLn ('-----------------------------------------------':60) ;

   WriteLn ('Autoa ----> ':36, IntToHex (Seg (Autoa)),':',
            IntToHex (Ofs (Autoa)), '   (', SizeOf (Autoa), ')') ;
   WriteLn ;
```

```
  WriteLn ('Autoa.Furgoneta ----> ':36, IntToHex (Seg (Autoa.Furgoneta)),':',
           IntToHex (Ofs (Autoa.Furgoneta)), '   (', SizeOf (Autoa.Furgoneta), ')') ;

  WriteLn ('Autoa.Oktanoak -----> ':36, IntToHex (Seg (Autoa.Oktanoak)),':',
           IntToHex (Ofs (Autoa.Oktanoak)), '   (', SizeOf (Autoa.Oktanoak), ')') ;

  WriteLn ('Autoa.Modeloa ------> ':36, IntToHex (Seg (Autoa.Modeloa)),':',
           IntToHex (Ofs (Autoa.Modeloa)), '   (', SizeOf (Autoa.Modeloa), ')') ;

  WriteLn ('Autoa.Prezioa ------> ':36, IntToHex (Seg (Autoa.Prezioa)),':',
           IntToHex (Ofs (Autoa.Prezioa)), '   (', SizeOf (Autoa.Prezioa), ')') ;

  WriteLn ('Autoa.Marka --------> ':36, IntToHex (Seg (Autoa.Marka)),':',
           IntToHex (Ofs (Autoa.Marka)), '   (', SizeOf (Autoa.Marka), ')') ;

  WriteLn ;
  WriteLn ('Tamainak eremuka batuz gero  (':49,
            SizeOf(Autoa.Furgoneta) + SizeOf(Autoa.Oktanoak) +
            SizeOf(Autoa.Modeloa) + SizeOf(Autoa.Prezioa) + SizeOf(Autoa.Marka), ')') ;
END.
```

`ErregistroAldakorrenHelbideak` programa exekutatu ondoren, ordenadore zehatz batean irteera hau dela ziurtatzen badigute, emaitza horren interpretazioak ez digu arazorik sortzen helbideak bi zatitan (segmentua eta desplazamendua) ematen direla dakigulako. Baina zer suposatzen du `Oktanoak` eta `Furgoneta` eremuek helbide berbera izateak?:

```
ERREGISTRO BATEN EREMUEN MEMORI HELBIDEAK
=========================================

PILARI dagokion segmentuaren hasiera:  5A50
DATUEI dagokien segmentuaren hasiera:  5A20
KODEARI dagokion segmentuaren hasiera: 5947


                    ALDAGAIA       HELBIDEA   TAMAINA
               ------------------------------------------------
                         Autoa ----> 5A20:005E   (33)

               Autoa.Furgoneta ----> 5A20:0079   (6)
               Autoa.Oktanoak -----> 5A20:0079   (1)
               Autoa.Modeloa ------> 5A20:0078   (1)
               Autoa.Prezioa ------> 5A20:0072   (6)
               Autoa.Marka --------> 5A20:005E   (20)

                    Tamainak eremuka batuz gero  (34)
_
```

Eremu aldakor guztiek gordetzen duten informazioa memori posizio amankomunan batean biltegitzen da, eta exekuzio denboran interpretatzen da toki “konpartitu” horretan dagoena nola interpretatzen den (zenbaki erreala edo balio boolearra, adibide horretan). Irudi bat eginez erregistro aldakor batek honelako itxura izango luke:



11.2.8.2 Adibidea

Erregistroen eremu finkoei balioak ematean ez da arazorik sortzen eremu bakoitzak datu-mota bakarra duelako, baina eremu aldakorrek datu-mota ezberdinak izan daitezke eta balioak ematean dagokion bereizketa egin beharra dago.

Erregistroen eremuetan informazioa gordetzeko orduan agertzen zaigun egoera errepikatu egiten da eremuetan gordetakoa pantailaratu nahi denean. Hots, eremu aldakorra aldagai batetik bestera ezberdina izan daitekeelako prozesaketa berezia daramala.

Demagun zati aldakorra dituzten `Auto1` eta `Auto2` aldagaietan esleipenaren bitartez datuak gorde nahi direla. Zati finkoko eremuetan ez da zalantzarik sortzen, baina zati aldakorrean kopuru erreal bat (oktanajea) edo balio boolear bat (furgoneta den ala ez) esleitu behar zaio; horregatik autoaren modeloa zehaztea eskatzen da. Helburua, azken finean eta labur esanda, eremu aldakorrean balio bakar bat gordetza da.

`ErregistroAldakorrak1` programan eremuek gorde behar dituzten datuak asignatu egiten dira.

```
PROGRAM ErregistroAldakorrak1 ;                { \TP70\11\RECORD017.PAS }
USES
  Crt ;
TYPE
  DM_Erregaia = (Gasolina, Gasoil) ;
  DM_Autoa = RECORD
                Marka   : String[19] ;
                Prezioa : Real ;
                CASE Modeloa : DM_Erregaia OF
                    Gasolina : (Oktanoak  : Real) ;
                    Gasoil   : (Furgoneta : Boolean) ;
             END ;

PROCEDURE ErregistroakBete (VAR Auto1, Auto2 : DM_Autoa) ;
VAR
  Karakterea : Char ;
BEGIN
   Auto1.Marka := 'Auto___1' ;
   Auto1.Prezioa := 1.6 ;
   Auto2.Marka := 'Auto___2' ;
   Auto2.Prezioa := 2.6 ;

   WriteLn ('    Erregaia') ;
   WriteLn ('================') ;
   WriteLn ('  0   Gasolina') ;
   WriteLn ('  1   Gasoil') ;
   REPEAT
      Write ('"Auto1" autoaren modeloa eman: ') ;
      Karakterea := ReadKey ;
      Karakterea := UpCase (Karakterea) ;
      WriteLn (Karakterea) ;
   UNTIL (Karakterea='0') OR (Karakterea='1') ;
   CASE Karakterea OF
     '0' : BEGIN
              Auto1.Modeloa := Gasolina ;
              Auto1.Oktanoak := 98.1 ;
              Auto2.Modeloa := Gasoil ;
              Auto2.Furgoneta := FALSE ;
           END ;
     '1' : BEGIN
              Auto1.Modeloa := Gasoil ;
              Auto1.Furgoneta := FALSE ;
              Auto2.Modeloa := Gasolina ;
              Auto2.Oktanoak := 98.1 ;
           END ;
   END ;
END ;
```



```
PROCEDURE ErregistroakIkus (CONST Auto1, Auto2 : DM_Autoa) ;
VAR
  Indize : Byte ;
BEGIN
   WriteLn ('Marka':8, 'Prezioa':14, 'Oktanoak/Furgoneta':25) ;
   WriteLn ('=============================================') ;
   Write (Auto1.Marka) ;
   Write (Auto1.Prezioa:12:1) ;
   CASE Auto1.Modeloa OF
     Gasolina : Write (Auto1.Oktanoak:15:1) ;
     Gasoil   : Write (Auto1.Furgoneta:25) ;
   END ;
   WriteLn ;
   Write (Auto2.Marka) ;
   Write (Auto2.Prezioa:12:1) ;
   CASE Auto2.Modeloa OF
     Gasolina : Write (Auto2.Oktanoak:15:1) ;
     Gasoil   : Write (Auto2.Furgoneta:25) ;
   END ;
   WriteLn ;
END;


VAR
  Auto1, Auto2 : DM_Autoa ;
BEGIN
   ClrScr ;
   WriteLn ('Autoen datuak memorian gordetzeko prozesua:') ;
   ErregistroakBete (Auto1, Auto2) ;
   WriteLn ;
   WriteLn ('Hona hemen autoei dagokien informazioa:') ;
   ErregistroakIkus (Auto1, Auto2) ;
END.
```

Adibide programa honek egiten duena `Auto1` aldagaiaren modeloa teklatuz eskatu eta erantzunaren arabera datuak esleitu. Ikusten denez `Auto2` aldagaia `Auto1` aldagaiaren aurkako modelokoa da). Hona hemen `ErregistroAldakorrak1` programaren balizko irteera bat:

```
Autoen datuak memorian gordetzeko prozesua:
    Erregaia
=================
  0   Gasolina
  1   Gasoil
"Auto1" autoaren modeloa eman: 0

Hona hemen autoei dagokien informazioa:
   Marka       Prezioa       Oktanoak/Furgoneta
=============================================
Auto___1         1.6           98.1
Auto___2         2.6                    FALSE
_
```

Informazioaren pantailaraketa egiten denean `Auto1` eta `Auto2` aldagaien eremu hautatzaileen arabera oktanoak ala furgoneta erakutsiko da. Baina zer agertuko litzateke gasoileko auto batean oktanajea pantailaratzean?. Hurrengo adibidean erantzuten da.

11.2.8.3 Adibidea

Puntu honetan aurreko adibide-programari aldaketa txikiak egingo dizkiogu. Bi erregistroekin aritu beharrean `Autoa` aldagai bakarrarekin lan egingo dugu. `Autoa` aldagaiak bi

eremu aldakor eduki beharrean hiru izango ditu. `Autoa` aldagaiari balioak ematean esleipenak egin ordez teklatua erabiliko dugu. Eta garrantzitsuena, `Autoa` erregistroaren eremuek gordetzen dutena pantailaratzeko bi prozedura programatu dira `ErregistroaErakuts` izeneko errutinan `Autoa` aldagaiari dagokion modeloa aintzat hartzen da (hau litzateke erregistro aldakorrekin ontzat emango dugun prozesaketa mota), eta `ErregistroOsoaIkus` izeneko prozedura non `Autoa` erregistroaren eremu guztiek daukatena pantailaratzen den:

```
PROGRAM ErregistroAldakorrak2 ;               { \TP70\11\RECORD018.PAS }
USES
  Crt ;
TYPE
  DM_Erregaia = (Super, Gasoil, BerunikEz) ;
  DM_Autoa = RECORD
                Marka   : String[19] ;
                Prezioa : Real ;
                CASE Modeloa : DM_Erregaia OF
                    Super     : (Oktanoak  : Real) ;
                    Gasoil    : (Furgoneta : Boolean) ;
                    BerunikEz : () ;
             END ;


PROCEDURE ErregistroaBete (VAR Autoa : DM_Autoa) ;
VAR
  Karakterea : Char ;
BEGIN
   Write ('Autoaren marka eman:      ') ;
   ReadLn (Autoa.Marka) ;

   Write ('Autoaren salneurria eman: ') ;
   ReadLn (Autoa.Prezioa) ;

   WriteLn ('     Erregaia');
   WriteLn ('======================') ;
   WriteLn ('  S    Super') ;
   WriteLn ('  G    Gasoil') ;
   WriteLn ('  B    Berunik gabekoa') ;
   REPEAT
      Write ('Autoari dagokion modeloa eman: ') ;
      Karakterea := ReadKey ;
      Karakterea := UpCase (Karakterea) ;
      WriteLn (Karakterea) ;
   UNTIL (Karakterea='S') OR (Karakterea='G') OR (Karakterea='B') ;
   CASE Karakterea OF
     'S' : Autoa.Modeloa := Super ;
     'G' : Autoa.Modeloa := Gasoil ;
     'B' : Autoa.Modeloa := BerunikEz ;
   END ;

   CASE Autoa.Modeloa OF
     Super  : BEGIN
                 Write ('Zenbat oktano?     ') ;
                 ReadLn (Autoa.Oktanoak) ;
              END ;
     Gasoil : BEGIN
                Write ('Furgoneta da  (B/E)?   ') ;
                REPEAT
                   Karakterea := ReadKey ;
                   Karakterea := UpCase (Karakterea) ;
                   WriteLn (Karakterea) ;
                UNTIL (Karakterea='B') OR (Karakterea='E') ;
                IF Karakterea='B' THEN
                    Autoa.Furgoneta := TRUE
                ELSE
                    Autoa.Furgoneta := FALSE ;
              END ;
   END ;
END ;
```

```
PROCEDURE ErregistroaErakuts (CONST Autoa : DM_Autoa) ;
VAR
  Indize : Byte ;
BEGIN
   WriteLn ('Marka':10, 'Prezioa':12, 'Oktanoak/Furgoneta':25) ;
   WriteLn ('=============================================') ;
   Write (Autoa.Marka:10) ;
   Write (Autoa.Prezioa:10:1) ;
   CASE Autoa.Modeloa OF
     Super  :  Write (Autoa.Oktanoak:15:1) ;
     Gasoil :  Write (Autoa.Furgoneta:25) ;
   END ;
   WriteLn ;
END;


PROCEDURE ErregistroOsoaIkus (CONST Autoa : DM_Autoa) ;
VAR
  Indize : Byte ;
BEGIN
   WriteLn ('Marka':10, 'Prezioa':12, 'Oktanoak/Furgoneta':25) ;
   WriteLn ('---------------------------------------------') ;
   Write (Autoa.Marka:10) ;
   Write (Autoa.Prezioa:10:1) ;
   Write ('       ', Autoa.Oktanoak:10) ;
   Write (Autoa.Furgoneta:8) ;
   WriteLn ;
END;

VAR
  Autoa : DM_Autoa ;
BEGIN
   ClrScr ;
   WriteLn ('Autoaren datuak memorian gordetzeko prozesua:') ;
   ErregistroaBete (Autoa) ;
   WriteLn ;
   WriteLn ('Hona hemen autoari dagokion informazioa:') ;
   ErregistroaErakuts (Autoa) ;
   ErregistroOsoaIkus (Autoa) ;
END.
```

Programa honek duen berezitasuna `ErregistroaErakuts` eta `ErregistroOsoaIkus` prozeduretan datza. Egia esan `ErregistroaErakuts` izeneko errutina lehenago ikusi dugu eta egokiena dela aipatu izan da, `ErregistroOsoaIkus` prozeduraren akatsa ulertzeko demagun gasolinako auto bati dagozkion datuak ematen ditugula `ErregistroaBete` errutinan (adibidez: `AutoHigi` / `3.7` / `Gasolina` / `98.1`), baina ez denez `Furgoneta` eremuaren baliorik zehaztu, zer erakutsiko du `ErregistroOsoaIkus` prozedurak?. Hona hemen erantzuna:

```
Autoaren datuak memorian gordetzeko prozesua:
Autoaren marka eman:      AutoHigi
Autoaren salneurria eman: 3.7
     Erregaia
======================
  S    Gasolina
  G    Gasoil
  B    Berunik gabekoa
Autoari dagokion modeloa eman: S
Zenbat oktano?     98.1

Hona hemen autoari dagokion informazioa:
     Marka     Prezioa       Oktanoak/Furgoneta
=============================================
  AutoHigi       3.7           98.1
     Marka     Prezioa       Oktanoak/Furgoneta
---------------------------------------------
  AutoHigi       3.7        9.810E+01    TRUE
_
```

`ErregistroOsoaIkus` prozedurak `Furgoneta` eremuan `TRUE` dagoela eremuaren funtsa handirik gabe adierazten digu, nahiz eta guk eremu horretan ezer ez dugun gorde. `TRUE` horren zergatia ulertzeko onena `Autoa` aldagaiari beste balio ezberdinak ematea da, eremu finkoko datuek garrantzirik ez dute baina bai `Oktanoak` eremurako hautatuko duguna, adibidez demangun `0` oktano tekleatzen dugula. Hona hemen `ErregistroAldakorrak2` programaren irteera berria:

```
Autoaren datuak memorian gordetzeko prozesua:
Autoaren marka eman:      AutoHigi
Autoaren salneurria eman: 3.7
     Erregaia
=====================
  S    Gasolina
  G    Gasoil
  B    Berunik gabekoa
Autoari dagokion modeloa eman: S
Zenbat oktano?     0

Hona hemen autoari dagokion informazioa:
     Marka     Prezioa       Oktanoak/Furgoneta
=============================================
  AutoHigi       3.7            0.0
     Marka     Prezioa       Oktanoak/Furgoneta
---------------------------------------------
  AutoHigi       3.7        0.000E+00   FALSE
_
```

Irteera hori eta aurrekoa ulertzeko gogoratu `Autoa.Oktanoak` eta `Autoa.Furgoneta` eremuek memori posizioak amankomunak dituztela, eta balio boolear bat byte bakarrean gordetzen[13] dela. Bigarren irteeran ikusten denez eremu aldakorraren hasierako zortzi bitetan zeroak daudenean, esate baterako esleipen hau egin delako `Autoa.Oktanoak:=0;` (edo `0` oktano teklatuz eman delako), `Autoa.Furgoneta` eremua pantailaratzean `FALSE` agertuko da. Aurreneko irteeran `Autoa.Oktanoak` eremuan `98.1` gorde delako `Autoa.Furgoneta` eremua pantailaratzean `TRUE` agertuko da.

Erregistro aldakorrekin bukatzeko `ErregistroAldakorrak2` programaren aldaera bat prestatu dugu, non `Modeloa` eremu hautatzailea enumeratua izan beharrean karakterea den:

```
PROGRAM ErregistroAldakorrak3 ;               { \TP70\11\RECORD19.PAS }
TYPE
  DM_Autoa = RECORD
                Marka : String[19] ;
                Prezioa : Real ;
                CASE Modeloa : Char OF
                    'S' : (Oktanoak  : Real) ;
                    'G' : (Furgoneta : Boolean) ;
                    'B' : () ;
             END ;

PROCEDURE ErregistroaBete (VAR Autoa : DM_Autoa) ;

PROCEDURE ErregistroaIkus (CONST Autoa : DM_Autoa) ;

VAR
  Autoa : DM_Autoa ;
BEGIN
   WriteLn ('Autoaren datuak memorian gorde:') ;
   ErregistroaBete (Autoa) ;
   WriteLn ('Hona hemen autoari dagokion informazioa:') ;
   ErregistroaIkus (Autoa) ;
END.
```

[13] Aldagai boolear batek `FALSE` balio duenean memorian zero bat gordetzen da (`0000 0000` kode bitarrean). Aldagai boolearrak `TRUE` balio duenean memorian zero ez den zerbait gordetzen da.

11.2.8.4 Adibidea

Jarraian ematen den programa aztertuz bere irteera posibleak zeintzu izan daitezkeen asmatu. Zergatik erabili izan dira erregistro aldakorrak?

```
PROGRAM ErregistroAldakorrakAdib1 ;              { \TP70\11\ERREGI1.PAS }
USES
   Crt ;
TYPE
  KlaseaDM = (LaukiZuzen, Lauki) ;
  IrudiaDM = RECORD
     X0Non, Y0Non: Byte ;
     CASE Mota: KlaseaDM OF
        LaukiZuzen: (Oina, Altuera: Byte) ;
        Lauki: (Aldea: Byte) ;
  END ;

FUNCTION fIrudiaAukeratu : KlaseaDM ;
VAR
  irakurgai : Byte ;
BEGIN
   WriteLn ('        1    Laukizuzena') ;
   WriteLn ('        2    Laukia') ;
   Write ('Irudia hautatu: ') ;
   REPEAT
      ReadLn (irakurgai) ;
   UNTIL (irakurgai >= 1) AND (irakurgai <= 2) ;
   CASE irakurgai OF
      1 : fIrudiaAukeratu:=LaukiZuzen ;
      2 : fIrudiaAukeratu:=Lauki ;
   END ;
END ;

VAR
  Zerbait: IrudiaDM ;
  i, j, k: Byte ;
BEGIN
  ClrScr ;
  REPEAT
     Write ('Irudiaren hasierako posizioa (1 <= X0 <= 79): ') ;
     ReadLn (Zerbait.X0Non) ;
  UNTIL (Zerbait.X0Non >= 1) AND (Zerbait.X0Non <= 79) ;
  REPEAT
     Write ('Irudiaren hasierako posizioa (1 <= Y0 <= 24): ') ;
     ReadLn (Zerbait.Y0Non) ;
  UNTIL (Zerbait.Y0Non >= 1) AND (Zerbait.Y0Non <= 24) ;

  Zerbait.Mota := fIrudiaAukeratu ;
  CASE Zerbait.Mota OF
     LaukiZuzen : BEGIN
                    Write ('Oina eman: ') ;
                    ReadLn (Zerbait.Oina) ;
                    Write ('Altuera eman: ') ;
                    ReadLn (Zerbait.Altuera) ;
                    ClrScr ;
                    GotoXY (Zerbait.X0Non, Zerbait.Y0Non) ;
                    k := 1 ;
                    FOR i:=Zerbait.X0Non TO Zerbait.X0Non+Zerbait.Altuera-1 DO
                    BEGIN
                      FOR j:=1 TO Zerbait.Oina DO Write ('=') ;
                      GotoXY (Zerbait.X0Non, Zerbait.Y0Non+k) ;
                      k := k + 1 ;
                    END ;
                    GotoXY (Zerbait.X0Non, Zerbait.Y0Non) ;
                    WriteLn ('+') ;
                  END;
```

```
    Lauki : BEGIN
              Write ('Aldea eman: ') ;
              ReadLn (Zerbait.Aldea) ;
              ClrScr ;
              GotoXY (Zerbait.X0Non, Zerbait.Y0Non) ;
              k := 1 ;
              FOR i:=Zerbait.X0Non TO Zerbait.X0Non+Zerbait.Aldea-1 DO
              BEGIN
                 FOR j:=1 TO Zerbait.Aldea DO
                   Write ('=') ;
                 GotoXY (Zerbait.X0Non, Zerbait.Y0Non+k) ;
                 k := k + 1 ;
              END ;
              GotoXY (Zerbait.X0Non, Zerbait.Y0Non) ;
              WriteLn ('+') ;
            END;
  END ;
END.
```

`ErregistroAldakorrakAdib1` programaren datuak eman ondoren `ClrScr` prozedura dator, beraz pantaila garbitzen da. Demagun `ErregistroAldakorrakAdib1` programa exekutatzean honako irteera agertzen zaigula, zeintzu izan dira dagozkion sarrerak?

```
         +=========
         ==========
         ==========
         ==========
         ==========
         ==========
         ==========
         ==========
         ==========
         ==========
_
```

Irteeraren interpretazio zuzena egiteko kontutan izan koordenatu horizontalaren zabalera eta koordenatu bertikalarena ezberdinak direla.

Hau horrela da `ErregistroAldakorrakAdib1` izeneko programak testu-pantailan lan egiten duelako, horregatik erabiltzen dira hain zuzen ere `79` eta `24` konstanteak. Programa irakurgarriagoa izan dadin honelako hasiera idatz daiteke, non mugeen izenak explizitoki agertzen diren:

```
PROGRAM ErregistroAldakorrakAdib1 ;              { \TP70\11\ERREGI1.PAS }
USES
   Crt ;
CONST
   LERRO_MAX = 24 ;
   ZUTABE_MAX = 79 ;
TYPE
  KlaseaDM = (LaukiZuzen, Lauki) ;
  IrudiaDM = RECORD
     X0Non, Y0Non: Byte ;
     CASE Mota: KlaseaDM OF
        LaukiZuzen: (Oina, Altuera: Byte) ;
        Lauki: (Aldea: Byte) ;
  END ;
```

11.3 SET DATU-MOTA

Matematika, konputazioa eta gainerako zientzi askotan *multzo* kontzeptua agertzen da. Multzo batek elementuak biltzen ditu eta elementuen arteko ordenik ez da konsideratzen, adibidez jarraian ematen diren segidak multzoak dira:

Bokalen multzoa:	{ a, e, i, o, u }
Europako zenbait hiriren multzoa:	{ Limerick, Donostia, Bordele, Lisboa, Odessa }
Uraren osagaien multzoa:	{ oxigenoa, hidrogenoa }
Txakur hegalarien multzoa:	{ }

Gorago erabili izan den notazioa matematikoa da, matematikan multzoak adierazteko osagaiak giltzen artean jarri egiten dira, azkeneko kasua “txakur hegalarien multzoa” multzo hutsa da.

Konputazioaren mundura jauzi eginez, multzoen eta arrayen arteko ezberdintasuna azal dezagun. Array bat multzo bat bezala elementuen sekuentzia bat da, baina arrayetan elementuen arteko posizioak garrantzizkoak dira eta multzoetan ez da elementuen posiziorik aintzat hartzen. Beraz M1 = { a, e, i, o, u } multzoa eta M2 = { u, e, a, o, i } multzoa berdinak lirateke, baina karaktereen arrayei buruz ari bagara sekuentzia horiek array ezberdinak lirateke (array baten lehen elementua a denean bestearen lehen elementua u delako).

Multzoek eta erregistroek antzekotasunak dituzte baina ez dira berdinak. Batetik erregistro baten osagaiak datu-mota ezberdinekoak izan daitezkeelako eta multzoen osagaiak berriz datu-mota bera daukatelako. Bestetik, erregistro baten osagaien atzipena zuzen eta zilegia den bitartean, multzoetan osagaiak zeharka tratatzen dira.

11.3.1 Definizioa

Multzo bat definitzeko oinarrizko datu-mota bat aukeratu beharra dago, oinarrizko datu-mota hori datu-mota ordinal bat izango da adibidez:

```
TYPE
   DM_Astegunak = (Astelehena, Asteartea, Asteazkena,
                   Osteguna, Ostirala, Larunbata, Igandea) ;
   DM_Erantzunak = (Bai, Ez, EzDakit) ;
   DM_Minuskulak = 'a'..'z' ;
   DM_Orduak = 0..23 ;
```

Oinarrizko datu-motak multzo baten unibertsoa adierazten du (multzoan egon daitezkeen elementu guztien zerrenda litzateke), behin oinarrizko datu-mota zehaztu ondoren multzo jakin bat definitzeko Turbo Pascal lengoaiak jarraian ematen den sintaxia behar du:

```
TYPE
   DM_Asteburua = SET OF DM_Astegunak ;
   DM_Baiezkoa =  SET OF DM_Erantzunak ;
   DM_Ukapena =   SET OF DM_Erantzunak ;
   DM_Bokalak =   SET OF DM_Minuskulak ;
   DM_Gaubekoa =  SET OF DM_Orduak ;
   DM_OrduBikoitiak =  SET OF DM_Orduak ;
```

Non `SET OF` hitz erreserbatuak diren eta bere ostean datorrena lehendik definituriko datu-mota ordinal bat den. Multzoaren datu-mota finkatu eta gero aldagai bat definituko litzateke eta honek onar ditzakeen balioak multzoaren oinarrizko datu-motan zehazturikoak izango dira. Adibidez, demagun `DM_Asteburua` datu motako `Jaiegunak` izeneko multzo-aldagaia dugula, bere balioak honelaxe ematen zaizkio:

```
   Jaiegunak := [Larunbata, Igandea] ;
```

Ikusten denez `Jaiegunak` izeneko multzo-aldagaiak oinarrizko datu-motaren unibertsoko bi elementu ditu, Turbo Pascal lengoaian multzo-aldagai baten elementuak komaz banatzen dira eta elementuen zerrenda osoa kako artean mugatzen da.

Jarraian erakusten den `MultzoenDefinizioa` programan multzo aldagaiei balioak ematen zaizkie, zeregin horretarako multzoen arteko esleipena aplikatzen da:

```
PROGRAM MultzoenDefinizioa ;                { \TP70\11\SET_02.PAS }

TYPE                                        { zenbait unibertso }
  DM_Orduak     = 0..23 ;
  DM_Astegunak  = (Astelehena, Asteartea, Asteazkena, Osteguna, Ostirala,
                   Larunbata, Igandea) ;
  DM_Erantzunak = (Bai, Ez, EzDakit) ;
  DM_Minuskulak = 'a'..'z' ;

TYPE                                        { zenbait multzo datu-mota }
  DM_Gaubekoak     = SET OF DM_Orduak ;
  DM_OrduBikoitiak = SET OF DM_Orduak ;
  DM_Asteburua     = SET OF DM_Astegunak ;
  DM_Baiezkoa      = SET OF DM_Erantzunak ;
  DM_Ukapena       = SET OF DM_Erantzunak ;
  DM_Bokalak       = SET OF DM_Minuskulak ;

VAR                                         { multzo aldagaiak }
  Ordutegia : DM_Gaubekoak ;
  Jaiegunak : DM_Asteburua ;
  Lanegunak : DM_Asteburua ;
  Iritzia   : DM_Ukapena ;
  Letrak    : DM_Bokalak ;

BEGIN
   WriteLn ('Multzo aldagaiei balioak ematen') ;

   Jaiegunak := [Larunbata, Igandea] ;
   Lanegunak := [Astelehena..Ostirala] ;

   Ordutegia := [0, 1, 2, 4, 5] ;
   Ordutegia := [0..3, 5] ;
   Ordutegia := [] ;

   Iritzia   := [Ez, EzDakit] ;

   Letrak    := ['a', 'e'] ;
   Letrak    := ['a' .. 'e'] ;
   Letrak    := ['a', 'b', 'c', 'd', 'e'] ;

   WriteLn ('Multzo aldagaiek balioak dituzte') ;
END.
```

Programa honen arabera `Jaiegunak` aldagaia multzo bat izango da eta dituen bi elementuak `Larunbata` eta `Igandea` dira.

`Ordutegia` izeneko aldagaiak berriz, hiru esleipen jarraian daudelako, azkeneko esleipenaren balioa izango du (hots, multzo hutsa). Baina lehenago `Ordutegia` multzo horrek izan dituen balioak hauek izan dira:

`0`, `1`, `2`, `4` eta `5`	enumerazioz emaniko balioak
`0`, `1`, `2`, `3` eta `5`	azpieremu bat definituz eta balio bat enumeratuz

Zeintzu lirateke `Iritzia` eta `Letrak` multzo aldagaien balioak `MultzoenDefinizioa` programaren arabera?.

11.3.2 Eragiketak multzoekin

Multzoek daukaten oinarrizko erlazioa *barnekotasun erlazioa* da. Hots, elementuren bat multzoaren barnean ote dagoen, edo beste multzoren bat erreferentziazko multzoaren barnean ote dagoen. Barnekotasun erlaziotik beste hiru ondoriotzen dira: azpimultzoa (multzo bat beste baten partaidea izatea), gainmultzoa (multzo batek beste baten elementu guztiak dituenean) eta berdintasuna (bi multzo aldiberean elkarrekiko azpimultzo eta gainmultzo direnean).

Multzoek onartzen dituzten eragiketak lau dira: multzoen bilketa, multzoen ebaketa, multzoaren osaketa eta multzoen diferentzia.

11.3.2.1 Multzoen arteko erlazioak

Multzoen arteko erlazioak azaltzeko has gaitezen lehenik barnekotasuna ikasten eta ondoren beste lau erlazioak: azpimultzo, gainmultzo, berdintasuna eta desberdintasuna.

11.3.2.1.1 Barnekotasuna

Osagai zehatz bat multzo baten barnean ote dagoen jakiteko, Pascal lengoaiak `IN` eragilea eskaintzen du, zein adierazpen boolear batean agertuko den. Ikus dezagun programa praktiko baten bitartez `IN` eragilea nola erabiltzen den.

`MenuaMultzoekin` programaren `Aukerak` aldagaia karaktere alfabetikoen multzo bat da, eta bere osagaiak programaren hasierako sententzian zehazten dira (`Aukerak` multzoaren osagaiak `'A'`, `'B'`, `'C'`, `'a'`, `'b'` eta `'c'` dira).

Hona hemen `MenuaMultzoekin` programa:

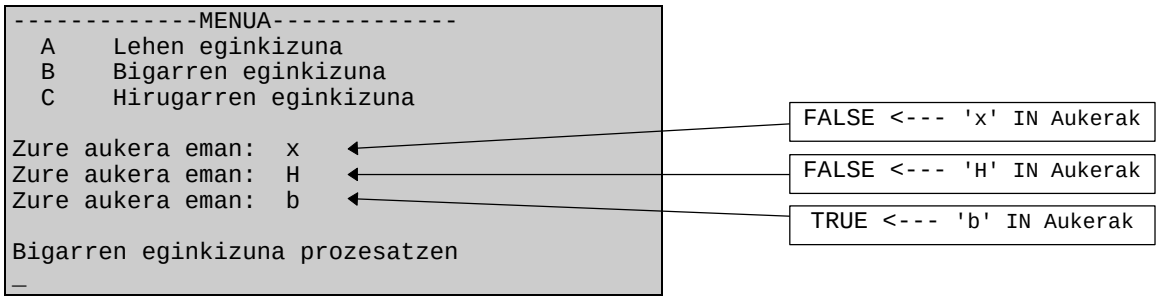
```
PROGRAM MenuaMultzoekin ;                          { \TP70\11\SET_03.PAS }
USES
   Crt ;

PROCEDURE MenuaErakutsi ;
BEGIN
   ClrScr ;
   WriteLn ('-------------MENUA-------------') ;
   WriteLn ('  A    Lehen eginkizuna') ;
   WriteLn ('  B    Bigarren eginkizuna') ;
   WriteLn ('  C    Hirugarren eginkizuna') ;
   WriteLn ;
END ;

PROCEDURE AukeraProzesatu (Hautapena : Char) ;
BEGIN
   WriteLn ;
   CASE Hautapena OF
      'A', 'a' : WriteLn ('Lehen eginkizuna prozesatzen') ;
      'B', 'b' : WriteLn ('Bigarren eginkizuna prozesatzen') ;
      'C', 'c' : WriteLn ('Hirugarren eginkizuna prozesatzen') ;
   END ;
END ;
```

```
TYPE
  DM_MajusEtaMinuskulak = 'A'..'z' ;             { unibertso osoa }
  DM_Aukerak = SET OF DM_MajusEtaMinuskulak ;    { multzo datu-mota }

VAR
  Aukerak : DM_Aukerak ;
  Karak : Char ;

BEGIN
   Aukerak := ['A', 'B', 'C', 'a', 'b', 'c'] ;   { multzo aldagaiaren osagaiak }

   MenuaErakutsi ;

   REPEAT
      Write ('Zure aukera eman:  ') ;
      ReadLn (Karak) ;
   UNTIL Karak IN Aukerak ;

   AukeraProzesatu (Karak) ;
END.
```

`MenuaMultzoekin` programaren punturik interesgarriena aukeraren irakurketarena da, baliagarriak diren karaktereen iragazketa alegia. Orain arte, multzorik ezagutzen ez genuen artean honelako zerbait egiten zen `Karak` aldagaiari zentzudun balioak mugatu ahal izateko, non `REPEAT-UNTIL` egitura biren baldintza baliokidea den:

```
   REPEAT
      Write ('Zure aukera eman:  ') ;
      ReadLn (Karak) ;
      Karak := UpCase (Karak) ;
   UNTIL (Karak = 'A') OR (Karak = 'B') OR (Karak = 'C') ;
```

`MenuaMultzoekin` programan argi ikusten denez, `IN` eragileak elementu bat multzo batekin konektatzen du, eta bere emaitza balio boolear bat dela frogatzeko `MenuaMultzoekin` programa exekuta daiteke:

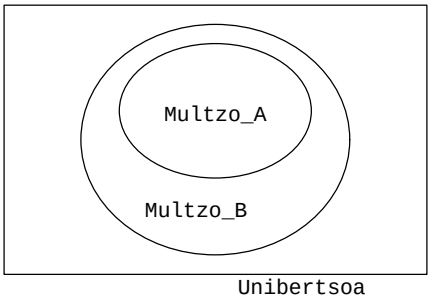


11.3.2.1.2 Azpi eta gainmultzoa

Dakigunez, elementu bat multzo batean ote dagoen aztertzeko `IN` eragilea erabiltzen dugu. Baina multzo baten elementu guztiak beste multzo batek barneratzen dituen jakiteko, hots multzo bat beste baten azpimultzoa den frogatzeko `IN` eragileak ez du balio. Multzoek `<=` eragilea onartzen dute, eta adierazpen boolear horrek `TRUE` itzuliko du baldin eta `Multzo_A` ezkerreko multzoa `Multzo_B` eskuineko multzoaren azpimultzoa bada.

```
Multzo_A <= Multzo_B
```

Venn-en diagrama eginez:

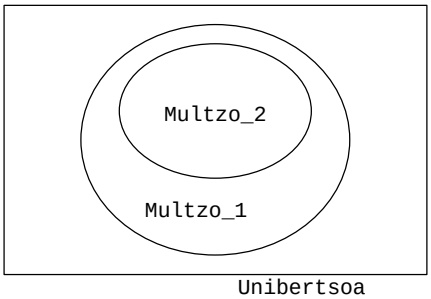


TRUE ◄——— Multzo_A <= Multzo_B

Zein litzateke jarraian ematen den `AzpiEtaGainMultzoak` programaren irteera?

```
PROGRAM AzpiEtaGainMultzoak ;                          { \TP70\11\SET_04.PAS }
USES
  Crt ;

TYPE
  DM_LetraMajuskulak = 'A'..'Z' ;                      { unibertso bat }
  DM_BokalMajuskulak = SET OF DM_LetraMajuskulak ;     { multzo datu-mota bat }

  DM_ZenbakiTxikiak = 0..35 ;                          { beste unibertso bat }
  DM_BikoitiTxikiak = SET OF DM_ZenbakiTxikiak ;       { multzo datu-mota bat }

VAR
  Multzo_A, Multzo_B : DM_BokalMajuskulak ;
  Multzo_1, Multzo_2 : DM_BikoitiTxikiak ;

BEGIN
   ClrScr ;
   Multzo_A := ['A', 'U', 'E'] ;                       { multzo aldagaiak }
   Multzo_B := ['A', 'E', 'I', 'O', 'U'] ;
   WriteLn ('Multzo_A Multzo_Bren azpimultzoa da:   ', Multzo_A <= Multzo_B) ;
   WriteLn ('Multzo_B unibertsoaren azpimultzoa da: ', Multzo_B <= ['A'..'Z']) ;

   WriteLn ;

   Multzo_1 := [0, 2, 4, 6, 12, 18] ;                  { multzo aldagaiak }
   Multzo_2 := [2, 6] ;
   WriteLn ('Multzo_1 Multzo_2ren gainmultzoa da:   ', Multzo_1 >= Multzo_2) ;
   WriteLn ('Unibertsoa Multzo_1en gainmultzoa da:  ', [0..32] >= Multzo_1) ;
END.
```

Ikus daitekeenez `AzpiEtaGainMultzoak` programan `Multzo_1` multzoa `Multzo_2` beste multzo baten gainmultzoa ote den frogatzen da. Horretarako `>=` eragilea aplikatzen da, eta Venn-en diagrama honek adierazten duen kasurako `Multzo_1 >= Multzo_2` adierazpen boolearrak `TRUE` balio du:



TRUE ◄——— Multzo_1 >= Multzo_2

11.3.2.1.3 Berdintasuna eta desberdintasuna

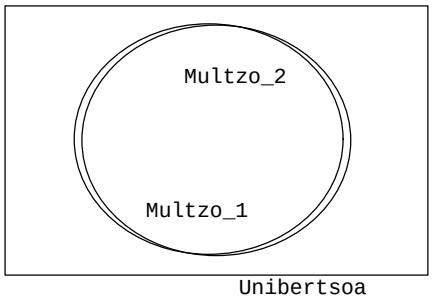
Bateragarriak diren multzo bi *berdinak* izan daitezen batek dituen osagai guztiak besteak izan behar ditu eta honek dituenak lehenengoak eduki behar ditu; beste era batez esanik bi multzo berdinak izan daitezen euretariko bat bestearen gainmultzoa eta azpimultzoa izango da aldi berean. `Multzo_1` eta `Multzo_2` multzoak berdinak dira baldin eta:

```
(Multzo_1 >= Multzo_2) AND (Multzo_1 <= Multzo_2)
```

Baina Pascal lengoaiak multzoen arteko berdintasuna adierazteko = eragilea onartzen duenez, aurreko adierazpen boolearra askoz trinkoago eta ulergarriagoa den honetaz ordezka daiteke:

```
Multzo_1 = Multzo_2
```

Berdintasunari dagokion Venn-en diagrama eginez:



Unibertsoa

TRUE ⟵ `Multzo_1 = Multzo_2`

Multzoen arteko *desberdintasuna* ematen denean, multzo batean beste multzoaren osagaien bat faltan ala soberan dagoelako da. Suposa daitekeenez Pascal lengoaiak multzoen arteko desberdintasunerako <> eragilea eskaintzen du, honezkero aurreko Venn-en diagraman agertzen den egoera adierazteko bi bide daude:

`Multzo_1 = Multzo_2` edo `NOT (Multzo_1 <> Multzo_2)`

Zein uste duzu izango litzatekeela jarraian erakusten den `MultzoBerdintasunak` programaren irteera?

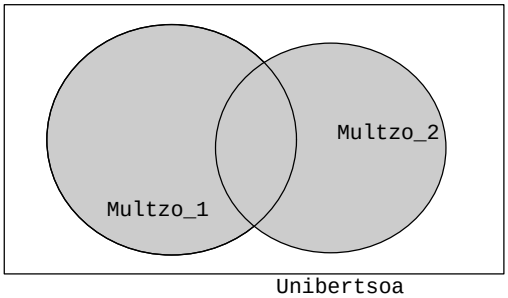
```
PROGRAM MultzoBerdintasunak ;                        { \TP70\11\SET_05.PAS }
USES
  Crt ;

TYPE
  DM_ZenbakiTxikiak = 0..35 ;                          { unibertso bat }
  DM_BikoitiTxikiak = SET OF DM_ZenbakiTxikiak ;       { multzo datu-mota bat }

VAR
  Multzo_1, Multzo_2 : DM_BikoitiTxikiak ;

BEGIN
   ClrScr ;
   Multzo_1 := [2, 6, 12, 18] ;                        { multzo aldagaiak }
   Multzo_2 := [18, 2, 6, 12] ;

   WriteLn ('Multzo_1 berdin Multzo_2:   ', Multzo_1 = Multzo_2 ) ;

   WriteLn ('Multzo_1 berdin Multzo_2:   ', NOT (Multzo_1 <> Multzo_2) ) ;

   WriteLn ('Multzo_1 berdin Multzo_2:   ', (Multzo_1 >= Multzo_2) AND
                                            (Multzo_1 <= Multzo_2) ) ;
END.
```

11.3.2.2 Multzoen eragileak

Barnekotasun kontzeptutik ondoriotzen diren erlaziozko[14] lau eragileak (>=, <=, =, <>) ikusi ondoren, multzoen eragiketa berezienak eta dagozkien eragileak ikas ditzagun: bilketa, ebaketa, diferentzia eta osaketa. Azken lau eragiketa hauek bi multzo eragigaitzat hartuz beste multzo berri bat sortzen dute.

11.3.2.2.1 Bilketa

Bi multzoen bilketa burutzean bildura multzoa lortzen da zein aurrekoen elementu guztiez osaturik dagoen. `Multzo_1` eta `Multzo_2` multzoen bilketa egiteko Pascal lengoaiak ezaguna zaigun + eragilea darabil, eta Venn-en diagramaren laguntzaz erraz ulertzen da:




```
Bildura := Multzo_1 + Multzo_2
```

Esan den bezala bi multzoen bildura eskuratzeko Pascal lengoaiak + eragilea darabil, hau da eragiketa aritmetikoa den batuketaren sinbolo berbera. Kontutan izan `Multzo_1` eta `Multzo_2` multzoek osagai amankomunik ez dutenean bilketaren emaitza den bildura eta batuketaren emaitza den batura berdinak izango liratekeela, baina `Multzo_1` eta `Multzo_2` multzoek osagai berbera barneratzen dutenean multzoen arteko “batuketa” egitean baturan osagai errepikatuak agertu beharko lirateke eta hori ez da multzoetan onartzen. Beraz bilketa egitean `Multzo_1` eta `Multzo_2` multzoen osagaiak “batu” egin ondoren osagai errepikatuak kentzen dira.

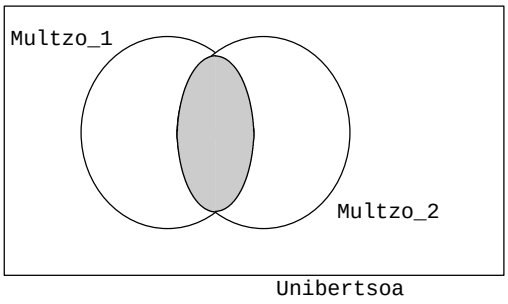
Esandakoa aintzat harturik ez da zaila izango `MultzoenBildura` programa honen irteera asmatzea:

```
PROGRAM MultzoenBildura ;                              { \TP70\11\SET_06.PAS }
TYPE
  DM_ZenbakiTxikiak = 0..9 ;                             { unibertso bat }
  DM_BikoitiTxikiak = SET OF DM_ZenbakiTxikiak ;         { multzo datu-mota bat }

VAR
  Multzo_1, Multzo_2, Bildura : DM_BikoitiTxikiak ;
  k : Byte ;

BEGIN
   Multzo_1 := [2, 4, 6] ;                               { multzo aldagaiak }
   Multzo_2 := [0, 4, 6, 8] ;
   Bildura := Multzo_1 + Multzo_2 ;                      { multzoen bildura }

   FOR k:=0 TO 9 DO
      IF k IN Bildura THEN
        WriteLn (k, ' osagaia Bildura multzoan dago') ;
END.
```

[14] Adierazpen boolearrak direnez euren emaitza `TRUE` ala `FALSE` izango da.

11.3.2.2.2 Ebaketa

Esan dugun bezala bi multzoen bilketa egitean bildura lortzen da eta jatorrizko multzo batean **edo** bestean dauden elementuz osatzen da. Bi multzoen ebakidura beste multzo bat da eta hori lortzeko jatorrizko multzo batean **eta** bestean dauden elementuak identifikatu behar dira. Ikus `Multzo_1` eta `Multzo_2` multzoen ebaketarako Venn-en diagrama:




```
Ebakidura := Multzo_1 * Multzo_2
```

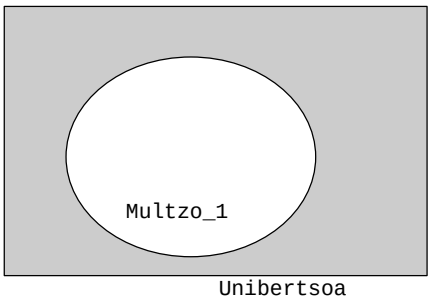
`Multzo_1` eta `Multzo_2` multzoen ebaketa formulatzeko Pascal lengoaiak `*` eragilea darabil eta ez du zerikusirik biderketa burutzen duen eragile aritmetikoarekin.

11.3.2.2.1 Bilketa puntuan ikusi dugun izango `MultzoenBildura` programan agertzen den `+` ordez `*` jarriko bagenu `MultzoenEbakidura` izeneko programa lortuko genuke. `Multzo_1` eta `Multzo_2` multzoen osagaiak aldatu gabe mantenduz, zein litzateke programa berriaren irteera aldaketa hau balitz?

```
   Ebakidura := Multzo_1 * Multzo_2 ;                { multzoen ebakidura }
```

11.3.2.2.3 Osaketa

Osaketa operazioak multzo bat eta bere unibertsoa elkartzen ditu, eragiketa hau **11.3.2.2.4** puntuan ikusiko dugun diferentzia eragiketaren aplikazioa da. `Multzo_1` multzo baten osagarria `Osagarri_1` multzo bat da, eta biak bildu ezkero lortuko genukeen bildura multzoa unibertsoa litzateke. Hona hemen Venn-en diagrama:




```
Osagarri_1 := Unibertsoa - Multzo_1
```

`Multzo_1` multzo baten osagarria eskuratzeko Pascal goimailako lengoaiak `-` eragilea darabil (azken finean multzo baten osagarria diferentzia batez lortzen baita), baina osaketaren kasuan diferentzia formulatzean aurreneko eragigaia unibertsoa izango da nahitaez.

11.3.2.2.4 Diferentzia

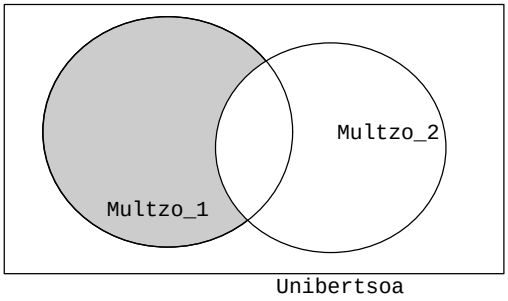
`Multzo_1` eta `Multzo_2` bi multzoen diferentzia - eragileaz lortzen da Pascal lengoaian, eta emaitza beste multzo bat da, jarraian definitzen diren elementuaz osaturik datorrena:

```
Diferentzia := Multzo_1 * (Unibertsoa - Multzo_2) ;
```

Hau da, `Multzo_2` multzoari dagokion osagarria eta `Multzo_1` multzoaren arteko ebaketa litzateke diferentziaren emaitza. `Multzo_1` eta `Multzo_2` bi multzoen diferentzia errazago formulatzeko honelaxe egingo litzateke:

```
Diferentzia := Multzo_1 - Multzo_2 ;
```

Non - eragile horrek kenketa burutzen duen eragile aritmetikoarekin zerikusirik ez duen. Hona hemen `Multzo_1` eta `Multzo_2` bi multzoen diferentzia eragiketari dagokion Venn-en diagrama:



`Diferentzia := Multzo_1 - Multzo_2`

Jarraian ematen den `MultzoenDiferentzia` izeneko programa exekutatuz gero honako irteera pantailaratuko litzateke:

```
PROGRAM MultzoenDiferentzia ;                        { \TP70\11\SET_09.PAS }
USES
  Crt ;

TYPE
  DM_ZenbakiTxikiak = 0..9 ;                          { unibertso bat }
  DM_BikoitiTxikiak = SET OF DM_ZenbakiTxikiak ;      { multzo datu-mota bat }

VAR
  Multzo_1, Multzo_2, Unibertsoa, Diferentzia : DM_BikoitiTxikiak ;
  k : Byte ;

BEGIN
   ClrScr ;
   Multzo_1 := [0, 2, 4, 6, 8] ;                      { multzo aldagaiak }
   Multzo_2 := [0, 1, 2, 3] ;
   Unibertsoa := [0..9] ;

   Diferentzia := Multzo_1 - Multzo_2 ;               { multzo diferentzia }
   FOR k:=0 TO 9 DO
      IF k IN Diferentzia THEN
        WriteLn (k, ' osagaia Diferentzia multzoan dago') ;

   WriteLn ;
   Diferentzia := Multzo_1 * (Unibertsoa - Multzo_2) ;
   FOR k:=0 TO 9 DO
      IF k IN Diferentzia THEN
        WriteLn (k, ' osagaia Diferentzia multzoan dago') ;
END.
```

`MultzoenDiferentzia` izeneko programaren irteera:

```
4 osagaia Diferentzia multzoan dago
6 osagaia Diferentzia multzoan dago
8 osagaia Diferentzia multzoan dago

4 osagaia Diferentzia multzoan dago
6 osagaia Diferentzia multzoan dago
8 osagaia Diferentzia multzoan dago
_
```

11.3.3 Multzoak parametro bezala

Multzoen prozesaketan, gainerako datu-motekin bezala, azpiprogramak idaztea gomendatzen da. Multzo aldagaiak parametro bezala nola erabiltzen diren irakasteko asmoz `MultzoenProzesaketa` izeneko programa prestatu dugu, eta bertan parametroen pasatze modu guztiak ikus daitezke, taula honetan biltzen dira multzoen pasatzen erak:

	Baliozko parametroak	**Erreferentziazko parametroak**	
		Aldagai-parametroak	**Konstante-parametroak**
Deia	Espresioa	Aldagaia	Espresioa
Markatzailea	~~VAR~~	VAR	CONST
Komunikazioa	Sarrera	Sarrera / Irteera Irteera	Sarrera
Param. formala	Balio bat	Helbide bat	Helbide bat
Param. erreala	Babesturik	Babestu gabe	Babesturik

`MultzoenProzesaketa` izeneko programan karaktere majuskulen unibertsoa aintzat hartzen da; eta horren arabera `UNIBERTSOA` deitu dugun multzo-konstante bat definitu dugu, `Multzoa` etiketaz identifikatuko dugun programa nagusiko multzo-aldagai bat definitu dugu ere. Sei dira `MultzoenProzesaketa` programan agertzen diren moduluak lau prozedura eta bi funtzio.

Adibide-programa honetan kontzepturi aipagarriena errutinetako `M` parametroak duen izaera litzateke. Hona hemen `MultzoenProzesaketa` programaren kodifikazioa:

```
PROGRAM MultzoenProzesaketa ;                          { \TP70\11\SET_10.PAS }
USES
  Crt ;

CONST
  BEHEMUGA   = 'A' ;
  GOIMUGA    = 'Z' ;
  UNIBERTSOA = [BEHEMUGA..GOIMUGA] ;                   { multzo konstante bat }

TYPE
  DM_LetraMajuskulak = BEHEMUGA..GOIMUGA ;             { unibertso bat }
  DM_BokalMajuskulak = SET OF DM_LetraMajuskulak ;     { multzo datu-mota bat }
```

```
PROCEDURE UnibertsoaPantailaratu ;
VAR
  Indizea : Char ;
BEGIN
   FOR Indizea:=BEHEMUGA TO GOIMUGA DO
      IF Indizea IN UNIBERTSOA THEN
        Write (Indizea:5) ;
   WriteLn ;
END ;



PROCEDURE MultzoaTeklatuzEraiki (VAR M : DM_BokalMajuskulak) ;
VAR
  Erantz, Osagaia : Char ;
  MultzoLaguntzailea : DM_BokalMajuskulak ;
BEGIN
   REPEAT
      Write ('Zein da multzoari gehitu nahi diozun osagaia?  ') ;
      ReadLn (Osagaia) ;
                                                  { osagaia zehaztu }
      MultzoLaguntzailea := [Osagaia] ;           { osagaia multzo bihurtu }
      M := M +  MultzoLaguntzailea ;              { multzoen bildura }

      REPEAT
         Write ('Osagai gehiagorik (B/E)?  ') ;
         Erantz := ReadKey ;
         Erantz := UpCase (Erantz) ;
         WriteLn (Erantz) ;
      UNTIL (Erantz='B') OR (Erantz='E') ;
   UNTIL Erantz = 'E' ;
END ;



FUNCTION MultzoarenOsagaienKopuru1 (CONST M : DM_BokalMajuskulak) : Byte ;
VAR
  Kontagailua : Byte ;
  Indizea : Char ;
BEGIN
   Kontagailua := 0 ;
   FOR Indizea:=BEHEMUGA TO GOIMUGA DO
      IF Indizea IN M THEN
        Kontagailua := Kontagailua + 1 ;
   MultzoarenOsagaienKopuru1 := Kontagailua ;
END ;



FUNCTION MultzoarenOsagaienKopuru2 (M : DM_BokalMajuskulak) : Byte ;
VAR
  Kontagailua : Byte ;
  OsagaiBat : Char ;
BEGIN
   Kontagailua := 0 ;
   OsagaiBat := BEHEMUGA ;
   WHILE M <> [] DO
   BEGIN
      IF OsagaiBat IN M THEN
      BEGIN
         Kontagailua := Kontagailua + 1 ;
         M := M - [OsagaiBat] ;
      END ;
      OsagaiBat := Succ (osagaiBat) ;
   END ;
   MultzoarenOsagaienKopuru2 := Kontagailua ;
END ;
```

```
PROCEDURE MultzoarenOsagaiakIkusi (CONST M : DM_BokalMajuskulak) ;
VAR
  Indizea : Char ;
BEGIN
   FOR Indizea:=BEHEMUGA TO GOIMUGA DO
      IF Indizea IN M THEN
        WriteLn (Indizea, ' osagaia M multzoan dago') ;
END ;



PROCEDURE MultzoarenOsagaiakTratatu (VAR M : DM_BokalMajuskulak) ;
VAR
  Indizea, OsagaiBat : Char ;
BEGIN
   FOR Indizea:=BEHEMUGA TO GOIMUGA DO
      IF Indizea IN M THEN
      BEGIN
         OsagaiBat := Indizea ;
         M := M - [OsagaiBat] ;                    { osagai zaharra kendu }
         OsagaiBat := Pred (OsagaiBat) ;           { osagai berria lortu }
         M := M + [OsagaiBat] ;                    { osagai berria sartu }
      END ;
END ;



VAR
  Multzoa : DM_BokalMajuskulak ;                   { multzo aldagai bat }
  Zenbat : Byte ;

BEGIN
   ClrScr ;
   WriteLn ('Hona hemen UNIBERTSO-aren osagaiak:') ;
   UnibertsoaPantailaratu ;

   Multzoa := [] ;                                 { multzoaren hasieraketa }
   MultzoaTeklatuzEraiki (Multzoa) ;
   Zenbat := MultzoarenOsagaienKopuru1 (Multzoa) ;
   WriteLn ('Multzoaren osagaien kopurua (1. era):  ', Zenbat ) ;
   Zenbat := MultzoarenOsagaienKopuru2 (Multzoa) ;
   WriteLn ('Multzoaren osagaien kopurua (2. era):  ', Zenbat ) ;

   WriteLn ;
   WriteLn ('Hona hemen Multzoa-ren osagaiak:') ;
   MultzoarenOsagaiakIkusi (Multzoa) ;

   WriteLn ;
   MultzoarenOsagaiakTratatu (Multzoa) ;
   WriteLn ('Hona hemen Multzoa-ren osagai berriak:') ;
   MultzoarenOsagaiakIkusi (Multzoa) ;
END.
```

`MultzoenProzesaketa` programa garrantzitsua iruditzen zaigulako xehetasunez azalduko dugu, esplikazioak ez nahastearren `MultzoenProzesaketa` programa sei zatitan banatuko dugu, konstante eta datu-moten definizioaz gain ondoko sei atal hauek:

1. Multzo-konstante baten osagaiak pantailaratu.
2. Multzo-aldagai baten osagaiak teklatuz zehaztu.
3. Multzo baten osagai kopurua kalkulatu (lehenengo erara).
4. Multzo baten osagai kopurua kalkulatu (bigarren erara).
5. Multzo-aldagai baten osagaiak pantailaratu.
6. Multzo baten osagai guztiak prozesatu.

1

Multzo-konstante baten osagaiak pantailaratu. Aldez aurretik definiturik daukagun UNIBERTSOA konstantearen osagaiak pantailan idazteko FOR-DO baten bitartez ibilera hau egitea aski da:

```
FOR Indizea:=BEHEMUGA TO GOIMUGA DO
   IF Indizea IN UNIBERTSOA THEN
     Write (Indizea:5) ;
```

Non Indizea karaktere bat den, eta idazketak gertatzeko Indizea-ren balioak UNIBERTSOA-rekiko barnekotasuna betetzen duen. Kontutan izan dezagun **UnibertsoaPantailaratu** prozeduran ez dela parametrorik behar UNIBERTSOA deituriko multzoa konstante bat delako.

2

Multzo-aldagai baten osagaiak teklatuz zehaztu. Zeregin honetan eta hurrengoetan Multzoa izeneko multzo-aldagai bat landuko dugu, eta ezer baino lehen hasieraketa bat egingo diogu multzo hutsa esleituko zaio Multzoa aldagaiari. Ondoren **MultzoaTeklatuzEraiki** prozedura piztuko dugu bertan karaktereak teklatuz irakurriz eta multzoen bildura gauzatuz:

```
ReadLn (Osagaia) ;                  { osagaia zehaztu }
MultzoLaguntzailea := [Osagaia] ;   { osagaia multzo bihurtu }
M := M +  MultzoLaguntzailea ;      { multzoen bildura }
```

Non Osagaia karaktere bat den eta MultzoLaguntzailea multzo bat den, sententziarik interesgarriena karakterea multzo bihurtzen duena da eta bi egoera ezberdin gerta daitezke:

Osagaia	Osagaia-k unibertsoarekiko	MultzoLaguntzailea
'H'	barnekotasuna betetzen du	['H']
'h'	barnekotasuna ez du betetzen	[]

Honezkero, karakterea den Osagaia irakurri eta multzo bihurtu ondoren M multzoari gehitzen zaio bilketa operazioaren bitartez.

MultzoaTeklatuzEraiki prozeduran M multzoari ematen zaizkion balioak irteerakoak dira eta programa nagusiko Multzoa izeneko aldagaiari transferitu behar zaizkio, horregatik **MultzoaTeklatuzEraiki** prozeduraren M parametroa *aldagai-parametroa* izango da.

3

Multzo baten osagai kopurua kalkulatu (lehenengo erara). Funtsean lehenengo puntuan ikusi dugun algoritmo berbera da, eta benetako aldaketa parametroarena da, hots, **MultzoarenOsagaienKopuru1** delako funtzioan M parametro bat erabiltzen dela. M parametroa *aldagai-parametro* balitz programa nagusiko bere kidea, Multzoa, babesik gabe egongo litzateke eta hori ez da komeni. Beraz M parametroa *konstante-parametro* bezala pasatuko zaio **MultzoarenOsagaienKopuru1** funtzioari.

4

Multzo baten osagai kopurua kalkulatu (bigarren erara). Bigarren era honetan ez da unibertsoko osagai guztiak banan-banan aztertzen baizik eta M multzoarenak. Algoritmo hau aurrekoarekin alderatuz hobea da unibertsoan osagai asko direnean, baina konplexuagoa da; izan ere, M multzoan aztertutako elementu oro M multzotik atera beharra dago. Beraz, algoritmoaren gakoa WHILE-DO egituraren baldintzan dago.

MultzoarenOsagaienKopuru2 funtzioan M parametro bat erabiltzen da, eta M parametroa *aldagai-parametro* balitz programa nagusiko bere kidea babesik gabe egongo litzatekeelako ez da pasatze era hori komeni. M parametroa *konstante-parametro* balitz programa nagusiko bere kidea,

`Multzoa`, babesturik legoke eta hori gauza ona da, baina `M` konstante bat delako ezingo litzateke hau konpilatu:

```
M := M - [OsagaiBat] ;
```

Ondorioz, **`MultzoarenOsagaienKopuru2`** funtzioan `M` parametroa baliozko parametroa izango da eta ez da erreferentziaz pasatuko.

5

Multzo-aldagai baten osagaiak pantailaratu. Ibilera burutzeko ezaguna zaigun algoritmoa aplikatu da berriro ere **`MultzoarenOsagaiakIkusi`** prozeduran, unibertso osoa arakatuz bere elementuak banan-banan aztertu eta `M`-rekiko barnekotasuna betetzen duten ala ez frogatu.

`M`-ren kidea programa nagusian ez du aldaketarik jasan behar (baliozko parametroa edo aldagai-parametroa), eta pilan `Multzoa` aldagaiaren kopia egitea ekidin nahi delako (erreferentzia bat: konstante-parametroa edo aldagai-parametroa). Beraz, beharkizun biak bete ahal izateko `Multzoa` aldagaiari `M` *konstante-parametroa* dagokio.

6

Multzo baten osagai guztiak prozesatu. Ibilera bat izango da ere, baina programa nagusiko `Multzoa` aldagaia aldatua suertatuko da, horregatik `M` *aldagai-parametroa* izango da.

`MultzoarenOsagaiakTratatu` prozeduraren helburua `Multzoa` aldagaia osatzen duten elementuak aldatzea da, `Multzoa` aldagaiaren elementu bakoitza karakteren zerrendan dagokion bere aurrekoaz ordezkatzen da.

```
FOR Indizea:=BEHEMUGA TO GOIMUGA DO
   IF Indizea IN M THEN
   BEGIN
      OsagaiBat := Indizea ;
      M := M - [OsagaiBat] ;           { osagai zaharra kendu }
      OsagaiBat := Pred (OsagaiBat) ;  { osagai berria lortu }
      M := M + [OsagaiBat] ;           { osagai berria sartu }
   END ;
```

`Indizea` karakterea `M`-ren osagaia bada, bitarteko `OsagaiBat` aldagai laguntzailea multzo bihurtzen da eta `M`-ren diferentzia kalkulatu egiten da (osagaia `M`-tik atera egiten da), `Pred` funtzio estandarri esker `OsagaiBat` aldagairen aurreko karakterea bilatu eta `M`-rekin bildu.

Aztertzen ari garen `MultzoenProzesaketa` programaren irteera bere sarrerako datuen araberakoa litzateke, jarraian exekuzio batetik ateratako pantailarakada bat erakusten da:

```
Hona hemen UNIBERTSO-aren osagaiak:
    A    B    C    D    E    F    G    H    I    J    K    L    M    O    P
    Q    R    S    T    U    V    W    X    Y    Z
Zein da multzoari gehitu nahi diozun osagaia?  H
Osagai gehiagorik (B/E)?  B
Zein da multzoari gehitu nahi diozun osagaia?  h
Osagai gehiagorik (B/E)?  B
Zein da multzoari gehitu nahi diozun osagaia?  Z
Osagai gehiagorik (B/E)?  B
Zein da multzoari gehitu nahi diozun osagaia?  A
Osagai gehiagorik (B/E)?  B
Zein da multzoari gehitu nahi diozun osagaia?  K
Osagai gehiagorik (B/E)?  E
Multzoaren osagaien kopurua (1. era):  4
Multzoaren osagaien kopurua (2. era):  4
```



```
Hona hemen Multzoa-ren osagaiak:
A osagaia M multzoan dago
H osagaia M multzoan dago
K osagaia M multzoan dago
Z osagaia M multzoan dago

Hona hemen Multzoa-ren osagai berriak:
G osagaia M multzoan dago
J osagaia M multzoan dago
Y osagaia M multzoan dago
_
```

`MultzoenProzesaketa` programaren exekuzio adibidea aurrean daukagula, galdera hauei erantzutea eskatzen da:

1. Zer gertatuko litzateke `UnibertsoaPantailaratu` prozeduran `FOR-DO` barneko `IF-THEN` sententziaren baldintza kenduko bagenu?

```
FOR Indizea:=BEHEMUGA TO GOIMUGA DO
{ IF Indizea IN UNIBERTSOA THEN        baldintza indargabetu }
     Write (Indizea:5) ;
```

2. `MultzoaTeklatuzEraiki` prozedura barruan 5 karaktere irakurri badira teklatuz (`'H'`, `'h'`, `'Z'`, `'A'` eta `'K'`), zergatik multzoaren kopurua kalkulatzean 4 besterik ez dira zenbatzen?

3. `MultzoaTeklatuzEraiki` prozedura barruan irakurri diren 5 karaktereek orden hau gordetzen dute: `'H'`, `'h'`, `'Z'`, `'A'`, `'K'`. Zergatik `Multzoa` aldagaiaren osagaiak erakustean `MultzoarenOsagaiakIkusi` prozedurak beste orden bat darabil?

4. `MultzoarenOsagaiakIkusi` prozeduraren bitartez `Multzoa` aldagaiak 4 osagai dituela frogatzen da. Zer dela eta, `MultzoarenOsagaiakTratatu` azpirrutinak multzo hori landu ondoren bere osagaiak 3 dira? Zein osagai galdu da?.

5. `MultzoarenOsagaiakTratatu` azpirrutinak multzo baten osagaiak euren aurrekoegatik ordezkatzen ditu. Zer aldaketak egin beharko geniozke prozedura horri multzoaren osagaiak euren atzekoengatik ordezkatzeko?

   Gogoratu karaktere bati jarraitzen dion hurrengoa lortzeko `Succ` funtzio estandarra erabil daitekeela, baina tamalez planteatzen dugun zereginean ez da nahikoa`Pred` funtzio estandarraren ordez `Succ` funtzioa jartzea. Zergatik lerro hauen bitartez ez da lortzen eskatzen duguna?

```
FOR Indizea:=BEHEMUGA TO GOIMUGA DO
   IF Indizea IN M THEN
   BEGIN
      OsagaiBat := Indizea ;
      M := M - [OsagaiBat] ;           { osagai zaharra kendu }
      OsagaiBat := Succ (OsagaiBat) ;  { osagai berria lortu }
      M := M + [OsagaiBat] ;           { osagai berria sartu }
   END ;
```

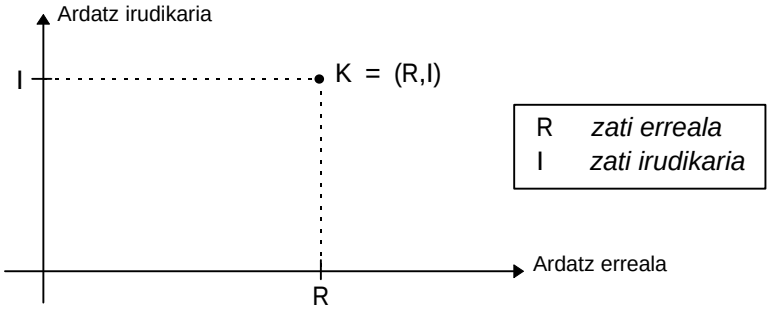
11.4 ZENBAKI KONPLEXUEN ERAGIKETATARAKO UNITATEA

Unitateen ikasgaian ikusitako kontzeptuak argitzeko asmoz eta erregistro datu-mota praktikan jar dezagun zenbaki konplexuak lantzen dituen `ZbkKonpl` unitatea sortuz.

Ingeniari eta zientzilarien eguneroko zeregin profesionalean ohizkoa da zenbaki konplexuak[15] tartean direla kalkuluak egin behar izatea. Goimailako programazio-lengoaiek ez dituzte zenbaki konplexuak lantzeko datu-mota berezirik eskaintzen, guri dagokigu horrelakorik sortzea.

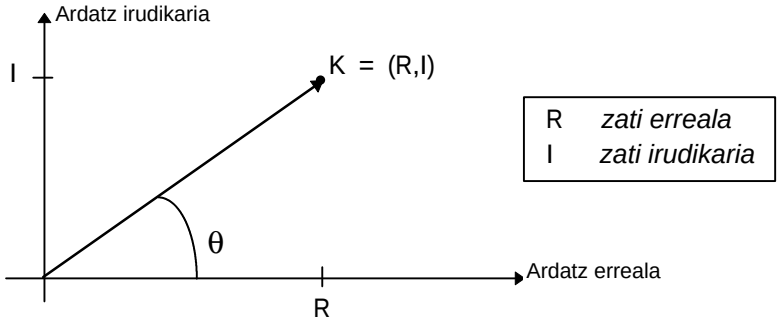
Ezer baino lehen zenbaki konplexuen adierazpide ezberdinak gogora ekar ditzagun.

K zenbaki konplexu bat, zenbaki errealen (R,I) bikote ordenatu bat da. Bikotearen lehenengo partaideari *zati erreala* esaten zaio eta bigarrenari *zati irudikaria*. Zenbaki konplexu bat puntu baten bidez adieraz daiteke plano konplexuan, non ardatz horizontala K-ren zati errealarekin loturik dagoen eta ardatz bertikala K-ren zati irudikariari dagokion. Ikus irudia:



K zenbaki konplexu bat honela idazten da: $R + I\,j$ non $j^2 = -1$

Koordenatu cartesiarren ordez K zenbaki konplexu bera koordenatu polarraz adieraz daiteke, koordenatu-jatorrirako distantzia eta ardatz erreal positiboaren angeluak hartuz. Zenbaki konplexuak zehazteko modu hau adierazpide bektoriala deitzen da, eta matematikoki aurrekoaren baliokidea da.



Hona hemen K zenbaki konplexu bat bere idazkera bektorialan: $(r \cos \theta) + j\,(r \operatorname{sen} \theta)$

Kontura gaitezkeenez, zenbaki konplexua adierazten duen r erradioa (bektorearen modulua) R^2+I^2 baturaren erro karratua da, θ angeluaren tangentea I / R zatidura delarik.

[15] Zenbaki konplexuak plano konplexuan koka daitezke. Beraz, abzisa-ardatz errrealan zenbaki konplexuaren zati erreala neur daiteke, eta, ordenatu-ardatz irudikarian zenbaki konplexuaren zati irudikaria neur daiteke.

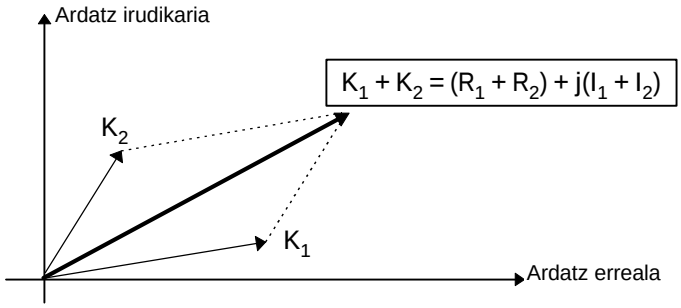
11.4.1 Zenbaki konplexuen unitatearen beharkizunak

Zenbaki konplexuen unitatearen sorrera, nolabait, datu-mota berri bat eraiketarekin pareka daiteke, zenbaki konplexuen arteko kalkuluak burutzeko datu-mota egoki bat sortzea alegia. Hori horrela ulerturik, ahal den neurrian Turbo Pascal lengoaiaren gainerako datu-moten antzera erabiltzeko prestaturik egongo da. Beraz, zenbaki konplexuei aplikagarriak diren funtsezko eragiketa aritmetikoak (batuketa, kenketa, biderketa, zatiketa) `ZbkKonpl` unitate berrian definiturik egongo dira. Bestalde, zenbaki konplexuek onar dezaketen zenbait eragiketa espezifiko ere gehituko zaio unitateari.

Batuketa:

Demagun K_1 eta K_2 zenbaki konplexuak ditugula, bektore horien batura beste bektore berri bat izango da:

$$\left.\begin{array}{l} K_1 = R_1 + I_1\, j \\ K_2 = R_2 + I_2\, j \end{array}\right| \Rightarrow \quad K_1 + K_2 = (R_1 + R_2) + j(I_1 + I_2)$$

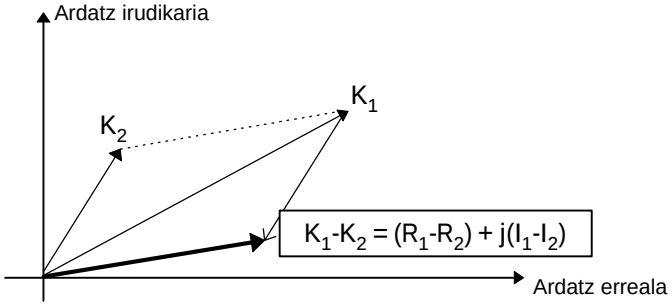


Kenketa:

Era berean, K_1 eta K_2 zenbaki konplexuen arteko kendura lortzeko kenketa bektoriala burutu behar da. Hau da, koordenatuen elkarrekiko diferentziak kalkulatuko dira:

$$\left.\begin{array}{l} K_1 = R_1 + I_1\, j \\ K_2 = R_2 + I_2\, j \end{array}\right| \Rightarrow \quad K_1 - K_2 = (R_1 - R_2) + j(I_1 - I_2)$$

Grafikoki:

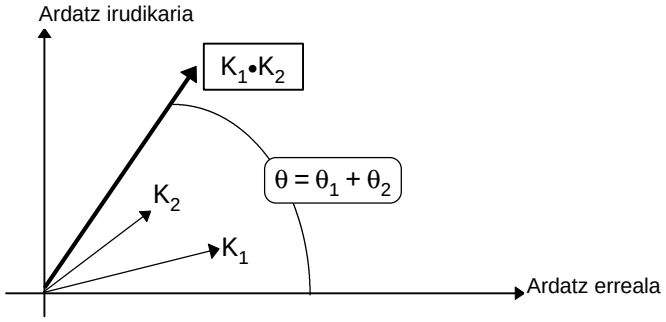


Biderketa:

K_1 eta K_2 zenbaki konplexuen arteko biderkadura lortzeko, zenbaki arrunten biderketa burutu behar da, baina $j^2 = -1$ gogoratuz. Beraz biderkadura bektorea hau da:

$$\left.\begin{array}{l} K_1 = R_1 + I_1\, j \\ K_2 = R_2 + I_2\, j \end{array}\right| \Rightarrow \quad K_1 \bullet K_2 = [(R_1 \bullet R_2) - (I_1 \bullet I_2)] + j[(R_1 \bullet R_2) + (I_1 \bullet I_2)]$$

Geometrikoki, K_1 eta K_2 zenbaki konplexuen arteko biderkadura beste bektore bat da. Emaitzaren modulua K_1 eta K_2 bektoreen moduluen arteko biderkadura da, eta emaitzaren θ argumentua K_1 eta K_2 bektoreen angeluen batura da. Ikus hurrengo irudia:

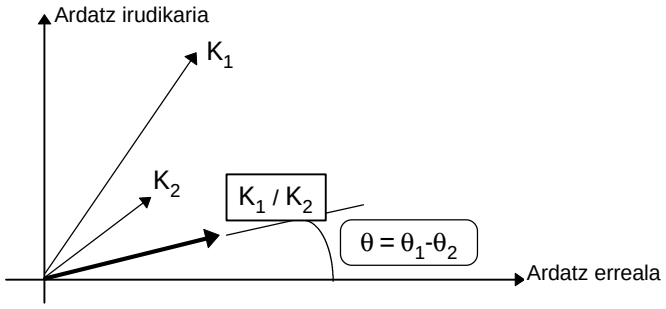


Zatiketa:

K_1 eta K_2 zenbaki konplexuen arteko zatiketa erdiesteko:

$$K_1 / K_2 = \{ [(R_1 \bullet R_2) + (I_1 \bullet I_2)] / (R_2^2 + I_2^2) \} + j\{ [(I_1 \bullet R_2) - (R_1 \bullet I_2)] / (R_2^2 + I_2^2) \}$$

Non, $K_1 = R_1 + I_1\, j$ $K_2 = R_2 + I_2\, j$ eta, $R_2^2 + I_2^2 \neq 0$



Lau eragiketa aritmetikoaz gain, zenbaki konplexuek berezko dituzten operazioak onartuko dituzte. Operazio berezi hauek zenbaki konplexuak irakurtzeko eta idazteko balioko dute, eta zenbaki konplexu baten *konjokatua* (ardatz errealarekiko simetrikoa) ematen duen errutina ere definituko da.

Zenbaki konplexuak lantzen dituen unitatearen beharkizunak ondoko taulan biltzen dira, konturatu, praktikan, `ZbkKonpl` unitatearen bitartez datu-mota berri bat eskaintzen zaio bezero-programari. Kontura gaitezan taularen **Egitura** delako tokian aldagai irudikariak edukiko duen itxura zehazten dela, eta beste aukerarik egon arren erregistro bat hautatu dugula zenbaki konplexuaren adierazpiderako.

Hona hemen `ZbkKonpl` unitateak beteko dituen beharkizunak:

`ZbkKonpl` unitatearen espezifikazioa

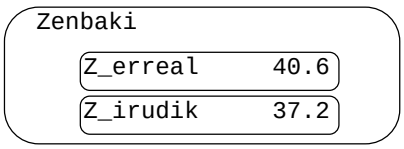
Egitura: Zenbaki konplexu baten ideia ondoen isladatzen duen datu egiturarik, zenbaki errealen bi eremuko erregistroa[16] litzateke. Lehenengo eremua zenbaki konplexuaren alde errealari dagokio, eta `Z_erreal` deituko da. Bigarren eremuak berriz, zenbaki konplexuaren alde irudikaria hartuko du eta `Z_irudik` deituko da.

Eragiketak: Hauek dira unitateari loturik dauden operazioak:

- Zenbaki konplexu jakin bati dagozkion koordenatu cartesiarrak teklatutik irakurri
- Teklatutik zenbaki konplexu bat irakurri koordenatu polarretan
- Zenbaki konplexu jakin batek dituen koordenatu cartesiarrak pantailan erakutsi
- Zenbaki konplexu batek dituen koordenatu polarrak pantailan erakutsi
- Zenbaki konplexu baten osagai erreala eskuratu
- Zenbaki konplexu baten osagai irudikaria eskuratu
- Zenbaki konplexu bat konposatu
- Zenbaki konplexu baten konjokatua lortu
- Zenbaki konplexu baten modulua eskuratu
- Zenbaki konplexu baten argumentua eskuratu
- Bi zenbaki konplexuen batura kalkulatu
- Bi zenbaki konplexuen kendura kalkulatu
- Bi zenbaki konplexuen biderkadura kalkulatu
- Bi zenbaki konplexuen zatidura kalkulatu

Zenbaki konplexuaren adierazpideari buruz, memorian izango duen egiturari buruz, laugarren kapituluan aipatutako `Record` datu-mota gogoratzea komeniko litzateke konkretuki erregistro baten osagaiak nola erreferentziatzen diren (**4.6.3 RECORD datu-mota** puntua).

```
Zenbaki.Z_erreal := 40.6 ;
Zenbaki.Z_irudik := 37.2 ;
```



Zerrenda irakurtzetik ondoriotzen den bezala, zenbaki konplexuek onartuko dituzten eragiketak bi mota nagusitan bana daitezke. Batetik sarrrea/irteera eragiketak (koordenatu sistema cartesiarran edo polarrean zenbaki konplexua teklatutik irakurri, zein pantailaratu), zenbaki konplexu baten osagaiak eskuratu edo zenbaki konplexu bati dagokion konjokatua lortu. Bestetik ohizko lau eragiketa aritmetikoak definitu dira ere.

11.4.2 Zenbaki konplexuen unitatearen interfazea

`ZbkKonpl` unitatearen beharkizunak edo espezifikazioa zehaztu ondoren hau litzateke dagokion interfazea:

[16] Diseinu fase honetan soluzio bat baino gehiago egon daiteke. esate baterako, zenbaki konplexuak bi osagaiz konposaturik daudenez beste adierazpide interesgarri bat irakuleari bururatuko zitzaion hau izan daiteke, hots bi kopuru erreal biltzen dituen array batez ordezka daiteke edozein zenbaki konplexu.

`ZbkKonpl` unitatearen interfazea

```
INTERFACE

   TYPE
      Konplexu = RECORD
                    Z_erreal : Real;
                    Z_irudik : Real;
                 END;
   PROCEDURE IrakurCartes (VAR ZbkKonpl : Konplexu);
   PROCEDURE IrakurPolar (VAR ZbkKonpl : Konplexu);
   PROCEDURE BistaraCartes (ZbkKonpl : Konplexu);
   PROCEDURE BistaraPolar (ZbkKonpl : Konplexu);
   FUNCTION OsagaiErreal (ZbkKonpl : Konplexu) : Real;
   FUNCTION OsagaiIrudikari (ZbkKonpl : Konplexu) : Real;
   PROCEDURE LorKonplexu (Erre, Irud : Real; VAR ZbkKonpl : Konplexu);
   PROCEDURE LorKonjokatu (ZbkKonpl : Konplexu; VAR Konjokatua : Konplexu);
   FUNCTION Modulua (ZbkKonpl : Konplexu) : Real;
   FUNCTION Angelua (ZbkKonpl : Konplexu) : Real;
   PROCEDURE Batuketa (Zbk1, Zbk2 : Konplexu; VAR Batura : Konplexu);
   PROCEDURE Kenketa (Zbk1, Zbk2 : Konplexu; VAR Kendura : Konplexu);
   PROCEDURE Biderketa (Zbk1, Zbk2 : Konplexu; VAR Biderkadura : Konplexu);
   PROCEDURE Zatiketa (Zbk1, Zbk2 : Konplexu; VAR Zatidura : Konplexu);
```

11.4.3 Zenbaki konplexuen unitatearen inplementazioa

Interfazean plazaturiko datu-mota eta azpirrutinez gain, radian eta gradu arteko bihurketak burutzen dituzten funtzioak beharrrezkoak direnez unitatearen inplementazioan zehaztuko dira (`fGrad_Bihur_Rad()` eta `fRad_Bihur_Gradu()` funtzioak pribatuak dira).

`ZbkKonpl` unitatearen inplementazioa

```
IMPLEMENTATION


{-------------------------------------------------------------------}
{ Gradutan dagoen angelu bat radianetan bihurtu.                    }
{-------------------------------------------------------------------}
FUNCTION fGrad_Bihur_Rad (Graduak:Real):Real;
{
  ONDORIOA:
           Radianetan itzultzen du gradutan sartutako angelua.
}
BEGIN
     fGrad_Bihur_Rad := PI*Graduak/180;
END;


{-------------------------------------------------------------------}
{ Radianetan dagoen angelu bat gradutan bihurtu.                    }
{-------------------------------------------------------------------}
FUNCTION fRad_Bihur_Gradu (Radianak:Real):Real;
{
  ONDORIOA:
           Gradutan itzultzen du radianetan sartutako angelua.
}
BEGIN
     fRad_Bihur_Gradu := 180*Radianak/PI;
END;
```

```
{--------------------------------------------------------------------}
{ Teklatutik irakurri zenbaki konplexu bat koordenatu cartesiarretan.}
{--------------------------------------------------------------------}
PROCEDURE IrakurCartes (VAR ZbkKonpl : Konplexu);
{
  ONDORIOA:
           ZbkKonpl-ek teklatutik irakurritako zenbaki konplexu
           baten osagai cartesiar biak barneratzen ditu.
}
BEGIN
     Write ('Osagai erreala sar ezazu: ');
     Readln (ZbkKonpl.Z_erreal);
     Write ('Osagai irudikaria sar ezazu: ');
     Readln (ZbkKonpl.Z_irudik);
END;


{--------------------------------------------------------------------}
{ Teklatutik irakurri zenbaki konplexu bat koordenatu polarretan.  }
{--------------------------------------------------------------------}
PROCEDURE IrakurPolar (VAR ZbkKonpl : Konplexu);
{
  ONDORIOA:
           ZbkKonpl-ek teklatutik polarretan irakurritako zenbaki
           konplexu baten osagai castesiar biak barneratzen ditu.
}
VAR
   Erradio, Angelu : Real;
BEGIN
     Write ('Modulua sar ezazu: ');
     Readln (Erradio);
     Write ('Angelua gradutan sar ezazu: ');
     Readln (Angelu);
     Angelu := fGrad_Bihur_Rad (Angelu);
     ZbkKonpl.Z_erreal := Erradio*Cos(Angelu);
     ZbkKonpl.Z_irudik := Erradio*Sin(Angelu);
END;


{--------------------------------------------------------------------}
{ Zenbaki konplexu bat koordenatu cartesiarretan pantailaratu.     }
{--------------------------------------------------------------------}
PROCEDURE BistaraCartes (ZbkKonpl : Konplexu);
{
  AURREBALDINTZA:
           ZbkKonpl aldagaiaren hasieraketa egin da.
  ONDORIOA:
           ZbkKonpl parametroa osatzen duten balioak pantailaratzen
           dira, koordenatu cartesiarrak bistaratuz.
}
BEGIN
     Writeln ('Osagai erreala: ',ZbkKonpl.Z_erreal:0:2);
     Writeln ('Osagai irudikaria: ',ZbkKonpl.Z_irudik:0:2);
END;


{--------------------------------------------------------------------}
{ Koordenatu polarretan pantailaratu zenbaki konplexu bat.         }
{--------------------------------------------------------------------}
PROCEDURE BistaraPolar (ZbkKonpl : Konplexu);
{
  AURREBALDINTZA:
           ZbkKonpl aldagaiaren hasieraketa egin da.
  ONDORIOA:
           ZbkKonpl parametroa osatzen duten balioak pantailaratzen
           dira, koordenatu polarrak bistaratuz.
}
VAR
   Angelu : Real;
```

```
BEGIN
   Writeln('Modulua: ',Sqrt(Sqr(ZbkKonpl.Z_erreal)+Sqr(ZbkKonpl.Z_irudik)):0:2);
   Write ('Angelua (gradutan): ');
   IF ZbkKonpl.Z_erreal = 0 THEN
      IF ZbkKonpl.Z_irudik < 0 THEN
                                  Angelu := 270
      ELSE
         Angelu := 90
   ELSE
      BEGIN
         Angelu := fRad_Bihur_Gradu(Arctan(ZbkKonpl.Z_irudik/ZbkKonpl.Z_erreal));

         {Angeluaren zuzenketa Arctan funtzioak bueltatzen dituen balioak
                        + PI/2 , -PI/2 tartean baitaude                  }
         IF ZbkKonpl.Z_erreal < 0 THEN
                                     Angelu := Angelu + 180
         ELSE
            Angelu := 360*Frac((Angelu + 360)/360);
      END;
   Writeln (Angelu:0:2)
END;


{---------------------------------------------------------------------}
{ Zenbaki konplexu baten osagai erreala eskuratu.                     }
{---------------------------------------------------------------------}
FUNCTION OsagaiErreal (ZbkKonpl : Konplexu) : Real;
{
  AURREBALDINTZA:
            ZbkKonpl aldagaiaren hasieraketa egin da.
  ONDORIOA:
            ZbkKonpl-aren osagai erreala itzultzen du.
}
BEGIN
     OsagaiErreal := ZbkKonpl.Z_erreal;
END;


{---------------------------------------------------------------------}
{ Zenbaki konplexu baten osagai irudikaria eskuratu.                  }
{---------------------------------------------------------------------}
FUNCTION OsagaiIrudikari (ZbkKonpl : Konplexu) : Real;
{
  AURREBALDINTZA:
            ZbkKonpl aldagaiaren hasieraketa egin da.
  ONDORIOA:
            ZbkKonpl-aren osagai irudikaria itzultzen du.
}
BEGIN
     OsagaiIrudikari := ZbkKonpl.Z_irudik;
END;


{---------------------------------------------------------------------}
{ Osagai erreala eta irudikaria emanik zenbaki konplexua lortu.     }
{---------------------------------------------------------------------}
PROCEDURE LorKonplexu (Erre, Irud : Real; VAR ZbkKonpl : Konplexu);
{
  AURREBALDINTZA:
            Erre eta Irud aldagaien hasieraketak egin dira.
  ONDORIOA:
            ZbkKonpl aldagaiak Erre eta Irud barneraturik dauzka.
}
BEGIN
     ZbkKonpl.Z_erreal := Erre;
     ZbkKonpl.Z_irudik := Irud;
END;
```

```
{-------------------------------------------------------------------}
{ Zenbaki konplexu baten konjokatua kalkulatu.                      }
{-------------------------------------------------------------------}
PROCEDURE LorKonjokatu (ZbkKonpl : Konplexu; VAR Konjokatua : Konplexu);
{
  AURREBALDINTZA:
           ZbkKonpl aldagaiaren hasieraketa egin da.
  ONDORIOA:
           Konjokatua aldagaia ZbkKonpl-aren konjokatua da.
}
BEGIN
     Konjokatua.Z_erreal := ZbkKonpl.Z_erreal;
     Konjokatua.Z_irudik := -ZbkKonpl.Z_irudik;
END;


{-------------------------------------------------------------------}
{ Zenbaki konplexu baten modulua eskuratu.                          }
{-------------------------------------------------------------------}
FUNCTION Modulua (ZbkKonpl : Konplexu) : Real;
{
  AURREBALDINTZA:
           ZbkKonpl aldagaiaren hasieraketa egin da.
  ONDORIOA:
           ZbkKonpl konplexuaren modulua itzultzen du.
}
BEGIN
     Modulua := Sqrt(Sqr(ZbkKonpl.Z_erreal)+Sqr(ZbkKonpl.Z_irudik));
END;


{-------------------------------------------------------------------}
{ Zenbaki konplexu baten argumentua eskuratu.                       }
{-------------------------------------------------------------------}
FUNCTION Angelua (ZbkKonpl : Konplexu) : Real;
{
  AURREBALDINTZA:
           ZbkKonpl aldagaiaren hasieraketa egin da.
  ONDORIOA:
           ZbkKonpl bektorearen eta x ardatz positiboaren arteko
           angelua itzultzen du. Angelua gradutan emanik dator.
}
VAR
   Ang : Real;
BEGIN
     Ang := fRad_Bihur_Gradu(Arctan(ZbkKonpl.Z_irudik/ZbkKonpl.Z_erreal));
         {Angeluaren zuzenketa Arctan funtzioak bueltatzen dituen balioak
                        + PI/2 , -PI/2 tartean baitaude                 }
         IF ZbkKonpl.Z_erreal < 0 THEN Ang := Ang + 180
                                  ELSE Ang := 360*Frac((Ang + 360)/360);
     Angelua := Ang;
END;


{-------------------------------------------------------------------}
{ Bi zenbaki konplexuen batura kalkulatu.                           }
{-------------------------------------------------------------------}
PROCEDURE Batuketa (Zbk1, Zbk2 : Konplexu; VAR Batura : Konplexu);
{
  AURREBALDINTZA:
           Zbk1 eta Zbk2 aldagaien hasieraketak egin dira.
  ONDORIOA:
           Batura aldagaiak (Zbk1+Zbk2) barneratzen du.
}
BEGIN
     Batura.Z_erreal := Zbk1.Z_erreal + Zbk2.Z_erreal;
     Batura.Z_irudik := Zbk1.Z_irudik + Zbk2.Z_irudik;
END;
```

```
{-------------------------------------------------------------------}
{ Bi zenbaki konplexuen kendura kalkulatu.                          }
{-------------------------------------------------------------------}
PROCEDURE Kenketa (Zbk1, Zbk2 : Konplexu; VAR Kendura : Konplexu);
{
  AURREBALDINTZA:
           Zbk1 eta Zbk2 aldagaien hasieraketak egin dira.
  ONDORIOA:
           Kendura aldagaiak (Zbk1-Zbk2) barneratzen du.
}
BEGIN
     Kendura.Z_erreal := Zbk1.Z_erreal - Zbk2.Z_erreal;
     Kendura.Z_irudik := Zbk1.Z_irudik - Zbk2.Z_irudik;
END;


{-------------------------------------------------------------------}
{ Bi zenbaki konplexuen biderkadura kalkulatu.                      }
{-------------------------------------------------------------------}
PROCEDURE Biderketa (Zbk1, Zbk2 : Konplexu; VAR Biderkadura : Konplexu);
{
  AURREBALDINTZA:
           Zbk1 eta Zbk2 aldagaien hasieraketak egin dira.
  ONDORIOA:
           Biderkadura aldagaiak (Zbk1*Zbk2) barneratzen du.
}
BEGIN
   Biderkadura.Z_erreal := Zbk1.Z_erreal*Zbk2.Z_erreal -
                           Zbk1.Z_irudik*Zbk2.Z_irudik;
   Biderkadura.Z_irudik := Zbk1.Z_erreal*Zbk2.Z_irudik +
                           Zbk1.Z_irudik*Zbk2.Z_erreal;
END;


{-------------------------------------------------------------------}
{ Bi zenbaki konplexuen zatiketa kalkulatu.                         }
{-------------------------------------------------------------------}
PROCEDURE Zatiketa (Zbk1, Zbk2 : Konplexu; VAR Zatidura : Konplexu);
{
  AURREBALDINTZA:
           Zbk1 eta Zbk2 aldagaien hasieraketak egin dira.
  ONDORIOA:
           Zatidura aldagaiak  (Zbk1/Zbk2) barneratzen du
}
VAR
   Konj_Zbk2, Biderk1, Biderk2 : Konplexu;
BEGIN
     LorKonjokatu (Zbk2, Konj_Zbk2);
     Biderketa (Zbk1, Konj_Zbk2, Biderk1);
     Biderketa (Zbk2, Konj_Zbk2, Biderk2);
     IF Biderk2.Z_erreal <> 0 THEN
       BEGIN
         Zatidura.Z_erreal := Biderk1.Z_erreal/Biderk2.Z_erreal;
         Zatidura.Z_irudik := Biderk1.Z_irudik/Biderk2.Z_erreal;
       END
     ELSE
       BEGIN
         Writeln ('Ezin da zatidura kalkulatu');
         Zatidura.Z_erreal := 0;
         Zatidura.Z_irudik := 0;
       END;
END;


BEGIN
       Writeln ('DMA_konpl unitatearen hasieraketa');
       ReadLn;
END.
```


11.4.4 Zenbaki konplexuen unitatea erabiltzen

`ZbkKonpl` unitatea erabiltzeko sei programa prestatu izan ditugu, guztietan zenbaki konplexuen arteko kalkulu errazak egin eta emaitza VGA pantaila batean grafikoki erakusten da. Jarraian programa bakoitzaren izen eta deskribapen laburra ematen da, hauen exekuzio zuzena lortzeko txartel grafikoaren kontrolagailua `C:\TP70\BGI` direktorioan aurkitzen dela suposatu da:

`ZenbakiKonplexuBatMarraztu` programa honek zenbaki konplexu baten bi koordenatu kartesiarrak teklatuz irakurri ondoren sistema polarrera igarotzen ditu. Ondoren sistema grafikoa irekitzen da `Ardatzak()` eta `Lerroa()` prozedurak deitu zenbakiaren adierazpide bektoriala bistaratu ahal izateko.

`Ardatzak()` prozeduraren bitartez `X-Y` ardatzak marraztu egiten dira. `Lerroa()` delako prozedurak berriz, teklatuz emaniko zenbaki konplexuari dagokion marra jartzen du pantailan.

`ZenbakiKonplexuBatenBektorea` aurrekoak bezala programa honek ere zenbaki konplexu baten bi koordenatu kartesiarrak teklatuz irakurri ondoren sistema polarrera igarotzen ditu. Gero sistema grafikoa irekitzen da `Ardatzak()` eta `Bektorea()` prozedurak deitu ahal izateko.

`Ardatzak()` prozeduraren bitartez `X-Y` ardatzak marraztu egiten dira.

`Bektorea()` delako prozedurak sarrerako zenbaki konplexuari dagokion gezia jartzen du pantailan, horretarako Turbo Pascal-aren `FillPoly()` prozedura erabiltzen du zeinek poligono baten erpinak hartzen dituen.

`ZenbakiKonplexuakBatzen` programa honek zenbaki konplexu biren lau koordenatu kartesiarrak teklatuz irakurri ondoren sistema polarrera igaro eta euren batura kalkulatzen du. Ondoren sistema grafikoa irekitzen da sarrerako zenbaki konplexuak eta dagokien batura grafikoki bistaratu ahal izateko.

`ZenbakiKonplexuakBiderkatzen` programa honek zenbaki konplexu biren lau koordenatu kartesiarrak teklatuz irakurri ondoren sistema polarrera igarotzen ditu eta euren biderkadura kalkulatu egiten du. Ondoren sistema grafikoa irekitzen da sarrerako zenbaki konplexuak eta dagokien biderkadura grafikoki bistaratu ahal izateko.

`ZenbakiKonplexuarenKonjokatua` programa honek zenbaki konplexu baten bi koordenatu kartesiarrak teklatuz irakurri ondoren dagokion zenbaki konplexu konjokatua lortzen du. Sistema grafikoa irekiz sarrerako zenbakia eta bere konjokatuaren adierazpide bektorialak erakusten dira.

`ZenbakiKonplexuenZatiketa` programa honek zenbaki konplexu biren lau koordenatu kartesiarrak teklatuz irakurri ondoren sistema polarrera igaro eta euren zatidura kalkulatzen du. Kasu honetan, eta ondoren sistema grafikoa irekitzen da sarrerako zenbaki konplexuak eta dagokien zatidura grafikoki bistaratu nahi delako, ez da `Grafiko` unitatetik hartutako `Zatiketa()` prozedura erabiltzen horren ordez zatidura urratsez urrats kalkulatzen da.

11.5 PROGRAMAK

Hona hemen 11. kapituluaren programak orrialdeen arabera sailkaturik:

Izena	*Programaren identifikadorea*	*ORRI.*	*Ikasgaia*
RECORD01.PAS	ArrayParaleloakLantzen	11-04	Record datu-mota
RECORD02.PAS	ErregistroaDatuaBetetzen	11-08	Record datu-mota
RECORD03.PAS	ErregistroarenEremuakMemorian	11-10	Record datu-mota
RECORD04.PAS	ErregistroenEsleipena	11-12	Record datu-mota
RECORD05.PAS	ErregistroenEsleipenOkerra	11-13	Record datu-mota
RECORD06.PAS	ErregistroaParametroBezala	11-14	Record datu-mota
RECORD07.PAS	ErregistroBiBerdinakOteDiren	11-16	Record datu-mota
RECORD08.PAS	ErregistroenArraya	11-19	Record datu-mota
RECORD09.PAS	ArrayenErregistroa	11-22	Record datu-mota
IKASLE.PAS	Erregistroen_Adibidea	11-24	Record datu-mota
LIBURU.PAS	Biblioteka_Kudeaketa	11-28	Record datu-mota
HASIERAK.PAS	AldagaienHasieraketa	11-35	Record datu-mota
RECORD10.PAS	ErregistroenHasieraketa	11-36	Record datu-mota
RECORD11.PAS	ErregistroKabiatuak	11-38	Record datu-mota
RECORD12.PAS	WithKabiatuak	11-40	Record datu-mota
RECORD13.PAS	WithKabiatuakZalantzazko	11-42	Record datu-mota
RECORD14.PAS	ErregistroAldakorrak0	11-45	Record datu-mota
RECORD15.PAS	ErregistroAldakorrenArrayak	11-46	Record datu-mota
RECORD16.PAS	ErregistroAldakorrenHelbideak	11-46	Record datu-mota
RECORD17.PAS	ErregistroAldakorrak1	11-48	Record datu-mota
RECORD18.PAS	ErregistroAldakorrak2	11-50	Record datu-mota
RECORD19.PAS	ErregistroAldakorrak3	11-52	Record datu-mota
ERREGI1.PAS	ErregistroAldakorrakAdib1	11-53	Record datu-mota
ERREGI2.PAS	ErregistroAldakorrakAdib2	11-54	Record datu-mota
SET_01.PAS	MultzoBatenDefinizioa	11-55	Set datu-mota
SET_02.PAS	MultzoenDefinizioa	11-56	Set datu-mota
SET_03.PAS	MenuaMultzoekin	11-57	Set datu-mota
SET_04.PAS	AzpiEtaGainMultzoak	11-59	Set datu-mota
SET_05.PAS	MultzoenBerdintasunak	11-60	Set datu-mota
SET_06.PAS	MultzoenBildura	11-61	Set datu-mota
SET_07.PAS	MultzoenEbakidura	11-62	Set datu-mota
SET_08.PAS	MultzoenOsagarria	11-62	Set datu-mota
SET_09.PAS	MultzoenDiferentzia	11-63	Set datu-mota
SET_10.PAS	MultzoenProzesaketa	11-64	Set datu-mota
ZBKKONPL.PAS ZKP_ADI1.PAS	ZbkKonpl ZenbakiKonplexuBatMarraztu	11-79	Konplexuen unitatea
ZBKKONPL.PAS ZKP_ADI2.PAS	ZbkKonpl ZenbakiKonplexuBatenBektorea	11-79	Konplexuen unitatea
ZBKKONPL.PAS ZKP_ADI3.PAS	ZbkKonpl ZenbakiKonplexuakBatzen	11-79	Konplexuen unitatea
ZBKKONPL.PAS ZKP_ADI4.PAS	ZbkKonpl ZenbakiKonplexuakBiderkatzen	11-79	Konplexuen unitatea
ZBKKONPL.PAS ZKP_ADI5.PAS	ZbkKonpl ZenbakiKonplexuarenKonjokatua	11-79	Konplexuen unitatea
ZBKKONPL.PAS ZKP_ADI6.PAS	ZbkKonpl ZenbakiKonplexuenZatiketa	11-79	Konplexuen unitatea

11.6 BIBLIOGRAFIA

- Borland, *TURBO PASCAL 7.0. Erabiltzailearen gida*, Borland International, 1992
- Borland, *TURBO PASCAL 7.0 Ingurunearen laguntza*, Borland International, 1992
- Leetsma, S., Nyhoff, L., *Programación en Pascal*, Prentice Hall, Madrid, 1999

12. ATALA: FILE ETA TEXT DATU-MOTAK

AURKIBIDEA

12.1 FILE DATU-MOTAREN SARRERA

Orain arte egindako programek datu gutxi behar izaten zuten, emaitzak ere gutxi ziren eta pantailaratu egiten ziren. Orain arteko adibideetan programei datuak emateko `Read` eta `ReadLn` prozeduraz egiten zen, programetatik informazioa kanporatzeko `Write` eta `WriteLn` prozedurak erabiltzen ohituta gaude.

Baina zenbait programetan prozesatu beharreko informazioa ugaria izaten da. Halako kasuetan datuak teklatuz ematea ezinezkoa bihurtzen da, bestalde programak lortzen dituen emaitzak pantaila askotan sakabanatzen badira ezinezkoa da horretaz jabetzea. Beraz, zenbait programetan datuak memorian kokatzeko teklatutik hartu beharrean kanpoan aurkitzen den fitxategietatik[1] hartuko dira. Ordenadorearen barne osagairik garrantzitsuenak hiru dira (Memoria, Unitate Aritmetiko-lógikoa eta Kontrol Unitatea) eta dakigunez programa bera eta programak darabiltza datuak Memorian kokatzen dira, baina programa exekutatzera goazenean programaren sententziak ordenadorearen barne egituratik kanpo daude (programa `EXE` luzapena duen fitxategia izango da eta datuak exekuzio denboran ematen dira); horregatik operadore/ordenadore komunikazioa ezinbestekoa da eta komunikazio hori, ikusiko dugun bezala, fitxategietan oinarritzen da.

Egia esan, nahiz eta ohartu ez fitxategiak gure lehen programatik erabili ditugu. Izan ere, teklatua eta monitorea benetan fitxategiak dira:

- Teklatutik irakurtzen diren datuak `input` izeneko fitxategi estandarretik hartzen dira (**12.2.2 `input` eta `output` fitxategi estandarrak** puntua).
- Monitorean edo inprimagailuan idazten dena `output` izeneko fitxategi estandarren bitartez egiten da (**12.2.2 `input` eta `output` fitxategi estandarrak** puntua).

Aurrera jarraitu baino lehen kapitulu honen izenburua oraintxe azaltzea bidezkoa iruditzen zaigu, Pascal lengoaiaren file eta text datu-motak fitxategiekin loturik badaude zergatik dira bi eta zein da euren arteko ezberdintasuna. Pascal lengoaian aurredefiniturik dagoen file datu-mota *fitxategi bitar*rekin erlazionatuko dugu, eta text datu-motak berriz *testu-fitxategi*ekin zerikusia du. Bai file eta bai text datu-motak memorian dagoen datu bat fitxategi batean gordetzeko balio dute, gertatzen dena modu ezberdinean gordetzen dutela, diferentzi horretaz jabetzeko aspaldiko `ZenbakiOsoenBarneAdierazpidea` eta `KaraktereenBarneAdierazpidea` programak gogora ekarriko ditugu eta euren aldaera bat eginez jarraian ematen den programa lortuko dugu:

```
PROGRAM FileTextBarneAdierazpidea ;        { \TP70\12\FILETEXT.PAS }
USES
   Crt ;

PROCEDURE FitxategiBitar (Zenbak : Integer) ;
VAR
   i : Integer ;
   Konta : Word ;
BEGIN
   Konta := MAXINT+1 ;
   Write (Zenbak, 'ren barne adierazpidea fitxategi bitar batean:  ') ;

   FOR i:= 16 DOWNTO 1 DO
   BEGIN
      IF (Konta AND Zenbak) = 0 THEN Write ('0')
      ELSE Write ('1') ;
      Konta := Konta DIV 2 ;
      IF (i+3) MOD 4 = 0 THEN Write (' ')
   END ;
   Writeln ;
END ;
```

[1] Fitxategi kontzeptua hurrengo puntuan definituko dugu.

```
PROCEDURE TestuFitxategi (Zenbak : Integer) ;
VAR
   i, j, Kopurua, Konta : Integer ;
   Katea : String ;
BEGIN
   Str (Zenbak, Katea) ;
   WriteLn (Zenbak, 'ren barne adierazpidea testu fitxategi batean:  ') ;

   FOR i:=1 TO Length (Katea) DO
   BEGIN
      Kopurua := Ord (Katea[i]) ;
      Konta := 256 ;
      Write (Katea[i]:28, 'ren ASCII adierazpidea:  ');

      FOR j:=8 DOWNTO 1 DO
      BEGIN
         IF (Konta AND Kopurua) = 0 THEN Write ('0')
         ELSE Write ('1');
         Konta := Konta DIV 2 ;
         IF j = 5 THEN Write (' ')
      END ;
      WriteLn ;
   END ;
END ;

VAR                                          { PROGRAMA NAGUSIA }
   Zenbakia : Integer ;

BEGIN
   ClrScr ;
   REPEAT
      Write ('Zenbaki osoa eta positiboa sartu: ') ;
      ReadLn (Zenbakia) ;
   UNTIL Zenbakia >= 0 ;

   WriteLn ;
   FitxategiBitar (Zenbakia) ;

   WriteLn ;
   TestuFitxategi (Zenbakia) ;
END.
```

`FileTextBarneAdierazpidea` izeneko programan osoa den zenbaki bat irakurtzen da teklatuz eta programa nagusiko `Zenbakia` aldagaian biltegitzen da, ondoren zenbaki hori parametrotzat harturik `FitxategiBitar` eta `TestuFitxategi` prozeduren deiak burutzen dira. `FitxategiBitar` prozeduraz zenbakiaren barne adierazpena pantailaratzen da (`Integer` datu-motak 16 bit hartzen dituela memorian oroitu). `TextFitxategi` prozeduraz zenbakiaren barne adierazpena pantailaratzen da ere baina ez `Integer` bati dagokion adierazpide bitarra, baizik kopurua osatzen duten digituei dagozkien ASCII kodearen adierazpidea.

`FileTextBarneAdierazpidea` programaren balizko exekuzio bat azter dezagun. Demagun `Zenbakia` aldagaian `27914` kopurua biltegitzen dela, hona hemen adibide-programaren irteera:

```
Zenbaki osoa eta positiboa sartu: 27914

27914ren barne adierazpidea fitxategi bitar batean:  0110 1101 0000 1010

27914ren barne adierazpidea testu fitxategi batean:
                           2ren ASCII adierazpidea:  0001 1001
                           7ren ASCII adierazpidea:  0001 1011
                           9ren ASCII adierazpidea:  0001 1100
                           1ren ASCII adierazpidea:  0001 1000
                           0ren ASCII adierazpidea:  0001 1010
_
```

`FitxategiBitar` prozedurak `27914` zenbakirako ematen duen adierazpidea (hobeto ikuste arren laukotetan banaturik) hau da: `0110 1101 0000 1010`. Kopuru bitar horri dagokion bitar-hamartar bihurketa egin dezagun (bigarren kapituluan ikusitako **2.4.2.3 2rako osagarria** puntua gogoratu):

$$\begin{aligned}
\mathbf{0}x2^{15} + \mathbf{1}x2^{14} + \mathbf{1}x2^{13} + \mathbf{0}x2^{12} &= 0 + 16384 + 8192 + 0 = 24576\\
\mathbf{1}x2^{11} + \mathbf{1}x2^{10} + \mathbf{0}x2^{9} + \mathbf{1}x2^{8} &= 2048 + 1024 + 0 + 256 = 3328\\
\mathbf{0}x2^{7} + \mathbf{0}x2^{6} + \mathbf{0}x2^{5} + \mathbf{0}x2^{4} &= 0 + 0 + 0 + 0 = 0\\
\mathbf{1}x2^{3} + \mathbf{0}x2^{2} + \mathbf{1}x2^{1} + \mathbf{0}x2^{0} &= 8 + 0 + 2 + 0 = 10\\
&\qquad\qquad\qquad\qquad\qquad\qquad 27914
\end{aligned}$$

`TestuFitxategi` prozedurak `27914` zenbaki berdinerako ematen duen adierazpidea digitu bakoitzari dagokion ASCII taulako kode bitarra litzateke. Adibidez `27914` zenbakiaren lehen digitua `'2'` karakterea da eta letra horri dagokion ASCII taulako ordinala 50 da, dakigunez 50 kode bitarrean `00110010` zortzikotea litzateke, eta hori da hain zuzen ere testu fitxategiak gordeko lukeena. Bigarren digituarekin jarraituz `'7'` karaktereari dagokion ASCII taulako ordinala 55 da eta bere barne adierazpidea `00110111` litzateke ...

`Ord('2') = 50`

$$50 = 0 + 0 + 32 + 16 + 0 + 0 + 2 + 0 = \mathbf{0}x2^{7} + \mathbf{0}x2^{6} + \mathbf{1}x2^{5} + \mathbf{1}x2^{4} + \mathbf{0}x2^{3} + \mathbf{0}x2^{2} + \mathbf{1}x2^{1} + \mathbf{0}x2^{0}$$

`Ord('7') = 55`

$$55 = 0 + 0 + 32 + 16 + 0 + 4 + 2 + 1 = \mathbf{0}x2^{7} + \mathbf{0}x2^{6} + \mathbf{1}x2^{5} + \mathbf{1}x2^{4} + \mathbf{0}x2^{3} + \mathbf{1}x2^{2} + \mathbf{1}x2^{1} + \mathbf{1}x2^{0}$$

`Ord('9') = 57`

$$57 = 0 + 0 + 32 + 16 + 8 + 0 + 0 + 1 = \mathbf{0}x2^{7} + \mathbf{0}x2^{6} + \mathbf{1}x2^{5} + \mathbf{1}x2^{4} + \mathbf{1}x2^{3} + \mathbf{0}x2^{2} + \mathbf{0}x2^{1} + \mathbf{1}x2^{0}$$

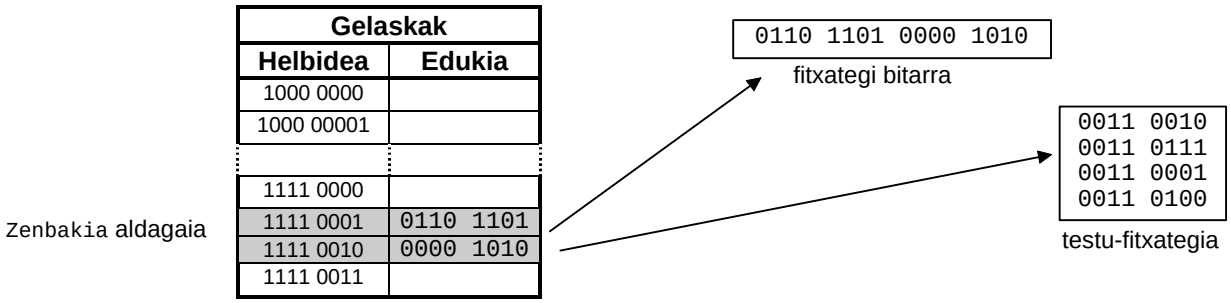
`Ord('1') = 49`

$$49 = 0 + 0 + 32 + 16 + 0 + 0 + 0 + 1 = \mathbf{0}x2^{7} + \mathbf{0}x2^{6} + \mathbf{1}x2^{5} + \mathbf{1}x2^{4} + \mathbf{0}x2^{3} + \mathbf{0}x2^{2} + \mathbf{0}x2^{1} + \mathbf{1}x2^{0}$$

`Ord('4') = 52`

$$52 = 0 + 0 + 32 + 16 + 0 + 4 + 0 + 0 = \mathbf{0}x2^{7} + \mathbf{0}x2^{6} + \mathbf{1}x2^{5} + \mathbf{1}x2^{4} + \mathbf{0}x2^{3} + \mathbf{1}x2^{2} + \mathbf{0}x2^{1} + \mathbf{0}x2^{0}$$

Beraz, ordenadorearen memorian aurkitzen den `Integer` datu-motako `27914` zenbakiak 16 bit izango ditu kode bitarrean, eta kopuru hori fitxategi bitar batetara transferitzean 16 biteko adierazpide berbera mantenduko du, baina testu-fitxategi batetara transferitzean banakako digituen adierazpide gordeko da. Eskematikoki:



Irudian ageri agerian azaltzen den bezala, fitxategi bitarrek biltegitzen dutena memoriaren isladapen zuzena da, `Integer` batek memorian bi byte hartzen baditu fitxategi bitar batean ere bi byte hartuko ditu, honez gero fitxategi bitarrak trinkoagoak[2] dira testu-fitxategiak baino. Bestalde testu-fitxategi bat ASCII kodeak gordetzen dituenez `DOS` sistema eragileko `type` komandoz (edo `edit` programa editorearekin) pantailara daiteke, baina fitxategi bitar baten edukia pantailan ikusteko programa berezia egin behar da.

[2] Fitxategi bitar batean `27914` gordetzeko 16 bit behar dira (2 byte). Baina `27914` kopuru bera testu-fitxategi batean biltegitzean, gutxienez 40 bit (5 byte) behar izango dira.

12.1.1 Definizioa

Informazio homogeneo gordetzen duen bilduma bat da fitxategia Sistema Eragilerako, Sistema Eragileek fitxategi jakin batekin lan egitean identifikadore bakar batez izendatuko du. Fitxategi bat datu-mota bereko osagaiak dituen multzo bat da eta disko zein zintetan egon ohi da.

Fitxategiek array linealekin zerikusia dutelako jarraian datu-mota biren arteko alderaketa egingo dugu:

	Array	**Fitxategi**
Osagaiak	Osagaiak datu-mota berekoak dira	Osagaiak datu-mota berekoak dira
Atzipena	Osagai bat identifikatzeko indizea erabiliko da	Osagai bat identifikatzeko fitxategiak erakusle bat dauka definiturik
Tamaina	Luzera aurredefiniturik dago	Ez dago zertan aldez aurretik luzera definitu beharrik
Euskarria	Memorian kokatzen dira	Memoriatik kanpo kokatzen dira

Arrayak eta fitxategiak datu-mota egituratu homogeneoak kontsidera daitezke, euren osagaiak datu-mota berekoak baitira. Fitxategien definizioa laster ikusiko dugu baina aurrera dezagun arrayetan bezala `OF <osagaien datu-mota>` hitz erreserbatua agertuko dela.

Array baten elementua prozesatzeko bere indizea erabiltzen ikasi dugu, fitxategietan elementuak banan-banan prozesatzen dira eta hurrengo operazioan zein elementu landuko den zehazteko erakusle bat dago definiturik.

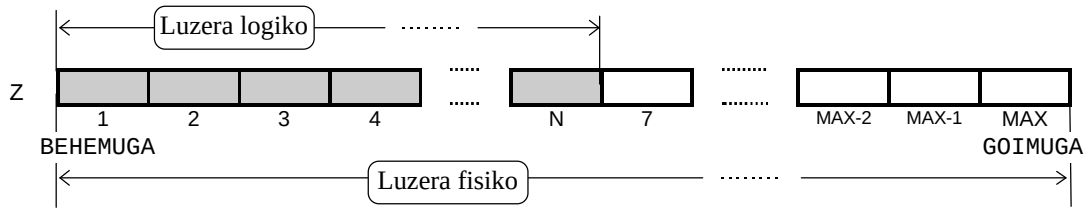
Arraya definitzean, bai aldagai bezala definitzean, edo bestela aldagaiari dagokion datu-mota ezartzen denean, arrayak izango duen luzera maximoa zehaztu egiten da; ondorioz array bat mugaturik dagoela esan dezakegu. Fitxategietan ez da tamaina maximorik definitzen eta fitxategi batek duen muga bere euskarrian datza (diskoa beterik egotea alegia).

Arrayak, eta orain arteko aldagai guztiak ere, memorian aurkitzen dira eta iheskorrak direla esan ohi da (ordenadorea elikatzen duen korronte elektrikoa desagertzean aldagaien edukiak galtzen dira). Fitxategiek berriz, korronte elektrikoaren menpekotasunik ez dutelako informazioa iraunkorki mantentzen dute.

Hona hemen zenbaki osoak gordetzen dituen array eta fitxategi baten eskemak:

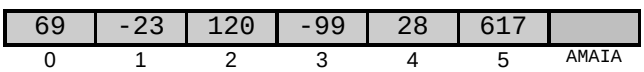
Arraya

```
VAR
   Z : ARRAY[1..80] OF LongInt ;
```



Fitxategia

```
VAR
   F : FILE OF LongInt ;
```



Fitxategiaren eskeman agertzen den egituran zenbaki osoak biltzen dira eta lehenago esan dugun bezala memoriatik kanpo aurkitzen da. Egitura horri *fitxategi fisiko* deritzo eta hura lantzeko beharrezkoa den `F` aldagaiari *fitxategi logiko* esaten zaio.

Fitxategi fisikoaren eta fitxategi logikoaren arteko desberdintasuna jarraian datozen **12.1.1.1** eta **12.1.1.2** puntuetan sakonduko dugu, orain fitxategi bat sortzen duen programa ikus dezagun:

```
PROGRAM FitxategiBatSortzen ;                  { \TP70\12\FILE_01.PAS }
USES
  Crt ;

TYPE
  DM_Fitxategi = FILE OF Real ;

VAR
   FitxIzen : String ;       { Fitxategi fisikoa }
   F : DM_Fitxategi ;        { Fitxategi logikoa }
   Elem : Real ;
   Kontag : LongInt ;
   Erantz : Char ;

BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   WriteLn ('Kopuru errealak ', FitxIzen, ' fitxategian gordetzen') ;

   Assign (F, FitxIzen) ;    { Fitxategi logikoa eta fisikoa elkartu }
   Rewrite (F) ;             { Idazketarako ireki }

   Kontag := 0 ;
   REPEAT
      Write ('Eman fitxategiaren ', Kontag, '. elementua: ') ;
      ReadLn (Elem) ;        { Zenbakia memorian dago }

      Write (F, Elem) ;      { Memoriatik fitxategira transferitu }

      Kontag := Kontag + 1 ;
      Write ('Gehiagorik? (B/E)  ') ;
      Erantz := ReadKey ;
      Erantz := UpCase (Erantz) ;
      WriteLn (Erantz) ;
   UNTIL Erantz = 'E' ;

   Close (F) ;               { Fitxategia itxi }
END.
```

`FitxategiBatSortzen` programaren bitartez zenbaki errealak gordeko dituen fitxategi bat sortuko dugu diskoan. Programaren hasieran fitxategiaren `DM_Fitxategi` datu-mota definitzen da eta aldagaien blokean berari dagokion `F` fitxategi logikoa. Baina zenbaki errealak ez dira fitxategi logikoaren barnean gordetzen fitxategi fisiko barruan baizik, fitxategi fisikoa zein izango den zehaztu beharko da eta horretarako diskoaren fitxategiak izango duen **izena**[3] finkatzea nahikoa da; horregatik `FitxategiBatSortzen` programan fitxategi fisikoak izango duen identifikadorea teklatuz sar dezala eskatzen zaio operadoreari.

Fitxategi fisikoa lantzeko `F` fitxategi logikoa erabiliko da eta euren arteko elkarketa burutzeko `Assign` prozedura estandarraren deia egiten da, gero fitxategia idazketarako irekitzen da eta azkenean, programa bukatu aurretik, fitxategia itxiko da. `FitxategiBatSortzen` izeneko programaren sententziarik garrantzitsuena une honetan beltzez jarrita dagoena da, horren bitartez teklatuz irakurri den zenbaki erreal bat fitxategian idazten[4] da.

[3] Fitxategi fisikoak diskoan edukiko duen izena `String` bat izango da.

[4] Ikus idazketa fitxategian `Write` prozeduraz lortzen dela, baina prozeduraren lehen parametroa fitxategia zehaztuz.

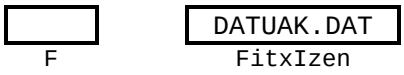
`FitxategiBatSortzen` programaren exekuzio bat hau litzateke:

```
Fitxategiaren izena eman: DATUAK.DAT
Kopuru errealak DATUAK.DAT fitxategian gordetzen
Eman fitxategiaren 0. elementua: 7.05
Gehiagorik? (B/E)  B
Eman fitxategiaren 1. elementua: 0.66
Gehiagorik? (B/E)  B
Eman fitxategiaren 2. elementua: 3.78
Gehiagorik? (B/E)  E
_
```

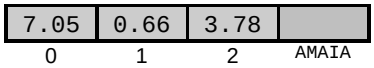
Horren arabera ordenadorearen diskoan `DATUAK.DAT` izeneko fitxategi bat sortuko da, hiru zenbaki baino ez denez eman fitxategi horrek izango duen tamaina 18 bytekoa izango da (memorian bezala, 6 byte zenbaki erreal bakoitzeko), eskematikoki:

```
VAR
   F : DM_Fitxategi ;        { Fitxategi logikoa }
   FitxIzen : String ;       { Fitxategi fisikoa }
```

Memorian



Diskoan



`FitxategiBatSortzen` programaren bitartez fitxategi bat sortu eta datuz bete izan dugu, baina fitxategi horren edukia pantailaratzeko beste programa bat beharko genuke. Hona hemen `FitxategiBatIkusten` programaren kodifikazioa:

```
PROGRAM FitxategiBatIkusten ;                  { \TP70\12\FILE_02.PAS }
TYPE
  DM_Fitxategi = FILE OF Real ;

VAR
   FitxIzen : String ;       { Fitxategi fisikoa }
   F : DM_Fitxategi ;        { Fitxategi logikoa }
   Elem : Real ;
   Kontag : LongInt ;

BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   WriteLn ('Kopuru errealak ', FitxIzen, ' fitxategitik pantailaratzen') ;

   Assign (F, FitxIzen) ;    { Fitxategi logikoa eta fisikoa elkartu }
   Reset (F) ;               { Irakurketarako ireki }

   Kontag := 0 ;
   WHILE NOT Eof (F) DO      { Fitxategia amaitzen ez den bitartean }
   BEGIN
      Read (F, Elem) ;       { Fitxategitik memoriara transferitu }

      WriteLn (Kontag, '. elementua = ', Elem:0:8) ;
      Kontag := Kontag + 1 ;
   END ;

   Close (F) ;               { Fitxategia itxi }
END.
```

`FitxategiBatIkusten` programaren sententziarik garrantzitsuena beltzez idatzi dena da (diskoan kokaturik dagoen fitxategitik elementu bat memoriara transferitzen duena). Lehen bezala eginkizun hori ezaguna zaigun `Read` prozedura estandarren bitartez egiten dela azpimarratuko genuke, oraingoan ere `Read` prozeduraren lehendabiziko parametrotzat fitxategiaren identifikadorea eman behar da.

`FitxategiBatIkusten` programan dauden azpiprograma berriak `Reset` prozedura eta `Eof` funtzioa lirateke, nahiz eta **12.1.2** puntuan sakonago ikusiko ditugun eman ditzagun orain horiei buruzko zenbait argibide. `Eof` funtzioa boolearra da, hots bere emaitza `TRUE` ala `FALSE` izango da, programaren `WHILE` egitura aztertuz ikasleak berak asma dezake `Eof` funtzioak noiz itzultzen duen `FALSE` balioa, horretarako kontutan izan `FitxategiBatIkusten` programaren ondoko irteera:

```
Fitxategiaren izena eman: DATUAK.DAT
Kopuru errealak DATUAK.DAT fitxategitik pantailaratzen
0. elementua = 7.05000000
1. elementua = 0.66000000
2. elementua = 3.78000000
_
```

`Reset` prozedurak fitxategia irakurketarako irekitzeko balio du eta `Rewrite` prozeduraren antzekoa da. `Rewrite`-k fitxategi fisikoa sortzen du eta ez du inoiz arazorik ematen, baina `Reset` bitartez fitxategi bat ireki nahi dugunean fitxategi hori aldez aurretik sortuta egongo da `Reset`-ek ez baitu fitxategirik sortzen. Horregatik fitxategiaren izena gaizki ematean exekuzio errorea eragingo da, ikus `FitxategiBatIkusten` programaren bigarren irteera hau:

```
Fitxategiaren izena eman: GAIZKI.DAT
Kopuru errealak GAIZKI.DAT fitxategitik pantailaratzen
Runtime error 002 at 0BF6:00FD.
_
```

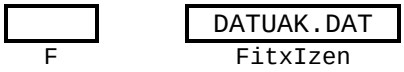
Non, fitxategi fisikoaren izena ematean `GAIZKI.DAT` karaktere-katea sartu den, izen horretako fitxategirik diskoan ez dagoelako `Reset` prozedurak ezin du fitxategirik ireki eta exekuzio-denboran `002` errorea[5] itzultzen duen.

12.1.1.1 Fitxategi fisikoa

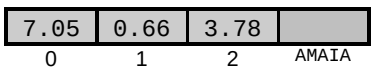
`FitxategiBatSortzen` eta `FitxategiBatIkusten` programetan hiru zenbaki erreal biltzen dituen `DATUAK.DAT` izeneko fitxategi bat sortu eta pantailaratu egin da, horretarako `F` fitxategi logikoa (`FILE` datu-motatakoa) eta `FitxIzen` izeneko string aldagaia behar izan dira:

```
VAR
   F : FILE OF Real ;        { Fitxategi logikoa }
   FitxIzen : String ;       { Fitxategi fisikoa }
```

Memorian

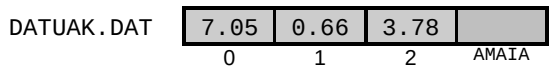


Diskoan



[5] `002` erroreari dagokion literala: `Error 2: File not found.`

Diskoan kokaturik dagoen egiturari *fitxategi fisiko* esango diogu, zenbaki errealak biltzen dituen `DATUAK.DAT` izeneko egiturari alegia. Fitxategi bitarretan biltzen den informazioak duen egituraketa memoria nagusian duena da (kapitulu honen lehendabiziko adibide-programa gogoratu). Beraz, aurreko fitxategi fisikoaren eskema hau litzateke:



Pascal programa batean fitxategi fisikoaren datu-mota string bat da, karaktere-kate horrek diskoan dagoen fitxategiaren izena edukiko du (`DATUAK.DAT` aurreko adibidean).

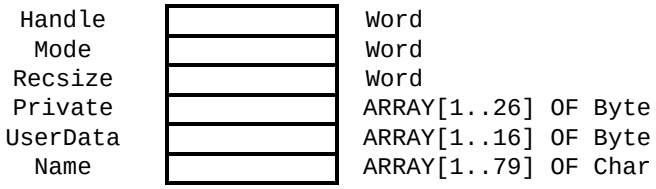
Baina, ordenadore batek prozesatzen duen informazioa memoria nagusian aurkitu behar delako diskoan kokaturik dagoen fitxategi fisikoa landu ahal izateko nolabaiteko amarrua egiten da. Ordenadorearen memoria eta kanpoko unitateetan dagoen informazioa elkartzeko *fitxategi logiko* delakoa erabiltzen da.

12.1.1.2 Fitxategi logikoa

Diskoan dagoen `DATUAK.DAT` izeneko fitxategi fisikoaren informazioa lantzeko `F` izena duen aldagaia erabiltzen da, `F` aldagaiaren datu-mota `FILE` bat da (hots, `FILE OF Real`) eta horri *fitxategi logiko* esango diogu.

`F` fitxategi logikoa izatez erregistro bat da, eta bere eremuetan gordetzen duena ez da `DATUAK.DAT` fitxategi fisikoaren edukia (`F` fitxategi logikoak ez ditu kopuru errealak gordetzen). Fitxategi logiko batek berari dagokion fitxategi fisikoari buruzko informazioa barneratzen du bere erregistro-eremuetan, esate baterako: fitxategi fisikoaren izena, atzipenaren modua, osagaien tamaina, e.a..

Jarraian fitxategi bitar batetarako memorian sortzen den fitxategi logikoaren eremuak erakusten dira (fitxategiekin lan egiteko ez da zertan eremu horien barne-antolaketa ezagutu behar):



`F` fitxategi logikoaren memori egitura

Pascal programa batean fitxategi logikoa definitzeko `FILE OF` hitz erreserbatuen bitartez egiten da. Aurreko adibidean:

```
TYPE
   DM_Fitxategi : FILE OF Real ;     { Fitxategi logikoaren datu-mota }
VAR
   F : DM_Fitxategi ;                { Fitxategi logikoaren aldagaia }
```

12.1.1.3 Fitxategi fisiko eta fitxategi logiko. Laburpena

Nahiz eta, une honetan, jarraian ematen den taula bere osotasunean ikasleak ulertu ez, interesgarria iruditu zaigu fitxategi fisikoaren eta fitxategi logikoaren ezaugarriak azpimarratzea,

bide batez euren arteko desberdintasunak agerian azalduko zaizkigu:

Fitxategi Fisiko	
Identifikadorea	Karaktere-katea
Parametroa	Fitxategi fisikoa erabiliko duten azpiprogrametara karaktere-katea igorriko da, fitxategi fisikoaren izena programa nagusian ezaguna den ala ez aintzat harturik, karaktere-katea aldagai-parametroa edo baliozko parametroa izango da
Biltegitze euskarria	Fitxategi fisikoa kanpoko unitate (disko edo zinta) batean kokatzen da
Egitura	Fitxategia egitura homogeneoa eta lineala da. Fitxategiaren osagaiak edozein datu-motatakoak izan daitezke (`FILE` datu-mota izan ezik) baina denek datu-mota bera mantenduko dute Fitxategiaren osagaien kopurua ez da definizioan mugatzen eta dinamikoki alda daiteke programaren exekuzioaren arabera
Elementuen aldaketak	Fitxategiaren osagaien bat aldatzeko bere edukia memoriara irakurriko da eta balioak eguneratu ostean berriro idatziko da fitxategian
Gainezkadak fitxategietan: “underflow” eta “overflow”	0 posizioaren baino txikiago den posizio batean irakurtzen saiatuz gero exekuzio errorea eragiten da 0 posizioaren baino txikiago den posizio batean idazten saiatuz gero exekuzio errorea eragiten da Azken posizioaren baino handiago den posizio batean irakurtzen saiatuz gero exekuzio errorea eragiten da Azken posizioaren baino handiago den posizio batean idazten saiatuz ez da errorerik sortzen (programadoreak kontrolatu beharko du)

Fitxategi Logiko	
Identifikadorea	`FILE` datu-motako aldagaia
Parametroa	Fitxategi logiko bat azpiprogrametara igortzen denean aldagai-parametroa izango da beti. Beraz `FILE` bat beti erreferentziaz
Biltegitze euskarria	Fitxategi logikoak memoriako aldagaiak dira
Egitura	Hainbat eremuko erregistroa

Eragiketak	`FILE` datu-motako aldagaiekin ezin da inolako eragiketarik egin (ezta asignazioa bera ere ez), `FILE` aldagaiak azpiprogramen parametroak izango dira. Honez gero, funtzio eta prozedura estandarrez lantzen dira fitxategi logikoaren eremuak. Aurredefinituriko azpiprogramak:
	Funtzioak
	`Eof (Fitxategi_logiko)`
	`FileSize (Fitxategi_logiko)`
	`FilePos (Fitxategi_logiko)`
	Prozedurak
	`Assign (Fitxategi_logiko, Fitxategi_fisiko)`
	`Rewrite (Fitxategi_logiko)`
	`Reset (Fitxategi_logiko)`
	`Close (Fitxategi_logiko)`
	`Read (Fitxategi_logiko, Osagai_aldagaia)`
	`Write (Fitxategi_logiko, Osagai_aldagaia)`
	`Seek (Fitxategi_logiko, Posizioa)`
	`Truncate (Fitxategi_logiko)`
	`Erase (Fitxategi_logiko)`
	`Rename (Fitxategi_logiko, Karaktere_katea)`
	`...`

12.1.2 Aurredefinituriko azpiprogramak

Esan den bezala `FILE` datu-motako aldagaiak azpiprogramen parametroak izango dira beti, eurekin ezin delako bestelako eragiketarik egin. Demagun jarraian ematen den definizioa daukagula:

```
TYPE
   DM_Fitxategi : FILE OF Real ;     { Fitxategi logikoaren datu-mota }
VAR
   F1, F2 : DM_Fitxategi ;           { Fitxategi logikoen aldagaiak }
```

eta programa nagusiko `F1` eta `F2` aldagaiak ondoko sententzietan agertzen direla:

```
IF F1 = F2 THEN        { Error 41: Operand types do no match operator. }
F2 := F1 ;             { Error 43: Illegal assignment. }
F1 = F1 + 1 ;          { Error 43: Illegal assignment. }
ReadLn (F1) ;          { Error 63: Invalid file type. }
WriteLn (F1) ;         { Error 63: Invalid file type. }
F1.Name := 'Datuak';   { Error 121: Invalid qualifier. }
```

Konpilazio-errorren mezuen arabera, nahiz eta fitxategi logikoen barne-egitura erregistro bat izan, ezin daiteke fitxategi logikoekin inolako eragiketak burutu. Sei adibide horietan dauden eragiketak oinarrizkoenak izanik ere konpiladoreak ez ditu onartzen, fitxategi logikoekin lan egiteko zeharkako bidea ibili behar da, hots, **12.1.2.1** eta **12.1.2.2** puntuetan azaltzen diren funtzio eta prozedurek eskaintzen duten bidea.

12.1.2.1 Funtzioak

Fitxategi bitarrei aplikatuko zaizkion funtzio estandarrak hiru dira Turbo Pascal goi-mailako lengoaian:

```
Eof (Fitxategi_logiko)
FileSize (Fitxategi_logiko)
FilePos (Fitxategi_logiko)
```

12.1.2.1.1 `Eof` funtzioa

Irakurleak gogoratuko duenez, `FitxategiBatIkusten` adibide-programan `Eof` funtzioa erabiltzen zen fitxategiaren osagai guztiak bistaratzeko:

```
Kontag := 0 ;
WHILE NOT Eof (F) DO      { Fitxategia amaitzen ez den bitartean }
BEGIN
   Read (F, Elem) ;       { Fitxategitik memoriara transferitu }
   WriteLn (Kontag, '. elementua = ', Elem:0:8) ;
   Kontag := Kontag + 1 ;
END ;
```

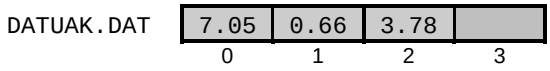
`WHILE` kontrol-egiturak duen baldintzaren bitartez fitxategiaren erakuslea amaieran aurkitzen den ala ez testeatzen da. Fitxategia bukatu ez denean `WHILE` barneko sententziak exekutatuko dira (fitxategitik osagai bat irakurri, pantaileraketa bat egin eta `Kontag` kontagailua inkrementatu). Fitxategia bukatzean `WHILE` blokearen hurrengo sententziara jauzi egingo du programaren fluxuak.

`Eof` funtzio estandarrak, funtzio boolearra delako, `TRUE` ala `FALSE` balioetatik bat itzultzen ditu. Hona hemen `Eof` funtzioaren ezaugarriak:

Sintaxia:	`Eof (Fitxategi_logiko)`
Parametroa:	`Fitxategi_logiko` `File` datu-mota, erreferentziaz (aldagai-parametroa)
Itzulitako datu-mota:	`Boolean`
Itzulitako balioa:	`FALSE` fitxategiaren erakusleak fitxategiaren osagaien bat erakusten ari bada. `TRUE` fitxategiaren erakusleak fitxategiaren osagairik erakusten ez badu

12.1.2.1.2 `FileSize` funtzioa

Fitxategi bitarren definizioan idatzi dugun `FitxategiBatSortzen` adibide-programan `DATUAK.DAT` izeneko fitxategia sortu eta datuz betetzen zen. `FitxategiBatSortzen` adibide-programaren exekuzioa gogoratuz honelako edukia zuela suposatu dugu, non hiru zenbaki erreal gorde izan diren `DATUAK.DAT` fitxategian:



`DATUAK.DAT` fitxategiaren azken osagairen ostean fitxategiaren amaiera marka marraztu dugu eskeman. Hiru zenbakien indizeek adierazten digutenez fitxategi baten lehen osagaiari zero posizioa dagokio eta gainerakoei 1 eta 2 hurrenez hurren, `DATUAK.DAT` fitxategiaren lehen posizio librea 3a delako amaiera marka hor aurkitzen da.

`FitxategiBatSortzen` programaren bitartez hiru elementuko `DATUAK.DAT` fitxategia lortu ondoren, bere tamaina ezagutzeko sistema eragileko `dir` komandoa aplika daiteke, edo bestela jarraian erakusten dugun `FitxategiBatNeurtzen` programa exekuta dezakegu:

```
PROGRAM FitxategiBatNeurtzen ;                  { \TP70\12\FILE_03.PAS }
TYPE
  DM_Fitxategi = FILE OF Real ;

VAR
   FitxIzen : String ;       { Fitxategi fisikoa }
   F : DM_Fitxategi ;        { Fitxategi logikoa }
   Elem : Real ;
   ByteKopurua : LongInt ;

BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;

   Assign (F, FitxIzen) ;    { Fitxategi logikoa eta fisikoa elkartu }
   Reset (F) ;               { Irakurketarako ireki }

   ByteKopurua := FileSize (F) * SizeOf (Elem) ;
   WriteLn (FitxIzen, ' fitxategiak ', ByteKopurua, ' byte ditu') ;

   Close (F) ;               { Fitxategia itxi }
END.
```

`DATUAK.DAT` fitxategiaren edukia goiko eskeman azaltzen dena izanik `FileSize` funtzioa darabilen `FitxategiBatNeurtzen` programak irteera hau izango luke:

```
Fitxategiaren izena eman: DATUAK.DAT
DATUAK.DAT fitxategiak 18 byte ditu
_
```

`FileSize(F)` deiak `DATUAK.DAT` fitxategiak zenbat osagai dituen itzultzen dio modulu deitzaileari (adibidean 3 dela suposatu egin da). `SizeOf(Elem)` deiak `Elem` aldagaiak zenbat byte hartzen dituen memorian itzultzen dio modulu deitzaileari (6 byte `Real` bat delako).

Hona hemen `FileSize` funtzioaren ezaugarriak:

Sintaxia:	`FileSize (Fitxategi_logiko)`
Parametroa:	`Fitxategi_logiko` `File` datu-mota, erreferentziaz (aldagai-parametroa)
Itzulitako datu-mota:	`LongInt`
Itzulitako balioa:	Fitxategiak duen elementuen kopurua. Beste era batean fitxategiaren lehen toki libreari dagokion posizioa, hau da, fitxategiaren amaiera markaren posizioa

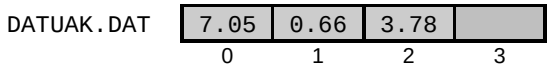
12.1.2.1.3 `FilePos` funtzioa

`DATUAK.DAT` fitxategiarekin jarraituz, bere edukia pantailaratzeko `FitxategiBatIkusten` adibide-programa idatzi dugu, zeinen `WHILE-DO` baten barruan fitxategiaren osagaiak irakurri egiten ziren (`WHILE-DO` kontrol-egituraren amaiera ziurtatzeko `Eof(F)` funtzioa erabiltzen zen). Demagun programan agertzen den `Kontag` aldagaia kentzen dugula eta bere zeregina betetzeko `FilePos` funtzioa aplikatuko dugula. Hona hemen `ElementuenPosizioak` programa:

```
PROGRAM ElementuenPosizioak ;                  { \TP70\12\FILE_04.PAS }
TYPE
  DM_Fitxategi = FILE OF Real ;

VAR
   FitxIzen : String ;       { Fitxategi fisikoa }
   F : DM_Fitxategi ;        { Fitxategi logikoa }
   Elem : Real ;
   Posizioa : LongInt ;

BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   WriteLn ('Kopuru errealak ', FitxIzen, ' fitxategitik pantailaratzen') ;

   Assign (F, FitxIzen) ;    { Fitxategi logikoa eta fisikoa elkartu }
   Reset (F) ;               { Irakurketarako ireki }

   WHILE NOT Eof (F) DO      { Fitxategia amaitzen ez den bitartean }
   BEGIN
      Posizioa := FilePos (F) ;
      Read (F, Elem) ;       { Fitxategitik memoriara transferitu }
      WriteLn (Posizioa, ' posizioan dagoen elementua = ', Elem:0:8) ;
   END ;

   Close (F) ;               { Fitxategia itxi }
END.
```

Aurrerago azalduko dugu baina kontura gaitezen `WHILE-DO` agindu errepikakorraren baldintza `Eof` funtzioz kontrolatzen dela, beraz `WHILE`-ren barnean `Eof` funtzioak aldaketa jasan beharko du. Zerk eragiten du `Eof` funtzioak `FALSE` itzuli ordez `TRUE` itzul dezan?.

Diskoan dagoen `DATUAK.DAT` fitxategiari dagokion eskema hau izanik:



Jarraian ematen da `ElementuenPosizioak` programa exekutatzean lortuko litzatekeen irteera

```
Fitxategiaren izena eman: DATUAK.DAT
Kopuru errealak DATUAK.DAT fitxategitik pantailaratzen
0 posizioan dagoen elementua = 7.05000000
1 posizioan dagoen elementua = 0.66000000
2 posizioan dagoen elementua = 3.78000000
_
```

Hona hemen `FilePos` funtzioaren ezaugarriak:

Sintaxia:	`FilePos (Fitxategi_logiko)`
Parametroa:	`Fitxategi_logiko` `File` datu-mota, erreferentziaz (aldagai-parametroa)
Itzulitako datu-mota:	`LongInt`
Itzulitako balioa:	Fitxategiaren erakusleari dagokion uneko posizioa

12.1.2.2 Prozedurak

Fitxategi bitarrekin lan egiteko ikasiko ditugun prozedura estandarrak hamar dira:

```
Assign (Fitxategi_logiko, Fitxategi_fisiko)
Rewrite (Fitxategi_logiko)
Reset (Fitxategi_logiko)
Close (Fitxategi_logiko)
Read (Fitxategi_logiko, Osagai_aldagaia)
Write (Fitxategi_logiko, Osagai_aldagaia)
Seek (Fitxategi_logiko, Posizioa)
Truncate (Fitxategi_logiko)
Erase (Fitxategi_logiko)
Rename (Fitxategi_logiko, Karaktere_katea)
```

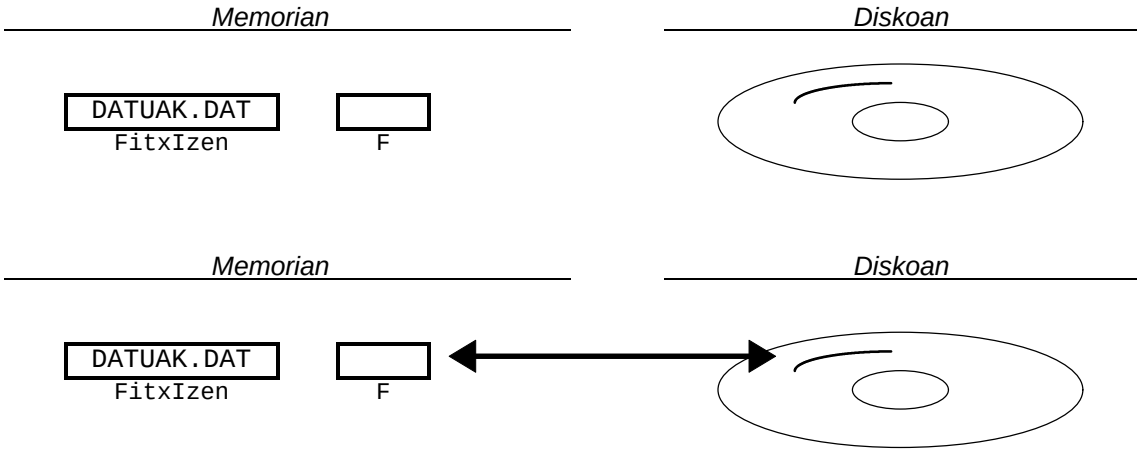
12.1.2.2.1 `Assign` prozedura

Fitxategi fisiko eta fitxategi logiko kontzeptuak lehenago azaldu ditugu. Diskoan kokaturik dagoen egiturari *fitxategi fisiko* esan diogu eta hura lantzeko memorian dagoen *fitxategi logiko* deituriko aldagaia behar da. Ordenadorearen memoria eta kanpoko unitateetan dagoen informazioa elkartzeko `Assign` prozedura estandarra erabiltzen da Pascal lengoaian:

```
TYPE
  DM_Fitxategi = FILE OF Real ;
VAR
   FitxIzen : String ;            { Fitxategi fisikoa }
   F1, F2 : DM_Fitxategi ;        { Fitxategi logikoa }
BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   Assign (F1, FitxIzen) ;        { Fitxategi logikoa eta fisikoa elkartu }
```

Goiko lerroetan agertzen den `Assign` prozeduraren bitartez `F1` eta `FitxIzen` aldagaien lotura lortzen da. Kontutan izan dezagun `F1` aldagaia `FILE` datu-motatakoa dela eta `FitxIzen`

karaktere-kateak balio zehatzen bat behar duela `Assign` prozeduran sarrerako parametroa baita. Ikus `Assign` prozedurak eragiten duen elkarketaren irudiak, bat `Assign` aplikatu aurretik eta bestea `Assign` ostean:



`Assign` prozeduraren ezaugarriak:

Sintaxia:	`Assign (Fitxategi_logiko, Fitxategi_fisiko)`

Parametroa:	`Fitxategi_logiko`
	`File` datu-mota, erreferentziaz (aldagai-parametroa)
	`Fitxategi_fisiko`
	`String` datu-mota, baliozko parametroa

Eginkizuna:	Prozedura hau deitu ondoren fitxategi logiko bat zehazturiko fitxategi fisiko batekin elkartzen da `Close` prozedura aurkitu arte

12.1.2.2.2 `Rewrite` prozedura

Fitxategi fisiko bat dagokion fitxategi logikorekin `Assign` bitartez elkartu ondoren, fitxategia irekiko dugu bertan idazketak edo/eta irakurketak egiteko[6]. Fitxategi bitarrak irekitzeko bi prozedura eskaintzen ditu Turbo Pascal lengoaiak: `Rewrite` eta `Reset`.

`Rewrite` prozedurak behar duen parametro bakarra fitxategi logikoa da eta `Rewrite` prozedura deitu baino lehen fitxategi logiko hori fitxategi fisiko batekin loturik egongo da. Beraz, `Rewrite` bitartez fitxategi fisiko bat irekitzeko `Assign` egokia aplikatu izan da lehendik.

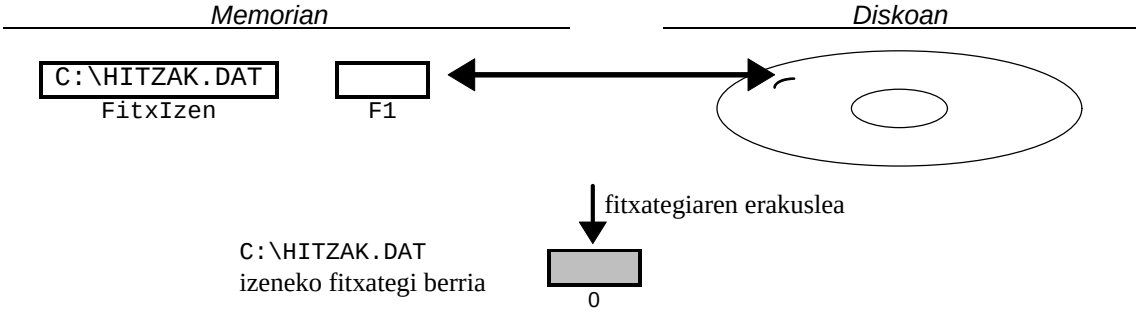
```
TYPE
  DM_Fitxategi = FILE OF String ;
VAR
   FitxIzen : String ;            { Fitxategi fisikoa }
   F1, F2 : DM_Fitxategi ;        { Fitxategi logikoa }
BEGIN
   FitxIzen := 'C:\HITZAK.DAT' ;
   Assign (F1, FitxIzen) ;        { Fitxategi logikoa eta fisikoa elkartu }
   Rewrite (F1) ;                 { Fitxategia idazketarako ireki }
```

Goiko lerroetan agertzen den `Assign` prozeduraren bitartez `F1` eta `FitxIzen` aldagaien

[6] Izan ere, fitxategi batek onartzen dituen operazioak bi dira:
1. Fitxategian informazioa idatzi (datuak erregistratu, gorde edo biltegitu)
2. Fitxategitik informazioa irakurri (datuak jaso, hartu edo eskuratu)

lotura lortzen da, ikusi nola `FitxIzen` karaktere-kateak balio zehatzen bat behar duela `Assign` prozeduran sarrerako parametroa delako (lerro horietan teklatuz balioa irakurri beharrean string konstante bat esleitzen zaio `FitxIzen` aldagaiari). Bigarren sententzian, `Rewrite` prozedurari deia egiten zaionean, `F1` aldagaiak asignaturik duen fitxategi fisikoa[7] sortu egiten da.

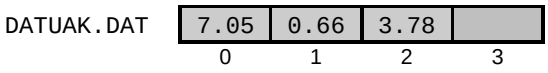
Lerro horien arabera C unitateko `HITZAK.DAT` fitxategia sortuta geratuko litzateke, hutsik legoke oraindik baina C unitatean fitxategi berria izango genuke, eskema honi erantzuten dion fitxategia alegia:



`Rewrite` bitartez fitxategi fisiko berri bat sortzen da. Izen bera duen fitxategi fisikoa lehendik existitzen bada, `Rewrite`-k lehenengo ezabatu egiten du eta ondoren sortu; hau arriskutsua izan daiteke fitxategi bat ezabatzean duen informazioa galtzen delako. Adibide batekin ikus dezagun balizko arrisku hau zertan den, demagun `FitxategiBatSortzen` izeneko programaren bitartez hiru zenbaki erreal diskoan biltegitzen duen `DATUAK.DAT` fitxategia daukagula lehendik sortuta, `FitxategiBatSortzen` programaren exekuzioa hau izanik:

```
Fitxategiaren izena eman: DATUAK.DAT
Kopuru errealak DATUAK.DAT fitxategian gordetzen
Eman fitxategiaren 0. elementua: 7.05
Gehiagorik? (B/E)  B
Eman fitxategiaren 1. elementua: 0.66
Gehiagorik? (B/E)  B
Eman fitxategiaren 2. elementua: 3.78
Gehiagorik? (B/E)  E
_
```

`DATUAK.DAT` fitxategiari dagokion eskema hauxe litzateke:



`FitxategiBatSortzen` programan fitxategia `Rewrite` baten bitartez irekitzen delako `DATUAK.DAT` fitxategia berriro sortzen da (lehendik dagoen informazioa galduz) eta ondorioz ezin zaio `DATUAK.DAT` fitxategiari balio berri bat gehitu. `FitxategiBatSortzen` programa bigarren aldiz honela exekutatzean, zein uste duzu izango dela `DATUAK.DAT` fitxategiari dagokion irudia?

```
Fitxategiaren izena eman: DATUAK.DAT
Kopuru errealak DATUAK.DAT fitxategian gordetzen
Eman fitxategiaren 0. elementua: 1.69
Gehiagorik? (B/E)  E
_
```

[7] C unitateko erroaren azpian dagoen `HITZAK.DAT` izeneko fitxategia.

Hona hemen `Rewrite` prozeduraren ezaugarriak:

Sintaxia:	`Rewrite (Fitxategi_logiko)`
Parametroa:	`Fitxategi_logiko` `File` datu-mota, erreferentziaz (aldagai-parametroa) baina lehendik `Assign` egokiaren bitartez dagokion fitxategi fisikoa zehazturik egongo da
Eginkizuna:	Prozedura hau deitu ondoren fitxategi logikoari dagokion fitxategi fisiko berria sortzen da diskoan. Fitxategia hutsik sortzen da eta fitxategiaren erakuslea 0 posizioan aurkituko da

12.1.2.2.3 `Reset` prozedura

Esan dugun bezala `Assign` prozedura deitu ondoren, fitxategi bitarrak irekitzeko bi prozedura eskaintzen ditu Turbo Pascal lengoaiak: `Rewrite` eta `Reset`. Ezagutzen dugun `Rewrite` bitartez fitxategi bat irekitzean gerta daitekeen arazoa informazioa galtzea litzateke, bestalde, fitxategia `Reset` bidez irekitzean ez da fitxategi fisikoaren edukia ezabatzen. Hori dela eta, `FitxategiBatIkusten` izeneko programan, zenbaki errealak gordetzen dituen `DATUAK.DAT` fitxategiaren edukia pantailaratzeko `Reset` egiten zen:

```
PROGRAM FitxategiBatIkusten ;                  { \TP70\12\FILE_02.PAS }
TYPE
  DM_Fitxategi = FILE OF Real ;

VAR
   FitxIzen : String ;       { Fitxategi fisikoa }
   F : DM_Fitxategi ;        { Fitxategi logikoa }
   Elem : Real ;
   Kontag : LongInt ;

BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   WriteLn ('Kopuru errealak ', FitxIzen, ' fitxategitik pantailaratzen') ;

   Assign (F, FitxIzen) ;    { Fitxategi logikoa eta fisikoa elkartu }
   Reset (F) ;               { Irakurketarako ireki }
```

Baina `Reset` prozedurak badu bere ahultasuna, hots, aldez aurretik sorturik ez dagoen fitxategia ezin du ireki eta exekuzio denboran errorea eragiten du. Esate baterako, existitzen ez den `KOPURUAK.DAT` fitxategiaren edukia `FitxategiBatIkusten` programaz pantailaratzea nahi badugu hauxe gertatuko litzateke:

```
Fitxategiaren izena eman: KOPURUAK.DAT
Kopuru errealak KOPURUAK.DAT fitxategitik pantailaratzen
Runtime error 002 at 0BF5:00F8
_
```

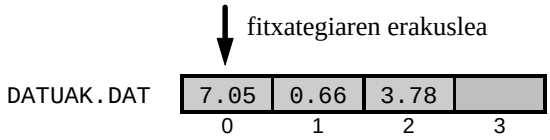
12.1.1 puntuan esandakoa errepikatuz, fitxategi fisikoaren izena ematean `KOPURUAK.DAT` karaktere-katea sartu da, izen horretako fitxategirik diskoan ez dagoelako `Reset` prozedurak ezin du ireki eta exekuzio-denboran `002` errorea[8] itzultzen du.

[8] `002` erroreari dagokion literala: `Error 2: File not found.`

Hona hemen `Reset` prozeduraren ezaugarriak:

Sintaxia: `Reset (Fitxategi_logiko)`
Parametroa: `Fitxategi_logiko`
`File` datu-mota, erreferentziaz (aldagai-parametroa) baina lehendik `Assign` egokiaren bitartez dagokion fitxategi fisikoa zehazturik egongo da
Eginkizuna: Prozedura hau deitu ondoren fitxategi logikoari dagokion fitxategi fisikoa irekitzen da eta fitxategiaren erakuslea 0 posizioan aurkituko da. Ireki nahi den fitxategi fisikoa lehendik existitzen ez bada exekuzio-denboran errorea eragiten da

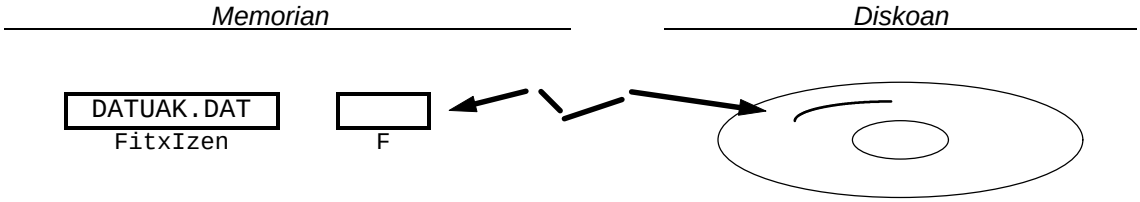
`Reset` prozedura arrakastaz exekutatzean daukagun eskema:



12.1.2.2.4 `Close` prozedura

Fitxategi fisiko bat dagokion fitxategi logikorekin `Assign` bitartez elkartu ondoren, `Reset` ala `Rewrite` prozeduretatik bati esker fitxategi fisikoa irekiko da. Fitxategia irekita dagoela programek datuak gorde edo/eta hartu ditzake fitxategitik, fitxategiaren prozesaketa bukatzean fitxategia itxi egingo dugu `Close` errutina deituz, fitxategia itxi aurretik `Close` prozedurak fitxategiaren eguneraketa behartzen du (azken aldaketak diskora kopiatu).

Hauek lirateke `Close` prozeduraren irudia eta ezaugarriak:



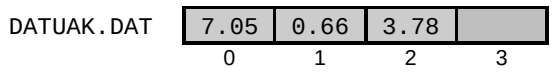
Sintaxia: `Close (Fitxategi_logiko)`
Parametroa: `Fitxategi_logiko`
`File` datu-mota, erreferentziaz (aldagai-parametroa) baina lehendik `Assign` egokiaren bitartez dagokion fitxategi fisikoa zehazturik egongo da
Eginkizuna: Prozedura hau deitu ondoren fitxategi logikoari dagokion fitxategi fisikoan azken aldaketak burutzen dira eta fitxategia itxi egiten da

12.1.2.2.5 `Read` prozedura

Fitxategi fisikoak prozesatzeko `Rewrite` eta `Reset` prozeduren bidez ireki daitezkeela esan dugu, bai ere fitxategiren prozesaketa bukatzean `Close` bitartez fitxategia itxi egin behar dela. Aipatu dugu ere, irekita dagoen fitxategi batek onartzen dituen operazio bakarrak bi direla

idazketak eta irakurketak, galdera honetan fitxategitik datuak irakurtzeari buruz arituko gara eta horretarako zenbaki errealak pantailaratzen dituen `FitxategiBatIkusten` izeneko programa gogoratuko dugu berriro.

Demagun `FitxategiBatSortzen` programa exekutatuz `DATUAK.DAT` fitxategi hau lortu ondoren `FitxategiBatIkusten` programa egikaritzen dugula:




```
PROGRAM FitxategiBatIkusten ;                  { \TP70\12\FILE_02.PAS }
TYPE
  DM_Fitxategi = FILE OF Real ;

VAR
   FitxIzen : String ;       { Fitxategi fisikoa }
   F : DM_Fitxategi ;        { Fitxategi logikoa }
   Elem : Real ;
   Kontag : LongInt ;

BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   WriteLn ('Kopuru errealak ', FitxIzen, ' fitxategitik pantailaratzen') ;

   Assign (F, FitxIzen) ;    { 1. urratsa }
   Reset (F) ;               { 2. urratsa }

   Kontag := 0 ;
   WHILE NOT Eof (F) DO      { 3. urratsa }
   BEGIN
      Read (F, Elem) ;       { 4. 5. eta 6. urratsak }

      WriteLn (Kontag, '. elementua = ', Elem:0:8) ;
      Kontag := Kontag + 1 ;
   END ;

   Close (F) ;               { 7. urratsa }
END.
```

Urratsez urrats `FitxategiBatIkusten` programaren exekuzioa aztertuz honelako irudiak marraztuko genituzke:

1. urratsa

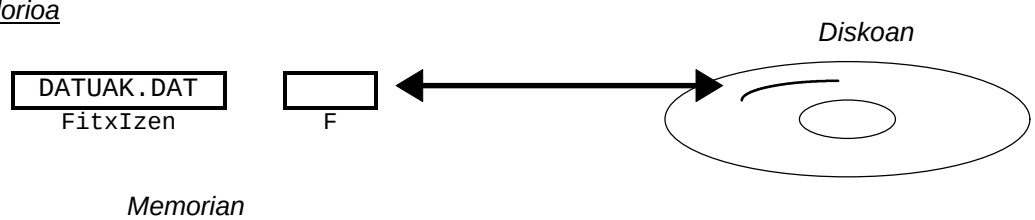
Aurrebaldintza

Fitxategi fisikoaren `DATUAK.DAT` izena teklatuz eman du operadoreak eta `FitxIzen` aldagaian gorde da.

Sententzia

```
Assign (F, FitxIzen) ;
```

Ondorioa



2. urratsa

Aurrebaldintza

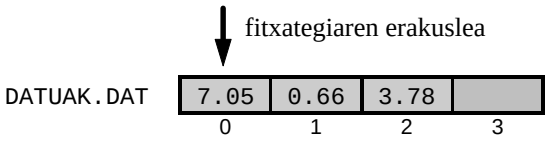
Fitxategi logikoari fitxategi fisiko bat elkartu zaio `Assign` prozedurari esker, eta fitxategi fisikoa diskoan existitzen da.

Sententzia

```
Reset (F) ;
```

Ondorioa

`DATUAK.DAT` izeneko fitxategi fisikoa irekitzen da eta erakuslea 0 posizioan kokaturik geratzen da. Honez gero `DATUAK.DAT` fitxategian sarrerak eta irteerak egin ahal izango dira.



3. urratsa

Aurrebaldintza

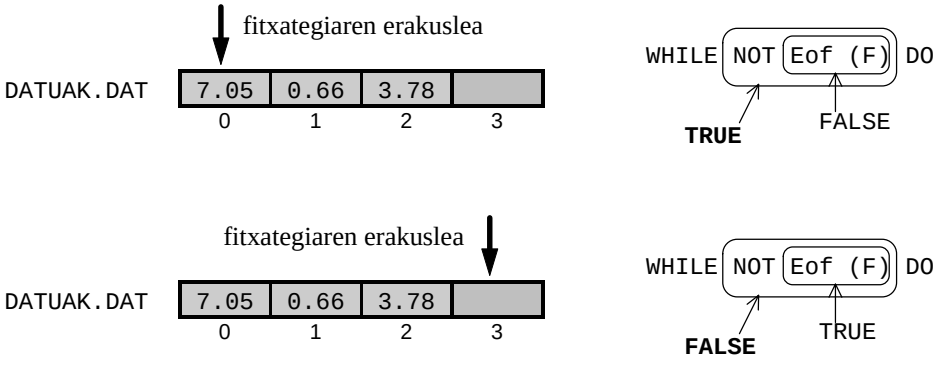
Fitxategi logikoari fitxategi fisiko bat elkartu zaio `Assign` prozedurari esker.

Sententzia

```
WHILE NOT Eof (F) DO
```

Ondorioa

`DATUAK.DAT` izeneko fitxategiren erakuslea datuen aldean dagoen bitartean (0, 1 eta 2 posizioetan dagoenean) `Eof` funtzioak `FALSE` itzuliko du eta `NOT` eragilearekin batera `WHILE-DO` aginduaren baldintzak `TRUE` balioko du, beraz programaren fluxua `WHILE-DO` aginduaren barneko sententzietatik igaroko da. Fitxategiaren erakuslea fitxategiaren amaiera marka gainean dagoenean `Eof` funtzioak `TRUE` itzuliko du eta `WHILE-DO` bigizta eten egingo da.



4. urratsa

Aurrebaldintza

Fitxategi logikoari fitxategi fisiko bat elkartu zaio `Assign` prozedurari esker, eta `WHILE-DO` barnean gaudelako fitxategiaren erakuslea datuen aldean aurkituko da (0, 1 edo 2 posizioetan). Laugarren urratsean erakuslea 0 posizioan dago `Reset` prozedura arestian deitu delako.

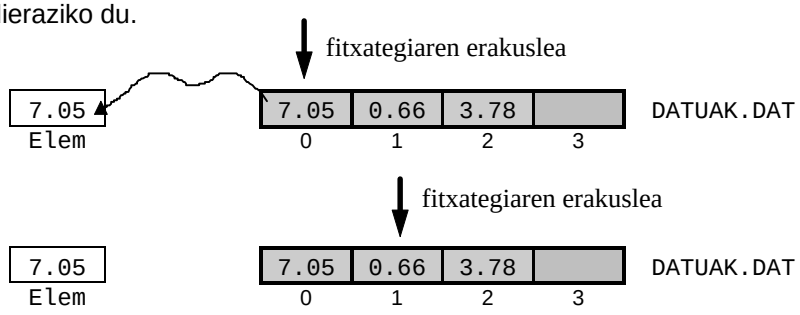
Sententzia

```
Read (F, Elem) ;
```

Ondorioa

Laugarren urratsean fitxategiaren erakusleak adierazten duen elementua (aurreneko zenbaki erreala den `7.05`) fitxategitik memoriara transferitzen da. Fitxategitik irakurriz memoriako `Elem` aldagaian `7.05` balioa gordetzen da.

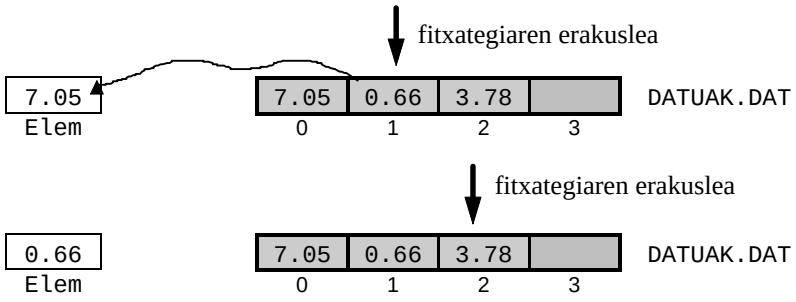
Fitxategiaren lehendabiziko elementuaren edukia memoriara igaro ondoren, fitxategiaren erakuslea desplazatzen da posizio bat inkrementatuz. `Read` baino lehen erakuslearen posizioa 0 zenez, `Read` egin eta gero erakusleak 1 posizioa adieraziko du.



5. urratsa

Bosgarren urratsa laugarren bezalakoa da, baina iterazio horri dagozkion aldaketak kontutan izan behar dira. `Elem` aldagaian `0.66` balioa gordetzen da eta erakuslea 1 posiziotik 2 posiziora higitzen da.

Ondorioa



6. urratsa

Aurrebaldintza

Fitxategi logikoari fitxategi fisiko bat elkartu zaio `Assign` prozedurari esker, eta `WHILE-DO` barnean gaudelako fitxategiaren erakuslea datuen aldean aurkituko da. Seigarren urratsa hastean fitxategiaren erakusleak 2. posizioa (hirugarren zenbaki erreala) adierazten du.

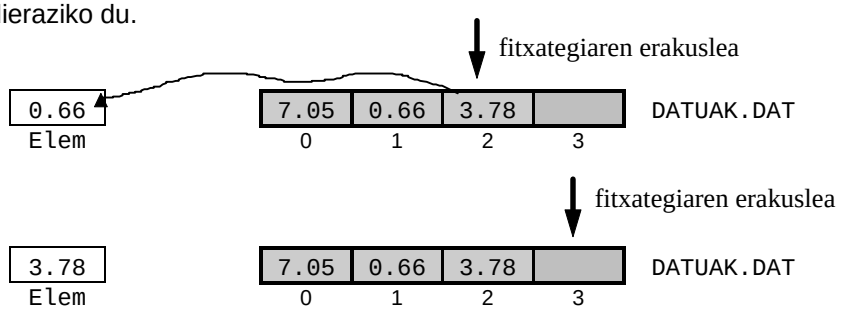
Sententzia

```
Read (F, Elem) ;
```

Ondorioa

Fitxategiaren erakusleak adierazten duen elementua (fitxategiko azken datua den `3.78` zenbaki erreala) fitxategitik memoriara transferitzen da. Fitxategitik irakurriz memoriako `Elem` aldagaiak `3.78` balio hori gordetzen du.

Fitxategiaren lehendabiziko elementuaren edukia memoriara igaro ondoren, fitxategiaren erakuslea desplazatzen da posizio bat inkrementatuz. `Read` baino lehen erakuslearen posizioa 2 zenez, `Read` egin eta gero erakusleak 3 posizioa adieraziko du.



Fitxategiaren erakuslea datuen zonaldetik at dagoenez `Eof` funtzioak `TRUE` itzuliko du eta ondorioz berriro `WHILE-DO` sententziaren baldintza ebaluatzean `FALSE` lortuko da eta bigizta bukatu egingo da.

7. urratsa

Aurrebaldintza

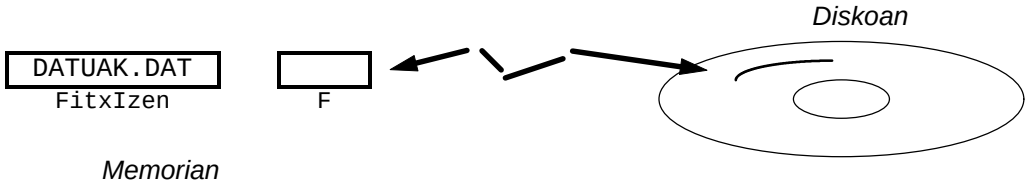
Fitxategi fisikoa irekita dago oraindik eta `WHILE-DO` agindu errepikakorra eten egin da.

Sententzia

```
Close (F) ;
```

Ondorioa

`DATUAK.DAT` izeneko fitxategi fisikoan azken datu-transferentziak burutu eta fitxategia ixten da.



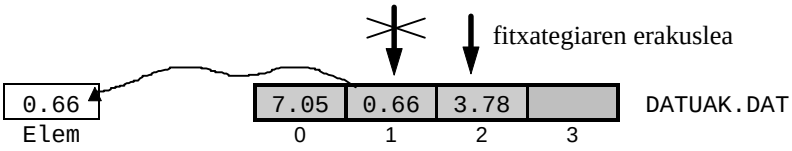
Hauek lirateke `Read` prozeduraren ezaugarriak eta irudia:

Sintaxia: `Read (Fitxategi_logiko, Osagai_aldagaia)`

Parametroa: `Fitxategi_logiko`
`File` datu-mota, erreferentziaz (aldagai-parametroa) baina lehendik `Assign` egokiaren bitartez dagokion fitxategi fisikoa zehazturik egongo da

`Osagai_aldagaia`
Fitxategiaren elementuen datu-motako aldagaia, erreferentziaz pasatutakoa (hots, aldagai-parametroa)

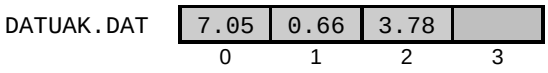
Eginkizuna: Prozedura hau deitu ondoren memorian dagoen aldagai batek ordenadore kanpoko fitxategi batetik hartutako balioa gordeko du. Fitxategiaren zein datu irakurri erabakitzeko erakuslearen uneko posizioa aintzat hartzen da, irakurketa burutu ondoren erakuslea automatikoki aurrerantz higitzen da



12.1.2.2.6 `Write` prozedura

Azaldu dugun `Read` prozeduraz ordenadoretik kanpo dagoen informazio bat eskura daiteke, fitxategitik datua irakurri eta memorian kokatu alegia. `Read` prozedurak komunikazioaren zentzu bat ahalbidetzen badu, `Write` prozedurak komunikazioaren beste norabidea (ordenadorearen memorian dagoena fitxategian gordetzea) alhalbidetzen digu.

Demagun `FitxategiBatSortzen` programa exekutatuz `DATUAK.DAT` fitxategi hau lortu nahi dugula:




```
PROGRAM FitxategiBatSortzen ;                  { \TP70\12\FILE_01.PAS }
USES
  Crt ;

TYPE
  DM_Fitxategi = FILE OF Real ;

VAR
   FitxIzen : String ;       { Fitxategi fisikoa }
   F : DM_Fitxategi ;        { Fitxategi logikoa }
   Elem : Real ;
   Kontag : LongInt ;
   Erantz : Char ;

BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   WriteLn ('Kopuru errealak ', FitxIzen, ' fitxategian gordetzen') ;

   Assign (F, FitxIzen) ;    { 1. urratsa }
   Rewrite (F) ;             { 2. urratsa }
```

```
   Kontag := 0 ;
   REPEAT
      Write ('Eman fitxategiaren ', Kontag, '. elementua: ') ;
      ReadLn (Elem) ;        { Zenbakia memorian dago }

      Write (F, Elem) ;      { 3. 4. eta 5. urratsak }

      Kontag := Kontag + 1 ;
      Write ('Gehiagorik? (B/E)  ') ;
      Erantz := ReadKey ;
      WriteLn (UpCase (Erantz)) ;
   UNTIL Erantz = 'E' ;

   Close (F) ;               { 6. urratsa }
END.
```

Urratsez urrats `FitxategiBatSortzen` programaren exekuzioa aztertuz honelako irudiak marraztuko genituzke:

1. urratsa

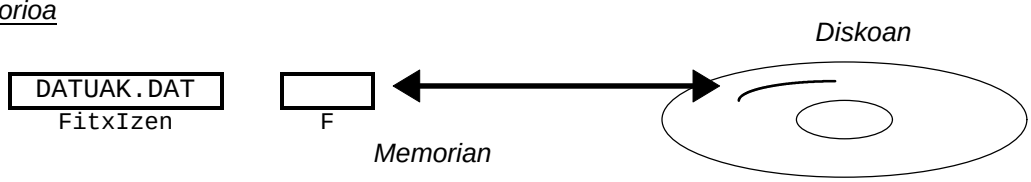
Aurrebaldintza

Fitxategi fisikoaren `DATUAK.DAT` izena teklatuz eman du operadoreak eta `FitxIzen` aldagaian gorde da.

Sententzia

```
Assign (F, FitxIzen) ;
```

Ondorioa



2. urratsa

Aurrebaldintza

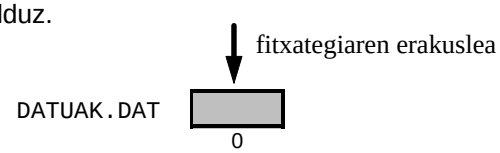
Fitxategi logikoari fitxategi fisiko bat elkartu zaio `Assign` prozedurari esker. Asignatutako fitxategi fisikoa ez da zertan diskoan existitu behar.

Sententzia

```
Rewrite (F) ;
```

Ondorioa

`DATUAK.DAT` izeneko fitxategi fisikoa sortu egiten da eta erakuslea 0 posizioan kokaturik geratzen da. Fitxategia berriki sortzen du `Rewrite` prozedurak eta `DATUAK.DAT` fitxategia lehendik existitzen bazen bere datuak ezabatuko dira zegoen informazioa galduz.



3. urratsa

Aurrebaldintza

Fitxategi logikoari fitxategi fisiko bat elkartu zaio `Assign` prozedurari esker eta memoriako `Elem` aldagaian operadoreak balioren bat gorde izan du. Hirugarren urratsean erakuslea 0 posizioan dago `Rewrite` prozedura arestian deitu delako.

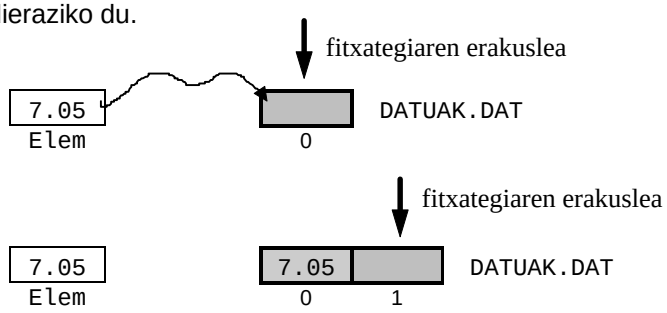
Sententzia

```
Write (F, Elem) ;
```

Ondorioa

Hirugarren urratsean fitxategiaren erakusleak adierazten duen posizioa 0 da (besterik ez baitago), `Elem` aldagaian erabiltzaileak gorde duen `7.05` balioa memoriatik fitxategira transferitzen da. Fitxategiaren erakuslea amaieran dagoelako idazketan elementu berria fitxategiari gehitu egingo zaio, erakuslea datuen aldean balego idazketa prozesuan elementu berriak zaharra ordezkatuko luke.

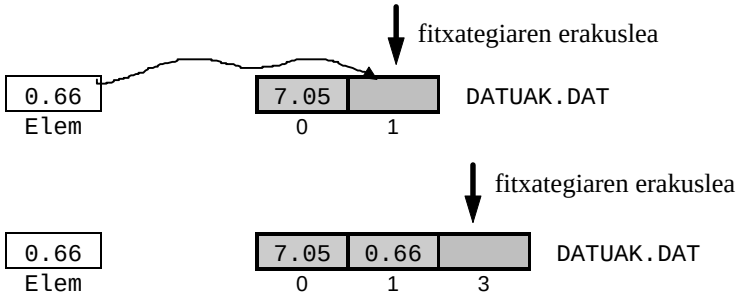
Fitxategiaren lehendabiziko elementuaren edukia memoriatik igaro ondoren, fitxategiaren erakuslea desplazatzen da posizio bat inkrementatuz. `Write` baino lehen erakuslearen posizioa 0 zenez, `Write` egin eta gero erakusleak 1 posizioa adieraziko du.



4. urratsa

Laugarren urratsa hirugarren bezalakoa da, baina iterazio horri dagozkion aldaketak kontutan izan behar dira. `Elem` aldagaian `0.66` balioa gordeta zegoen eta erakuslea 1 posiziotik 2 posiziora higitzen da.

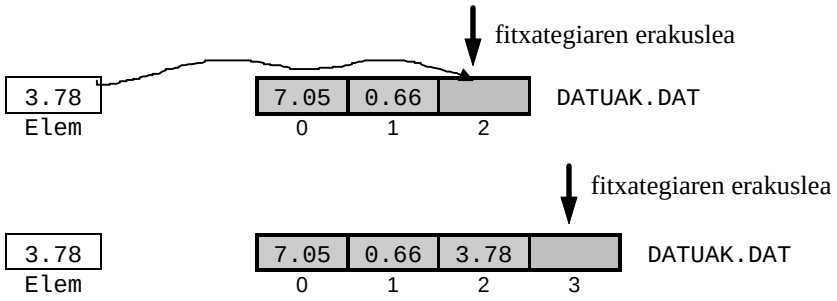
Ondorioa



5. urratsa

Bosgarren urratsa hirugarren eta laugarren bezalakoa da, kasu honetan `Elem` aldagaian operadoreak teklatuz `3.78` balioa gordetzen du eta erakuslea 2 posiziotik 3 posiziora higitzen da.

Ondorioa



6. urratsa

Aurrebaldintza

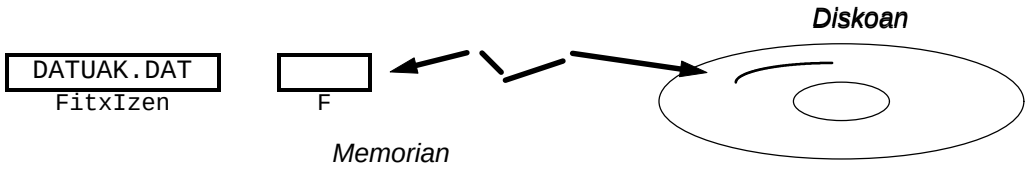
Fitxategi fisikoa irekita dago oraindik eta `REPEAT-UNTIL` agindu errepikakorra eten egin da programaren erabiltzaileak horrela erabakita.

Sententzia

```
Close (F) ;
```

Ondorioa

`DATUAK.DAT` izeneko fitxategi fisikoan azken datu-transferentziak burutu eta fitxategia ixten da.



Hauek lirateke `Write` prozeduraren ezaugarriak eta irudia:

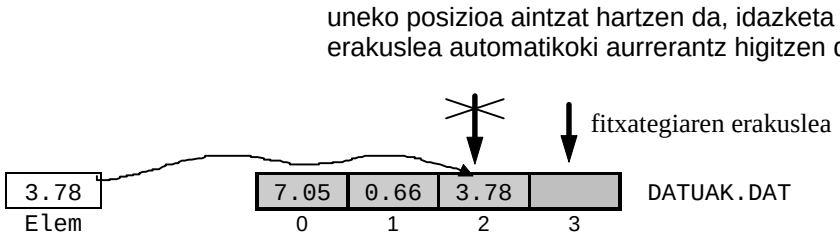
Sintaxia: `Write (Fitxategi_logiko, Osagai_aldagaia)`

Parametroa:

`Fitxategi_logiko`
: `File` datu-mota, erreferentziaz (aldagai-parametroa) baina lehendik `Assign` egokiaren bitartez dagokion fitxategi fisikoa zehazturik egongo da

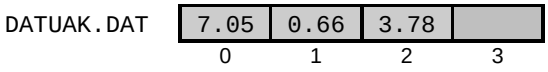
`Osagai_aldagaia`
: Fitxategiaren elementuen datu-motako aldagaia, balioz pasatutakoa (hots, baliozko parametroa)

Eginkizuna: Prozedura hau deitu ondoren memorian dagoen aldagai batek duen balioa ordenadore kanpoko fitxategi batean biltegituko da. Fitxategiaren zein tokitan idaztea erabakitzeko erakuslearen



12.1.2.2.7 Seek prozedura

Demagun `FitxategiBatSortzen` programa exekutatuz `DATUAK.DAT` fitxategi hau lortu dugula eta bere balioak pantailaratu nahi ditugula:



Irakurleak gogoratuko duenez zeregin hori gauzatzeko `FitxategiBatIkusten` adibide-programa erabil daiteke. Baina suposa dezagun programatu behar dugun pantailaraketan `DATUAK.DAT` fitxategiaren elementuak atzekoz aurrera agertzea eskatzen digutela, gure programak honelako irteera izan dezala:

```
Fitxategiaren izena eman: DATUAK.DAT
Kopuru errealak DATUAK.DAT fitxategitik atzekoz aurrera erakusten
2. elementua = 3.78000000
1. elementua = 0.66000000
0. elementua = 7.05000000
_
```

Hona hemen `FitxategiaAtzekozAurrera` programaren kodifikazioa, non agertzen den berrikuntza bakarra `Seek` prozeduraren dei parea den. `Seek` bitartez fitxategiaren erakuslea posizio jakin batean kokatzen da, horretarako bi parametro beharko dira: fitxategi logikoa eta helburuko posizioa zehazten duen zenbaki osoa:

```
PROGRAM FitxategiaAtzekozAurrera ;              { \TP70\12\FILE_05.PAS }
TYPE
  DM_Fitxategi = FILE OF Real ;

VAR
   FitxIzen : String ;                          { Fitxategi fisikoa }
   F : DM_Fitxategi ;                           { Fitxategi logikoa }
   Elem : Real ;
   Posizioa, AzkenDatuaNon, UnekoPosizioa, Kont : LongInt ;

BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   WriteLn ('Kopuru errealak ', FitxIzen, ' fitxategitik atzekoz aurrera erakusten') ;

   Assign (F, FitxIzen) ;
   Reset (F) ;                                  { Irakurketarako ireki }

   AzkenDatuaNon := FileSize (F) - 1 ;
   Seek (F, AzkenDatuaNon) ;                    { Erakuslea azken datuan kokatu }
```

```
   FOR Kont := AzkenDatuaNon DOWNTO 0 DO       { Fitxategiaren datuen aldea mugatu }
   BEGIN
      Read (F, Elem) ;                         { Fitxategitik memoriara transferitu }

      WriteLn (Kont, '. elementua = ', Elem:0:8) ;

      UnekoPosizioa := FilePos (F) ;           { Erakuslearen posizioa zehaztu }
      Seek (F, UnekoPosizioa - 2) ;            { Erakusle bi posizioz atzeratu }
   END ;

   Close (F) ;
END.
```

Fitxategien prozesaketetan gomendiogarriena izan arren `FitxategiaAtzekozAurrera` programan ezin da fitxategiaren bukaera `Eof` funtzioaren bitartez kontrolatu, horren ordez `Kont` aldagaiaz gidaturiko `FOR-DO` sententzia jarri dugu.

Fitxategia atzekoz aurrera tratatu beharrak eragiten duen arazoa fitxategiaren irekitze prozesuarekin loturik dago, `Reset` prozedurarekin loturik dago. Izan ere, `Reset` batek lehendik existitzen den fitxategi bat ireki ondoren erakuslea 0 posizioan kokatzen du, eta proposatutako ariketan azken datuaren gainean egotea nahiago genuke. Horixe da hain zuzen ere `Seek` prozedura estandarren helburua, erakuslea programadoreak dituen beharren arabera higitu ahal izatea.

Hauek lirateke `Seek` prozeduraren ezaugarriak:

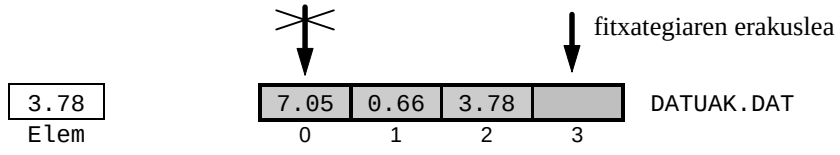
Sintaxia:	`Seek (Fitxategi_logiko, Posizioa)`

Parametroa:	`Fitxategi_logiko`
	`File` datu-mota, erreferentziaz (aldagai-parametroa) baina lehendik `Assign` egokiaren bitartez dagokion fitxategi fisikoa zehazturik egongo da
	`Posizioa`
	`LongInt` datu-mota, erakuslearen posizio berria, balioz pasatutakoa (hots, baliozko parametroa)

Eginkizuna:	Prozedura hau deitu ondoren fitxategiaren erakuslea tokiz aldatzen da, fitxategiaren erakusleari dagokion posizio berria prozeduraren bigarren parametroak zehazten du

Esan bezala `Seek` batek fitxategiaren identifikadoreaz gain erakuslearen posizio berria adierazten duen kopuru oso bat behar du. Esate baterako demagun `DATUAK.DAT` fitxategia `Reset` batez ireki ondoren, erakuslea azken posizio librean pausatu nahi dela, honela egingo genuke:

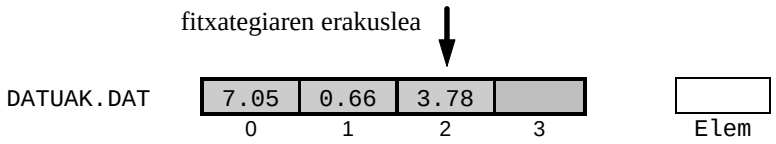
```
   Assign (F, 'DATUAK.DAT') ;          { hiru osagai ditu }
   Reset (F) ;                         { erakuslea 0 posizioan }
   Posizioa := FileSize (F) ;          { Posizioa aldagaian 3 gorde }
   Seek (F, Posizioa) ;                { Erakuslea 3 posiziora eraman }
```



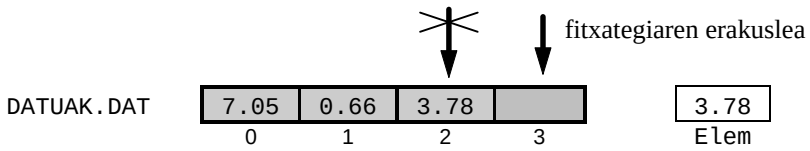
`FitxategiaAtzekozAurrera` programara itzuliz, `FileSize(F)` funtzioak fitxategiaren lehen posizio librea itzultzen badigu, `FileSize(F)-1` adierazpen aritmetikoak fitxategiaren azken datuaren posizioa emango digu (horrela justifikatzen da `FOR-DO` sententziaren aurrean dagoen `Seek` prozeduraren bigarren parametroa). Ikus ditzagun orain `FOR-DO` bigiztaren hiru iterazioak:

1. iterazioa

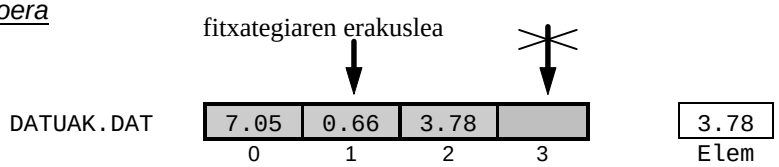
Hasierako egoera



Irakurketa



Bukaerako egoera



Azalpena

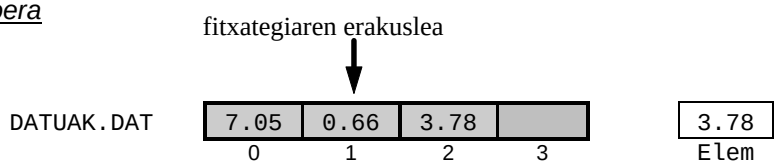
Aurreneko iterazioan `DATUAK.DAT` izeneko fitxategiari dagokion erakuslea azken elementuaren posizioa adierazten du `FOR-DO` bigiztan sartu aurretik `Seek(F,2)` bezalako zerbait egin delako. Iterazio honen hasieran `Elem` aldagaiak ez du balio ezagunik.

`Read(F,Elem)` irakurketa burutzean `Elem` aldagaiak `3.78` balioa hartzen duenez dagokion pantailaraketa gerta daiteke. Erakuslea automatikoki aurreratzen da.

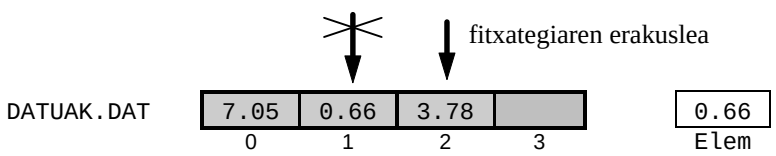
`Read` prozedurak erakuslea aurreratu egiten duelako, hurrengo iteraziorako bi posizioz atzeratuko dugu `Seek(F,UnekoPosizio-2)` bitartez.

2. iterazioa

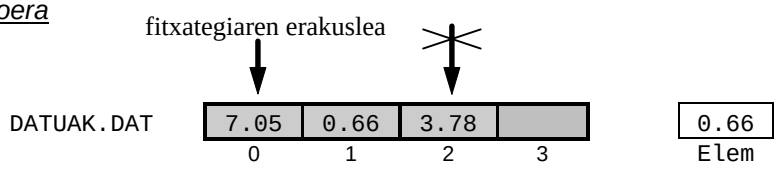
Hasierako egoera



Irakurketa



Bukaerako egoera



Azalpena

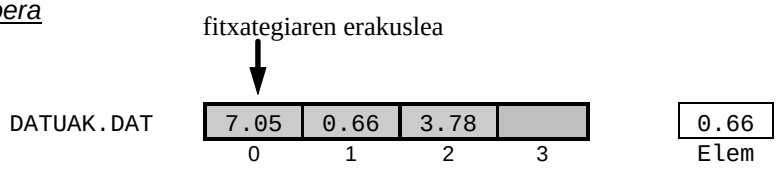
Aurreneko iterazioaren bukaeran `DATUAK.DAT` izeneko fitxategiari dagokion erakuslea 1 posizioa adierazten geratu da, dagoenekoz `Elem` aldagaiak `3.78` balioa mantentzen du.

`Read(F,Elem)` irakurketa burutzean `Elem` aldagaiak `0.66` balioa hartzen du fitxategitik zeukan `3.78`a galduz, eta erakuslea automatikoki aurreratzen da.

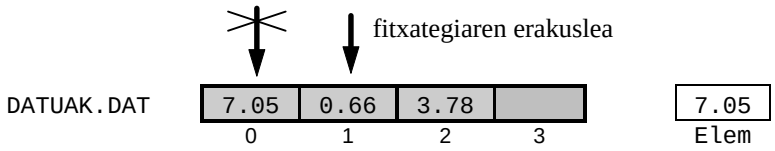
Lehen bezala, `Read` prozedurak erakuslea aurreratu egiten duelako, hirugarren iteraziorako bi posizioz atzeratuko dugu `Seek(F,UnekoPosizio-2)` bitartez. Hau da, `Read` ostean erakuslea zegoen 2 posiziotik 0 posiziora desplazatu.

3. iterazioa

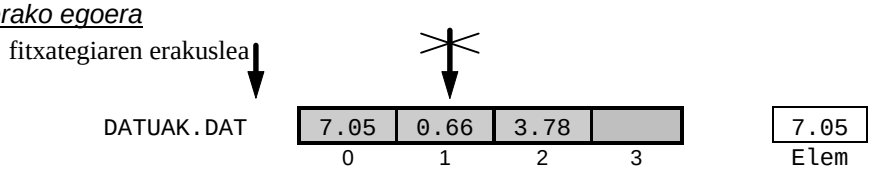
Hasierako egoera



Irakurketa



Bukaerako egoera



Azalpena

Bigarren iteraziotik irtetean `DATUAK.DAT` fitxategiaren erakuslea 0 posizioan geratu da, eta `Elem` aldagaiak `0.66` balioa dauka.

`Read(F,Elem)` irakurketa berriro burutzean `Elem` aldagaiak `7.05` balioa hartzen du fitxategitik, ondorioz zeukan `0.66`a galtzen da. `Read` prozedura exekutatzean erakuslea automatikoki aurreratzen da 1 posiziora igaroz.

Hirugarren iterazio hau azkenekoa delako ez litzateke fitxategiaren erakuslea bi

posizioz atzeratu beharrik (1 posiziotik –1 posiziora desplazatuz). Hala ere `FOR-DO` bigiztatik irten eta gero ez denez fitxategian irakurketa/idazketa operazio gehiagorik egiten algoritmoa ontzat jo daiteke.

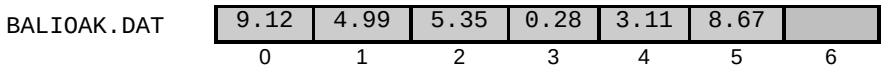
12.1.2.2.8 `Truncate` prozedura

`Truncate` prozeduraren bitartez fitxategia moztu ahal izango dugu. Erakuslearen uneko posizioa aintzat harturik `Truncate`–k bere ostean dauden fitxategiko elementu guztiak ezabatu egiten ditu, eta uneko posizio horretan fitxategiaren amaiera marka kokatzen da (erakuslea tokiz aldatzen ez denez `Truncate` eta gero `Eof` funtzioak `TRUE` balioko du).

`Truncate` darabilen `FitxategiBatMozten` adibide-programa ikus dezagun:

```
PROGRAM FitxategiBatMozten ;                  { \TP70\12\FILE_06.PAS }
TYPE
  DM_Fitxategi = FILE OF Real ;

VAR
   FitxIzen : String ;       { Fitxategi fisikoa }
   F : DM_Fitxategi ;        { Fitxategi logikoa }
   Elem : Real ;
   Posizioa : LongInt ;

BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;

   Assign (F, FitxIzen) ;
   Reset (F) ;               { Irakurketarako ireki }

   WriteLn (FitxIzen, ' fitxategiaren datuen esparrua:  0..', FileSize(F)-1) ;
   REPEAT
      Write ('Zein elementutik moztu nahi duzu? ') ;
      ReadLn (Posizioa) ;
   UNTIL (Posizioa >= 0) AND (Posizioa < FileSize(F)) ;

   Seek (F, Posizioa) ;      { Mozteko prestatu }
   Truncate (F) ;            { Moztu }

   Seek (F, 0) ;             { Fitxategiaren hasieran kokatu }
   WHILE NOT Eof (F) DO      { Fitxategia amaitzen ez den bitartean }
   BEGIN
      Read (F, Elem) ;
      WriteLn (FilePos(F)-1, '. elementua = ', Elem:0:8) ;
   END ;

   Close (F) ;
END.
```

Demagun `FitxategiBatSortzen` programa exekutatuz `BALIOAK.DAT` fitxategi hau lortu dugula bere edukia hau delarik:

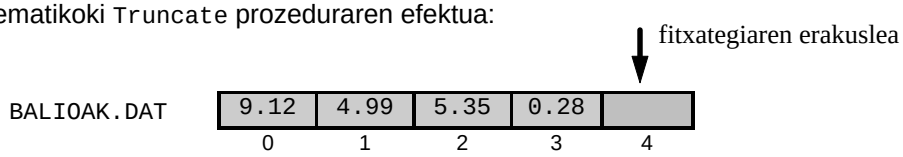


`Truncate`–k zertarako balio duen aurreratu dugunez `FitxategiBatMozten` programa erraz uler daiteke. Hasteko zenbaki errealak biltzen dituen fitxategi bat irekitzen da, ondoren fitxategiaren datuen aldearen mugak kontsideratuz zenbaki oso bat teklatuz hartzen da (zenbaki oso horrek mozketa non gertatuko den adieraziko du).

Hona hemen `FitxategiBatMozten` programaren baliozko exekuzio bat:

```
Fitxategiaren izena eman: BALIOAK.DAT
BALIOAK.DAT fitxategiaren datuen esparrua:  0..5
Zein elementutik moztu nahi duzu? 4
0. elementua = 9.12000000
1. elementua = 4.99000000
2. elementua = 5.35000000
3. elementua = 0.28000000
_
```

`Truncate`–k fitxategiari dagokion erakuslearen uneko posiziotik aurrera mozten du fitxategia. Horregatik `Truncate` aplikatu baino lehen `Seek(F,Posizioa)` prozeduraz erakuslea erabiltzaileak erabakitako tokian kokatzen da (adibidean 4 posizioan jartzen da erakuslea). Hau litzateke eskematikoki `Truncate` prozeduraren efektua:



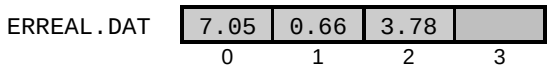
Hauek lirateke `Truncate` prozeduraren ezaugarriak:

Sintaxia:	`Truncate (Fitxategi_logiko)`
Parametroa:	`Fitxategi_logiko` `File` datu-mota, erreferentziaz (aldagai-parametroa) baina lehendik `Assign` egokiaren bitartez dagokion fitxategi fisikoa zehazturik badago, `Close` prozedurari esker itxita egongo da
Eginkizuna:	Fitxategiaren erakuslearen uneko posizioa zein den aintzat harturik, fitxategia moztu egiten du prozedura honek

12.1.2.2.9 `Erase` prozedura

`Erase` prozeduraren bitartez diskoan dagoen fitxategi bat ezabatu ahal izango dugu, D.O.S. sistema eragileko `delete` komandoaren zeregina betetzen du `Erase` prozedurak. `Erase` prozedura aplikatzean diskoan dagoen fitxategia itxita aurkituko da (`Close` ondoren aplikatzen da `Erase` errutina).

Demagun `FitxategiBatSortzen` programa exekutatuz `ERREAL.DAT` fitxategi hau lortu dugula eta bere balioak pantailaratu ondoren fitxategia ezabatzea nahi dugula:



`Erase` ikasteko balio duen `FitxategiBatEzabatzen` programa ikus dezagun:

```
PROGRAM FitxategiBatEzabatzen ;                  { \TP70\12\FILE_07.PAS }
TYPE
  DM_Fitxategi = FILE OF Real ;

VAR
   FitxIzen : String ;       { Fitxategi fisikoa }
   F : DM_Fitxategi ;        { Fitxategi logikoa }
   Elem : Real ;
```

```
BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   WriteLn ('Kopuru errealak dituen ', FitxIzen, ' fitxategitik pantailaratzen') ;

   Assign (F, FitxIzen) ;
   Reset (F) ;               { Irakurketarako ireki }

   WHILE NOT Eof (F) DO      { Fitxategia amaitzen ez den bitartean }
   BEGIN
      Read (F, Elem) ;       { Fitxategitik memoriara transferitu }
      WriteLn (FilePos(F) - 1, '. elementua = ', Elem:0:8) ;
   END ;

   Close (F) ;
   Erase (F) ;               { Fitxategia itxi ondoren ezabatu }
   WriteLn (FitxIzen, ' fitxategia diskotik ezabatuta') ;
END.
```

`ERREAL.DAT` fitxategia existituz gero `FitxategiBatEzabatzen` adibide-programaren irteera hau litzateke:

```
Fitxategiaren izena eman: ERREAL.DAT
Kopuru errealak dituen ERREAL.DAT fitxategitik pantailaratzen
0. elementua = 7.05000000
1. elementua = 0.66000000
2. elementua = 3.78000000
ERREAL.DAT fitxategia diskotik ezabatuta
_
```

`FitxategiBatEzabatzen` programa `ERREAL.DAT` fitxategiarekin bigarrenez exekutatzen saiatzean honelako irteera edukiko genukeela nabaria da:

```
Fitxategiaren izena eman: ERREAL.DAT
Kopuru errealak ERREAL.DAT fitxategitik pantailaratzen
Runtime error 002 at 0BF4:01F7.
_
```

Hauek lirateke `Erase` prozeduraren ezaugarriak:

Sintaxia: `Erase (Fitxategi_logiko)`

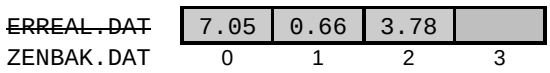
Parametroa: `Fitxategi_logiko`
`File` datu-mota, erreferentziaz (aldagai-parametroa) baina lehendik `Assign` egokiaren bitartez dagokion fitxategi fisikoa zehazturik badago, `Close` prozedurari esker itxita egongo da

Eginkizuna: Fitxategi jakin bat diskotik ezabatzen du. `Erase` eta `Rename` prozedurak `Close` ondoren aplikatzen dira.

12.1.2.2.10 `Rename` prozedura

`Rename` prozeduraren helburua eta D.O.S. sistema eragileko `ren` izeneko komandoaren helburua bera da, hots, fitxategi bati izena aldatzea. Kasu honetan ere `Rename` prozedura aplikatzean diskoan dagoen fitxategia itxita aurkituko da.

Demagun `FitxategiBatSortzen` programa exekutatuz `ERREAL.DAT` fitxategi hau lortu dugula eta bere balioak pantailaratu ondoren fitxategiaren izena `ZENBAK.DAT` izan dadila nahi dugula:



Hauxe da `FitxategiBatBerrizendatzen` programa::

```
PROGRAM FitxategiBatBerrizendatzen ;                { \TP70\12\FILE_08.PAS }
TYPE
  DM_Fitxategi = FILE OF Real ;

VAR
   FitxIzen, IzenBerri : String ;       { Fitxategi fisikoa }
   F : DM_Fitxategi ;                   { Fitxategi logikoa }
   Elem : Real ;

BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   WriteLn ('Kopuru errealak ', FitxIzen, ' fitxategitik pantailaratzen') ;

   Assign (F, FitxIzen) ;
   Reset (F) ;                      { Irakurketarako ireki }

   WHILE NOT Eof (F) DO
   BEGIN
      Read (F, Elem) ;
      WriteLn (FilePos(F) - 1, '. elementua = ', Elem:0:8) ;
   END ;

   Close (F) ;                      { Fitxategia itxi }

   Write ('Fitxategiaren izen berria: ') ;
   ReadLn (IzenBerri) ;

   Rename (F, IzenBerri) ;          { Fitxategia itxi ondoren berrizendatu }
   WriteLn (FitxIzen, ' fitxategia diskotik desagertuta') ;
   WriteLn (IzenBerri, ' fitxategia diskoan aurkitzen da') ;
END.
```

`ERREAL.DAT` fitxategia existituz gero `FitxategiBatBerrizendatzen` programaren irteera hau litzateke:

```
Fitxategiaren izena eman: ERREAL.DAT
Kopuru errealak dituen ERREAL.DAT fitxategitik pantailaratzen
0. elementua = 7.05000000
1. elementua = 0.66000000
2. elementua = 3.78000000
Fitxategiaren izen berria: ZENBAK.DAT
ERREAL.DAT fitxategia diskotik desagertuta
ZENBAK.DAT fitxategia diskoan aurkitzen da
_
```

Ez dugu ahaztu behar D.O.S. sistema eragileak fitxategien izenetarako baldintzak jartzen dituela, edozein izen ez dela onartzen alegia. Esate baterako aurreko programa hori exekutatzean `ZEN?.DAT` izena aukeratuko bagenu errorea sortuko litzteke `Rename` prozeduran:

```
Fitxategiaren izena eman: ERREAL.DAT
Kopuru errealak ERREAL.DAT fitxategitik pantailaratzen
Runtime error 002 at 0BF6:01FA.
_
```

Hauek lirateke laburbilduz `Rename` prozeduraren ezaugarriak:

Sintaxia:	`Rename (Fitxategi_logiko, Karaktere_katea)`

Parametroa:	`Fitxategi_logiko`
	`File` datu-mota, erreferentziaz (aldagai-parametroa)
	`Karaktere_katea`
	`String` datu-mota, fitxategi fisikoak izango duen izen berria. Bigarren hau baliozko parametroa izango da

Eginkizuna:	Prozedura deitu ondoren bere edukia ez da aldatzen bai ordea bere izena. `Karaktere_katea` aldagaiak gordeko duena onartua izan dadin sistema eragileak baldintzatzen du

12.1.3 Fitxategiak parametro bezala

12.1.1.3 puntuan esandakoa gogora ekarriz `FILE` fitxategi logikoa azpiprogrametara igortzen denean beti aldagai-parametro bezala pasatzen da, `FILE` aldagai bat beti erreferentziaz agertuko da azpiprogramen goiburuetan. Programadoreak fitxategiak lantzen dituen errutinak idazten dituenean aukera bi izango ditu, fitxategi logikoa bidaltzea azpiprogramara edo bestela fitxategi fisikoa igortzea (`FILE` datu-motako aldagaia edo `String` datu-motakoa).

Guk fitxategi fisikoaren izena pasatzearen alde gaudenez liburu honetan gehienetan horrela egingo dugu. Kontutan izan dezagun, nahiz eta azpiprogramaren barruan fitxategia aldatua izan, bere izena aldatzen ez den bitartean parametroa sarrerakoa izango dela (baliozko parametroa alegia).

Azpiprogrametan fitxategiekin lan egiteko hiru urrats emango dira (jarraian ikusten denez benetan urratsak lau dira):

0. Fitxategi logikoa eta fitxategi fisikoa `Assign` bitartez elkartu
1. Fitxategia ireki (`Reset` edo `Rewrite` prozedurak)
2. Idazketa edo/eta irakurketa operazioak burutu (`Write` edo/eta `Read` prozedurak)
3. Fitxategia `Close` bitartez itxi

Aspaldiko `FitxategiBatSortzen` programa berridatziko dugu, zenbaki errealak biltzen dituen fitxategiaren sorrera prozedura batean dagoelarik. `FitxategiBatSortzen` programa berrituaren izena `FitxategiaSortzekoProzedura` izango da, hona hemen bere sententziak:

```
PROGRAM FitxategiaSortzekoProzedura ;              { \TP70\12\FILE_09.PAS }
USES
  Crt ;
TYPE
  DM_Fitxategi = FILE OF Real ;

PROCEDURE DatuakBiltegitzen (FitxFisiko : String) ;
VAR
   F : DM_Fitxategi ;        { Fitxategi logikoa }
   Elem : Real ;
   Erantz : Char ;
BEGIN
   Assign (F, FitxFisiko) ;  { 0. urratsa:  fitxategiak elkartu }
   Rewrite (F) ;             { 1. urratsa:  idazketarako ireki }
```

```
   REPEAT
      Write ('Eman fitxategiaren ', FilePos(F), '. elementua: ') ;
      ReadLn (Elem) ;

      Write (F, Elem) ;      { 2. urratsa:  fitxategian idatzi }

      Write ('Gehiagorik? (B/E)  ') ;
      Erantz := ReadKey ;
      Erantz := UpCase (Erantz) ;
      WriteLn (Erantz) ;
   UNTIL Erantz = 'E' ;

   Close (F) ;               { 3. urratsa:  fitxategia itxi }
END ;


VAR
   FitxIzen : String ;
BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   WriteLn ('Kopuru errealak ', FitxIzen, ' fitxategian gordetzen') ;
   DatuakBiltegitzen (FitxIzen) ;
END.
```

`FitxategiaSortzekoProzedura` programan fitxategi bat sortuko dugu, programa nagusian dagoen aldagai bakarra fitxategi fisikoaren izenarena da eta `FitxIzen` aldagai horri programa nagusian balioa ematen zaionez `DatuakBiltegitzen` prozeduran sarrerako portaera izango du, hots, baliozko parametroa izango da.

`DatuakBiltegitzen` prozedura barruan fitxategiarekin lan egingo delako `FILE` datu-motako aldagairen beharko da. Fitxategia sortu eta datuz betetzen duen `DatuakBiltegitzen` prozeduran lehenago zerrendatutako urratsak ikus daitezke: fitxategien elkarketa eta irekiera, sarrerak edo/eta irteerak (adibide honetan operazio guztiak idazketak dira), eta, prozedura bukatu baino lehen fitxategiaren itxiera.

Ikaslearen lana litzateke 12-10 orrialdeko `FitxategiBatIkusten` programatik abiatuta bere ordezkoa lortzea, non fitxategiaren prozesaketa prozedura batean burutzen den. Honelako zerbait izango litzateke:

```
PROGRAM FitxategiaIkustekoProzedura ;              { \TP70\12\FILE_10.PAS }
TYPE
  DM_Fitxategi = FILE OF Real ;

PROCEDURE DatuakErakusten (FitxFisiko : String) ;
VAR
   F : DM_Fitxategi ;        { Fitxategi logikoa }
   Elem : Real ;
BEGIN
      { 0 eta 1. urratasak:  elkarketa eta irakurketarako ireki }

      { 2. urratsa:  irakurketak fitxategian }

      { 3. urratsa:  fitxategia itxi }

END ;                    { DatuakErakusten prozeduraren amaiera }

VAR
   FitxIzen : String ;
BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   WriteLn ('Kopuru errealak ', FitxIzen, ' fitxategitik pantailaratzen') ;
   DatuakErakusten (FitxIzen) ;
END.
```

12.1.4 Fitxategien gaineko eragiketak

Hurrengo puntuetan fitxategien gaineko zenbait eragiketa azalduko dugu, eragiketok beti azpiprogrametan gauzatuko ditugu eta horrela programa konplexu batetarako oinarrizko software elementuak izan daitezke.

Zenbaitetan zenbaki errealen fitxategiekin lan egingo dugu, baina beste batzutan erregistroen fitxategiak landuko ditugu. Beraz hemendik aurrera honelako definizioa sarritan ikusi ahal izango dugu, non fitxategien oinarria hiru eremuko erregistroa den:

```
TYPE
   DM_Fitxa = RECORD
                IzenDeiturak : String[49] ;
                Deialdia : Byte ;
                Nota : Real ;
              END ;
   DM_Fitxategi = FILE OF DM_Fitxa ;
```

Lau elementuko fitxategiaren eskema hau delarik:

GELA_1.DAT

Jokin	Pedro	Maite	Ane	
3	1	1	2	
6.5	8.6	7.3	4.7	
0	1	2	3	4

12.1.4.1 Sorrera

Fitxategi bat sortzeko aski da ezagutzen dugun `Rewrite` prozeduraren deia egitea, horren bitartez hutsik dagoen fitxategi berri bat sortzen da diskoan. Esate baterako jarraian erakusten den programa exekutatu ondoren fitxategi berri bat izango dugu diskoan, beltzez dagoen aginduak sortzen du fitxategia, eta, ez denez fitxategian inolako idazketarik egin bere tamaina 0 bytekoa izango da:

```
PROGRAM FitxategiBatenSorrera1 ;                  { \TP70\12\FILE_11.PAS }
TYPE
  DM_Fitxategi = FILE OF Real ;

PROCEDURE FitxategiaSortu (FitxFisiko : String) ;
VAR
   F : DM_Fitxategi ;                            { Fitxategi logikoa }
BEGIN
   Assign (F, FitxFisiko) ;                      { Fitxategiak elkartu }
   Rewrite (F) ;                                 { Idazketarako ireki }
   Close (F) ;                                   { Fitxategia itxi }
END ;

FUNCTION ElementuKopurua (FitxFisiko : String) : LongInt ;
VAR
   F : DM_Fitxategi ;                            { Fitxategi logikoa }
BEGIN
   Assign (F, FitxFisiko) ;                      { Fitxategiak elkartu }
   Reset (F) ;                                   { Irakurketarako ireki }
   ElementuKopurua := FileSize (F) ;
   Close (F) ;                                   { Fitxategia itxi }
END ;
```

```
VAR
   FitxIzen : String ;
   ByteKop : LongInt ;
BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;

   FitxategiaSortu (FitxIzen) ;
   WriteLn (FitxIzen, ' izeneko fitxategia sortu egin da') ;

   ByteKop := ElementuKopurua (FitxIzen) * SizeOf (Real) ;
   WriteLn (FitxIzen, ' fitxategiak ', ByteKop, ' byte ditu') ;
END.
```

Bi izan daitezke `FitxategiBatenSorrera1` programaren erantzunak hura exekutatzen dugunean. Baldin eta erabiltzaileak `BERRIA.DAT` izena ematen badu `FitxategiBatenSorrera1` programaren irteera zein den erraz asmatzen da, kontutan izan dezagun fitxategi fisikorentzat aukeratu den izena “egokia” dela D.O.S. sistema eragilearen ikuspegitik begiratuta, beraz ez da arazorik suertatuko `FitxategiBatenSorrera1` exekutatzean:

```
Fitxategiaren izena eman: BERRIA.DAT
BERRIA.DAT izeneko fitxategia sortu egin da
BERRIA.DAT fitxategiak 0 byte ditu
_
```

Baldin eta erabiltzaileak sartzen duen izenak D.O.S. sistema eragileak fitxategien izenetarako jartzen dituen baldintzak betetzen ez baditu, errorea gertatuko da `Rewrite`-ra iristean. Esate baterako erabiltzaileak izentzat `BERRI?.DAT` ematen badu[9] aurreko programak honelako erantzuna izango du:

```
Fitxategiaren izena eman: BERRI?.DAT
Runtime error 002 at 0BF5:01EB.
_
```

Exekuzio denborako `002` erroreari dagokion literala “`file not found`“ da eta *normalean* `FitxategiBatenSorrera1` programa abortatu egingo litzateke. Exekuzio denborako errore hori gertatzean programa eten egingo den ala ez `$I` konpilazio direktibaren menpekoa da (ikus **12.1.4.2 Existentzia** puntua).

`FitxategiBatenSorrera1` programan ez da informaziorik fitxategian gordetzen, baina gehienetan fitxategia sortzeaz gain datuak gorde behar izaten dira, adibidez hau litzateke ikasleari lehenago proposatu zaion lanaren ebazpena (`FitxategiBatSortzen` aspaldiko programatik abiatuta bere ordezkoa lortzea erabiltzailearen prozedura diseinatuz):

```
PROGRAM FitxategiBatenSorrera2 ;                { \TP70\12\FILE_12.PAS }
USES
  Crt ;
TYPE
  DM_Katea = String [49] ;
  DM_Fitxa = RECORD
                IzenDeiturak : DM_Katea ;
                Deialdia : Byte ;
                Nota : Real ;
             END ;
  DM_Fitxategi = FILE OF DM_Fitxa ;
```

[9] Dakigunez D.O.S. sistema eragilerako `?` karaktereak esanahi berezia du eta ezin da fitxategien izenen partaidea izan.

```
PROCEDURE DatuakBiltegitzen (FitxFisiko : String) ;
VAR
   F : DM_Fitxategi ;        { Fitxategi logikoa }
   Elem : DM_Fitxa ;
   Erantz : Char ;
BEGIN
   Assign (F, FitxFisiko) ;
   Rewrite (F) ;             { Idazketarako ireki }

   REPEAT
      Write ('Eman ', FilePos(F), '. ikaslearen izen-deiturak: ') ;
      ReadLn (Elem.IzenDeiturak) ;
      Write ('Eman ', FilePos(F), '. ikaslearen deialdia:      ') ;
      ReadLn (Elem.Deialdia) ;
      Write ('Eman ', FilePos(F), '. ikaslearen nota:          ') ;
      ReadLn (Elem.Nota) ;

      Write (F, Elem) ;      { Memoriatik fitxategira transferitu }

      Write ('Gehiagorik? (B/E)  ') ;
      Erantz := ReadKey ;
      Erantz := UpCase (Erantz) ;
      WriteLn (Erantz) ;
   UNTIL Erantz = 'E' ;

   Close (F) ;
END ;

VAR
   FitxIzen : String ;
BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   WriteLn ('Ikasleen datuak ', FitxIzen, ' fitxategian gordetzen') ;
   DatuakBiltegitzen (FitxIzen) ;
END.
```

12.1.4.2 Existentzia

Dakigunez `Assign` prozedura deitu ondoren, fitxategi bitarrak irekitzeko bi prozedura eskaintzen ditu Turbo Pascal lengoaiak: `Rewrite` eta `Reset`. Aurreko puntuan azaldu den bezala `Rewrite` bitartez fitxategi bat irekitzean, fitxategia berriki sortzen delako ez da arazorik gertatzen izena ondo emanik egonez gero. Baina `Reset` bidez fitxategia irekitzean aldez aurretik sorturik egon behar duela ikasi genuen **12.1.2.2.3** puntuan, hots, nahiz eta fitxategi fisikoaren izena D.O.S.ren bat etorri `Reset`-ek ezin du ireki fitxategia diskoan aurkitzen ez badu. Ikas dezagun orain fitxategi bat diskoan ote dagoen aztertzen.

Fitxategi baten existentzia kontrolatzeko aurredefinituriko `IOresult` funtzioaren bitartez egiten da Turbo Pascal lengoaian. Gogora dezagun zortzigarren kapituluan aipatutako `{$I±}` konpilazio direktiba (sarrera/irteeraren froga burutzen duen direktiba).

Sarrera eta irteera prozeduretan konpiladoreak erroreak konprobatzen ditu, erroreren bat suertatzean dagokion mezua erakutsi eta programa gelditzen da. Esate baterako, programa batean `Zbk` aldagaia `Integer` bezala definitu bada `ReadLn(Zbk)` egitean zenbaki oso bat itxaroten du programak, eta zenbakia sartu beharrean letra bat ematen badugu programaren exekuzioa eteten da `Run time error 106 at XXXX:YYYY` agertuz[10].

Gerta daiteke nahiz eta sarrerak gaizki eman programa ez etetea nahi izatea, horretarako `$I` direktiba indargabetu behar dugu `{$I-}` jarriz. Horrela zenbaki oso bat itxaroten duen `Zbk` aldagairi `M` letra teklatuz esleitzen bazaio, `{$I-}` direktibari esker errorea gertatu arren

[10] Programak `106` errore-kodea itzultzen du, bere literala: `Error 106: Invalid numeric format.`

exekuzioa ez da eteten. Baina programadorea `IOresult` funtzioaren emaitza aztertuz arazoa prozesatuko du, izan ere Turbo Pascal lengoaiaren aurredefinituriko `IOresult` funtzioak sarrera/ireera azken eragiketaren balioa itzultzen du.

Beraz, programa batean `Zbk` irakurtzean agertzen den errorearen kodea (`106` zenbaki osoa) `IOresult` funtzioak jaso eta itzultzen digu, azken sarrera/irteera operazioan errorerik ez bada egon `IOresult` funtzioak `0` itzultzen du. Modu beretsuan, existitzen ez den fitxategia `Reset` bitartez ireki nahi denean `IOresult` funtzioak `2` kode-errorea itzultzen du (fitxategia existitzean errorerik ez da egongo eta `IOresult` funtzioak `0` itzultzen du).

Froga dezagun existitzen ez den fitxategi bat `Reset` bitartez irekitzen saiatzean `IOresult` funtzioak zer itzultzen digun Turbo Pascal 7.0 bertsioan. `IOresultFitxategiekin` programa eten ez dadin `$I` direktiba indargabeturik dago.

```
PROGRAM IOresultFitxategiekin ;                  { \TP70\12\FILE_13.PAS }
TYPE
  DM_Fitxategi = FILE OF Real ;

VAR
   FitxIzen : String ;
   F : DM_Fitxategi ;        { Fitxategi logikoa }
   Emaitza : Integer ;
BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;

   Assign (F, FitxIzen) ;
{$I-}
   Reset (F) ;               { Irakurketarako ireki }
{$I+}

   Emaitza := IOresult ;
   WriteLn ('IOresult = ', Emaitza) ;

   IF Emaitza = 0 THEN
     Close (F) ;             { Ireki bada itxi }
END.
```

`IOresultFitxategiekin` programa egikaritzean sartuko dugun izena D.O.S. sistema eragileak onar ditzakeenak izango dira (izena gaizki emanez gero ezinezkoa zaiolako fitxategia irekitzea). Arazoa beste toki batetik etor daiteke, hots, sartutako izena egokia izanik horrelako fitxategirik ez egotea diskoan, esate baterako, demagun zenbaki errealak gordetzen dituen `DATUAK.DAT` diskoan existitzen dela baina `KOPURUAK.DAT` izeneko fitxategirik ez dagoela (**12.1.2.2.3 `Reset` prozedura** puntuan emandako adibideak dira) eta `IOresultFitxategiekin` programa exekutatzen dugula. Hauek dira irteerak:

```
Fitxategiaren izena eman: DATUAK.DAT
IOresult = 0
_
```

```
Fitxategiaren izena eman: KOPURUAK.DAT
IOresult = 2
_
```

Ikusten denez fitxategi fisikoa diskoan aurkitzen ez denean `IOresult` funtzioak `2` kodea itzultzen du, eta `0` kodea aurkitzen denean. Baina gauzarik garrantzitsuena exekuzio bietan `IOresultFitxategiekin` programa bukaeraino iristen dela da, nahiz eta sarrera errorerik egon ez dela eteten. Hori, esan dugun bezala, `$I` direktibari esker lortzen da. Amaitu baino lehen `Close` prozedura barneratzen duen `IF-THEN` sententzia kenduko bagenu, existitzen ez den `KOPURUAK.DAT` izeneko fitxategiari irekita ez delako ezin zaio `Close` aplikatu[11]:

[11] `103` erroreari dagokion literala: `Error 103: File not open.`

```
   { IF Emaitza = 0 THEN        indargabeturik }
     Close (F) ;             { Ireki bada itxi }
END.
```

```
Fitxategiaren izena eman: KOPURUAK.DAT
IOresult = 2
Runtime error 103 at 0BF4:00D4.
_
```

Fitxategi baten existentzia kontrolatzeko funtzio boolear bat idatziz fitxategiaren datuak pantailaratzen duen programa osatua azal dezagun jarraian:

```
PROGRAM FitxategiBatenExistentzia ;                  { \TP70\12\FILE_14.PAS }
TYPE
  DM_Fitxategi = FILE OF Real ;

FUNCTION Existentzia (FitxFisiko : String) : Boolean ;
VAR
   F : DM_Fitxategi ;        { Fitxategi logikoa }
BEGIN
   Assign (F, FitxFisiko) ;
{$I-}
   Reset (F) ;               { Irakurketarako ireki }
{$I+}
   IF IOresult = 0 THEN      { Errorerik gertatu den frogatu }
   BEGIN
      Existentzia := TRUE ;
      Close (F) ;            { Ireki bada itxi }
   END
   ELSE
      Existentzia := FALSE ;
END ;

PROCEDURE DatuakErakusten (FitxFisiko : String) ;
VAR
   F : DM_Fitxategi ;        { Fitxategi logikoa }
   Elem : Real ;
BEGIN
   Assign (F, FitxFisiko) ;
   Reset (F) ;
   WHILE NOT Eof (F) DO
   BEGIN
      Read (F, Elem) ;
      WriteLn (FilePos(F)-1, '. elementua = ', Elem:0:8) ;
   END ;
   Close (F) ;
END ;

VAR
   FitxIzen : String ;
BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;

   IF Existentzia (FitxIzen) THEN
   BEGIN
      WriteLn ('Kopuru errealak ', FitxIzen, ' fitxategitik pantailaratzen') ;
      DatuakErakusten (FitxIzen) ;
   END
   ELSE
      WriteLn (FitxIzen, ' izeneko fitxategirik ez da aurkitu') ;
END.
```

Fitxategiaren existentziaz jabetzeko `$I` direktiba indargabetu eta `IOresult` funtzioaren emaitza aztertzen da. Fitxategia diskoan ez dagoenean `IOresult` funtzioak konpiladore batetik bestera balio ezberdinak itzul ditzake, baina edozein bertsiotan sarrera/irteera errorerik ez dagoenean `IOresult` funtzioak `0` itzultzen du. Horregatik, bateragarritasuna gordetzeagatik, `Existentzia` funtzioan nahiago izan dugu `IOresult` funtzioaren emaitza `0`-rekin konparatu eta ez `2`-rekin.

12.1.4.3 Ibilera

Arrayak azaldu genituenean ibilera zer den definitu genuen: array baten elementu guztiei prozesaketa jakin bat aplikatu (ahaztu gabe arrayaren luzera efektiboak bere elementuen kopurua ematen digula). Fitxategietan, `SizeOf` eta `Eof` funtzioak erabilgarri ditugula ñabardura erantsiz, gauza bera esan daiteke diskoan dagoen fitxategiaren elementu guztiei prozesaketa jakin bat aplikatzen zaiela alegia.

Esaterako, erregistroak biltegitzen dituen `GELA_1.DAT` fitxategiaren edukia pantailatik ateratzea fitxategiaren ibilera izango da:

GELA_1.DAT

Jokin	Pedro	Maite	Ane	
3	1	1	2	
6.5	8.6	7.3	4.7	
0	1	2	3	4

Hauek lirateke `FitxategiBatenIbilera1` programaren sententziak eta irteera:

```
PROGRAM FitxategiBatenIbilera1 ;                { \TP70\12\FILE_15.PAS }
TYPE
  DM_Katea = String [49] ;
  DM_Fitxa = RECORD
               IzenDeiturak : DM_Katea ;
               Deialdia : Byte ;
               Nota : Real ;
             END ;
  DM_Fitxategi = FILE OF DM_Fitxa ;

PROCEDURE DatuakErakusten (FitxFisiko : String) ;
VAR
   F : DM_Fitxategi ;        { Fitxategi logikoa }
   Elem : DM_Fitxa ;
BEGIN
   Assign (F, FitxFisiko) ;
   Reset (F) ;

   WHILE NOT Eof (F) DO
   BEGIN
      Read (F, Elem) ;
      Write (FilePos(F)-1, '. ikaslea:  ', Elem.IzenDeiturak:49) ;
      WriteLn (Elem.Deialdia:5, Elem.Nota:8:2) ;
   END ;

   Close (F) ;
END ;

VAR
   FitxIzen : String ;
BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   WriteLn ('Ikasleen zerrenda ', FitxIzen, ' fitxategian') ;
   DatuakErakusten (FitxIzen) ;
END.
```

```
Fitxategiaren izena eman: GELA_1.DAT
Ikasleen zerrenda GELA_1.DAT fitxategian
0. ikaslea:                                          Jokin     3    6.50
1. ikaslea:                                          Pedro     1    8.60
2. ikaslea:                                          Maite     1    7.30
3. ikaslea:                                            Ane     2    4.70
_
```

Demagun `GELA_1.DAT` fitxategiari bigarren prozesaketa bat egin nahi diogula, ikasleen deialdiak inkrematzea eta diskoan erregistratzea lortuko dugu `FitxategiBatenIbilera2` izena duen programarekin. `FitxategiBatenIbilera2` programa exekutatu ondoren `GELA_1.DAT` fitxategiaren irudia hauxe izango da:

GELA_1.DAT

Jokin	Pedro	Maite	Ane	
4	**2**	**2**	**3**	
6.5	8.6	7.3	4.7	
0	1	2	3	4

Hau litzateke `FitxategiBatenIbilera2` programa:

```
PROGRAM FitxategiBatenIbilera2 ;                { \TP70\12\FILE_16.PAS }
TYPE
  DM_Katea = String [49] ;
  DM_Fitxa = RECORD
               IzenDeiturak : DM_Katea ;
               Deialdia : Byte ;
               Nota : Real ;
             END ;
  DM_Fitxategi = FILE OF DM_Fitxa ;

PROCEDURE DatuakProzesatzen (FitxFisiko : String) ;
VAR
   F : DM_Fitxategi ;                           { Fitxategi logikoa }
   Elem : DM_Fitxa ;
BEGIN
   Assign (F, FitxFisiko) ;
   Reset (F) ;                                  { Irakurketa-Idazketarako ireki }

   WHILE NOT Eof (F) DO
   BEGIN
      Read (F, Elem) ;                          { Fitxategitik irakurri }
      Elem.Deialdia := Elem.Deialdia + 1 ;      { Deialdia inkrementatu }
      Seek (F, FilePos(F) - 1) ;                { Erakuslea kokatu }
      Write (F, Elem) ;                         { Fitxategian idatzi }
   END ;

   Close (F) ;                                  { Fitxategia itxi }
END ;

VAR
   FitxIzen : String ;
BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   WriteLn ('Ikasleen deialdi kopurua ', FitxIzen, ' fitxategian eguneratzen') ;
   DatuakProzesatzen (FitxIzen) ;
END.
```

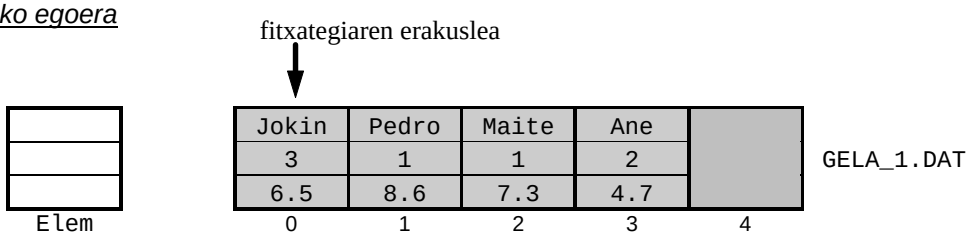
`GELA_1.DAT` fitxategiaren gainean burututako bigarren ibilera honetan lau iterazio egongo dira ere, baina lehen ez bezala iterazio bakoitzean fitxategian sarrera/irteera bi egiten dira (hasteko fitxategian irakurketa batez `Elem` aldagaiari balioak ematen zaizkio, eta deialdiaren kopurua emendatu ondoren `Elem` berrituaren idazketa burutzen da fitxategian).

Prozesaketaren errepikapena hau delarik, aurreneko eta azkeneko iterazioei dagozkien eskemak egin ditzagun:

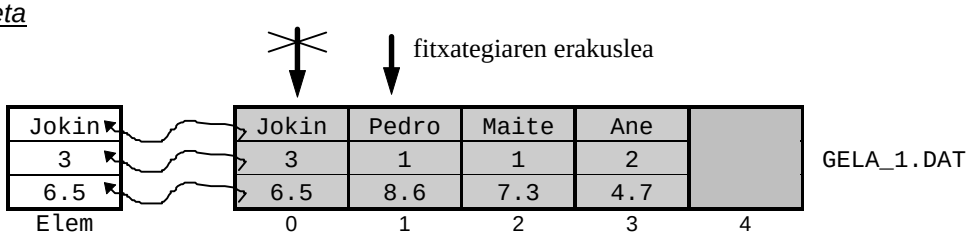
```
WHILE NOT Eof (F) DO                          { Fitxategiaren amaiera testatu }
BEGIN
   Read (F, Elem) ;                           { Erregistro oso bat irakurri }
   Elem.Deialdia := Elem.Deialdia + 1 ;       { Deialdia inkrementatu }
   Seek (F, FilePos(F) - 1) ;                 { Fitxategiaren erakusle atzeratu }
   Write (F, Elem) ;                          { Erregistro oso bat idatzi }
END ;
```

1. iterazioa

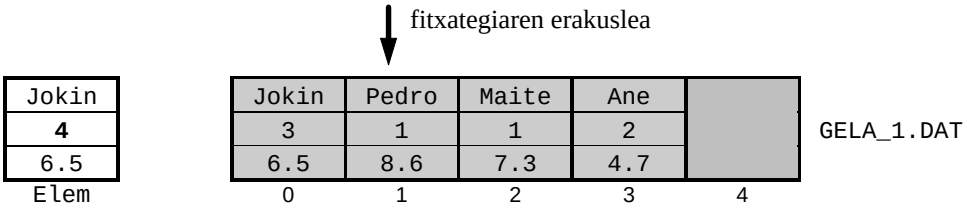
Hasierako egoera



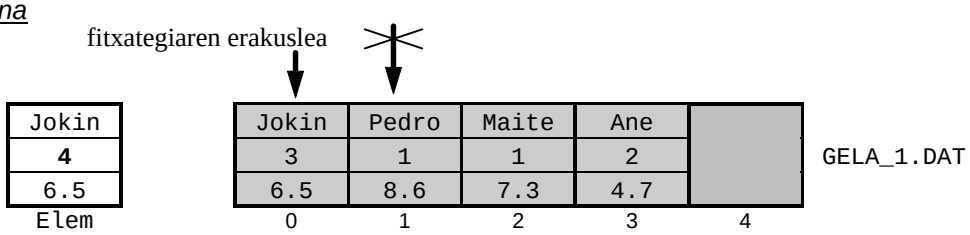
Irakurketa



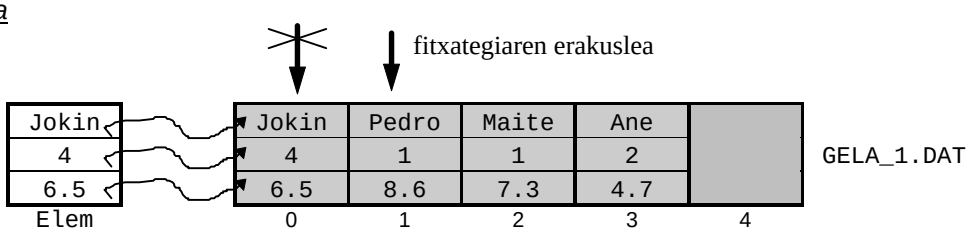
Aldaketa



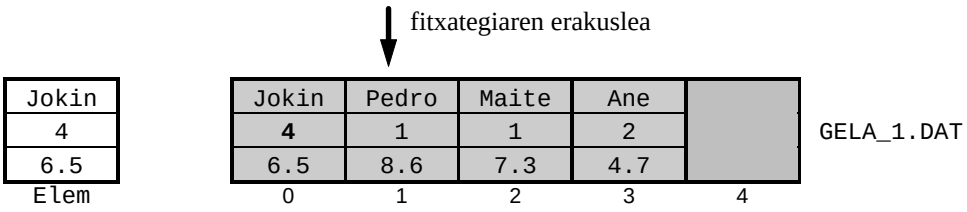
Kokapena



Idazketa



Bukaerako egoera



Azalpena

Aurreneko iterazioan `GELA_1.DAT` izeneko fitxategiari dagokion erakuslea `0` posizioan dago arestian egindako `Reset` prozedurak bertan uzten duelako. Iterazio honen hasieran `Elem` aldagaiak ez du balio ezagunik.

`Read(F,Elem)` irakurketa burutzean `Elem` aldagaiak `Jokin/3/6.5` balioak hartzen ditu memorian gordez. Erakuslea automatikoki aurreratzen da.

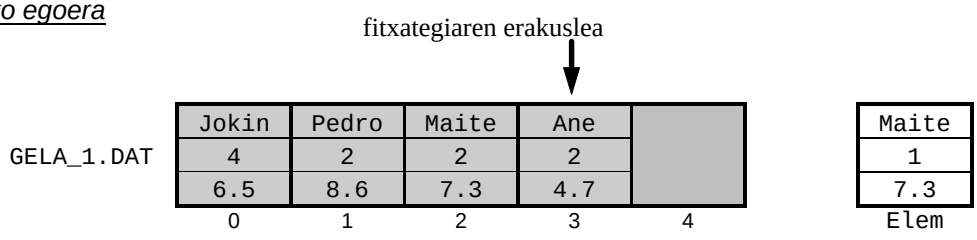
Memorian dagoen `Elem` aldagaiaren deialdiaren eremua inkrementa daiteke, beraz `Elem` aldagaiak `Jokin/`**`4`**`/6.5` balioak izango ditu.

`Read` prozedurak fitxategiaren erakuslea aurreratu egiten duelako, hurrengo idazketa `1` posizioan egingo litzateke `Pedro/1/8.6` erregistroa zapalduz. Hori dela eta, fitxategiaren erakuslea atzeratuko dugu `Seek(F,FilePos(F)-1)` eginez.

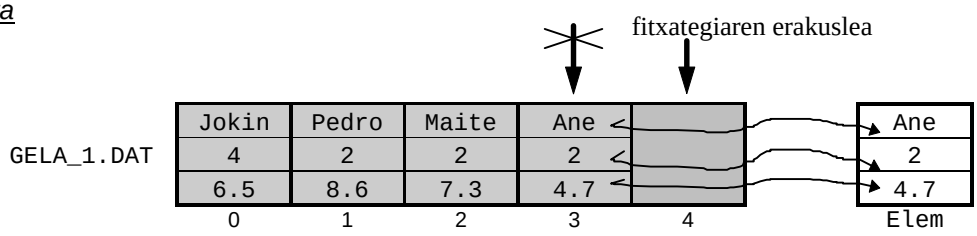
`Write(F,Elem)` idazketa egitean fitxategian `Jokin/4/6.5` balioak biltegituko dira zaharrak ordezkatuz, eta erakuslea automatikoki aurreratzen da. Kontutan izan dezagun `Read` egitean fitxategitik irakur daitekeen elementuen kopuru maximoa (eta minimoa) erregistro bat dela, hau da, ezin daitezke `Read` bitartez elementu bat baino gehiago irakurri aldi bakoitzeko, bestalde ezin da inolaz `Read` bitartez elementu baten eremu bakarra irakurri (erregistro osoa irakurri beharra dago). `Read` prozedurari buruzkoak `Write` prozedura estandarrarentzat errepikatuko ditugu: idazketa batean ezin da elementu bat baino gehiago fitxategian erregistratu, eta ezinezkoa zaigula idazketa selektiboak egitea (hots, erregistro baten eremu isolatua fitxategian idaztea).

4. iterazioa

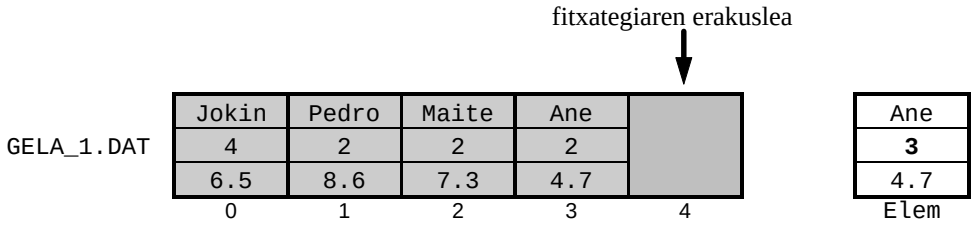
Hasierako egoera



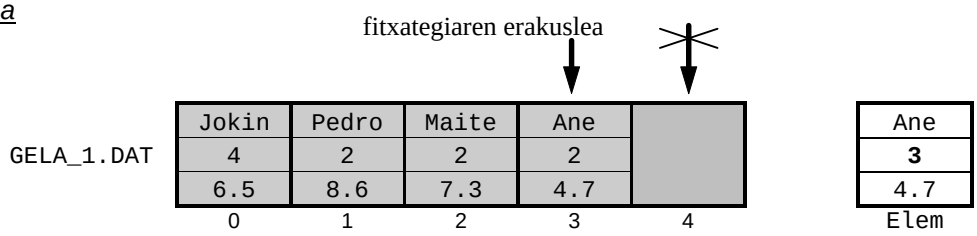
Irakurketa



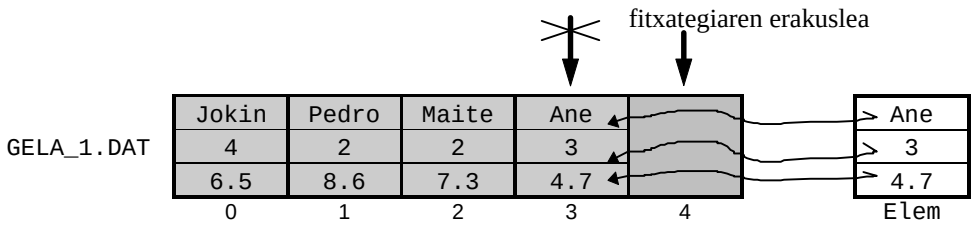
Aldaketa



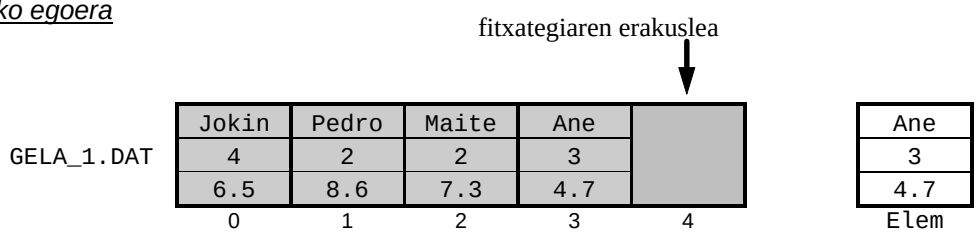
Kokapena



Idazketa



Bukaerako egoera



Azalpena

Hirugarren iterazioaren bukaeran `Maite/2/7.3` balioak dituen erregistroa idatziko litzateke fitxategiaren `3` posizioan eta erakuslea automatikoki aurreratuko litzateke. Beraz, laugarren eta azken iterazioaren hasieran fitxategiari dagokion erakuslea `3` posizioan dagoelako `WHILE-DO` barruko sententziak exekutatuko dira. Iterazio honen hasieran `Elem` aldagaiak dituen balioak aipatutako `Maite/2/7.3` dira.

`Read(F,Elem)` irakurketa burutzean `Elem` aldagaiak `Ane/2/4.7` balioak hartzen ditu memoriara eramanez. Erakuslea automatikoki aurreratzen da eta ondorioz `Eof` funtzioak `TRUE` itzuliko luke.

Orain memorian dagoen `Elem` aldagaiaren `Deialdia`-ren eremua `2`-tik `3`-ra inkrementatzen delako `Elem` aldagaiak `Ane/3/4.7` balio berriak izango ditu.

`Read` prozedurak fitxategiaren erakuslea aurreratu egiten duelako une honetan fitxategiaren amaierako marka adieraziko du erakusleak, hor dagoela idatziz gero elementu berri bat gehituko genioke fitxategiari, horregatik orain ere fitxategiaren erakuslea atzeratuko dugu `Seek(F,FilePos(F)-1)` eginez.

`Write(F,Elem)` idazketa egitean fitxategian `Ane/3/4.7` balioak biltegituko dira zaharrak ordezkatuz. Kontura gaitezen idazketa honek fitxategiaren erakuslea aurreratzen duela berriro eta fitxategiaren bukaeran kokatzen dela, ondorioz `WHILE-DO` aginduaren baldintza ebaluatzerakoan `FALSE` lortuko denez bigizta eten egingo da fitxategia itxiz eta prozedura bukatutzat emanez.

Fitxategi baten osagaia aldatzen duen prozedura **12.1.4.6 Aldaketa** izeneko puntuan garatuko dugu.

12.1.4.4 Bilaketa

Informazioaren tratamenduan askotan gertatzen den zeregina da bilaketa eragiketa. Array datu-mota landu genuenean bilaketa honela enuntziatu zen: gako bat emanik, balio hori duen fitxategiaren lehen[12] elementuari dagokion posizioa aurkitu. Bilaketa eragiketaren emaitza `LongInt` datu-motako zenbaki oso bat izango da, zehatzago `0` eta `FileSize` artean dagoen zenbaki osoa. Bilaketaren emaitza aipatu dugun `0..FileSize` esparruko zenbakia denean gakoa fitxategian aurkitzen dela uler daiteke, baina bilatzen den gakoa fitxategian ez dagoela adierazi ahal izateko bilaketaren emaitza `0` eta `FileSize` artean ez dagoen edozein zenbaki izan daiteke (gure adibideetan bilatzen den gakoa fitxategian ez dagoenean bilaketa gauzatzen duen funtzioak `-1` itzuliko dio modulu deitzaileari).

Bilaketa batean egoera bi egon daitezkeela kontutan izanik (gakoa fitxategian egotea ala ez egotea) hura ebazten duen algoritmoan bikoitza den baldintza bat izango dugu beti. Fitxategiaren hasieratik abiatuta prozesu errepikakor batean oinarritzen da algoritmoa, iterazio bakoitzean fitxategiaren elementu bat irakurri eta gakoarekin alderatuko da (prozesatzen ari den elementua gakoarekin pareketzen bada bigizta moztu, bestela jarraitu); edozein kasutan bigiztan fitxategiaren datuen aldea landuko da soilik (`Eof` funtzioak `TRUE` itzultzean bigizta eten egingo da). Ikus algoritmoaren pseudokodea:

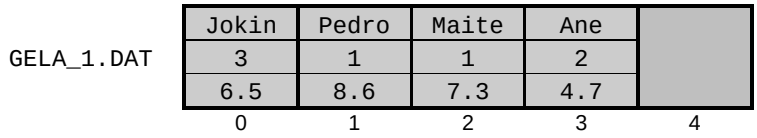
```
Reset(F)
Aurkitua ← FALSE
BAI-BITARTEAN (NOT Eof(F)) AND (NOT Aurkitua) EGIN
     Read(F,Elem)
     BALDIN-ETA Elem = Gakoa ORDUAN
          Aurkitua ← TRUE
AMAIA_ BAI-BITARTEAN

BALDIN-ETA Aurkitua ORDUAN
    Gakoa-ren posizioa FilePos(F)-1 da
BESTELA
    Gakoa ez dago fitxategian
AMAIA_BALDIN-ETA

Close(F)
```

[12] Gako hori elementu batek baino gehiagok dutela fitxategian suposatuz.

Hau litzateke `FitxategiBatenIbilera2` programa:

```
PROGRAM BilaketaFitxategietan ;                  { \TP70\12\FILE_17.PAS }
TYPE
  DM_Katea = String [49] ;
  DM_Fitxa = RECORD
               IzenDeiturak : DM_Katea ;
               Deialdia : Byte ;
               Nota : Real ;
             END ;
  DM_Fitxategi = FILE OF DM_Fitxa ;

FUNCTION BilaketaEgin (FitxFisiko : String; Gakoa : DM_Katea) : LongInt ;
VAR
   F : DM_Fitxategi ;
   Elem : DM_Fitxa ;
   Aurkitua : Boolean ;
BEGIN
   Assign (F, FitxFisiko) ;
   Reset (F) ;                                   { Irakurketarako ireki }

   Aurkitua := FALSE ;
   WHILE (NOT Eof (F)) AND (NOT Aurkitua) DO
   BEGIN
      Read (F, Elem) ;                           { Fitxategitik irakurri }
      IF Elem.IzenDeiturak = Gakoa THEN
        Aurkitua := TRUE ;
   END ;

   IF Aurkitua THEN
     BilaketaEgin := FilePos(F) - 1
   ELSE
     BilaketaEgin := -1 ;

   Close (F) ;                                   { Fitxategia itxi }
END ;

PROCEDURE DatuakPantailaratu (FitxFisiko : String; Posizioa : LongInt) ;
VAR
   F : DM_Fitxategi ;
   Elem : DM_Fitxa ;
BEGIN
   Assign (F, FitxFisiko) ;
   Reset (F) ;                                   { Irakurketarako ireki }

   Seek (F, Posizioa) ;
   Read (F, Elem) ;                              { Fitxategitik irakurri }
   Write (Posizioa, '. ikaslea:  ') ;
   WriteLn (Elem.IzenDeiturak:49, Elem.Deialdia:5, Elem.Nota:8:2) ;

   Close (F) ;                                   { Fitxategia itxi }
END ;


VAR
   FitxIzen : String ;
   Gakoa : DM_Katea ;
   Posizioa : LongInt ;
BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   Write ('Ikaslearen izen deiturak eman: ') ;
   ReadLn (Gakoa) ;
   Posizioa := BilaketaEgin (FitxIzen, Gakoa) ;
   IF Posizioa = -1 THEN
      WriteLn (Gakoa, ' izeneko ikaslerik ez dago ', FitxIzen, ' fitxategian')
   ELSE
      DatuakPantailaratu (FitxIzen, Posizioa) ;
END.
```

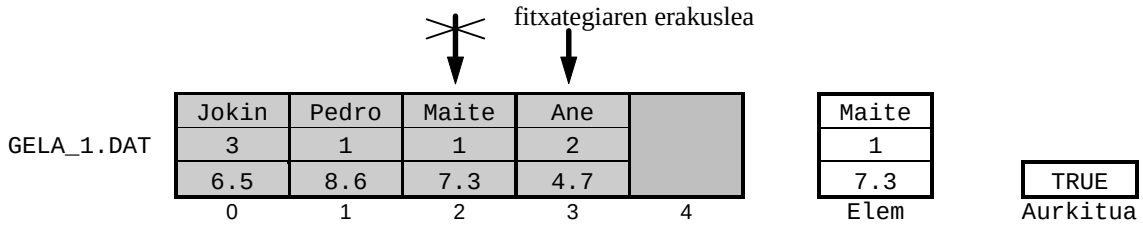
Demagun erregistroak biltegitzen dituen `GELA_1.DAT` fitxategiaren edukia hau dela eta `Maite` izena duen lehen ikaslearen datuak pantailan erakutsi nahi direla:



`BilaketaFitxategietan` programaren irteera hau izango litzateke:

```
Fitxategiaren izena eman: GELA_1.DAT
Ikaslearen izen deiturak eman: Maite
2. ikaslea:                                        Maite     1    7.30
_
```

`Maite` datu horrekin `BilaketaEgin` funtzioan fitxategia itxi baino lehentxeago honelako irudia izango genuke, non azken irakurketak (`Maite/1/7.3` balioak memoriara eramaten dituena) erakuslea `3` posizioan utzi duen:

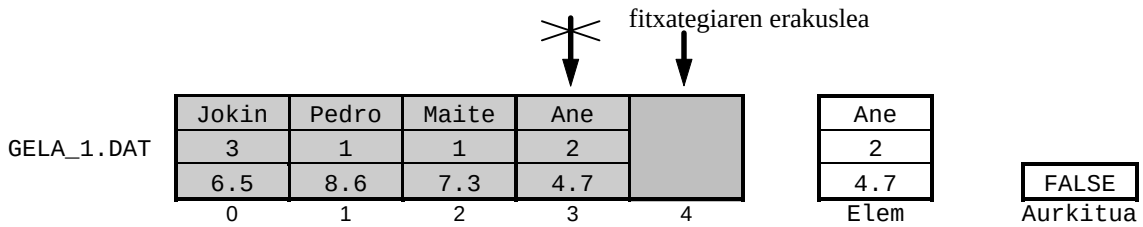


Erraz ulertzen denez, azken iterazio horren barruan `Gakoa` parametroak duen balioa `Maite` izanik `Elem.IzenDeiturak` eremuak duenarekin parekatzen da eta ondorioz `Aurkitua` aldagaia `TRUE` izango da, eta honek `WHILE-DO` aginduaren baldintza ez betetzea suposatzen du (nahiz eta erakuslean fitxategiaren amaieran egon ez bigizta eten egingo da). `Maite` gakoaren bilaketa arrakastatsua izan delako fitxategiaren elementuari dagokion posizioa itzuliko du `BilaketaEgin` funtzioak, hots, fitxategiaren erakusleak duen uneko posizioaren aurrekoa.

Arrakastatsua ez den bilaketa azaltzeko `Leire` izena duen ikaslearen datuak pantailan erakutsiko ditugu `GELA_1.DAT` fitxategitik hartuta. `GELA_1.DAT` fitxategiaren edukiari begirada bat emanez `Leire` izeneko ikaslerik ez dagoela konturatzen gara, beraz `BilaketaFitxategietan` programaren irteera hau izango litzateke:

```
Fitxategiaren izena eman: GELA_1.DAT
Ikaslearen izen deiturak eman: Leire
Leire izeneko ikaslerik ez dago GELA_1.DAT fitxategian
_
```

`Leire` datua fitxategian aurkitu ez arren `BilaketaEgin` funtzio barneko bigiztak amaierarik badu, izan ere azken elementuaren irakurketa egitean (`Ane/2/4.7` balioak memoriara eramatean) fitxategiaren erakuslea `4` posizioan geratzen da, datuen zonaldetik kanpo alegia. Ondorioz, `Eof` funtzioak `TRUE` balioko du eta `WHILE-DO` bigizta moztuko da (nahiz eta `Aurkitua` aldagaia, hasieraketan bezala, `FALSE` izan). Eskematikoki:



Bigarren exekuzio honetan, `BilaketaEgin` funtzioak `-1` itzultzen dio modulu deitzaileari `Leire` gakoaren bilaketa arrakastatsua ez delako izan.

12.1.4.5 Gehiketa

Demagun erregistroak biltegitzen dituen `GELA_1.DAT` fitxategiaren edukia hau dela eta beste ikasle berri baten datuak erregistratu nahi ditugula fitxategia elementu batez handituz.

Gehiketa baino lehen:

GELA_1.DAT

Jokin	Pedro	Maite	Ane	
3	1	1	2	
6.5	8.6	7.3	4.7	
0	1	2	3	4

Eta gehiketa ondoren:

GELA_1.DAT

Jokin	Pedro	Maite	Ane	Luis	
3	1	1	2	1	
6.5	8.6	7.3	4.7	5.9	
0	1	2	3	4	5

Gehiketa fitxategiaren bukaeran egiten denez `GELA_1.DAT` ireki ondoren erakuslea `Seek(F,Filesize(F))` bitartez mugituko da idazketa toki egokian burutu ahal izateko, `GehiketaFitxategietan` programa hau izango litzateke:

```
PROGRAM GehiketaFitxategietan ;                  { \TP70\12\FILE_18.PAS }
TYPE
  DM_Katea = String [49] ;
  DM_Fitxa = RECORD
               IzenDeiturak : DM_Katea ;
               Deialdia : Byte ;
               Nota : Real ;
             END ;
  DM_Fitxategi = FILE OF DM_Fitxa ;

PROCEDURE DatuakBildu (VAR Elementua : DM_Fitxa) ;
BEGIN
   Write ('Ikaslearen izen deiturak eman: ') ;
   ReadLn (Elementua.IzenDeiturak) ;
   Write ('Deialdia eman: ') ;
   ReadLn (Elementua.Deialdia) ;
   Write ('Nota eman: ') ;
   ReadLn (Elementua.Nota) ;
END ;

PROCEDURE Gehiketa (FitxFisiko : String) ;
VAR
   F : DM_Fitxategi ;
   Elem : DM_Fitxa
BEGIN
   Assign (F, FitxFisiko) ;
   Reset (F) ;                                   { Irakurketarako ireki }

   Seek (F, FileSize(F)) ;                       { Bukaeran kokatu }
   DatuakBildu (Elem) ;
   Write (F, Elem) ;                             { Fitxategian idatzi }

   Close (F) ;                                   { Fitxategia itxi }
END ;
```

```
VAR
   FitxIzen : String ;
BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   Gehiketa (FitxIzen) ;
END.
```

`GehiketaFitxategietan` programa horrek gabezi bi ditu, batetik exekuzio bakoitzeko elementu bakarra gehitzeko ahalmena duela, eta, bestetik fitxategiak lehendik duen informazioa mantendu nahi denez fitxategia irekitzerakoan behartuta gaude `Reset` aplikatzera (fitxategia existitzen dela suposatzen ari gara `GehiketaFitxategietan` programan). Jarraian ematen den programan nahi beste ikasleren datuak erregistratuko dira, eta gainera fitxategiaren existentzia kontrolatzen da:

```
PROGRAM GehiketakFitxategietan ;                { \TP70\12\FILE_19.PAS }
USES
  Crt ;

TYPE
  DM_Katea = String [49] ;
  DM_Fitxa = RECORD
               IzenDeiturak : DM_Katea ;
               Deialdia : Byte ;
               Nota : Real ;
             END ;
  DM_Fitxategi = FILE OF DM_Fitxa ;


PROCEDURE DatuakBildu (VAR Elementua : DM_Fitxa) ;
BEGIN
   Write ('Ikaslearen izen deiturak eman: ') ;
   ReadLn (Elementua.IzenDeiturak) ;
   Write ('Deialdia eman: ') ;
   ReadLn (Elementua.Deialdia) ;
   Write ('Nota eman: ') ;
   ReadLn (Elementua.Nota) ;
END ;

PROCEDURE Gehiketak (FitxFisiko : String) ;
VAR
   F : DM_Fitxategi ;
   Elem : DM_Fitxa ;
   Erantz : Char ;
BEGIN
   Assign (F, FitxFisiko) ;
{$I-}
   Reset (F) ;                                  { Irakurketarako ireki }
{$I+}
   IF IOresult = 0 THEN
     Seek (F, FileSize(F))                      { Fitxategiaren amaierara }
   ELSE
     Rewrite (F) ;                              { Fitxategia sortu }

   REPEAT
      DatuakBildu (Elem) ;
      Write (F, Elem) ;                         { Fitxategian idatzi }

      Write ('Gehiagorik? (B/E)  ') ;
      Erantz := ReadKey ;
      Erantz := UpCase (Erantz) ;
      WriteLn (Erantz) ;
   UNTIL Erantz = 'E' ;

   Close (F) ;                                  { Fitxategia itxi }
END ;
```

```
VAR
   FitxIzen : String ;
BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   Gehiketak (FitxIzen) ;
END.
```

Ikasle bat baino gehiagoren datuak fitxategian gordetzeak ez duenez zailtasunik suposatzen ez gara hortaz luzatuko, baina `Gehiketak` errutinan agertzen diren irekitze prozedura biren (`Reset` eta `Rewrite`) azalpena interesgarria iruditzen zaigu.

Hasteko, kontura gaitezen irekitze prozedura biak ez direla exekuzio jakin batetan burutzen, `Reset` egitea lortzen bada ez da `Rewrite` prozedura aktibatzen, eta alderantziz, fitxategia existitzen ez delako `Reset` prozedurak huts egiten badu `Rewrite` exekutatzen da:

```
{$I-}
   Reset (F) ;                                  { Irekitzen saiatu }
{$I+}
   IF IOresult = 0 THEN
     Seek (F, FileSize(F))                      { Fitxategiaren amaierara }
   ELSE
     Rewrite (F) ;                              { Fitxategia sortu }
```

12.1.4.6 Aldaketa

12.1.4.3 Ibilera izeneko puntuan `GELA_1.DAT` fitxategiaren ikasle guztien deialdia inkrementatu eta aldaketa diskoan erregistratzen zen. Prozesaketaren muina laburbilduz: datua memoriara igarotzeko irakurri beharra zegoen, baina inkrementua egin ondoren fitxategian aldaketa gauzatzeko erakuslea atzeratu behar zen idatzi baino lehenago.

Ondoren eransten den programan ikasle baten izena teklatuz hartzen da, hura fitxategian bilatu eta aurkituz gero bere nota aldatzeko aukera izango dugu, bilaketak huts egitean mezu batez adierazten zaio erabiltzaileari.

Hona hemen `AldaketaFitxategietan` programa eta balizko irteera bat:

```
PROGRAM AldaketaFitxategietan ;                 { \TP70\12\FILE_20.PAS }
TYPE
  DM_Katea = String [49] ;
  DM_Fitxa = RECORD
               IzenDeiturak : DM_Katea ;
               Deialdia : Byte ;
               Nota : Real ;
             END ;
  DM_Fitxategi = FILE OF DM_Fitxa ;

FUNCTION Bilaketa (FitxFisiko : String; Gakoa : DM_Katea) : LongInt ;
VAR
   F : DM_Fitxategi ;
   Elem : DM_Fitxa ;
   Aurkitua : Boolean ;
BEGIN
   Assign (F, FitxFisiko) ;
   Reset (F) ;                                  { Irakurketarako ireki, }
                                                { fitxategiaren existentzia }
                                                { ez da lehenago frogatu }
```

```
   Aurkitua := FALSE ;
   WHILE (NOT Eof (F)) AND (NOT Aurkitua) DO
   BEGIN
      Read (F, Elem) ;                          { Fitxategitik irakurri }
      IF Elem.IzenDeiturak = Gakoa THEN
        Aurkitua := TRUE ;
   END ;
   IF Aurkitua THEN
     Bilaketa := FilePos(F) - 1
   ELSE
     Bilaketa := -1 ;
   Close (F) ;                                  { Fitxategia itxi }
END ;

PROCEDURE DagoenaIkusi (CONST Elem: DM_Fitxa) ;
BEGIN
   Write ('Ikaslea:':13, Elem.IzenDeiturak:49) ;
   Write (Elem.Deialdia:5) ;
   WriteLn (Elem.Nota:8:2) ;
END ;

PROCEDURE DatuaBildu (VAR Elementua : DM_Fitxa) ;
BEGIN
   Write (Elementua.IzenDeiturak,' ikaslearen nota berria eman: ') ;
   ReadLn (Elementua.Nota) ;
END ;

PROCEDURE Aldaketa (FitxFisiko : String; Posizioa : LongInt) ;
VAR
   F : DM_Fitxategi ;
   Elem : DM_Fitxa ;
BEGIN
   Assign (F, FitxFisiko) ;
   Reset (F) ;                                  { Irakurketarako ireki }

   Seek (F, Posizioa) ;
   Read (F, Elem) ;
   DagoenaIkusi (Elem) ;

   DatuaBildu (Elem) ;
   Seek (F, Posizioa) ;
   Write (F, Elem) ;                            { Fitxategian idatzi }

   Close (F) ;                                  { Fitxategia itxi }
END ;


VAR
   FitxIzen : String ;
   Gakoa : DM_Katea ;
   Posizioa : LongInt ;
BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   Write ('Aldatu nahi den ikaslearen izen deiturak eman: ') ;
   ReadLn (Gakoa) ;
   Posizioa := Bilaketa (FitxIzen, Gakoa) ;
   IF Posizioa = -1 THEN
     WriteLn (Gakoa, ' izeneko ikaslerik ez dago ', FitxIzen, ' fitxategian')
   ELSE
     Aldaketa (FitxIzen, Posizioa) ;
END.
```

```
Fitxategiaren izena eman: GELA_1.DAT
Aldatu nahi den ikaslearen izen deiturak eman: Pedro
     Ikaslea:                                        Pedro     1    8.60
Pedro ikaslearen nota berria eman: 9.1
_
```

12.1.4.7 Tartekaketa

Erregistroak biltegitzen dituen `A:\TP70\12\GELA_2.DAT` fitxategiaren[13] edukia hau izanik, `2` posizioan ikasle berri baten datuak erregistratu nahi dira (lehendik dauzkagun datuak galdu gabe noski).

Tartekaketa baino lehen:

`A:\TP70\12\GELA_2.DAT`

Garcia	Garat	Gondra	Goñi	
2	3	1	1	
4.6	7.1	2.5	6.9	
0	1	2	3	4

Eta tartekaketa ondoren:

`A:\TP70\12\GELA_2.DAT`

Garcia	Garat	Gomez	Gondra	Goñi	
2	3	4	1	1	
4.6	7.1	8.3	2.5	6.9	
0	1	2	3	4	5

Eginkizun hau burutzeko bi algoritmo oso ezberdinak aurkeztuko ditugu. Aurrenekoan `GELA_2.DAT` fitxategi bakarraren gainean lan egiten da, eta, bigarrenean `GELA_2.DAT` prozesatzeko fitxategi lagungarri bat erabiltzen da. Algoritmo bakoitzeko programa bana idatzi izan dugu `TartekaketaFitxategietan1` eta `TartekaketaFitxategietan2` izenekoak, programak funtsean berdinak direnez amankomunak dituzten kontzeptuak batera azalduko ditugu `Tartekaketa1` eta `Tartekaketa2` prozedurei arreta berezia eskainiz.

`TartekaketaFitxategietan1` eta `TartekaketaFitxategietan2` programetan ikasle berria `A:\TP70\12\GELA_2.DAT` fitxategiaren zein posiziotan kokatuko den zehazten da funtzio berezi batean, balizko posizioaren goimuga fitxategiaren tamainak jartzen du (posizioaren behemuga, dakigunez, `0` izango da). Ikasle berriaren datuak non gordeko diren ezaguna denean `Tartekaketa1` edo `Tartekaketa2` prozedura aktibatzen da, honek beharko dituen parametroak fitxategiaren izena eta ikasle berriaren posizioa izango dira (ikaslearen datuak `Tartekaketa1` edo `Tartekaketa2` prozedura barnean lortuko dira beste errutina bati deituz).

`Tartekaketa1` prozedura

Ikasle berriaren informazioa zehaztutako fitxategiaren posizioan gorde aurretik fitxategiak duen edukia "prestatu" behar da. Irudi honek adierazten duen egoera lortu behar da, non `2` posizioan datu berriak kokatuko direnez fitxategiaren erregistro horren informazioa errepikaturik dagoen:

`A:\TP70\12\GELA_2.DAT`

Garcia	Garat	Gondra	Gondra	Goñi	
2	3	1	1	1	
4.6	7.1	2.5	2.5	6.9	
0	1	2	3	4	5

Fitxategia horrela prestaturik dagoenean ikasle berriaren datuak teklatutik jaso eta fitxategian idaztea besterik ez zen beharko.

[13] `GELA_2.DAT` fitxategia `A` diskoaren errotik eskegita dagoen `TP70\12` direktorioan aurkitzen da. Landuko den fitxategiaren izena eta bidea teklatuz ondo emango dira, bestela erroreen literalak hauek lirateke:

1. Fitxategian ez dagoenean: `Error 2: File not found.`
2. Direktorioa ez dagoenean: `Error 3: Path not found.`

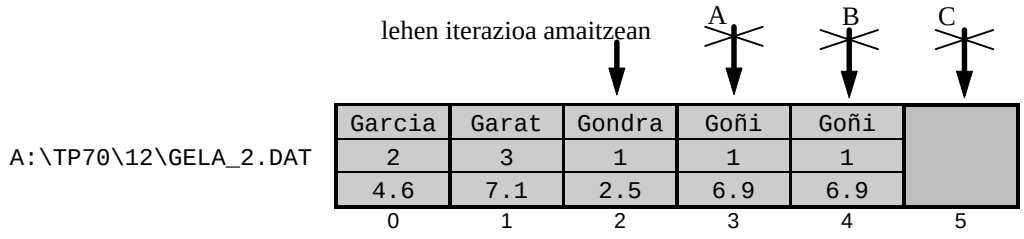
Hau da `TartekaketaFitxategietan1` programa:

```
PROGRAM TartekaketaFitxategietan1 ;              { \TP70\12\FILE_21.PAS }
TYPE
  DM_Katea = String [49] ;
  DM_Fitxa = RECORD
               IzenDeiturak : DM_Katea ;
               Deialdia : Byte ;
               Nota : Real ;
             END ;
  DM_Fitxategi = FILE OF DM_Fitxa ;


FUNCTION PosizioaZehaztu (FitxFisiko : String) : LongInt ;
VAR
   F : DM_Fitxategi ;
   Non : LongInt ;
BEGIN
   Assign (F, FitxFisiko) ;
   Reset (F) ;                                   { Irakurketarako ireki }

   REPEAT
      Write ('Elementu berria zer posiziotan?  (0..', FileSize(F)-1, ')  ') ;
      ReadLn (Non) ;
   UNTIL (Non >= 0) AND (Non <= FileSize(F)-1) ;

   PosizioaZehaztu := Non ;
   Close (F) ;                                   { Fitxategia itxi }
END ;


PROCEDURE DatuakBildu (VAR Elementua : DM_Fitxa) ;
BEGIN
   Write ('Ikaslearen izen deiturak eman: ') ;
   ReadLn (Elementua.IzenDeiturak) ;
   Write ('Deialdia eman: ') ;
   ReadLn (Elementua.Deialdia) ;
   Write ('Nota eman: ') ;
   ReadLn (Elementua.Nota) ;
END ;


PROCEDURE Tartekaketa1 (FitxFisiko : String; Posizioa : LongInt) ;
VAR
   F : DM_Fitxategi ;
   Elem : DM_Fitxa ;
   Kont, Azkena : LongInt ;
BEGIN
   Assign (F, FitxFisiko) ;
   Reset (F) ;                                   { Irakurketarako ireki }

   Azkena := FileSize(F) - 1 ;
   Seek (F, Azkena) ;

   FOR Kont:=Azkena DOWNTO Posizioa DO
   BEGIN
      Read (F, Elem) ;                           { Fitxategitik irakurri }
      Write (F, Elem) ;                          { Fitxategian idatzi }
      Seek (F, FilePos(F)-3) ;                   { Erakuslea atzeratu }
   END ;

   DatuakBildu (Elem) ;
   Seek (F, Posizioa) ;
   Write (F, Elem) ;                             { Ikasle berria idatzi }

   Close (F) ;                                   { Fitxategia itxi }
END ;
```

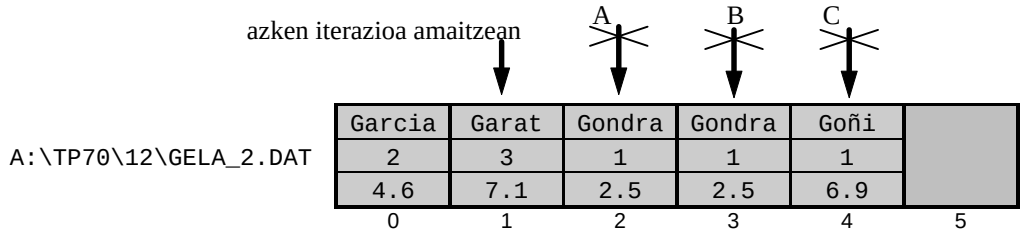
```
VAR
   FitxIzen : String ;
   NonSartu : LongInt ;
BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;

   NonSartu := PosizioaZehaztu (FitxIzen) ;
   Tartekaketa1 (FitxIzen, NonSartu) ;
END.
```

`Tartekaketa1` prozedurak duen punturik zailena fitxategiaren prestatze lana da, ikus daitekeenez `GELA_2.DAT` izena duen fitxategiaren azken elementuan kokatzen da erakuslea eta `FOR-DOWNTO-DO` bigizta bat hasten da, edozein iteraziotan elementu bat irakurri eta jarraian idazten da (aipatzekoa da iterazio bakoitzeko hiru elementuko jauzia egiten dela atzerantz).

Lehen iterazioa. Hasieran erakuslea `3` posizioan dago (A erakuslea), eta irakurketa egin ondoren `4` posiziora joango da automatikoki (B erakuslea). Erakuslea `4` toki horretan dagoela idazketa egiten da (`Goñi/1/6.9` balioak fitxategian errepikatuz) eta erakuslea `5` posizioan kokatuz. Lehendabiziko iterazioa bukatu aurretik erakuslea hiru posizioz atzeratzen da `5`-tik `2`-ra higituz:



Azken iterazioa. Adibidean bigarren iterazioa azkena da; erakuslea hasieran `2` posizioan dago (A erakuslea), eta irakurketa egin ondoren `3`-ra joango da (B erakuslea). Erakuslea `3`-an dagoela idazketa egiten da (`Gondra/1/2.5` balioak fitxategian errepikatuz) eta erakuslea `4` posiziora higituz. Bigarren iterazioa bukatu aurretik erakuslea hiru posizioz atzeratzen da `4`-tik `1`-ra higituz:



Horrekin fitxategiaren prestatze lana bukatutzat eman daiteke

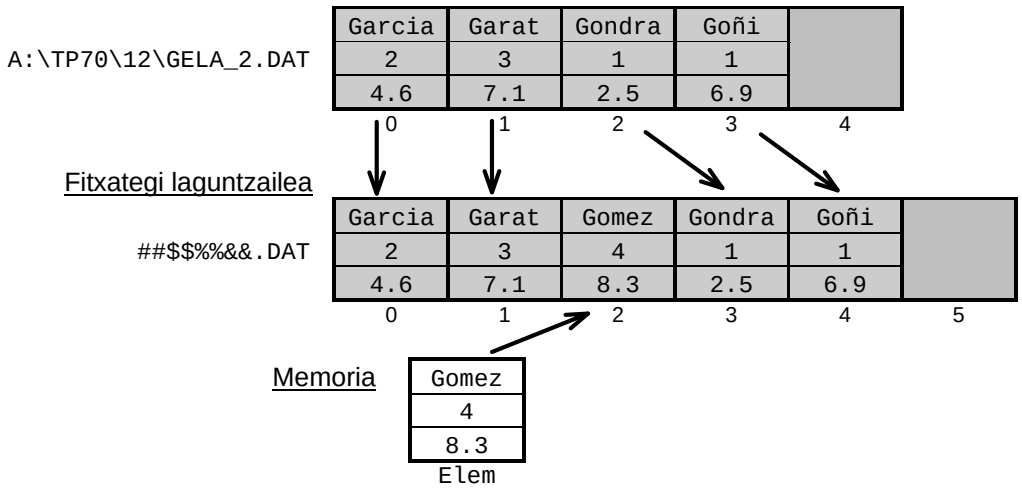
`Tartekaketa1` prozeduraren hurrengo zeregina erraza da, ikasle berriaren datuak `Gomez/ /4/8.4` teklatuz jaso, eta erakuslea programa nagusitik pasatu zaion posiziora eraman (adibidean, dagoen posiziotik `2`-ra), amaitzeko datu berriak `GELA_2.DAT` fitxategian idatzi.

```
   DatuakBildu (Elem) ;       { Teklatutik irakurri }
   Seek (L, Posizioa) ;       { Erakuslea kokatu }
   Write (L, Elem) ;          { Ikasle berria idatzi }
```

Tartekaketa2 prozedura

Ikasle berriaren informazioa fitxategian tartekatzeko, bigarren fitxategi laguntzaile batean oinarrituko da errutina honen algoritmoa. Ikaslearen datuak fitxategian non gordeko diren ezaguna da Tartekaketa2 prozeduran Posizioa izeneko parametroari esker, horregatik GELA_2.DAT fitxategiaren edukia bi zatitan bana daiteke: 0-tik Posizioa-1 bitartera hasierako tartea, eta bigarren tartea Posizioa-tik fitxategia amaitu arte.

0-tik Posizioa-1 bitarteko tartean elementuak, banan banan, GELA_2.DAT fitxategitik irakurri eta fitxategi languntzailean idatziko dira (eginkizun hau kontrolatzeko egiturarik zehatzena FOR-TO-DO bigizta da). Bigarren tartearekin jarraitu aurretik, erabiltzaileak teklatuz emango ditu ikasle berriaren datuak eta fitxategi laguntzailearen Posizioa tokian idazten da. Ondoren, GELA_2.DAT fitxategiaren bigarren zatia prozesatuko da, bertatik irakurri eta fitxategi languntzailean idatziz (hau kontrolatzeko berriz hobe da WHILE-DO bigizta formulatzea).



Beraz, fitxategi laguntzaileak jatorrizko fitxategiak baino osagai bat gehiago izango du. Beste era batera esanda hobe litzateke hasierako GELA_2.DAT fitxategia ezabatzea eta laguntzaileari bere izena esleitzea.

```
PROGRAM TartekaketaFitxategietan2 ;              { \TP70\12\FILE_22.PAS }
TYPE
  DM_Katea = String [49] ;
  DM_Fitxa = RECORD
               IzenDeiturak : DM_Katea ;
               Deialdia : Byte ;
               Nota : Real ;
             END ;
  DM_Fitxategi = FILE OF DM_Fitxa ;

FUNCTION PosizioaZehaztu (FitxFisiko : String) : LongInt ;
VAR
   F : DM_Fitxategi ;
   Non : LongInt ;
BEGIN
   Assign (F, FitxFisiko) ;
   Reset (F) ;                                   { Irakurketarako ireki }
   REPEAT
      Write ('Elementu berria zer posiziotan?  (0..', FileSize(F)-1, ')  ') ;
      ReadLn (Non) ;
   UNTIL (Non >= 0) AND (Non <= FileSize(F)-1) ;
   PosizioaZehaztu := Non ;
   Close (F) ;                                   { Fitxategia itxi }
END ;
```

```
PROCEDURE DatuakBildu (VAR Elementua : DM_Fitxa) ;
BEGIN
   Write ('Ikaslearen izen deiturak eman: ') ;
   ReadLn (Elementua.IzenDeiturak) ;
   Write ('Deialdia eman: ') ;
   ReadLn (Elementua.Deialdia) ;
   Write ('Nota eman: ') ;
   ReadLn (Elementua.Nota) ;
END ;


PROCEDURE Tartekaketa2 (FitxFisiko : String; Posizioa : LongInt) ;
CONST
   LAGUNTZAILEA = '##$$%%&&.DAT' ;
VAR
   F, L : DM_Fitxategi ;
   Elem : DM_Fitxa ;
   Kont : LongInt ;
BEGIN
   Assign (F, FitxFisiko) ;
   Reset (F) ;                                  { Irakurketarako ireki }
   Assign (L, LAGUNTZAILEA) ;
   Rewrite (L) ;                                { Idazketarako ireki }

   FOR Kont:=0 TO Posizioa-1 DO
   BEGIN
      Read (F, Elem) ;                          { Fitxategitik irakurri }
      Write (L, Elem) ;                         { Fitxategian idatzi }
   END ;

   DatuakBildu (Elem) ;
   Seek (L, Posizioa) ;                         { Soberan egon daiteke }
   Write (L, Elem) ;                            { Ikasle berria idatzi }

   WHILE NOT Eof(F) DO
   BEGIN
      Read (F, Elem) ;                          { Fitxategitik irakurri }
      Write (L, Elem) ;                         { Fitxategian idatzi }
   END ;

   Close (F) ;                                  { Fitxategia itxi }
   Close (L) ;                                  { Fitxategia itxi }

   Erase (F) ;                                  { bat, fitxategia ezabatu }
   Rename (L, FitxFisiko) ;                     { bi, izena aldatu }
END ;

VAR                                             { Programa nagusia }
   FitxIzen : String ;
   NonSartu : LongInt ;
BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;

   NonSartu := PosizioaZehaztu (FitxIzen) ;
   Tartekaketa2 (FitxIzen, NonSartu) ;
END.
```

`Tartekaketa2` prozedurari buruzko zenbait galdera bururatzen zaigu. Esate baterako, fitxategi laguntzailearen izena aukeratzean zergatik hautatu da horren izen arraro eta zaila?. Zergatik dago soberan `Seek(L,Posizioa)` sententzia?. `Tartekaketa2` errutinaren azken sententzia biak (`Erase` eta `Rename`) ordenez trukatzean `005` errorea[14] gertatzen da, zergatik?

[14] `5` erroreari dagokion literala: `Error 5: File access denied.`

12.1.4.8 Ezabaketa

Ezabaketa eragiketa tartekaketaren aurkakoa kontsidera daiteke. Ezabatu beharreko elementua fitxategiaren azkenekoa bada `Truncate` prozedura aplikatuko dugu (ikus **12.1.2.2.8** puntua), baina ezabatu behar den elementua azkena ez denean `Truncate` bitartez informazioa galduko litzatekeelako beste modu batean egingo dugu.

`A:\TP70\12\GELA_2.DAT` fitxategiak bost ikasleren informazioa biltegitzen duela suposatuz, `1` posizioan dagoen ikaslearen datuak diskotik ezabatu nahi dira.

Ezabaketa baino lehen:

`A:\TP70\12\GELA_2.DAT`

Garcia	Garat	Gomez	Gondra	Goñi	
2	3	4	1	1	
4.6	7.1	8.3	2.5	6.9	
0	1	2	3	4	5

Eta ezabaketa ondoren:

`A:\TP70\12\GELA_2.DAT`

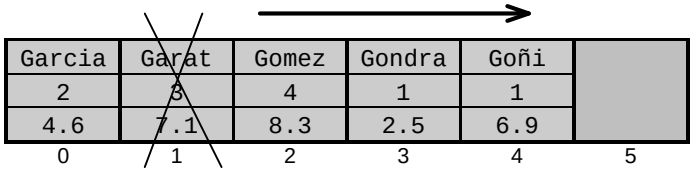
Garcia	Gomez	Gondra	Goñi	
2	4	1	1	
4.6	8.3	2.5	6.9	
0	1	2	3	4

Lehen bezala ezabaketarako ezberdinak diren bi algoritmo aurkeztuko ditugu. Batean `GELA_2.DAT` fitxategia bakar-bakarrik lantzen da, eta, bestean bigarren fitxategi laguntzaile bat erabiltzen da. Algoritmo bakoitzari dagokion programa idatzi dugu `EzabaketaFitxategietan1` eta `EzabaketaFitxategietan2`, asma daitekeenez programa biren arteko ezberdintasuna `Ezabaketa1` eta `Ezabaketa2` prozeduretan datza.

`EzabaketaFitxategietan1` eta `EzabaketaFitxategietan2` programetan fitxategitik desagertuko den ikasleren posizioa zehazten da funtzio berezi batez, balizko posizioaren goimuga fitxategiaren tamainak jartzen du (behemuga, halabeharrez, `0` izango da). Ezabatuko den ikaslearen posizioa ezaguna denean `Ezabaketa1` edo `Ezabaketa2` prozedura aktibatzen da.

`Ezabaketa1` prozedura

Fitxategiak bost osagai baditu eta ezabatu behar den ikaslea `1` posizioan aurkitzen bada, prozesu errepikakorrak `2`-tik `4`-ra iraungo du. Hots, ezabatu behar den elementuaren hurrengo posiziotik fitxategia bukatu arte:



`A:\TP70\12\GELA_2.DAT`

Garcia	Garat	Gomez	Gondra	Goñi	
2	3	4	1	1	
4.6	7.1	8.3	2.5	6.9	
0	1	2	3	4	5

`2`-tik `4` bitarteko tartean, elementuak irakurri eta berriro idatziko dira posizio bat aurrerago `GELA_2.DAT` fitxategian (`Gomez` ikaslea `2` posiziotik irakurri eta `1` posizioan idatzi). Prozesu errepikakor hau kontrolatzeko `FOR-TO-DO` egitura aplika daiteke baina egokiago eta errazago iruditzen zaigu `WHILE-DO` bigiztaz programatzea.

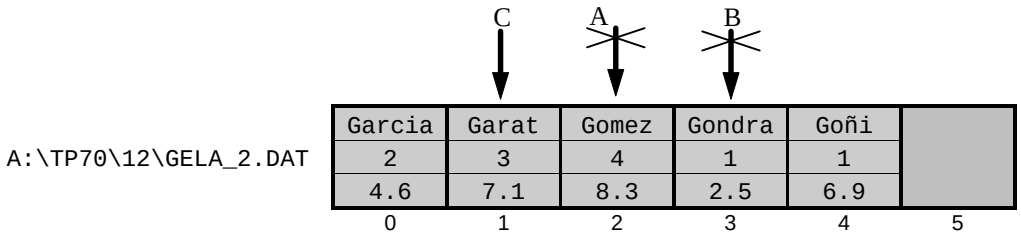
Hau da `EzabaketaFitxategietan1` programa:

```
PROGRAM EzabaketaFitxategietan1 ;                { \TP70\12\FILE_23.PAS }
TYPE
  DM_Katea = String [49] ;
  DM_Fitxa = RECORD
               IzenDeiturak : DM_Katea ;
               Deialdia : Byte ;
               Nota : Real ;
             END ;
  DM_Fitxategi = FILE OF DM_Fitxa ;


FUNCTION PosizioaZehaztu (FitxFisiko : String) : LongInt ;
VAR
   F : DM_Fitxategi ;
   Non : LongInt ;
BEGIN
   Assign (F, FitxFisiko) ;
   Reset (F) ;                                   { Irakurketarako ireki }

   REPEAT
      Write ('Zein da ezabatu behar den elementuaren posizioa?  (0..',) ;
      Write (FileSize(F)-1, ')  ') ;
      ReadLn (Non) ;
   UNTIL (Non >= 0) AND (Non <= FileSize(F)-1) ;

   PosizioaZehaztu := Non ;
   Close (F) ;                                   { Fitxategia itxi }
END ;


PROCEDURE Ezabaketa1 (FitxFisiko : String; Posizioa : LongInt) ;
VAR
   F : DM_Fitxategi ;
   Elem : DM_Fitxa ;
BEGIN
   Assign (F, FitxFisiko) ;
   Reset (F) ;                                   { Irakurketarako ireki }

   Seek (F, Posizioa+1) ;                        { Erakuslea kokatu }

   WHILE NOT Eof(F) DO
   BEGIN
      Read (F, Elem) ;                           { Fitxategitik irakurri }
      Seek (F, FilePos(F)-2) ;                   { Erakuslea atzeratu }
      Write (F, Elem) ;                          { Fitxategian idatzi }
      Seek (F, FilePos(F)+1) ;                   { Erakuslea aurreratu }
   END ;

   Seek (F, FileSize(F)-1) ;                     { Errepikaturik geratzen }
   Truncate (F) ;                                { den azkena ezabatu }

   Close (F) ;                                   { Fitxategia itxi }
END ;


VAR
   FitxIzen : String ;
   ZeinKendu : LongInt ;
BEGIN
   Write ('Fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;

   ZeinKendu := PosizioaZehaztu (FitxIzen) ;
   Ezabaketa1 (FitxIzen, ZeinKendu) ;
END.
```

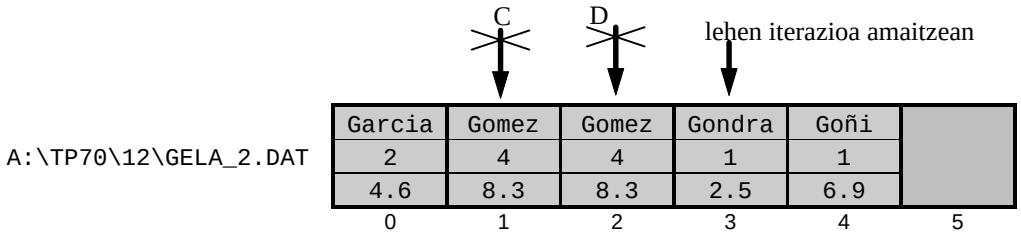
`Ezabaketa1` prozeduraren mamia `WHILE-DO` bigizta da, non iterazio bakoitzeko elementu bat bere aurreko posizioan berridazten da. Iterazio jakin batetik ateratzean prozesatu den elementua `GELA_2.DAT` fitxategian birritan erregistraturik geratzen da. Argi dagoenez `Posizioa` aldagaiak adierazten duen elementua ezabatzeko `WHILE-DO` bigiztaren exekuzio esparrua `Posizioa+1` elementuan hasi eta fitxategiaren amaieran bukatzen da, horregatik prozesu errepikakorran sartu aurretik erakuslea `Posizioa+1` elementuaren gainean kokatu beharra dago.

Lehen eta azken iterazioak azal ditzagun:

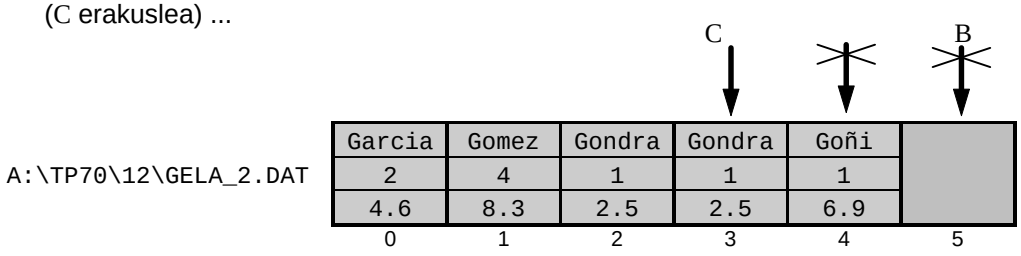
Lehen iterazioa. Hasieran erakuslea `2` posizioan dago (A erakuslea), eta irakurketa egin ondoren `3` posiziora joango da automatikoki (B erakuslea). Erakuslea `3` toki horretatik `1` posiziora eramaten da (C erakuslea), eta ...



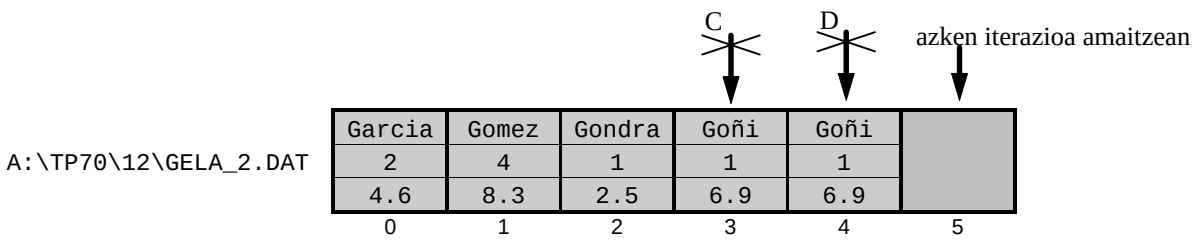
erakuslea C delarik idazketa egiten da (`Gomez/4/8.3` balioak fitxategian errepikatuz) eta erakuslea berriro `2` posizioan kokatzen da (D erakuslea). Lehendabiziko iterazioa bukatu aurretik erakuslea posizio bat gehiago aurreratzen da `2`-tik `3`-ra higituz:



Azken iterazioa. Adibidean `Gondra` errepikaturik egongo da fitxategian eta erakuslea `4` posizioan aurkituko da (A erakuslea), `Goñi/1/6.9` balioen irakurketa egin ondoren `5`-ra joango da (B erakuslea), eta ondoren `Seek` bitartez bi posizio atzeratuz `3`-an geratzen da (C erakuslea) ...



erakuslea C delarik idazketa egiten da (`Goñi/1/6.9` balioak fitxategian errepikaturik geratuko dira) eta erakuslea berriro `4` posizioan kokatzen da (D erakuslea), iterazioa honetan erakuslea posizio bat aurreratzen denez `Eof` funtzioak `TRUE` itzuliko du:



`Eof` funtzioaren `TRUE` itzuliko horrekin `WHILE-DO` prozesua bukatutzat eman daiteke, baina tamalez fitxategiaren azken elementua errepikaturik geratzen da. Horregatik `Ezabaketa1` prozeduratik irten aurretik `GELA_2.DAT` fitxategian `Truncate` aplikatzen da, dakigunez `Truncate` eraginkorra izan dadin fitxategiaren erakusleak azken elementua adieraziko du:

```
Seek (F, FileSize(F)-1) ;       { Errepikaturik geratzen }
Truncate (F) ;                  { den azkena ezabatu }
```

`Ezabaketa2` prozedura

Ezabatu nahi den ikaslearen datuak `Garat/3/7.1` izanik gogoratuz, beste era batean lor daitekeela frogatuko dugu. Tartekaketaren kasuan bezala bigarren fitxategi baten laguntzaz balia gaitezke fitxategi baten elementu jakin bat ezabatzeko, desagertuko den elementuaren posizioa ezaguna da programa nagusian `PosizioaZehaztu` funtzioari esker. `Ezabaketa2` prozedurak behar dituen parametroak bi dira, batetik fitxategiaren izena eta bestetik `PosizioaZehaztu` funtzioak itzultzen duen balioa (`FitxFisiko` eta `Posizioa` izeneko parametroak).

`Ezabaketa2` prozeduran `GELA_2.DAT` fitxategiaren edukia bi zatitan banatzen da: `0`-tik `Posizioa-1` bitartera hasierako tartea, eta bigarren tartea `Posizioa`-tik fitxategia amaitu arte.

Hau da `Ezabaketa2` prozedura darabilen `EzabaketaFitxategietan2` programaren zatia:

```
PROCEDURE Ezabaketa2 (FitxFisiko : String; Posizioa : LongInt) ;
CONST
   LAGUNTZAILEA = '$$$$%%%%.DAT' ;
VAR
   F, L : DM_Fitxategi ;
   Elem : DM_Fitxa ;
   Kont : LongInt ;
BEGIN
   Assign (F, FitxFisiko) ;
   Reset (F) ;                                  { Irakurketarako ireki }

   Assign (L, LAGUNTZAILEA) ;
   Rewrite (L) ;                                { Idazketarako ireki }

   FOR Kont:=0 TO Posizioa-1 DO                 { 1. zatia }
   BEGIN
      Read (F, Elem) ;                          { Fitxategitik irakurri }
      Write (L, Elem) ;                         { Fitxategian idatzi }
   END ;

   Seek (F, Posizioa+1) ;                       { Ikaslea sahiestu }

   WHILE NOT Eof(F) DO                          { 2. zatia }
   BEGIN
      Read (F, Elem) ;                          { Fitxategitik irakurri }
      Write (L, Elem) ;                         { Fitxategian idatzi }
   END ;

   Close (F) ;                                  { Fitxategia itxi }
   Close (L) ;                                  { Fitxategia itxi }

   Erase (F) ;                                  { Fitxategia ezabatu }
   Rename (L, FitxFisiko) ;                     { Izena aldatu }
END ;
```

12.1.4.9 Fitxategi/Array

Eragiketa hau ibilera mota bat da eta ez du **12.1.4.3 Ibilera** puntuan ikusitakoarekin alde larregirik, orain datuak fitxategitik irakurri eta memorian aurkitzen den array batean kokatuko dira.

`FitxategitikArrayera` adibide-programa ikus dezagun non errutinarik garrantzitsuena `FitxategiArray` den:

```
PROGRAM FitxategitikArrayera ;                    { \TP70\12\FILE_25.PAS }
USES
  Crt ;
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 40 ;
TYPE
  DM_Katea = String [49] ;
  DM_Fitxa = RECORD
               IzenDeiturak : DM_Katea ;
               Deialdia : Byte ;
               Nota : Real ;
             END ;
  DM_Fitxategi = FILE OF DM_Fitxa ;
  DM_Zerrenda = ARRAY[BEHEMUGA..GOIMUGA] OF DM_Fitxa ;


FUNCTION ErantzunaJaso : Char ;
VAR
  Erantz : Char ;
BEGIN
   REPEAT
      Erantz := ReadKey ;
      Erantz := UpCase (Erantz) ;
      WriteLn (Erantz) ;
   UNTIL (Erantz = 'B') OR (Erantz = 'E') ;
   ErantzunaJaso := Erantz ;
END ;


PROCEDURE FitxategiArray (VAR A : DM_Zerrenda; VAR Luzera : Integer ;
                          FitxFisiko : String) ;
VAR
  F : DM_Fitxategi ;
  Indize : Integer ;
BEGIN
   Assign (F, FitxFisiko) ;
   Reset (F) ;

   IF FileSize(F) > GOIMUGA THEN
   BEGIN
     WriteLn ('Fitxategiaren informazio guztia ezin da arrayean gorde') ;
     Luzera := 0 ;
   END
   ELSE
   BEGIN
      Indize := BEHEMUGA ;
      WHILE NOT Eof(F) DO
      BEGIN
         Read (F, A[Indize]) ;
         Indize := Indize + 1 ;
      END ;
   END ;
   Luzera := Indize - 1 ;

   Close (F) ;
END ;
```

```
PROCEDURE ArrayBatIkusi (CONST Gela : DM_Zerrenda; VAR Luzera : Integer) ;
VAR
  Kont : Integer ;
BEGIN
   FOR Kont:=BEHEMUGA TO Luzera DO
   BEGIN
      Write (Kont, '. ikaslea:  ') ;
      Write (Gela[Kont].IzenDeiturak:49) ;
      Write (Gela[Kont].Deialdia:5) ;
      WriteLn (Gela[Kont].Nota:8:2) ;
   END ;
END;


VAR
  GelaArray : DM_Zerrenda ;
  LuzeraLogiko : Integer ;
  Erantzuna : Char ;
  FitxIzen : String ;
BEGIN
   Write ('INFORM.DAT fitxategiaren informazioa arrayera igaro? (B/E) ') ;
   Erantzuna := ErantzunaJaso ;
   IF Erantzuna = 'B' THEN
   BEGIN
      FitxIzen := 'INFORM.DAT' ;
      FitxategiArray (GelaArray, LuzeraLogiko, FitxIzen) ;
   END ;

   WriteLn ('Arrayaren edukia: ') ;
   ArrayBatIkusi (GelaArray, LuzeraLogiko) ;
END.
```

`FitxategiArray` prozeduran `INFORM.DAT` fitxategitik irakurtzean ez da ohi denez `Elem` aldagairik behar, bere papera `A[Indize]`-k joka baitezake. Bestalde, `INFORM.DAT` fitxategiaren elementuen kopurua arrayaren dimentsio maximoa baina handiago bada prozesua ez hastea erabaki da `FitxategiArray` prozeduran.

Amaitzeko galdera bat, nolako aldaketa egin beharko litzateke `Luzera` aldagaia prozeduratik kendu ahal izateko?.

12.1.4.10 Array/Fitxategi

Array batean daukagun informazioa fitxategi batera eramateko hona hemen prozedura:

```
PROCEDURE ArrayFitxategi (CONST A : DM_Zerrenda; Luzera : Integer ; FitxF : String) ;
VAR
   F : DM_Fitxategi ;
   Elem : DM_Fitxa ;
   Indize : Integer ;
BEGIN
   Assign (F, FitxF) ;
   Rewrite (F) ;

   FOR Indize:=BEHEMUGA TO Luzera DO
   BEGIN
      Elem := A[Indize] ;
      Write (F, Elem) ;
   END ;

   Close (F) ;
END ;
```

Prozedura honetan, lehen ez bezala, guztiz beharrezkoa da `Elem` aldagaia definitzea eta horren bitartez `INFORM.DAT` fitxategian idazketa burutzea. Izan ere, `Write(F, A[Indize])` prozeduran `A[Indize]` uneko parametroa onartezina da `A` parametroa `ArrayFitxategi` azpi-programan konstante-parametroa delako.

12.1.4.11 Ordenazioa fitxategi txikietan

Hamargarren kapituluko **10.4.6 Ordenazioa** puntuan arrayak ordenatzeko bost algoritmo erakutsi ziren. Teorian behintzat, algoritmo berberak ezar daitezke informazioa biltegitzen duen datu-egitura array bat izan ordez fitxategi bat denean, baina praktikan ez da horrelakorik egiten fitxategiak lantzean algoritmo horien konputazio-kostua oso handia delako.

Beraz bi bide izango dugu fitxategi baten informazioa ordenatu behar dugunean. Batetik, fitxategia txikia denean eta bere informazioa ordenadorearen memorian, arazorik gabe, gorde daitekeenean; halakoetan fitxategiaren edukia array batera igaroko da eta memorian dagoela arraya ordenatu egingo da, ondoren array ordenatua abiapuntuz harturik fitxategia berridatziko da. Eta bestetik fitxategia benetan handia delako arrayetan oinarritzerik ez dagoenean (ikus **12.1.4.12** puntua).

Demagun ordenatu gabeko `A:\TP70\12\GELA_2.DAT` fitxategi hau daukagula[15] eta noten arabera ordenatu nahi dela, notarik txikien duen ikasle lehena delarik.

Ordentau baino lehen:

`A:\TP70\12\GELA_2.DAT`

Garcia	Garat	Gomez	Gondra	Goñi	
2	3	4	1	1	
4.6	7.1	8.3	2.5	6.9	
0	1	2	3	4	5

Eta ordenatu ondoren:

`A:\TP70\12\GELA_2.DAT`

Gondra	Garcia	Goñi	Garat	Gomez	
1	2	1	3	4	
2.5	4.6	6.9	7.1	8.3	
0	1	2	3	4	5

Ostean ematen den `FitxategiTxikiaOrdenez` programan `GELA_2.DAT` fitxategiaren edukia programa barruan definitu den arrayean sar daitekeela suposatu dugu.

```
PROGRAM FitxategiTxikiaOrdenez ;                { \TP70\12\FILE_27.PAS }
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 40 ;
TYPE
  DM_Katea = String [49] ;
  DM_Fitxa = RECORD
               IzenDeiturak : DM_Katea ;
               Deialdia : Byte ;
               Nota : Real ;
             END ;
  DM_Fitxategi = FILE OF DM_Fitxa ;
  DM_Zerrenda = ARRAY[BEHEMUGA..GOIMUGA] OF DM_Fitxa ;
```

[15] Egia esan `A:\TP70\12\GELA_2.DAT` fitxategia izenei begira ordenaturik dago.

```
PROCEDURE FitxategiArray (VAR A : DM_Zerrenda; VAR Luzera : Integer ;
                          FitxFisiko : String) ;
VAR
   F : DM_Fitxategi ;
   Indize : Integer ;
BEGIN


END ;   { FitxategiArray prozeduraren amaiera}


{ Parametro bat irteerakoa den bitartean bestea sarrerakoa }
{ da. Horregatik A arraya erreferentziaz pasatzen da baina }
{ aldatzen ez den N luzera logikoari dagokion osoa balioz  }
PROCEDURE ArrayaAukeraketazOrdenatu (VAR A : DM_Zerrenda; N : Byte) ;
VAR
   Pos, j, k : Byte ;
   Min : Real ;
   Elem : DM_Fitxa ;
BEGIN
     FOR k:=1 TO N-1 DO                          { noten arabera ordetzen }
     BEGIN
        Elem := A[k] ;
        Min := A[k].Nota ;
        Pos := k ;
        FOR j:=k+1 TO N DO                       { ordenatu gabekoen artean }
        BEGIN
           IF Min > A[j].Nota THEN               { minimoa aurkitu }
           BEGIN
              Elem := A[j] ;
              Min := A[j].Nota ;
              Pos := j ;
           END ;
        END ;
        A[Pos] := A[k] ;                         { trukatu, tokiz aldatuz  }
        A[k] := Elem ;
     END;
END ;


PROCEDURE ArrayFitxategi (CONST A : DM_Zerrenda; Luzera : Integer ;
                          FitxFisiko : String) ;
VAR
   F : DM_Fitxategi ;
   Elem : DM_Fitxa ;
   Indize : Integer ;
BEGIN


END ;   { ArrayFitxategi prozeduraren amaiera}


VAR                                               { Programa Nagusia }
  GelaArray : DM_Zerrenda ;
  LuzeraLogiko : Integer ;
  Erantzuna : Char ;
  FitxIzen : String ;
BEGIN
   Write ('Noten arabera ordenatuko den fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   FitxategiArray (GelaArray, LuzeraLogiko, FitxIzen) ;
   ArrayaAukeraketazOrdenatu (GelaArray, LuzeraLogiko) ;
   ArrayFitxategi (GelaArray, LuzeraLogiko, FitxIzen) ;
END.
```

`FitxategiTxikiaOrdenez`-en programa nagusia irakurriz hauxe litzateke fitxategi bat ordenatu ahal izateko hiru urratsak: lehenik array batera igarotzen da gero arraya ordenatzeko, azkenean arrayaren informazio gaurkoratua fitxategira eramaten da.

12.1.4.12 Ordenazioa fitxategi handietan

Demagun noten arabera ordenatu gabeko A:\TP70\12\GELA_2.DAT fitxategia daukagula eta bere edukia noten arabera ordenatu nahi dela, baina array laguntzailerik erabili gabe.

GELA_2.DAT fitxategia ordenatu baino lehen eta ordenatu ondoren:

Ordenatu gabe:

A:\TP70\12\GELA_2.DAT

Garcia	Garat	Gomez	Gondra	Goñi	
2	3	4	1	1	
4.6	7.1	8.3	2.5	6.9	
0	1	2	3	4	5

Ordenaturik:

A:\TP70\12\GELA_2.DAT

Gondra	Garcia	Goñi	Garat	Gomez	
1	2	1	3	4	
2.5	4.6	6.9	7.1	8.3	
0	1	2	3	4	5

Hamargarren kapituluko **10.4.6 Ordenazioa** puntuan arrayak ordenatzeko ikusi ziren bost algoritmoetatik edozein har daiteke eta fitxategiekin lan egiteko moldatu, baina fitxategian irakurketa-idazketa asko behar direlako oso prozesu geldoa izango litzateke. Horregatik hobe da bigarren fitxategi laguntzaile batean oinarritzea GELA_2.DAT izena duen fitxategia ordenatu ahal izateko.

Fitxategi laguntzaileak jatorrizko GELA_2.DAT fitxategiak dituen eremuak izateaz gain beste berri bat ere edukiko du, elementu bat emaitza-fitxategira kopiatua izan den ala ez adierazten duen eremu boolearra (adibide-programan eremu osagarri horri Kopiatua deitu diogu).

Jarraian ematen den FitxategiHandiaOrdenez programan GELA_2.DAT fitxategiaren edukia programa barruan berriki sortuko den WWWMMM.DAT fitxategi laguntzailera transferitzen da, ahaztu fitxategi laguntzaileren elementuen Kopiatua eremua hasieran FALSE izango dela. Jatorrizko fitxategiaren edukia WWWMMM.DAT fitxategian bikoizturik dagoenean Rewrite(F) egin daiteke GELA_2.DAT fitxategiaren informazioa ordenaturik berreskuratzen hasteko.

FitxategiHandiaOrdenez programan GELA_2.DAT fitxategia ordenatzeko erabiltzen den algoritmoa benetan erraza da: fitxategi laguntzailean notarik txikiena duen ikaslea[16] lokalizatu ondoren bere datuak GELA_2.DAT emaitza-fitxategian kopiatzen dira, eta ikasle horri dagokion fitxategi laguntzaileko elementuari TRUE jartzen zaio Kopiatua eremuan.

```
PROGRAM FitxategiHandiaOrdenez ;                { \TP70\12\FILE_28.PAS }
TYPE
  DM_Katea = String [49] ;
  DM_Fitxa = RECORD
               IzenDeiturak : DM_Katea ;
               Deialdia : Byte ;
               Nota : Real ;
             END ;
  DM_Fitxategi = FILE OF DM_Fitxa ;

  DM_FitxaLag = RECORD
                  IzenDeiturak : DM_Katea ;
                  Deialdia : Byte ;
                  Nota : Real ;
                  Kopiatua : Boolean ;
                END ;
  DM_FitxategiLag = FILE OF DM_FitxaLag ;       { Fitxategi laguntzailea }
```

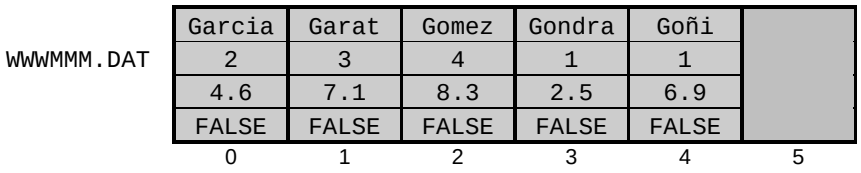
[16] Kopiatua eremua FALSE duten ikasleen arteko notarik txikiena duen ikaslea lokalizatu behar da.

```
FUNCTION TxikienaZehaztu (Iz : DM_Katea) : LongInt ;
VAR
   FLagun : DM_FitxategiLag ;
   Errg, Txiki : DM_FitxaLag ;
   Non : LongInt ;
   Aurkitua : Boolean ;
BEGIN
     Assign (FLagun,Iz) ;
     Reset (FLagun) ;

     { Lehen "ez kopiatua" bilatu eta txikiena dela suposatu }
     Aurkitua := FALSE ;
     WHILE (NOT Eof (FLagun)) AND (NOT Aurkitua) DO
     BEGIN
         Read (FLagun,Txiki) ;
         Non := Filepos(FLagun)-1 ;
         IF NOT Txiki.Kopiatua THEN Aurkitua := TRUE
     END ;

     IF NOT Aurkitua THEN Non := -1              { Ezinezko balioa }
     ELSE { Txikiena bilatu }
     BEGIN
        WHILE NOT Eof(FLagun) DO
        BEGIN
          Read (FLagun,Errg) ;
          IF (Errg.Kopiatua = FALSE) AND (Errg.Nota < Txiki.Nota) THEN
          BEGIN
             Non := Filepos(FLagun)-1 ;
             Txiki := Errg ;
          END ;
        END ;
     END ;
     TxikienaZehaztu := Non ;

     Close (FLagun) ;
END ;



PROCEDURE OrdenatuFitxategiz (FitxIzen : DM_Katea) ;
VAR
   F : DM_Fitxategi;
   FLagun : DM_FitxategiLag;
   Elem : DM_Fitxa;
   ElemLag : DM_FitxaLag;
   IzenLagun : DM_Katea ;
   TxikiPos : LongInt ;
BEGIN
   Assign (F, FitxIzen) ;
   Reset (F) ;

   { Fitxategi laguntzailea zehazteko }
   IzenLagun := 'WWWMMM.DAT' ;
   Assign (FLagun, IzenLagun) ;
   Rewrite (FLagun) ;

   { Kopiatu fitxategi berrian "Kopiatua" eremua FALSE ipiniz }
   WHILE NOT Eof(F) DO
   BEGIN
      Read (F,Elem) ;
      ElemLag.IzenDeiturak := Elem.IzenDeiturak ;
      ElemLag.Deialdia := Elem.Deialdia ;
      ElemLag.Nota := Elem.Nota ;
      ElemLag.Kopiatua := FALSE ;
      Write (FLagun,ElemLag) ;
   END ;
```

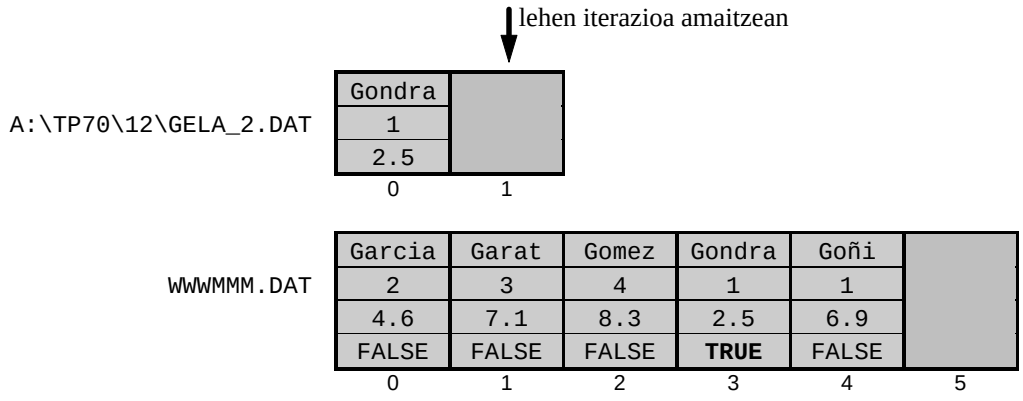
```
   { Fitxategian informazioa ordenaturik berridatzi }
   Rewrite (F) ;

   TxikiPos := TxikienaZehaztu (IzenLagun) ;
   WHILE TxikiPos <> -1 DO
   BEGIN
      Seek (FLagun,TxikiPos) ;
      Read (FLagun,ElemLag) ;
      Elem.IzenDeiturak := ElemLag.IzenDeiturak ;
      ElemLag.Deialdia := Elem.Deialdia ;
      ElemLag.Nota := Elem.Nota ;
      Write (F,Elem) ;

      ElemLag.Kopiatua := TRUE ;
      Seek (FLagun,TxikiPos) ;
      Write (FLagun,ElemLag) ;
      TxikiPos := TxikienaZehaztu (IzenLagun) ;
   END ;

   Close (F) ;
   Close (FLagun) ;
   Erase (FLagun) ;          { Ezabatu fitxategi laguntzailea }
END ;


VAR                                  { Programa Nagusia }
  FitxIzen : String ;
BEGIN
   Write ('Noten arabera ordenatuko den fitxategiaren izena eman: ') ;
   ReadLn (FitxIzen) ;
   OrdenatuFitxategiz (FitxIzen) ;
END.
```

`OrdenatuFitxategiz` prozedurak bi urrats ditu, lehenengoan `WWWMMM.DAT` fitxategi laguntzailea sortu eta informazioz elikatzen da, bigarrenean `GELA_2.DAT` fitxategiak informazioa ordenaturik berreskuratzen du.

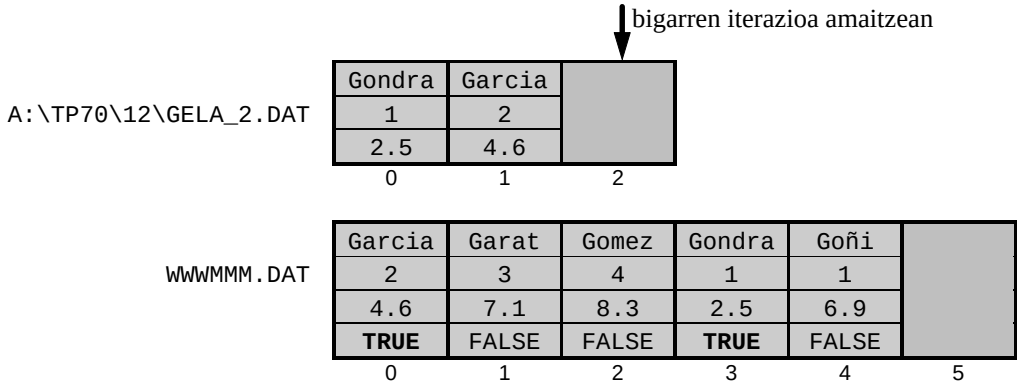
Hauxe da lehenengo urratsaren amaierari dagokion irudia:



Bigarren urratsaren `WWWMMM.DAT` fitxategi languntzailean kopiatu gabeko elementuak dauden bitartean, prozesu errepikakor bat iraunkorki mantentzen da. Aurreneko iterazioa bukatzean honela geratzen dira bi fitxategiak:



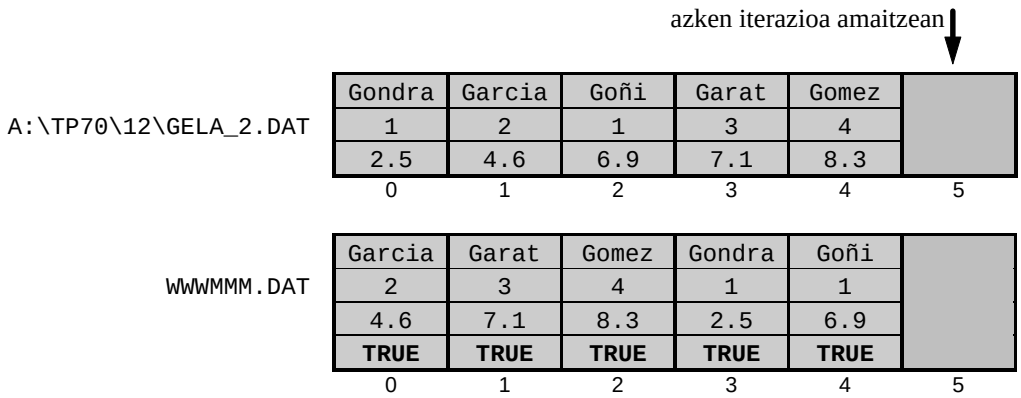
Hurrengo iterazioan `Garcia/2/4.6` ikaslea eramango da `GELA_2.DAT` fitxategira, izan ere `TxikienaZehaztu` funtzioak nota baxuen duen ikaslea aurkitzeko ez du kontutan izaten dagoenekoz kopiaturik dagoen `Gondra/1/2.5/TRUE` erregistroa. Bigarren iterazioaren amaieran honela geratzen dira bi fitxategiak:



Elementuen kopiaketa eteteko `WWWMMM.DAT` fitxategi languntzailearen elementu guztiak kopiatu direla frogatzen da, eta horixe da hain zuzen ere `TxikienaZehaztu` funtzioaren lehen zeregina:

```
{ Lehen "ez kopiatua" bilatu eta txikiena dela suposatu }
Aurkitua := FALSE ;
WHILE (NOT Eof (FLagun)) AND (NOT Aurkitua) DO
BEGIN
    Read (FLagun,Txiki) ;
    Non := Filepos(FLagun)-1 ;
    IF NOT Txiki.Kopiatua THEN Aurkitua := TRUE
END ;
```

`Aurkitua` aldagaiak `FALSE` balio badu fitxategi laguntzailearen elementu guztiak emaitza fitxategian kopiatu direlako izango da, ondorioz `TxikienaZehaztu` funtzioak `-1` itzuliko dio modulu deitzaileari fitxategiak honela geratzen direla:



`GELA_2.DAT` ordenatzen duen `OrdenatuFitxategiz` prozeduratik irten aurretik, diskoan zaramarik gera ez dadin, `WWWMMM.DAT` fitxategi laguntzailea ezabatzen da.

12.2 TEXT DATU-MOTAREN SARRERA

Fitxategi bitarrak zer diren aurreratu eta testuak aipatu BARNE1.PAS fitxategia berriro erakutsi eta azaldu.

Fitxategi bitaren sarrerako irudia berri jarri eta `27914` kopurua gordetzen duen fitxategi bikotea egin (bitarra eta testua TYPE_00.PAS kodea).

Leestma-Nyhoff liburutik 586 orrialdeko adibidea, konpiladore bat testu-fitxategi batetik abiatuta.

Azpiprogramen erabilpena justifikatzeko arrazoi ezberdinak daude, ondoko puntuetan garrantzitsuenak aipatuko ditugu: ahalik eta kode gutxiago idaztea, programa antolaturik egon dadila eta kodearen berrerabilpena.

12.2.1 Definizioa

Kodearen errepikapena ekiditea interesgarria izanik, azpiprogramak idazteak beste eragin garrantziago bat dakar: programa bera antolatzea. Hau da, programa bat eginkizun zehatzetan banatzea posiblea balitz eta eginkizun bakoitzeko azpiprograma bat idatziko balitz, orduan programaren antolatua suertatzen da ondoko abantailak dituelarik:

12.2.2 `Input` eta `Output` fitxategi estandarrak

Leestma-Nyhoff liburutik 216 orrialdetik hartuta, WriteLn Write ReadLn eta Read bigarren asaltoa.

Kodearen errepikapena ekiditea interesgarria izanik, azpiprogramak idazteak beste eragin garrantziago bat dakar: programa bera antolatzea. Hau da, programa bat eginkizun zehatzetan banatzea posiblea balitz eta eginkizun bakoitzeko azpiprograma bat idatziko balitz, orduan programaren antolatua suertatzen da ondoko abantailak dituelarik:

12.2.2.1 `WriteLn` prozedura eta `Output` fitxategia

12.2.2.2 `Write` prozedura eta `Output` fitxategia

12.2.2.3 `ReadLn` prozedura eta `Input` fitxategia

12.2.2.4 `Read` prozedura eta `Input` fitxategia

12.2.3 Aurredefinituriko azpiprogramak

12.2.3.1 Funtzioak

12.2.3.2 Prozedurak

12.1.2.2.1 `Assign` prozedura

Irakurleak gogoratuko duenez, `FitxategiBatIkusten` adibide-programan `Eof` funtzioa erabiltzen zen fitxategiaren osagai guztiak bistaratzeko:

```
Kontag := 0 ;
WHILE NOT Eof (F) DO      { Fitxategia amitzen ez den bitartean }
BEGIN
   Kontag := Kontag + 1 ;
END ;
```

`WHILE` kontrol-egiturak duen baldintzaren bitartez fitxategiaren amaieran aurkitzen garen ala ez testatzen da. Fitxategia bukatu ez denean `WHILE` barneko sententziak exekutatuko dira Fitxategia bukatzean `WHILE` blokearen hurrengo sententziara jauzi egingo du programaren fluxuak.

12.1.2.2.2 `Rewrite` prozedura

Irakurleak gogoratuko duenez, `FitxategiBatIkusten` adibide-programan `Eof` funtzioa erabiltzen zen fitxategiaren osagai guztiak bistaratzeko:

12.1.2.2.3 `Reset` prozedura

Irakurleak gogoratuko duenez, `FitxategiBatIkusten` adibide-programan `Eof` funtzioa erabiltzen zen fitxategiaren osagai guztiak bistaratzeko:

12.1.2.2.4 `Close` prozedura

Irakurleak gogoratuko duenez, `FitxategiBatIkusten` adibide-programan `Eof` funtzioa erabiltzen zen fitxategiaren osagai guztiak bistaratzeko:

12.1.2.2.5 `Read` prozedura

Irakurleak gogoratuko duenez, `FitxategiBatIkusten` adibide-programan `Eof` funtzioa erabiltzen zen fitxategiaren osagai guztiak bistaratzeko:

12.1.2.2.6 `Write` prozedura

Irakurleak gogoratuko duenez, `FitxategiBatIkusten` adibide-programan `Eof` funtzioa erabiltzen zen fitxategiaren osagai guztiak bistaratzeko:

12.1.2.2.7 `Seek` prozedura

Irakurleak gogoratuko duenez, `FitxategiBatIkusten` adibide-programan `Eof` funtzioa erabiltzen zen fitxategiaren osagai guztiak bistaratzeko:

12.1.2.2.8 `Truncate` prozedura

Irakurleak gogoratuko duenez, `FitxategiBatIkusten` adibide-programan `Eof` funtzioa erabiltzen zen fitxategiaren osagai guztiak bistaratzeko:

12.1.2.2.9 `Erase` prozedura

Irakurleak gogoratuko duenez, `FitxategiBatIkusten` adibide-programan `Eof` funtzioa erabiltzen zen fitxategiaren osagai guztiak bistaratzeko:

12.1.2.2.10 `Rename` prozedura

Irakurleak gogoratuko duenez, `FitxategiBatIkusten` adibide-programan `Eof` funtzioa erabiltzen zen fitxategiaren osagai guztiak bistaratzeko:

12.3 DOS UNITATEA

Sarrera bat

... eta funtzio/prozeduren taula

12.3.1 DOS unitateko funtzioak

12.3.1.1 `DiskFreef` eta `DiskSize` funtzioak

12.3.1.2 `DosExitCode` eta `DosVersion` funtzioak

12.3.1.3 `EnvCount` eta `EnvStr` funtzioak

12.3.1.4 `GetEnv` funtzioa

12.3.1.5 `FSearh` funtzioa

12.3.1.6 `FExpand` eta `FSplit` funtzioak

12.3.2 DOS unitateko prozedurak

12.3.2.1 `Exec` prozedura

12.3.2.2 `MsDos` prozedura

12.3.2.3 `FindFirst` eta `FindNext` prozedurak

12.3.2.4 `GetDate` eta `SetDate` prozedurak

12.3.2.5 `GetTime` eta `SetTime` prozedurak

Dagoenekoz ORDU1.PAS ORDU2.PAS ORDU3.PAS eta ERLOJU.PAS fitxategiak erakutsi eta azaldu.

12.3.2.6 `GetFTime` eta `SetFTime` prozedurak

12.3.2.7 `GetFAttr` eta `SetFAttr` prozedurak

12.4 PROGRAMAK

Hona hemen 12. kapituluaren programak orrialdeen arabera sailkaturik:

Izena	*Programaren identifikadorea*	*ORRI.*	*Ikasgaia*
FILETEXT.PAS	FileTextBarneAdierazpidea	12-05	File eta Text datu-mota
FILE_01.PAS	FitxategiBatSortzen	12-09	File datu-mota
FILE_02.PAS	FitxategiBatIkusten	12-10	File datu-mota
FILE_03.PAS	FitxategiBatNeurtzen	12-15	File datu-mota
FILE_04.PAS	ElementuenPosizioak	12-16	File datu-mota
FILE_05.PAS	FitxategiaAtzekozAurrera	12-30	File datu-mota
FILE_06.PAS	FitxategiBatMozten	12-34	File datu-mota
FILE_07.PAS	FitxategiBatEzabatzen	12-35	File datu-mota
FILE_08.PAS	FitxategiBatBerrizendatzen	12-37	File datu-mota
FILE_09.PAS	FitxategiaSortzekoProzedura	12-38	File datu-mota
FILE_10.PAS	FitxategiaIkustekoProzedura	12-39	File datu-mota
FILE_11.PAS	FitxategiBatenSorrera1	12-40	File datu-mota
FILE_12.PAS	FitxategiBatenSorrera2	12-41	File datu-mota
FILE_13.PAS	IOresultFitxategiekin	12-43	File datu-mota
FILE_14.PAS	FitxategiBatenExistentzia	12-44	File datu-mota
FILE_15.PAS	FitxategiBatenIbilera1	12-45	File datu-mota
FILE_16.PAS	FitxategiBatenIbilera2	12-46	File datu-mota
FILE_17.PAS	BilaketaFitxategietan	12-51	File datu-mota
FILE_18.PAS	GehiketaFitxategietan	12-53	File datu-mota
FILE_19.PAS	GehiketakFitxategietan	12-54	File datu-mota
FILE_20.PAS	AldaketaFitxategietan	12-55	File datu-mota
FILE_21.PAS	TartekaketaFitxategietan1	12-58	File datu-mota
FILE_22.PAS	TartekaketaFitxategietan2	12-60	File datu-mota
FILE_23.PAS	EzabaketaFitxategietan1	12-63	File datu-mota
FILE_24.PAS	EzabaketaFitxategietan2	12-65	File datu-mota
FILE_25.PAS	FitxategitikArrayera	12-66	File datu-mota
FILE_26.PAS	ArrayetikFitxategira	12-67	File datu-mota
FILE_27.PAS	FitxategiTxikiaOrdenez	12-68	File datu-mota
FILE_28.PAS	FitxategiHandiaOrdenez	12-70	File datu-mota

12.5 BIBLIOGRAFIA

- Borland, *TURBO PASCAL 7.0. Erabiltzailearen gida*, Borland International, 1992
- Borland, *TURBO PASCAL 7.0 Ingurunearen laguntza*, Borland International, 1992
- Leetsma, S., Nyhoff, L., *Programación en Pascal*, Prentice Hall, Madrid, 1999

KONTZEPTUEN INDIZE ALFABETIKOA

—E—

—F—

—G—

—H—

—I—

—K—

—L—

—M—

—N—

—O—

—P—

—R—

—S—

—T—

—U—

—V—

—W—

—X—

—Z—