eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

BILBOKO INGENIARIEN GOI ESKOLA TEKNIKOA

KONPUTAGAILUEN PROGRAMAZIOA
TURBO PASCAL BITARTEZ

III

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

8. ATALA: ERABILTZAILEAREN DATU-MOTAK

AURKIBIDEA

8.1 SARRERA

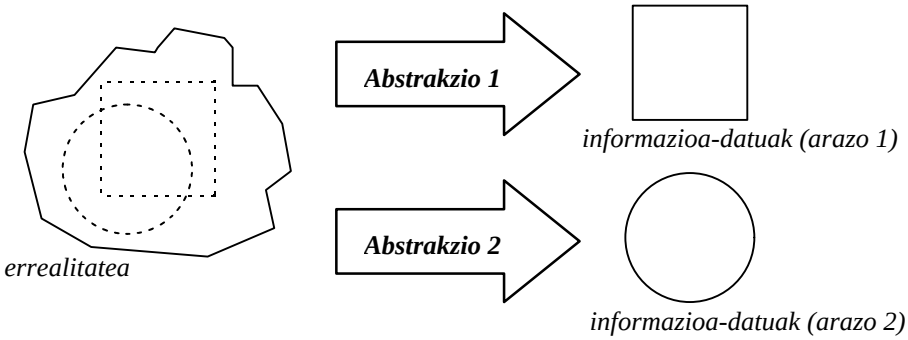
Arazo bat ebazten hasten garenean, ordenadorea erabiliz ala ordenadorerik gabe, bi urrats bete behar dira:

1. urratsa Datu esangurutsuen aukeraketa
2. urratsa Datuak errepresentatzeko moduaren aukeraketa

Bete beharreko urrats bi horiek zertan diren azal dezagun:

1. urratsa: Datu esangurutsuen aukeraketa

Esan dugun bezala arazo bat ebatzi nahi denean ordenadoreak erabiltzen dituen informazioak errealitatearen *abstrakzio* bat da (errealitatearen hurbilpen bat, errealitatearen sinplifikazio bat). Ordenadoreak landuko duen informazioa **datuen selekzio** bat izango da, arazoan zerikusia duten datuena hain zuzen ere, beste modu batez esanda bilatzen ari garen emaitzak lortzeko esangurutsuak diren datuak aukeratzen dira.



Goian erakusten den eskeman, errealitatea irudikatzeko konplexua eta aberatsa den zerbaitetan oinarritu gara. Errealitate horretan aurkitzen diren ezaugarri edo propietate guztietatik, *arazo 1* deitu dugun problemarako batzuk bakarrik interesatzen zaizkigu (laukiaren bitartez adierazi ditugunak), eta *arazo 1* problemari dagokion abstrakzioaz horiek jasoko dira besteak ahastuz (ondorioz *Abstrakzio 1* bitartez errealitatea sinplifikatu eta erraztu izan dugu). Errealitatea bat izanik *arazo 2* deitu dugun problemarako esangurutsuak diren ezaugarriak beste batzuk dira (zirkuluak adierazten dituena), horregatik bigarren arazoari dagokion abstrakzio-prozesuan abiapuntuko errealitatea bera izanik lortzen diren datuak bestelakoak dira.

Esate baterako, demagun errealitatea gure lagun taldea dela eta dauzkagun bi arazoak edo beharrak hauek direla: agenda bat egitea eta aprobetxamendu akademikoa neurtzea.

Gure lagunen taldean, besteak beste, honako ezaugarriak edo propietateak egon daitezke:

lagunaren izena
lagunaren deiturak
lagunaren telefonoa
lagunaren helbidea
lagunaren pisua
lagunaren altuera
lagunaren begien kolorea
lagunaren egoera zibila
lagunaren zaletasunak
lagunaren ikastetxea
lagunaren nota aurreskolan
lagunaren nota 1. mailan
lagunaren nota 2. mailan
lagunaren nota 3. mailan

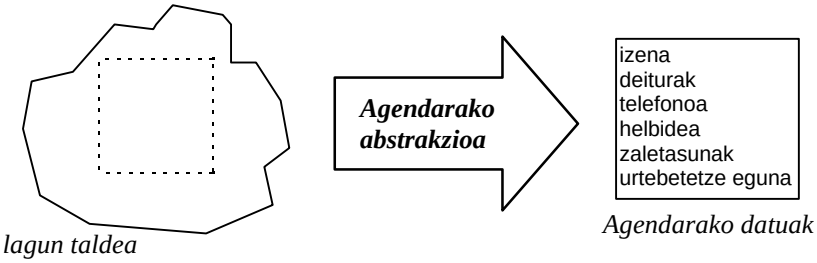
lagunaren nota 4. mailan
lagunaren urtebetetze eguna
...

Zer esanik ez, goian emandako ezaugarrien zerrenda baino askoz aberatsago eta konplexuago izan daiteke lagun baten kontzeptua.

Hona hemen lehen abstrakzioa burutu ondoren aukeratutako datuak:

Errealitatea (lagun taldea)	**Aukeratutako datuak (agenda)**	**Baztertutako errealitatearen zatia**
izena	izena	
deiturak	deiturak	
telefonoa	telefonoa	
helbidea	helbidea	
pisua		pisua
altuera		altuera
begien kolorea		begien kolorea
egoera zibila		egoera zibila
zaletasunak	zaletasunak	
ikastetxea		ikastetxea
nota aurreskolan		nota aurreskolan
nota 1. mailan		nota 1. mailan
nota 2. mailan		nota 2. mailan
nota 3. mailan		nota 3. mailan
nota 4. mailan		nota 4. mailan
urtebetetze eguna	urtebetetze eguna	
...		...

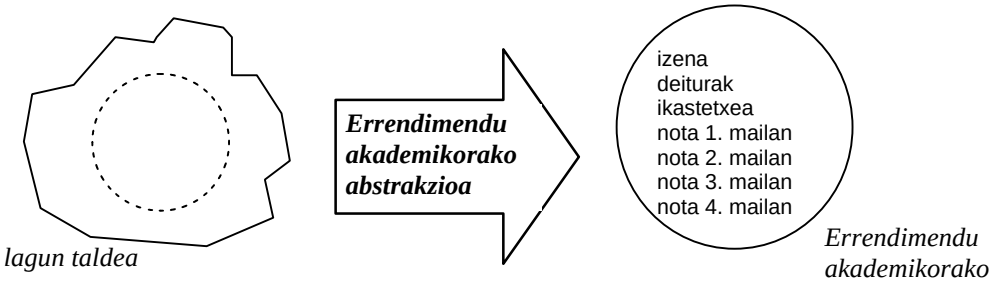
Ikus daitekeenaz agenda bat egiterakoan funtsezko datuak lagunen identifikagarriak (izena, deiturak, telefonoa eta helbidea) dira, eta horiekin batera urteak noiz betetzen dituen eta gustuko dituen zaletasunak. Beraz, agenda eraikitzearen arazoari dagokion abstrakzioaz errealitatea sinplifikatu egiten da.



lagun taldea

Agendarako datuak

Abstrakzioa egiterakoan arazoak berak markatzen du zein ezaugarri onartu eta zein baztertu, horregatik ez da harritzekoa bigarren arazoa ebazteko (errendimendu akademikoa neurtzeko) behar izango diren propietateak bestelakoak izatea. Jarraian bigarren abstrakzio-prozesu hau erakusten da, demagun hauxe litzatekeela bigarren abstrakzioari dagokion eskema:



lagun taldea

Errendimendu akademikorako datuak

Lagun baten aprobetxamendu akademikoa baloratzeko funtsezko datuak erdiko zutabekoak izanik, abstrakzio-prozesuan gainerakoak arbuiatu egiten dira errealitatearen hurbilpen laburtua lortuz:

Errealitatea (lagun taldea)	**Aukeratutako datuak (errendimendu akademikoa)**	**Baztertutako errealitatearen zatia**
izena	izena	
deiturak	deiturak	
telefonoa		telefonoa
helbidea		helbidea
pisua		pisua
altuera		altuera
begien kolorea		begien kolorea
egoera zibila		egoera zibila
zaletasunak		zaletasunak
ikastetxea	ikastetxea	
nota aurreskolan		nota aurreskolan
nota 1. mailan	nota 1. mailan	
nota 2. mailan	nota 2. mailan	
nota 3. mailan	nota 3. mailan	
nota 4. mailan	nota 4. mailan	
urtebetetze eguna		urtebetetze eguna
...		...

Arazo bat abazteko 1. urratsa den datuen aukeraketa arazoak berak gidatzen du, eta bere ondorioa errealitatearen sinplifikazioa eta idealizazioa litzateke.

2. urratsa: Datuak errepresentatzeko moduaren aukeraketa

Bigarren urrats hau ez da aurrekoarekiko erabat independente, eta batez ere arazoa ebazteko erabiliko den tresnan oinarritzen da. Lehen urratsak bigarrenari zer eragina sortzen dion adibide baten bitartez ikus dezagun: Demagun 1. urratsaren ondorioz errealitatetik hartutako propietate edo ezaugarriren bat **K** kopuru baten bitartez sinplifikatzen edo abstraitzen ari garela; nolako adierazpidea erabaki **K** kopuru horretarako litzateke 2. urratsaren zioa.

K kopurua **K** makiltxo bidez adieraz daiteke. Adierazpide hau oso intuitiboa denez ondo ulertzen da kopuru txikien batuketak eta kenketak burutzeko, eta horixe litzateke adierazpide honen abantaila bakarra.

Bestalde, **K** kopurua zenbaketa sistema hindoarabigoaren bidez adieraz daiteke ere. Errepresentazio honetan kopurua digituz ematen da, eta digituen pisua euren posizioaren araberakoa da (**2.4.1 Zenbaketaren Oinarrizko Teorema** puntua gogoratu). Oso intuitiboa ez delako adierazpide honek kalkuluak egiteko arauak behar ditu, baina kopuru handiak lantzeko aurrekoa baino aproposagoa da, halaber zenbaketaren sistemak erabiliz kopuruen arteko zatiketak eta biderkaketak burutu ahal dira.

Ikusten denez 1. urratsak markatzen du **K** kopurua eta mugatzen ditu bide batez 2. urratsaren posibilitateak, **K** txikia denean makiltxoen adierazpidea aukera daiteke baina **K** kopurua handia denenan inolaz ere.

Arazo baten ebazpidean 2. urratsa datuak errepresentatzeko modua aukeratu behar dela esan dugu, eta urrats hau arazoa ebazteko erabiliko den tresnan oinarritzen da gehien bat. Gainera errepresentazioa edo adierazpidea mailakatuta egon ohi da, ondoko adibidean azaltzen den bezala.

Demagun 1. urratsaren ondorioz errealitatetik hartutako propietateren bat posizio baten koordenatuak direla, eta hori abstraitzeko **X** eta **Y** zenbaki errealak hautatu izan direla. **X** zenbaki erreal baten adierazpidea ordenadorearen bitartez lantzeko honelako mailetan bana daiteke:

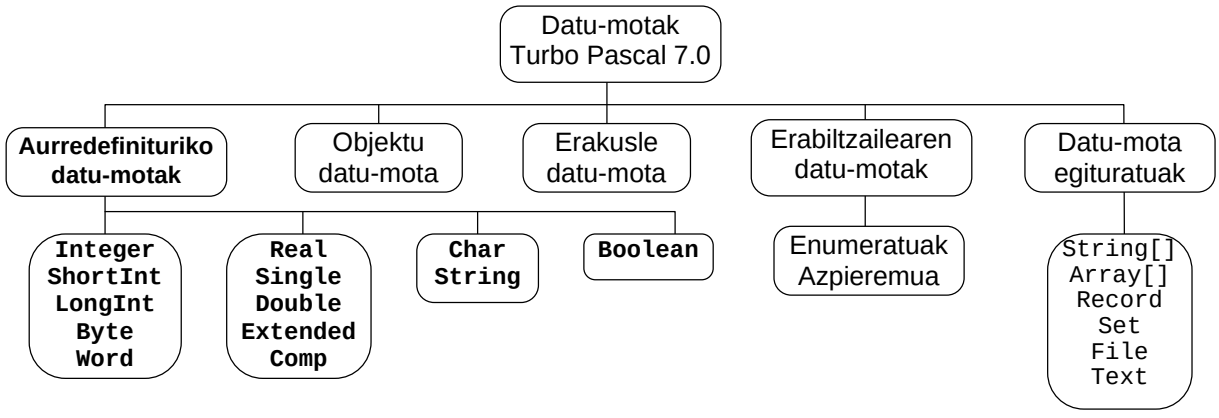
1. Bigarren kapituluan azaldutako **2.4.3.1 Koma higikorra** puntua gogoratuz **X** zenbaki erreal bat osoak diren **B** eta **M** zenbaki osoen bitartez adieraz daiteke. Non **B** eta **M** berretzailea eta mantisa diren.
2. Behin **X** zenbaki erreal bat zenbaki osoen arabera jarriz gero, maila honek gainditu behar duen zailtasuna **B** edo **M** bezalako zenbaki osoei dagokien adierazpidea finkatzea. Gai hau bigarren kapituluko **2.4.2 Zenbaki osoen adierazpidea** puntuan landu genuen, non kopuru oso bat sistema bitarrean oinarriturik teknikak aurkezten genituen eta hura gogoratzea gomendatzen dugu.
3. Kopuru oso bat zenbait bit bitartez adierazterik badago, azken mailaren zeregina bit bat nola errepresentatu litzateke. Bit baten adierazpiderako modu bat baino gehiago dagoela dakigu, hala nola tentsio edo korrontearen presentzia, fluxu magnetikoaren direkzioa, paper edo euskarri plastiko baten gaineko zuloa, etab.

Xehetasun guzti hauek programadoreak ez ditu aintzat hartuko, bere laguntzarako goimailako lengoaiak asmatu direlako. Izan ere goimailako lengoaia baten oinarrizko abstrakzioak eskaintzen dizkio programadoreari, adibidez zenbaki osoekin lan egin ahal izateko `Integer` eta bere familiako datu-motak eskainiz.

Urrunago joanda, aurredefinituriko datu-mota bakunak abiapuntutzat harturik goimailako lengoaiek datu-mota egituratuekin (estrukturatuekin) lan egiteko aukera ematen dute.

8.2 DATU-MOTAK TURBO PASCAL LENGOAIAN

Laugarren kapituluan Turbo Pascal lengoaian datu-moten sailkapen hau egin genuen, Non lengoaiak aurredefiniturik dituen datu-motak (belzturik dagoen zatia) beste datu-mota guztien oinarria den:



Erabiltzailearen datu-mota, suposa daitekeenez, programadoreak sortzen duen datu-mota berria izango da. Datu-mota berri horrek sortu den programarako baliagarria izango da eta lengoaiak aurredefinituriko dituen datu-motak oinarritzat izango ditu. Laugarren kapituluan edozein lengoaiak aurredefinituriko dituen datu-motak bi zatiz osatuak agertzen direla aipatu genuen ere:

1. Aurrez determinaturiko memorian biltegitzeko sistema
2. Datu-mota horrentzat egokiak diren eragiketak

Kontzeptu bi horiek kontutan izan beharko dira norberak bere datu-mota propioak sortuko dituenean, baina zein da datu-mota berriak sortzearen beharra?. Aldagai jakin bat erazagutzean ez al da bere datu-mota deklaratzen? Zergatik ez da nahikoa `String`, `Array`, `Record`, `Integer`, `File`, `Char` edo den delako aldagaia erabiltzea?. Galdera horien erantzuna ondoko adibidea aztertuz asma daiteke:

```
PROGRAM DatuMotarikGabe1 ;                          { \TP70\08\DATU_MO1.PAS }
VAR
  Langile : RECORD                          { erregistro aldagai bat }
               Izena : String ;
               Adina : Byte ;
            END ;
  Ikasle : RECORD                           { beste erregistro aldagai bat }
              Izena : String ;
              Adina : Byte ;
           END ;
BEGIN
   Write ('Langilearen izena eman: ') ;
   ReadLn (Langile.Izena) ;
   Write (Langile.Izena, '-(r)en adina eman: ') ;
   ReadLn (Langile.Adina) ;

   Ikasle := Langile ;                      { datu-mota errorea }

   WriteLn ;
   WriteLn ('Ikasle ALDAGAIAREN EDUKIA:') ;
   WriteLn ('=========================') ;
   WriteLn ('Ikasle.Izena -----> ', Ikasle.Izena) ;
   WriteLn ('Ikasle.Adina -----> ', Ikasle.Adina) ;
END.
```

`DatuMotarikGabe1` programaren sententziei begiraturik bere zeregina erraz ulertzen da: `Langile` etiketa duen erregistro bat deklaratu da (erregistro hori bi eremuz konposatzen da karaktere-kate bat eta positiboa den zenbaki oso bat), gero `Ikasle` etiketa duen bigarren erregistro bat erazagutu da ere (kontutan izan `Langile` erregistroa eta `Ikasle` erregistro “berdinak” diren bi aldagaiak direla); programaren hasieran `Langile` aldagaian balioak gordetzen dira eremu biak teklatuz irakurriz; ondoren, eta hemen dago programa osoaren koska “berdinak” edo “baliokideak” diren bi aldagaien artean esleipena egiten da `Ikasle` erregistroari `Langile` aldagaiak dituen datuak transferitu nahi baitzaizkio; programa bukatu aurretik `Ikasle` aldagaiaren edukia pantailaratzen da.

`DatuMotarikGabe1` programa ezin daiteke exekutatu konpilazio garaian errorea ematen duelako. Konpiladorearentzat `Ikasle` eta `Langile` aldagaiak baliokideak ez direla salatzen du errore-mezuak: `Error 26: Type mismatch.`

Dagoenekoz dakiguna aplikatuz erregistroen arteko gertaera hori ekiditeko bi bide daude, eta biak nahiko traketsak dira:

Esleipenak eremuka burutzea:

```
Ikasle.Izena := Langile.Izena ;
Ikasle.Adina := Langile.Adina ;
```

Aldagaiak elkarrekin erazagutzea:

```
VAR
  Langile, Ikasle : RECORD
                       Izena : String ;
                       Adina : Byte ;
                    END ;
```

Esleipenak eremuka burutzea ez da orokorrean komeniko, batez ere, erregistroak eremu asko izango dituenean. Bestalde aldagaiak elkarrekin batera deklaratzeak baimendu egiten digu erregistroen arteko esleipen bakarra eginez informazioa ezartzea, baina arazoa `Ikasle` edo `Langile` aldagaiak azpiprograma batera pasatu nahi denean agertzen zaigu, parametro formala nola erazagutu ez baitakigu.

Erregistroa parametro bezala ez:

```
PROCEDURE P1 (Langile : ? ) ;
BEGIN

END ;
```

Erregistroen arteko esleipena bai:

```
VAR
  Langile, Ikasle : RECORD
                      Izena : String ;
                      Adina : Byte ;
                    END ;
```

Beraz gauzarik onena datu-mota berri bat eraikitzea litzateke, ondorioz aldagaiak erazagutzean honelako zerbait idatzi ahal izango litzateke:

```
VAR
   Langile, Ikasle : DM_Pertsona ;
```

Soluzio bakoitzari dagokion programa idatzi dugu, esleipenak eremuka egiten dituena `DatuMotarikGabe2` deitu dugu, aldagaiak elkarrekin deklaratzen dituen programaren izena `DatuMotarikGabe3` da eta `DM_Pertsona` delako datu-mota berriaz baliatzen den programa `DatuMotaBerriarekin` izendatu dugu. Hona hemen `DatuMotarikGabe2`, `DatuMotarikGabe3` edo `DatuMotaBerriarekin` programak exekutatzean lor daitekeen irteera bat:

```
Langilearen izena eman: Eguzkiñe
Eguzkiñe-(r)en adina eman: 23

Ikasle ALDAGAIAREN EDUKIA:
==========================
Ikasle.Izena -----> Eguzkiñe
Ikasle.Adina -----> 23
_
```

Ikus `DatuMotaBerriarekin` programaren sententziak jarraian datorren **8.3** puntuan.

8.2.1 Datu-moten arteko bihurketak

Zenbaitetan datu-mota bateko espresioa edo aldagaia beste datu-mota desberdin batera bihurtzea komeniko zaigu. Horrelako zerbait lortzeko gure programetan bihurketaren maila kontutan izango dugu, labur esanda hiru dira bereiziko ditugun bihurketa-mailak:

1. Bihurketa automatikoa
2. Datu-mota moldaketa
3. Funtzio eta prozeduraz lorturiko bihurketa

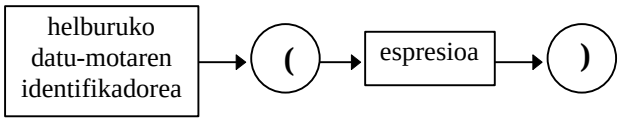
Esate baterako, jarraian erakusten den `DatuMotaBihurketak` izeneko programan ikus daitezke hiru bihurketa mailak:

```
PROGRAM DatuMotaBihurketak ;                    { \TP70\08\BIHURKET.PAS }
VAR
  Osokoa : Integer ;
  Erreala : Real ;
  Positiboa : Word ;
  Karakterea : Char ;
  Katea : String ;
BEGIN
   Write ('Zenbaki oso bat eman: ') ;
   ReadLn (Osokoa) ;

   Erreala := Osokoa ;              { 1. bihurketa (bihurketa automatikoa) }
   Write('Osokotik errealera    -----> ');
   WriteLn(Osokoa, ' (Integer) = ', Erreala, ' (Real)');
```

```
   Positiboa := Word(Osokoa) ;      { 2. bihurketa (moldaketa) }
   Write('Osokotik zeinu gabera -----> ');
   WriteLn(Osokoa, ' (Integer) = ', Positiboa, ' (Word)');

   Karakterea := Char(Osokoa) ;     { 3. bihurketa (moldaketa) }
   Write('Osokotik karakterera  -----> ');
   WriteLn(Osokoa, ' (Integer) = ', Karakterea, ' (Char)');

   Karakterea := Chr(Osokoa);       { 4. bihurketa (funtzioa) }
   Write('Osokotik karakterera  -----> ');
   WriteLn(Osokoa, ' (Integer) = ', Karakterea, ' (Char)');

   Str(Osokoa, Katea );             { 5. bihurketa (prozedura) }
   Write('Osokotik kateara      -----> ');
   WriteLn(Osokoa, ' (Integer) = ', Katea, ' (String)');

END.
```

`DatuMotaBihurketak` programan, teklatuaren bitartez `Integer` bat irakurri ondoren bihurketa desberdinak burutzen dira. Lehenengoan `Integer` datua `Real` zenbaki errealera igarotzen da, bihurketa hau automatikoa delako lehen mailakoa da eta gerta dadin ez da ezer berezirik egin behar.

Bigarren bihurketan `Integer` datu-motatik abiatuta `Word` datu-mota lortzen da, kasu honetan moldaketa bat egin behar da honelako sintaxia gordez:



Hirugarren bihurketa bigarrena bezalakoa da, baina `Integer` datu-motatik abiatuta `Char` datu-mota lortzen da, kasu honetan ere dagokion moldaketa egin behar da aurreko sintaxia zainduz.

`DatuMotaBihurketak` adibide-programaren laugarren eta bosgarren bihurketak azken mailakoak dira, hots, jatorrizko eta helburuko datu-motak urrunak direlako azpirrutinak aplikatu behar dira. Adibidez, `Integer` datu-motatik String datu-mota lortzeko `Str` prozedura, edo, `Integer` datu-motatik `Char` datu-motara igarotzeko `Chr` funtzioa erabil daiteke (nahiz eta helburu bera bete ez nahastu `Char()` moldaketa eta `Chr()` funtzioa).

Hona hemen `DatuMotaBihurketak` programaren exekuzioaren emaitza irakurriko datua `69` denean:

```
Zenbaki oso bat eman: 69
Osokotik errealara    -----> 69 (Integer) =  6.9000000000+E01 (Real)
Osokotik zeinu gabera -----> 69 (Integer) =  69 (Word)
Osokotik karakterera  -----> 69 (Integer) =  E (Char)
Osokotik karakterera  -----> 69 (Integer) =  E (Char)
Osokotik kateara      -----> 69 (Integer) =  69 (String)
_
```

Datu-moten bihurketak bideratzen dituzten azpirrutinak asko dira Turbo Pascal programazio-lengoaian, dakigunez gehieneak funtzioak dira eta gutxi batzuk prozedurak.

Moldaketaren adibidea

```
Positiboa := Word(Osokoa) ;
```

Ezenezko moldaketa

```
Katea := String(Osokoa) ;
```

8.3 DATU-MOTA BERRIAK SORTZEN

Erabiltzaileak eraikiko duen datu-mota berri bat definitzen den programan baliagarria izango da. Hurrengo puntuetan lehenik erabiltzailearen datu-mota sinpleak (datu-mota enumeratuak eta azpieremu datu-mota) ikusiko ditugu eta ondoren datu-mota egituratuen sarrera bat egingo dugu.

Edozein kasutan datu-mota berri bat sortzeko, programaren deklarazioen blokean `TYPE` hitz erreserbatuaren bitartez egiten da. Adibidez, jarraian ematen diren erazagupenetan `DM_AsteBurua`, `DM_Erantzunak`, `DM_Minuskulak` eta `DM_Orduak` datu-mota sinpleak deklaratzen dira:

```
TYPE
   DM_AsteBurua = (Ostirala, Larunbata, Igandea) ;
   DM_Erantzunak = (Bai, Ez) ;
   DM_Minuskulak = 'a'..'z' ;
   DM_Orduak = 0..23 ;
```

Eta modu berdintsuan `DM_Pertsona`, `DM_KateLuze`, `DM_Onargarri` eta `DM_Zerrenda` datu-mota egituratuak deklara daitezke:

```
TYPE
   DM_Pertsona =  RECORD
                     Izena : String ;
                     Adina : Byte ;
                  END ;
   DM_KateLuze =  String[250] ;
   DM_Onargarri = SET OF DM_Erantzunak ;
   DM_Zerrenda =  ARRAY[1..80] OF DM_Pertsona ;
```

`DM_Pertsona` datu-mota egituratua bi eremuko erregistro bat da, `DM_KateLuze` berriz gehienez 250 karaktere gordetzeko string bat da, `DM_Onargarri` datu-mota multzo bat da zein `DM_Erantzunak` datu-motan oinarritzen den, azkenik, `DM_Zerrenda` erregistroen array bat da.

Beraz, datu-mota berriak lortu ahal izateko programaren deklarazioen blokean `TYPE` hitz erreserbatua jarriko da, eta gero informazioa gordeko duen aldagaia dagokion `VAR` arloan erazagutuko da. Ikus lehenago aipatu den `DatuMotaBerriarekin` programa, non `Langile` eta `Ikasle` aldagaiak `DM_Pertsona` erregistroak diren:

```
PROGRAM DatuMotaBerriarekin ;                    { \TP70\08\DATU_MO4.PAS }
TYPE
  DM_Pertsona = RECORD                        { erregistro datu-mota bat }
                   Izena : String ;
                   Adina : Byte ;
                END ;
VAR
  Langile, Ikasle : DM_Pertsona ;             { erregistro aldagai bi }
BEGIN
   Write ('Langilearen izena eman: ') ;
   ReadLn (Langile.Izena) ;
   Write (Langile.Izena, '-(r)en adina eman: ') ;
   ReadLn (Langile.Adina) ;

   Ikasle := Langile ;                        { errorerik ez }

   WriteLn ;
   WriteLn ('Ikasle ALDAGAIAREN EDUKIA:') ;
   WriteLn ('========================') ;
   WriteLn ('Ikasle.Izena -----> ', Ikasle.Izena) ;
   WriteLn ('Ikasle.Adina -----> ', Ikasle.Adina) ;
END.
```

`DatuMotaBerriarekin` programan `DM_Pertsona` esplizitoki definituriko datu-mota bat da eta derrigorrez aldagaien deklarazioaren baino lehenago jarri behar da. Erregistro datu-mota berrirako guk aukeratu dugun etiketa `DM_` aurrizkiaz jantzirik dago, identifikadore horren bitartez datu-mota batez ari garela berehala kontura gaitezke jokamolde honek bere abantailak dituen ohitura gomendagarri bat da.

8.4 DATU-MOTA ENUMERATUAK

Datu-mota enumeratuak edo zerrendatuak erabiltzaileak definitzen ditu har ditzakeen balio posibleak esplizitoki zehaztuz. Beraz, datu-mota enumeratuak deklaratzean horrelako aldagai batek har ditzakeen balio posibleak ezaguna izango da, lehenago ikusitako `DM_AsteBurua` datu-mota adibidez hiru balio hartzeko definitu da balioen zerrenda honela idatziz:

```
TYPE
   DM_AsteBurua = (Ostirala, Larunbata, Igandea) ;
```

Ondoren `DM_AsteBurua` datu-motako `Eguna` eta `Noiz` aldagaiak erazagut daitezke, jakinik horiek gordeko dutena hiru baliotatik bat izango dela. Izan ere, zerrendatu diren balioak `Eguna` eta `Noiz` aldagaiek biltegituko dituzten konstanteak izango dira.

```
VAR
   Eguna, Noiz : DM_AsteBurua ;
```

Datu-mota enumeratuak (zenbaki osoak eta karaktereak bezala) ordinalak dira, hau da, zerrendako balioen artean orden erlazio bat dagoelako lehena bigarrena baino txikiagoa kontsidera daiteke, bigarrena hirugarrena baino txikiagoa, eta abar. Datu-mota enumeratutik ondoriotzen den zerrendako lehen etiketak (`Ostirala`-k adibidean) duen orden balioa `0` da, bigarren etiketari (`Larunbata`-ri) dagokion orden balioa `1` da, eta, azkenekoari `3`.

Barne ordena dela eta, enumeratuen artean, ondoko konparaketak egin daitezke:

```
IF Eguna < Larunbata THEN
BEGIN
   ...
END ;
```

```
FOR Noiz:=Ostirala TO Igandea DO
BEGIN
   ...
END ;
```

Ordinaltasunaren arrazoi beragatik `Pred()`, `Succ()`, eta `Ord()` funtzio estandarrak aplika daitezke datu-mota enumeratua den aldagai orori (ikus **8.4.1 Datu-mota enumeratuak. Adibidea** izenburuko puntua). Datu-mota enumeratuekin `Pred()`, `Succ()`, eta `Ord()` funtzioak honela erabiltzen dira:

`Pred()` sarrera: zerrendaren balio jakin bat
irteera: zerrendaren balio horren aurreko konstantea

`Succ()` sarrera: zerrendaren balio jakin bat
irteera: zerrendaren balio horren ondoko konstantea

`Ord()` sarrera: zerrendaren balio jakin bat
irteera: balio horri zerrendan dagokion ordinala

Ondorioz, agindua hauek baliokideak izango dira:

```
Eguna := Larunbata ;
Eguna := Pred (Igandea) ;
Eguna := Succ (Ostirala) ;
```

8.4.1 Datu-mota enumeratuak. Adibidea

Datu-mota enumeratuen adibiderako programa txiki hau asmatu dugu, bertan piztien zerrenda bat egiten da eta aldagai bi deklaraturik esleipen bana egiten dugu:

```
PROGRAM ZooDatuMota ;                          { \TP70\08\DATU_MO5.PAS }
TYPE
  DM_Zoo = (Kaiman, Krokodrilo, Suge, Elefante, Hartz,
            Lehoi, Tigre, Orein, Arrano, Ostruka, Sai) ;
VAR
  Pizti1, Pizti2, Animali : DM_Zoo ;
BEGIN
   WriteLn ('Zookideak: Kaiman, Krokodrilo, Suge, Elefante, Hartz,') ;
   WriteLn ('           Lehoi, Tigre, Orein, Arrano, Ostruka, Sai.') ;
   WriteLn ('-----------------------------------------------------') ;

   Pizti1 := Elefante ;
   WriteLn ('Dumbo-ri dagokion ordinala: ', Ord(Pizti1)) ;
   Pizti2 := Orein ;
   WriteLn ('Bambi-ri dagokion ordinala: ', Ord(Pizti2)) ;

   FOR Animali:=Kaiman TO Suge DO
   BEGIN
      Pizti1 := Pred(Pizti1) ;
      Pizti2 := Succ(Pizti2) ;
   END ;
   WriteLn ('Azkenean Pizti1-k Kaiman balio izango du: ', Ord(Pizti1)) ;
   WriteLn ('Azkenean Pizti2-k Sai balioa hartuko du: ', Ord(Pizti2)) ;
END.
```

`ZooDatuMota` programak zati bi ditu, `Pizti1` eta `Pizti2` aldagaiei esleipen bana egin ondoren euren zerrendako ordinalak pantailaratzen dira; programaren bigarren zatian `FOR-DO` egitura baten erabilpena erakusten da zeinean `Pred()` eta `Succ()` funtzioak aplikatzen diren.

`ZooDatuMota` programaren exekuzioak honako hau ematen du:

```
Zookideak: Kaiman, Krokodrilo, Suge, Elefante, Hartz,
           Lehoi, Tigre, Orein, Arrano, Ostruka, Sai.
-----------------------------------------------------
Dumbo-ri dagokion ordinala: 3
Bambi-ri dagokion ordinala: 7
Azkenean Pizti1-k Kaiman balio izango du: 0
Azkenean Pizti2-k Sai balioa hartuko du: 10
_
```

8.4.2 Datu-mota enumeratuak. Sendotasuna

Datu-mota enumeratuaz baliatzen diren programen sendotasuna irakurgarritasunean datza. Hots, datu-mota enumeratuak definiturik duen programaren kodea errazago irakurri eta ulertzen da datu-motaren etiketak oso esplizitoak baitira.

Adibidez, `OrdutegiaDatuMotarikGabe` programan astearen egunak (astelehenetik ostirala bitarte) lantzen dira, lanegun bakoitzari dagokion ikasgaiak pantailaratuz. Horretarako batetik bostera aldatuko den `Kontagailu` zenbakizko aldagai positiboa erabiltzen da, kodea aztertzean programak honelako barne taula darabilela kontura gaitezke.

Eguna	`Kontagailu`**-ren balioa**
Astelehena	1
Asteartea	2
Asteazkena	3
Osteguna	4
Ostirala	5

Programa txiki honetan egunaren kontzeptua zenbaki osoen bitartez adierazteak ez du kalterik eragiten. Baina demagun programa konplexuago batean jokabide bera erabiltzen delako `1` konstanteak kontzeptu ezberdinak adierazten dituela (adibidez: astelehena, urtarrila, gizonezko, bekadun, irakasle, ezkongabea, ...), kasu honetan `OrdutegiaDatuMotarikGabe` programaren `FOR Kontagailu:=1 TO 5 DO` sententzia ez da horren erraz ulertuko.

```
PROGRAM OrdutegiaDatuMotarikGabe ;            { \TP70\08\DATU_MO6.PAS }
VAR
  Kontagailu : Byte ;
BEGIN
   WriteLn ('Astegunaren ordutegia:') ;
   WriteLn ('---------------------') ;

   FOR Kontagailu:=1 TO 5 DO
   BEGIN
      CASE Kontagailu OF
         1 : WriteLn ('Astelehen: Fisika, Algebra, Marrazketa') ;
         2 : WriteLn ('Astearte:  Kimika, Kalkulu, Informatika') ;
         3 : WriteLn ('Asteazken: Algebra, Marrazketa, Kimika') ;
         4 : WriteLn ('Ostegun:   Kalkulu, Informatika, Algebra') ;
         5 : WriteLn ('Ostiral:   Informatika, Marrazketa') ;
      END ;
   END ;
END.
```

Datu-mota enumeratuak aintzat harturik `OrdutegiaDatuMotarikGabe` programaren bertsioa honela geratuko litzateke:

```
PROGRAM OrdutegiaDatuMotarekin ;              { \TP70\08\DATU_MO7.PAS }
TYPE
  DM_Astegun = (Astelehen, Astearte, Asteazken, Ostegun, Ostiral) ;
VAR
  LanEgun : DM_Astegun ;
BEGIN
   WriteLn ('Astegunaren ordutegia:') ;
   WriteLn ('---------------------') ;

   FOR LanEgun:=Astelehen TO Ostiral DO
   BEGIN
      CASE LanEgun OF
         Astelehen : WriteLn ('Astelehen: Fisika, Algebra, Marrazketa') ;
         Astearte : WriteLn ('Astearte:  Kimika, Kalkulu, Informatika') ;
         Asteazken : WriteLn ('Asteazken: Algebra, Marrazketa, Kimika') ;
         Ostegun : WriteLn ('Ostegun:   Kalkulu, Informatika, Algebra') ;
         Ostiral : WriteLn ('Ostiral:   Informatika, Marrazketa') ;
      END ;
   END ;
END.
```

`OrdutegiaDatuMotarekin` programa ulergarriagoa dela zalantzarik ez dago. Bi dira horren arrazoiak, batetik `Kontagailu` identifikadore neutroa erabili beharrean `LanEgun` delako aldagaia jarri, bestetik, eta garrantzitsuagoa, datu-mota enumeratuek berehala salatzen dute `FOR-DO` eta `CASE-OF` sententzien asmoa.

8.4.3 Datu-mota enumeratuak. Ahulezia

Datu-mota enumeratuek duten muga sarrera/irteera azpiprograma estandarrekin loturik dago. Izan ere, ezinezkoa baita erabiltzaileak eraikitako datu-mota enumeratu batetik ondoriotzen den aldagaiari `ReadLn()` edo `WriteLn()` prozedurak aplikatu.

Beraz, aurreko `OrdutegiaDatuMotarekin` programari egin diezaiokegun gehigarri hau konpilaezina da:

```
FOR LanEgun:=Astelehen TO Ostiral DO
BEGIN
   Write ('Astegun bat eman: ') ;
   ReadLn (LanEgun) ;
   CASE LanEgun OF
      Astelehen : BEGIN
                     Write (LanEgun) ;
                     WriteLn (': Fisika, Algebra, Marrazketa') ;
                  END ;
```

Sarrera/irteera estandarren ezintasunak erabat baldintzaten du erabiltzailearen datu-mota enumeratua.

8.5 AZPIEREMU DATU-MOTA

Ordinalak diren datu-motak oinarritzat harturik azpieremu datu-motak defini daitezke. Orain arte, testuaren zehar, aipatu ditugun datu-mota ordinal estandarrak hauek dira:

Datu-mota ordinal estandarrak		
Datu-mota	**Heina**	
	Behe-muga	**Goi-muga**
`ShortInt` (8 bit)	`-128`	`127`
`Integer` (16 bit)	`-32768`	`32767`
`LongInt` (32 bit)	`-2147483648`	`2147483647`
`Byte` (8 bit)	`0`	`255`
`Char` (8 bit)	`Chr(0)`	`Chr(255)`
`Boolean` (8 bit)	`FALSE`	`TRUE`

Ordinal estandarren azpieremu bat definitzeko, mota berri horretako aldagaiek har ditzaketen baliorik txikiena eta handiena zehaztuz egiten da. Esate baterako, honela sortzen dira karakteren eta zenbaki osoen erabiltzailearen azpieremu bana:

```
TYPE
   DM_Minuskulak = 'a'..'z';
   DM_Orduak = 0..23 ;
   DM_Bokalak = 'a'..'u';         { hau ondo dago? }
```

`DM_Minuskulak` datu-mota berriaren oinarria `Char` datu-mota da, eta `DM_Orduak`-rena `Integer` datu-mota. Azpieremu datu-mota batek oinarritzat duen motari *ostalari* deritzo, adibidera joz `DM_Minuskulak` datu-motaren ostalaria `Char` datu-mota da eta `DM_Orduak`-rena `Integer` datu-mota.

`DM_Bokalak` datu-motari buruz ohar txiki bat. Ondo definiturik ote dagoen galdetzen da, eta egia esan sintaktikoki onargarria denez ez luke inolako arazorik emango, baina, datu-motaren identifikadoreari erreparatuz `DM_Bokalak` azpieremu datu-mota berriak bost bokal minuskulak deklaratzeko asmatuta dagoela ematen du. Orduan bai gaizki ibiliko litzatekeela, `'a'` eta `'u'` artean dauden karaktere guztiak onartzen batitu (ASCII taula nola dagoen antolatuta ikusirik bokalak definitzeko soluzio bakarra bostak banan banan enumeratzea litzateke).

Datu-mota enumeratuak, definizioz, ordinalak direnez gero euretan oinarriturik ere azpieremuak deklara daitezke. Horretarako, lehen bezala, azpieremuaren behe-muga eta goi-muga zehaztuko dira bi puntuz banaturik (ikus `AzpieremuakZooan` programa).

```
PROGRAM AzpieremuakZooan ;                    { \TP70\08\DATU_MO8.PAS }
TYPE
  DM_Zoo = (Kaiman, Krokodrilo, Suge, Elefante, Hartz,
            Lehoi, Tigre, Orein, Arrano, Ostruka, Sai) ;
  DM_Narrasti = Kaiman..Suge ;
  DM_Ugaztun = Elefante..Orein ;
  DM_Hegazti = Arrano..Sai ;
VAR
  Animali : DM_Ugaztun ;
BEGIN
   Animali := Orein ;
   WriteLn ('Bambi-ri dagokion ordinala: ', Ord(Animali)) ;

  {                                    (* ezinezkoa *)
   Animali := Kaiman ;
   WriteLn ('Joantxo-ri dagokion ordinala: ', Ord(Animali)) ;
  }

  {                                    (* ezinezkoa *)
   Write ('Pizti bat eman: ') ;
   ReadLn (Animali) ;
  }
END.
```

`AzpieremuakZooan` programan lehenik `DM_Zoo` datu-mota enumeratua eraikitzen da, animalien zerrenda egitean taldeka bildu direnez, ondoren `DM_Zoo` ostalaritzat harturik `DM_Narrasti`, `DM_Ugaztun` eta `DM_Hegazti` azpieremuak sortzen dira. Ugaztuna izango den `Animali` aldagaiarekin konparaketak esleipenak egin daitezke, zer esanik ez ordinalen funtzio estandarrak (`Ord()`,`Pred()` eta `Succ()`) aplikatu ahal izatea.

`AzpieremuakZooan` programaren hiru zatiak azter ditzagun:

1. Esleipena eta `Ord()` funtzioa daudenez ez da arazorik sortuko.

   ```
   Animali := Orein ;
   WriteLn ('Bambi-ri dagokion ordinala: ', Ord(Animali)) ;
   ```

2. Esleipena eta `Ord()` funtzioa daudenez ez litzateke arazorik sortu beharko, baina iruzkinak kenduz zati hau ezekutagarria jarriko bagenu bagenitu honelako mezua agertuko litzaiguke: `Error 76: Constant out of range.`

   ```
   Animali := Kaiman ;
   WriteLn ('Joantxo-ri dagokion ordinala: ', Ord(Animali)) ;
   ```

3. Baliogatu dagoen hirugarren zatian teklatuaren bitarteko irakurketa bat egin nahi da. Azpieremu datu-motek ez dute datu-mota enumeratuek agertzen duten sarrera/irteera arazoa baldin eta oinarria den ostalari datu-motak sarrera/irteerak onartzen baditu. `AzpieremuakZooan` programaren kasua, tamalez, ugaztunen azpieremua den `Animali` aldagaiari dagokion datu-mota ostalaria enumeratua itatean honelako errore mezua aterako zaigu: `Error 64: Cannot Read or Write variables of this type.`

   ```
   Write ('Pizti bat eman: ') ;
   ReadLn (Animali) ;
   ```

Azpieremu datu-mota aldagai bat teklatuz irakurtzen duen `OrdutegiaDatuMotekin` programa erakusten da jarraian. `Irakurgaia` aldagaiak zenbaki osoak osalatariak dituenez balioa har dezake teklatuz, ondoren enumeratua den `LanEgun`-ari irakurketaren arabera dagokiona ezarri behar zaio; guk, `FOR-DO` egitura errepikakorraren bitartez egin dugu (aintzat hartu `LanEgun`-aren hasieraketa).

```
PROGRAM OrdutegiaDatuMotekin ;                { \TP70\08\DATU_MO9.PAS }
TYPE
  DM_Astegun = (Astelehen, Astearte, Asteazken, Ostegun, Ostiral) ;
VAR
  Irakurgaia : 1..5 ;
  LanEgun : DM_Astegun ;
  Kontagailu : Byte ;
BEGIN
   REPEAT
      Write ('Eguna eman (asth=1, astr=2, astz=3, ostg=4, ostr=5) ---> ') ;
      ReadLn (Irakurgaia) ;
   UNTIL (Irakurgaia >= 1) AND (Irakurgaia <= 5) ;

   LanEgun := Astelehen ;
   FOR Kontagailu:=2 TO Irakurgaia DO
      LanEgun := Succ(LanEgun) ;

   WriteLn ('Ordutegia:') ;
   WriteLn ('---------') ;
   CASE LanEgun OF
         Astelehen : WriteLn ('Astelehen: Fisika, Algebra, Marrazketa') ;
         Astearte : WriteLn ('Astearte:  Kimika, Kalkulu, Informatika') ;
         Asteazken : WriteLn ('Asteazken: Algebra, Marrazketa, Kimika') ;
         Ostegun : WriteLn ('Ostegun:   Kalkulu, Informatika, Algebra') ;
         Ostiral : WriteLn ('Ostiral:   Informatika, Marrazketa') ;
   END ;
END.
```

`OrdutegiaDatuMotekin` exekutatuz:

```
Eguna eman (asth=1, astr=2, astz=3, ostg=4, ostr=5) ---> 4
Ordutegia:
---------
Ostegun:   Kalkulu, Informatika, Algebra
_
```

8.5.1 Azpieremuak eta heina. {$R±} konpilazio direktiba

`OrdutegiaDatuMotekin` programari aldaketa txiki bat eragingo diogu. Azpieremua den `Irakurgaia` aldagaia teklatutik irakutzean `REPEAT-UNTIL` mugatzen da, balioak `1` eta `5` artean egon daitezen. Laugarren kapituluan konpiladorearen direktiba bat aipatu genuen, iruzkin itxura izan arren konpiladoreari ematen zaion agindu berezia da konpiladorearen direktiba.

`OrdutegiaDatuMotekin` programari eragingo diogun aldaketa `{$R+}` direktibarekin loturik dago, horren bitartez konpilatzean sortzen den kodea azpieremuen heina kontutan izanten da sarrerak egiten direnean. `OrdutegiaDatuMotekin` programan `{$R+}` direktiba jarri eta `REPEAT-UNTIL` egitura kenduz `KonpilazioDirektiba` programa lortu dugu.

`KonpilazioDirektiba` programa deskriba dezagun. Sententzien atala heinaren kontrolerako `{$R+}` direktibarekin hasten da. Ondoren `Irakurgaia` aldagaiari `123` konstantea esleitzen zaio, `{$R+}` direktibari esker balio hori `1..5` azpieremutik at dagoela oharturik programak honelako errore mezua erakusten du konpilatzean:

```
Error 76: Constant out of range.
```


Beraz, `KonpilazioDirektiba` programa konpilatu ahal izateko egin beharreko lehen zeregina hasierako esleipen hori indargabetzea izango da. Bigarren sententzia beste esleipen bat da, `1` konstantea gordetzen da `Irakurgaia` aldagaian, baina oraingoan heina barnean dagoenez ez da konpilazio denboran arazorik detektatzen. Hirugarren sententzia espresio aritmetiko baten bitartez `Irakurgaia` aldagaiari `1+9` balioa ezartzen zaio, heinetik kanpo dagoena nabaria denez, hala ere programa konpilatzean horren bistakoa den errore hori ez da detektatzen. Baina `KonpilazioDirektiba` programa egikaritzean exekuzio denboran `{$R+}` direktibari esker errorea suertatzen da eta honako mezua agertuko da:

```
Error 201: Range check error.
```

Hona hemen `KonpilazioDirektiba` programa:

```
PROGRAM KonpilazioDirektiba ;                { \TP70\08\DATU_M10.PAS }
TYPE
  DM_Astegun = (Astelehen, Astearte, Asteazken, Ostegun, Ostiral) ;

VAR
  Irakurgaia : 1..5 ;
  LanEgun : DM_Astegun ;
  Kontagailu : Byte ;

BEGIN
{$R+}
   Irakurgaia := 123 ;                 { errorea konpilazio denboran }
   Irakurgaia := 1 ;
   Irakurgaia := Irakurgaia + 9 ;      { errorea exekuzio denboran }

   Write ('Eguna eman (asth=1, astr=2, astz=3, ostg=4, ostr=5) ---> ') ;
   ReadLn (Irakurgaia) ;

   LanEgun := Astelehen ;
   FOR Kontagailu:=2 TO Irakurgaia DO
      LanEgun := Succ(LanEgun) ;

   WriteLn ('Ordutegia:') ;
   WriteLn ('---------') ;
   CASE LanEgun OF
         Astelehen : WriteLn ('Astelehen: Fisika, Algebra, Marrazketa') ;
         Astearte : WriteLn ('Astearte:  Kimika, Kalkulu, Informatika') ;
         Asteazken : WriteLn ('Asteazken: Algebra, Marrazketa, Kimika') ;
         Ostegun : WriteLn ('Ostegun:   Kalkulu, Informatika, Algebra') ;
         Ostiral : WriteLn ('Ostiral:   Informatika, Marrazketa') ;
   END ;
END.
```

`{$R+}` direktiba indarrean dagoenean programa astiroago exekutatzen da. Programa bat garatzen den bitartean komeni da heina kontrolatzen duen direktiba jartzea, frogak egin ondoren `REPEAT-UNTIL` bezalako iragazkien mugak erabat zehazturik egongo dira eta behin betiko programan `{$R+}` direktiba kendu edo baliogabetuko da `{$R-}` idatziz.

8.6 DATU-MOTA EGITURATUEN SARRERA

Laugarren kapituluan (**4.6.1** eta **4.6.7** puntuen artean) aldagai egituratuak aipatu egin ziren baina datu-mota berezirik erabili gabe. Jarraian datozen **8.6.1** eta **8.6.7** puntuen artean erabiltzaileren datu-mota egituratuak nola definitzen diren erakusten da.

8.6.1 STRING datu-mota egituratua

`String` datu-motako aldagaiak hitzak edo esaldiak gordetzeko balio du, string datu-motako aldagaia definitzean memorian zenbat karaktere biltegitu nahi den zehaztu beharra dago. Horregatik string aldagaiari dagokion datu-mota sortzean goimuga eman beharko da.

`DM_KateLabur` eta `DM_KateLuze` deituko diren datu-motak honela definituko dira:

```
TYPE
   DM_KateLabur = String[10] ;
   DM_KateLuze =  String[999] ;
```

`DM_KateLuze`-ren goimuga `1` eta `255` bitartean ez dagoenez, konpilazio denboran honelako mezua agertuko da:

```
Error 25: Invalid string length.
```

Behin `DM_KateLabur` eta `DM_KateLuze` datu-mota esplizitoak definitu ondoren horren aldagaiak deklara daitezke, eta azpiprogrametara parametro bezala pasatu ahal izango dira.

8.6.2 ARRAY datu-mota egituratua

Array aldagai bat definitzean honela egitean, ezingo da azpiprogrametan parametro bezala erabili:

```
VAR
   Soldatak : ARRAY [1..100] OF Real ;
```

Horregatik hobe da lehendik datu-mota berezi bat sortu eta ondoren aldagaiak deklaratu:

```
TYPE
   DM_Irabaziak = ARRAY [1..100] OF Real ;;

VAR
   Soldatak : DM_Irabaziak ;
```

Array-ak hamargarren kapituluan aztertuko dira.

8.6.3 RECORD datu-mota egituratua

Hau litzateke `Langile` erregistro aldagaiaren definizioa:

```
VAR
   Langile :  RECORD
                  Izena : String ;
                  Maila : Char ;
                  Dirua : Real ;
                  Adina : Byte ;
              END ;
```

`Langile` aldagai hori, egituratua delako, ezingo da azpiprogrametan parametro bezala erabili. Horregatik hobe da lehenik erregistro datu-mota bat sortzea eta ondoren aldagaia, hau da `TYPE` klausula batekin hasiko gara eta datu-mota ezaguna denean aldagaia deklaratzeko `VAR` klausularekin jarraituko dugu.

Honelaxe, lehenik `DM_Pertsona` datu-mota berria sortu eta ondoren `DM_Pertsona` datu-motako behar diren aldagaiak erazagutu:

```
TYPE
   DM_Pertsona =  RECORD
                     Izena : String ;
                     Maila : Char ;
                     Dirua : Real ;
                     Adina : Byte ;
                  END ;

VAR
   Langile : DM_Pertsona ;
```

Erregistroak hamaikagarren kapituluan aztertuko dira.

8.6.4 SET datu-mota egituratua

`Set` datu-motari multzoa esaten zaio eta ordinalak diren elementuz osaturik dago (adibidez `Byte` edo `Char` datu-motako osagaiaz osaturik egon daiteke). Erabiltzaileak enumeraturiko datu-mota izan daiteke `Set`-aren oinarria, horrlakoetan ezin izango dira 256 osagai baino gehiago. `Set` datu-mota sortzeko `Type` klausulaz egingo da:

```
TYPE
   DM_Astea = (Astlh, Astrt, Astzk, Ostgn, Ostrl, Larunb, Igand) ;
   DM_AsteBurua = Set OF DM_Astea ;
   DM_Minuskulak = Set OF Char ;
   DM_Digituak = Set OF 0..9 ;
   DM_Bakoitiak = Set OF Byte ;
```

Multzo definitu ondoren bere datu-motako aldagaiak deklara daitezke:

```
VAR
   Eguna : DM_AsteBurua ;
   Letra : DM_Minuskulak ;
   Adina : DM_Bakoitiak ;
```

Hamaikagarren kapituluan ikusiko denez, multzoari balioak emateko kortxeteen bitartez egiten da:

```
   Eguna := [Larunb, Igand] ;
   Letra := [] ;
   Adina := [1, 3, 5, 7, 9, 11, 13, 15, 17] ;
```

8.6.5 FILE datu-mota egituratua

Fitxategi aldagaiak definitzean honelako zerbait egin genuen **4.6.5** puntuan:

```
VAR
   Zenbakiak : FILE OF Byte ;
   Izenak : FILE OF String ;
```

Baina fitxategiaren oinarrizko elementua egituratua denean derrigorrez `Type` klausulaz baliatu beharko gara. Hona hemen `DM_Pertsona` erregistroa oinarritzat duen fitxategi datu-motaren sorrera eta berari dagokion `Zerrenda` aldagaia:

```
TYPE
   DM_Pertsona =  RECORD
                  |  Izena : String ;
                  |  Maila : Char ;
                  |  Dirua : Real ;
                  |
```

```
                  |
                  |  Adina : Byte ;
                 END ;
   DM_Fitxategi = File OF DM_Pertsona ;

VAR
   Zerrenda : DM_Fitxategi ;
```

8.6.6 Erakusle datu-mota egituratua

Erakusle bat oinarrizko datu-mota bat kontutan izango du. Aurreko **8.6.5 FILE datu-mota egituratua** puntuan fitxategi aldagaiak definitu beharrean erakusleak nahi badira honelako erazagupena egingo litzateke:

```
VAR
   Zenbakiak : ^ Byte ;
   Izenak : ^ String ;
```

Baina erakuslearen oinarrizko elementua egituratua denean `Type` klausulaz baliatu gara. Hona hemen `DM_Pertsona` erregistroa oinarritzat duen erakusle datu-motaren sorrera eta berari dagokion `Erak` aldagaia:

```
TYPE
   DM_Pertsona =  RECORD
                     Izena : String ;
                     Maila : Char ;
                     Dirua : Real ;
                     Adina : Byte ;
                  END ;
   DM_Erakusle = ^ DM_Pertsona ;

VAR
   Erak : DM_Erakusle ;
```

Erakuslea memoria dinamikoa erabiltzeko datu-mota aproposa da.

Erakusle datu-motaren batera memoriaren banaketa eta memoria dinamikoaren kontzeptuak hamahirugarren kapituluan ikusiko dira. Ostean, hamahirugarren kapituluan ere, egitura dinamikoak (zerrenda kateatuak, pilak, ilarak eta zuhaitzak) lantzeko aukera izango dugu.

8.6.7 Objektu datu-mota egituratua

Laugarren kapituluan esan izan denez `Object` datu-motek erregistroen antza dute haiek bezala eremutan banatzen direlako, objektuek eremuaz gain metodoak ere barneratzen dituzte.

```
TYPE
   DM_Ordua = OBJECT
                Ordu : Byte ;
                Minutu : Byte ;
                Segundo : Byte ;
                PROCEDURE Hasieraketa (oo, mm, ss : Byte) ;
                FUNCTION OrduaKateBezala : String ;
              END ;

VAR
   Noiz : DM_Ordua ;
```

`Object` datu-motaren kontzeptuetan hamalaugarren kapituluan sakonduko dugu.

8.7 KONPILADOREAREN DIREKTIBAK

Konpiladorearen direktiben bitartez konpiladoreak ekoizten duen kodearen zenbait ezaugarri kontrola daiteke. Konpiladorearen direktibak jartzeko iturburu-programan iruzkin bereziak tartekatu behar dira, edo bestela konpiladorearen ingurune integratuaren `Options` komandoaren bitartez.

Direktibak iruzkinen bitartez jartzeko **8.5.1 Azpieremuak eta heina. {$R±} konpilazio direktiba** izeneko puntua eta `OrdutegiaDatuMotekin` delako programa gogoratzea komeniko zaigu, bertan ikus daitekeenez konpilazio direktiba `$` karakterez hasten den iruzkina da.

Hiru motatako direktibak daude:

1. Switch edo konmutadore direktibak
2. Parametrodun direktibak
3. Baldintza-direktibak

8.7.1 Konmutadore direktibak

Switch edo konmutadore direktibak konpiladorearen lan egiteko modua aktibatu edo desaktibatzeko balio dute. Konpiladorearen ezaugarriren bat indarrean jartzeko `+` sinboloa jarriko da, eta, baliogabetzeko berriz `-` sinboloa idazten da iruzkinaren barruan. Konmutadore direktibarik jartzen ez bada bi balioetatik bat indarrean egongo da, hau da, balio lehenetsia[1] dute switch direktibek.

Ikusiko ditugun konmutadore direktibak honako hauek dira:

1. `{$R±}` heinaren froga burutzen duen direktiba
2. `{$B±}` boolearren ebaluazio osoa bermatzen duen direktiba
3. `{$Q±}` gainezkada kontrolatzen duen direktiba
4. `{$I±}` sarrera/irteeraren froga burutzen duen direktiba
5. `{$V±}` karaktere-kateak kontrolatzen dituen direktiba
6. `{$P±}` parametro “irekiak” onartzeko/eragozteko direktiba
7. `{$X±}` sintaxi hedatua onartzeko/eragozteko direktiba
8. `{$A±}` byte eta bi byteteko aldagaien memori erreserbak

Hauetaz gain badira beste switch direktiba batzuk, informazio gehiago lortzeko Turbo Pascal lengoaiak eskaintzen duen laguntza sistema erabil daiteke. Direktibak taldeka jartzeko aukera dago, horretarako komaz banatuko dira, esate baterako: `{$R+,I-,P+,B-}`

8.7.1.1 {$R±} direktiba

Esan den bezala, azpieremuak tratatzean (**8.4.1** puntuan) aipatu izan dugu direktiba hau. Heinaren direktiba `{$R+}` denean array-en, karaktere-kateen eta azpieremuen indizeak konprobatzen dira.

[1] Balio lehenetsia direktiba berarekin definiturik dago. Horregatik, direktiba batek duen balio lehenetsia `+` (aktibatua) izan daiteke eta beste batek duen balio lehenetsia `-` (indargabetua) izan daiteke.

Hona hemen bere ezaugarriak:

Sintaxia:	`{$R+}` edo `{$R-}`
Balio lehenetsia:	`{$R-}`
Menu komandoa:	`Options` | `Compiler` | `Range checking`

Jarraian array baten indexazioa kontrolatzen duen programa erakusten da:

```
PROGRAM R_KonpilazioDirektiba ;                    { \TP70\08\DIREKT1.PAS }
TYPE
  DM_Zerrenda = Array [1..10] OF Integer ;
CONST
  ZBK = 12345 ;
VAR
  Zenbakiak : DM_Zerrenda ;
  Kontagailu : Byte ;
BEGIN
 { Hasieran R direktiba indargabetua. Balio lehenetsia - delako }
   Kontagailu := 99 ;
   Zenbakiak [Kontagailu] := ZBK ;       { errorerik ez da detektatzen }
 {$R+}
   Kontagailu := 99 ;
   Zenbakiak [Kontagailu] := ZBK ;       { errorea exekuzio denboran }
 {$R-}
   Kontagailu := 99 ;
   Zenbakiak [Kontagailu] := ZBK ;       { errorerik ez da detektatzen }
END.
```

8.7.1.2 {$B±} direktiba

`{$B±}` direktibaren bitartez `AND` eta `OR` eragile boolearrak programak nola aplikatuko diren erabakitzen da. Hona hemen bere ezaugarriak:

Sintaxia:	`{$B+}` edo `{$B-}`
Balio lehenetsia:	`{$B-}`
Menu komandoa:	`Options` | `Compiler` | `Boolean evaluation`

`{$B+}` direktibarekin (indarrean dagoenean) programan aurkitzen diren adierazpen boolearrak *osoan* ebaluatzen dira, horretarako konpiladoreak kode aproposa gehituko dio iturburu-programari. Nahiz eta adierazpen boolearraren azken balioa ezaguna izan bere atal guztiak banan banan ebaluatuko dira.

`{$B-}` direktibarekin (indargabeturik dagoenean) programaren adierazpen boolearrei dagokien azken balioa ebidentea egiten denean adierazpenaren ebaluazioari utzi egiten dio. Adibidez, `{$B-}` direktibarekin, `fHitzGako(Giltza)` funtzio boolearra izanik `IF-THEN` hauetan `fHitzGako(Giltza)` funtzioaren deia ez da sekulan egingo:

```
IF False AND fHitzGako(Giltza) THEN
BEGIN
    ...
END ;
```

```
IF True OR fHitzGako(Giltza) THEN
BEGIN
    ...
END ;
```

8.7.1.3 {$Q±} direktiba

Gainezkada suertatzen den konprobatzeko balio duen direktiba.

Hona hemen bere ezaugarriak:

Sintaxia:	`{$Q+}` edo `{$Q-}`
Balio lehenetsia:	`{$Q-}`
Menu komandoa:	`Options | Compiler | Overflow checking`

`{$Q+}` direktibarekin (indarrean dagoenean) programan egiten diren osoen arteko eragiketak konprobatzen dira, testatu egiten diren eragiketak hauek dira: `+`, `-`, `*`, `Abs()`, `Sqr()`, `Succ()` eta `Pred()`. Eragiketa hauen kodeari gainezkada testatzeko gehigarriaz hornitzen da, heinetik kanpo joanez gero programa gelditzen errore mezua erakutsiz. `{$Q+}` direktiba hau indarrean jartzen denean, askotan `{$R+}` direktibarekin batera erabiltzen da.

`{$Q+}` direktibak programa exekutagarria handiagoa eta motelagoa izatea eragiten duenez, aplikazioaren garatze prozesuan komeni da baina behin betiko programan indargabetuko litzateke.

8.7.1.4 {$I±} direktiba

Sarrera eta irteera prozedura estanderretan konpiladoreak erroreak konprobatzen ditu, erroreren bat suertatzean dagokion mezua erakutsi eta programa gelditzen da. Esate baterako, ondoko `I1_KonpilazioDirektiba` delako programan `Zbk` aldagaiak zenbaki oso bat itxaroten du eta `ReadLn(Zbk)` egitean letra bat ematen badugu programaren erantzuna honako hau da:

```
Zenbaki oso bat eman beharrean letra bat sartu: M
Run time error 106 at 0BEF:0073
_
```

`I1_KonpilazioDirektiba` programaren hasieran `{$I }` direktiba indarrean dago eta `Write()` sententziaren bitartez mezu bat pantailaratzenm da inolako arazorik detektatzen ez dela, baina `ReadLn(Zbk)` den bigarren sententzia exekutatzean sarrera-errorea gerta daiteke programa abortatuz. Gauzak aldatu egiten dira direktiba horrek `{$I-}` bitartez indargabeturik dagoenean:

```
PROGRAM I1_KonpilazioDirektiba ;                    { \TP70\08\DIREKT2.PAS }
VAR
  Zbk : Integer ;
BEGIN
 {}     { Hasieran I direktiba indarrean dago. Balio lehenetsia + delako }
   Write ('Zenbaki oso bat eman beharrean letra bat sartu: ') ;
   ReadLn (Zbk) ;                        { errorea exekuzio denboran }
 {$I-}
   Write ('Zenbaki oso bat eman beharrean letra bat sartu: ') ;
   ReadLn (Zbk) ;                        { errorerik ez da detektatzen }
 {$I+}
   WriteLn ('Zbk: ', Zbk) ;              { errorea exekuzio denboran }
END.
```

`I1_KonpilazioDirektiba` programarekin Turbo Pascal ingurune integratuan lanean ari bagara, errorea eragin duen `ReadLn(Zbk)` edo `WriteLn(Zbk)` sententzian kokatuko da kurtsorea, sarrera/ireera azpirrutinek arazoren mezu hau erakutsiz:

```
Error 106: Invalid numeric format.
```

Turbo Pascal lengoaiaren aurredefinituriko `IOresult` funtzioak sarrera/ireera azken eragiketaren balioa itzultzen du. Beraz, `I1_KonpilazioDirektiba` programan `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.

Ikus dezagun `I2_KonpilazioDirektiba` izeneko programa zeinean `{$I-}` direktiba dagoelako `Zbk` gaizki irakurtzean abortatzen ez den. Arazoa egon arren programa gelditzen ez bada logikoena programadoreak okerreko egoera hori programadoreak tratatzea, hori da, hain zuzen ere, `IOresult` funtzioaren zeregina.

```
PROGRAM I2_KonpilazioDirektiba ;                    { \TP70\08\DIREKT3.PAS }
VAR
  Zbk, ErroreKode : Integer ;
BEGIN
 {$I-}
   Write ('Zenbaki oso bat eman beharrean letra bat sartu: ') ;
   ReadLn (Zbk) ;                          { errorerik ez da detektatzen }
   ErroreKode := IOresult ;

   IF ErroreKode <> 0 THEN
   BEGIN
     WriteLn ('Arazoak Zbk irakurtzean.') ;
     WriteLn ('Errore kodea ---> ', ErroreKode) ;
   END ;
END.
```

`I2_KonpilazioDirektiba` programan, zenbaki oso bat itxaroten duen `Zbk` aldagairi `M` letra teklatuz esleitzen bazaio, `{$I-}` direktibari esker errorea gertatu arren exekuzioa ez da eteten. Hala ere, programadorea `IOresult` funtzioaren emaitza aztertuz arazoaz kontura daiteke:

```
Zenbaki oso bat eman beharrean letra bat sartu: M
Arazoak Zbk irakurtzean.
Errore kodea ---> 106
_
```

Zenbakia ondo sartzean ez da mezurik agertzen:

```
Zenbaki oso bat eman beharrean letra bat sartu: 62
_
```

`{$I±}` direktibak fitxategiekin[2] berebiziko garrantzia du eta hamabigarren kapituluan berriro aipatu beharko dugu. Hona hemen, laburki, bere ezaugarriak:

Sintaxia:	`{$I+}` edo `{$I-}`
Balio lehenetsia:	`{$I+}`
Menu komandoa:	`Options` | `Compiler` | `I/O checking`

8.7.1.5 {$V±} direktiba

`{$V±}` direktibak karaktere-kateak parametro bezala pasatzen direnean lan egiten du, uneko parametroak eta parametro formalak duten luzera fisikoa kontrolatzen du. Karaktere-kateak bederatzigarren kapituluan ikusiko denez, orduan hobeto ulertu ahal izango da jarraian erakusten den `V1_KonpilazioDirektiba` programa:

[2] Fitxategi batekin lan egiteko lehendik existitzen dela bermatu beharra dago, eta froga hori burutzeko `{$I±}` direktiba erabiltzen da.

```
PROGRAM V1_KonpilazioDirektiba ;                    { \TP70\08\DIREKT4.PAS }
TYPE
  DM_KateLuze = String [100] ;
  DM_KateMotz = String [25] ;
VAR
  Helbide : DM_KateLuze ;
  Abizen1 : DM_KateMotz ;
  Abizen2 : DM_KateMotz ;

PROCEDURE KateLuzeaIrakur (VAR Katea : DM_KateLuze) ;
BEGIN
   Write ('Kate bat eman: ') ;
   ReadLn (Katea) ;
END ;

BEGIN
 {}     { Hasieran V direktiba indarrean dago. Balio lehenetsia + delako }
   KateLuzeaIrakur (Abizen1) ;             { errorea konpilazio denboran }
 {$V-}
   KateLuzeaIrakur (Abizen2) ;             { errorerik ez da detektatzen }
 {$V+}
   KateLuzeaIrakur (Helbide) ;             { uneko parametroa egokia da }
END.
```

`V1_KonpilazioDirektiba`-ren arazoak ez du `Helbide` aldagaiarekin zerikusirik, izan ere `{$V±}` direktiba edozein modutan jarririk `Katea` parametro formala eta `Helbide` uneko parametroa `DM_KateLuze` baitira. Baina `Abizen1` edo `Abizen2` aldagaiak `Katea` parametro formalarekiko parakagarriak ez direnez, konpilazio errorerik[3] gertatu ez dadin `{$V-}` idatzi beharko dugu.

Hona hemen `{$V±}` direktibaren ezaugarriak:

Sintaxia:	`{$V+}` edo `{$V-}`
Balio lehenetsia:	`{$V+}`
Menu komandoa:	`Options` | `Compiler` | `Strict var-strings`

Dena den ez da komeni `{$V-}` direktiba erabiltzea, adibiderako programa berbera pixka bat aldatu dugu eta birritan exekutatu. Lehedabiziko exekuzioan emaniko datuen luzeren arabera ez da errorerik sortzen (emaitza erabat zuzena[4] ez bada ere). Bigarrenean berriz, `Abizen1` aldagaian gordeko den deitura luzeegia denez `V2_KonpilazioDirektiba` izeneko programak `Abizen1` karaktere-katea testu-fitxategi batekin nahasten du.

`V2_KonpilazioDirektiba` programa:

```
PROGRAM V2_KonpilazioDirektiba ;                    { \TP70\08\DIREKT5.PAS }
TYPE
  DM_KateLuze = String [12] ;
  DM_KateMotz = String [5] ;
VAR
  Helbide : DM_KateLuze ;
  Abizen1 : DM_KateMotz ;

PROCEDURE KateLuzeaIrakur (VAR Katea : DM_KateLuze) ;
BEGIN
   Write ('Kate bat eman: ') ;
   ReadLn (Katea) ;
END ;
```

[3] Mezuak honela dio → `Error 26: Type mismatch.`

[4] Bederatzigarren kapituluan azalduko da zergatik helbidearen pantailaraketan `Herriko Plaz` baino ez den agertzen, dagoenekoz `Helbide` karaktere-katearen `12`-ko luzerarekin zerikusia duela esango dugu.

```
BEGIN
 {$V-}
   KateLuzeaIrakur (Abizen1) ;
   KateLuzeaIrakur (Helbide) ;            { errorerik ez da detektatzen }
   WriteLn ;
   WriteLn('Abizen1-->', Abizen1);
   Writeln('Helbide-->', Helbide);
END.
```

`V2_KonpilazioDirektiba`-ren lehen exekuzioa:

```
Kate bat eman: Imaz
Kate bat eman: Herriko Plaza 2
Abizen1-->Imaz
Helbide-->Herriko Plaz
_
```

`V2_KonpilazioDirektiba`-ren bigarren exekuzioan errorea[56] eragiten da `Salazar` datua `Abizen1` aldagaian ezin delako gorde:

```
Kate bat eman: Salazar
Kate bat eman: Runtime error 006 at 0BFE:0053
_
```

`{$V±}` direktiba, egungo Turbo Pascal 7.0 bertsioan oraindik mantentzen da baina aurrerantzean merkaturatuko diren bertsioetan, dirudienez, jarraian ikasiko dugun `{$P±}` direktibak ordezkatuko du.

8.7.1.6 {$P±} direktiba

`{$P±}` direktibak ere karaktere-kateak parametro bezala pasatzen direnean lan egiten du. Honen bitartez karaktere-kate baten parametro formala, luzera maximoa duen, string generiko bat izan dadila onartzen da.

`{$P+}` direktibarekin (indarrean dagoenean) uneko parametroa edozein luzerako karaktere-katea izanik ez da errorerik gertatzen baldin eta parametro formala `String` balitz.
`{$P-}` direktibarekin (indargabeturik dagoenean) programan dauden karaktere-kateen uneko parametroak dagokien aparametro formalekin erabat parekagarriak izan beharko dira konpilazio errorerik gertatu ez dadin. `{$P-}` direktiba azpirrutina baino lehen idatziko da.

Hona hemen `{$P±}` direktibaren ezaugarriak:

Sintaxia:	`{$P+}` edo `{$P-}`
Balio lehenetsia:	`{$P-}`
Menu komandoa:	`Options` | `Compiler` | `Open parameters`

`V2_KonpilazioDirektiba` programan `{$V-}` direktiba `{$P+}`-gatik ordezkatuz eta datu luzeegiak emanez gero errorerik ez da gertatzen (kateak, hori bai, moztuta gordeko dira memorian). Ikus `P_KonpilazioDirektiba` programa:

[5] Mezuak honela dio → `Error 6: Invalid file handle.`
[6] Edo bestela honela → `Error 104: File not open for input.`

```
PROGRAM P_KonpilazioDirektiba ;                    { \TP70\08\DIREKT6.PAS }
TYPE
  DM_KateLuze = String [12] ;
  DM_KateMotz = String [5] ;
VAR
  Helbide : DM_KateLuze ;
  Abizen1 : DM_KateMotz ;

{$P+}
PROCEDURE KateBatIrakur (VAR Katea : String) ;
BEGIN
   Write ('Kate bat eman: ') ;
   ReadLn (Katea) ;
END ;

BEGIN
   KateBatIrakur (Abizen1) ;            { errorerik ez da detektatzen }
   KateBatIrakur (Helbide) ;            { errorerik ez da detektatzen }
   WriteLn ;
   WriteLn('Abizen1-->', Abizen1);
   Writeln('Helbide-->', Helbide);
END.
```

`P_KonpilazioDirektiba` programa exekutatzean:

```
Kate bat eman: Salazar
Kate bat eman: Herriko Plaza 2
Abizen1-->Salaz
Helbide-->Herriko Plaz
_
```

8.7.1.7 {$X±} direktiba

`{$X±}` direktibaren bitartez funtzioen deiak prozedurak baitira egin daitezke (funtzioak itzultzen duen emaitza ez da aintzat hartuko), direktiba honek `#0` karakterez amaitutako karaktere-kateak erabiltzea ahalbidetzen du (ikus konpiladorearen laguntzan `Strings` unitate estandarreri buruzkoa esanikoa). `{$X±}` direktibari esker beste programazio lengoai batzuk dituzten sintaxia erabiltzeko aukera ematen duenez *sintaxi hedatua* onartzeko/eragozteko direktiba esaten zaio.

Hona hemen `{$X±}` direktibaren ezaugarriak:

Sintaxia:	`{$X+}` edo `{$X-}`
Balio lehenetsia:	`{$X+}`
Menu komandoa:	`Options` | `Compiler` | `Extended syntax`

8.7.1.8 {$A±} direktiba

Seigarren kapituluko **6.4 PARAMETRO MOTAK ETA MEMORI HELBIDEAK** puntuan helbideak bistaratzeko `AldagaiNagusienHelbideak` programa erakutsi izan zen. Programa horrek `Ikus` izeneko unitate bat darabil, zeinean `IntToHex()` funtzioa garaturik aurkitzen den, baina garrantzitsuena une honetan `AldagaiNagusienHelbideak` programaren irteera aztertzea da:

```
PROGRAMA NAGUSIAN EZAGUTZEN DIREN ALDAGAIEN MEMORI HELBIDEAK
============================================================

PILARI dagokion segmentuaren hasiera:  5008
DATUEI dagokien segmentuaren hasiera:  4FD9
KODEARI dagokion segmentuaren hasiera: 4EE5

Pilaren erakusleari dagokion memoria:  4FD9:001A


                     ALDAGAIA        HELBIDEA   TAMAINA
                -----------------------------------------

                   BesteErreala --->   4FD9:0070    (6)
                        ZbkByte --->   4FD9:006E    (1+1)
                      OsoaLuzea --->   4FD9:006A    (4)
                      Boolearra --->   4FD9:0068    (1+1)
                    ZbkErreala --->   4FD9:0062    (6)
                    Karakterea --->   4FD9:0060    (1+1)
                       ZbkOsoa --->   4FD9:005E    (2)
_
```

Ikus daitekeenez, programa nagusian lehenago deklaraturiko aldagaiek helbide baxuagoak dituzte azkenean erazagututakoek baino. Bestalde, zortzikote bakar bana beharko luketen `ZbkByte`, `Boolearra` eta `Karakterea` aladagaietarako egitan 2 zortzikote hartzen dira memorian, nahiz eta bigarrena erabili ez. `AldagaiNagusienHelbideak` programari `{$A-}` direktiba gehitu eta `+1` mezuak kenduta `A_KonpilazioDirektiba` programa lortuko genuke zeini honako irteera hau dagokion:

```
PROGRAMA NAGUSIAN EZAGUTZEN DIREN ALDAGAIEN MEMORI HELBIDEAK
============================================================

PILARI dagokion segmentuaren hasiera:  5008
DATUEI dagokien segmentuaren hasiera:  4FD9
KODEARI dagokion segmentuaren hasiera: 4EE5

Pilaren erakusleari dagokion memoria:  4FD9:001A


                     ALDAGAIA        HELBIDEA   TAMAINA
                -----------------------------------------
                   BesteErreala --->   4FD9:006D    (6)
                        ZbkByte --->   4FD9:006C    (1)
                      OsoaLuzea --->   4FD9:0068    (4)
                      Boolearra --->   4FD9:0067    (1)
                    ZbkErreala --->   4FD9:0061    (6)
                    Karakterea --->   4FD9:0060    (1)
                       ZbkOsoa --->   4FD9:005E    (2)
_
```

Ondorioz, `{$A-}` direktiba jartzean memoriaren aprobetxamendua hobea da, baina ordenadorearen mikroprozesadoreak azkarrago tratatzen ditu memori helbide bikoitietan hasten diren aldagaiak. Hona hemen `{$A±}` direktibaren ezaugarriak:

Sintaxia:	`{$A+}` edo `{$A-}`
Balio lehenetsia:	`{$A+}`
Menu komandoa:	`Options` | `Compiler` | `Word align data`

`{$A+}` direktibarekin (indarrean dagoenean) aldagai eta kostante guztien helbide bikoitietan hasten dira, nahiz eta byte hutsak tartekatu behar. Array eta erregistro bezalako datu mota egituratuak orokorrean kontsideratzen dira.

`{$A-}` direktibarekin (indargabeturik dagoenean) ez da aldagaien artean inolako byte hutsik tartekatzen eta aldagaiak erabili gabeko lehen memori posizioan jartzen dira.

8.7.2 Parametrodun direktibak

Ikusiko dugun parametrodun direktibak `{$I XXX}` eta `{$L XXX}` izango dira, direktiba hauek behar duten `XXX` parametroa fitxategi baten izena da. Baten kasuan fitxategi horrek iturburu-programa adierazten du, eta bestean aurretik konpilaturiko dagoen objektu-programa da.

8.7.2.1 {$I XXX} direktiba

Programa baten kodea fitxategi ezberdinetan badago euren irispideak konpiladoreari informatzea da `{$I XXX}` direktibaren helburua, bere sintaxia honako hau da:

```
{$I fitxategi_izen}
```

Non `fitxategi_izen` identifikadorea (direktibaren parametroa) barneratu nahi den fitxategiaren izena den. Nahi izanez gero, barneratu nahi den fitxategi izenaren aurrean disko unitatea eta direktorioak zehaz daitezke. Parametroak fitxategiaren izena solik adierazten badu, konpiladoreak fitxategia bilatzeko direktorio hauek aztertzen ditu:

- Hasteko, konpiladoreak direktorio lehenetsian bilatzen du barneratu behar duen fitxategia
- Aurreko saiakeran aurkitzen ez badu, Turbo Pascal 7.0 bertsioak duen ingurune integratuaren `Options | Directories | Include directories` kutxan markaturiko direktorioak aintzat hartuko ditu

Barneratu egiten den `fitxategi_izen` fitxategia konpilaturiko programan sartzen da, `{$I fitxategi_izen}` direktibak finkatzen duen tokian (direktibaren ordez `fitxategi_izen` fitxategi konpilatua kokatzen da). Barneratu egiten den `fitxategi_izen` fitxategiak berak ere beste `{$I fitxategi_izen1}` direktiba bat izan dezake, horrek esan nahi du barneratu behar diren fitxategiak kabia daitezkeela (gehien jota 15 mailatan kabiatuko dira).

Zenbaki oso bi teklatuz irakurri ondoren euren batura eta kendura programa idatzi dugu, baina iturburu-kodea hiru fitxategitan banatu dugu `DIREK8_0.PAS`, `DIREK8_1.PAS`, eta `DIREK8_2.PAS`. Ikusten denez `DIREK8_0.PAS` fitxategiak fitxategi nagusiaren papera jokatzen du, eta bertan beste biak barneratzen dira `{$I DIREK8_1.PAS}` eta `{$I DIREK8_2.PAS}` bitatez. Hona hemen `DIREK8_0.PAS` fitxategia:

```
PROGRAM I_KonpilazioDirektiba ;              { \TP70\08\DIREK8_0.PAS }
{$I DIREK8_1.PAS}
{$I DIREK8_2.PAS}
VAR
   Eragigai1, Eragigai2, Batura, Kendura : Integer ;

BEGIN
   Write ('Lehen eragigaia eman: ') ;
   ReadLn (Eragigai1) ;
   Write ('Bigarren eragigaia eman: ') ;
   ReadLn (Eragigai2) ;

   Batu (Eragigai1, Eragigai2, Batura) ;      { DIREK8_1.PAS fitxategian
garaturik }
   Kendu (Eragigai1, Eragigai2, Kendura) ;    { DIREK8_2.PAS fitxategian
garaturik }

   WriteLn (Eragigai1, ' + ', Eragigai2, ' = ', Batura) ;
   WriteLn (Eragigai1, ' - ', Eragigai2, ' = ', Kendura) ;
END.
```

Eta hauek DIREK8_1.PAS eta DIREK8_2.PAS fitxategiak lirateke:

```
{ DIREK8_1.PAS fitxategia }

PROCEDURE Batu (Eragigai1, Eragigai2 : Integer; VAR Batura : Integer) ;
BEGIN
   Batura := Eragigai1 + Eragigai2 ;
END ;
```

```
{ DIREK8_2.PAS fitxategia }

PROCEDURE Kendu (Eragigai1, Eragigai2 : Integer; VAR Kendura : Integer) ;
BEGIN
   Kendura := Eragigai1 - Eragigai2 ;
END ;
```

Konpilatzean eta exekutatzean ez da inolako arazorik sortzen hiru fitxategiak laneko direktorio lehenetsian baitaude. Ikusten denez DIREK8_0.PAS fitxategi nagusiak beste biak barneratzen ditu eta mailaketa bakarra dago adibide honetan. Ariketa bezala honako hau plantea daiteke: "DIREK8_0.PAS *fitxategi nagusiak bakarrik* DIREK8_1.PAS *fitxategia barnera dezala, eta honek* DIREK8_2.PAS *fitxategia barnera dezala*"

8.7.2.2 {$L XXX} direktiba

{$L XXX} direktibaren zeregina oraintsu ikusi den {$I XXX} direktibaren helburu bera da, baina barneratu behar den fitxategia testu-fitxategi izan beharrean konpilatuta dagoen objektu-fitxategi bat izango da. Adibidez mihiztadura-lengoaia batean idatziriko KALKUL.OBJ fitxategia barneratzeko hau idatziko litzateke:Parametrodun

```
{$L KALKUL.OBJ }
```

KALKUL.OBJ fitxategiari dagozkion sententziak {$L KALKUL.OBJ} direktibak finkatzen duen tokian linkatu edo estekatuko dira. Estekatzaileak KALKUL.OBJ fitxategia bilatu ahal izateko direktorio hauek aztertzen ditu:

- Hasteko, estekatzaileak direktorio lehenetsian bilatzen du linkatu behar duen objektu-fitxategia
- Aurreko saiakeran aurkitzen ez badu, Turbo Pascal 7.0 bertsioak duen ingurune integratuaren `Options | Directories | Object directories` kutxan markaturiko bideak aintzat hartuko ditu

8.7.3 Baldintza-direktibak

Zenbaitetan programa osoa konpilatu beharrean, programaren zati bat konpilatua izatea iteresatzen zaigu eta kodearen beste zati bat iruzkintzat hartua izan dadila. Hori lortzeko { eta }, edo, (* eta *) sinboloak egoki jartzea aski litzateke. Orain konpilazio baldintzatua ahalbidetzen dituen direktibak ikasiko ditugu.

Baldintza-direktibak zazpi dira:Baldintza-di

1. {$IFDEFINE etiketa} konstante sinboliko bat definitzeko
2. {$UNDEF etiketa} sinbolo baten definizioa desegiteko

3. `{$IFDEF etiketa}` jarraian dagoen kodea konpilatzen du baldin eta `etiketa` sinboloa definiturik badago
4. `{$IFNDEF etiketa}` jarraian dagoen kodea konpilatzen du baldin eta `etiketa` sinboloa definiturik ez badago
5. `{$IFOPT konmuta}` jarraian dagoen kodea konpilatzen du baldin eta `konmuta` konmutadore direktiba indarturik badago
6. `{$ELSE}` hau `{$IF...}` baten menpekoa da eta bere kontrakoa egingo du
7. `{$ENDIF}` konpilazio baldintzatuaren amaiera adierazteko

Konpilazio baldintzatuaren adibiderako programa pare bat eta bakoitzari dagokion irteera ikus dezagun:

```
PROGRAM KonpilazioBaldintzatua1 ;
VAR
   Izena, Abizen1, Abizen2, Herria :
                          String ;
{$DEFINE arazketa1}

BEGIN
   Write('Izena eman: ') ;
   ReadLn (Izena) ;
{$IFDEF arazketa1}
   Write ('1. abizena eman: ') ;
   ReadLn (Abizen1) ;
{$ELSE}
   Write ('2. abizena eman: ') ;
   ReadLn (Abizen2) ;
{$ENDIF}
   Write ('Herria eman: ') ;
   ReadLn (Herria) ;
   WriteLn ;
   WriteLn ('Izena---->', Izena)  ;
   WriteLn ('Abizen1-->', Abizen1) ;
   Writeln ('Abizen2-->', Abizen2) ;
   Writeln ('Herria--->', Herria) ;
END.
```

```
PROGRAM KonpilazioBaldintzatua2 ;
VAR
   Izena, Abizen1, Abizen2, Herria :
                          String ;
{$DEFINE arazketa1}

BEGIN
   Write('Izena eman: ') ;
   ReadLn (Izena) ;
{$IFNDEF arazketa1}
   Write ('1. abizena eman: ') ;
   ReadLn (Abizen1) ;
{$ELSE}
   Write ('2. abizena eman: ') ;
   ReadLn (Abizen2) ;
{$ENDIF}
   Write ('Herria eman: ') ;
   ReadLn (Herria) ;
   WriteLn ;
   WriteLn ('Izena---->', Izena)  ;
   WriteLn ('Abizen1-->', Abizen1) ;
   Writeln ('Abizen2-->', Abizen2) ;
   Writeln ('Herria--->', Herria) ;
END.
```

Ikusten denez `KonpilazioBaldintzatua1` eta `KonpilazioBaldintzatua2` programa biak berdintsuak dira, hasiera batean `arazketa1` izeneko etiketa definiturik dago eta horren arabera kodearen zati ezberdinak konpilatzen dira ala ez. Konpilazioari dagokiola programa bi horiek lau zati dituzte:

1. Programa nagusiaren lehengo sententziek `Izena` aldagaiari balio bat teklatuz emateko balio dute (mezu bat eta irakurketa bat). Horiek beti konpilatuko dira programa bietan `arazketa1` sinboloarekin zerikusirik ez baitute.
2. Bigarren zatia ezberdina da programa bakoitzean. `KonpilazioBaldintzatua1` delakoan `{$IFDEF arazketa1}` direktiba dagoenez, eta `arazketa1` izeneko etiketa dagoelako `Abizen1` aldagaiari balioa eman ahal izango da. Baina beste programan, `KonpilazioBaldintzatua2`-an, kontrako `{$IFNDEF arazketa1}` direktiba jarri denez `Abizen1` aldagaiari ezingo zaio baliorik eman, zatitxo hori konpilatzen ez delako.
3. `KonpilazioBaldintzatua1` eta `KonpilazioBaldintzatua2` programa bietan hirugarren zatia berbera da, baina `{$ELSE}` direktibaz baldintzaturik dagoenez aurreko bigarren zatiaren menpekoa da. Ondorioz, bigarren zatian konpilatzen zena orain ez da konpilatuko eta alderantziz. Beraz, `KonpilazioBaldintzatua1` programan `Abizen2` aldagaiari ezin zaio baliorik eman, eta beste programan berriz bai.
4. `KonpilazioBaldintzatua1` eta `KonpilazioBaldintzatua2` programa bietan laugarren zatia berbera da ere. Programa nagusiaren amaierako sententzia

hauek beti konpilatuko dira {$ENDIF} direktibak konpilazio baldintzatutua eteten baitu. Baina kontuz, Abizen1 eta Abizen2 aldagaiei loturiko sententziak konpilatuta ala konpilatu gabe egon daitezke KonpilazioBaldintzatua1 eta KonpilazioBaldintzatua2 programen arabera, eta ondorioz laugarren zatiak programa bakoitzean pantailaratuko duena ezberdina izango da.

Hona hemen KonpilazioBaldintzatua1 eta KonpilazioBaldintzatua2 programen irteerak elkar ondoan jarririk:

```
Izena eman: Fernando
1. abizena eman: Arretxe
Herria eman: Luzaide

Izena---->Fernando
Abizen1-->Arretxe
Abizen2-->
Herria--->Luzaide
_
```

```
Izena eman: Fernando
2. abizena eman: Kaminondo
Herria eman: Luzaide

Izena---->Fernando
Abizen1-->
Abizen2-->Kaminondo
Herria--->Luzaide
_
```

Baina ikus dezagun jarritako {$IFDEF arazketa1} direktibaz zehazturiko definizioa desegitean zer gertatzen den, horretarako KonpilazioBaldintzatua3 programa idatzi dugu. Programa horrek ere lau zati izango lituzke:

1. Beti konpilatuko diren pograma nagusiaren lehengo bi sententziak, Izena aldagaiari balioa emateko balio dutenak arazketa1 sinboloarekin zerikusirik ez dutenez beti konpilatuko dira.
2. Bigarren zatian abizenak eskatzen dira (lehenengo eta bigarren deitura eskatu eta teklatuz irakurtzen dira), zati hau {$IFDEF arazketa1} eta {$ENDIF} direktibek mugatzen dute. KonpilazioBaldintzatua3 adibide-programan arazketa1 izeneko etiketa dagoelako Abizen1 eta Abizen2 aldagai biei balioa eman ahal izango zaie.
3. Hirugarren zatia ere {$IFDEF arazketa1} eta {$ENDIF} direktiba bikoteak mugatzen du, baina lehentxoago {$UNDEF arazketa1} idatzi denez arazketa1 etiketaren definizioa desegin da eta ondorioz herria teklatzeatzo aukerarik ez dugu izango zati hau konpilatzen ez baita.
4. KonpilazioBaldintzatua3 programaren laugarren zatia aurrekoen berbera da ere, programa nagusiaren amaierako sententzia hauek beti konpilatuko dira {$ENDIF} direktibak konpilazio baldintzatutua eteten duelako.

Hauek lirateke KonpilazioBaldintzatua3 programaren kodea eta berari dagokion balizko irteera bat:

```
PROGRAM KonpilazioBaldintzatua3 ;            { \TP70\08\DIREKT11.PAS }
VAR
  Izena, Abizen1, Abizen2, Herria : String ;

{$DEFINE arazketa1}

BEGIN
   Write('Izena eman: ') ;
   ReadLn (Izena) ;

{$IFDEF arazketa1}
   Write ('1. abizena eman: ') ;
   ReadLn (Abizen1) ;
   Write ('2. abizena eman: ') ;
   ReadLn (Abizen2) ;
{$ENDIF}
```



```
{$UNDEF arazketa1}

{$IFDEF arazketa1}
   Write ('Herria eman: ') ;
   ReadLn (Herria) ;
{$ENDIF}

   WriteLn ;
   WriteLn ('Izena---->', Izena)  ;
   WriteLn ('Abizen1-->', Abizen1) ;
   Writeln ('Abizen2-->', Abizen2) ;
   Writeln ('Herria--->', Herria) ;
END.
```

```
Izena eman: Fernando
1. abizena eman: Arretxe
Herria eman: Luzaide

Izena---->Fernando
Abizen1-->Arretxe
Abizen2-->Kaminondo
Herria--->
_
```

Konpilazio baldintzatuarekin jarraitzeko `KonpilazioBaldintzatua4` adibide-programa azalduko dugu, fitxategi exekutagarriek duten tamaina ezberdina azpimarratzeko asmoz.

`KonpilazioBaldintzatua4` programa honelako zerbait da:

```
PROGRAM KonpilazioBaldintzatua4 ;             { \TP70\08\DIREKT12.PAS }
VAR
  Izena, Abizen1, Abizen2, Herria : String ;

{$DEFINE herriaBAI}

BEGIN
   Write('Izena eman: ') ;
   ReadLn (Izena) ;
   Write ('1. abizena eman: ') ;
   ReadLn (Abizen1) ;
   Write ('2. abizena eman: ') ;
   ReadLn (Abizen2) ;

{$IFDEF herriaBAI}
   Write ('Herria eman: ') ;
   ReadLn (Herria) ;
{$ENDIF herriaBAI}

   WriteLn ;
   WriteLn ('Izena---->', Izena)  ;
   WriteLn ('Abizen1-->', Abizen1) ;
   Writeln ('Abizen2-->', Abizen2) ;
{$IFDEF herriaBAI}
   Writeln ('Herria--->', Herria) ;
{$ENDIF herriaBAI}
END.
```

Ikusten denez `{$IFDEF herriaBAI}` eta `{$ENDIF herriaBAI}` direktibak jarri dira `KonpilazioBaldintzatua4` adibidean. Ondorioz programa exekutagarri bi sortuko ditugu, bat programaren hasieran `{$DEFINE herriaBAI}` definizioa eginez eta bestea programa hasieran `{$DEFINE herriaEZ}` eginez.

`KonpilazioBaldintzatua4` programan hasierako definizioa `{$DEFINE herriaBAI}` eta `{$DEFINE herriaEZ}` eginez hauek dira `KonpilazioBaldintzatua4` programaren irteerak elkar ondoan jarririk:

```
Izena eman: Julian
1. abizena eman: Retegi
2. abizena eman: Barberia
Herria eman: Eratsun

Izena---->Julian
Abizen1-->Retegi
Abizen2-->Barberia
Herria--->Eratsun
_
```

```
Izena eman: Julian
1. abizena eman: Retegi
2. abizena eman: Barberia

Izena---->Julian
Abizen1-->Retegi
Abizen2-->Barberia
_
```

Orain arte ikusitakoarekin irteerak asmatzea erraza izango litzateke, programaren bertsio batean `{$DEFINE herriaBAI}` definizioari esker herria eskatu eta pantailaratu egiten da, eta bigarren bertsioan `{$DEFINE herriaEZ}` direktiba definitzean ez da herriaren aipamenik programan egingo.

`{$DEFINE herriaBAI}` eta `{$DEFINE herriaEZ}` direktibek konpilazio ezberdinak eragiten dituzteela frogatzeko modu bat, `KonpilazioBaldintzatua4` programaren fitxategi exekutagarriak konparatzea litzateke. Hau da, `KonpilazioBaldintzatua4` adibide-programa `{$DEFINE herriaBAI}` direktibarekin konpilatzean sortzen den fitxategi exekutagarria `\TP70\08\DIREKT12.EXE` deitzen da eta duen tamaina 2832 byte da; bestalde programa bera `{$DEFINE herriaEZ}` direktibarekin konpilatzean sortzen den fitxategi exekutagarriari izen bera dagokio baina 2672 bytekoa izango da.

Konpilazio baldintzatuarekin amaitzeko `KonpilazioBaldintzatua5` adibide-programa aztertuko dugu, non konpilazioa bi konmutadoreen egoerak kontutan izanik burutzen den. Kasu honetan `{$IFOPT konmuta}` itxurako direktiba bakoitzak bere `{$ENDIF}` du eta kontutan izan `{$IFOPT V+}-{$ENDIF}` blokeak bestea barneratzen duela, hots, `{$IFOPT V+}-{$ENDIF}` blokea kabiaturik dagoela:

```
PROGRAM KonpilazioBaldintzatua5 ;             { \TP70\08\DIREKT13.PAS }
VAR
  Izena, Abizen1, Abizen2, Herria : String ;

{$V+,B-}

BEGIN
   Write('Izena eman: ', Izena) ;
   ReadLn (Izena) ;
{$IFOPT V+}
   Write ('1. abizena eman: ') ;
   ReadLn (Abizen1) ;
   {$IFOPT B+}
   Write ('2. abizena eman: ') ;
   ReadLn (Abizen2) ;
   {$ENDIF}
   Write ('Herria eman: ') ;
   ReadLn (Herria) ;
   WriteLn ;
   WriteLn ('Izena---->', Izena)  ;
   WriteLn ('Abizen1-->', Abizen1) ;
   Writeln ('Abizen2-->', Abizen2) ;
   Writeln ('Herria--->', Herria) ;
{$ENDIF}
END.
```

Ikusten den bezala `KonpilazioBaldintzatua5` programaren `{$V+,B-}` direktibari esker `V` eta `B` konmutadoreek `+` eta `-` balio dute hurrenez hurren, zein litzateke programaren irteera?. Eta direktiba honelaxe jarriz `{$V-,B+}` zein litzateke `KonpilazioBaldintzatua4` programaren irteera?

8.8 PROGRAMAK

Hona hemen 8. kapituluaren programak orrialdeen arabera sailkaturik:

Izena	*Programaren identifikadorea*	*ORRI.*	*Ikasgaia*
DATU_MO1.PAS	DatuMotarikGabe1	8-07	Norberaren datu-motak
DATU_MO2.PAS	DatuMotarikGabe2	8-08	Norberaren datu-motak
BIHURKET.PAS	DatuMotaBihurketak	8-08	Datu-moten bihurketak
DATU_MO3.PAS	DatuMotarikGabe3	8-10	Norberaren datu-motak
DATU_MO4.PAS	DatuMotaBerriarekin	8-10	Norberaren datu-motak
DATU_MO5.PAS	ZooDatuMota	8-12	Norberaren datu-motak
DATU_MO6.PAS	OrdutegiaDatuMotarikGabe	8-13	Norberaren datu-motak
DATU_MO7.PAS	OrdutegiaDatuMotarekin	8-13	Norberaren datu-motak
DATU_MO8.PAS	AzpieremuakZooan	8-15	Norberaren datu-motak
DATU_MO9.PAS	OrdutegiaDatuMotekin	8-16	Norberaren datu-motak
DATU_M10.PAS	KonpilazioDirektiba	8-17	Norberaren datu-motak
DIREKT1.PAS	R_KonpilazioDirektiba	8-22	Konpilazio direktibak
DIREKT2.PAS	I1_KonpilazioDirektiba	8-23	Konpilazio direktibak
DIREKT3.PAS	I2_KonpilazioDirektiba	8-24	Konpilazio direktibak
DIREKT4.PAS	V1_KonpilazioDirektiba	8-25	Konpilazio direktibak
DIREKT5.PAS	V2_KonpilazioDirektiba	8-25	Konpilazio direktibak
DIREKT6.PAS	P_KonpilazioDirektiba	8-27	Konpilazio direktibak
DIREKT7.PAS	A_KonpilazioDirektiba	8-28	Konpilazio direktibak
DIREK8_0.PAS	I_KonpilazioDirektiba	8-29	Konpilazio direktibak
DIREK8_1.PAS DIREK8_2.PAS	Batu Kendu	8-30	Konpilazio direktibak
DIREKT9.PAS	KonpilazioBaldintzatua1	8-31	Konpilazio direktibak
DIREKT10.PAS	KonpilazioBaldintzatua2	8-31	Konpilazio direktibak
DIREKT11.PAS	KonpilazioBaldintzatua3	8-32	Konpilazio direktibak
DIREKT12.PAS	KonpilazioBaldintzatua4	8-33	Konpilazio direktibak
DIREKT13.PAS	KonpilazioBaldintzatua5	8-34	Konpilazio direktibak

8.9 BIBLIOGRAFIA

- Salmon, W., *Introducción a la computación con Turbo Pascal. Estructuta y abstracciones*, Addison-Wesley Iberoamericana, Wilmington, 1993
- Brookshear, J.G., *Introducción a las Ciencias de la Computación*, Addison-Wesley Iberoamericana, Wilmington, 1995
- Wirth, N., *Algoritmos + Estructuras de Datos = Programas*, Ed. Castillo, 1986

9. ATALA: STRING DATU-MOTA

AURKIBIDEA

9.1 SARRERA

Laugarren kapituluan, **4.6.1 STRING datu-mota** izenburuko puntuan, karaktere-kateak gordetzeko aldagaien aurkezpena egin izan dugu, orain datu-mota horretaz sakonduko dugu.

9.1.1 Definizioa

Programa batean karaktere-kateak erabiltzeko aldagaiak `VAR` blokean erazagu daitezke zuzen-zuzenean, baina onena datu-mota berri bat sortzea litzateke. Esate baterako, jarraian ematen diren erazagupenetan `DM_KateLuze`, `DM_KateMotz` eta `DM_Kate25` datu-motak deklaratzen dira:

```
TYPE
   DM_KateLuze =  String[100] ;
   DM_KateMotz =  String[10] ;
   DM_Kate25 =    String[25] ;
```

Ondoren, datu-mota horiek kontutan izanik aldagaiak letozke:

```
VAR
   Helbidea : DM_KateLuze ;
   Izena : DM_KateMotz ;
   Herria : DM_Kate25 ;
```

Ondorioz hiru karaktere-kate izango genuke memorian, jokamolde honi definizio esplizitoa esango diogu datu-motak tartean zehatz eta esplizitoki agertzen direlako, baina badakigu memori erreserbak egiteko aldagaien deklarazio blokean egin daiteke hau idatziz:

```
VAR
   Helbidea1 : String[100] ;
   Izena1 : String[10] ;
   Herria1 : String[25] ;
```

Jokamolde inplizitoa den honek duen arazo nagusia erabiltzailearen azpiprogramen parametroena da. `Helbidea1`, `Izena1` eta `Herria1` aldagaiak ezin direla erabiltzailearen funtzio edo prozeduretara pasatu alegia.

9.1.2 Luzera fisiko vs luzera logiko

STRING datu-motako aldagaiak, hitzak edo esaldiak gordetzeko balio du, string bat zenbait karakterez osaturiko katea izango da. Horregatik, STRING aldagaia definitzean memorian zenbat karaktere biltegitu nahi den zehaztu beharra dago, hots, aldagaiak zenbat zortzikote hartuko duen memorian (karaktere bakoitzeko byte bat beharko baita). Ondoko programaren bitartez `KateNagusi`, `KateLabur`, `KateLuze` eta `Kate1` karaktere-kateen datu motak defininitzen dira eta horietan oinarritutako string aldagaiek zenbat byte hartzen duten memorian azter daiteke:

```
PROGRAM StringDatuMotenLuzera ;                    { \TP70\09\STRING1.PAS }
USES
    Crt ;
TYPE
   DM_KateNagusi = String[255] ;
   DM_KateLuze = String[199] ;
   DM_KateLabur = String[10] ;
   DM_Kate1 = String[1] ;
```

```
VAR
   Itxaron : Char ;
BEGIN                    { Programa Nagusia }
   ClrScr ;
   WriteLn ('STRING DATU-MOTEN MEMORI POSIZIOEN BEHARRAK') ;
   WriteLn ('=========================================') ;
   WriteLn ('DM_KateNagusi --------> ', SizeOf (DM_KateNagusi):3, ' byte') ;
   WriteLn ('DM_KateLuze ----------> ', SizeOf (DM_KateLuze):3, ' byte') ;
   WriteLn ('DM_KateLabur ---------> ', SizeOf (DM_KateLabur):3, ' byte') ;
   WriteLn ('DM_Kate1 -------------> ', SizeOf (DM_Kate1):3, ' byte') ;
   Itxaron := ReadKey ;
END .
```

`StringDatuMotenLuzera` programaren irteeran agertzen diren zenbakiak ezaguna zaigun `SizeOf()` funtzio estandarren emaitzak dira:

```
STRING DATU-MOTEN MEMORI POSIZIOEN BEHARRAK
=========================================
DM_KateNagusi --------> 256 byte
DM_KateLuze ----------> 200 byte
DM_KateLabur ---------->  11 byte
DM_Kate1 ------------->   2 byte
_
```

Ikus daitekeenez STRING datu-mota sor dadin `String` hitz erreserbatua erabiltzen da. Karaktere-kate baten luzera zehazteko `String` hitzarekin batera zenbaki oso bat kortxete artean eman behar da, geroago justifikatuko dugu baina orain karaktere-katerik luzeena 255-ekoa izan daitekeela konturatzen gara. Zer gertatzen da, adibidez, `KateLuze` aldagaia horrela deklaratzean?

```
DM_KateLuze = String[999] ;
```

Katearen tamaina 255 maximotik gora edo 1 minimotik behera jartzean konpilazio errorea azalduko da: `Error 25: Invalid string length.`

Eta zer gertatzen da, adibidez, `KateLuze` aldagaia horrela deklaratzean?

```
DM_KateLuze = String ;
```

Luzerarik finkatzen ez denean konpiladoreak maximoa ezartzen dio aldagaiari. Esan den bezala gehinez 255 karaktere dituen esaldi bat gorde daiteke string batean, baina esaldi luzeegi bat gordetzen saiatzean esaldia moztuko litzateke azkeneko karaktereak galduz. Karaktere-katea aldagaia erazagutzean zehazten den luzerari *luzera fisiko* esaten zaio, eta aldagai horretan kabituko diren karaktereen kopuru maximoa mugatzen du.

`StringDatuMotenLuzera` programaren irteeran azaltzen den gauzarik harrigarriena, eta aldiberean, karaktere-kateen ezaugarri funtsezkoena karaktere-kate bat `L` bytekoa definitzen bada, dagokion `SizeOf()` funtzioak `L+1` byte hartzen direla memorian salatzen du. Hau da karaktereak gordetzeko `L` posizioak gehi bat.

Ikus dezagun orain karaktere-kate batean balioak nola gordetzen diren. String datu-motako aldagai bati balioa emateko (gainerako datu-motatako aldagaiei bezalaxe) teklatuaren bitartez egindako sarreraz edo asignazioaz egin daiteke. `StringenEsleipena` izeneko programa honetan `Izena` string aldagaiari teklatuz ematen zaio balioa. Bestalde, `DEITURA1` konstantea string bat da ere eta eman nahi den `Salazar` balioa, `Char` datu-motaren moduan, komatxoen artean mugatzen da. Gauza bera esan daiteke `Deitura2` aldagaiari buruz, hots, bertan `Etxezarreta` gordetzeko honela idatziko da `'Etxezarreta'` bestela konpiladoreak aldagaitzat joko luke etiketa hori komatxoak ez daudelako.

Karaktere-kateetan balioak esleipen bitartez gordetzeko komatxoak nahitaezkoak dira, baina teklatuaren bitartez balioa ematean ez zaio komatxo parea jartzen.


```
PROGRAM StringenEsleipena ;                          { \TP70\09\STRING2.PAS }
CONST
   DEITURA1 = 'Salazar' ;
VAR
   Izena, Deitura2 : String[7] ;   { 7 karaktere gordetzeko aldagaiak }
BEGIN
   Write ('Zure izena eman: ') ;
   ReadLn (Izena) ;
   Deitura2 := 'Etxezarreta' ;

   WriteLn (Izena, ' ', Deitura1, ' ', DEITURA2) ;
END .
```

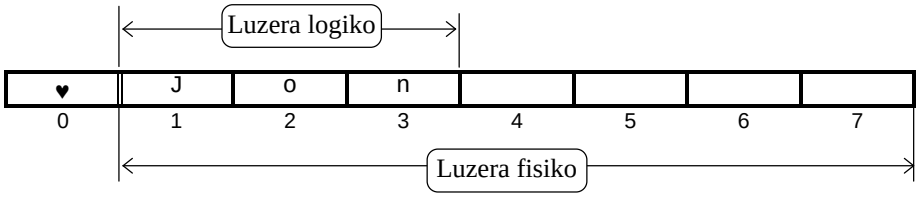
`StringenEsleipena` izeneko programa exekutatzean honelako zerbait ager daiteke monitorearen pantailan:

```
Zure izena eman: Jon
Jon Salazar Etxezar
_
```

Ikus daitekeenez `Izena` aldagaiak teklatuaren bitartez emandako `Jon` balioa (teklatutik komatxorik gabe) jasotzen du, eta `Deitura2` aldagaia saiatzen da `Etxezarreta` balioa gordetzen baina luzeegia delako lehenengo zazpiak baino ez ditu memorizatuko. Programaren azken agindua den `WriteLn()` prozeduraren emaitzak erakusten duenez `Izena` aldagaiak bete gabe dituen azken lau posizioak ez dira bistaratzen, beraz, karaktere-kate batean hainbat sinbolo gorde ahal izango da baina erabiliko gabeko azken posizioak ez dira aintzat hartzen. Nola kontrolatzen duen konpiladoreak zenbat karaktere gorden diren string batean?

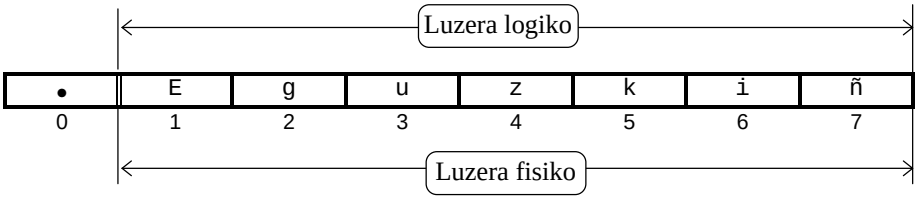
Arestian definitu luzera fisikoa karaktere-kate baten luzera maximoa baldin bada, gorde diren karaktereen kopurua luzera efektiboa edo *luzera logikoa* izango da. Luzera logikoa eta lehen aipatu dugun, eta `SizeOf()`-ek frogatu digun, soberazko karakterea erlazionaturik daude.

Aurreko adibidean `Izena` aldagaian `Jon` balioa gorde denean, luzera fisiko eta logikoaren arteko ezberdintasuna zenbakietan argi dago. Luzera fisikoa 7 da (nahiz eta ordenadorearen memorian 8 byte erreserbatu diren), `Izena` aldagaia deklaratzean aukeratu da eta programa bukatu arte ez da aldatuko. Luzera logikoa berriz, aldakorra da `Izena` deituriko katean `Jon` balioa pilatzean luzera logikoa 3 izango da. Ondoko eskeman adierazten da kontzeptu biren diferentzia, hau litzateke une horretan `Izena` aldagaiaren edukia:



Luzera fisikoa karaktere-katea definitzean zehaztu egiten da kortxete artean zenbaki oso bat emanik (`Izena` aldagaia deklaratzean 7 eman da), eta kateak zenbat karaktere gorde dezakeenez adierazten duenez programa amaitu arte luzera fisikoa ez da aldatuko. Luzera logikoaren kontzeptua berriz bestelakoa da, une jakin batean string batean gordetzen denaren tamaina adierazten du, hots, `Izena` aldagaian, gehienez, 7 karaktere gordetzeko ahalmena du baina adibidean 3 besterik ez ditu aprobetxatzen (lehenengo hirurak) beste laurak hutsik geratzen direlarik.

Baina `Izena` aldagaian `'Jon'` gorde beharrean `'Eguzkiñe'` biltegitu nahi bada, aurreko eskema honela agertuko litzateke:



Ikusten denez kasu honetan luzera fisikoa eta luzera efektiboa berdinak dira (kontutan izan `'Eguzkiñe'`-ren azken karakterea biltegiezina dela). Beste alde batetik, 7 karaktere gordetzeko ahalmena duen `Izena` aldagaiak 8 byte hartzen ditu memorian (0-tik 7-ra zenbatzen direnak), bi adibideetan luzera logikoa zero posizioaren bitartez adierazten dela ikus daiteke, horixe baita zero posizioak duen helburua.

Luzera logikoa, adibide bakoitzean, ♥ eta • sinboloz zehazturik geratzen da hurrengo puntuan ikur horien zergatia azalduko dugu.

9.1.2.1 String baten osagaiak

Errepikapena izan arren string baten osagaiak karaktereak dira, horretatik datorkio karaktere-kate izena. Bigarren kapituluan informazioaren adierazpidea aipatu izan zen eta bertan sinboloak ordenadorean gordetzeko, besteak beste, ASCII kodeaz egin daitekeela esan genuen.

Informazio testuala ordenadorean gordetzeko biderik zuzenena sinbolo bakoitzari kode bitarrean dagokion ordezkoa asmatzea litzateke, horrela konputagailu-kodeak deituriko zenbait kode agertu izan da, adibidez ASCII kodea (*American Standard Code for Information Interchange*, Informazioaren Trukaketarako Kode Estandar Amerikarra).

Zortzi biteko ASCII taularen hasiera eta amaiera honelakoa da:

ASCII-8				
Hamar.	**8tar**	**16tar**	**Bitar**	**Ikurra**
0	000	00	0000 0000	
1	001	01	0000 0001	☺
2	002	02	0000 0010	●
3	003	03	0000 0011	♥
4	004	04	0000 0100	♦
5	005	05	0000 0101	♣
6	006	06	0000 0110	♠
7	007	07	0000 0111	•
8	010	08	0000 1000	□
9	011	09	0000 1001	○
10	012	0A	0000 1010	■
⋮	⋮	⋮	⋮	⋮
251	373	FB	1111 1011	√
252	374	FC	1111 1100	^{n}
253	375	FD	1111 1101	^{2}
254	376	FE	1111 1110	■
255	377	FF	1111 1111	

Jakina denez, 8 bitekin berrehun eta berrogeitasei konbinazio bitar lor daitekezke ($8^2 = 256$), horregatik zerotik hastean azken balioa 255 izango da. String baten osagai indibidualak karaktereak, sinboloak, izango dira.

9.1.2.2 Zero posizioaren edukia

String baten osagai indibidualak karaktereak baldin badira, zero posizioaren edukia ere karaktere bat izango da (sistema hamartarrean `0`-tik `255` bitarteko karakterea hain zuzen). Beraz, kate baten zero posizioak katearen luzera efektiboa zehazteko karaktere batez adierazi beharrean dago, horregatik `'Jon'` kateari dagokion 3 luzera adierazteko hiruharrena ♥ den karakterea gordetzen da zero posizioan, `'Iker'` izan balitz ♦ eta `'Pedro'` izan balitz ♣.

Beraz, zero posizioaren edukiak karaktere baten bitartez kopuru oso bat aditzera ematen du.

Esandakoak karaktere-katerik luzeena 255-ekoa izan daitekeela justifikatzen du. Izan ere, string baten osagai indibidualak 8 bitez adierazitako karaktereak badira, katerik luzeena `1111 1111` konbinazioak emango du, hots, 255-eko luzera efektiboa duen katea.

Baina `Izena` aldagaian `'Jon'` edo `'Eguzkiñe'` balioei dagozkien ♥ eta • sinboloak nork eta noiz biltegitzen ditu?. Aurrerago zehaztuko dugu zerbait gehiago baina printzipioz luzera logikoa finkatzen duen zero posizioko karakterea *automatikoki* berritzen da `Izena` aldagaiarekin lan egitean aldaketak suertatzen direnean.

9.1.2.3 Karaktere-kateen eragiketak

Karaktere-kateek hiru eragiketa onartzen dituzte:

1. Kateen arteko esleipena
2. Kateen arteko konparaketak
3. Karaktere-kateen kateaketa

9.1.2.3.1 Kateen arteko esleipena

Eragiketarik oinarrizkoena da, lehenago ere aipatu dugu eta orain komatxoen beharra berretsiko dugu. Demagun string bat den `Deiturak` aldagaiari `Mendoza Esparza` balioa gorde nahi dela, aldagaiari teklatuaren bitartez balioa eman nahi denean komatxorik ez da jarriko:

```
Write (Deiturak eman: ) ;
ReadLn (Deiturak) ;
WriteLn (Deiturak) ;
```

```
Deiturak eman: Mendoza Esparza
Mendoza Esparza
-
```

Baina `Deiturak` string aldagaian balio berbera esleipenez emateko komatxoak erabat beharrezkoak dira:

```
Deiturak := 'Mendoza Esparza' ;
WriteLn ('Deiturak-> ', Deiturak) ;
```

```
Deiturak-> Mendoza Esparza
-
```

Ikusten denez, `Deiturak` aldagaiaren edukia pantailaratzean komatxoak ez dira agertuko. Baina esleipena nola egingo litzateke `O'Connor Uonegan` abizenak gordetzeko?.

Kasu honetan kateak berak komatxo bat dauka, teklatuaren bitarteko irakurketa egitean ez litzateke ezer berezirik idatzi beharko; baina `O'Connor Uonegan` katea esleipenez `Deiturak` katean biltegitu nahi bada aldagai bati asignatzen zaion konstantea kontsidera daiteke eta komatxoz mugatu beharra dago:

```
Deiturak := 'O'Connor Uonegan' ;
```

Argi dagoenez konpiladoreak ez luke aurreko esleipen hori onartuko komatxo bat falta delako. Ebazpidea komatxoa birritan idaztean dago, hots, esleipena egitean katearen erdian dagoen komatxoa bikoiztu beharra dago:

```
Deiturak := 'O''Connor Uonegan' ;
WriteLn ('Deiturak-> ', Deiturak) ;
```

```
Deiturak-> O'Connor Uonegan
-
```

9.1.2.3.2 Kateen arteko konparaketak

Kateek erlaziozko eragileak onartzen dituzte, jarraian ematen den taulan erlaziozko sei eragileak biltzen dira:

Eragilea	Eragiketa	Adibidea
>	Handiago baino	`Deiturak > 'Uonegan'`
<	Txikiago baino	`Deiturak < 'Jaio'`
>=	Handiago edo berdin	`Deiturak >= 'B'`
<=	Txikiago edo berdin	`Deiturak <= 'Uonegan'`
=	Berdin	`Deiturak = 'Beloki'`
<>	Desberdin	`Deiturak <> 'Beloki'`

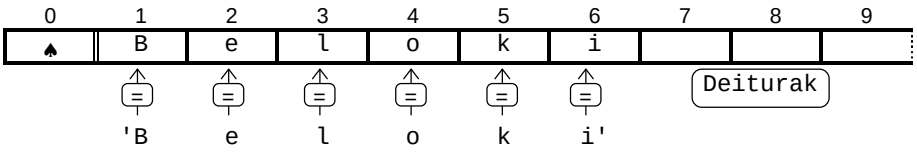
Azken eragile biak erraz ulertzen dira, esate baterako, ondoko `IF-THEN` egiturak `TRUE` balioko du baldin eta `Deiturak` aldagaian `Beloki` gorderik badago:

```
IF Deiturak = 'Beloki' THEN
```

Eta beste honetan, `IF-THEN` egiturak `TRUE` balioko du baldin eta `Deiturak` aldagaian `Beloki` ez den edozer gauza gorderik badago:

```
IF Deiturak <> 'Beloki' THEN
```

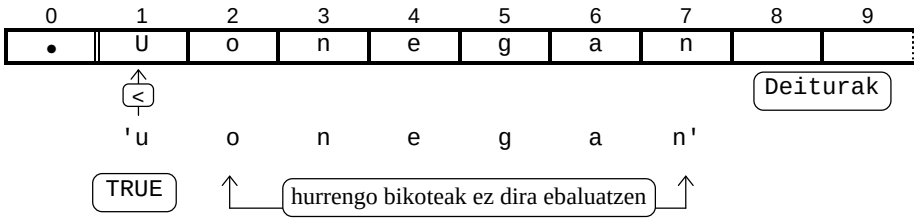
Hau da azken eragile bietan konparazioren emaitza lehen eta bigarren eragigaien (adibidean `Deiturak` aldagaiaren edukia `'Beloki'` eta konstantea) menpekoa da, eragigaiak zehatz-mehatz berdinak edo desberdinak izan beharko dute. Honezkero, arestian aipatu dugun `IF Deiturak = 'Beloki' THEN` sententziak `TRUE` balio dezan kate biren karaktere guztiek binaka kointziditu beharko lukete:



Lehenengo lau eragileak (> < >= <=) aplikatzean ere, konpiladoreak karaktereak binaka tratatzen ditu Adibidez, jarraian ematen diren bi aginduetatik `IF-THEN` egiturak `TRUE` balioko du:

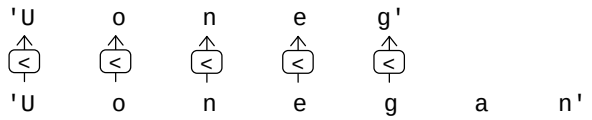
```
Deiturak := 'Uonegan' ;
IF Deiturak < 'uonegan' THEN
```

Izan ere aurreneko bikotearen (`U` eta `u` karaktereak) konparaketak `TRUE` ematen du[1], eta ondorioz `'Uonegan' < 'uonegan'` kateen arteko konparaketak ere `TRUE` balioko du, bigarren eta hurrengo bikoteen prozesaketari ez dio konpiladoreak heltzen:



Honekin jarraituta, gerta daiteke bi kateek hasierako karaktereak berberak edukitzea baina bata bestea baino luzeagoa izatea. Esate baterako `'Uoneg'` eta `'Uonegan'` kate biek ezberdinak direla zalantzarik ez dago baina < operadorez eginiko konparaketak zer emango luke?

```
IF 'Uoneg' < 'Uonegan' THEN
```



Horrela ikusita `IF-THEN` egitura horrek `FALSE` balioko duela ematen du, lehen bost karakteen bikoteak berdinak baitira. Baina ez, berdinak direnean kate laburrena luzeena baino txikiagoa kontsideratzen denez `IF-THEN` konparaketa horretatik `TRUE` ondoriotzen da.

Kateen konparaketak ASCII taulaz burutzen direla ikusi dugu, string batean digituak agertzen direnean konparaketak berdintsu egiten dira honako bi arauek erakusten diguten bezala:

1. Kateetan digituak baldin badaude ez dituzte kopuruak adierazten, letrak balira kontsideratzen dira. Digituei ASCII taularen 48tik 57 bitarteko karaktereak dagozkie:

   ```
   '31' < '1234'       { espresioak FALSE balio du }
   ```

2. Zuriunea karaktere bat gehiago da, ASCII taularen 32.a hain zuzen ere. Honezkero digituz beteriko kate bi luzera berekoak izan daitezen zuriuneak jartzen badira hasieran, konparaketa zenbakiak balira egin daiteke:

   ```
   '  31' < '1234'     { espresioak TRUE balio du }
   ```

9.1.2.3.3 Karaktere-kateen kateaketa

Operazio hau ez da operazio aritmetikoa, nahiz eta kateaketaren eragilea + izan. Kate bi kateatu ondoren emaitza beste karaktere-kate bat da, zeinek gordetzen duena eragigai biren edukiak den:

```
Deitura := 'Apraiz' ;
Atzizki := '-tar' ;
Gentilizio := Deitura + Atzizki ;
```

[1] Karaktere isolatuen arteko konparaketak ASCII taularen arabera burutzen denez `u` karakterea `U` karakterea baino handiagoa da (kontutan izan u karakterea `U` karakterea baino atzerago dagoela taulan, u-ren ordinala `117` da eta `U` karaktereari txikiagoa den `85` ordinala dagokio).

Demagun `Deitura`, `Atzizki` eta `Gentilizio` hiru aldagaiak karaktere-kateak direla ikusten denez `Deitura`-ren edukia `'Apraiz'` da eta `Atzizki`-rena `'-tar'` balio biak hirugarren aldagaian biltzeko kateaketa bat egiten da. Ondorioz, `Gentilizio` aldagaiak `'Apraiz-tar'` gordeko du. Hona hemen memoriaren egoera:

String bi edo gehiagoren kateaketa burutzean eragigaien ordena kontutan izaten da, nolabait esateko kateaketa bat elkarketa bat da baina ez batuketa bat (karaktereen kateaketa ez da konmuntatiboa[2] ez eta asoziatiboa[3]). Bestalde, kateaketaren emaitza jaso behar duen string aldagaiaren zero posizioa *automatikoki* berritzen da eta hau bai eragigaien zero posizioen batura aritmetikoa den (zehatzago[4] eragigaien zero posizioei dagozkien ordinalen batuketa eginez lortzen da `Gentilizio[0]` karakterearen balioa). Arestian aipatu bezala kateaketaren emaitza jaso behar duen `Gentilizio` aldagaiak duen luzera fisikoa laburregia bada ahal duen guztia biltegituko du gainerakoa galduz.

Jarraian ematen den `StringenKateaketa` programan zenbait string kateatuz emaitza pantailaratu egiten da:

```
PROGRAM StringenKateaketa ;                          { \TP70\09\STRING3.PAS }
CONST
   DEITURA1 = 'Salazar' ;
VAR
   Izena, Deitura2 : String[20] ;   { 20 karaktere gordetzeko aldagaiak }
   Emaitza : String[100] ;          { 100 karaktere gordetzeko aldagaia }
BEGIN
   Write ('Zure izena eman: ') ;
   ReadLn (Izena) ;
   Deitura2 := 'Etxezarreta' ;

   Emaitza := DEITURA1 + ' ' + Deitura2 + ', ' + Izena ;

   WriteLn (Emaitza) ;
END .
```

`StringenKateaketa` programan karaktere-kate bat den `Emaitza` delako aldagaian lau kateaketen emaitza gordetzen da. Eragigaiak bost kate dira; lehenengoa eta bigarrena `DEITURA1` eta `' '` konstanteak, hirugarrena `Deitura2` aldagaia, laugarrena `', '` konstantea eta bosgarrena `Izena` aldagaia.

Ondorioz `Izena`-ri `'Luis'` balioa emantean `StringenKateaketa` programaren irteera hauxe litzateke:

```
Zure izena eman: Luis
Salazar Etxezarreta, Luis
_
```

[2] Karaktere-kateen + eragiketak trukatze-legea betetzen ez duenez ezin daitezke `Deitura+Atzizki` kateaketa eta `Atzizki+Deitura` berdintzat jo.

[3] Karaktere-kateen + eragiketak ez du elkartze-legea betetzen.

[4] Zero posizioan, egitan, ezin da zenbaki bat jarri karaktere bat baizik.

9.1.3 Kateekin lan egiteko modua

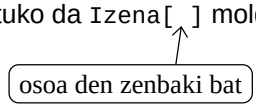
Kateak datu-mota egituratuak dira, string bat zenbait sinbolo biltzen duen aldagai edo konstantea izan daiteke. Horregatik kateekin bi eratara egin daiteke lan:

1. Katearen osagaiak banan-banan tratatuz. Lan egiteko modu hau ez da aproposena oraintxe bertan adibide batean ikusiko dugunez
2. Katea, bere osotasunean, kontzeptu bakar bat balitz bezala kontsideratuz. Irakurlea berehala konturatuko da hau dela kateekin lan egiteko modurik gomendagarriena, batez ere **9.1.4 Karaktere-kateen funtzio eta prozedura estandarrak** izenburua duen puntuan azaldutakoa ikusi ostean

Kateekin lan egiteko modu biak konparatzeko `Karaktereka1` programa prestatu dugu, non string bat teklatuz irakurri ondoren bere edukia era bietan agertzen den:

```
PROGRAM Karaktereka1 ;                              { \TP70\09\STRING4.PAS }
CONST
   MAX = 7 ;
VAR
   Izena : String[MAX] ;             { 7 karaktere gordetzeko aldagaia }
   Kont : Byte ;
BEGIN
   Write ('Zure izena eman: ') ;
   ReadLn (Izena) ;
   WriteLn (Izena) ;                 { Kate osoa }

   FOR Kont:=1 TO MAX DO
      Write (Izena[Kont], '-') ;     { Karaktereka }
   WriteLn ('/') ;
END .
```

`Izena` katea osoa kontsideratzen duen `WriteLn()` prozedura estandarrak katearen luzera efektiboa aintzat hartuko du pantailaraketa egitean. Bestalde, `Izena` kateren edukia karaktereka lortuz bistara daiteke, horretarako katearen etiketaz gain une bakoitzean landu behar den osagaia zenbaki batez zehaztuko da `Izena[ ]` moldea erabiliz.



`Izena` aldagaia teklatuz irakurtzean `'Ana'` emanez gero `Karaktereka1` programak irteera hau izango luke:

```
Zure izena eman: Ana
Ana
A-n-a- - - - -/
_
```

Karaktereka lan egiteak zailtasunak eragiten dituelako, adibidez `FOR-DO` egitura baten beharra, estrategia hau saihesteko joera izango dugu gehienetan kateak osorik erabiliz.

9.2 KATEEN FUNTZIO ETA PROZEDURA ESTANDARRAK

Puntu nagusi honen menpeko azpipuntuetan erakutsiko diren funtzio eta prozedura estandarrek karaktere-kateekin lan egiten dute, funtzio eta prozedua estandar hauek kateak egitura osatua bezala tratatzen dituzteenez zero posizioen balioa automatikoki berritzen da.

9.2.1 Funtzioak

Hau litzateke ikusiko ditugun karaktere-kateen funtzio estandarren zerrenda:

Azpirrutina	Mota	Deskribapen laburra
`Length()`	Funtz.	String baten luzera dinamikoa (luzera logikoa edo luzera efektiboa) itzultzen du
`Copy()`	Funtz.	Kate baten azpikatea itzultzen du
`Pos()`	Funtz.	Azpikate bat kate batean bilatzen du
`Concat()`	Funtz.	Zenbait kate kateatzeko balio du

Funtzio hauek banan-banan azter ditzagun adibide esanguratsuak erakutsiz.

9.2.1.1 `Length` funtzioa

`Length()` funtzioak sarrerako parametroaren luzera dimamikoa edo luzera logikoa itzultzen du, funtzioaren goiburukoak salatzen duenez emaitza zenbakizkoa da:

```
FUNCTION Length (KateBat : String) : Integer ;
```

Sarrerako parametroaren zero posizioak gordetzen duena aztertuz lan egiten du `Length()` funtzioak, itzultzen duen balioa zenbaki oso bat denez `Length()` funtzioak datu-mota bihurketa bat burutzera beharturik dago.

`Length()` funtzioa estandarra da eta bere barne inplementazioz ez gara zertan arduratu behar, baina, kasu honetan, erraza da asmatzea seguruenik nola dagoen garaturik. Jarraian ematen den adibide-programan `Length()` funtzioa erabiltzen da, eta, bide batez, bere barne funtzionamendua erakusten da ere:

```
PROGRAM KatearenLuzera ;                                { \TP70\09\STRING5.PAS }
CONST
   MAX = 20 ;
TYPE
   DM_Kate = String[MAX] ;
VAR
   Izena : DM_Kate ;             { 20 karaktere gordetzeko aldagaia }
   LuzeraLogiko1, LuzeraLogiko2 : Integer ;
BEGIN
   Write ('Zure izena eman: ') ;
   ReadLn (Izena) ;

   LuzeraLogiko1 := Length (Izena) ;
   WriteLn (' I:  ', Izena, '-(r)en luzera ', LuzeraLogiko1, ' da') ;


   LuzeraLogiko2 := Ord (Izena[0]) ;
   WriteLn ('II:  ', Izena, '-(r)en luzera ', LuzeraLogiko2, ' da') ;
   WriteLn ('     ', Izena, '-(r)en luzera ', Izena[0], ' da') ;
END .
```

`KatearenLuzera` programan `Izena` etiketa duen katea teklatuz irakurri ondoren, dagokion luzera dinamikoa bi modutan bilatzen da. Lehengoan `Length()` funtzioa aplikatzen da `LuzeraLogiko1` aldagaian emaitza gordez, bigarrenean `Izena[0]` kontzeptuan oinarrituz `LuzeraLogiko2` aldagaian karaktere horri dagokion ordinala biltegitzen da.

Hona hemen KatearenLuzera programan Izena-ren irakurketan Ana balioa ematean lortzen den pantailaraketa:

```
Zure izena eman: Ana
 I:  Ana-(r)en luzera 3 da
II:  Ana-(r)en luzera 3 da
     Ana-(r)en luzera ♥ da
_
```

Length() funtzioa ezagutzen dugula **9.1.3 Kateekin lan egiteko modua** izenburuko puntuan egindako Karaktereka1 programa hobetuko dugu FOR-DO bigiztaren goimugan luzera fisikoa jarri beharrean luzera efektiboa jarriz. Horrezkero, Karaktereka2 programan Izena katea birritan pantailaratzen da, lehenik WriteLn() baten bitartez kate osoa erakutsiz eta gero FOR-DO egiturak kontrolatzen duen hainbat Write() eginez katearen osagaiak diren karaktereak bistaratuz.

Karaktereka1 programaren aldaketa den Karaktereka2 programa:

```
PROGRAM Karaktereka2 ;                                  { \TP70\09\STRING6.PAS }
VAR
   Izena : String[7] ;
   Kont : Byte ;
BEGIN
   Write ('Zure izena eman: ') ;
   ReadLn (Izena) ;
   WriteLn ('Kate osoa------->', Izena) ;    { Kate osoa }

   Write ('Karaktereka----->') ;
   FOR Kont:=1 TO Length(Izena) DO
      Write (Izena[Kont]) ;                  { Karaktereka }

   WriteLn ;
END .
```

9.1.3 Kateekin lan egiteko modua puntuan ikusitako Karaktereka1 programan eman izan den sarrera berbera Karaktereka2-n errepikatuz irteera hauxe litzateke:

```
Zure izena eman: Ana
Kate osoa------->Ana
Karaktereka----->Ana
_
```

Pantailaraketa biak berdinak izan arren karaktereka lan egiteko modua saihesten ahaleginduko gara. Baina zenbaitetan beharturik aurkituko gara kateak elementuka tratatzen, esate baterako demagun teklatuz irakurritako Izena aldagaiaren edukia letra maiuskuletan bistaratu nahi izatea, kasu honetan justifikaturik legoke karaktereka lan egitea. Izan ere UpCase() funtzio estandarra karaktere bati aplika dakioke ez kate oso bati:

```
PROGRAM Karaktereka3 ;                                  { \TP70\09\STRING7.PAS }
VAR
   Izena : String[7] ;                 { 7 karaktere gordetzeko aldagaia }
   Kont : Byte ;
BEGIN
   Write ('Zure izena eman: ') ;
   ReadLn (Izena) ;

   FOR Kont:=1 TO Length(Izena) DO
      Write ( UpCase (Izena[Kont]) ) ;

   WriteLn ;
END .
```

`Karaktereka3` programa egikaritzean `Izena`-ri `Pedro` balioa teklatuz emanez gero hau pantailaratuko litzateke:

```
Zure izena eman: Pedro
PEDRO
_
```

9.2.1.2 Copy funtzioa

`Copy()` funtzioak behar dituen sarrerako parametroak hiru dira eta eskaintzen duen irteera string bat da, hartzen dituen sarrerak karaktere-kate bat eta bi zenbaki oso dira. Edozein azpiprogrametan parametroen ordena garrantzi handikoa denez `Copy()` funtzioaren goiburukoa arretaz aztertzea eta gogoan izatea komeniko da:

```
FUNCTION Copy (KateBat : String; Nondik, Zenbat : Integer) : String ;
```

`Copy()` funtzioak `KateBat` katearen azpikate bat itzultzen du, azpikate horren edukia `Nondik`-garren karakterean hasi eta hurrengo `Zenbat` karaktereaz osatzen da. Parametroen esanahia ikus dezagun:

`KateBat` parametro honek daukan edukia abiapuntutzat harturik lan egingo du `Copy()` funtzioak, `Copy()` funtzioak modulu deitzaileari itzuliko dion emaitza `KateBat` kateak duenaren zati bat izango da.

`Nondik` zenbaki honen bitartez `KateBat` katean oinarriturik lortuko den azpikatearen hasierako karakterea adierazten da. `Nondik` parametroaren balioa `KateBat` katearen luzera logikoa baino handiagoa baldin bada, orduan `Copy()` funtzioak itzultzen duena kate hutsa da (besterik ezin baitu).

`Zenbat` zenbaki honek `Copy()` funtzioaren emaitzak izango dituen karaktere kopurua adierazten du. `Zenbat` parametroaren balioa `Nondik` hasita `KateBat` katearen luzera logikoraino dagoen tartea baino handiagoa baldin bada, orduan `Copy()` funtzioak itzultzen duen azpikatea `Nondik` hasi eta `KateBat`-en azken karaktere bitartekoa izango da (ahal duena alegia).

Esate baterako, demagun string bat den `JatorrizkoKate` aldagaian `'ABCDEFGHIJ'` balioa biltegitu izan dela eta `EmaitzaKate` izeneko beste aldagai batean, `JatorrizkoKate`-ren bosgarren eta seigarren karaktereak gorde nahi direla:

```
JatorrizkoKate := 'ABCDEFGHIJ' ;
EmaitzaKate := Copy(JatorrizkoKate,5,2) ;
WriteLn ('EmaitzaKate->', EmaitzaKate) ;
```

```
EmaitzaKate->EF
-
```

9.2.1.3 Pos funtzioa

Funtzio honek dituen parametro biak kateak dira sarrerako kateak hain zuzen ere, eta modulu deitzaileari itzuliko dion emaitza positiboa eta osoa den zenbaki bat izango da. `Pos()` funtzioari honako goiburukoa dagokio:

```
FUNCTION Pos (AzpikateBat : String; KateBat : String) : Byte ;
```


Sarrekoak diren `AzpikateBat` eta `KateBat` karaktere-kateak balio ezagunak izango dituzte `Pos()` funtzioaren deia egiten denerako, eta emaitza `Byte` datu-mota duen zenbakiak `AzpikateBat` katea `KateBat` katearen zatia den adierazten du. `KateBat` katearen barnean `AzpikateBat` katea aurkitzen ez bada `Pos()` funtzioak `0` itzultzen du, eta barnean aurkitzen bada itzultzen duen zenbaki osoa `AzpikateBat` katearen lehen agerpenaren posizioa itzuliko du. Parametroen esanahia ikus dezagun:

`AzpikateBat` parametro honek daukan edukia aintzat harturik `KateBat` karaktere-katean bilatuko du `Pos()` funtzioak, behin baino gehiagotan balego lehendabiziko agerpena baino ez du kontsideratuko, `AzpikateBat` katearen edukia ez balego funtzioak `0` itzuliko du.

`KateBat` `Pos()` funtzioaren lehen parametroa (hots `AzpikateBat` katea) “zer bilatu” baldin bada, `KateBat` bigarren parametro hau “non bilatu” izango litzateke.

Esate baterako, demagun string bat den `NonBilatuKate` aldagaian `'HHHZZZIIIZZZ'` balioa biltegitu ondoren, `'ZZZ'` eta `'ZZZZ'` azpikateak bilatu nahi direla emaitzak `Non1` eta `Non2` aldagaietan gordetzen direlarik:

```
NonBilatuKate := 'HHHZZZIIIZZZ' ;
Non1 := Pos ('ZZZ', NonBilatuKate) ;
Non2 := Pos ('ZZZZ', NonBilatuKate) ;
WriteLn ('Non1->', Non1) ;
WriteLn ('Non2->', Non2) ;
```

```
Non1-> 4
Non2-> 0
-
```

Ondoko adibidean ikus daitekeen bezala bilatu behar den `AzpikateBat` parametroa kate bat izan beharrean karaktere bat izan daiteke, datu-mota biak konpatibleak direnez string bat jar daitekeen tokian `Char` bat ipintzea zilegia da (baina ez alderantziz). Labur-labur azalduta `ZeroakTartekatzen` izeneko programan kate batean dauden zuriune guztiak zeroz ordezkatzen dira:

```
PROGRAM ZeroakTartekatzen ;                          { \TP70\09\STRING8.PAS }
CONST
   ZURIUNEA = ' ' ;
   ZEROA = '0' ;
TYPE
   DM_Kate = String[20] ;
VAR
   KopuruKate : DM_Kate ;             { 20 karaktere gordetzeko aldagaia }
   Non : Byte ;
BEGIN
   KopuruKate := '   69.315';
   WriteLn ('Hasieran--->', KopuruKate) ;

   REPEAT
      Non := Pos(ZURIUNEA, KopuruKate) ;
      IF Non <> 0 THEN
      BEGIN
         KopuruKate [Non] := ZEROA ;
         WriteLn ('--->', KopuruKate) ;
      END
      ELSE
        WriteLn ('Amaieran--->', KopuruKate) ;
   UNTIL Non = 0 ;
END .
```

Zuriuneak dauden bitartean `REPEAT-UNTIL` kontrol-egituraren bidez testatzen da, eta baiezkoan zuriunean izan den tokian zero karakterea jartzen da. Prozesaketaren hasieran eta amaieran `KopuruKate` aldagaiak duen luzera logikoa berbera denez gero, berari dagokion `KopuruKate[0]` posizioa ez da aldatu behar horregatik bigiztaren barruan `IF-THEN-ELSE` kontrol-egitura idatzi izan da, bestela `Pos()` funtzioak zuriunerik ez aurkitzean `0` itzuliko

duenez `KopuruKate[Non]:=ZEROA;` aginduaren bitartez luzera logikoa aldaturik geratuko litzateke. Hona hemen `ZeroakTartekatzen` delako programaren irteera:

```
Hasieran--->   69.315
--->0  69.315
--->00 69.315
--->00069.315
Amaieran--->00069.315
_
```

9.2.1.4 `Concat` funtzioa

Funtzio honen bitartez **9.1.2.3.3 Karaktere-Kateen** puntuan ikusitako `+` eragiketa burutzerik dago. `Concat()` funtzioak kateak diren hainbat parametro onartzen du eta, euren ordenari jarraituz kateatzen dutu. Hauxe da bere goiburukoa:

```
FUNCTION Concat (Kate1:String; Kate2:String; ...) : String ;
```

Beraz, kateak diren `Emaitza1` eta `Emaitza2` aldagaietan balio berbera biltegituko da jarraian ematen diren sententzia biak exekutatzean:

```
Emaitza1 := 'Turbo' + ' ' + 'Pascal' ;
Emaitza2 := Concat ('Turbo', ' ', 'Pascal') ;
```

9.2.2 Prozedurak

Hau litzateke ikusiko ditugun karaktere-kateen prozeduren zerrenda:

Azpirrutina	Mota	Deskribapen laburra
`Delete()`	Proze.	Kate batetik azpikate bat ezabatzen du
`Insert()`	Proze.	Azpikate bat kate batean txertatzen du
`Str()`	Proze.	Zenbakizko balio bat string bihurtzen du
`Val()`	Proze.	String bat zenbaki bihurtzen du

Azpiprogramok banan-banan azter ditzagun adibide esanguratsuak erakutsiz.

9.2.2.1 `Delete` prozedura

`Delete()` prozedura estandarrak hiru parametro behar ditu eta kate bati zenbait karaktere kentzen dio. Hauxe da `Delete()` prozeduraren goiburukoa:

```
PROCEDURE Delete (VAR KateBat : String; Nondik, Zenbat : Integer) ;
```

Non,

`KateBat` parametro hau sarrera/irteerakoa da, hots, `Delete()` deitu aurretik `KateBat` kateak balio ezagunen bat izan beharko du, eta, prozedura exekutatu ondoren `KateBat` kateak jasotako aldaketak modulu deitzailearen dagokion uneko parametroan isladatzen da.

`Nondik` zenbaki honen bitartez `KateBat` katean ezabatuko den azpikatearen hasierako karakterea adierazten da. `Nondik` parametroaren balioa `KateBat` katearen luzera logikoa baino handiagoa baldin bada, orduan `Delete()` prozedurak ez du ezer ezabatzen.

`Zenbat` zenbaki honek `Delete()` prozedurak ezabatuko dituen karakteren kopurua adierazten du. `Zenbat` parametroaren balioa `Nondik` hasita `KateBat` katearen luzera logikoraino dagoen tartea baino handiagoa bada, orduan `Delete()` prozedureak `Nondik`-garren karakterean hasi eta `KateBat`-en gainerako karaktere guztiak ezabatuko ditu (ahal duena alegia).

Esate baterako, demagun string bat den `SarreraKate` aldagaian `'ABCDEFGHIJKLM'` balioa biltegitu izan dela eta `SarreraKate`-ren bosgarren eta seigarren karaktereak kendu nahi direla:

```
SarreraKate := 'ABCDEFGHIJKLM' ;
Delete (SarreraKate, 5, 2) ;
WriteLn ('SarreraKate->', SarreraKate) ;
```

```
SarreraKate->ABCDGHIJKLM
-
```

Kontutan izan adibidearen katearentzat `SarreraKate` izena aukeratu izan dela, horrek esan nahi du etiketa hori guztiz egokia ez dela `SarreraKate` katea sarrerakoa izateaz gain irteerakoa delako.

9.2.2.2 `Insert` prozedura

Prozedura hau `Delete()` prozeduraren aurkakoa dela esan dezakegu. Hiru parametro behar ditu (bi karaktere-kate eta zenbaki oso bat) eta honetan ere bi kateetatik bat sarrera/irteerako izango da. Hona hemen `Insert()` prozeduraren goiburukoa:

```
PROCEDURE Insert (AzpikateBat : String; VAR KateBat : String; Non : Integer) ;
```

`Insert()` prozeduraren bitartez `KateBat` katea aldatua suertatuko da, azkenean zuen edukiaz gain `AzpikateBat` kateak duena gordetzen duelarik. `Non` aldagaiaren balioaren bitartez `AzpikateBat`-ren edukia zer tokitan txertatuko den adierazten da.

Parametroen azalpena banaka:

`AzpikateBat` parametro hau sarrerakoa dela ikus daiteke, eta bigarren parametroa den `KateBat` kateari gehituko zaion karaktere multzoa da. Zer esanik ez `Insert()` prozedura deitu aurretik `AzpikateBat` kateak balio ezaguna izan beharko luke.

`KateBat` parametro hau sarrera/irteerakoa da, hots, `Insert()` deitu aurretik `KateBat` kateak balio ezagunen bat izan beharko du, eta, prozedura exekutatu ondoren `KateBat` kateak jasotako aldaketak modulu deitzailearen dagokion uneko parametroan isaladatzen da.

`Non` zenbaki honen bitartez azpikatearen karaktereak `KateBat` katean nondik hasita tartekatuko edo txertatuko diren adierazten da. `Non`, `AzpikateBat` eta `KateBat` parametroen balioak direla eta lortzen den emaitza luzeegia bada (255 karaktere baino gehiago baditu) irteerako string hori moztu egiten da.

Esate baterako, demagun string bat den `Karaktereak` aldagaian `'MMM'` balioa gorde izan den bitartean `EmaitzaKate`-ren edukia `'AAAZZZ'` dela, eta azkenean `EmaitzaKate` kateak `'AAAMMMZZZ'` izan dezala nahi dugula:

```
Karaktereak := 'MMM' ;
EmaitzaKate := 'AAAZZZ' ;
Insert (Karaktereak, EmaitzaKate, 4) ;
WriteLn ('EmaitzaKate->', EmaitzaKate) ;
```

```
EmaitzaKate->AAAMMMZZZ
-
```

9.2.2.3 `Str` prozedura

Prozedura hau eta hurrengo puntuan azaltzen den `Val()` prozedura informazioa bat datu-motaz aldatzeko erabiltzen dira. `Str()` prozedurak bi parametro behar ditu (lehen parametroa zenbaki bat izango da eta bigarrena prozeduraren emaitza izango den karaktere-katea). Hona hemen `Str()` prozeduraren goiburukoa:

```
PROCEDURE Str (ZenbakiBat : Integer[5]; VAR KateBat : String) ;
```

`Str()` prozeduraren bitartez `KateBat` katea aldatua suertatuko da, azkenean geratuko zaion edukia `ZenbakiBat` zenbakiak duena izango da.

Parametroen azalpena banaka:

`ZenbakiBat` parametro hau sarrerakoa dela ikus daiteke, eta nahi izanez gero formatua ere zehaz daiteke (ikus adibideak). Gogoratu `Str()` prozedura deitu baino lehen `ZenbakiBat` parametroak balio ezaguna izan beharko lukeela.

`KateBat` parametro hau irteerakoa da, hots, `Str()` deitu aurretik `KateBat` kateak balio ezagunen bat izango du eta prozedura exekutatu ondoren `KateBat` kateak jasotako balioa modulu deitzailearen dagokion uneko parametroari bidaliko diola.

Ondoko adibidean ikus daitekeen bezala bi zenbaki irakurtzen dira eta dagozkien kateak eskurstzen dira `Str()` prozedura estandarrari dei egokiak eginez (konturatu formatuen erabilpenaz):

```
PROGRAM DatuMotenTrukaketaStr ;                    { \TP70\09\STRING9.PAS }
TYPE
   DM_Kate = String[20] ;    { 20 karaktere gordetzeko aldagaien datu-mota }
VAR
   Kate1, Kate2, Kate3, Kate4 : DM_Kate ;
   DatuOsoa : Integer ;
   DatuErreala : Real ;
BEGIN
   Write ('Zenbaki oso bat eman======>') ;
   ReadLn (DatuOsoa) ;
   Write ('Zenbaki erreal bat eman===>') ;
   ReadLn (DatuErreala) ;

   Str (DatuOsoa, Kate1) ;
   Str (DatuErreala, Kate2) ;
   Str (DatuOsoa:8, Kate3) ;            { katea formatuaz }
   Str (DatuErreala:0:3, Kate4) ;       { katea formatuaz }

   WriteLn ('Kate1---->', Kate1) ;
   WriteLn ('Kate2---->', Kate2) ;
   WriteLn ('Kate3---->', Kate3) ;
   WriteLn ('Kate4---->', Kate4) ;
END .
```

[5] `Integer` datu-mota izan beharrean beste zenbakizko bat izan daiteke (`Real`, `Byte`, `LongInt` ...).

Hauxe da `DatuMotenTrukaketaStr` izeneko programari dagokion irteeraren bat:

```
Zenbaki oso bat eman======>12345
Zenbaki erreal bat eman===>123.45678
Kate1---->12345
Kate2----> 1.2345678000E+02
Kate3---->   12345
Kate4---->123.457
_
```

9.2.2.4 `Val` prozedura

Esan den bezala prozedura hau `Str()` errutinaren antzekoa da, baina kate batetik abiatuta dagokion zenbakia eskainiko du emaitzat. Kontutan izan dezagun `Str()` prozedurak ezin duela errorerik eman (zenbaki bat beti bihur daiteke kate); baina karaktere-kate batean daukagun informazioa zenbaki bezala landu nahi badugu gerta daiteke ezinezkoa izatea, esate baterako `'2.73M+01'` edukia gordetzen duen katea ezin da zenbaki bihurtu. Horregatik `Val()` prozedurak hiru parametro ditu (karaktere-katebat eta zenbaki bi). Hona hemen `Val()` prozeduraren goiburukoa:

```
PROCEDURE Val (KateBat : String; VAR ZenbakiBat : [6]; VAR ErroreKode : Integer) ;
```

`Val()` prozeduraren bitartez, `KateBat` katea jaso ondoren, zenbaki bi lortuko dira kateari dagokion zenbakia eta arazoak egonez gero `ErroreKode` parametroan errorea suertatu den posizioa.

Parametroen azalpena banaka:

`KateBat` parametro hau sarrerakoa da eta zenbaki baten itxura izan beharko luke.

`ZenbakiBat` irteerako parametro honek aurrekoaren informazioa zenbaki bezala gordeko du, honi dagokion datu-mota zenbakizkoa izango da.

`ErroreKode` parametro hau irteerakoa da, eta bihurketaren ezintasuna adieraziko du baldin eta `ErroreKode`-k zero ez den zenbaki oso bat balio badu. `KateBat` katetik zenbaki bat lor badaiteke `ErroreKode`-k zero balioko du, eta ezinezkoa bada errorea eragin duen karakterearen posizioa gordetzen da `ErroreKode`-an. Aurreko adibidean `ErroreKode`-k bost gordeko luke **`M`** karakterearen posizioa `'2.73M+01'` katean bosgarrena delako.

9.2.3 Kateen funtzio eta prozedura estandarren adibideak

Hona hemen karaktere-kateen bi adibide.

9.2.3.1 Adibidea

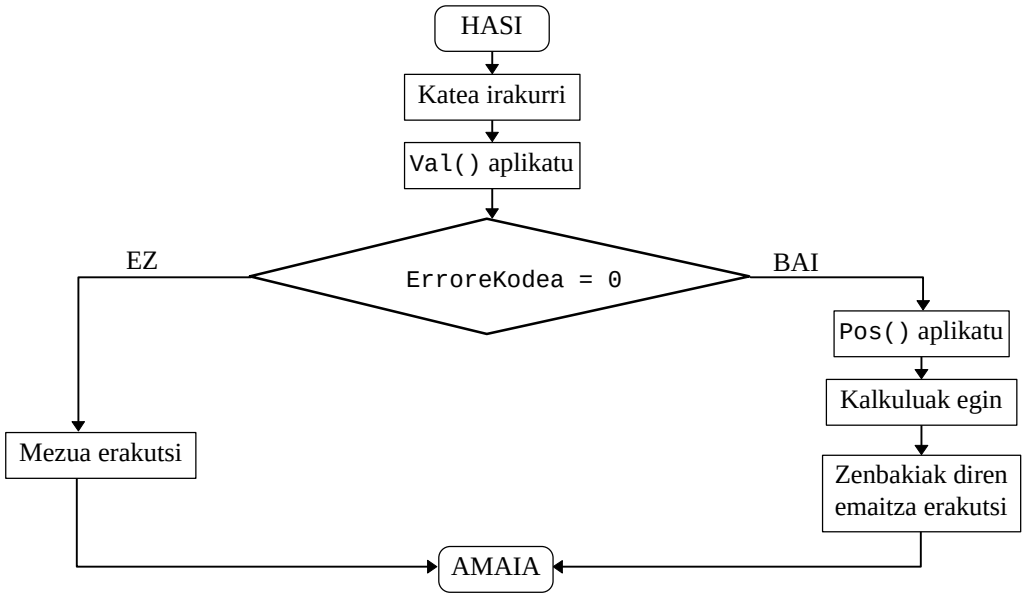
Zenbaki erreal bat errepresentatzen duen karaktere-kate bat teklatuz irakurri, eta bi zenbaki errealen zatiketa bezala pantailan agertu.

[6] Zenbakizko bat izan behar da (`Real`, `Integer`, `Byte`, `LongInt` ...).

Esate baterako `'12.3456'` balioa string datu-motako aldagai batean gorde ondoren honela bistaratu: `123456 / 10000`

Non `123456` eta `10000` zenbaki errealak diren

Adibide-programa honen gakoa dezimalen kopurua kalkulatzean datza, horretarako karaktere-katean punturik dagoen aztertuko da eta egotekotan bere posizioa lortu beharra dago. Hona hemen organigrama eta kodifikazioa:



Programari `DatuMotenTrukaketa` izena jarri diogu:

```
PROGRAM DatuMotenTrukaketa ;                          { \TP70\09\STRING10.PAS }
TYPE
   DM_Kate = String[10] ;       { 10 karaktere gordetzeko aldagaien datu-mota }
VAR
   SarreraDatua : DM_Kate ;
   EmaitzaErreala, Zatitzailea : Real ;
   ErroreKodea, PuntuaNon, DezimalKop, Kont : Integer ;
BEGIN
   Write ('Zenbaki erreal bat karaktere-kate bezala eman: ') ;
   ReadLn (SarreraDatua) ;

   Val (SarreraDatua, EmaitzaErreala, ErroreKodea) ;
   IF ErroreKodea = 0 THEN
     BEGIN
        PuntuaNon := Pos ('.', SarreraDatua) ;
        IF PuntuaNon > 0 THEN
          DezimalKop := Length(SarreraDatua) - PuntuaNon
        ELSE
          DezimalKop := 0 ;

        Zatitzailea := 1 ;
        FOR Kont:=1 TO DezimalKop DO
           Zatitzailea := Zatitzailea * 10 ;

        EmaitzaErreala := EmaitzaErreala * Zatitzailea ;

        WriteLn (EmaitzaErreala:0:0, '/', Zatitzailea:0:0) ;
     END
   ELSE
     BEGIN
        WriteLn ('Datuak sartzean ', ErroreKodea, '. posizioan errorea dago') ;
     END ;
END .
```


`DatuMotenTrukaketa` programa aztertzean `EmaitzaErreala` eta `Zatitzailea` zenbakiak lortu behar direla konturatzen gara, horretarako, `Val()` prozeduraren bitartez katea zenbaki erreala bihurtzen da. Ondoren, esan den bezala, sarrerako karaktere-katean puntua non dagoen bilatzen da eta datu horrekin dezimal kopurua lortzen da.

Hauxe dira `DatuMotenTrukaketa` programaren irteerak `SarreraDatua` karaktere-katearen aldagaian `'123.4567'` eta `'123,4567'` balioak irakurtzen direnean:

```
Zenbaki erreal bat karaktere-kate bezala eman: 123.4567
1234567/10000
Zenbaki erreal bat karaktere-kate bezala eman: 123,4567
Datuak sartzean 4. posizioan errorea dago
_
```

Ikus daitekeenez `'123.4567'` karaktere-katea ematean `Val()` prozedura estandarrak dagokion zenbakia (zenbaki erreala kasu honetan) lor dezake, horregatik `ErroreKodeak` `0` balioko du eta puntua non dagoen zehaztuz `EmaitzaErreala` eta `Zatitzailea` zenbakiak arazorik gabe kalkulatzen dira.

Baina `'123,4567'` karaktere-katea ematean `Val()` prozedura estandarrari ezinezkoa zaio zenbakirik lortzea, izan ere laugarren karakterea den koma hori onartezina da Pascal lengoaiaren zenbakietan. Beraz, `ErroreKodea` aldagaian komaren posizioa gordeko da eta horretan oinarriturik `DatuMotenTrukaketa` programari dagokion organigramaren ezkerreko abarra, errore-mezua erakusten duena, exekutatuko da.

Eta, zer gertatzen da `SarreraDatua` karaktere-katean punturik ez duen zenbaki bat ematean?. Hona hemen irteera, irakurlearen lana litzateke zergatia asmatzea:

```
Zenbaki erreal bat karaktere-kate bezala eman: 7654321
7654321/1
_
```

9.2.3.2 Adibidea

Bigarren adibidean ere karaktere-kate bat teklatuz irakurriko dugu, ondoren bokalak eta kontsonanteak bereiziz beste bi karaktere-kate lortuko dira, batean sarrerako katean emandako bokalak eta bigarrenean kontsonanteak baina alderantzizko ordenean (sarrerako katean gainerako karaktererik egonez gero ez dira aintzat hartuko eta galdu egingo dira).

Esate baterako `'Kaixo X42.AuM'` balioa sarrerako string aldagaian gorde ondoren `'uAoia'` eta `'MXxK'` bi kateak bistaratu. Hau da, sarreratik abiatuta ondorioztatzen diren bokalak `'aioAu'` dira eta kontsonanteak berriz hauek `'KxXM'`, baina kate horiek atzekoz aurrera jarriz `'uAoia'` eta `'MXxK'` aterako dira.

Jarraian `KaraktereBanaketa1` programaren sententziak erakusten dira, non bi azpiprograma agertzen diren. Lehenengoa `Banaketa` izeneko prozedura da zeinek sarrera aztertuz bokalen eta kontsonanteen kateak eskaintzen dituen, eta bigarrena `AtzekozAurrera` funtzioa zeini esker kate baten karaktereen ordena aldatzen den.

`Banaketa` eta `AtzekozAurrera` errutinak bi kontzeptutan oinarritzen dira eta horiek azpimarratuko genituzke. Batetik edozein kateren hasieraketa lortzeko modua komatxoak jartzeaz egiten dela, eta bestetik kate bati + operadorearen bitartez karaktere bat edo beste kate bat *gehitu*[7] ahal zaiola.

[7] Operazio horri kateaketa esaten zaio.

```
PROGRAM KaraktereBanaketa1 ;                        { \TP70\09\STRING11.PAS }
TYPE
   DM_Kate = String[80] ;    { 80 karaktere gordetzeko aldagaien datu-mota }

PROCEDURE Banaketa (Datua:DM_Kate; VAR Bokal, Konts:DM_Kate) ;
VAR
   i : Integer ;
BEGIN
   Bokal := '' ;     (* Hasieran kate horiek hutsik egongo dira *)
   Konts := '' ;
   FOR i:=1 TO Length(Datua) DO
   BEGIN
      CASE UpCase (Datua[i]) OF
         'A', 'E', 'I', 'O', 'U' : Bokal := Bokal + Datua[i] ;
         'B'..'D',
         'F'..'H',
         'J'..'N',
         'P'..'T',
         'V'..'Z'  : Konts := Konts + Datua[i] ;
      END ;
   END ;
END ;


FUNCTION AtzekozAurrera (Katea:DM_Kate) : DM_Kate ;
VAR
   i : Integer ;
   Emaitza : DM_Kate ;
BEGIN
   Emaitza := '' ;     (* Hasieran katea hutsik egongo da *)
   FOR i:=Length(Katea) DOWNTO 1 DO
   BEGIN
      Emaitza := Emaitza + Katea[i] ;
   END ;
   AtzekozAurrera := Emaitza ;
END ;


VAR
   Sarrera, Bokalak, Kontsonanteak : DM_Kate ;
BEGIN
   Write ('Karaktere segida bat eman: ') ;
   ReadLn (Sarrera) ;
   WriteLn ;
   Banaketa (Sarrera, Bokalak, Kontsonanteak) ;
   WriteLn ('Bokal(ordenez)--->', Bokalak) ;
   WriteLn ('Konts(ordenez)--->', Kontsonanteak) ;
   WriteLn ;
   Bokalak := AtzekozAurrera (Bokalak) ;
   Kontsonanteak := AtzekozAurrera (Kontsonanteak) ;

   WriteLn ('Sarrera---------->', Sarrera) ;
   WriteLn ('Bokalak---------->', Bokalak) ;
   WriteLn ('Kontsonanteak---->', Kontsonanteak) ;
END .
```

Hau izan daiteke `KaraktereBanaketa1` programa horren irteera bat:

```
Karaktere segida bat eman: Bokalak eta Kontsonanteak.

Bokal(ordenez)--->oaaeaooaea
Konts(ordenez)--->BklktKntsnntk

Sarrera---------->Bokalak eta Kontsonanteak.
Bokalak---------->aeaooaeaao
Kontsonanteak---->ktnnstnKtklkB
_
```


Bigarren adibide-programa hau beste modu batean plantea daiteke. Hots, emaitzak izango diren `Bokalak` eta `Kontsonanteak` kateak osorik erabili beharrean karaktereka erabiliz, horretarako zenbaki osoak izango diren `BokKont` eta `KonKont` kontagailuak behar izango dira.

`KaraktereBanaketa2` programaren berezitasun nagusiena `Bokal` eta `Konts` kateen zero posizioak aldatu beharrean datza. Izan ere, aurreko `KaraktereBanaketa1` programan karaktere-kateak osoan erabiltzean zero posizioa automatikoki aldatzen zaien bitartean, `KaraktereBanaketa2` bigarren bertsioan sententzia espezifikoa jarri behar da.

`KaraktereBanaketa2` programa ikus dezagun:

```
PROGRAM KaraktereBanaketa2 ;                    { \TP70\09\STRING12.PAS }
TYPE
   DM_Kate = String[80] ;   { 80 karaktere gordetzeko aldagaien datu-mota }

PROCEDURE BanatuAldatu (Datua:DM_Kate; VAR Bokal, Konts:DM_Kate) ;
VAR
   i, BokKont, KonKont : Integer ;
BEGIN
   BokKont := 0 ;     (* Hasieran kate horiek hutsik egongo direlako *)
   KonKont := 0 ;
   FOR i:=Length(Datua) DOWNTO 1 DO
   BEGIN
      CASE UpCase (Datua[i]) OF
         'A', 'E', 'I', 'O', 'U' : BEGIN
                                      BokKont := BokKont + 1 ;
                                      Bokal[BokKont] := Datua[i] ;
                                   END ;
         'B'..'D',
         'F'..'H',
         'J'..'N',
         'P'..'T',
         'V'..'Z'  : BEGIN
                        KonKont := KonKont + 1 ;
                        Konts[KonKont] := Datua[i] ;
                     END ;
      END ;
   END ;
   Bokal[0] := Chr (BokKont) ;     (* luzera logikoak finkatu *)
   Konts[0] := Chr (KonKont) ;
END ;


VAR
   Sarrera, Bokalak, Kontsonanteak : DM_Kate ;
BEGIN
   Write ('Karaktere segida bat eman: ') ;
   ReadLn (Sarrera) ;
   WriteLn ;
   BanatuAldatu (Sarrera, Bokalak, Kontsonanteak) ;

   WriteLn ('Sarrera---------->', Sarrera) ;
   WriteLn ('Bokalak---------->', Bokalak) ;
   WriteLn ('Kontsonanteak---->', Kontsonanteak) ;
END .
```

`KaraktereBanaketa2` programa exekutatzean:

```
Karaktere segida bat eman: KARAKTEREKA lan eginez.

Sarrera---------->KARAKTEREKA lan eginez.
Bokalak---------->eieaAEEAA
Kontsonanteak---->zngnlKRTKRK
_
```

9.3 NULL KARAKTEREZ BUKATURIKO KATEAK

NULL karakterea ASCII taulako lehen karakterea da eta dagokion ordinala `0` da. Dakigunez ASCII taulako lehendabiziko karaktereak kontrol-karaktereak dira, hots, idatziak eta irakurriak izan daitezen karaktereak izan ordez beste esanahi bereziak dituzte. Esate baterako, karaktere segida bat eman ahal izateko bere bukaera adierazteko NULL karakterea erabiltzen delarik[8].

`String` datu-motak duen muga bat katerik luzeena 255-koa izan daitekeela azaldu dugu kapitulu honen aurreneko puntuetan, baina NULL karakterez amaituriko kate batek izan ditzakeen karaktereak 65534 dira.

NULL karakterez bukatzen diren kateak honelako egituran biltegitzen dira memorian. Ikusten denez, oinarrian `Char` datu-mota duen array bat da. Azpimarratzekoa da array horrek dituen indizeak zenbaki osoak izanik `0`-tik hasi behar direla:

```
CONST
   LUZERAFISIKOA : 12345 ;     { zenbaki osoa eta positiboa }
VAR
   NullKateBat =ARRAY [0..LUZERAFISIKOA] OF Char ;
```

NULL karakterez bukatzen diren kateekin lan ahal izateko `Strings` deituriko unitatea erazagutu behar da `USES` klausula bitartez, unitate honek hainbat funtzio eskaintzen ditu NULL kateak kudeatzeko. `Strings` unitatea erabiltzeaz gain derrigorrezkoa da ere `{$X±}` direktiba indarrean egotea (ikus **8.6.1.7 `{$X±}` direktiba** izenburuko puntua zortzigarreneko kapituluan).

`Strings` unitatearen funtzio garrantzitsuenak[9] hauek lirateke:

Funtzioa	Deskribapen laburra
`StrLen()`	Kate baten luzera itzultzen du
`StrEnd()`	Kate baten amaieran kokatutako erakuslea itzultzen du
`StrCopy()`	Kate bat beste kate batean osorik kopiatzeko
`StrLCopy()`	Kate batetik hainbat karaktere beste kate batean kopiatzeko
`StrCat()`	Kate baten kopia beste kate baten atzean jarri kateaketa osoa itzultzeko
`StrLCat()`	Kate baten hainbat karaktereak beste kate baten atzean jarri kateaketa osoa itzultzeko
`StrComp()`	Kate bi konparatzen ditu zenbaki osoa itzuliz
`StrIComp()`	Kate bi konparatzen ditu maiuskula eta minuskulak aintzat hartu gabe
`StrLComp()`	Luzera baten muga barruan kate bi konparatzen ditu zenbaki osoa itzuliz
`StrLIComp()`	Luzera baten mugan kate bi konparatzen ditu maiuskula eta minuskulak aintzat hartu gabe
`StrLower()`	Kate bat minuskuletan jartzeko
`StrUpper()`	Kate bat maiuskuletara bihurtzeko
`StrPas()`	NULL karakterez amaituriko katetik abiatuta Pascal estiloko `String` datu-motako katea lortzeko
`StrPCopy()`	Pascal estiloko `String` datu-motako kate batetik abiatuta NULL karakterez amaituriko katea lortzeko
`StrPos()`	Azpikate bat kate batean bilatzen du, azpikatearen lehen agerpenean kokaturiko erakuslea itzultzen du
`StrECopy()`	Kate bat beste kate baten amaieran kopiatu eta erakuslea itzultzen du

[8] Horixe da hain zuzen ere Windows sistema eragilearen API-k (Application Programming Interface) karaktere-kateentzat darabilen formatua.

[9] Bost hauek 13. kapitulua ikasi ondoren hobeto ulertuko dira: `StrNew()`, `StrDispose()`, `StrMove()`, `StrScan()` eta `StrRScan()`.

Kapitulu honetako **9.1.3 Kateekin lan egiteko modua** puntuan `Karaktereka1` programa aztertu izan da, eta ondorio bezala `String` datu-motako aldagaiak karaktereka landu ordez osotasunean landuko ditugula aipatu zen. NULL karakterez bukaturiko kateak prozesatzeko ere bi bide ditugu, bat karaktere isolatuak landuz eta bestea kate osoaren prozesaketa eginez.

Esate baterako, jarraian ematen den `Str_Karaktereka` programan, NULL kate baten balioa teklatutik hartu ondoren bere edukia pantailaratzen da (kate osoa lehenik eta karaktereka ondoren). Bide batez NULL karaktereari dagokion ikurra ezagutu ahalko dugu.

```
PROGRAM Str_Karaktereka ;                    { \TP70\09\NULL_00.PAS }
USES
   Strings, Crt ;
CONST
   MAX = 7 ;
VAR
   NullKatea : ARRAY [0..MAX] OF Char ;
   Kont : Byte ;
BEGIN
   ClrScr ;
   Write ('Zure izena eman------>') ;
   ReadLn (NullKatea) ;

   WriteLn ('Emaniko kate osoa---->', NullKatea) ;

   FOR Kont:=0 TO MAX DO
      WriteLn (Kont, '. indizeari dagokion karakterea----->', NullKatea[Kont]) ;
END.
```

`Str_Karaktereka` programa egikaritzean izen luze bat tekleatzen badugu, honelako irteera izango genuke:

```
Zure izena eman------>Eguzkiñe
Emaniko kate osoa---->Eguzkiñ
0. indizeari dagokion karakterea----->E
1. indizeari dagokion karakterea----->g
2. indizeari dagokion karakterea----->u
3. indizeari dagokion karakterea----->z
4. indizeari dagokion karakterea----->k
5. indizeari dagokion karakterea----->i
6. indizeari dagokion karakterea----->ñ
7. indizeari dagokion karakterea----->
_
```

karaktere huts edo karaktere nulu

Ikusten denez `'Eguzkiñe'` datua katean osorik sartzen ez denez moztu egiten da. Eta bigarren ondorio bat NULL karakterearekin loturik dago, dirudienez NULL karaktereari dagokion ikurra zuriunea da `Str_Karaktereka` programaren irteerari erreparatzen badiogu, egia esan NULL karakterearen ikurra ez da inprimatzen eta ASCII taula aztertzen badugu 0 ordinala duen karakterea (NULL karakterea) ikurra hutsa da. Horregatik 0 ordinala duen NULL karaktereari *karaktere nulu* edo *karaktere huts* esaten zaio.

Baina, `Str_Karaktereka` programa egikaritzean izen laburra tekleatuz:

```
Zure izena eman------>Jon
Emaniko kate osoa---->Jon
0. indizeari dagokion karakterea----->J
1. indizeari dagokion karakterea----->o
2. indizeari dagokion karakterea----->n
3. indizeari dagokion karakterea----->
4. indizeari dagokion karakterea----->
5. indizeari dagokion karakterea----->
6. indizeari dagokion karakterea----->
7. indizeari dagokion karakterea----->
_
```

karaktere huts edo karaktere nulu

9.3.1 `StrLen` eta `StrEnd` funtzioak

`String` datu-motako karaktere-kate baten luzera logikoa kalkulatzeko `Length()` funtzio ezaguna aplikatzen ikasi dugu, era beretsuan karaktere nuluko formatua duen kate baten luzera logikoa `StrLen()` funtzioz lor daiteke (helburu horretarako `StrEnd()` funtzioa erabil daiteke ere).

Hona hemen `StrLen()` funtzioaren goiburukoa;

```
FUNCTION StrLen (Katea : PChar) : Word ;
```

`StrLen()` funtzioaren goiburukoan ikus daitekeenez `Word` datu-motako zenbakia itzultzen da modulu deitzaileari, zenbaki horrek `Katea` sarrerako parametroaren karaktere kopurua adierazten du (funtzioaren emaitzan ez da karaktere nulua kontatzen).

Karaktere nuluko formatua duen kate baten luzera logikoa kalkulatzen duen adibide-programa bat hau izan daiteke:

```
PROGRAM StrLen_Adibidea ;                    { \TP70\09\NULL_01.PAS }
USES
   Strings, Crt ;
VAR
   NullKatea : ARRAY [0..25] OF Char ;
BEGIN
   ClrScr ;
   Write ('Kate bat eman--->') ;
   ReadLn (NullKatea) ;
   WriteLn ('"', NullKatea, '" katearen luzera: ', StrLen (NullKatea) ) ;
END.
```

`StrLen_Adibidea` programaren balizko exekuzio bat:

```
Kate bat eman--->KAIXO LAGUNOK!
"KAIXO LAGUNOK!" katearen luzera: 14
_
```

Lehenago esan dugunez, NULL kate baten luzera logikoa zehazteko helburuarekin erabil daiteke `StrEnd()` funtzioa; honek katea amaiarazten duen karaktere nuluari erakusten dion erakuslea itzultzen baitu. Beste era batera esanda `StrEnd()` funtzioak katearen zati baliagarriaren amaieran kokatzen du erakusle bat eta erakusleen arteko eragiketa bat eginez (honetaz 13. kapituluan sakonduko dugu) luzera logikoa atera daiteke.

`StrEnd_Adibidea` programa hau funtzionalki aurreko `StrLen_Adibidea` programaren baliokidea da, sarrera bat emanda programa biek ateratzen baitute irteera bera:

```
PROGRAM StrEnd_Adibidea ;                    { \TP70\09\NULL_02.PAS }
USES
   Strings, Crt ;
VAR
   NullKatea : ARRAY [0..25] OF Char ;
BEGIN
   ClrScr ;
   Write ('Kate bat eman--->') ;
   ReadLn (NullKatea) ;
   Write ('"', NullKatea, '" katearen luzera: ') ;
   WriteLn (StrEnd (NullKatea) - NullKatea) ;
END.
```

9.3.2 `StrCopy` eta `StrLCopy` funtzioak

Jarraian agertzen den `StrCopy_Adibidea` programan ikusten denez 25 karaktere gorde ahal izateko `NullKatea` karaktere-katea dugu (0 eta 24 bitarteko posizioetan datuak, azkena den 25. posizioan karaktere nulua). `StrCopy()` funtzioa ezaguna izan dadin `Strings` unitatea erazagutu da.

```
PROGRAM StrCopy_Adibidea ;                  { \TP70\09\NULL_03.PAS }
USES
   Strings, Crt ;
VAR
   NullKatea : ARRAY [0..25] OF Char ;
BEGIN
   ClrScr ;
   StrCopy (NullKatea, 'Kaixo') ;
   WriteLn ('1=', NullKatea, '=1') ;
END.
```

`StrCopy()` funtzioaren goiburukoa hauxe da:

```
FUNCTION StrCopy (Helburua, Datua : PChar[10]) : PChar ;
```

`StrCopy()` funtzioaren bitartez `Helburua` kateak `Datua` katearen edukia gordeko du. Funtzioak ez du kateen luzerak konprobatzen, horregatik `Helburua` kateak izan behar duen luzera minimoa formula honek adierazten digu: `StrLen(Datua)+1`.

Parametroen azalpena banaka:

`Helburua` emaitza da, funtzioa deitu ondoren hartuko du balioa kate honek.

`Datua` parametro hau sarrerakoa da, hots, `StrCopy()` deitu aurretik `Datua` kateak balio ezagunen bat izango du.

`StrCopy_Adibidea` programan karaktere-kate konstanteekin lan egiten denez, hura egikaritzean gertatzen den irteera beti berdina izango da. Zehazkiago, `Kaixo` konstantea `NullKatea` karaktere-katean gorde ondoren bere pantailaraketa[11] eskatzean, hauxe gertatuko litzateke:

```
1=Kaixo=1
_
```

`StrCopy()` funtzioa eta `StrLCopy()` funtzioa berdintsuak dira, azken honek ere NULL amaiera duen kate bati balioak emateko erabil daiteke, baina kopiatuko den karaktere kopurua espreski adierazi behar da hirugarren parametro baten bitartez. Hau litzateke `StrLCopy()` funtzioaren goiburukoa:

```
FUNCTION StrLCopy (Helburua, Datua : PChar; Kopurua : Word) : PChar ;
```

[10] `PChar` datu-mota erakusle bat da, NULL amaiturik dagoen karaktere-kate baten erakuslea hain zuzen ere. Erakusle datu-mota 13. kapituluko gaia da.

[11] `Array` datu-motako aldagai baten edukia pantailaratzeko egin behar dena 11. kapituluan ikasiko dugu, baina aurrera dezagun NULL karakterez bukaturiko kateak array bereziak direlako egin daiteke adibide honetako `WriteLn()` hori.

Hona hemen `StrLCopy()` funtzioa ulertzeko balio duen adibide-programa:

```
PROGRAM StrLCopy_Adibidea ;                { \TP70\09\NULL_04.PAS }
USES
   Strings, Crt ;
VAR
   NullKatea : ARRAY [0..25] OF Char ;
BEGIN
   ClrScr ;
   StrLCopy (NullKatea, 'Kaixo, zer moduz zaude?', 4) ;
   WriteLn ('1=', NullKatea, '=1') ;
END.
```

`StrLCopy_Adibidea` programan daukagun datua `'Kaixo, zer moduz zaude?'` katea da eta kopiatuko diren karaktereak 4 direnez, bere irteera hauxe izango da:

```
1=Kaix=1
_
```

9.3.3 `StrCat` eta `StrLCat` funtzioak

`StrCat()` eta `StrLCat()` funtzioak kateen loturak egiteko erabil daitezke.

Hau da `StrCat()` funtzioaren adibide-programa bat:

```
PROGRAM StrCat_Adibidea ;                  { \TP70\09\NULL_05.PAS }
USES
   Strings, Crt ;
VAR
   Bat : PChar ;
   Bi : PChar ;
   NullKatea : ARRAY [0..25] OF Char ;
BEGIN
   ClrScr ;
   Bat := 'Bat' ;
   Bi := 'Bi' ;
   StrCopy (NullKatea, Bat) ;
   WriteLn ('1=', NullKatea, '=1') ;
   StrCat (NullKatea, ' eta ') ;                    { kateaketa }
   WriteLn ('2=', NullKatea, '=2') ;
   StrCat (NullKatea, Bi) ;                         { kateaketa }
   WriteLn ('3=', NullKatea, '=3') ;
END.
```

Eta berari dagokion irteera:

```
1=Bat=1
2=Bat eta =2
3=Bat eta Bi=3
_
```

`StrCat()` funtzioaren goiburukoa hauxe da; non `Helburua` katea sarrera/irteerako parametroa den, eta `Datua` katea berriz sarrera parametroa den. Funtzioaren emaitza den `Helburua+Datua` kateaketa `Helburua` katean gordetzen da:

```
FUNCTION StrCat (Helburua, Datua : PChar) : PChar ;
```

`StrCat()` funtzioak ez dituenez kateen luzerak konprobatzen; `Helburua` kateak izan behar duen luzera minimoa honela kalkulatzen da: `StrLen(Helburua)+StrLen(Datua)+1`.

`StrCat()`–ren adibidea ikusi ondoren `StrLCat()` erraz ulertzen da. Kateaketa burutzean hirugarren parametroak adierazten du azken emaitzak zenbat karaktere izango dituen. Ikus `StrLCat()` funtzioaren goiburukoa, adibide-programa eta bere egikaritzapena:

```
FUNCTION StrLCat (Helburua, Datua : PChar; Kopurua : Word) : PChar ;
```

```
PROGRAM StrLCat_Adibidea ;                  { \TP70\09\NULL_06.PAS }
USES
   Strings, Crt ;
VAR
   NullKatea : ARRAY [0..25] OF Char ;
BEGIN
   ClrScr ;
   StrCopy (NullKatea, 'Kaixo') ;
   WriteLn ('1=', NullKatea, '=1') ;
   StrLCat (NullKatea, ', zer moduz?', 3) ;
   WriteLn ('2=', NullKatea, '=2') ;
   StrLCat (NullKatea, ', zer moduz?', 13) ;
   WriteLn ('3=', NullKatea, '=3') ;
END.
```

```
1=Kaixo=1
2=Kaixo=2
3=Kaixo, zer mo=3
_
```

9.3.4 `StrComp`, `StrIComp`, `StrLComp` eta `StrLIComp` funtzioak

Lau funtzio horien helburua karaktere-kateak konparatzea da. Karaktere-kateak konparatzean erabiltzen den irizpidea **9.1.2.3.2 Kateen arteko konparaketak** puntuan sakonki ikusi dugu `String` datu-motarentzat, eta gauza bera esan daiteke NULL karakterez bukaturiko kateei buruz, hots, kate biren karaktereak binaka hartu eta ASCII taula aintzat izanik burutzen direla konparaketak. Beraz, sei eragileak bilduz adibide-taula hau jar daiteke, non hirugarren zutabeko espresioek `TRUE` balio duten:

Eragilea	Eragiketa	Adibidea
>	Handiago baino	`'Eneko' > 'Asier'`
<	Txikiago baino	`'Jaime' < 'Jaio'`
>=	Handiago edo berdin	`'Olatz' >= 'B'`
<=	Txikiago edo berdin	`'Jokine' <= 'jokin'`
=	Berdin	`'Beloki' = 'Beloki'`
<>	Desberdin	`'BELOKI' <> 'Beloki'`

`StrComp()` funtzioak kate bi konparatzen ditu zenbaki osoa itzuliz.

`StrIComp()` funtzioak kate bi konparatzen ditu maiuskulak eta minuskulak aintzat hartu gabe.

`StrLComp()` funtzioak kate bi konparatzen ditu luzera baten muga barruan.

`StrLIComp()` funtzioak kate bi konparatzen ditu emaniko luzera baten muga barruan, baina maiuskulak eta minuskulak aintzat hartu gabe.

`StrComp()` funtzioaren goiburukoa eta itzultzen duenaren azalpena:

```
FUNCTION StrComp (Kate1, Kate2 : PChar) : Integer ;
```

`StrCopy()` funtzioaren bitartez `Kate1` eta `Kate2` kateak konparatuko dira, funtzioak 0 itzultzen badu `Kate1` eta `Kate2` kateek eduki bera daukate, funtzioaren emaitza negatiboa denean `Kate1 < Kate2` delako da, eta, funtzioaren emaitza positiboa izatean `Kate1 > Kate2` adierazten dio funtzioak modulu deitzaileari. Laburbilduz:

Funtzioaren emaitza	Esanahia
StrComp(Kate1,Kate2) < 0	Kate1 < Kate2
StrComp(Kate1,Kate2) = 0	Kate1 = Kate2
StrComp(Kate1,Kate2) > 0	Kate1 > Kate2

Jarraian `StrComp()` funtzioaren adibide-programa bat eta balizko egikaritzapen bi erakusten ditugu:

```
PROGRAM StrComp_Adibidea ;                  { \TP70\09\NULL_07.PAS }
USES
   Strings, Crt ;
VAR
   Emaitza : Integer ;
   NullKate1, NullKate2 : ARRAY [0..25] OF Char ;
   Irteera : PChar ;
BEGIN
   ClrScr ;
   Write ('Kate bat eman--->') ;
   ReadLn (NullKate1) ;
   Write ('Beste kate bat-->') ;
   ReadLn (NullKate2) ;

   Emaitza := StrComp (NullKate1, NullKate2) ;

   IF Emaitza < 0 THEN
     Irteera := '" txikiago "'
   ELSE
     IF Emaitza > 0 THEN
       Irteera := '" handiago "'
     ELSE
       Irteera := '" berdin "' ;

   WriteLn ('"', NullKate1, Irteera, NullKate2, '"') ;
END.
```

```
Kate bat eman--->Alex
Beste kate bat-->Luis
"Alex" txikiago "Luis"
_
```

```
Kate bat eman--->Luis
Beste kate bat-->LUIS
"Luis" handiago "LUIS"
_
```

`StrComp_Adibidea` adibide-programan `StrComp()` konparaketa funtzioa aldatuko bagenu `StrIComp()` idatziz, ez lirateke maiuskula eta minuskulak desberdinduko horregatik programa aldatuarekin (`StrIComp_Adibidea` deitu duguna) emaitza hau lortuko genuke:

```
Kate bat eman--->Luis
Beste kate bat-->LUIS
"Luis" berdin "LUIS"
_
```

`StrLComp()` funtzioaren goiburukoan `Kopurua` deitu dugun hirugarren parametro bat agertzen da, dakigunez kateen konparaketa karaktereka egiten da eta `Kopurua` parametro horrek mugatzen du zenbat karaktere hartuko diren kate bakoitzetik konparaketa burutzeko (konparaketa ebaluatzeko, karaktereak bikoteka hartzen dira kateen ezkerretik hasita):

```
FUNCTION StrLComp (Kate1, Kate2 : PChar; Kopurua : Word) : Integer ;
```

`StrLCopy()` funtzioaren bitartez `Kate1` eta `Kate2` konparatuko dira, kateen lehen `Kopurua` posizioak kontsideratuz. Funtzioaren emaitza lehen bezala:

Funtzioaren emaitza	Esanahia
StrLComp(Kate1,Kate2, Kopurua) < 0	Kate1 < Kate2
StrLComp(Kate1,Kate2, Kopurua) = 0	Kate1 = Kate2
StrLComp(Kate1,Kate2, Kopurua) > 0	Kate1 > Kate2

Hau litzateke `StrLComp()` funtzioaren adibide-programa bat eta balizko exekuzioak:

```
PROGRAM StrLComp_Adibidea ;                { \TP70\09\NULL_09.PAS }
USES
   Strings, Crt ;
CONST
   LUZERA_MAX = 3 ;
VAR
   NullKate1, NullKate2 : ARRAY [0..25] OF Char ;
   Irteera : PChar ;
BEGIN
   ClrScr ;
   Write ('Kate bat eman--->') ;
   ReadLn (NullKate1) ;
   Write ('Beste kate bat-->') ;
   ReadLn (NullKate2) ;

   IF StrLComp (NullKate1, NullKate2, LUZERA_MAX) = 0 THEN
     Irteera := ' berdinak dira'
   ELSE
     Irteera := ' desberdinak dira' ;

   Write ('"', NullKate1, '" eta "', NullKate2, '"-(r)en lehen ') ;
   WriteLn (LUZERA_MAX, ' karaktereak' ,Irteera) ;
END.
```

```
Kate bat eman--->Alex
Beste kate bat-->Alexader
"Alex" eta "Alexander"-(r)en lehen 3 karaktereak berdinak dira
_
```

```
Kate bat eman--->ALEX
Beste kate bat-->Alexader
"ALEX" eta "Alexander"-(r)en lehen 3 karaktereak desberdinak dira
_
```

Bigarren egikaritzapenean `ALEX` katearen lehen hiru karaktereak ez dira berdinak `Alexander` katearen lehen hirurekin alderatzean, horregatik `StrLComp()` funtzioak 0 ez den zerbait itzuliko du. Kate biren lehen hiru karaktereak konparatzeko maiuskulak eta minuskulak ote diren arduratu gabe, `StrLIComp()` funtzioa aplikatuko genuke hau lortuz:

```
Kate bat eman--->ALEX
Beste kate bat-->Alexader
"ALEX" eta "Alexander"-(r)en lehen 3 karaktereak berdinak dira
_
```

9.3.5 `StrLower` eta `StrUpper` funtzioak

Karaktere-kate bat minuskuletara eta maiuskuletara bihurtzeko balio dute funtzio horiek hurrenez hurren. Funtzio biren goiburukoak aztertuz sarrerako karaktere-katea aldatu egiten dela kontura gaitezke:

```
FUNCTION StrLower (Katea : PChar) : PChar ;
FUNCTION StrUpper (Katea : PChar) : PChar ;
```

Hau da `StrLower()` eta `StrUpper()` funtzioak erabiltzen dituen adibide-programarik errazena:

```
PROGRAM StrLower_StrUpper ;               { \TP70\09\NULL_11.PAS }
USES
   Strings, Crt ;
VAR
   NullKatea : ARRAY [0..225] OF Char ;
BEGIN
   ClrScr ;
   Write ('Kate bat eman-->') ;
   ReadLn (NullKatea) ;
   WriteLn ('1-->', StrLower(NullKatea), '<--1') ;
   WriteLn ('2-->', StrUpper(NullKatea), '<--2') ;
END.
```

`StrLower_StrUpper` programaren exekuzio bat:

```
Kate bat eman-->NULL Karakterez Bukaturiko Kateak Errazak Dira
1-->null karakterez bukaturiko kateak errazak dira<--1
2-->NULL KARAKTEREZ BUKATURIKO KATEAK ERRAZAK DIRA<--2
_
```

9.3.6 `StrPas` eta `StrPCopy` funtzioak

`String` datu-motako karaktere-kateak Pascal formatua dutela esan dugu, eta NULL karakterez bukatzen diren kateei kontrajartzen dira.

`StrPas()` eta `StrPCopy()` funtzioek formatu biren arteko trukaketak egiteko balio dute. `StrPas()` funtzioak NULL karakterez bukatzen den katea hartu eta string bihurtzen du; `StrPCopy()` funtzioak berriz, string batetik abiatuta dagokion NULL karakterez amaitzen den katea ematen du.

Ikus goiburukoak:

```
FUNCTION StrPas (Datua : PChar) : String ;
FUNCTION StrPCopy (Emaitza : PChar; Datua : String) : PChar ;
```

`StrPas_Adibidea` izena duen programan `NullKatea` aldagaia teklatuz irakurtzen da, eta `StrPas()` funtzioari esker string bat den `Katea` aldagaiak balio hori hartzen du, pantailaraketan ikusten da sarrerako datua `Katea` aldagaira transferitu dela.

`StrPCopy_Adibidea` izeneko programan string bat den `Katea` aldagaia teklatuz irakurri ondoren, `StrPCopy()`-ren bitartez bere balioa NULL karakterez bukaturiko `NullKatea` aldagaira igarotzen da.


```
                { \TP70\09\NULL_12.PAS }
PROGRAM StrPas_Adibidea ;
USES
   Strings, Crt ;
VAR
   ClrScr ;
   NullKatea : ARRAY [0..25] OF Char ;
   Katea : String [25] ;
BEGIN
   Write ('Izen bat eman--->') ;
   ReadLn (NullKatea) ;
   Katea := StrPas (NullKatea) ;
   Write ('--->', Katea, '<---') ;
   WriteLn (Length(Katea), ' letra')
END.
```

```
                { \TP70\09\NULL_13.PAS }
PROGRAM StrPCopy_Adibidea ;
USES
   Strings, Crt ;
VAR
   ClrScr ;
   NullKatea : ARRAY [0..25] OF Char ;
   Katea : String [25] ;
BEGIN
   Write ('Izen bat eman--->') ;
   ReadLn (Katea) ;
   StrPCopy (NullKatea, Katea) ;
   Write ('--->', NullKatea, '<---') ;
   WriteLn (StrLen(NullKatea), ' letra')
END.
```

`StrPas_Adibidea` eta `StrPCopy_Adibidea` programa biek irteera bera ematen dute sarrera berdina izanez gero. Hona hemen:

```
Kate bat eman--->Zuriñe
--->Zuriñe<---6 letra
_
```

9.3.7 `StrPos` funtzioa

`StrPos()` funtzioaren helburua azpikate bat kate batean ote dagoen aztertzea da. Hauxe da bere goiburukoa, non `Katea` eta `Azpikatea` parametroak sarrerakoak diren eta funtzioaren emaitza erakusle bat den:

```
FUNCTION StrPos (Katea, Azpikatea : PChar) : PChar ;
```

`Azpikatea` deituriko parametroaren edukia `Katea` parametroaren edukiaren barruan aurkituko balitz, `StrPos()` funtzioak lehen agerpenari erakusten dion `PChar` datu-motako erakusle bat itzultzen dio modulu deitzaileari. Baina, `Katea` parametroak `Azpikatea` deituriko parametroaren edukia barneratzen ez badu, `StrPos()` funtzioak itzultzen duen erakuslea `NIL` izango da.

Hona hemen `StrPos()` funtzioa aplikatzen duen adibide-programa bat:

```
PROGRAM StrPos_Adibidea ;                { \TP70\09\NULL_14.PAS }
USES
   Strings, Crt ;
VAR
   NullKate : ARRAY [0..225] OF Char ;
   NullAzpikate : ARRAY [0..55] OF Char ;
   Irteera : PChar ;
BEGIN
   ClrScr ;
   Write ('Kate bat eman--->') ;
   ReadLn (NullKate) ;
   Write ('Azpikatea eman-->') ;
   ReadLn (NullAzpikate) ;
   Irteera := StrPos (NullKate, NullAzpikate) ;
   IF Irteera = NIL THEN
     WriteLn ('"', NullAzpikate, '" azpikatea ez dago "', NullKate, '" katean')
   ELSE
     BEGIN
        Write ('"', NullAzpikate, '" azpikatearen lehen agerpena ') ;
        WriteLn (Irteera - NullKate, ' indizean') ;
     END ;
END.
```

`StrPos_Adibidea` programaren egikaritzapen biren irteerak:

```
Kate bat eman--->eder basoan haritza, haran aldean garitza ...
Azpikatea eman-->basoa
"basoa" azpikatearen lehen agerpena 5 indizean
_
```

```
Kate bat eman--->eder basoan haritza ...
Azpikatea eman-->basoaren
"basoaren" azpikatea ez dago "eder basoan haritza ..." katean
_
```

9.3.8 `StrECopy` funtzioa

`StrECopy()` funtzioak kate bat beste kate batean kopiatzen du eta emaitzaren amaierari erakusten dion erakuslea itzultzen du:

```
FUNCTION StrECopy (Helburua, Datua : PChar) : PChar ;
```

Parametroen azalpena banaka:

`Helburua` emaitza da, funtzioa deitu ondoren hartuko du balioa kate honek.

`Datua` parametro hau sarrerakoa da, hots, `StrECopy()` deitu aurretik `Datua` kateak balio ezagunen bat izango du.

`StrECopy()`-ren bitartez kateaketak egin daitezke hurrengo `StrECopy_Adibidea` programan erakusten den bezala:

```
PROGRAM StrECopy_Adibidea ;                { \TP70\09\NULL_15.PAS }
USES
   Strings, Crt ;
VAR
   Bat, Bi, Hiru : PChar ;
   NullKate1, NullKate2 : ARRAY [0..25] OF Char ;
BEGIN
   ClrScr ;

   Bat := 'BAT' ;
   Bi := 'BI' ;
   StrECopy ( StrECopy ( StrECopy (NullKate1, Bat), ' eta '), Bi) ;
   WriteLn ('1=', NullKate1, '=1') ;

   Hiru := 'HIRU' ;
   StrECopy ( StrECopy ( StrECopy (NullKate2, NullKate1), ' berdin '), Hiru) ;
   WriteLn ('2=', NullKate2, '=2') ;

   StrECopy (NullKate2, 'ADOS?') ;
   WriteLn ('3=', NullKate2, '=3') ;
END.
```

`StrECopy_Adibidea` programaren irteera:

```
1=BAT eta BI=1
2=BAT eta BI berdin HIRU=2
3=ADOS?=3
_
```

9.4 PROGRAMAK

Hona hemen 9. kapituluaren programak orrialdeen arabera sailkaturik:

Izena	*Programaren identifikadorea*	*ORRI.*	*Ikasgaia*
STRING1.PAS	StringDatuMotenLuzera	9-03	String datu-mota
STRING2.PAS	StringenEsleipena	9-05	String datu-mota
STRING3.PAS	StringenKateaketa	9-10	String datu-mota
STRING4.PAS	Karaktereka1	9-11	String datu-mota
STRING5.PAS	KatearenLuzera	9-12	String datu-mota
STRING6.PAS	Karaktereka2	9-13	String datu-mota
STRING7.PAS	Karaktereka3	9-13	String datu-mota
STRING8.PAS	ZeroakTartekatzen	9-15	String datu-mota
STRING9.PAS	DatuMotenTrukaketaStr	9-18	String datu-mota
STRING10.PAS	DatuMotenTrukaketa	9-20	String datu-mota
STRING11.PAS	KaraktereBanaketa1	9-22	String datu-mota
STRING12.PAS	KaraktereBanaketa2	9-23	String datu-mota
NULL_00.PAS	Str_Karaktereka	9-25	NULL formatuko katea
NULL_01.PAS	StrLen_Adibidea	9-26	NULL formatuko katea
NULL_02.PAS	StrEnd_Adibidea	9-26	NULL formatuko katea
NULL_03.PAS	StrCopy_Adibidea	9-27	NULL formatuko katea
NULL_04.PAS	StrLCopy_Adibidea	9-28	NULL formatuko katea
NULL_05.PAS	StrCat_Adibidea	9-28	NULL formatuko katea
NULL_06.PAS	StrLCat_Adibidea	9-29	NULL formatuko katea
NULL_07.PAS	StrComp_Adibidea	9-30	NULL formatuko katea
NULL_08.PAS	StrIComp_Adibidea	9-30	NULL formatuko katea
NULL_09.PAS	StrLComp_Adibidea	9-31	NULL formatuko katea
NULL_10.PAS	StrLIComp_Adibidea	9-31	NULL formatuko katea
NULL_11.PAS	StrLower_StrUpper	9-32	NULL formatuko katea
NULL_12.PAS	StrPas_Adibidea	9-33	NULL formatuko katea
NULL_13.PAS	StrPCopy_Adibidea	9-33	NULL formatuko katea
NULL_14.PAS	StrPos_Adibidea	9-33	NULL formatuko katea
NULL_15.PAS	StrECopy_Adibidea	9-34	NULL formatuko katea

9.5 BIBLIOGRAFIA

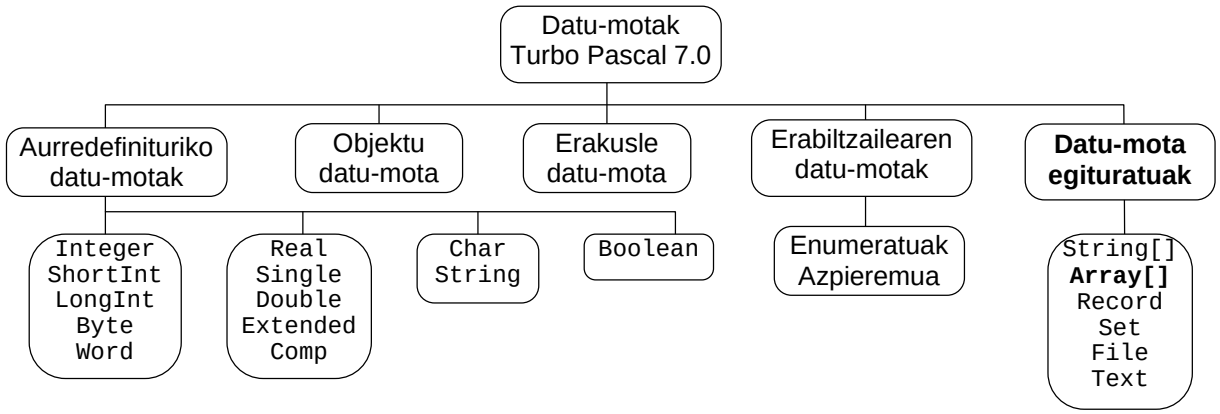
- Borland, *TURBO PASCAL 7.0. Erabiltzailearen gida*, Borland International, 1992
- Borland, *TURBO PASCAL 7.0 Ingurunearen laguntza*, Borland International, 1992
- Joyanes, *TURBO PASCAL 6.0 a su alcance*, McGraw-Hill, 1993

10. ATALA: ARRAY DATU-MOTA

AURKIBIDEA

10.1 SARRERA

Laugarren kapituluaren azken puntuetan *datu-mota egituratuak* aipatu izan ziren eta jarraian ematen den eskema aurkeztu genuen. Ezkerragora dagoen abarra, aurredefinituriko datu-moteena, aspaldian hasi ginen lantzen eta ikus daitekeenez eta gogora dezakegunez lengoaiaren oinarrizko datu-motak biltzen ditu. Hurrengo bi abarrak, objektuak eta erakusleak, 14. eta 13. kapituluetan azalduko ditugu. Erabiltzailearen datu-motak 8. kapituluan ikasi genituen. Eta amaitzeko, azken abarraren (datu-mota egituratuen) string edo karaktere-kateak 9. kapituluaren gaia izan zen; ikus dezagun orain array datu-mota zertan den (Record, set, file eta text datu motak 11. eta 12. kapituluetan ikasiko dira).



Kurtsoaren zehar ikusi dugun bezala ordenadore baten helburua informazioaren tratamendua da, hots, sarrerako informazioa landuz, datuak landuz, irteerako informazioa edo emaitzak lortzea. Baina datuak askotan multzotan etortzen dira, adibidez gela baten ikasleen notak, hau da ikasleak hirurogei baldin badira euren notak hirurogei zenbaki erreal izango dira eta astakeria litzateke `Nota1`, `Nota2`, `Nota3`, ... `Nota60` aldagai isolatuak[1] deklaratzea. Horren ordez hirurogei zenbakiak biltzen dituen datu-mota konplexuago bat erabiltzen da; nolabait esateko karaktereen bilkura den string datu-motarekin egiten zen bezala, zenbaki errealak diren notak elkarrekin biltegitzeko eta lantzeko array datu-mota dago.

Array batek, sinpleagoak diren beste elementu batzuk biltzen dituenez egitura (estruktura) jakin bat izango du, hori dela eta array datu-motari egituratua esaten zaio. Array batek izen edo identifikadore bakar batez hainbat elementuen multzoa errepresentatzen du, ikusiko dugun bezala operazioak array osoari edo arrayaren elementu zehatzari aplika dakizkioke.

10.1.1 Definizioa

Array datu-motaren definizioa eman dezagun. Datu-egitura estatikoa[2] 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.

[1] Pentsa dezagun hirurogei balio erreal horiei aplikatuko zaizkien operazioak berdintsuak izango direla. Izan ere, hirurogei aldagai errealekin noten informazioa izango genuke baina *gelaren* ideia ez litzateke isladatuko.

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

Array datu-mota definitzeko erabili ditugun kontzeptuak sei izan dira:

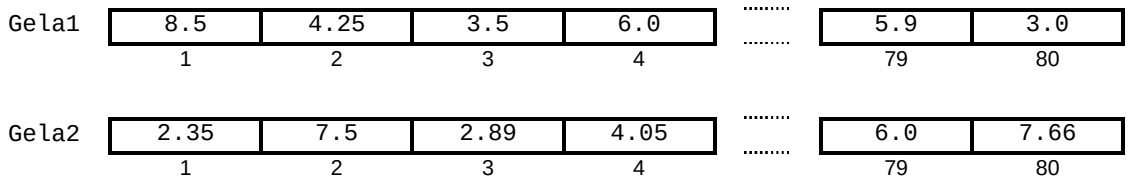
1. Arraya datu-mota egituratua dela, hots, zenbait elementu biltzen dituela.
2. Arraya datu-mota estatikoa dela. Hau da, programa edo errutina baten hasieran memorian array aldagai bat sortu ondoren programa (edo errutina) amaitu arte ez dira arrayaren memori-posizioak aldatuko.
3. Arraya finitua denez mugatuta dago. Array bat deklaratzean zenbat elementu onar dezakeen zehaztu beharra dago, eta kopuru hori aldatzerik ez dago programaren exekuzioan.
4. Arraya datu mota homogenoa da. Honek esan nahi duena zera da, arrayaren elementu guztiak mota berekoak direla.
5. Array datu-motaren elementuak elkar ondoan kokatzen dira memorian egitura lineal bat osatuz.
6. Array baten edozein elementu izendatzeko indize bat[3] erabiltzen da.

Ondorioz array datu-mota bat deklaratzean, besteak beste, elementuei dagokien datu-mota explizitoki jartzearekin batera lehen elementua eta azken elementua izendatzeko indizeak erazagutuko dira. Demagun array aldagaiak deklaratzeko `DM_Zerrenda` izeneko datu-mota sortu nahi dela programa zehatz batean, suposa dezagun `DM_Zerrenda` datu-motako arrayetan zenbaki errealak gordeko direla eta gehienez 80 zenbaki izango direla. Hona hemen `DM_Zerrenda` datu-mota egituratuari dagokion erazagupena:

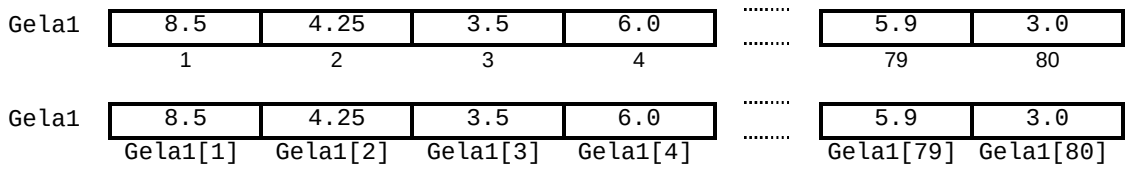
```
TYPE
   DM_Zerrenda = ARRAY[1..80] OF Real ;
VAR
   Gela1, Gela2 : DM_Zerrenda ;
```

Ikusten denez arrayen datu-mota den `DM_Zerrenda` deklaratzean `ARRAY` eta `OF` hitz erreserbatuak jarri behar dira, baina ez da hori garrantzitsuena, kontura gaitezen arraya finitua izango dela adierazten duten bi indize aukeratu beharra dagoela, eta, nola ez, arrayaren oinarrizko elementuen datu-mota zein izango den finkatu egingo dela.

`DM_Zerrenda` datu-mota aintzat harturik, aldagaien erazagupenaren atalean `Gela1` eta `Gela2` arrayak programan erabiliko direla deklaratzen da. Aldagai horietan balioak gorde ondoren euren eskema eginez gero honako zerbait izango genuke:



Ikusten denez `Gela1` eta `Gela2` aldagai egituratuek zenbaki errealen zerrendak biltegitzeko ahalmena dute, ikusten da halaber egitura homogenoak direla eta array baten partaide bakoitzak izen eta indize bana duela, adibidez:



[3] Dimentsio bakarra duten arrayetan (bektoretan) indizea ere bakarra izango da, bi dimentsio dituzten arrayetan (tauletan) indizeak bi izango dira, eta, n dimentsioko arrayetan indizeak ere n izango dira.

Jarraian ematen den `ArrayBatDatuzBete` izeneko programan `Gela1` array[4] bat definitu ondoren balioak gordetzen dira bertan, programa amaitu aurretik arrayaren edukia pantailaratzen da:

```
PROGRAM ArrayBatDatuzBete ;                          { \TP70\10\ARRAY_1.PAS }
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 8 ;
TYPE
  DM_Zerrenda = ARRAY[BEHEMUGA..GOIMUGA] OF Real ;
VAR
  Gela1 : DM_Zerrenda ;
  Kont : Integer ;
BEGIN
   FOR Kont:=BEHEMUGA TO GOIMUGA DO
   BEGIN
      Write (Kont, '. nota eman: ') ;
      ReadLn ( Gela1[Kont] ) ;
   END ;

   WriteLn ;
   WriteLn ('Gela1 arrayaren edukia: ') ;
   FOR Kont:=BEHEMUGA TO GOIMUGA DO
   BEGIN
      Write ( Gela1[Kont] :7:2 ) ;
   END ;

   WriteLn ;
END.
```

`ArrayBatDatuzBete` izeneko programan bi gauza azpimarratuko genituzke. Batetik arrayaren indizeak zenbakizko konstante esplizituak direla `BEHEMUGA` eta `GOIMUGA`, bestetik `Gela1` arrayaren elementu indibidual bat izendatzeko arrayaren identifikadorearekin batera kortxete arteko indize bat jarriko dela.

Hauxe litzateke `ArrayBatDatuzBete` programaren irteera posible bat:

```
1. nota eman: 8.5
2. nota eman: 4.25
3. nota eman: 3.5
4. nota eman: 6
5. nota eman: 7.1
6. nota eman: 8.46
7. nota eman: 5.9
8. nota eman: 3

Gela1 arrayaren edukia:
   8.50   4.25   3.50   6.00   7.10   8.46   5.90   3.00
_
```

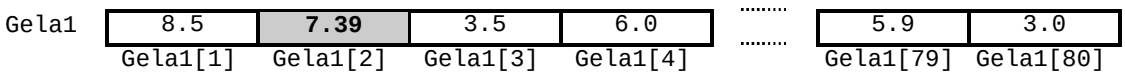
10.1.2 Indizeak

Array baten indizeek betebehar bi dituzte, batetik arrayaren esparrua edo tamaina mugatzen dute eta bestetik arrayaren elementu jakin bat identifikatzeko bidea ematen dute. Horrela, aurreko adibideko `Gela1` arrayaren bigarren elementuaren balioa aldatu nahi izanez gero ondoko esleipen hau jar daiteke:

```
Gela[2] := 7.39 ;
```

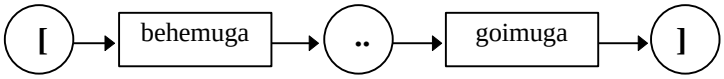
[4] Programaren kodean ikus daitekeenez datu-mota 80 elementukoa izan beharrean 8koa da, bestela datuak sartzeko prozesua astunegia bailiteke.

Ondorioz `Gela1` arrayaren bigarren elementuan zegoen `4.25` balioa ordezkatuta geratuko da honelako arraya geratzen zaigularik:



Indizeei buruzko xehetasunak eman ditzagun:

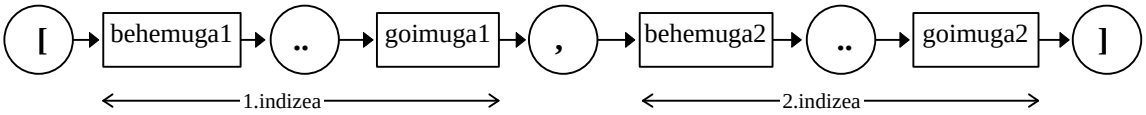
- Indizeek arrayaren behemuga eta goimuga definitzen dituzte, horretarako array datu-motaren deklarazioan molde hau jarraituko da:



  Indizearen mugak definitzean arrayak izan dezakeen elementu kopurua zehazturik geratzen da: Ord(goimuga) - Ord(behemuga) + 1.

  Beraz, behemuga eta goimuga konstanteetarako edozein balio aukeratuz gero array baten elementuen kopurua nahi dugun bezain handia izan daitekeela pentsa daiteke, baina bestelako beharkizunak ere badaude, esate baterako array aldagaia memorian kokatu beharra dagoela eta izango dituen elementuek ezingo dute memoriak berak duen neurri fisikoa gainditu. Areago, DOS sistema eragilerako garatutako programa exekutagarriek datuetarako duten memori zatia zehazki mugaturik dago (datuek gehien jota memoriaren 64 Kb hartzen dute) eta praktikan array batek ezingo du kopuru hori gainditu.

- Indizeek arrayaren dimentsioa markatzen dute. Bektore bat definitzeko (dimentsio bakarra duen arraya) behemuga eta goimuga bikote bakarra behar da, baina taula bat definitzeko (bi dimentsioko arraya) goimuga eta behemuga bikote bi jarriko dira koma batez banaturik:



- Indizeak ordinalak izango dira, hau da indizeei dagokien datu-motak `Integer`, `Byte`, `Char`, `Boolean`, etab. izan daitezke, baina inolaz ere `Real` edo ordinalak ez diren gainerako datu-motak (ikus **10.1.4.2** puntuan garatutako `MajuskulenMaiztasunak` adibidea, non arrayaren indizeak karaktereak diren).

10.1.3 Eragiketak arrayekin

Arrayek onartzen duten eragiketa bakarra esleipena da. Arrayen arteko esleipena adibideekin ikus dezagun.

Lehenago aztertu dugun `ArrayBatDatuzBete` izeneko programan aldaketa txikiak eginez bi arrayen arteko esleipenaren operazioa froga dezakegu, hau da `Gela1` eta `Gela2` array bi definitu ondoren batean balioak gordetzen dira eta esleipena egin ostean programa amaitu aurretik beste arrayaren edukia pantailaratzen da.

Hona hemen `ArrayenEsleipena` deituriko programa, sententziarik garrantzitsuena orain beltzez dagoena da, hots, `Gela2` arrayari `Gela1` arrayaren balioak esleitzen dizkionak:

```
PROGRAM ArrayenEsleipena ;                              { \TP70\10\ARRAY_2.PAS }
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 4 ;
TYPE
  DM_Zerrenda = ARRAY[BEHEMUGA..GOIMUGA] OF Real ;
VAR
  Gela1, Gela2 : DM_Zerrenda ;
  Kont : Integer ;
BEGIN
   WriteLn ('Gela1 arraya betetzen: ') ;
   FOR Kont:=BEHEMUGA TO GOIMUGA DO
   BEGIN
      Write ('Gela1[', Kont, '] nota eman: ') ;
      ReadLn ( Gela1[Kont] ) ;
   END ;

   Gela2 := Gela1 ;                          { arrayen arteko esleipena }

   WriteLn ;
   WriteLn ('Gela2 arrayaren edukia: ') ;
   FOR Kont:=BEHEMUGA TO GOIMUGA DO
   BEGIN
      Write ('   Gela2[', Kont, '] = ', Gela2[Kont] :0:2 ) ;
   END ;

   WriteLn ;
END.
```

Hau da `ArrayenEsleipena` programari dagokion irteera bat:

```
Gela1 arraya betetzen:
Gela1[1] nota eman: 5.7
Gela1[2] nota eman: 8.12
Gela1[3] nota eman: 6
Gela1[4] nota eman: 3.5

Gela2 arrayaren edukia:
   Gela2[1] = 5.70   Gela2[2] = 8.12   Gela2[3] = 6.00   Gela2[4] = 3.50
_
```

Irteeran erabat argi geratzen da `Gela1` eta `Gela2` arrayen arteko esleipena egin ondoren zerrenda batek eta besteak dituzten balioak berdinak direla.

Aipatzekoa da esleipena “anai bikoitzak” diren arrayen artean egin daitekeela soilik, array bi “anai bikoitzak” izango dira baldin eta biek datu-mota bera badaukate, adibidez `ArrayenEsleipenOkerra1` deituriko programa honetan `Gela1` eta `Gela2` arrayak errealen zerrendak dira biak baina datu-motei begiratzen badiegu array horiek parekagarriak ez dira, izan ere `Gela1` array aldagaiak bost kopuru erreal biltegitzen du eta dagokion datu-mota `DM_Zerrenda1` da, bestalde, `Gela2` arraya `DM_Zerrenda2` datu-motaz definitu da zeinek gehienez lau erreal onartzen dituen:

```
PROGRAM ArrayenEsleipenOkerra1 ;                        { \TP70\10\ARRAY_3.PAS }
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 4 ;
TYPE
  DM_Zerrenda1 = ARRAY[BEHEMUGA..GOIMUGA+1] OF Real ;
  DM_Zerrenda2 = ARRAY[BEHEMUGA..GOIMUGA] OF Real ;
VAR
  Gela1 : DM_Zerrenda1 ;
  Gela2 : DM_Zerrenda2 ;
  Kont : Integer ;
```

```
BEGIN
   WriteLn ('Gela1 arraya betetzen: ') ;
   FOR Kont:=BEHEMUGA TO GOIMUGA+1 DO
   BEGIN
      Write ('Gela1[', Kont, '] nota eman: ') ;
      ReadLn ( Gela1[Kont] ) ;
   END ;

   Gela2 := Gela1 ;            { arrayen arteko esleipen desegokia }

   WriteLn ;
   WriteLn ('Gela2 arrayaren edukia: ') ;
   FOR Kont:=BEHEMUGA TO GOIMUGA DO
   BEGIN
      Write ('   Gela2[', Kont, '] = ', Gela2[Kont] :0:2 ) ;
   END ;

   WriteLn ;
END.
```

Hori dela eta, `ArrayenEsleipenOkerra1` programa konpilatzean errorea detektatuko da `Gela1` eta `Gela2` arrayen arteko esleipena ezinezkoa baita, honelako mezu aski ezaguna erakutsiz: `Error 26: Type mismatch.`

Ondorio bera lor genezake arrayen desberdintasuna tamainan izan gabe elementuen datu-motan baldin badago, adibidez:

```
PROGRAM ArrayenEsleipenOkerra2 ;                    { \TP70\10\ARRAY_4.PAS }
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 4 ;
TYPE
  DM_Zerrenda1 = ARRAY[BEHEMUGA..GOIMUGA] OF Integer ;
  DM_Zerrenda2 = ARRAY[BEHEMUGA..GOIMUGA] OF Real ;
VAR
  Gela1 : DM_Zerrenda1 ;
  Gela2 : DM_Zerrenda2 ;
  Kont : Integer ;
BEGIN
   WriteLn ('Gela1 arraya betetzen: ') ;
   FOR Kont:=BEHEMUGA TO GOIMUGA DO
   BEGIN
      Write ('Gela1[', Kont, '] nota eman: ') ;
      ReadLn ( Gela1[Kont] ) ;
   END ;

   Gela2 := Gela1 ;            { arrayen arteko esleipen desegokia }
```

Kontutan izan zenbaki osoak dituen `Gela1` arrayaren elementuak banan-banan esleituz gero, zenbaki errealeko `Gela2` arraya bete daitekeela: Baina hori ez litzateke arrayen arteko operazioa, arrayen elementuen arteko operazioa baizik (ikus jarraian datorren **10.1.4.1** puntuan erakusten den `ArrayBiBerdinak` programa eta bere irteera).

10.1.4 Eragiketak arrayen elementuekin

Array baten elementuei aplikatuko zaizkien eragiketak elementu horiei dagokien datu-motak onartzen dituen eragiketak izango dira. Hau da, zenbaki errealak biltzen dituen `Notak` array bat izanik bere elementuei ezin zaie `Div` zatiketa osoa aplikatu, zenbaki errealen datu-motak hori galarazten duelako. Beste adibide bat jarriz gero, demagun karakterez osaturiko

`Letrak` array bat daukagula, 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).

```
TYPE
   DM_Notak = ARRAY[1..70] OF Real ;
   DM_Letrak = ARRAY[1..5] OF Char ;

VAR
   Notak : DM_Notak ;
   Letrak : DM_Letrak ;
```

Honezkero, jarraian ematen diren lerroetan erroreak daude:

```
ZbkOsoa := Notak[15] Div 2 ;
ZbkOsoa := Letrak[2] * 5 ;
EmaitzaBoolear := Letrak[2] AND Amaiturik ;
```

Arrayen elementuekin egin daitezkeen operazioen adibideak jar ditzagun ere, jarraian datozen puntuetan ariketa bi erakusten dira.

10.1.4.1 Adibidea

Demagun bi array ditugula eta berdinak ote diren jakin nahi dugula, array bi berdinak kontsideratzeko baten eta bestearen edukia bera izango da. Deskribaturiko problema hau ebazteko bi arrayak jasoko dituen funtzio boolear bat plantea genezake, baina oraindik ez dakigu datu-mota egituratuak parametro bezala erabiltzen horregatik lan guztia programa nagusian burutuko dugu.

Hona hemen `ArrayBiBerdinak` programaren sententziak, non `Gela1` eta `Gela2` array bikotea definitu den. Konturatzen bagara, posible da array aldagaiak sortzea `DM_Zerrenda` datu-mota espezifikoa erabili gabe, hala ere programadorearen `DM_Zerrenda` datu-mota edukitzeak abantailak ditu eta gure testu honetan gehienetan bide hori jorratuko dugu. Array biren konparaketa `IF Gela1=Gela2 THEN` baten bitartez ezin daitekeenez egin arrayen elementuak bikoteka hartuz azterketa partzialak egingo dira:

```
PROGRAM ArrayBiBerdinak ;                          { \TP70\10\ARRAY_5.PAS }
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 4 ;
VAR
  Gela1, Gela2 : ARRAY[BEHEMUGA..GOIMUGA] OF Real ;
  Kont : Integer ;
  BerdinakBai : Boolean ;
BEGIN
   WriteLn ('Gela1 arraya betetzen: ') ;
   FOR Kont:=BEHEMUGA TO GOIMUGA DO
   BEGIN
      Write ('Gela1[', Kont, '] nota eman: ') ;
      ReadLn ( Gela1[Kont] ) ;
   END ;

   WriteLn ;
   WriteLn ('Gela2 arraya betetzen: ') ;
   FOR Kont:=BEHEMUGA TO GOIMUGA DO
   BEGIN
      Write ('Gela2[', Kont, '] nota eman: ') ;
      ReadLn ( Gela2[Kont] ) ;
   END ;
```

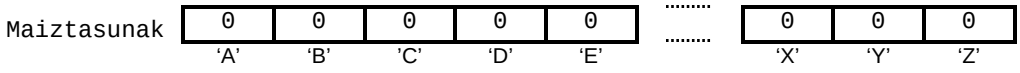
```
   { Arrayen arteko konparaketa burutu, hau da Gela1
     arrayaren edukia eta Gela2-rena berdina ote den }

   BerdinakBai := TRUE ;
   Kont := BEHEMUGA ;
   WHILE (Kont <= GOIMUGA) AND BerdinakBai DO
   BEGIN
      IF Gela1[Kont] <> Gela2[Kont] THEN
           BerdinakBai := FALSE ;
      Kont := Kont + 1 ;
   END ;
   IF BerdinakBai THEN
     WriteLn ('Gela1 eta Gela2 arrayak berdinak dira')
   ELSE
     WriteLn ('Gela1 eta Gela2 arrayak desberdinak dira') ;
END.
```

Zergatik uste duzu `ArrayBiBerdinak` programan, `Gela1[Kont]` zenbaki erreala eta dagokion `Gela2[Kont]` bikote kidea, konparaketa errepikakorrak egiteko `WHILE-DO` kontrol-egitura erabili da eta ez errazagoa den `FOR-DO` kontrol-egitura?.

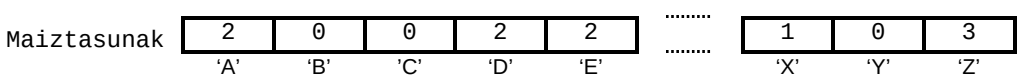
10.1.4.2 Adibidea

Demagun majuskuletan idatzitako esaldi bat string baten bitartez memorian gordetzen dela, eta programaren bitartez sarrien (maizen) agertzen den alfabetoko letra ezagutu nahi dugula. Biderik zuzenena letra guztien agerpen kopurua esaldian, euren maiztasuna, array batean erregistratzea litzateke eta ondoren kopuruen artean handiena bilatu beharko genuke.

Esaldiaren letren maiztasunak gordeko dituen arrayaren itxura honako hau litzateke, non indizeak alfabetoko karaktereak diren eta arrayaren oinarrizko elementuen datu-mota zenbaki osoarena den (letra baten agerpen kopurua zenbaki osoa baita):



`Maiztasunak` izeneko arraya karaktereen agerpen kopurua zenbatzeko baldin bada, hasiera batean, datua den esaldia aztertu baino lehen, maiztasun guztiak zero izango dira. Baina letra bakoitzaren maiztasuna zenbatu ondoren `Maiztasunak` izeneko arrayaren edukia bestelako izango da, esate baterako esaldia `'KAIXO, ZER MODUZ ZAUDE?'` izanik arrayaren edukia hauxe izango litzateke:



Adibide horretan sarrien agertzen den karakterea `'Z'` da, eta kontutan izan hiru aldiz errepikatzen den zuriune karakterearen maiztasuna `Maiztasunak` arrayan ez dela gordetzen (gauza bera esan daiteke `','` eta `'?'` karaktereetako).

Array baten elementuak karakterez izendatzeko behemuga eta goimuga karaktere konstanteak izango dira zenbaki osoak izan ordez, adibidearen kasurako honela definituriko genuke `DM_Maiztasunak` datu-mota eta dagokion `Maiztasunak` aldagaia:

```
TYPE
   DM_Maiztasunak = ARRAY['A'..'Z'] OF Byte ;
VAR
   Maiztasunak : DM_Maiztasunak ;
```


Hona hemen `MajuskulenMaiztasunak` programa:

```
PROGRAM MajuskulenMaiztasunak ;                              { \TP70\10\ARRAY_6.PAS }
USES
  Crt ;
TYPE
  DM_Maiztasunak = ARRAY['A'..'Z'] OF Byte ;
VAR
  Esaldia : String ;
  Maiztasunak : DM_Maiztasunak ;
  ZnbkiKont, Kopurua : Integer ;
  KarakKont, LetraLaguntzaile, Maizena : Char ;
BEGIN
   ClrScr ;
   Write ('Esaldi bat eman------->') ;
   ReadLn (Esaldia) ;

   { Datua den esaldia majuskuletan jarri }
   FOR ZnbkiKont:=1 TO Length(Esaldia) DO
   BEGIN
      Esaldia[ZnbkiKont] := UpCase(Esaldia[ZnbkiKont]) ;
   END ;
   WriteLn ('Esaldia majuskulaz---->', Esaldia) ;

   { Arrayaren hasieraketa. Zati hau ez da beharrezkoa baina bai komenigarria }
   FOR KarakKont:='A' TO 'Z' DO
   BEGIN
      Maiztasunak[KarakKont] := 0 ;
   END ;

   { Esaldiaren azterketa arrayari balio berriak emateko }
   FOR ZnbkiKont:=1 TO Length(Esaldia) DO
   BEGIN
      LetraLaguntzaile := Esaldia[ZnbkiKont] ;
      Maiztasunak[LetraLaguntzaile] := Maiztasunak[LetraLaguntzaile] + 1 ;
   END ;

   { Maiztasunak gordetzen dituen arrayaren pantailaraketa }
   WriteLn ('Maiztasunak deituriko arrayaren edukia: ') ;
   FOR KarakKont:='A' TO 'Z' DO
   BEGIN
      Write ('    Maiztasunak[', KarakKont, ']=', Maiztasunak[KarakKont]) ;
   END ;

   { Array batean maximoa zein den zehazten }
   Maizena := ' ' ;
   Kopurua := 0 ;
   FOR KarakKont:='A' TO 'Z' DO
   BEGIN
      IF Maiztasunak[KarakKont] > Kopurua THEN
      BEGIN
         Maizena := KarakKont ;
         Kopurua := Maiztasunak[KarakKont] ;
      END ;
   END ;
   WriteLn ;
   WriteLn ('Maizen azaltzen den letra ', Maizena, ' da ', Kopurua, ' agerpenekin') ;
END.
```

`MajuskulenMaiztasunak` programa garrantzitsua da eta xehetasunez azalduko dugu, esplikazioak ez nahastearren `MajuskulenMaiztasunak` programa bost zatitan banatuko dugu, lehentsuago ikusi dugun `Maiztasunak` arrayaren definizioaz gain ondoko bost atal hauek:

1. Teklatuz irakurritako esaldia majuskuletan jarri.
2. `Maiztasunak` arrayaren hasieraketa.
3. `Maiztasunak` arrayaren balio berrien kalkulua.

4. `Maiztasunak` arrayaren balioen pantailaraketa.
5. Array baten elementuen artean maximoa zein den zehaztea.

1

Teklatuz irakurritako esaldia majuskuletan jarri. `Esaldia` string aldagaia teklatuz irakurtzean ez da bermatzen karaktere guztiak majuskuletan etortzea, horregatik jarraian ematen diren lerroen bitartez kate baten karaktereak majuskuletara bihurtzen dira:

```
FOR ZnbkiKont:=1 TO Length(Esaldia) DO
BEGIN
   Esaldia[ZnbkiKont] := UpCase(Esaldia[ZnbkiKont]) ;
END ;
```

Kontutan izan, nahiz eta katea karaktereka landu, kasu honetan `Esaldia` karaktere-katearen zero posizioa ez da zertan berritu behar, luzera logikoa aldatzen ez delako sarrerako kateatik majuskuletako kateara.

2

`Maiztasunak` arrayaren hasieraketa. Aldagai egituratuen erreserbak egiten direnean hasieraketa automatiko bat jartzen zaie, arrayen kasuan bere elementuen datu-mota araberako hasieraketa egiten zaie (zenbaki osoen arrayak `0`-z hasieratzen dira, zenbaki errealen arrayak `0.0`-z, karaktereen eta stringen arrayak `' '` bitartez hasieratzen dira, ...).

`Maiztasunak` arrayak, ariketa honen hasieran zero balio behar duenez eta berezko hasieraketa `0`-z egiten denez, ondoko lerroak ez lirateke derrigorrezkoak izango:

```
FOR KarakKont:='A' TO 'Z' DO
BEGIN
   Maiztasunak[KarakKont] := 0 ;
END ;
```

`Maiztasunak` arrayaren indizea karaktere bat denez `FOR-DO` aginduaren kontrol-kontagailua karaktere bat izango da, `KarakKont` alegia.

3

`Maiztasunak` arrayaren balio berrien kalkulua. Prozesu honetan `Esaldia` karaktere-katea goitik behera aztertuko dugu `'A'` eta `'Z'` bitarteko letra bat aurkitzean dagokion `Maiztasunak` arrayaren posizioa inkrementatuz:

```
FOR ZnbkiKont:=1 TO Length(Esaldia) DO
BEGIN
   LetraLaguntzaile := Esaldia[ZnbkiKont] ;
   Maiztasunak[LetraLaguntzaile] :=
                          Maiztasunak[LetraLaguntzaile] + 1 ;
END ;
```

`LetraLaguntzaile` aldagaia irakurgarritasuna zaintzeko jarri dugu, zeren soberan egon daiteke `FOR-DO` aginduaren barruan esleipen hau jartzen badugu:

```
Maiztasunak[Esaldia[ZnbkiKont]] :=
                          Maiztasunak[Esaldia[ZnbkiKont]] + 1 ;
```

Izan ere `ZnbkiKont` zenbaki oso bat da, baina `Esaldia[ZnbkiKont]` karaktere bat izanen da eta letra hori `Maiztasunak` arrayaren indizea izan daiteke, ondorioz `Maiztasunak[Esaldia[ZnbkiKont]]` zenbaki oso bat izango da zeinek `+` eragiketa onartzen duen.

4 `Maiztasunak` arrayaren balioak pantailatik erakutsi. Zeregin hau funtsean hasieraketa bezalakoa da (array baten hasierako muturretik abiatu eta bere bukaeraraino mugituz elementu bakoitzari tratamendu bera aplikatzea). Kasu honetan arrayaren osagaiek pairatzen duten prozesaketa honelako zerbait da: `Write (Maiztasunak[KarakKont])`

5 `Maiztasunak` arrayaren elementuen artean maximoa finkatu. Laugarren puntuan bezala `Maiztasunak` arrayaren hasierako muturretik hasi eta bere bukaeraraino elementu bakoitzari tratamendu bera aplikatuko zaio, kasu honetan `Maizena` (dagoenekoz[5] sarrien agertzen den karakterea) eta `Kopurua` (dagoenekoz[6] sarrien azaltzen den karaktereen agerpenak) aldagaien hasieraketak ezinbestekoak dira:

```
Maizena := ' ' ;
Kopurua := 0 ;
FOR KarakKont:='A' TO 'Z' DO
BEGIN
   IF Maiztasunak[KarakKont] > Kopurua THEN
   BEGIN
      Maizena := KarakKont ;
      Kopurua := Maiztasunak[KarakKont] ;
   END ;
END ;
```

`Maiztasunak` arrayaren elementuen artean maximoa finkatzeko balio duen algoritmo honek ahultasun nabarmen bat du, zer gertatzen da balio maximoa letra batean baino gehiagotan ematen bada?. Hona hemen zer emaitza eskaintzen duen `MajuskulenMaiztasunak` programak halako kasuetan:

```
Esaldi bat eman------->Kaixo Eneritz, zer moduz zaude?
Esaldia majuskulaz---->KAIXO ENERITZ, ZER MODUZ ZAUDE?
Maiztasunak deituriko arrayaren edukia:
    Maiztasunak[A]=2    Maiztasunak[B]=0    Maiztasunak[C]=0    Maiztasunak[D]=2
    Maiztasunak[E]=4    Maiztasunak[F]=0    Maiztasunak[G]=0    Maiztasunak[H]=0
    Maiztasunak[I]=2    Maiztasunak[J]=0    Maiztasunak[K]=1    Maiztasunak[L]=0
    Maiztasunak[M]=1    Maiztasunak[N]=1    Maiztasunak[O]=2    Maiztasunak[P]=0
    Maiztasunak[Q]=0    Maiztasunak[R]=2    Maiztasunak[S]=0    Maiztasunak[T]=1
    Maiztasunak[U]=2    Maiztasunak[V]=0    Maiztasunak[W]=0    Maiztasunak[X]=1
    Maiztasunak[Y]=0    Maiztasunak[Z]=4
Maizen azaltzen den letra E da 4 agerpenekin
_
```

Pantailaraketan ikus daitekeenez agerpen maximoa `4`-koa da, zein `E` eta `Z` bi karaktereetan ematen den. Baina `FOR-DO` barneko `IF-THEN` aginduan erabiltzen den konparatzailea `>` denez, azken iterazio errepikakorrean `Z`-ren maiztasunak (`4` delako) ez du `Kopurua` (maximo partziala `4` da ere) gainditzen. Ondorioz, maximoa karaktere batean baino gehiagotan ematean lehenengoa hartuko da soluziotzat `IF-THEN` aginduaren barnean konparatzailea `>` delako, baina `IF-THEN` barnean konparatzailea `>=` balitz soluzioa azken karaktere maximoa litzateke.

Maizen agertzen den letra bakar bat pantailaratu beharrean, agerpen maximoak dituzten guztiak bilatu nahi izango bagenu arazoa zaildu egiten da, izan ere `Maizena` aldagaia `Char` datu-motakoa izan ordez karaktereak bil ditzakeen arraya izan beharko litzateke.

[5] Dagoenekoz horrekin esan nahi duguna zera da, `Maiztasunak` arraya aztertzen ari garen uneraino zein den maiztasun handien duen karakterea `Maizena` aldagaian gordeta daukagu.

[6] Dagoenekoz horrekin esan nahi duguna zera da, `Maiztasunak` arraya aztertzen ari garen uneraino zein den balio maximoa.

10.1.5 Arrayen luzera fisikoa eta luzera logikoa

Luzera fisiko eta luzera logiko kontzeptuak karaktere-kateekin landu genituen, ikasgai hura gogoratuz luzera fisikoa katearen luzera potentziala zen eta aldaezina zitzaigun, luzera logikoak berriz karaktere-katearen luzera efektiboa izanik programaren exekuzioarekin balio ezberdinak har ditzakeela esan genuen.

String edo karaktere-kate datu-mota definitzean luzera fisikoa zehazten da, eta string aldagai bat memorian sortzean berak biltzen ditu bere baitan integraturik luzera fisikoaren eta logikoaren kontzeptu biak. Luzera fisikoaren muga datu-motaren definiziotik datorkio, eta, luzera logiko edo efektiboarena string aldagaiaren zero posizioak markatzen du.

Oraintxe ikusiko dugunez luzera fisikoa eta logikoa arrayetan ere ematen dira, baina biak array aldagaian integraturik ez dira etorriko.

10.1.5.1 Arrayen luzera fisikoa

Array batek gehienez onar dezakeen elementuen kopurua bere definizioan finkatzen da, eta horixe izango da arrayaren luzera fisikoa. Hots, zenbat elementu gorde daitekeen arrayan bere luzera fisikoa da, **10.1.2 Indizeak** izenburuko puntua gogoratuz, arrayaren indizearen mugak zehaztean array horrek izan dezakeen elementu kopuru maximoa honako formulaz[7] finkatu egiten da: Ord(goimuga) - Ord(behemuga) + 1.

Baina gerta daiteke array datu-motaren definizioa oso osorik ez ezagutzea eta luzera fisikoa behar izatea, halako kasu batean `SizeOf()` funtzio estandarra lagungarria izan daiteke guretzat. Ikus jarraian erakusten den `ArrayBatenLuzeraFisikoa` programa:

```
PROGRAM ArrayBatenLuzeraFisikoa ;                          { \TP70\10\ARRAY_7.PAS }
CONST
  BEHEMUGA = 'A' ;
  GOIMUGA = 'Z' ;
TYPE
  DM_Soldatak = ARRAY[BEHEMUGA..GOIMUGA] OF Real ;
VAR
  Lantegi : DM_Soldatak ;
BEGIN
   WriteLn ('Lantegi aldagaiaren eta DM_Soldatak datu-motaren luzera fisikoa:') ;
   WriteLn ('Lantegi------->', Ord(GOIMUGA)-Ord(BEHEMUGA)+1) ;
   WriteLn ('Lantegi------->', SizeOf(Lantegi) DIV SizeOf(Real)) ;
   WriteLn ('DM_Soldatak--->', Ord(GOIMUGA)-Ord(BEHEMUGA)+1) ;
   WriteLn ('DM_Soldatak--->', SizeOf(DM_Soldatak) DIV SizeOf(Real)) ;
END.
```

Arrayen propietateen artean egitura homogeneoak direla esan dugu, hots bere osagai guztiak datu-mota bakar bat dute oinarritzat, `SizeOf()` funtzioaren erabilpenak arrayen ezaugarri hori aintzat hartzen du eta onartzen duen parametroa aldagai edo datu-mota denez deia birritan egin dezakegu. Hauxe litzateke `ArrayBatenLuzeraFisikoa` programari dagokion irteera:

```
Lantegi aldagaiaren eta DM_Soldatak datu-motaren luzera fisikoa:
Lantegi------->26
Lantegi------->26
DM_Soldatak--->26
DM_Soldatak--->26
_
```

[7] Non `Ord()` funtzio estandarrak parametroaren ordinala ematen duen.

10.1.5.2 Arrayen luzera logikoa

Esan dugunez luzera fisikoa arrayen definizioan emaniko datuen menpekoa da, eta hura ezagutzeko goimuga eta behemuga darabilen formulaz balia gaitezke, edo bestela `SizeOf()` funtzio estandarrean oinarri gaitezke.

Baina programaren une jakin batean arrayak ez du zertan erabat beterik egon beharko, esate baterako demagun ikasleen notak gordetzeko zenbaki errealen array bat nahi dugula, eta, ikasleen kopuru maximoa 80 izan daitekeela aldez aurretik ezaguna dela, hau litzateke `DM_Zerrenda` datu-mota egituratuari dagokion erazagupena eta `Gela1` arraya:

```
TYPE
   DM_Zerrenda = ARRAY[1..80] OF Real ;
VAR
   Gela1 : DM_Zerrenda ;
```

Demagun ikasle talde baten notak `Gela1` arrayan gordeko ditugula, ikus jarraian ematen dugun `ArrayBatenLuzeraLogiko1` programa:

```
PROGRAM ArrayBatenLuzeraLogiko1 ;                    { \TP70\10\ARRAY_8.PAS }
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 80 ;
TYPE
  DM_Zerrenda = ARRAY[BEHEMUGA..GOIMUGA] OF Real ;
VAR
  Gela1 : DM_Zerrenda ;
  Kont, Luzera : Integer ;
  Erantz : Char ;
BEGIN
   Erantz := 'B' ;
   Kont := 0 ;
   WHILE (Kont < GOIMUGA) AND (Erantz = 'B') DO
   BEGIN
      Kont := Kont + 1 ;
      Write (Kont, '. nota eman: ') ;
      ReadLn ( Gela1[Kont] ) ;
      REPEAT
         Write ('Ikasle gehiagorik? (B/E): ') ;
         ReadLn (Erantz) ;
         Erantz := UpCase (Erantz) ;
      UNTIL (Erantz = 'B') OR (Erantz = 'E') ;
   END ;

   Luzera := Kont ;         { luzera logikoa zehaztu }

   WriteLn ('Gela1 arrayaren luzera logikoa ', Luzera, ' da') ;
   WriteLn ('Hau da Gela1 arrayaren eduki efektiboa:') ;
   FOR Kont:=BEHEMUGA TO Luzera DO
      Write ( Gela1[Kont] :7:2 ) ;

   WriteLn ;
END.
```

Uler daitekeenez `ArrayBatenLuzeraLogiko1` programa `WHILE-DO` aginduaz egin ordez `REPEAT-UNTIL` bitartez egin daiteke (honi `ArrayBatenLuzeraLogiko2` izena jarri diogu), `REPEAT-UNTIL` aginduan oinarritutako `ArrayBatenLuzeraLogiko2` programaren bertsioa idaztea irakurlearen lana litzateke. Edozein kasutan ere, `WHILE-DO` edo `REPEAT-UNTIL` aginduak ikasle bakoitzaren nota gordetzeko balio du, prozesu errepikakor hori eteteko baldintza bikoitza jarri da eta horren arrazoia honako hau da.

Batetik, gerta daiteke, eta egoerarik arruntena izan beharko litzateke aurrikusi den ikasleen kopuru maximoa 80-koa bada, `WHILE-DO` bigizta amaitu nahi izatea nota gehiagorik

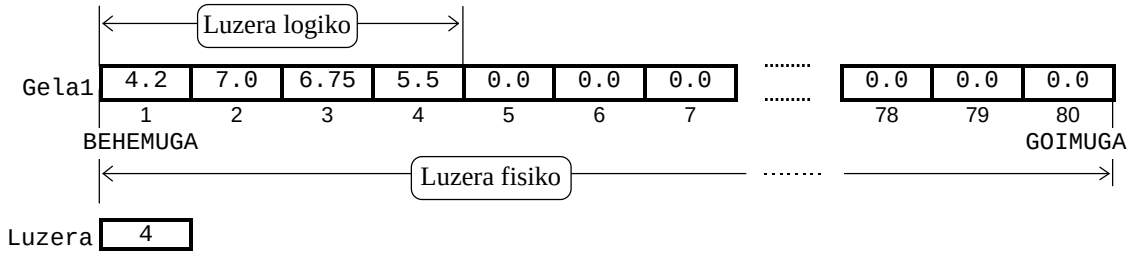
ez dugunean, horretarako da hain zuzen ere baldintzaren `Erantz='B'` kondizioa. Halakoetan `Kont` ez da `GOIMUGA` baino handiagoa, eta `Luzera`[8] izeneko aldagaiaren balioa `Kont` konta-gailuarena izango da.

Bestetik, gerta daiteke, ikasleen kopurua 80-koa baino handiagoa izatea, eta horrekin batera arraya betetzen ari den programaren operadoreak ez jakitea bere muga fisikoa 80 elementukoa dela, horregatik programa sendo batek saihestu beharko du arrayaren mugetatik kanpo balioak gorde ahal izatea. Gure adibidean, `WHILE-DO` aginduaren baldintza bikoitzaren `Kont < GOIMUGA` kondizioak lan hori betetzen du, laurogeigarren nota sartu egin denean eta hurrengoren bat sartu nahi ote den galdetzean `'B'` erantzutean ez da baldintza bikoitzaren lehen kondizio hori betetzen `80 < 80` gezurra baita. Ondorioz, `Kont` aldagaiak 80 balio duenean `'B'` erantzun arren `WHILE-DO` aginduaren errepikapena eteten da. Ikusten denez `Kont` aldagaiak bebetako luzera logikoa zehazten du uneoro, hau guztiagatik `Luzera:=Kont` esleipenak bermatzen du adibide honetan `Gela1` arrayaren luzera logikoa.

Hau izan daiteke `ArrayBatenLuzeraLogiko1` programaren exekuzio-adibide bat, non laurogei zenbaki erreal gordetzeko ahalmena duen array batean lehen lau posizioak betetzen diren:

```
1. nota eman: 4.2
Ikasle gehiagorik? (B/E): b
2. nota eman: 7
Ikasle gehiagorik? (B/E): b
3. nota eman: 6.75
Ikasle gehiagorik? (B/E): b
4. nota eman: 5.5
Ikasle gehiagorik? (B/E): e
Gela1 arrayaren luzera logikoa 4 da
Hau da Gela1 arrayaren eduki efektiboa:
   4.20   7.00   6.75   5.5
_
```

Eta exekuzio-adibide horri dagokion `Gela1` arrayaren eskema:



Kontura gaitezen, string[9] datu-motaren ez bezala, une jakin batean array batek duen luzera logikoa finkatzeko beste aldagai bat beharko dela (`ArrayBatenLuzeraLogiko1` izeneko programan `Integer` datu-motatako `Luzera` aldagaia). Beraz, programa askotan array lineal bat bi aldagaien bitartez zehaztuko da: array batez eta indizearen[10] datu-motatako beste aldagai batez. Arraya lineala izan gabe taula bat bada, hura zehazteko aldagaiak hiru izango dira: bi dimentsioko array bat eta taularen alde baliagarria mugatzen duten indizeen datu-motatako beste aldagai pare bat.

[8] Argi dago `Luzera` aldagaiaren interpretazioa `Gela1` arrayari dagokion luzera logikoa, edo luzera efektiboa, dela.

[9] Gogoratu strig aldagai batek zero posizioan luzera logikoa gordetzen duenez, string aldagai berak adieraz dezakeela une bakoitzean edukiaren zein zati den baligarria.

[10] Adibide honetan luzera logikoa zenbakizko bat da, baina aurreko adibidearen `Maiztasunak` array aldagaiaren luzera efektiboa karaktere aldagai batez eman beharko genuke.

10.1.5.3 {$R±} direktiba

Azpieremuak tratatzean (**8.4.1** puntuan) eta konpiladorearen direktibak deskribatzean (**8.7.1.1** puntuan) aipatu izan dugu direktiba hau. Heina edo barrutiaren direktiba indarturik dagoenean (`{$R+}` denean) arrayen, karaktere-kateen eta azpieremuen indizeak konprobatu egiten dira. Gogora ditzagun hemen bere ezaugarriak:

Sintaxia:	`{$R+}` edo `{$R-}`
Balio lehenetsia:	`{$R-}`
Menu komandoa:	`Options` | `Compiler` | `Range checking`

Jarraian array pare baten indexazioa kontrolatzen duen programa erakusten da, non arrayen indizea den `Kont` aldagaia goimugatik gora dagoen:

```
PROGRAM R_Direktiba ;                          { \TP70\10\ARRAY_10.PAS }
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 10 ;
TYPE
  DM_Zerrenda = Array [BEHEMUGA..GOIMUGA] OF Integer ;
VAR
  Zenbakiak1, Zenbakiak2 : DM_Zerrenda ;
  Kont : Byte ;
BEGIN
   WriteLn ('Programa hasieran Zenbakiak2[1]=', Zenbakiak2[1]) ;
 { Hasieran R direktiba indargabetua. Balio lehenetsia (-) delako, }
 { errorerik ez da detektatzen nahiz eta Kont goimugatik gora izan }
   Kont := GOIMUGA + 1 ;
   Write (Kont, ' posizioan gordeko den zenbakia eman: ') ;
   ReadLn ( Zenbakiak1[Kont] ) ;
   WriteLn ('Zenbakiak1[', Kont, ']=', Zenbakiak1[Kont]) ;
   WriteLn ('Zenbakiak2[1]=', Zenbakiak2[1]) ;

 {$R+}
   Kont := GOIMUGA + 1 ;
   Zenbakiak1[Kont] := 99 ;       { errorea exekuzio denboran }

END.
```

`R_Direktiba` programan `Zenbakiak1` arrayaren hamaikagarren elementua teklatuz ematea eskatzen da, baina kontura gaitezen bere goimuga hamar delako array horrek hamaikagarren elementurik ez duela. Heina kontrolatzen duen `R` direktiba indargabeturik dagoenez zenbakia eman eta pantailara daiteke, baina non gordetzen da teklatuz emaniko kopurua?; zergatik?. Ondoren `R` direktiba indarrean jartzen delarik ez da 1..10 esparrutik kanpo dagoen indizerik onartzen errorea gertatuz.

Galdera horien erantzunak `R_Direktiba` programaren irteera aztertuz asma daitezke:

```
Programa hasieran Zenbakiak2[1]=0
11 posizioan gordeko den zenbakia eman: -123
Zenbakiak1[11]=-123
Zenbakiak2[1]=-123
Runtime error 201 at 59F7:018C.
_
```

Aurrerago ikusiko den puntu batean arrayen elementuen memori posizioak erakutsiko dira eta orduan hobeto ulertuko litzateke `R_Direktiba` programaren irteera, baina oso intuitiboa denez hona hemen bere esplikazioa eta aurreko bi galderen erantzunak. Esparrutik kanpo dagoen array elementu bat irakurtzean balioa memoriaren beste aldagai bati jartzen zaio, adibide honetan `Zenbakiak1` eta `Zenbakiak2` aldagaiak orden horretan erazagutu direnez lehenengoaren “hamaikagarren” elementua bigarrenaren lehena da.

10.1.6 Arrayak 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[11] 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.

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[12] 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:

[11] Memori posizioen helbideak.

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


```
PROGRAM ArrayBatParametroBezala ;                { \TP70\10\ARRAY_11.PAS }
USES
    Crt, Ikus ;      (* IKUS.TPU  unitatea darabil *)
TYPE
    DM_Zerrenda = ARRAY[1..10] OF Integer ;

PROCEDURE Azpirrutina (Talde1 : DM_Zerrenda;
                       VAR Talde2 : DM_Zerrenda;
                       CONST Talde3 : DM_Zerrenda) ;
BEGIN
   WriteLn ; WriteLn ;
   WriteLn ('ALDAGAIA':29, '       HELBIDEA', '   TAMAINA') ;
   WriteLn ('-----------------------------------------------':60) ;

   WriteLn ('Talde3 ---(CONST)---->   ':36, IntToHex (Seg (Talde3)),':',
            IntToHex (Ofs (Talde3)), '    (', SizeOf (Talde3), ')') ;
   WriteLn ('Talde2 ---(VAR)------>   ':36, IntToHex (Seg (Talde2)),':',
            IntToHex (Ofs (Talde2)), '    (', SizeOf (Talde2), ')') ;
   WriteLn ('Talde1 ---(balioz)--->   ':36, IntToHex (Seg (Talde1)),':',
            IntToHex (Ofs (Talde1)), '    (', SizeOf (Talde1), ')') ;
END ;


VAR
   TaldeOso1, TaldeOso2, TaldeOso3 : DM_Zerrenda ;

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 ('TaldeOso3 --->   ':36, IntToHex (Seg (TaldeOso3)),':',
            IntToHex (Ofs (TaldeOso3)), '    (', SizeOf (TaldeOso3), ')') ;
   WriteLn ('TaldeOso2 --->   ':36, IntToHex (Seg (TaldeOso2)),':',
            IntToHex (Ofs (TaldeOso2)), '    (', SizeOf (TaldeOso2), ')') ;
   WriteLn ('TaldeOso1 --->   ':36, IntToHex (Seg (TaldeOso1)),':',
            IntToHex (Ofs (TaldeOso1)), '    (', SizeOf (TaldeOso1), ')') ;

   Azpirrutina (TaldeOso1, TaldeOso2, TaldeOso3) ;

END .
```

`ArrayBatParametroBezala` programan hamar oso gordetzeko ahalmena arrayak erazagutzen dira, programa nagusiko hiru aldagaiak (`TaldeOso1`, `TaldeOso2` eta `TaldeOso3`) datuen segmentuan sortzen dira eta `Azpirrutina()` prozedurara bidaltzean bakoitzak bere pasatze modua du:

- `TaldeOso1` aldagaia baliozko parametroa da eta `Talde1` dagokio
- `TaldeOso2` arraya aldagai-parametro bat da eta `Talde2` dagokio
- `TaldeOso3` konstante-parametroari `Talde3` dagokio azpirrutinan

`ArrayBatParametroBezala` programa egikaritu eta bere irteera azter dezagun, ikus daitekeenez, programa exekutatu den une eta ordenadore horretan pilaren segmentua `5A5A` helbidean hasten da, datuak `5A29` helbidetik aurrera kokatzen dira eta programaren lehen sententzia `5940` helbidean dago:

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

PILARI dagokion segmentuaren hasiera:  5A5A
DATUEI dagokien segmentuaren hasiera:  5A29
KODEARI dagokion segmentuaren hasiera: 5940


                    ALDAGAIA       HELBIDEA   TAMAINA
                ------------------------------------------
                   TaldeOso3 --->   5A29:0086    (20)
                   TaldeOso2 --->   5A29:0072    (20)
                   TaldeOso1 --->   5A29:005E    (20)


                    ALDAGAIA       HELBIDEA   TAMAINA
            ------------------------------------------------
             Talde3 ---(CONST)---->   5A29:0086    (20)
             Talde2 ---(VAR)------>   5A29:0072    (20)
             Talde1 ---(balioz)--->   5A5A:3DDA    (20)
_
```

`TaldeOso1`, `TaldeOso2` eta `TaldeOso3` arrayak programa nagusiko aldagaiak direlako datuen segmentuan sortzen dira. Bestalde `Talde2` eta `Talde3` aldagaiak erreferentziak dira eta ez dute pilan tokirik hartzen, baina `Talde1` arraya berriz pilan aurkitzen den `TaldeOso1` 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[13] (nahiz eta adibideko arrayak oso handiak ez izan), hori dela eta datu-mota egituratu bat azpiprograma batera bidaltzean erreferentziaz pasatuko da.

Array 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 bat ematea aski da, hots, array baten portaera azpirrutinaren barnean datuak eskaintzearena denean konstante-parametro bat izango da (parametro erreala aldatua izatea ez baitzaigu interesatzen), baina array batek balioak azpiprograman barnean hartzen dituenean modulu deitzaileak horren berri jaso behar duenez arraya aldagai-parametroa izango da.

Esandakoaren aplikazioa `ArrayBiBerdinakOteDiren` adibide-programan ikusten da, programa hau lehenago ikusi dugun `ArrayBiBerdinak` programaren aldaera bat da, non bi azpiprogramak, `ArrayBatBete()` prozedura eta `ArrayakKonparatu()` funtzioa, erabiltzen diren:

```
PROGRAM ArrayBiBerdinakOteDiren ;                          { \TP70\10\ARRAY_12.PAS }
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 40 ;
TYPE
  DM_Zerrenda = ARRAY[BEHEMUGA..GOIMUGA] OF Real ;

PROCEDURE ArrayBatBete (VAR Gela : DM_Zerrenda; VAR Luzera : Integer) ;
VAR
  Kont : Integer ;
  Erantz : Char ;
BEGIN
   Erantz := 'B' ;
   Kont := 1 ;
   WHILE (Kont <= GOIMUGA) AND (Erantz = 'B') DO
   BEGIN
      Write (Kont, '. nota eman: ') ;
      ReadLn ( Gela[Kont] ) ;
```

[13] Kopia egiteko denbora eta memoria behar direlako.

```
      REPEAT
         Write ('Ikasle gehiagorik? (B/E): ') ;
         ReadLn (Erantz) ;
         Erantz := UpCase (Erantz) ;
      UNTIL (Erantz = 'B') OR (Erantz = 'E') ;

      IF Erantz = 'B' THEN
        Kont := Kont + 1 ;
   END ;    { baldintza bikoitza duen WHILE-DO bigiztaren amaiera }

   IF Kont > GOIMUGA THEN
     Luzera := GOIMUGA
   ELSE
     Luzera := Kont ;
END;

FUNCTION ArrayakKonparatu (CONST Gela1 : DM_Zerrenda; Luzera1 : Integer;
                           CONST Gela2 : DM_Zerrenda; Luzera2 : Integer) : Boolean ;
VAR
   BerdinakBai : Boolean ;
   Kont : Integer ;
BEGIN
   IF Luzera1 <> Luzera2 THEN
       ArrayakKonparatu := FALSE
   ELSE
   BEGIN
      BerdinakBai := TRUE ;
      Kont := BEHEMUGA ;
      WHILE (Kont <= Luzera1) AND BerdinakBai DO
      BEGIN
         IF Gela1[Kont] <> Gela2[Kont] THEN
               BerdinakBai := FALSE ;
         Kont := Kont + 1 ;
      END ;
      ArrayakKonparatu := BerdinakBai ;
   END ;
END;

VAR                                                  { aldagai orokorrak }
   Gela1, Gela2 : DM_Zerrenda ;
   Luzera1, Luzera2 : Integer ;
BEGIN                                                { Programa Nagusia }
   WriteLn ('Gela1 arraya betetzen: ') ;
   ArrayBatBete (Gela1, Luzera1) ;

   WriteLn ('Gela2 arraya betetzen: ') ;
   ArrayBatBete (Gela2, Luzera2) ;

   { Arrayen arteko konparaketa burutu, hau da Gela1 arrayaren
     luzera logikoa eta edukia Gela2-rena berdinak ote diren }

   IF ArrayakKonparatu(Gela1, Luzera1, Gela2, Luzera2) THEN
     WriteLn ('Gela1 eta Gela2 arrayak berdinak dira')
   ELSE
     WriteLn ('Gela1 eta Gela2 arrayak desberdinak dira') ;
END.
```

Esandakoarekin bat etorriz `ArrayBatBete()` prozeduraren eta `ArrayakKonparatu()` funtzioaren parametroek dituzten pasatze moduak hauek dira:

`ArrayBatBete()`

`Gela` prozedura honen bitartez `Gela` arrayak balioak hartzen ditu teklatuz zenbaki errealak irakurtzen baitira, teklatuz emandako kopuru horiek programa nagusiko `Gela1` eta `Gela2` arrayek har ditzaten parametroaren pasatze modua aldagai-parametroarena izango da.

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

`CONST` bitartez markaturiko konstante-parametroa ezin daiteke errutina barruan aldatu, horregatik `ReadLn(Gela[Kont])` sententzia ezin izango litzateke konpilatu nahiko ezaguna den errore hau agertzen zaigulako: `Error 122: Invalid variable reference.`

`Luzera` parametro honek ez du inolako zailtasunik suposatzen, aldagaia bakuna (ez-egituratua) denez baliozko parametroa edo aldagai-parametroa izango da. `Luzera` zenbakia, portaeraz, irteerakoa denez erreferentziaz pasatuko da bere aurretik `VAR` marka jartzen delarik.

`ArrayakKonparatu()`

`Gela1` bi arrayen arteko konparaketa burutzen duen `ArrayakKonparatu()` funtzioaren lehen parametroa egituratua da, eta portaeraz sarrerakoa da. Egituratua delako `Gela1` arraya 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 `Gela1` parametroari aurretik `VAR` jarriko balitzaio aldagai-parametroa bihurtuz?.

Ezer ere ez, `ArrayakKonparatu()` funtzioaren zeregina egokitasun eta zuzentasun osoz beteko litzateke. Baina kontuz, arrisku bat egon badago `Gela1` parametro hori `VAR` bitartez markatzean, aldagai-parametroa izango litzatekeenez errutina barruan aldatzea zilegi litzaiguke eta nahi gabe modulu nagusiaren parametro erreala errutina barnean alda zitekeen.

`Luzera1` parametro hau kopuru oso bat delako, ez-egituratua delako, baliozko parametroa edo aldagai-parametroa izango da. `Luzera1` zenbakia, portaeraz, sarrerakoa denez ez da erreferentziaz pasatuko balioz baizik, horregatik ez da `Luzera1` aurrean inolako markarik jartzen.

`Gela2` azaldu dugun `Gela1` parametro bezala.

`Luzera2` parametroa eta azaldu dugun `Luzera1` parametroa berdintsuak direnez, lehenengoari buruz esandakoak balio du hemen ere.

10.2 ARRAY DIMENTSIOBAKARRAK

Orain arte ikusi ditugun array guztiak linealak ziren, hau da bere dimentsioa adierazteko indize bakar bat behar da. Array dimentsiobakar diren egitura linealei *zerrenda* edo *bektore* esaten zaie.

Ikus ditzagun, hurrengo bi puntuetan, array lineal baten biltegitzea memorian eta egitura horri dagokion hasieraketa mota bat. Array lineal bat memorian nola kokatzen den **10.2.1 Array dimentsiobakar baten biltegitzea memorian** puntuan garatuko dugu, eta array aldagai bati dagokion hasieraketa **10.2.2 Array dimentsiobakarra den aldagai baten hasieraketa** izeneko puntuan.

10.2.1 Array dimentsiobakar baten biltegitzea memorian

Array datu-motaren elementuak elkar ondoan kokatzen dira memorian. Honezkero array dimentsiobakar batek egitura lineal bat osatzen du, eta elementu bakoitzari dagokion helbidea aurrekoaren jarraian datorrena izango dela froga daiteke. Ikus bi array definitzen dituen `ArrayDimentsiobakarra` programa:

```
PROGRAM ArrayDimentsiobakarra ;                  { \TP70\10\ARRAY_13.PAS }
USES
    Crt, Ikus ;      (* IKUS.TPU  unitatea darabil *)
CONST
    BEHEMUGA = 1 ;
    GOIMUGA = 5 ;
TYPE
    DM_Zerrenda1 = ARRAY[BEHEMUGA..GOIMUGA] OF Char ;
    DM_Zerrenda2 = ARRAY[BEHEMUGA..GOIMUGA] OF Integer ;
VAR
   TaldeKar : DM_Zerrenda1 ;
   TaldeOso : DM_Zerrenda2 ;
   i : Byte ;
BEGIN              (* Programa Nagusia *)
   ClrScr ;
   WriteLn ('ARRAY DIMENTSIOBAKAR BATEN ELEMENTUEN 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 ('TaldeOso ------> ':36, IntToHex (Seg (TaldeOso)),':',
            IntToHex (Ofs (TaldeOso)), '    (', SizeOf (TaldeOso), ')') ;
   WriteLn ('TaldeKar ------> ':36, IntToHex (Seg (TaldeKar)),':',
            IntToHex (Ofs (TaldeKar)), '    (', SizeOf (TaldeKar), ')') ;
   WriteLn ;
   WriteLn ;
   WriteLn ;
   FOR i:=GOIMUGA DOWNTO BEHEMUGA DO
   BEGIN
      Write ('TaldeOso[':28, i, '] ---> ', IntToHex (Seg (TaldeOso[i])) ) ;
      Write (':', IntToHex (Ofs (TaldeOso[i])) ) ;
      WriteLn ( '    (', SizeOf (TaldeOso[i]), ')') ;
   END ;
   WriteLn ;
   FOR i:=GOIMUGA DOWNTO BEHEMUGA DO
   BEGIN
      Write ('TaldeKar[':28, i, '] ---> ', IntToHex (Seg (TaldeKar[i])) ) ;
      Write (':', IntToHex (Ofs (TaldeKar[i])) ) ;
      WriteLn ( '    (', SizeOf (TaldeKar[i]), ')') ;
   END ;
END .
```

`ArrayDimentsiobakarra` programa egikaritu eta bere irteera aztertzean array biak `5A0C` helbidea duen segmentuan aurkituko direla ikusten da. `TaldeKar` arraya `5A0C:005E` helbidean hasten da (`TaldeKar` arrayaren lehen elementua `5A0C:005E` helbide horretan hasten da), eta gainerako beste elementuak jarraian aurkitzen dira (`Char` direnez byte bateko diferentzia dago helbideetan).

`TaldeKar` arrayari buruzkoa esan duguna `TaldeOso` aldagaiarako errepika daiteke, hots, hasten den `5A0C:0064` helbidetik aurrera hurrengo hamar zortzikoteak `TaldeOso` arrayaren bost elementuei dagozkiela (elementuok `Integer` direnez gero euren helbideetan bi byteko diferentzia dago).

```
ARRAY DIMENTSIOBAKAR BATEN ELEMENTUEN MEMORI HELBIDEAK
======================================================

PILARI dagokion segmentuaren hasiera:  5A3B
DATUEI dagokien segmentuaren hasiera:  5A0C
KODEARI dagokion segmentuaren hasiera: 593C


                    ALDAGAIA       HELBIDEA   TAMAINA
               ---------------------------------------------
                    TaldeOso ------> 5A0C:0064    (10)
                    TaldeKar ------> 5A0C:005E    (5)



                    TaldeOso[5] ---> 5A0C:006C    (2)
                    TaldeOso[4] ---> 5A0C:006A    (2)
                    TaldeOso[3] ---> 5A0C:0068    (2)
                    TaldeOso[2] ---> 5A0C:0066    (2)
                    TaldeOso[1] ---> 5A0C:0064    (2)

                    TaldeKar[5] ---> 5A0C:0062    (1)
                    TaldeKar[4] ---> 5A0C:0061    (1)
                    TaldeKar[3] ---> 5A0C:0060    (1)
                    TaldeKar[2] ---> 5A0C:005F    (1)
                    TaldeKar[1] ---> 5A0C:005E    (1)
_
```

`TaldeKar` arraya `5A0C:0062` helbidean amaitzen bada, eta `TaldeOso` aldagaiaren hasierako helbidea `5A0C:0064` bada, zer gordetzen da `5A0C:0063` helbidean?. Galdera horren erantzuna asmatzeko `ArrayDimentsiobakarra` programaren `TaldeKar` eta `TaldeOso` array aldagaiak alderantzizko ordenez erazagutu[14] eta berriro exekutatu, lortzen den emaitza aurrekoarekin konparatzean memori hutsune bat dela kontura gaitezke.

Laburbilduz, `A` array lineal baten elementu jakin bati dagokion helbidea ezagutu ahal izateko, `n`-garren elementua adibidez, ondoko formula aplikatuko da:

Helbide (`A[n]` *) = Helbide (* `A` *) + SizeOf (* `A[n]` *)* $*$ *(* `n - BeheMuga` *)*

Non *Helbide (* `A` *)* arrayaren hasierako helbidea den eta *SizeOf (* `A[n]` *)* array horren oinarrizko elementuaren tamaina bytetan. Aurreko adibidean `TaldeOso` arrayaren hasierako helbidea `5A0C:0064` da eta bere hirugarren elementuari dagokiona honela kalkulatzen dugu:

Helbide (`TaldeOso[3]`*) = Helbide (*`TaldeOso`*) + 2* $*$ *(*`3 - BEHEMUGA`*)*

Helbide (`TaldeOso[3]`*) =* `5A0C:0064 + 2` $*$ *(*`3 - 1`*) =* `5A0C:0068`

10.2.2 Array dimentsiobakarra den aldagai baten hasieraketa

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[15].

[14] Erazagupen berria:
```
VAR
  TaldeOso : DM_Zerrenda2 ;
  TaldeKar : DM_Zerrenda1 ;
```

[15] Aldagaiek izaten duten hasieraketa sententzien atalean gertatzen da (`BEGIN` eta `END` artean).

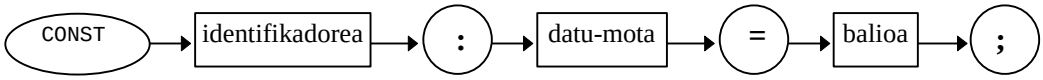
Baina Turbo Pascal lengoaian aldagaiak `CONST` bloke barruan deklara daitezke ere, guk ez dugu aukera hori erabili duen abantaila bakarra aldagai ez-egituratuekin interesgarria ez delako. Hots, `CONST` blokean deklaraturiko aldagaiari bertan egiten zaio hasieraketa, esate baterako:

```
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\10\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. Hauxek 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
_
```

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

```
PROGRAM ArrayenHasieraketa ;                      { \TP70\10\ARRAY_14.PAS }
CONST
   EGUNKOPURU = 7 ;
TYPE
   DM_Euria = ARRAY[1..EGUNKOPURU] OF Real ;
CONST
   EuriaIrunea : DM_Euria = (0.5,  2.1,  2.2,  6.8,  0.0,  0.0,  3.4) ;
VAR
   EuriaBaiona : DM_Euria ;
   Kont : Byte ;
```

```
BEGIN
   WriteLn ('EuriaIrunea aldagaia hasieratuta dago') ;
   WriteLn ('EuriaBaiona aldagaia hasieratzen ari da') ;
   FOR Kont:= 1 TO EGUNKOPURU DO
      EuriaBaiona[Kont] := Random * 100 ;

   WriteLn ('EuriaIrunea aldagaiaren balioak:') ;
   FOR Kont:= 1 TO EGUNKOPURU DO
      Write (EuriaIrunea[Kont]:10:2) ;
   WriteLn ;

   WriteLn ('EuriaBaiona aldagaiaren balioak:') ;
   FOR Kont:= 1 TO EGUNKOPURU DO
      Write (EuriaBaiona[Kont]:10:2) ;
   WriteLn ;
END.
```

`ArrayenHasieraketa` programan konstanteen deklarazioaren bloke bi daude, batean `EGUNKOPURU` benetako konstantea erazagutzen da eta bigarrenean `EuriaIrunea` aldagaia. `EuriaIrunea` array aldagaiaren hasieraketa egiteko lehenago erakutsi den sintaxia jarraitu behar da (balioak komaz banatuz eta multzoa parentesi artean jarriz). Dena den, ez ahaztu array baten hasieraketa `MajuskulenMaiztasunak` izeneko programan burutu zen bezala egin daitekeela, `FOR-DO` bigizta baten barruan alegia.

`ArrayenHasieraketa` programaren exekuzioan `EuriaBaiona` aldagaiaren elementuei `Random` funtzioaren araberako balioak ematen zaizkio, eta `ArrayenHasieraketa`-ren irteera posible bat hauxe izan daiteke:

```
EuriaIrunea aldagaia hasieratuta dago
EuriaBaiona aldagaia hasieratzen ari da
EuriaIrunea aldagaiaren balioak:
      0.50      2.10      2.20      6.80      0.00      0.00      3.40
EuriaBaiona aldagaiaren balioak:
      0.10      3.14     86.20     20.26      5.90      3.70     12.61
_
```

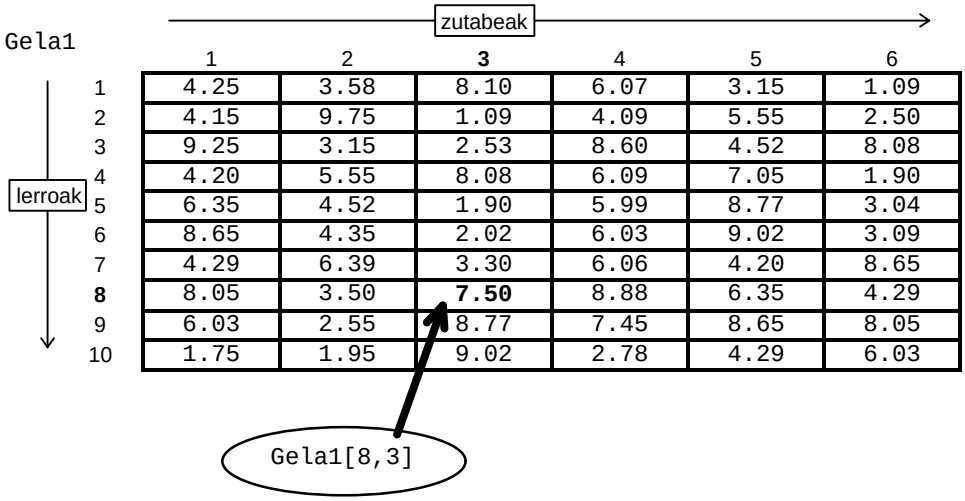
10.3 ARRAY DIMENTSIOANITZAK

Array dimentsio bakarraren kontzeptua bi edo gehiago dimentsiotara heda daiteke, bi dimentsio dituen array bati *taula* esaten zaio eta bere elementu jakin bat indize pare batez atzituko da.

Demagun array aldagaiak deklaratzeko `DM_Notak` izeneko datu-mota sortu nahi dela programa zehatz batean, suposa dezagun `DM_Notak` datu-motako arrayetan zenbaki errealak gordeko direla eta taularen tamaina 10 x 6 izango dela. Hona hemen `DM_Notak` datu-motari dagokion erazagupena:

```
TYPE
   DM_Notak = ARRAY[1..10, 1..6] OF Real ;

VAR
   Gela1, Gela2 : DM_Notak ;
```

Ikusten den bezala, arrayari dagozkion indize biren goimugak eta behemugak `..` markaren bitartez azpieremu bana mugatzen dute, eta azpieremu biren artean komatxo bat jarri egin dela. `DM_Notak` datu-motako edozein aldagai zenbaki osoen multzo bat da, eta berak duen egitura 10x6 izango da, hots, 10 lerro eta 6 zutabe:



	1	2	**3**	4	5	6
1	4.25	3.58	8.10	6.07	3.15	1.09
2	4.15	9.75	1.09	4.09	5.55	2.50
3	9.25	3.15	2.53	8.60	4.52	8.08
4	4.20	5.55	8.08	6.09	7.05	1.90
5	6.35	4.52	1.90	5.99	8.77	3.04
6	8.65	4.35	2.02	6.03	9.02	3.09
7	4.29	6.39	3.30	6.06	4.20	8.65
8	8.05	3.50	**7.50**	8.88	6.35	4.29
9	6.03	2.55	8.77	7.45	8.65	8.05
10	1.75	1.95	9.02	2.78	4.29	6.03

`Gela1` aldagaian lerroak ikasleak izanik eta zutabeak azterketaren galdera bakoitzeko puntuazioa bada. Hauxe da `Gela1[8,3]` elementuaren interpretazioa: `Gela1` ikasgelako zortzigarren ikasleak bere azterketaren hirugarren galderan `7.50` puntuko nota lortu egin du.

Taula baten elementuaren erreferentzia egiteko, taularen identifikadorea eta lerro-zutabe parea eman beharko da. Turbo Pascal lengoaiak bi notazio onartzen ditu, erabili dugun `Gela1[8,3]` non indize biak kortxete pare batez biltzen diren eta lerro eta zutabearen artean koma bat dagoen, eta beste idazkera hau `Gela1[8][3]` non indize bakoitza kortxete artean dagoen. Liburu honetan bi notazioetatik lehena aukeratu dugu.

```
Gela1[8,3] ≡ Gela1[8][3]
```

Arrayaren dimentsioak ikusitakoak baino handiagoak izan daitezke ere, adibidez orain erakutsiko den programan labe baten barruko puntuen tenperaturak array batean gordetzen dira. Demagun labe txiki bat dugula eta bere barneko neurriak[16] zentimetroetan 4x5x6 direla eta zentimetro kubiko bakoitzeko tenperatura gordetzeko array aldagai bat sortzen dugula, hona hemen `LabeaTenperaturazBete` izeneko programa non `Labea` arrayan balioak gordetzen diren eta programa amaitu aurretik arrayaren edukia pantailaratzen den.

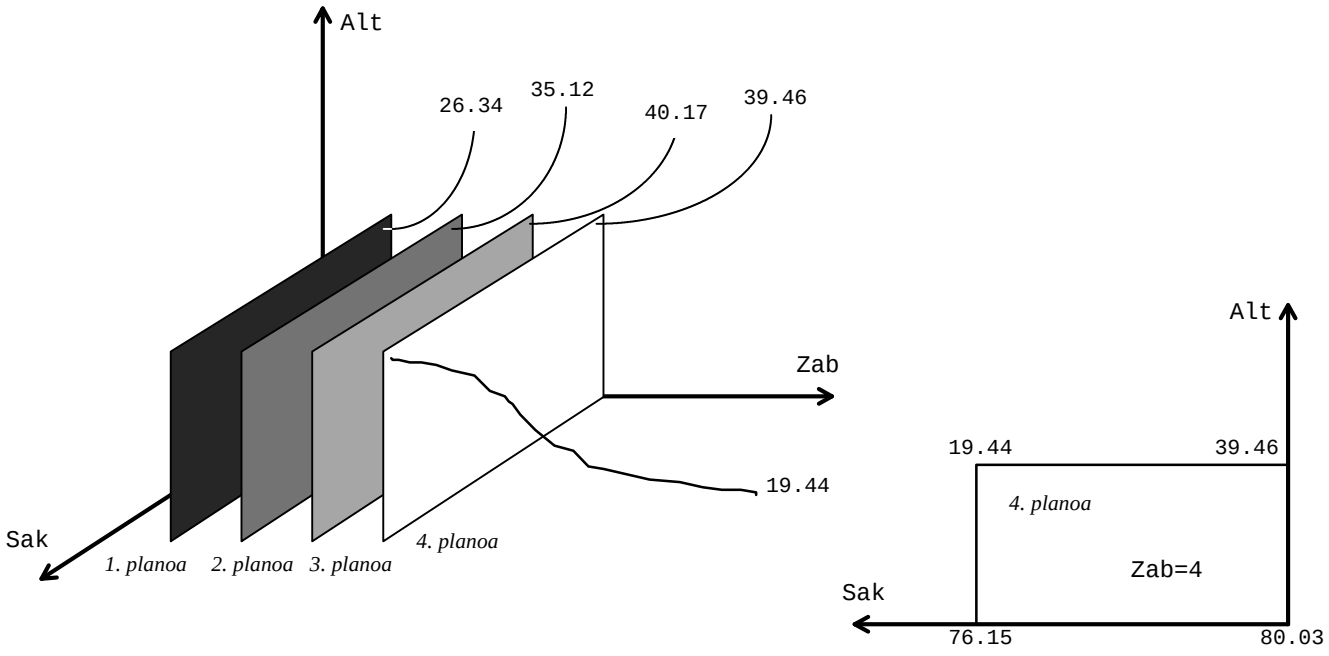
`Labea` arrayan balioak gordetzeko `LabeaBete()` prozedura erabiltzen da non balioak aleatorioki esleitzen diren, arrayaren edukia bistaratzeko `TenperaturakErakutsi()` izeneko prozedura bat idatzi izan da:

```
PROGRAM LabeaTenperaturazBete ;                    { \TP70\10\ARRAY_15.PAS }
CONST
    ZABALERAMAX = 4 ;
    ALTUERAMAX = 5 ;
    SAKONERAMAX = 6 ;
TYPE
    DM_Labe = ARRAY[1..ZABALERAMAX, 1..ALTUERAMAX, 1..SAKONERAMAX] OF Real ;

PROCEDURE LabeaBete (VAR L:DM_Labe) ;
VAR
  Zab, Alt, Sak : Byte ;
BEGIN
   FOR Zab:=1 TO ZABALERAMAX DO
      FOR Alt:=1 TO ALTUERAMAX DO
         FOR Sak:=1 TO SAKONERAMAX DO
              L[Zab,Alt,Sak] := Random * 100 ;
END ;
```

[16] Zabalera-Altuera-Sakonera.

```
PROCEDURE TenperaturakErakutsi (CONST L:DM_Labe) ;
VAR
  Zab, Alt, Sak : Byte ;
BEGIN
   FOR Zab:=1 TO ZABALERAMAX DO
   BEGIN
      FOR Alt:=1 TO ALTUERAMAX DO
      BEGIN
         FOR Sak:=1 TO SAKONERAMAX DO
         BEGIN
            Write (L[Zab,Alt,Sak] :10:2) ;
         END ;
         WriteLn ;
      END ;
      Write ('     ------------------------------------------------------ ') ;
      WriteLn (Zab, '. planoa') ;
   END ;
END ;

VAR
   Labea : DM_Labe ;

BEGIN              (* Programa Nagusia *)
   LabeaBete (Labea) ;
   TenperaturakErakutsi (Labea) ;
END .
```

Ikus daitekeenez `LabeaTenperaturazBete` izeneko programan bi azpirrutina daude batetik datuak biltegitzeko `LabeaBete()` prozedura eta bestetik datuak bistaratzeko balio duen `TenperaturakErakutsi()` prozedura. Azpiprograma biek hiru koordenatuko erreferentzi jakin bat darabilte, ondoko irudian erakusten den sistema koordenatua:



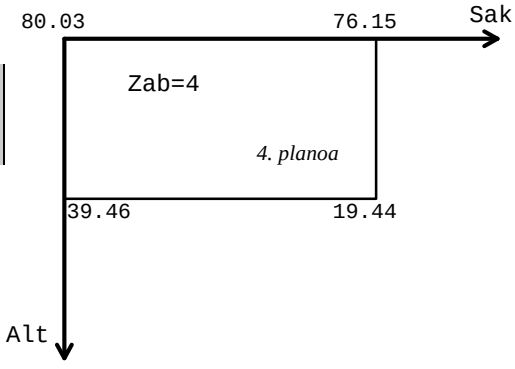
Irudietan agertzen denez `Labea` izeneko arrayak 4 x 5 x 6 zentimetroko labea ordezka dezake. Zentimetroko doitasun edo prezisioarekin ari bagara, labearen zabalera lau planotan bana daiteke eta plano bakoitzaren neurria 5 x 6 cm^2 izango da, beraz plano bakoitzak 30 puntu izango dituenez 30 tenperatura gorde ahal izango da. Esaterako, adibide-programa exekutatzean `LabeaBete()` prozeduran aleatorioki honako balioak gorde daitezke `Labea` arrayan:

```
     23.67     40.61     51.09     61.98     51.01     96.11
     11.27     17.33     50.27      4.00      1.99     92.96
     12.02      0.25     17.19     55.03     40.40      6.31
      2.37     46.92      5.10     64.44     55.55     13.63
     26.34     48.37     71.24     25.69     18.05     16.00
     ------------------------------------------------------- 1. planoa
     27.07     90.03     33.09     43.98     33.03     60.33
     33.27     37.77     30.27      9.00      3.99     92.90
     32.02      0.23     37.39     33.07     90.90      0.73
      2.77     90.92      3.30      9.99     34.33     37.07
     35.12     98.77     63.29     23.09     38.03     40.00
     ------------------------------------------------------- 2. planoa
      7.60     11.36     36.48     88.68     23.23     76.42
     12.25     90.45      0.57     99.40     53.92      2.92
      2.42      1.52     77.93      3.47      5.04     47.03
      2.07     92.29     15.21     44.38      4.96     69.35
     40.17     57.10     43.41     11.19     83.23     21.80
     ------------------------------------------------------- 3. planoa
     80.03     91.03      3.02     34.58     13.03     76.15
     53.78      7.74     11.47     29.18     93.83      2.94
     32.02      0.23     37.39     33.07     90.90      0.73
     11.70     92.94     63.37     57.39     32.73     32.47
     39.46      9.68      0.48      3.49     33.44     19.44
     ------------------------------------------------------- 4. planoa
_
```

Beltzago idatzi ditugun tenperaturak irudietan jarri ditugunak dira. Pantailaraketa ondo ulertzeko kontutan izan `TenperaturakErakutsi()` prozeduraren `FOR-DO` kabiatuak zabalera-altuera-sakonera ordenean daudela.

Beraz pantailaratzen den lehen tenperatura `L[1,1,1]` da, dagokion balioa `23.67` izanik, pantailaratzen den bigarren tenperatua `L[1,1,2]` da dagokion balioa `40.61` delarik, eta, barnerago dagoen `FOR-DO` kontrol-egitura bukatzean (hots `L[1,1,6]` puntuan) erakusten den balioa `96.11` da; ondoren erdiko `FOR-DO` kontrol-egitura inkrementatzen da bigarren lerroa pantailaratuz eta horrela jarraitzen du prozesuak 1. plano osoa agertu arte. Une horretan 2. planoaren lehen puntua tratatzen da `L[2,1,1]` zeinek duen balioa `27.07` den, eta aurreko modu beretsuan gainerako puntuak agertuz joango dira. Kanpoko `FOR-DO` sententziaren `Zab` kontrol-aldagaiak `ZABALERAMAX` balio duenean (`4` balio duenean) 4. planoaren tratamendua hasiko da, plano horren tenperaturak hauek dira:

```
     80.03     91.03      3.02     34.58     13.03     76.15
     53.78      7.74     11.47     29.18     93.83      2.94
     32.02      0.23     37.39     33.07     90.90      0.73
     11.70     92.94     63.37     57.39     32.73     32.47
     39.46      9.68      0.48      3.49     33.44     19.44
```

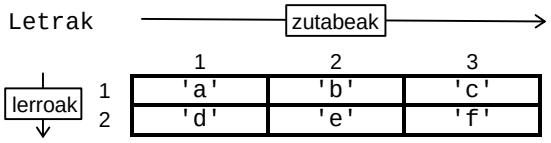


Ikus ditzagun, hurrengo bi puntuetan, dimentzioanitza den array baten biltegitzea memorian, eta egitura horri dagokion hasieraketa mota bat. Array dimentzioanitz bat memorian nola kokatzen den **10.3.1 Array dimentsioanitz baten biltegitzea memorian** puntuan garatuko dugu, eta array aldagai bati dagokion hasieraketa **10.3.2 Array dimentsioanitza den aldagai baten hasieraketa** izeneko puntuan. Ukatzerik ez dago puntu bi hauek lehentsuago azaldutako **10.2.1 Array dimentsiobakar baten biltegitzea memorian** eta **10.2.2 Array dimentsiobakarra den aldagai baten hasieraketa** izeneko puntuen hedapenak direla. Ondorioz haietan esandakoa, funtsean, baliagarria izango da array dimentsioanitzetan ere.

10.3.1 Array dimentsioanitz baten biltegitzea memorian

Ordenadoreen memoriaren helbidea begiraturik memoria lineala dela konturatzen gara, horregatik taula bat gordetzean (orokorrago hitz eginez, array dimentzioanitz bat memorian biltegitzean) linealtasun hori mantenduko da nahitaez. Demagun karaktereak biltzen dituen `Letrak` array baten dimentsioak 2 x 3 direla eta bere kokapena memorian aztertu nahi dugula, bi dira izan daitezkeen aukerak:

1. Zutabeka
2. Lerroka

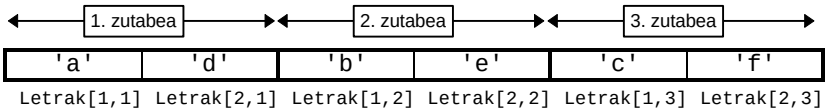



```
VAR
   Letrak = ARRAY[1..2, 1..3] OF Char ;
```

Zutabeka

Programazio lengoaia batek bi dimentsioko `Letrak` array bat zutabeka memorian biltegitzen duenean, lehen zutabearen helbide guztiak bigarren zutabearen helbide guztiak baino txikiagoak dira, eta bigarren zutabekoak hirugarren zutabeko helbideak baino lehen daude.

Honezkero, zutabeka biltegitzen den `Letrak` arrayak honelako itxura izango luke. Ikusten denez elementu guztiak elkar ondoan daude baina zutabeen arabera jarririk:

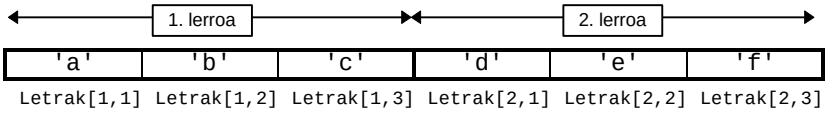


Array dimentsioanitzak zutabeka pilatzen dituzten lengoaien adibideak Fortran eta Basic dira.

Lerroka

Pascal eta C programazio lengoaiek berriz, bi dimentsioko `Letrak` array hori lerroka biltegituko lukete memorian. Hau da, hasierako memori posizioetan lehen lerroko elementu guztiak, ondoren bigarren lerrokoak eta horrela arrayaren azken lerroa kokatu arte.

Beraz, lerroka biltegitzen den `Letrak` array berak bestelako itxura izango luke zutabeka antolatzen denarekin alderatzen badugu. Ikusten denez, lerrokako biltegitzean ere elementu guztiak elkar ondoan daude:



Jarraian ematen den `ArrayDimentsioBikoitza` izeneko programan Turbo Pascal goi mailako lengoaiak memori posizioak betetzeko darabilen estrategia erakusten da. Lerroka lan egiten duela nabaria da, horregatik `Letrak` arrayaren helbideak elkar jarraian agertzen dira, izan ere lehen indizea (lerroarena) finkatu eta bigarren indizea (zutabeena) aldatuz elementu guztien edukia eta helbideak pantailaratzen ditugu lerrokako antolaketa dagoela frogatuz.

Hona hemen `ArrayDimentsioBikoitza` programaren aginduak:

```
PROGRAM ArrayDimentsioBikoitza ;                    { \TP70\10\ARRAY_16.PAS }
USES
    Crt, Ikus ;      (* IKUS.TPU  unitatea darabil *)
CONST
    BEHEMUGA = 1 ;
    LERROMAX = 2 ;
    ZUTABMAX = 3 ;
TYPE
    DM_Letrak = ARRAY[BEHEMUGA..LERROMAX, BEHEMUGA..ZUTABMAX] OF Char ;


PROCEDURE TaulaBete (VAR L:DM_Letrak) ;
VAR
   ler, zut : Byte ;
   Karaktere : Char ;
BEGIN
   Karaktere := 'a' ;
   FOR ler:=BEHEMUGA TO LERROMAX DO
     FOR zut:=BEHEMUGA TO ZUTABMAX DO
     BEGIN
        L[ler,zut] := Karaktere ;
        Karaktere := Succ (Karaktere) ;
     END ;
END ;


VAR
   Letrak : DM_Letrak ;
   l, z : Byte ;

BEGIN              (* Programa Nagusia *)
   ClrScr ;

   TaulaBete (Letrak) ;

   WriteLn ('BI DIMENTSIOKO ARRAY BATEN ELEMENTUEN 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':19, '       HELBIDEA', '   TAMAINA', '   EDUKIA') ;
   WriteLn ('-----------------------------------------------------':60) ;

   WriteLn ('Letrak --------> ':26, IntToHex (Seg (Letrak)),':',
            IntToHex (Ofs (Letrak)), '    (', SizeOf (Letrak), ')') ;
   WriteLn ;
   FOR l:=BEHEMUGA TO LERROMAX DO
     FOR z:=BEHEMUGA TO ZUTABMAX DO
     BEGIN
        Write ('Letrak[':16, l,',',z,'] ---> ', IntToHex (Seg (Letrak[l,z])) ) ;
        Write (':', IntToHex (Ofs (Letrak[l,z])) ) ;
        WriteLn ( '    (', SizeOf (Letrak[l,z]), ')', Letrak[l,z]:8) ;
     END ;
END .
```

`TaulaBete()` prozeduran `Letrak` arraya datuz betetzen da, horretarako `Karaktere` izeneko aldagai laguntzaile bat erabiltzen da, `Karaktere`-k hasieran `'a'` konstantea balioko du eta `FOR-DO` egitura kabiatuaren barnean inkrementatzen da `Succ()` funtzio estandarri esker.

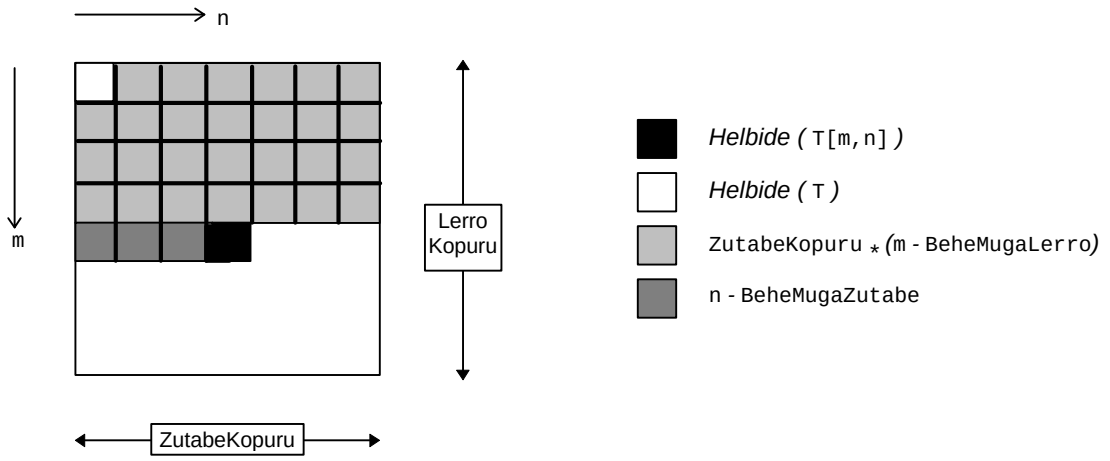
`ArrayDimentsioBikoitza` programari dagokion sententzien atalean pilaren, datuen eta kodearen helbideak lortu ondoren, `Letrak` arrayaren elementu bakoitzak duen helbidea tamaina eta edukia pantailaratzen dira. Elementuen edukiak eta elementuak identifikatzeko `Letrak[l,z]` moldea aztertuz, atera daitekeen ondorioa hauxe da: `Letrak` delako arrayaren elementuak lerroka antolatu ditu Turbo Pascal lengoaiak.

Lehendabiziko lerroa `59F8:00E2` eta `59F8:00E4` bitarteko helbideetan kokatzen da, eta bigarrena berriz `59F8:00E5` eta `59F8:00E7` artean. Hau da `ArrayDimentsioBikoitza` programaren irteera:

```
BI DIMENTSIOKO ARRAY BATEN ELEMENTUEN MEMORI HELBIDEAK
======================================================

PILARI dagokion segmentuaren hasiera:  5A26
DATUEI dagokien segmentuaren hasiera:  59F8
KODEARI dagokion segmentuaren hasiera: 5945


          ALDAGAIA       HELBIDEA   TAMAINA   EDUKIA
   ---------------------------------------------------------
         Letrak --------> 59F8:00E2    (6)

         Letrak[1,1] ---> 59F8:00E2    (1)       a
         Letrak[1,2] ---> 59F8:00E3    (1)       b
         Letrak[1,3] ---> 59F8:00E4    (1)       c
         Letrak[2,1] ---> 59F8:00E5    (1)       d
         Letrak[2,2] ---> 59F8:00E6    (1)       e
         Letrak[2,3] ---> 59F8:00E7    (1)       f
_
```

10.3.1 Array dimentsioanitz baten biltegitzea memorian puntua gogoratuz array baten edozein elementuaren helbidea ezagutzeko honako formula hau aplika dezakegu, non `T` bi dimentsioko arraya den eta `m`-garren lerroko eta `n`-garren zutabeko elementuaren helbidea lortu nahi den:

Helbide (`T[m,n]` *)* = *Helbide (* `T` *)* + *SizeOf (* `T[m,n]` *)* $*$ *[* `ZutabeKopuru` $*$ *(*`m - BeheMugaLerro`*)* + *(*`n - BeheMugaZutabe`*)]*



Esate baterako, aurreko adibidean `Letrak` arrayaren hasierako helbidea `59F8:00E2` da eta `Letrak[2,2]` bere bigarren lerroko eta bigarren zutabeko elementuaren helbidea honela kalkulatuko genuke formula erabiliz. Jakinik arrayak duen oinarrizko elementuaren tamaina `1` dela karakterea delako, eta lerroen zein zutabeen behemugak `1` direla:

Helbide (`T[m,n]` *)* = *Helbide (* `T` *)* + *SizeOf (* `T[m,n]` *)* $*$ *[* `ZutabeKopuru` $*$ *(*`m - BeheMugaLerro`*)* + *(*`n - BeheMugaZutabe`*)]*

Helbide (`Letrak[2,2]` *)* = *Helbide (*`Letrak`*)* + `1` $*$ *[* `3` $*$ *(*`2 - 1`*)* + *(*`2 - 1`*)]*

Helbide (`Letrak[2,2]` *)* = `59F8:00E2` + `1` $*$ *[* `3` $*$ *(*`2 - 1`*)* + *(*`2 - 1`*)]* = `59F8:00E2` + `1` $*$ *(*`3 + 1`*)*

Helbide (`Letrak[2,2]` *)* = `59F8:00E2` + `4` = `59F8:00E6`

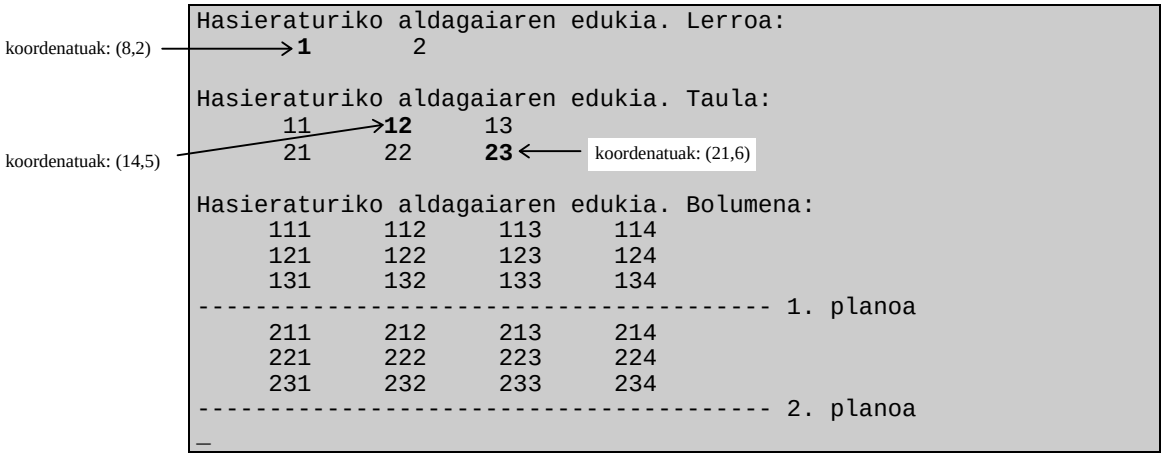
Guztira, `Letrak[2,2]` karakterearen helbidea `59F8:00E6` ateratzen da (aldez aurretik genekien bezala, ikus `ArrayDimentsioBikoitza` programaren irteera).

10.3.2 Array dimentsioanitza den aldagai baten hasieraketa

10.2.2 Array dimentsiobakarra den aldagai baten hasieraketa puntua aintzat harturik `CONST` bloke barruan aldagaiak deklara daitezkeela gogoratzen dugu, batez ere egituratuak diren aldagaiak hasieratzeko erabiltzen dena. Hau da, `CONST` blokearen barruan deklaraturiko aldagaiari bertan egiten zaio hasieraketa, ikus `ArrayenHasieraketak` adibide-programa:

```
PROGRAM ArrayenHasieraketak ;                    { \TP70\10\ARRAY_17.PAS }
USES
   Crt ;
TYPE
   DM_Lerroa   = ARRAY [1..2]             OF Integer ;
   DM_Taula    = ARRAY [1..2, 1..3]       OF Integer ;
   DM_Bolumena = ARRAY [1..2, 1..3, 1..4] OF Integer ;
CONST
   Lerroa   : DM_Lerroa   = ( 1, 2) ;

   Taula    : DM_Taula    = (
                              ( 11, 12, 13 ) ,
                              ( 21, 22, 23 )
                            ) ;

   Bolumena : DM_Bolumena = (
                              (
                                ( 111, 112, 113, 114 ) ,
                                ( 121, 122, 123, 124 ) ,
                                ( 131, 132, 133, 134 )
                              ) ,
                              (
                                ( 211, 212, 213, 214 ) ,
                                ( 221, 222, 223, 224 ) ,
                                ( 231, 232, 233, 234 )
                              )
                            ) ;
VAR
   i, j, k : Byte ;
BEGIN
   ClrScr ;
   WriteLn ('Hasieraturiko aldagaiaren edukia. Lerroa:') ;
   FOR i:= 1 TO 2 DO
   BEGIN
      GotoXY (i*8, 2) ;
      Write (Lerroa[i]) ;
   END ;

   WriteLn ;
   WriteLn ;
   WriteLn ('Hasieraturiko aldagaiaren edukia. Taula:') ;
   FOR i:= 1 TO 2 DO
      FOR j:=1 TO 3 DO
      BEGIN
         GotoXY (j*7, i+4) ;
         Write (Taula[i, j]) ;
      END ;

   WriteLn ;
   WriteLn ;
   WriteLn ('Hasieraturiko aldagaiaren edukia. Bolumena:') ;
   FOR i:= 1 TO 2 DO
   BEGIN
      FOR j:=1 TO 3 DO
      BEGIN
         FOR k:=1 TO 4 DO
            Write (Bolumena[i, j, k]:8) ;
         WriteLn ;
      END ;
      WriteLn ('-------------------------------------- ', i, '. planoa') ;
   END ;
END.
```

`ArrayenHasieraketak` adibide-programa honen irteera ikustean aztertu bereziki `GotoXY()` funtzioaren eta formatuen erabilpenak:




```
Hasieraturiko aldagaiaren edukia. Lerroa:
      1       2

Hasieraturiko aldagaiaren edukia. Taula:
      11     12     13
      21     22     23

Hasieraturiko aldagaiaren edukia. Bolumena:
     111     112     113     114
     121     122     123     124
     131     132     133     134
---------------------------------------- 1. planoa
     211     212     213     214
     221     222     223     224
     231     232     233     234
---------------------------------------- 2. planoa
_
```

Arrayen hasieraketa prozesuak izan dezakeen garrantziaz ohartzeko ondoko adibide-programa aztertzea proposatzen dugu. `Asamakizuna` deituriko adibide-programa horretan bi dimentsioko arraya erazagutu da `CONST` bloke barruan, kasu honetan `Taula` aldagaia ez da programa barruan aldatuko.

Programa nagusiaren hasieran, batetik hogeitamaika bitartera zenbaki bat asmatzea eskatzen zaio erabiltzaileari, ondoren, `Taula` lerroka agertuz pentsaturiko zenbakia bost zerrendetatik zeinetan dagoen galdetzen zaio. Erantzunen arabera zerrenden lehen kopuruak batuz asmaturiko zenbakia lortzen da.

Hona hemen `Asamakizuna` deituriko adibide-programa:

```
PROGRAM Asmakizuna ;                                  { \TP70\10\ARRAY_18.PAS }
USES
   Crt ;

FUNCTION DatuaIrakur : Char ;
VAR
   Erantz : Char ;
BEGIN
   REPEAT
      Write ('Zure zenbakia zerrenda ordenatu honetan dago? (B/E)  ') ;
      ReadLn (Erantz) ;
      Erantz := UpCase (Erantz) ;
   UNTIL (Erantz = 'B') OR (Erantz = 'E') ;
   DatuaIrakur := Erantz ;
END ;

CONST
   LERROMAX = 5 ;
   ZUTABMAX = 16 ;
TYPE
   DM_Taula = ARRAY [1..LERROMAX, 1..ZUTABMAX] OF Byte ;
CONST
   Taula : DM_Taula =
          (
            (  1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31 ) ,
            (  2,  3,  6,  7, 10, 11, 14, 15, 18, 19, 22, 23, 26, 27, 30, 31 ) ,
            (  4,  5,  6,  7, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29, 29, 31 ) ,
            (  8,  9, 10, 11, 12, 13, 14, 15, 24, 25, 26, 27, 28, 29, 30, 31 ) ,
            ( 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31 )
          ) ;
VAR
   i, j, Emaitza : Byte ;
   Erantzuna : Char ;
```



```
BEGIN                              { programa nagusia }
   ClrScr ;
   WriteLn ('1-etik 31-ra bitarteko zenbaki oso bat pentsatu eta gogoan hartu') ;
   WriteLn ('============================================================') ;

   Emaitza := 0 ;
   FOR i:= 1 TO LERROMAX DO
   BEGIN
      WriteLn ;
      FOR j:=1 TO ZUTABMAX DO
      BEGIN
         Write (Taula[i,j] : 4) ;
      END ;
      WriteLn ;
      Erantzuna := DatuaIrakur ;

      IF Erantzuna = 'B' THEN
        Emaitza := Emaitza + Taula [i,1] ;
   END ;

   WriteLn ;
   Write ('Zuk asmatutako zenbakia ', Emaitza, ' da') ;
END.
```

Eta bere balizko exekuzio bat:

```
1-etik 31-ra bitarteko zenbaki oso bat pentsatu eta gogoan hartu
============================================================

   1   3   5   7   9  11  13  15  17  19  21  23  25  27  29  31
Zure zenbakia zerrenda ordenatu honetan dago? (B/E)  b

   2   3   6   7  10  11  14  15  18  19  22  23  26  27  30  31
Zure zenbakia zerrenda ordenatu honetan dago? (B/E)  b

   4   5   6   7  12  13  14  15  20  21  22  23  28  29  29  31
Zure zenbakia zerrenda ordenatu honetan dago? (B/E)  e

   8   9  10  11  12  13  14  15  24  25  26  27  28  29  30  31
Zure zenbakia zerrenda ordenatu honetan dago? (B/E)  e

  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31
Zure zenbakia zerrenda ordenatu honetan dago? (B/E)  b

Zuk asmatutako zenbakia 19 da
_
```

10.4 ARRAY DIMENTSIOBAKARREN GAINEKO ERAGIKETAK

Array dimentsiobakarren gaineko eragiketak azaltzeko erabiliko dugun notazioa honako hau izango da:

`Aldagaia ← Balioa`	agindu honek esleipena adierazten du
`A[Indize]`	honek `A` arrayaren elementu bat erreferenziatzen du, `Indize` izeneko indizedun elementua hain zuzen ere
`A[Indize] > Gakoa`	hau `A` arrayaren `Indize`-dun elementua eta `Gakoa`-ren arteko konparaketa bat litzateke

Goiko horiek aintzat harturik Turbo Pascal lengoaiaren ezagunak diren kontrol-egiturak pseudokodean jar ditzagun:

- IF-THEN

```
BALDIN Kondizioa ORDUAN
     Sententzia
AMAIA_BALDIN
```

- IF-THEN-ELSE

```
BALDIN-ETA Kondizioa ORDUAN
     Sententzia_1
BESTELA
     Sententzia_2
AMAIA_BALDIN-ETA
```

- CASE-OF

```
BALORATUZ Espresioa ORDUAN
     Balio_1 : Sententzia_1
     Balio_2 : Sententzia_2
     Balio_3 : Sententzia_3
GAINERAKOAK
     Sententzia_N
AMAIA_BALORATUZ
```

- WHILE-DO

```
BAI-BITARTEAN Kondizioa EGIN
     Sententzia
AMAIA_BAI-BITARTEAN
```

- REPEAT-UNTIL

```
ERREPIKA
     Sententzia
EZ-BITARTEAN Kondizioa
```

- FOR-TO-DO

```
BEHEMUGATIK Indize ← Balio1 GOIMUGARAINO Balio2 EGIN
     Sententzia
AMAIA_BEHEMUGATIK
```

- FOR-DOWNTO-DO

```
GOIMUGATIK Indize ← Balio1 BEHEMUGARAINO Balio2 EGIN
     Sententzia
AMAIA_GOIMUGATIK
```

Hurrengo puntuetan erabiliko den Z arrayaren ezaugarriak hauek lirateke. Z-ren esparrua finkatzen duten balioak 1 eta MAX dira, beraz Z arrayaren luzera fisikoa MAX izango da. Z arrayaren luzera efektiboa edo luzera logikoa N izango da, zein 1 eta MAX bitartekoa izango den. Z-ren elementuak atzitzeko Indize izeneko aldagai laguntzaile batez egingo da.

10.4.1 Ibilera

Enuntziatua array baten elementu “guztiei”[17] prozesaketa jakin bat aplikatu.

[17] Array baten elementu “guztien” kopurua bere luzera logikoari esker jakin daiteke.

Ebazpena array baten elementu guztiak luzera logikoak zehazten dituenak direnez, prozesatu beharreko elementuen indizeak 1, 2, 3, ... , N dira. Hau da, prozesatuko den lehen elementua 1 da eta azkena berriz N, ondorioz lan hori burutzeko kontrol-egiturarik aproposena FOR-TO-DO denez ez lirateke WHILE-DO edo REPEAT-UNTIL erabili beharko.

Pseudokodea

(Gomendatua)

```
BEHEMUGATIK Indize ← 1 GOIMUGARAINO N EGIN
        prozesatu Z[Indize]
AMAIA_BEHEMUGATIK
```

(Ez gomendatuak)

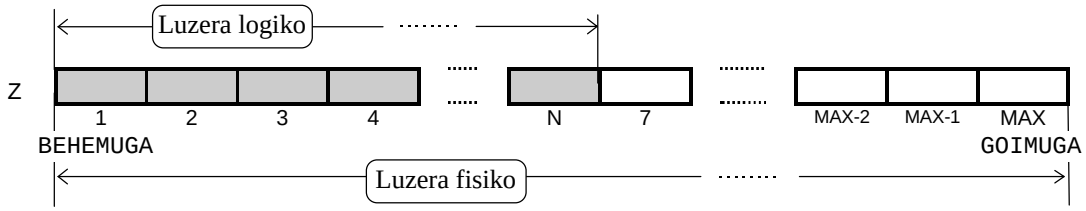
```
Indize ← 1
BAI-BITARTEAN Indize <= N EGIN
      prozesatu Z[Indize]
      Indize ← Indize + 1
AMAIA_BAI-BITARTEAN
```

```
Indize ← 1
ERREPIKA
      prozesatu Z[Indize]
      Indize ← Indize + 1
EZ-BITARTEAN Indize > N
```

Kodea

```
FOR Indize:=1 TO N DO
  WriteLn (Z[Indize]) ;   { prozesaketa elementuak bistaratzea izanik }
```

Eskema



Adibidea array baten zenbait elementu Random() funtzio estandarraz bete ondoren gordetako balioak pantailaratu:

```
PROGRAM IbileraArrayetan ;                          { \TP70\10\ALGOR_01.PAS }
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 40 ;
TYPE
  DM_Zerrenda = ARRAY[BEHEMUGA..GOIMUGA] OF Integer ;

PROCEDURE ArrayaBete (VAR A : DM_Zerrenda; VAR Luzera : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   Randomize ;
   Luzera := Random(GOIMUGA) + 1 ;

   WriteLn ('A arrayan ', Luzera, ' datu gordetzen') ;
   FOR Indizea:=BEHEMUGA TO Luzera DO
   BEGIN
      A[Indizea] := Random(100) ;         { 0 eta 99 arteko balioak }
   END ;
END ;
```

```
PROCEDURE ArrayaIkusi (CONST A : DM_Zerrenda; Luzera : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   WriteLn ('A arrayaren edukia: ') ;
   FOR Indizea:=BEHEMUGA TO Luzera DO
   BEGIN
      Write (A[Indizea] : 5) ;
   END ;
   WriteLn ;
END ;

VAR
  A : DM_Zerrenda ;
  Luzera : Byte ;
BEGIN
   ArrayaBete (A, Luzera) ;
   ArrayaIkusi (A, Luzera) ;
END.
```

`IbileraArrayetan` programaren irteeraren bat hau izan daiteke:

```
A arrayan 12 datu gordetzen
A arrayaren edukia:
   15   59    6   87   13    0   34   96   99   25   41   79
_
```

10.4.2 Bilaketa

Array batean burutzen den bilaketa honela definituko dugu: bilatu nahi den balioa gakoa izango da eta aldez aurretik ezaguna da, arrayaren elementuak aztertuko dira gako horrekin konparatuz, baldin eta elementu baten balioa gakoarena bada elementuari dagokion indizea[18] bilaketaren soluzioa da, gakoak duen balioa arrayan aurkitzen ez bada bilaketak huts egiten du.

Bilaketa mota bi ikasiko ditugu, bilaketa sekuentziala eta bilaketa bitarra. Lehenengoa edozein kasutan erabil daiteke baina bilaketa bitarra egin ahal izateko Z arrayaren elementuak ordenaturik egon beharko dira. Bilaketa sekuentzialak, bilatzen ari den gakoak Z arrayan duen posizioaren arabera, denbora eta kostua handiak behar izango ditu. Bilaketa bitarra ere bilatzen ari den gakoak Z arrayan duen posizioaren araberakoa da baina algoritmo honen konbergentzia bizkoarragoa denez prozesaketa kostua txikiagoa izaten da.

10.4.2.1 Bilaketa lineala

Enuntziatua gako bat emanik, balio hori duen arrayaren lehen elementuari dagokion indizea aurkitu.

Ebazpena ebazpenak bi urrats ditu.

Lehenengoan, arrayaren hasieratik abiatuta bere elementuak gakoarekin alderatuko dira bigizta batean, bigiztatik irteteko baldintza konposatua

[18] Gakoarekin parekatzen diren elementuak bat baino gehiago izatean, bilaketaren soluzioa lehen agerpenaren indizea litzateke.

izango da (gakoarekin paraketzen den elementuren bat aurkitzean bigizta moztu, eta Z arrayaren N luzera efektibotik kanpoko elementuak ez kontsideratu).

Ebazpenaren bigarren urratsak bigiztaren kondizio konposatuarekin zerikusia du, hots, baldintza konposatuak honela dio gure hizkuntza naturalean: *gakoa aurkitzen ez dugun bitartean*[1] *eta arrayaren zati efektiboan gauden bitartean*[2] *jarraitu hurrengo konparaketa eginez*. Eta bigiztatik irtetean zehaztu beharra dago 1 ala 2 baldintzak huts egin duen, hots, elementu bat bilatu egin den Z arrayan ala gakoaren balioko elementurik ez dagoen.

Hau guztiagatik proposatzen dugun kontrol-egitura WHILE-DO da, nahiz eta REPEAT-UNTIL ere aproposa izan. Baina kontutan izan FOR-TO-DO egitura ezin daitekeela erabili, zergatik?.

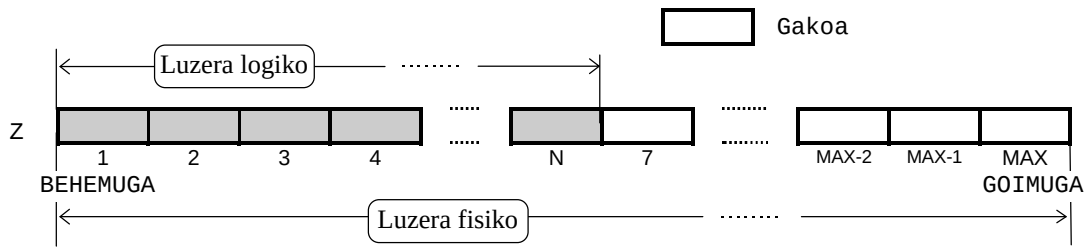
Pseudokodea

```
Indize ← 1
Aurkitua ← FALSE
BAI-BITARTEAN (Indize <= N) AND NOT Aurkitua EGIN
     BALDIN-ETA Z[Indize] = Gakoa ORDUAN
          Aurkitua ← TRUE
     BESTELA
          Indize ← Indize + 1
     AMAIA_BALDIN-ETA
AMAIA_ BAI-BITARTEAN

BALDIN-ETA Aurkitua ORDUAN
    Gakoa-ren posizioa Indize da
BESTELA
    Gakoa ez dago Z arrayan
AMAIA_BALDIN-ETA
```

Kodea

```
ReadLn (Gakoa) ;      { suposatuz Gakoa teklatuz irakurtzen dela }
Indize := 1 ;
Aurkitua := FALSE ;
WHILE (Indize <= N) AND NOT Aurkitua DO
BEGIN
   IF Z[Indize] = Gakoa THEN
     Aurkitua := TRUE
   ELSE
     Indize := Indize + 1 ;
END ;
IF Aurkitua THEN
  WriteLn (Gakoa, '-ren lehen agerpena ', Indize, ' posizioan ematen da')
ELSE
  WriteLn (Gakoa, ' ez da arrayan aurkitzen') ;
```

Eskema



Adibidea array baten zenbait elementu `Random()` funtzio estandarraz bete ondoren gako bat teklatuz irakurtzen da. Gakoaren lehen agerpena itzultzen duen funtzioak bilaketa lineala edo bilaketa sekuentziala darabil, gakoa arrayan aurkitzen denean funtzioak bere lehen agerpenari dagokion indizea itzuliko dio programa nagusiari, baina, gakoa arrayan ez dagoenaren egoerataz programa nagusia konturatu dadin funtzioak balio berezi bat itzultzen dio (posizioa ezin daitekeen izan balio berezia, adibidez 0).

```
PROGRAM BilaketaLinealaArrayetan ;                { \TP70\10\ALGOR_02.PAS }
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 40 ;
TYPE
  DM_Zerrenda = ARRAY[BEHEMUGA..GOIMUGA] OF Integer ;

PROCEDURE ArrayaIkusi (CONST A : DM_Zerrenda; Luzera : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   WriteLn ('A arrayaren edukia: ') ;
   FOR Indizea:=BEHEMUGA TO Luzera DO
   BEGIN
      Write (A[Indizea] : 8) ;
   END ;
   WriteLn ;
END ;

PROCEDURE ArrayaBete (VAR A : DM_Zerrenda; VAR Luzera : Byte) ;
VAR
  Kont : Byte ;
  Zeinua : Integer ;
BEGIN
   Randomize ;
   Luzera := Random(GOIMUGA) + 1 ;
   WriteLn ('A arrayan ', Luzera, ' datu gordetzen') ;
   FOR Kont:=BEHEMUGA TO Luzera DO
   BEGIN
      IF Random(2)=0 THEN            { zeinua aleatorioki lortzen da }
        Zeinua := -1
      ELSE
        Zeinua := 1 ;
      A[Kont] := Zeinua*Random(100) ;
   END ;
END ;

FUNCTION BilaketaLineala (CONST A : DM_Zerrenda;
                          Luzera : Byte;
                          Gakoa : Integer) : Byte ;
VAR
   Indizea : Byte ;
   Aurkitua : Boolean ;
BEGIN
   Indizea := BEHEMUGA ;
   Aurkitua := FALSE ;
   WHILE (Indizea <= Luzera) AND NOT Aurkitua DO
   BEGIN
      IF A[Indizea] = Gakoa THEN
        Aurkitua := TRUE
      ELSE
        Indizea := Indizea + 1 ;
   END ;

   IF Aurkitua THEN
     BilaketaLineala := Indizea
   ELSE
     BilaketaLineala := 0 ;          { 0 gezurrezko posizioa litzateke }
END ;
```

```
VAR
  A : DM_Zerrenda ;
  Luzera, Posizioa : Byte ;
  Gakoa : Integer ;
BEGIN
   ArrayaBete (A, Luzera) ;
   ArrayaIkusi (A, Luzera) ;

   Write ('Bilatu nahi den balioa eman: ') ;
   ReadLn (Gakoa) ;

   Posizioa := BilaketaLineala (A, Luzera, Gakoa) ;

   IF Posizioa = 0 THEN
     WriteLn (Gakoa, ' ez dago arrayan')
   ELSE
     WriteLn (Gakoa, ' balioari dagokion indizea arrayan ', Posizioa, ' da')
END.
```

`BilaketaLinealaArrayetan` programaren irteeraren bat hau izan daiteke:

```
A arrayan 9 datu gordetzen
A arrayaren edukia:
     -78      59      -6      87      23     -59      10     -59      25
Bilatu nahi den balioa eman: -59
-59 balioari dagokion indizea arrayan 6 da
_
```

10.4.2.2 Bilaketa bitarra

Enuntziatua demagun `Z` arrayaren *elementuak ordenaturik daudela*, gako ezagun bat zehaztu ondoren, balio hori aintzat harturik arrayaren lehen elementuari dagokion indizea aurkitu.

Ebazpena ebazpenak hiru urrats dituela onar daiteke.

Lehenengoan, arrayaren esparru baliagarria zehaztu behar da. Hasierako esparrua arrayaren `1` behemuga eta `Z` arrayaren `N` luzera efektiboaren bitartekoa izango da.

Ebazpenaren bigarren urratsean arrayaren esparru baliagarria erdibituz bi zatitan banatzen da, eta, esparruaren erdian dagoen elementuaren balioa gakoarekin konparatzen da ondoko egoera bat gertatuz:

- Berdinak izatea, bilatzen ari ginen gakoa `Z` arrayan aurkitzen da eta berari dagokion posizioa esparruaren erdiko elementuaren indizearena da
- Erdiko elementuaren balioa gakoa baino handiagoa bada, esparru baliagarria aldatu behar da (berria izango den esparrua aurrekoaren hasierako erdia delarik)
- Erdiko elementuaren balioa gakoa baino txikiagoa bada, esparru baliagarria aldatu behar da ere, baina berria izango den esparrua aurrekoaren amaierako erdia izanik

Hirugarren urratsak ebazpenaren bigarren urratseko prozesu errepikakorraren irteerako kondizioarekin zerikusia du. Bigizta bukatzeko, bi baldintza hauetatik bat betetzea aski da: *gakoa arrayan aurkitzea*[1]

(esparru baliagarriaren erdiko elementuaren balioa eta gakoa berdinak direla frogatu da), edo bestela, *gakoa ez dago arrayan*[2] (esparru baliagarriak ez du elementurik barneratzen, duen goimuga bere behemuga baino txikiagoa delako).

Kasu honetan ere proposatzen ditugun kontrol-egiturak `WHILE-DO` eta `REPEAT-UNTIL` dira, baina ezin daiteke inolaz ere `FOR-TO-DO` egitura erabili.

Pseudokodea Hauek lirateke goiko algoritmoa burutzeko beharko liratekeen aldagaien identifikadoreak eta euren esanahia:

`Gakoa` bilatzen ari den balioa aldagai honetan gordetzen da.

`Aurkitua` aldagai boolear laguntzailea, hasiera batean `Gakoa` arrayan topatu ez denean bere balioa `FALSE` izango da.

`Hasi` bilatzen ari garen gakoa `Z` arrayaren barruti jakin batean egon daiteke, esparru horren behemuga `Hasi` aldagaian pilatzen da. Hots, `Hasi` bilaketa-esparruaren mutur bat definitzen duen indizea da. Horregatik bilaketaren hasieran `Hasi`-k duen balioa `1` da, behemuga alegia.

`Amai` bilatzen ari garen gakoa `Z` arrayan egon daitekeen barrutiari dagokion esparruaren goimuga `Amai` izeneko aldagaian biltegitzen da. Indizea den `Amai` bitartez bilaketa-esparruaren beste muturra definitzen denez, bilaketa hasieran dagokion balioa `N` luzera logikoa da.

`Erdi` bilatzen ari garen gakoa `Z` arrayan egon daitekeen barrutiaren erdiko balioaren indizea. Gakoaren posizioa arrayan, bilaketa algoriaren emaitza, aldagai honetan aurkitzen da.

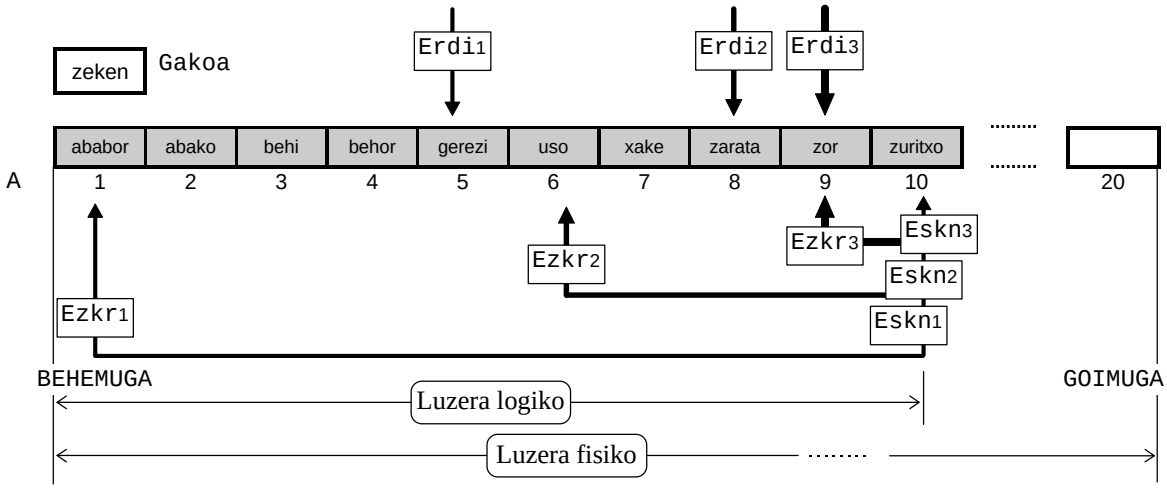
```
Aurkitua ← FALSE
Hasi ← 1
Amai ← N

ERREPIKA
     Erdi ← (Hasi + Amai) DIV 2          {zatiketa osoa}

     BALDIN-ETA Z[Erdi] = Gakoa ORDUAN
          Aurkitua ← TRUE
     BESTELA
          BALDIN-ETA Z[Erdi] > Gakoa ORDUAN
               Amai ← Erdi-1
          BESTELA
               Hasi ← Erdi+1
          AMAIA_BALDIN-ETA
     AMAIA_BALDIN-ETA
EZ-BITARTEAN Aurkitua EDO (Hasi > Amai)

BALDIN-ETA Aurkitua ORDUAN
    Gakoa-ren posizioa Erdi da
BESTELA
    Gakoa ez dago Z arrayan
AMAIA_BALDIN-ETA
```

Kodea

```
ReadLn (Gakoa) ;      { suposatuz Gakoa teklatuz irakurtzen dela }
Aurkitua := FALSE ;
Hasi := 1 ;
Amai := N ;
```

```
REPEAT
    Erdi := (Hasi+Amai) DIV 2 ;
    IF Z[Erdi] = Gakoa THEN
      Aurkitua := TRUE
    ELSE
      IF Z[Erdi] > Gakoa THEN
            Amai := Erdi - 1
      ELSE
            Hasi := Erdi + 1 ;
UNTIL Aurkitua OR (Hasi > Amai) ;

IF Aurkitua THEN
   WriteLn (Gakoa, '-ren lehen agerpena ', Erdi, ' posizioan ematen da')
ELSE
   WriteLn (Gakoa, ' ez da arrayan aurkitzen') ;
```

Eskema eskema hau jarraian datorren adibidearen exekuzio bati dagokio:



Adibidea array baten bitartez hitzak gordeko ditugu memorian, arrayari balioak ematean hitzen orden alfabetikoa zainduko denez esan daiteke arraya hiztegi bat dela.

Hiztegia datuz bete ondoren gako bat teklatuz irakurtzen da, adibide honetan gakoa hitz bat izango da noski. Gakoaren lehen[19] agerpena itzultzen duen funtzioak bilaketa bitarra darabil, gakoa arrayan aurkitzen denean funtzioak bere agerpen horri dagokion indizea itzuliko dio programa nagusiari, baina, gakoa arrayan ez dagoenaren egoerataz programa nagusia konturatu dadin funtzioak balio berezi bat itzultzen dio (posizioa ezin daitekeen izan balio berezia, adibidez 0).

Jarraian erakusten den `BilaketaBitarraArrayetan` programaren bi exekuzio egingo ditugu. Hamar hitz gorde ondoren, hiztegia bete ondoren, lehenengo exekuzioan hiztegian ez dagoen hitz bat emango dugu (`zeken` adjektiboa alegia), hori dela eta programak gako hori ez duela aurkitu erantzungo du. Bigarren exekuzioan hiztegian dagoen `uso` hitza emango dugu gakotzat, oraingoan gako horri hiztegian dagokion posizioa agertuko du programak.

[19] Hiztegian elementu errepikatuak ezin dira egon bere datuak gordetzeko erabiltzen den prozedurak hori debekatzen duelako, ondorioz bilaketa arrakastatsua gertatzen denean (hots, gako-hitza hiztegian aurkitzen denean) bilaketa bitarrak zehazten duen emaitza hitzaren lehen agerpena da, lehena eta bakarra baita.

```
PROGRAM BilaketaBitarraArrayetan ;                        { \TP70\10\ALGOR_03.PAS }
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 20 ;

TYPE
  DM_Hitza = String[15] ;
  DM_Hiztegia = ARRAY[BEHEMUGA..GOIMUGA] OF DM_Hitza ;


PROCEDURE ArrayaIkusi (CONST A : DM_Hiztegia; Luzera : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   WriteLn ('A arrayaren edukia: ') ;
   FOR Indizea:=BEHEMUGA TO Luzera DO
   BEGIN
      Write (A[Indizea] : 20) ;
   END ;
   WriteLn ;
END ;


PROCEDURE ArrayaOrdenezBete (VAR A : DM_Hiztegia; VAR Luzera : Byte) ;
VAR
  Kont : Byte ;
  Laguntzaile : DM_Hitza ;
BEGIN
   Randomize ;
   Luzera := Random(GOIMUGA) + 1 ;

   WriteLn ('A arrayan ', Luzera, ' hitz gordetzen') ;

   A[BEHEMUGA] := 'ababor' ;          { hiztegiaren aurreneko hitza }
   FOR Kont:=BEHEMUGA+1 TO Luzera DO
   BEGIN
      REPEAT
         Write (Kont, '. hitza eman: ') ;
         ReadLn (A[Kont]) ;
      UNTIL A[Kont] > A[Kont-1] ;     { hurrengo elementua aurrekoa }
   END ;                              { baino handiagoa izango da   }
END ;


FUNCTION BilaketaBitarra (CONST A : DM_Hiztegia;
                          Luzera : Byte;
                          Gakoa : DM_Hitza) : Byte ;
VAR
   Ezkr, Eskn, Erdi : Byte ;
   Aurkitua : Boolean ;
BEGIN
   Aurkitua := FALSE ;
   Ezkr := BEHEMUGA ;
   Eskn := Luzera ;

   REPEAT
      Erdi := (Ezkr+Eskn) DIV 2 ;

      WriteLn ('Ezkr =', Ezkr:2,'     Eskn =', Eskn:2,'     Erdi =', Erdi:2) ;

      IF A[Erdi] = Gakoa THEN
        Aurkitua := TRUE
      ELSE
        IF A[Erdi] > Gakoa THEN
          Eskn := Erdi - 1
        ELSE
          Ezkr := Erdi + 1 ;

   UNTIL (Ezkr > Eskn) OR Aurkitua ;
```

```
   Write ('Ezkr =', Ezkr:2,'     Eskn =', Eskn:2,'     Erdi =', Erdi:2) ;
   WriteLn ('  <------Azkenekoa');

   IF Aurkitua THEN
     BilaketaBitarra := Erdi
   ELSE
     BilaketaBitarra := 0 ;          { 0 gezurrezko posizioa litzateke }
END ;


VAR
  A : DM_Hiztegia ;
  Luzera, Posizioa : Byte ;
  Gakoa : DM_Hitza ;
BEGIN
   ArrayaOrdenezBete (A, Luzera) ;
   WriteLn ;
   ArrayaIkusi (A, Luzera) ;

   WriteLn ;
   Write ('Bilatu nahi den hitza eman: ') ;
   ReadLn (Gakoa) ;

   Posizioa := BilaketaBitarra (A, Luzera, Gakoa) ;

   IF Posizioa = 0 THEN
     WriteLn ('''', Gakoa, ''' ez dago arrayan')
   ELSE
     WriteLn ('''', Gakoa, ''' hitzari dagokion indizea arrayan ', Posizioa, ' da')
END.
```

`BilaketaBitarraArrayetan` programaren irteeraren bat hau izan daiteke. Suposatuz aukeratutako gakoa hiztegian gorde diren datuekin paraketzen ez dela:

```
A arrayan 10 hitz gordetzen
A arrayaren edukia:
2. hitza eman: abako
3. hitza eman: behi
4. hitza eman: behor
5. hitza eman: gerezi
6. hitza eman: uso
7. hitza eman: xake
8. hitza eman: zarata
9. hitza eman: zor
10. hitza eman: zuritxo

A arrayaren edukia:
             ababor               abako                behi                behor
             gerezi                 uso                xake              zarata
                zor             zuritxo

Bilatu nahi den hitza eman: zeken
Ezkr = 1     Eskn =10     Erdi = 5
Ezkr = 6     Eskn =10     Erdi = 8
Ezkr = 9     Eskn =10     Erdi = 9
Ezkr = 9     Eskn = 8     Erdi = 9  <------Azkenekoa
'zeken' ez dago arrayan
_
```

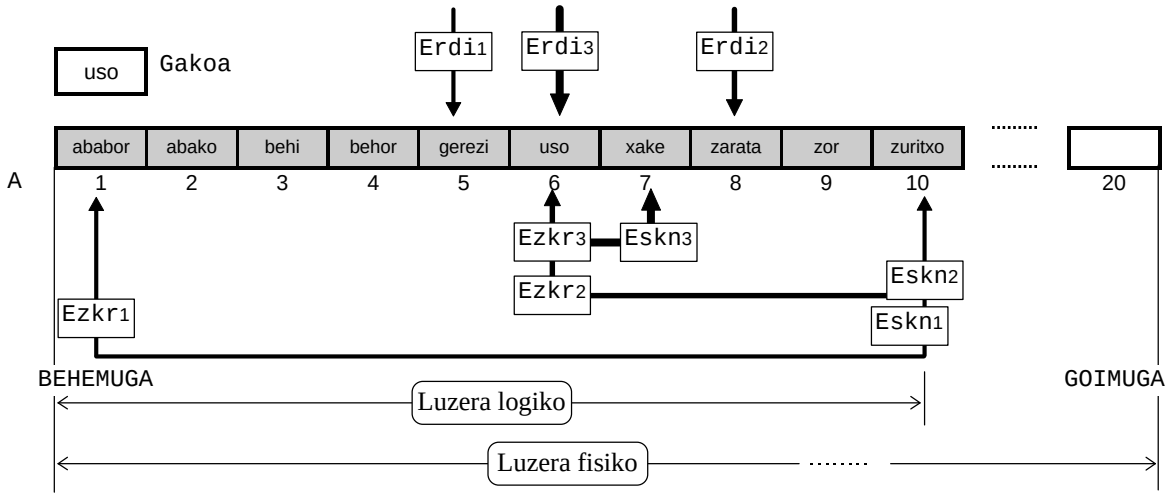
Hiztegiaren datuak eta gakoa direla eta, algoritmoak iterazio bakoitzean `zeken` hitz helburuari dagokion barrutia gero eta xehetasun handiagoz mugatzen du (ikus 10-45 orrialdeko eskema). Lehen iterazioan barrutia 1-tik 10-ra doa, eta `zeken` gakoa `gerezi` hitzarekin konparatzen denez bigarren iterazioan barrutia 6-tik 10-ra hedatzen da. Bigarren iterazioan gakoa zortzigarren hitza den `zarata`-rekin parekatzen da eta ondorioz hirugarren iteraziorako barrutia estuagoa izango da (9-tik 10-ra). Hirugarren eta azken iterazioan

bederatzigarren hitza den `zor` eta gakoa konparatzean barrutiaren goimuga aldatu beharra dagoela kontura gaitezke (`Eskn` aldagaian `Erdi-1` balioa gorde behar da, hau da, `Eskn` aldagaian `9-1` balioa gordeko da). Ondorioz barruti berria 9-tik 8-ra zabaltzen da, honek gakoa arrayan ez dagoela adierazten digu.

Ekin diezaiogun `BilaketaBitarraArrayetan` programaren bigarren exekuzioari, hiztegiaren datu berdinekin baina `uso` gakorako programa beraren irteera honako hau litzateke:

```
A arrayan 10 hitz gordetzen
A arrayaren edukia:
2. hitza eman: abako
3. hitza eman: behi
4. hitza eman: behor
5. hitza eman: gerezi
6. hitza eman: uso
7. hitza eman: xake
8. hitza eman: zarata
9. hitza eman: zor
10. hitza eman: zuritxo

A arrayaren edukia:
             ababor              abako               behi               behor
             gerezi                uso               xake             zarata
                zor            zuritxo

Bilatu nahi den hitza eman: uso
Ezkr = 1     Eskn =10     Erdi = 5
Ezkr = 6     Eskn =10     Erdi = 8
Ezkr = 6     Eskn = 7     Erdi = 6
Ezkr = 6     Eskn = 7     Erdi = 6  <------Azkenekoa
'uso' hitzari dagokion indizea arrayan 6 da
_
```

Bigarren exekuzioaren eskema hau izanik:



`BilaketaBitarraArrayetan` programan erakutsi den bilaketa bitarraren algoritmoa egokia izango litzateke hiztegi batetarako, bertan agertzen diren hitzak errepikatzen ez direlako. Baina zer aldaketak egin beharko lirateke, arrayaren datuak ordenatuak egoteaz gain, datuak errepikaturik agertzea onartuko balitz?. Galdera hori erantzuteko asmoz beste programa bat idatzi dugu `BilaketaBitarraArrayErrepikatuak` deitu duguna, eta dituen aldaketak bi dira.

Lehenengo berrikuntza `ArrayaOrdenezBete()` prozedurari dagokion amaieran dagoena da, hots, hitz berri bat ematean bere aurrekoa baino handiagoa edo berdina izatea ahalbidetzen duen aldaketa:

```
A[BEHEMUGA] := 'ababor' ;          { hiztegiaren aurreneko hitza }
FOR Kont:=BEHEMUGA+1 TO Luzera DO
BEGIN
   REPEAT
      Write (Kont, '. hitza eman: ') ;
      ReadLn (A[Kont]) ;
   UNTIL A[Kont] >= A[Kont-1] ;    { hurrengo elementua aurrekoa baino }
END ;                              { handiagoa edo berdina izango da   }
```

Bigarrena, aldaketa izan ezik gehiketa da, bilaketa arrakastatsua denean beltzez markaturiko kodea gehituko zaio bilaketa burutzen duen `BilaketaBitarra()` funtzioari, non `Kont` kontagailu laguntzailea funtzio barnean deklaraturiko bertako aldagaia den:

```
IF Aurkitua THEN
  BEGIN
     Kont := Erdi ;
     WHILE (Kont > BEHEMUGA) AND (A[Kont] = A[Kont-1]) DO
        Kont := Kont - 1 ;

     BilaketaBitarra := Kont ;
  END
ELSE
  BilaketaBitarra := 0 ;           { 0 gezurrezko posizioa litzateke }
```

Ikusten denez gehitu zaion kodearen bitartez aurkitutako `Erdi` soluzioa ez da gakoaren lehendabiziko agerpena kontsideratzen, horregatik bere aurreko elementuak berdinak ote diren ikertzen da. Zergatik jarri da `(Kont > BEHEMUGA)` baldintza?. Zein kasutan beharrezkoa da?.

10.4.3 Tartekaketa

Enuntziatua demagun `Z` arraya erabat beterik ez dagoela, eta `K`-garren posizioan elementu berri bat txertatu (tartekatu) nahi dela. Arrayaren luzera logikoa `N` bada, tartekaketa ondoren `Z`-ren luzera efektibo hori `N+1` izango da.

Ebazpena ebazpenak hiru urrats ditu.

Ezaguna den `K`-garren posiziotik `N`-garren posizioraino (`N` arrayaren luzera logikoa litzateke) elementu guztiak eskuinerantz posizio bat desplazatu. Lehen urrats honen bitartez arrayan “tokia” egiten da `K`-garren posizioan, eta lan hori gauzatzeko kontrol-egiturarik aproposena `FOR-DOWNTO-DO` denez ez lirateke `WHILE-DO` edo `REPEAT-UNTIL` sententziak erabili beharko.

Bigarrenean tartekaketa burutu egiten da, lortu den toki librean elementu berria kopiatzen da alegia.

Tartekatu nahi den elementu berria arrayaren `K`-garren posizioan kokaturik izan arren, hirugarren urrats bat behar da tartekaketa prozesua amaitutzat jo ahal izateko. Arrayaren luzera logikoa unitate batean inkrementatu beharra dago `N` aldagaian `N+1` balioa gordez.

Pseudokodea Hauek lirateke goiko algoritmoa burutzeko beharko liratekeen aldagaien identifikadoreak eta euren esanahia (gogoratu Z arrayaren luzera logikoa N dela eta luzera fisikoa MAX):

`Elementu` tartekatu nahi den balioa aldagai honetan gorderik dago eta tartekaketa prozesua hasi aurretik ezaguna den balioren bat izango du.

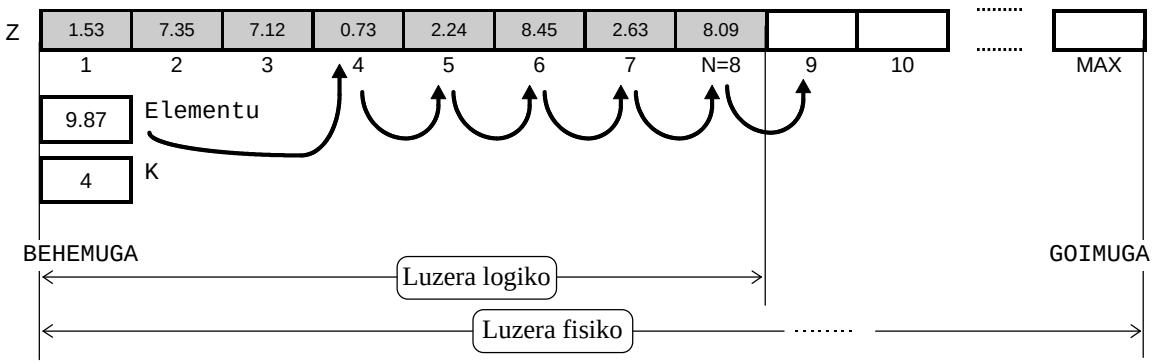
`K` aurreko `Elementu` aldagaiak duen balioa Z arrayaren zer posiziotan kokatu nahi den adierazten du. Argi dagoenez K-ren balioa Z arrayaren BEHEMUGA eta N luzera logikoaren artekoa izango da.

```
GOIMUGATIK Indize ← N BEHEMUGARAINO K EGIN
     Z[Indize + 1] ← Z[Indize]
AMAIA_GOIMUGATIK
Z[K] ← Elementu
N ← N + 1
```

Kodea

```
ReadLn (Elementu) ;              { Elementu teklatuz irakurtzen da }
REPEAT
   ReadLn (K) ;                  { K ere teklatuz irakurtzen da }
UNTIL (K >= BEHEMUGA) AND (K <= N) ;

FOR Indize:=N DOWNTO K DO
    Z[Indize+1] = Z[Indize] ;     { Z arrayan tokia egin }

Z[K] = Elementu ;                 { Elementu K-n tartekatu }

N := N + 1 ;                      { luzera logikoa inkrementatu }
```

Eskema eskema honetan luzera logikoa 8 dela suposatzen da (ikus jarraian datorren adibidea):



Adibidea array batean zenbaki errealak gorde izan dira, eta teklatuz beste kopuru erreal bat irakurri ondoren arrayaren posizio jakin batean tartekatu nahi da.

Array jakin batean elementu berri bat tartekatzean (edo txertatzean), lehenagotik dauden datuak zapaldu egiten direnez orain azalduko ditugun urratsak orden eta zuhurtziaz ez egin behar dira informazioa gal ez dadin, hots, algoritmoak zehazten dituen urratsak hurrenez hurren beteko dira aurretik daukagun informazioaren iraupena bermatu nahi badugu.

Arraya datuz bete ondoren elementu berria jasotzeko toki librerik duen ala ez ikertzen da. Baiezkoan, teklatuz bi irakurketa egingo dira, tartekatuko den zenbaki errealaren balioa eta non tartekatuko den balio hori (zenbaki berria `ElementuBerri` aldagaiaren bitartez irakurriko da, eta tartekatzeko posizioa aldiz `Non` bitartez).

```
PROGRAM TartekaketaArrayetan ;                    { \TP70\10\ALGOR_05.PAS }
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 20 ;
TYPE
  DM_Zerrenda = ARRAY[BEHEMUGA..GOIMUGA] OF Real ;

PROCEDURE ArrayaBete (VAR A : DM_Zerrenda; VAR Luzera : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   Randomize ;
   Luzera := Random(GOIMUGA) + 1 ;

   WriteLn ('A arrayan ', Luzera, ' datu gordetzen') ;
   FOR Indizea:=BEHEMUGA TO Luzera DO
      A[Indizea] := 10*Random ;      { 0.00 eta 9.99 arteko balioak }
END ;

PROCEDURE ArrayaIkusi (CONST A : DM_Zerrenda; Luzera : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   WriteLn ('A arrayaren edukia: ') ;
   FOR Indizea:=BEHEMUGA TO Luzera DO
   BEGIN
      Write (A[Indizea] :10:2) ;
   END ;
   WriteLn ;
END ;

PROCEDURE Tartekaketa (VAR Z : DM_Zerrenda; VAR N : Byte;
                       Elementu : Real; K : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   FOR Indizea:=N DOWNTO K DO
   BEGIN
      Z[Indizea+1] := Z[Indizea] ;
   END ;
   Z[K] := Elementu ;
   N := N + 1 ;
END ;

VAR
  A : DM_Zerrenda ;
  Luzera, Non : Byte ;
  ElementuBerri : Real ;
BEGIN
   ArrayaBete (A, Luzera) ;
   ArrayaIkusi (A, Luzera) ;

   IF Luzera < GOIMUGA THEN
   BEGIN
      Write ('Eman zenbaki berriaren balioa: ') ;
      ReadLn (ElementuBerri) ;
      REPEAT
         Write ('Zenbaki berriaren posizioa arrayan: ') ;
         ReadLn (Non) ;
      UNTIL (Non >= BEHEMUGA) AND (Non <= Luzera) ;

      Tartekaketa (A, Luzera, ElementuBerri, Non) ;
      ArrayaIkusi (A, Luzera) ;
   END ;
END.
```

Eskemarekin bat datorren programaren irteera hauxe da:

```
A arrayan 8 datu gordetzen
A arrayaren edukia:
    1.53    7.35    7.12    0.73    2.24    8.45    2.63    8.09
Eman zenbaki berriaren balioa: 9.87
Zenbaki berriaren posizioa arrayan: 4
A arrayaren edukia:
    1.53    7.35    7.12    9.87    0.73    2.24    8.45    2.63    8.09
_
```

10.4.4 Ezabaketa

Enuntziatua demagun Z arrayaren K-garren posizioan dagoen elementua ezabatu nahi dela. Arrayaren luzera logikoa N bada, ezabaketa ondoren Z-ren luzera efektibo hori N-1 izango da.

Ebazpena ebazpenak tartekaketan bezala hiru urrats ditu.

Lehenengo urratsan galduko den datuaren kopia bat egiten da. Suposatuz ezabatu nahi dugun arrayaren elementua aurrerago erabiliko dela, ezabatu baino lehen bere balioa beste aldagai laguntzaileren batean gordetzen da.

Ezaguna den K-garren posiziotik N-garren posizioraino (N arrayaren luzera logikoa litzateke) elementu guztiak ezkerrerantz posizio bat desplazatu. Bigarren urrats honen bitartez arrayaren K-garren posizioko datua ezabatzen da, eta lan hori gauzatzeko kontrol-egiturarik aproposena FOR-TO-DO denez ez lirateke WHILE-DO edo REPEAT-UNTIL sententziak erabili beharko.

Ezabaketa borobiltzeko hirugarren urrats bat behar da. Desplazamenduak direla eta, arrayaren N-garren elementua eta bere aurreko N-1 posizioko elementua bikoiztuta daude, horren zuzenketa arrayaren luzera logikoa unitate batean dekrementatzean lortzen da. Ezabaketa deitzen den eragiketa ostean arrayaren elementu kopurua bat gutxiago denez, luzera logikoa zehazten duen N aldagaian N-1 balioa gordetzen da algoritmoaren hirugarren urrats honetan.

Pseudokodea Hauek lirateke goiko algoritmoa burutzeko beharko liratekeen aldagaien identifikadoreak eta euren esanahia (gogoratu berriro ere Z arrayaren luzera logikoa N dela eta luzera fisikoa MAX):

Kopia ezabatu nahi den balioa aldagai honetan gorde daiteke baldin eta aurrerago datu horren beharra arrikusten bada. Askotan lehendabiziko urrats hau ez da behar izaten.

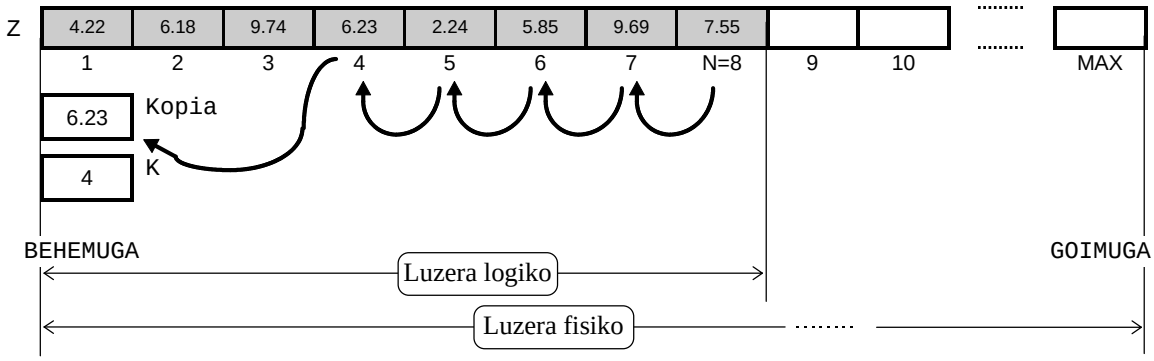
K ezabatuko den datua Z arrayaren zer posiziotan kokaturik dagoen aldagai honek adierazten du. Argi dagoenez K-ren balioa Z arrayaren BEHEMUGA eta N luzera logikoaren artekoa izango da.

```
Kopia ← Z[K]
BEHEMUGATIK Indize ← K GOIMUGARAINO N EGIN
     Z[Indize] ← Z[Indize + 1]
AMAIA_BEHEMUGATIK
N ← N- 1
```


Kodea

```
REPEAT
   ReadLn (K) ;                      { K teklatuz irakurtzen da }
UNTIL (K >= BEHEMUGA) AND (K <= N) ;

Kopia := Z[K] ;                      { datuaren kopia bat egin }

FOR Indize:=K TO N DO
    Z[Indize] = Z[Indize+1] ;        { elementuak desplazatu }

N := N - 1 ;                         { luzera logikoa dekrementatu }
```

Eskema eskema honetan luzera logikoa 8 dela suposatzen da (ikus jarraian datorren adibidea):



Adibidea array batean zenbaki errealak gorde izan dira, eta datu bati dagokion posizioa teklatuz eman ondoren (kopuru oso bat), indize horrek identifikatzen duen elementua arrayetik ezabatu nahi da.

Arraya datuz bete ondoren, ezabatu nahi den elementuaren indizea teklatuz irakurtzen da. Ezabaketa burutu baino lehen `ElemKopia` izeneko aldagai laguntzaile batean desagertuko den datua gorde behar da (bere balioa programaren azken sententzian pantailaratzen baita).

```
PROGRAM EzabaketaArrayetan ;                    { \TP70\10\ALGOR_06.PAS }
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 20 ;
TYPE
  DM_Zerrenda = ARRAY[BEHEMUGA..GOIMUGA] OF Real ;

PROCEDURE ArrayaBete (VAR A : DM_Zerrenda; VAR Luzera : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   Randomize ;
   Luzera := Random(GOIMUGA) + 1 ;

   WriteLn ('A arrayan ', Luzera, ' datu gordetzen') ;
   FOR Indizea:=BEHEMUGA TO Luzera DO
   BEGIN
      A[Indizea] := 10*Random ;      { 0.00 eta 9.99 arteko balioak }
   END ;
END ;
```

```
PROCEDURE ArrayaIkusi (CONST A : DM_Zerrenda; Luzera : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   WriteLn ('A arrayaren edukia: ') ;
   FOR Indizea:=BEHEMUGA TO Luzera DO
   BEGIN
      Write (A[Indizea] :8:2) ;
   END ;
   WriteLn ;
END ;

PROCEDURE Ezabaketa (VAR Z : DM_Zerrenda; VAR N : Byte; K : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   FOR Indizea:=K TO N DO
   BEGIN
      Z[Indizea] := Z[Indizea+1] ;
   END ;
   N := N - 1 ;
END ;

VAR
  A : DM_Zerrenda ;
  Luzera, Non : Byte ;
  ElemKopia : Real ;
BEGIN
   ArrayaBete (A, Luzera) ;
   ArrayaIkusi (A, Luzera) ;

   REPEAT
      Write ('Ezabatuko den zenbakiaren posizioa arrayan: ') ;
      ReadLn (Non) ;
   UNTIL (Non >= BEHEMUGA) AND (Non <= Luzera) ;

   ElemKopia := A[Non] ;        { ezabatuko den datuaren kopia egin }

   Ezabaketa (A, Luzera, Non) ;

   ArrayaIkusi (A, Luzera) ;
   WriteLn ('Hona hemen A-tik ezabatu den zenbakia: ', ElemKopia:0:2) ;
END.
```

Eskemarekin bat datorren programaren irteera hauxe da:

```
A arrayan 8 datu gordetzen
A arrayaren edukia:
    4.22    6.18    9.74    6.23    2.24    5.85    9.69    7.55
Ezabatuko den zenbakiaren posizioa arrayan: 4
A arrayaren edukia:
    4.22    6.18    9.74    2.24    5.85    9.69    7.55
Hona hemen A-tik ezabatu den zenbakia: 6.23
_
```

10.4.5 Nahasketa

Enuntziatua demagun oinarrizko datu-mota bera daukaten Z1 eta Z2 arrayak ditugula, eta biren nahasketa izango den Z arraya lortu nahi dugula. Balizko nahasketak asko izan daitezkeenez programatuko duguna zehazki definitu beharra dago: Z-ren osagaiak Z1 eta Z2 arrayen elementuak izango dira baina alternatiboki harturik (bikoteka eramango dira Z1 eta Z2 arrayen

elementuak emaitza den Z-ra, eta beti Z1 arrayen elementua Z2-ren elementua baino aurrerago kokatuko da Z-an). Datuak diren Z1 eta Z2 arrayaren luzera logikoak N1 eta N2 izanik Z-ren luzera efektiboak N1+N2 balioko du[20].

Ebazpena ebazpenak ibilera sofistikatu bat dirudi, Z1 eta Z2 arrayen luzera logikoak desberdinak izan daitezkeenez ibilera bi tartetan burutzen da:

- Luzera logiko biren artetik txikienaren balioa aintzat harturik Z1 eta Z2 arrayen elementuak paraleloki tratatzen dira
- Bi arrayentzat amankomuna den tartea landu ondoren, luzeena den arrayaren hondarra geratzen da. Beraz, array luzeenaren azken elementuak sekuentzialki Z-ra eramango dira

Nahasketa bukatzeko Z-ren luzera logikoa zehaztuko da, azkenean emaitza arrayaren luzera Z1 eta Z2 arrayen luzera logikoen batura izango da.

Pseudokodea Hauek lirateke goiko algoritmoa burutzeko beharko liratekeen aldagaien identifikadoreak eta euren esanahia (Z emaitza arrayaren luzera logikoa N izanik):

Z1 eta N1 lehen arrayaren izena eta luzera logikoa. Hau da, Z1 arrayak N1 elementu gordetzen ditu.

Z2 eta N2 bigarren array hau datuz horniturik aurkituko da nahasketa prozesua hasi aurretik, konkretuki Z2 arrayan N2 elementu gorde izan dira.

Non kopuru osoak biltegitzen dituen aldagai honen zeregina, uneoro, Z arrayaren zer posiziotan kokatuko den hurrengo datua zehaztea da. Algoritmoan argi ikusten denez nahasketa amaitzean Non aldagaiaren balioa N1+N2 izango da.

```
Z1, N1, Z2, N2 aldagaiek balioak dituzte
Non ← BEHEMUGA
BALDIN-ETA N1 > N2 ORDUAN
    BEHEMUGATIK Indize ← BEHEMUGA GOIMUGARAINO N2 EGIN
        Z[Non] ← Z1[Indize]
        Z[Non+1] ← Z2[Indize]
        Non ← Non+2
    AMAIA_BEHEMUGATIK
    BEHEMUGATIK Indize ← N2+1 GOIMUGARAINO N1 EGIN
        Z[Non] ← Z1[Indize]
        Non ← Non+1
    AMAIA_BEHEMUGATIK
BESTELA
    BEHEMUGATIK Indize ← BEHEMUGA GOIMUGARAINO N1 EGIN
        Z[Non] ← Z1[Indize]
        Z[Non+1] ← Z2[Indize]
        Non ← Non+2
    AMAIA_BEHEMUGATIK
```

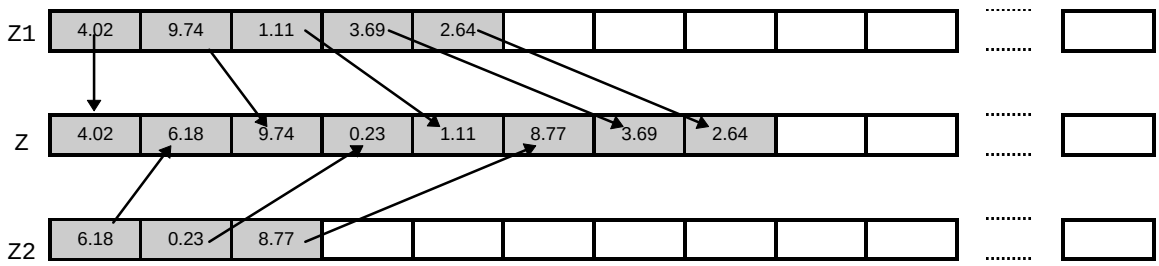
[20] Emaitza den Z arrayaren luzera logikoa N deituko dugu, GOIMUGA baino txikiagoa izanik, bere balioa N1+N2 batura izango dago da.

```
        BEHEMUGATIK Indize ← N1+1 GOIMUGARAINO N2 EGIN
            Z[Non] ← Z2[Indize]
            Non ← Non+1
        AMAIA_BEHEMUGATIK
    AMAIA_BALDIN-ETA
    N ← Non-1                   { Z-ren luzera logikoa  }
                                { N1+N2 jar daiteke ere }
```

Kodea

```
ArrayaBete (Z1, N1) ;
ArrayaBete (Z2, N2) ;
Non := BEHEMUGA ;
IF N1 > N2 THEN
BEGIN
   FOR Indize:=BEHEMUGA TO N2 DO
   BEGIN
      Z[Non] = Z1[Indize] ;          { lehen Z1-en elementua }
      Z[Non+1] = Z2[Indize] ;        { gero Z2-ren elementua }
      Non := Non+2 ;
   END ;
   FOR Indize:=N2+1 TO N1 DO
   BEGIN
      Z[Non] = Z1[Indize] ;          { Z1 arrayaren hondarra }
      Non := Non+1 ;
   END ;
END
ELSE
BEGIN
   FOR Indize:=BEHEMUGA TO N1 DO
   BEGIN
      Z[Non] = Z1[Indize] ;          { lehen Z1-en elementua }
      Z[Non+1] = Z2[Indize] ;        { gero Z2-ren elementua }
      Non := Non+2 ;
   END ;
   FOR Indize:=N1+1 TO N2 DO
   BEGIN
      Z[Non] = Z2[Indize] ;          { Z2 arrayaren hondarra }
      Non := Non+1 ;
   END ;
END ;
N := Non-1 ;                        { Z-ren luzera logikoa }
```

Eskema eskema honetan `Z1` eta `Z2` arrayen luzera logikoak 5 eta 3, hurrenez hurren, direla suposatzen da (ikus jarraian datorren adibidea):



Adibidea bi arrayetan zenbaki errealak gorde izan dira, euren balioak beste array batera eramanez nahastuko dira. Abiapuntuko arrayei `Random` funtzio estandarraz balioak ematen zaizkie, eginkizun hau errepikatuko da lortzen diren luzera logiko biren baturak `GOIMUGA` gainditzen ez duen arte.

Arrayak datuz bete ondoren nahasketa hasten da, prozesu honetan Z emaitza arrayak non jasoko duen hurrengo elementua Non izeneko aldagai laguntzaileak kontrolatzen du. Nahasketa prozesua ez da bukatutzat joko Z-ren luzera logikoa zehaztu gabe.

```
PROGRAM NahasketaArrayetan ;                          { \TP70\10\ALGOR_07.PAS }
USES
  Crt ;
CONST
  BEHEMUGA = 1 ;
  GOIMUGA = 20 ;
TYPE
  DM_Zerrenda = ARRAY[BEHEMUGA..GOIMUGA] OF Real ;

PROCEDURE ArrayaBete (VAR A : DM_Zerrenda; VAR Luzera : Byte; Zein : String) ;
VAR
  Indizea : Byte ;
BEGIN
   Luzera := Random(GOIMUGA) + 1 ;

   WriteLn (Zein, ' arrayan ', Luzera, ' datu gordetzen') ;

   FOR Indizea:=BEHEMUGA TO Luzera DO
   BEGIN
      A[Indizea] := 10*Random ;      { 0.00 eta 9.99 arteko balioak }
   END ;
END ;

PROCEDURE ArrayaIkusi (CONST A : DM_Zerrenda; Luzera : Byte; Zein : String) ;
VAR
  Indizea : Byte ;
BEGIN
   WriteLn (Zein, ' arrayaren edukia: ') ;
   FOR Indizea:=BEHEMUGA TO Luzera DO
   BEGIN
      Write (A[Indizea] :8:2) ;
   END ;
   WriteLn ;
END ;

PROCEDURE Nahasketa (CONST Z1 : DM_Zerrenda; N1 : Byte;
                     CONST Z2 : DM_Zerrenda; N2 : Byte;
                     VAR Z : DM_Zerrenda; VAR N : Byte) ;
VAR
  Indizea, Non : Byte ;
BEGIN
   Non := BEHEMUGA ;
   IF N1 > N2 THEN
   BEGIN
      FOR Indizea:=1 TO N2 DO
      BEGIN
         Z[Non] := Z1[Indizea] ;        { lehen Z1-en elementua }
         Z[Non+1] := Z2[Indizea] ;      { gero Z2-ren elementua }
         Non := Non + 2 ;
      END ;
      FOR Indizea:=N2+1 TO N1 DO
      BEGIN
         Z[Non] := Z1[Indizea] ;        { Z1 arrayaren hondarra }
         Non := Non + 1 ;
      END ;
   END
   ELSE
   BEGIN
      FOR Indizea:=1 TO N1 DO
      BEGIN
         Z[Non] := Z1[Indizea] ;        { lehen Z1-en elementua }
         Z[Non+1] := Z2[Indizea] ;      { gero Z2-ren elementua }
         Non := Non + 2 ;
      END ;
```

```
      FOR Indizea:=N1+1 TO N2 DO
      BEGIN
         Z[Non] := Z2[Indizea] ;        { Z2 arrayaren hondarra }
         Non := Non + 1 ;
      END ;
   END ;

   N := Non - 1 ;                       { Z-ren luzera logikoa }
END ;

VAR
  A1, A2, Emaitza : DM_Zerrenda ;
  Luz1, Luz2, Luzera : Byte ;
  Zein : String ;
BEGIN
   ClrScr ;
   Randomize ;
   REPEAT
      Zein := 'Aurreneko' ;
      ArrayaBete (A1, Luz1, Zein) ;
      ArrayaIkusi (A1, Luz1, Zein) ;
      Zein := 'Bigarren' ;
      ArrayaBete (A2, Luz2, Zein) ;
      ArrayaIkusi (A2, Luz2, Zein) ;
   UNTIL Luz1+Luz2 <= GOIMUGA ;

   Nahasketa (A1, Luz1, A2, Luz2, Emaitza, Luzera) ;
   WriteLn ;
   Zein := 'Emaitza' ;
   ArrayaIkusi (Emaitza, Luzera, Zein) ;
END.
```

Eskemarekin bat datorren `NahasketaArrayetan` programaren irteera hauxe da:

```
Aurreneko arrayan 5 datu gordetzen
Aurreneko arrayaren edukia:
    4.02    9.74    1.11    3.69    2.64
Bigarren arrayan 3 datu gordetzen
Bigarren arrayaren edukia:
    6.18    0.23    8.77

Emaitza arrayaren edukia:
    4.02    6.18    9.74    0.23    1.11    8.77    3.69    2.64
_
```

10.4.6 Ordenazioa

Array dimentsiobakar izanik aldagai egituratu horrek biltegitzen dituen elementuak ordenatu behar ditu. Demagun gure arrayak, 0-tik 9-ra bitarteko alde osoa eta dezimal bakarra duten zenbaki errealak gordetzen dituela eta ordenaziorako irizpidea txikitik handira dela, hots, arrayaren elementurik txikiena lehendabiziko posizioan jarri, hurrengo txikiena bigarrenean, ... , eta handiena arrayaren azken[21] posizioan kokatu.

Ikasiko ditugun ordenazio algoritmoak bost izango dira:

1. Aukeraketa metodoaren bitartez
2. Tartekaketaren bitartez (egin behar den bilaketa lineala izanik)
3. Tartekaketaren bitartez (egin behar den bilaketa bitarra izanik)
4. Binakako trukaketaren bitartez
5. Binakako trukaketaren bitartez (amaitzeko testigantzarekin)

[21] Arrayaren azken posizioa, nola ez, luzera logikoak markatzen duena da.

Adibideetarako erabiliko dugun arrayaren ezaugarriak hauek dira:

```
CONST
    MAX = 15 ;
TYPE
    DM_Zerrenda = ARRAY [1..MAX] OF Real ;
```

Non arrayaren elementuak zenbaki errealen azpimultzo bat diren (denak positiboak, alde osoa 0-tik 9 bitartekoa eta alde zatigarria dezimal bakar batekin).

10.4.6.1 Ordenazioa aukeraketaren bitartez

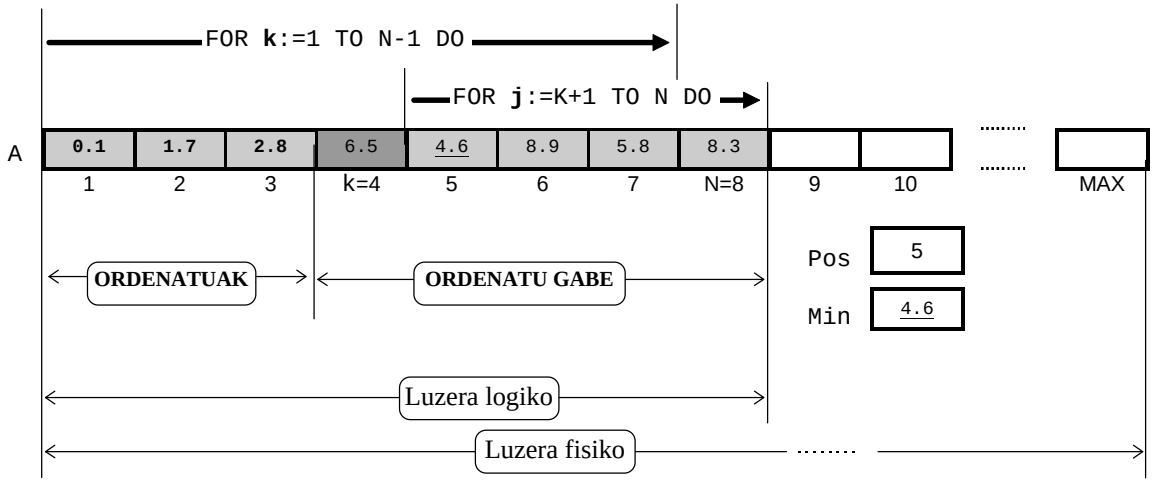
Hona hemen zenbaki errealen array bat ordenatzen duen `OrdenaAukeraketa` izeneko programaren sententziak:

```
(* Array baten ordenazioa AUKERAKETA bitartez *)
PROGRAM OrdenaAukeraketa ;                          { \TP70\10\ORDEN_01.PAS }
USES
    Crt ;
CONST
    MAX = 15 ;
TYPE
    DM_Zerrenda = ARRAY [1..MAX] OF Real ;

(* Alde osorik ez duen zenbaki erreal bat hartu eta alde *)
(* osoan digitu bakarra eta dezimal bakarra duen zenbaki *)
(* erreala itzultzen duen funtzioa:   0.12345 ---> 1.2   *)
FUNCTION Moldatu (Datua : Real) : Real ;
BEGIN
   Datua := Datua * 100 ;
   Datua := Int (Datua) ;
   Moldatu := Datua / 10 ;
END ;

(* Lehenengo parametroa array bat delako eta irteerakoa *)
(* delako erreferentziaz pasatzen da.  N erreferentziaz *)
(* ere prozeduran balioa hartu eta irteerakoa delako    *)
PROCEDURE ArrayaBete (VAR A : DM_Zerrenda; VAR Luzera : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   Randomize ;
   Luzera := Random(MAX) + 1 ;
   WriteLn ('A arrayan ', Luzera, ' datu gordetzen') ;
   FOR Indizea:=1 TO Luzera DO
   BEGIN
      A[Indizea] := Random ;      { 0.0000 eta 0.9999 arteko balioak }
      A[Indizea] := Moldatu (A[Indizea]) ;    { 0.0 eta 9.9 artekoak }
   END ;
END ;

(* Nahiz eta parametro biak sarrerakoak izan, lehenengoa  *)
(* array bat delako eta memorian kopia berriak ekiditeko  *)
(* erreferentziaz pasatzen da.  Bigarrena berriz balioz   *)
PROCEDURE ArrayaIkusi (CONST A : DM_Zerrenda; N : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   WriteLn ('A arrayaren edukia: ') ;
   FOR Indizea:=1 TO N DO
   BEGIN
      Write (A[Indizea] :8:1) ;
   END ;
   WriteLn ;
END ;
```

```
(* 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 AukeraketazOrdenatu (VAR A : DM_Zerrenda; N : Byte) ;
VAR
   Pos, j, k : Byte ;
   Min : Real ;
BEGIN
     FOR k:=1 TO N-1 DO
     BEGIN
        Min := A[k] ;
        Pos := k ;
        FOR j:=k+1 TO N DO               (* ordenatu gabekoen artean *)
        BEGIN
           IF Min > A[j] THEN            (* minimoa aurkitu *)
           BEGIN
              Min := A[j] ;
              Pos := j ;
           END ;
        END ;
        A[Pos] := A[k] ;                 (* trukatu, tokiz aldatuz  *)
        A[k] := Min ;
        Write (k, '. iterazioan, ') ;    (* ordenatzeko algoritmoak *)
        ArrayaIkusi (A, N) ;             (* darabilen logika ikertu *)
     END;
END ;

VAR
    A : DM_Zerrenda ;
    Luzera : Byte ;
BEGIN
   ClrScr ;
   ArrayaBete (A, Luzera) ;
   Write ('Ordenatu baino lehen ') ;
   ArrayaIkusi (A, Luzera) ;

   WriteLn ('-------------------------------------------------------') ;
   AukeraketazOrdenatu (A, Luzera) ;
   WriteLn ('-------------------------------------------------------') ;

   Write ('Ordenaturik, ') ;
   ArrayaIkusi (A, Luzera) ;
END.
```

Hona hemen `OrdenaAukeraketa` programaren balizko irteera bat eta bere eskema:

```
A arrayan 8 datu gordetzen
Ordenatu baino lehen A arrayaren edukia:
     6.5     8.3     5.8     0.1     4.6     8.9     2.8     1.7
-------------------------------------------------------
1. iterazioan, A arrayaren edukia:
     0.1     8.3     5.8     6.5     4.6     8.9     2.8     1.7
2. iterazioan, A arrayaren edukia:
     0.1     1.7     5.8     6.5     4.6     8.9     2.8     8.3
3. iterazioan, A arrayaren edukia:
     0.1     1.7     2.8     6.5     4.6     8.9     5.8     8.3
4. iterazioan, A arrayaren edukia:
     0.1     1.7     2.8     4.6     6.5     8.9     5.8     8.3
5. iterazioan, A arrayaren edukia:
     0.1     1.7     2.8     4.6     5.8     8.9     6.5     8.3
6. iterazioan, A arrayaren edukia:
     0.1     1.7     2.8     4.6     5.8     6.5     8.9     8.3
7. iterazioan, A arrayaren edukia:
     0.1     1.7     2.8     4.6     5.8     6.5     8.3     8.9
-------------------------------------------------------
Ordenaturik, A arrayaren edukia:
     0.1     1.7     2.8     4.6     5.8     6.5     8.3     8.9
_
```



Eskeman ikus daitekeenez `k` kontagailuaren laugarren iterazioan aurkitzen gara, ordurako arrayaren lehenengo hiru elementuak zehaztu dira eta iterazio honetan aurkituko den minimoa ilunago den gelaskan jarriko da (`k=4` posizioan alegia). Laugarren iterazioko une horretan minimoa bilatzeko ordenatu gabeko aldean begiratu beharra dago, eta horretarako `j` kontagailua erabiliko da.

`k=4` iterazioari dagokion minimoa zehazteko `Min` eta `Pos` aldagai laguntzaileak erabiltzen dira. Hasiera batean `Min` aldagaian ordenatu gabeko tartearen lehen elementuaren balioa gordetzen da, hots, `Min` aldagaian `A[k]` elemnetuaren balioa gordetzen da, eta `Pos` aldagaian horri dagokion `k` posizioa. Hasieraketak egin eta gero, barneko `FOR-DO` bigiztan ordenatu gabeko aldea aztertzen da elementurik txikiena zehaztuz (bere balioa eta posizioa zehaztuz). Laugarren iterazioan minimoa zein den jakitean, array ordenatuan dagokion tokira (`k` posiziora) eraman behar da, horregatik `k` eta `Pos` indizedun elementuak trukatu egiten dira.

Hurrengo iteraziorako, kanporago dagoen `FOR-DO` bigiztak automatikoki inkrementatuko du `k` aldagaia, eta horrekin batera arrayaren zati ordenatua emendaturik suertatzen da eta ordenatu gabea, aldiz, murriztua. Hau da, bosgarren iterazioan arrayaren zati ordenatua `k=5` izango da eta ordenatzeke geratzen dena `3` elementuz osaturik egongo da.

Ikusten denez `k` kontagailua behemugatik `N-1` bitartera aldatzen da, zergatik ez da luzera logiko guztira hedatzen?.

Laburbilduz, arraya ordenatzean aukeraketaren metodoaren bitartez zailtasuna une edo iterazio bati dagokion minimoa zehaztean datza, izan ere ezaguna[22] baita minimo hori non txertatu behar den (ikus eskemaren gelaskarik ilunena).

10.4.6.2 Ordenazioa tartekaketaren bitartez (bilaketa lineala)

Aurreko metodoan zailtasuna elementu minimoa zehaztean baldin badago eta non txertatu ezaguna bada, ordenazioa tartekaketaren bitartez erabat alderantzizkoa da. Izan ere, zein elementu txertatuko dugun iterazio bakoitzeko ezaguna izango da, baina non tartekatu behar den kalkulatu beharra dago. Eta horretarako bilaketa bat burutuko da, bilaketa sekuentziala **10.4.6.2** puntu honetan eta bilaketa bitarra hurrengo **10.4.6.3** puntuan.

Besterik gabe, hona hemen zenbaki errealen array bat, tartekaketaren bitartez ordenatzen duen `OrdenaTartekaketaSekuentzialki` izeneko programa:

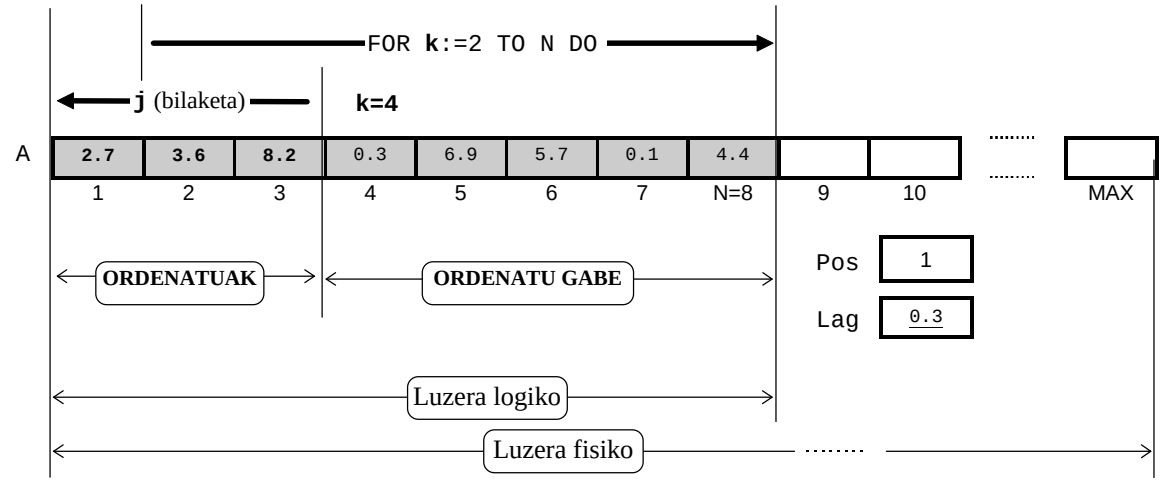
[22] Zati ordenatuaren eta ez-ordenatuaren mugan jarri behar da uneko minimoa.

```
(* Array baten ordenazioa TARTEKAKETA bitartez *)
(*     eta BILAKETA SEKUENTZIALA erabiliz      *)
PROGRAM OrdenaTartekaketaSekuentzialki ;          { \TP70\10\ORDEN_02.PAS }
USES
    Crt ;
CONST
    MAX = 15 ;
TYPE
    DM_Zerrenda = ARRAY [1..MAX] OF Real ;

(* Alde osorik ez duen zenbaki erreal bat hartu eta alde *)
(* osoan digitu bakarra eta dezimal bakarra duen zenbaki *)
(* erreala itzultzen duen funtzioa:   0.12345 ---> 1.2   *)
FUNCTION Moldatu (Datua : Real) : Real ;
BEGIN
   Datua := Datua * 100 ;
   Datua := Int (Datua) ;
   Moldatu := Datua / 10 ;
END ;

(* Lehenengo parametroa array bat delako eta irteerakoa *)
(* delako erreferentziaz pasatzen da.  N erreferentziaz *)
(* ere prozeduran balioa hartu eta irteerakoa delako    *)
PROCEDURE ArrayaBete (VAR A : DM_Zerrenda; VAR Luzera : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   Randomize ;
   Luzera := Random(MAX) + 1 ;
   WriteLn ('A arrayan ', Luzera, ' datu gordetzen') ;
   FOR Indizea:=1 TO Luzera DO
   BEGIN
      A[Indizea] := Random ;      { 0.0000 eta 0.9999 arteko balioak }
      A[Indizea] := Moldatu (A[Indizea]) ;    { 0.0 eta 9.9 artekoak }
   END ;
END ;

(* Nahiz eta parametro biak sarrerakoak izan, lehenengoa  *)
(* array bat delako eta memorian kopia berriak ekiditeko  *)
(* erreferentziaz pasatzen da.  Bigarrena berriz balioz   *)
PROCEDURE ArrayaIkusi (CONST A : DM_Zerrenda; N : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   WriteLn ('A arrayaren edukia: ') ;
   FOR Indizea:=1 TO N DO
   BEGIN
      Write (A[Indizea] :8:1) ;
   END ;
   WriteLn ;
END ;

(* 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 TartekaketaSekuentzialazOrdenatu (VAR A : DM_Zerrenda; N : Byte) ;
VAR
   Pos, j, k : Byte ;
   Lag : Real ;
BEGIN
   FOR k:=2 TO N DO
   BEGIN
      Lag := A[k] ;
      j := k-1 ;
      WHILE (A[j] > Lag) AND (j >= 1) DO
      BEGIN                          (* tokiaren bilaketa sekuentziala,  *)
         A[j+1] := A[j] ;            (* bilaketarekin batera tartekatuko *)
         j := j - 1 ;                (* den balioarentzeat hutsunea egin *)
      END ;
```

```
      A[j+1] := Lag ;
      Write ('k=', k, ' denean, ') ;    (* ordenatzeko algoritmoak *)
      ArrayaIkusi (A, N) ;              (* darabilen logika ikertu *)
   END ;
END ;

VAR
  A : DM_Zerrenda ;
  Luzera : Byte ;
BEGIN
   ClrScr ;
   ArrayaBete (A, Luzera) ;

   Write ('Ordenatu baino lehen ') ;
   ArrayaIkusi (A, Luzera) ;

   WriteLn ('-----------------------------------------------------') ;
   TartekaketaSekuentzialazOrdenatu (A, Luzera) ;
   WriteLn ('-----------------------------------------------------') ;

   Write ('Ordenaturik, ') ;
   ArrayaIkusi (A, Luzera) ;
END.
```

Eta hau litzateke `OrdenaTartekaketaSekuentzialki` programa egikaritzean atera daiteken balizko irteera bat, eta horri dagokion eskema:

```
A arrayan 8 datu gordetzen
Ordenatu baino lehen A arrayaren edukia:
     8.2     3.6     2.7     0.3     6.9     5.7     0.1     4.4
-----------------------------------------------------------
K=2 denean, A arrayaren edukia:
     3.6     8.2     2.7     0.3     6.9     5.7     0.1     4.4
K=3 denean, A arrayaren edukia:
     2.7     3.6     8.2     0.3     6.9     5.7     0.1     4.4
K=4 denean, A arrayaren edukia:
     0.3     2.7     3.6     8.2     6.9     5.7     0.1     4.4
K=5 denean, A arrayaren edukia:
     0.3     2.7     3.6     6.9     8.2     5.7     0.1     4.4
K=6 denean, A arrayaren edukia:
     0.3     2.7     3.6     5.7     6.9     8.2     0.1     4.4
K=7 denean, A arrayaren edukia:
     0.1     0.3     2.7     3.6     5.7     6.9     8.2     4.4
K=8 denean, A arrayaren edukia:
     0.1     0.3     2.7     3.6     4.4     5.7     6.9     8.2
-----------------------------------------------------------
Ordenaturik, A arrayaren edukia:
     0.1     0.3     2.7     3.6     4.4     5.7     6.9     8.2
_
```



Eskeman ikus daitekeenez k kontagailuak lau balio duenean hirugarren iterazioan aurkitzen gara, ordurako arrayaren lehenengo hiru elementuak zehaztu dira eta iterazio honetan arrayaren zati ordenatuan txertatuko den elementua azpimarratuta dagoen A[k] izango da. Arazoa 0.3 balioari dagokion posizioa finkatzean datza, horretarako egiten da, hain zuzen ere, bilaketa sekuentziala j kontagailuaren bitartez.

j kontagailuak kontrolatzen duen WHILE-DO bigiztak bilaketa egiten duen bitartean, arrayaren elementuen desplazamenduak burutzen ditu ere, ondorioz hutsune bat sortzen da non k-garren elementua txertatuko den. Desplazamenduak egitean k=4 iterazioko 0.3 balioa gal ez dadin Lag aldagai laguntzailean pilatzen da.

k=4 iterazioko 0.3 balioari array ordenatuan dagokion posizioa 1 da zein Pos aldagaian gordetzen den. Iterazio honi loturik dauden bilaketa prozesua eta hutsune sortzea amaitzen direnean Pos indizeak adierazten duen posiziora Lag aldagai laguntzailearen 0.3 balioa ekarriko da.

Hurrengo iteraziorako, kanporago dagoen FOR-DO bigiztak automatikoki inkrementatuko du k aldagaia, eta horrekin batera arrayaren zati ordenatua emendaturik suertatzen da eta ordenatu gabea, aldiz, murriztua.

Ikusten denez k kontagailuak duen behemuga FOR-DO egituran 2 da, zergatik ez da kontagailu hori 1-etik hasten?.

10.4.6.3 Ordenazioa tartekaketaren bitartez (bilaketa bitarra)

10.4.6.2 puntuan esandako guztiek hemen ere balio dute, kontutan izanik bilaketa lineala izan ordez dikotomikoa dela. Izan ere, k kontagailuak zehazten duen balioari arrayaren zati ordenatuan dagokion posizio bilatzeko **10.4.2.2 Bilaketa bitarra** puntuan ikasitako algoritmoa aplika daiteke.

Hona hemen beraz, zenbaki errealen array bat, tartekaketaren bitartez, ordenatzen duen OrdenaTartekaketaBitarra izeneko programa:

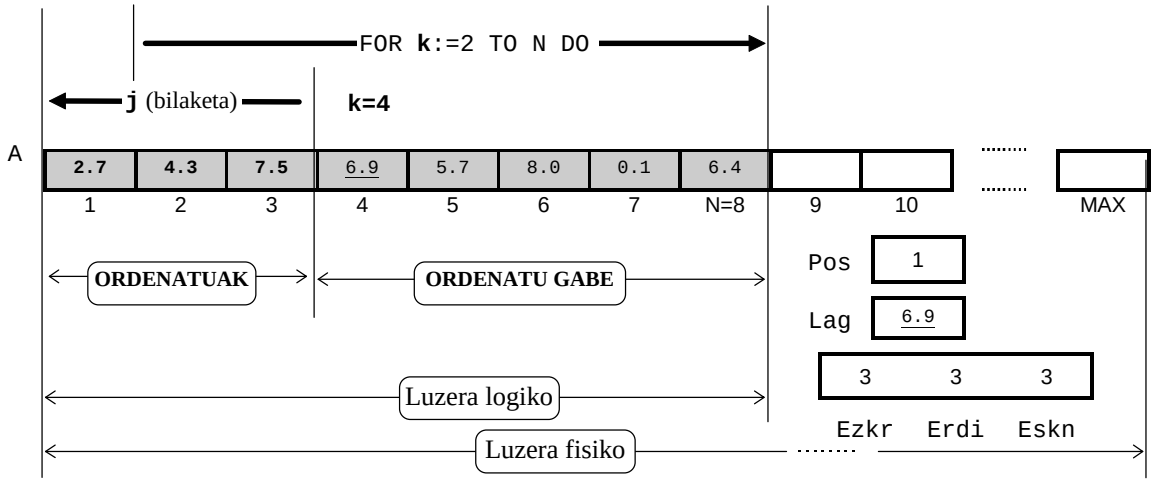
```
(* Array baten ordenazioa TARTEKAKETA bidez eta BILAKETA BITARRA erabiliz *)
PROGRAM OrdenaTartekaketaBitarra ;                { \TP70\10\ORDEN_03.PAS }
USES
    Crt ;
CONST
    MAX = 15 ;
TYPE
    DM_Zerrenda = ARRAY [1..MAX] OF Real ;

(* Alde osorik ez duen zenbaki erreal bat hartu eta alde *)
(* osoan digitu bakarra eta dezimal bakarra duen zenbaki *)
(* erreala itzultzen duen funtzioa:   0.12345 ---> 1.2   *)
FUNCTION Moldatu (Datua : Real) : Real ;
BEGIN
   Datua := Datua * 100 ;
   Datua := Int (Datua) ;
   Moldatu := Datua / 10 ;
END ;

(* Lehenengo parametroa array bat delako eta irteerakoa *)
(* delako erreferentziaz pasatzen da.  N erreferentziaz *)
(* ere prozeduran balioa hartu eta irteerakoa delako    *)
PROCEDURE ArrayaBete (VAR A : DM_Zerrenda; VAR Luzera : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   Randomize ;
   Luzera := Random(MAX) + 1 ;
```



```
   WriteLn ('A arrayan ', Luzera, ' datu gordetzen') ;
   FOR Indizea:=1 TO Luzera DO
   BEGIN
      A[Indizea] := Random ;      { 0.0000 eta 0.9999 arteko balioak }
      A[Indizea] := Moldatu (A[Indizea]) ;    { 0.0 eta 9.9 artekoak }
   END ;
END ;

(* Nahiz eta parametro biak sarrerakoak izan, lehenengoa  *)
(* array bat delako eta memorian kopia berriak ekiditeko  *)
(* erreferentziaz pasatzen da.  Bigarrena berriz balioz   *)
PROCEDURE ArrayaIkusi (CONST A : DM_Zerrenda; N : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   WriteLn ('A arrayaren edukia: ') ;
   FOR Indizea:=1 TO N DO
   BEGIN
      Write (A[Indizea] :8:1) ;
   END ;
   WriteLn ;
END ;

(* 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 TartekaketaBitarrazOrdenatu (VAR A : DM_Zerrenda; N : Byte) ;
VAR
   Pos, j, k, Ezkr, Eskn, Erdi : Byte ;
   Lag : Real ;
BEGIN
   FOR k:=2 TO N DO
   BEGIN
      Lag := A[k] ;
      Ezkr := 1 ;
      Eskn := k-1 ;
      Erdi := (Ezkr+Eskn) DIV 2 ;
      WHILE Ezkr <= Eskn DO             (* tokiaren bilaketa bitarra *)
      BEGIN
         IF Lag < A[Erdi] THEN
           Eskn := Erdi-1
         ELSE
           Ezkr := Erdi+1 ;
         Erdi := (Ezkr+Eskn) DIV 2 ;
      END ;
      FOR j:=k-1 DOWNTO Ezkr DO
      BEGIN
          A[j+1] := A[j] ;              (* hutsunea egin *)
      END ;
      A[Ezkr] := Lag;                   (* tartekaketa burutu *)

      Write ('k=', k, ' denean, ') ;    (* ordenatzeko algoritmoak *)
      ArrayaIkusi (A, N) ;              (* darabilen logika ikertu *)
   END ;
END ;

VAR
  A : DM_Zerrenda ;
  Luzera : Byte ;
BEGIN
   ClrScr ;
   ArrayaBete (A, Luzera) ;
   Write ('Ordenatu baino lehen ') ;
   ArrayaIkusi (A, Luzera) ;

   WriteLn ('-----------------------------------------------------') ;
   TartekaketaBitarrazOrdenatu (A, Luzera) ;
   WriteLn ('-----------------------------------------------------') ;

   Write ('Ordenaturik, ') ;
   ArrayaIkusi (A, Luzera) ;
END.
```

Eta hau litzateke `OrdenaTartekaketaBitarra` programa egikaritzean atera daiteken balizko irteera bat, eta horri dagokion eskema:

```
A arrayan 8 datu gordetzen
Ordenatu baino lehen A arrayaren edukia:
     7.5     2.7     4.3     6.9     5.7     8.0     0.1     6.4
-------------------------------------------------------------
K=2 denean, A arrayaren edukia:
     2.7     7.5     4.3     6.9     5.7     8.0     0.1     6.4
K=3 denean, A arrayaren edukia:
     2.7     4.3     7.5     6.9     5.7     8.0     0.1     6.4
K=4 denean, A arrayaren edukia:
     2.7     4.3     6.9     7.5     5.7     8.0     0.1     6.4
K=5 denean, A arrayaren edukia:
     2.7     4.3     5.7     6.9     7.5     8.0     0.1     6.4
K=6 denean, A arrayaren edukia:
     2.7     4.3     5.7     6.9     7.5     8.0     0.1     6.4
K=7 denean, A arrayaren edukia:
     0.1     2.7     4.3     5.7     6.9     7.5     8.0     6.4
K=8 denean, A arrayaren edukia:
     0.1     2.7     4.3     5.7     6.4     6.9     7.5     8.0
-------------------------------------------------------------
Ordenaturik, A arrayaren edukia:
     0.1     2.7     4.3     5.7     6.4     6.9     7.5     8.0
_
```



Eskeman ikus daitekeenez `k` kontagailuak lau balio duenean hirugarren iterazioan aurkitzen gara, ordurako arrayaren lehenengo hiru elementuak zehaztu dira eta iterazio honetan arrayaren zati ordenatuan txertatuko den elementua azpimarratuta dagoen `A[k]` izango da. Arazoa `6.9` balioari dagokion posizioa finkatzean datza, horretarako egiten da, hain zuzen ere, bilaketa bitarra `Ezkr`, `Erdi` eta `Eskn` aldagaien bitartez. Ezer baino lehen kopuru hori `Lag` aldagai laguntzailean pilatzen da.

Adibide horretan bilaketa bitarra bigarren saiakeran lortzen da, lehenengoan `Ezkr=1` eta `Eskn=3` beraz `Erdi=2`. Ondorioz `6.9` eta `4.3` kopuruak konparatzean ezkerreko erditartearekin geratuko gara (`Ezkr=3` eta `Eskn=3` beraz `Erdi=3`), hurrengo saiakeran `WHILE-DO` bigiztatik irtetea lortzen da, posizioaren indizea `Ezkr` delarik.

Elementua tartekatzeko tokia egin behar denez `j` kontagailuaren bitartez hutsunea eskuratu eta `Lag` aldagai laguntzailean gordetzen duena `Ezkr` indizeak adierazten duen posizioan gorde.

10.4.6.4 Ordenazioa burbuilaren bitartez

Burbuilaren metodoari trukaketa esaten zaio ere. Izan ere, arrayaren elementuak binaka hartuz ordenazioaren irizpidea zainduz euren balioak trukatu egiten direlako, elementu bikotea lantzean hurrenez hurren daudenak hartzen dira.

Hona hemen zenbaki errealen array bat, kopuru txikienetik handienera burbuilaren bitartez ordenatzen duen `OrdenaBurbuila` izeneko programa:

```
(* Array baten ordenazioa BURBUILA edo trukaketaren bitartez *)
PROGRAM OrdenaBurbuila ;                          { \TP70\10\ORDEN_04.PAS }
USES
    Crt ;
CONST
    MAX = 15 ;
TYPE
    DM_Zerrenda = ARRAY [1..MAX] OF Real ;

(* Alde osorik ez duen zenbaki erreal bat hartu eta alde *)
(* osoan digitu bakarra eta dezimal bakarra duen zenbaki *)
(* erreala itzultzen duen funtzioa:   0.12345 ---> 1.2   *)
FUNCTION Moldatu (Datua : Real) : Real ;
BEGIN
   Datua := Datua * 100 ;
   Datua := Int (Datua) ;
   Moldatu := Datua / 10 ;
END ;

(* Lehenengo parametroa array bat delako eta irteerakoa *)
(* delako erreferentziaz pasatzen da.  N erreferentziaz *)
(* ere prozeduran balioa hartu eta irteerakoa delako    *)
PROCEDURE ArrayaBete (VAR A : DM_Zerrenda; VAR Luzera : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   Randomize ;
   Luzera := Random(MAX) + 1 ;
   WriteLn ('A arrayan ', Luzera, ' datu gordetzen') ;
   FOR Indizea:=1 TO Luzera DO
   BEGIN
      A[Indizea] := Random ;      { 0.0000 eta 0.9999 arteko balioak }
      A[Indizea] := Moldatu (A[Indizea]) ;    { 0.0 eta 9.9 artekoak }
   END ;
END ;

(* Nahiz eta parametro biak sarrerakoak izan, lehenengoa  *)
(* array bat delako eta memorian kopia berriak ekiditeko  *)
(* erreferentziaz pasatzen da.  Bigarrena berriz balioz   *)
PROCEDURE ArrayaIkusi (CONST A : DM_Zerrenda; N : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   WriteLn ('A arrayaren edukia: ') ;
   FOR Indizea:=1 TO N DO
   BEGIN
      Write (A[Indizea] :8:1) ;
   END ;
   WriteLn ;
END ;

(* 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 TrukaketazOrdenatu (VAR A : DM_Zerrenda; N : Byte) ;
VAR
   j, k : Byte ;
   Lag : Real ;
```

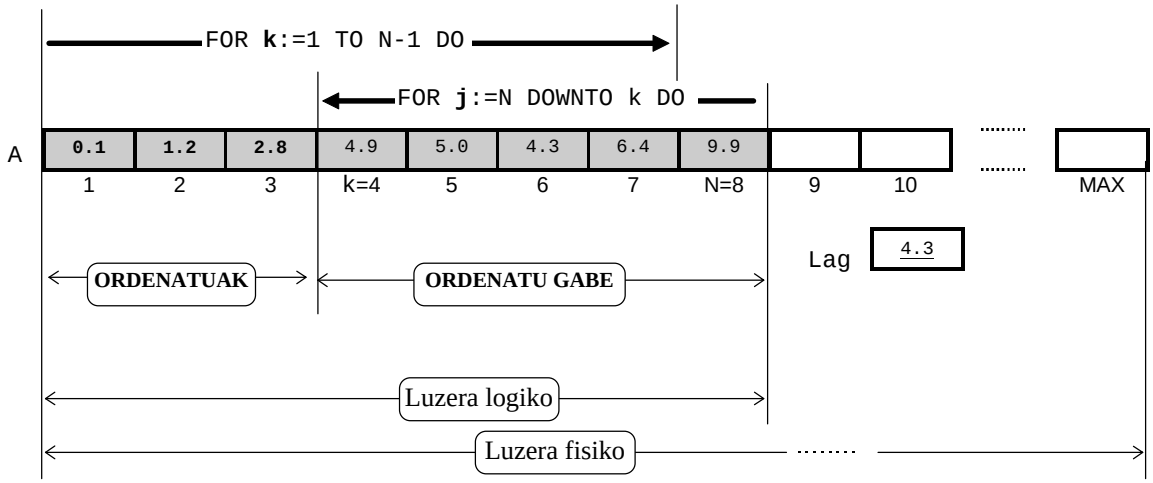
```
BEGIN
     FOR k:=1 TO N-1 DO
     BEGIN
        FOR j:=N DOWNTO k DO
        BEGIN
           IF A[j-1] > A[j] THEN          (* binakako konparaketa   *)
           BEGIN
              Lag := A[j-1] ;             (* binakako trukaketa Lag *)
              A[j-1] := A[j] ;            (* laguntzailearen bidez  *)
              A[j] := Lag ;
           END ;
        END ;
        Write (k, '. iterazioan, ') ;     (* ordenatzeko algoritmoak *)
        ArrayaIkusi (A, N) ;              (* darabilen logika ikertu *)
     END ;
END ;

VAR
    A : DM_Zerrenda ;
    Luzera : Byte ;
BEGIN
   ClrScr ;
   ArrayaBete (A, Luzera) ;

   Write ('Ordenatu baino lehen ') ;
   ArrayaIkusi (A, Luzera) ;

   WriteLn ('-------------------------------------------------------') ;
   TrukaketazOrdenatu (A, Luzera) ;
   WriteLn ('-------------------------------------------------------') ;

   Write ('Ordenaturik, ') ;
   ArrayaIkusi (A, Luzera) ;
END.
```

Hona hemen `OrdenaBurbuila` programaren balizko irteera bat, eta horri dagokion eskema:

```
A arrayan 8 datu gordetzen
Ordenatu baino lehen A arrayaren edukia:
     4.9     5.0     0.1     4.3     9.9     2.8     1.2     6.4
-------------------------------------------------------
1. iterazioan, A arrayaren edukia:
     0.1     4.9     5.0     1.2     4.3     9.9     2.8     6.4
2. iterazioan, A arrayaren edukia:
     0.1     1.2     4.9     5.0     2.8     4.3     9.9     6.4
3. iterazioan, A arrayaren edukia:
     0.1     1.2     2.8     4.9     5.0     4.3     6.4     9.9
4. iterazioan, A arrayaren edukia:
     0.1     1.2     2.8     4.3     4.9     5.0     6.4     9.9
5. iterazioan, A arrayaren edukia:
     0.1     1.2     2.8     4.3     4.9     5.0     6.4     9.9
6. iterazioan, A arrayaren edukia:
     0.1     1.2     2.8     4.3     4.9     5.0     6.4     9.9
7. iterazioan, A arrayaren edukia:
     0.1     1.2     2.8     4.3     4.9     5.0     6.4     9.9
-------------------------------------------------------
Ordenaturik, A arrayaren edukia:
     0.1     1.2     2.8     4.3     4.9     5.0     6.4     9.9
_
```

`A` arrayaren datuak direla eta, irteerarai so eginez laugarren iteraziorako `A` arraya ordenaturik dagela kontura gaitezke. Burbuilaren algoritmoak ez dirudi oso efizientea denik, halako egoera zuzendu nahirik hurrengo galderan burbuila hobetuaren algoritmoa ikasiko dugu.

Jarraian ematen den eskeman FOR-TO bigizta kabiatu biak agertzen dira, kanpoan dagoena FOR-TO bat da eta barnekoa, berriz, FOR-DOWN-TO egitura da.



Eskeman ikus daitekeenez k kontagailuaren laugarren iterazioan aurkitzen garenean j kontagailuak kontrolatzen duen barneko FOR-DOWNTO egitura binakako konparaketak egiten dira eta mugitzen den datu bakarra 4.3 balio duena da. Oraingoan ere k kontagailua behemugatik N-1 bitartera aldatzen da, zergatik ez da luzera logiko guztira hedatzen?.

Galderaren erantzunak algoritmoaren efizientziarekin zerikusia du. Hala eta guztiz programaren irteeran ikus daitekeenez, azken hiru iterazioak soberan daude datu konkretu horiek dituen arrayaren kasuan. Esan daiteke bi FOR dituen burbuilaren algoritmoak, orokorrean, hasieran elementu asko kokatzen ditu dagozkien tokietan baina amaieran soberazko iterazioak egin ditzakeela.

Laburbilduz, arraya ordenatzean burbuilaren metodoaren bitartez, iterazio bakoitzean, elementurik txikiena aurreraino eramaten dela bermatuta dago, baina horrekin batera zonalde desordenatuan beste txiki batzuek ere euren posizioak aurrera ditzakete.

10.4.6.5 Ordenazioa burbuila hobetuaren bitartez

Zenbait kasutan, eta datuen arabera, bi FOR kabiatu dituen burbuilaren algoritmoan azkeneko iterazioek ez dute lanik egiten, arrayaren ordenazioa k=N-1 baino lehenago lortu delako. Horregatik burbuilaren metodoari hobekuntza bat erants dakioke: kanpoko FOR-DO sententzia WHILE-DO edo REPEAT-UNTIL batez ordezkatu.

Jarraian erakusten dugun OrdenaBurbuilaHobetua programan baldintza bikoitza duen WHILE-DO sententzia jarri dugu. Baldintzaren bikoitzaren lehen atalari esker k aldagaiak, burbuila arruntaren kasuan bezala, 1-tik N-1 bitarteko balioak hartuko lituzke soberazko iteraziorik beharko ez lukeen kasuetan, noiz gertatzen da berezitasun hau?. Baldintzaren bikoitzaren bigarren atalari esker, barneko FOR-DOWNTO sententzian trukaketarik suertatzen ez denean (arraya dagoeneko ordenatuta dagoela baiezta daitekeenean) OrdenatuGabe aldagai boolearrak FALSE balioko du eta ez da iterazio gehiagorik burutuko.

```
(* Array baten ordenazioa BURBUILA HOBETUA edo trukaketaren bitartez *)
PROGRAM OrdenaBurbuilaHobetua ;              { \TP70\10\ORDEN_05.PAS }
USES
    Crt ;
CONST
    MAX = 15 ;
TYPE
    DM_Zerrenda = ARRAY [1..MAX] OF Real ;
```

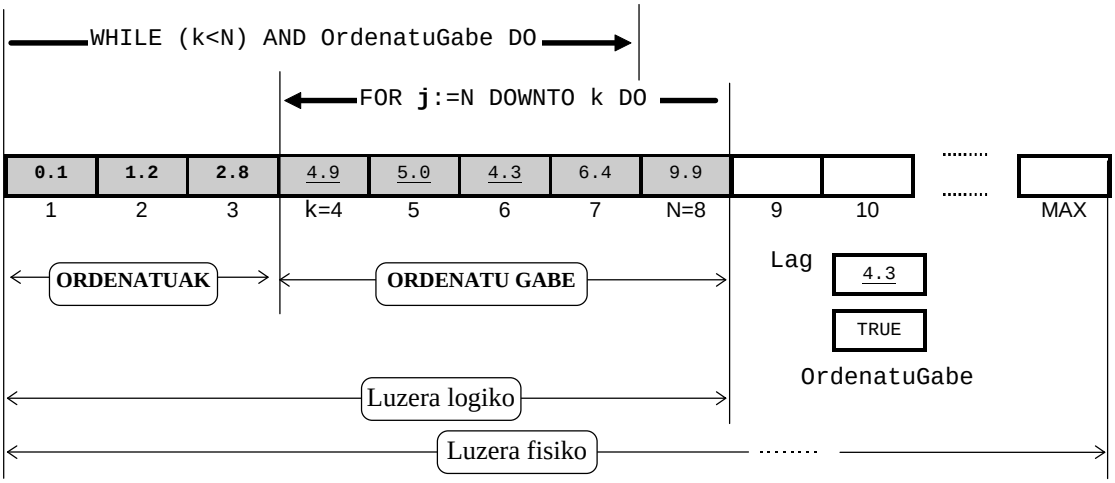
```
(* Alde osorik ez duen zenbaki erreal bat hartu eta alde *)
(* osoan digitu bakarra eta dezimal bakarra duen zenbaki *)
(* erreala itzultzen duen funtzioa:   0.12345 ---> 1.2   *)
FUNCTION Moldatu (Datua : Real) : Real ;
BEGIN
   Datua := Datua * 100 ;
   Datua := Int (Datua) ;
   Moldatu := Datua / 10 ;
END ;

(* Lehenengo parametroa array bat delako eta irteerakoa *)
(* delako erreferentziaz pasatzen da.  N erreferentziaz *)
(* ere prozeduran balioa hartu eta irteerakoa delako    *)
PROCEDURE ArrayaBete (VAR A : DM_Zerrenda; VAR Luzera : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   Randomize ;
   Luzera := Random(MAX) + 1 ;
   WriteLn ('A arrayan ', Luzera, ' datu gordetzen') ;
   FOR Indizea:=1 TO Luzera DO
   BEGIN
      A[Indizea] := Random ;      { 0.0000 eta 0.9999 arteko balioak }
      A[Indizea] := Moldatu (A[Indizea]) ;    { 0.0 eta 9.9 artekoak }
   END ;
END ;

(* Nahiz eta parametro biak sarrerakoak izan, lehenengoa  *)
(* array bat delako eta memorian kopia berriak ekiditeko  *)
(* erreferentziaz pasatzen da.  Bigarrena berriz balioz   *)
PROCEDURE ArrayaIkusi (CONST A : DM_Zerrenda; N : Byte) ;
VAR
  Indizea : Byte ;
BEGIN
   WriteLn ('A arrayaren edukia: ') ;
   FOR Indizea:=1 TO N DO
   BEGIN
      Write (A[Indizea] :8:1) ;
   END ;
   WriteLn ;
END ;

(* 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 ZelataridunTrukaketazOrdenatu (VAR A : DM_Zerrenda; N : Byte) ;
VAR
   j, k : Byte ;
   Lag : Real ;
   OrdenatuGabe : Boolean ;              (* aldagai zelataria *)
BEGIN
     k := 1 ;
     OrdenatuGabe := TRUE ;
     WHILE (k < N) AND (OrdenatuGabe = TRUE) DO
     BEGIN
        OrdenatuGabe := FALSE ;
        FOR j:=N DOWNTO k DO
        BEGIN
           IF A[j-1] > A[j] THEN          (* binakako konparaketa   *)
           BEGIN
              Lag := A[j-1] ;             (* binakako trukaketa Lag *)
              A[j-1] := A[j] ;            (* laguntzailearen bidez  *)
              A[j] := Lag ;
              OrdenatuGabe := TRUE ;
           END ;
        END ;
        k := k + 1 ;
        Write (k-1, '. iterazioan, ') ;   (* ordenatzeko algoritmoak *)
        ArrayaIkusi (A, N) ;              (* darabilen logika ikertu *)
     END ;
END ;
```

```
VAR
    A : DM_Zerrenda ;
    Luzera : Byte ;
BEGIN
   ClrScr ;
   ArrayaBete (A, Luzera) ;

   Write ('Ordenatu baino lehen ') ;
   ArrayaIkusi (A, Luzera) ;

   WriteLn ('-----------------------------------------------------------') ;
   ZelataridunTrukaketazOrdenatu (A, Luzera) ;
   WriteLn ('-----------------------------------------------------------') ;

   Write ('Ordenaturik, ') ;
   ArrayaIkusi (A, Luzera) ;
END.
```

Hona hemen **10.4.6.4** puntuan `OrdenaBurbuila` programari emandako datu berdinekin `OrdenaBurbuilaHobetua` irteera:

```
A arrayan 8 datu gordetzen
Ordenatu baino lehen A arrayaren edukia:
     4.9     5.0     0.1     4.3     9.9     2.8     1.2     6.4
-----------------------------------------------------------
1. iterazioan, A arrayaren edukia:
     0.1     4.9     5.0     1.2     4.3     9.9     2.8     6.4
2. iterazioan, A arrayaren edukia:
     0.1     1.2     4.9     5.0     2.8     4.3     9.9     6.4
3. iterazioan, A arrayaren edukia:
     0.1     1.2     2.8     4.9     5.0     4.3     6.4     9.9
4. iterazioan, A arrayaren edukia:
     0.1     1.2     2.8     4.3     4.9     5.0     6.4     9.9
5. iterazioan, A arrayaren edukia:
     0.1     1.2     2.8     4.3     4.9     5.0     6.4     9.9
-----------------------------------------------------------
Ordenaturik, A arrayaren edukia:
     0.1     1.2     2.8     4.3     4.9     5.0     6.4     9.9
_
```

`OrdenaBurbuilaHobetua` programaren eskema ez da aurrekoarekin konparatzean askorik aldatzen (kanpoko `FOR-DO` ordez `WHILE-DO` bat eta `OrdenatuGabe` zelataria):



10.5 ARRAYEN ERAGIKETA ARITMETIKOETARAKO UNITATEA

Zazpigarren kapituluan ikasitako unitateen kontzeptuak gogora ekarriz, array karratuen arteko eragiketa aritmetikoak burutzen lagunduko duen unitatea osa dezagun.

10.5.1 Array karratuen aritmetikarako unitatearen beharkizunak

Array karratuen aritmetika bideratuko duen unitatearen sorrera, nolabait, datu-mota berri bat eraiketarekin pareka daiteke, zenbaki errealen taulen 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, array karratuei aplikagarriak diren funtsezko eragiketa aritmetikoak (batuketa, kenketa, biderketa, zatiketa) `ArraErag` unitate berrian definiturik egongo dira. Bestalde, array karratuek onar dezaketen zenbait eragiketa espezifiko ere gehituko zaio unitateari.

Ordenadoreen baliabideak mugaturik daudelako `ArraErag` unitateak landuko dituen arrayak ere mugatu beharra dago. Gure adibiderako arayek izango duten itxura honako hau izango da:

```
CONST
   MAX = 8 ;
TYPE
   DM_Taula = ARRAY[1..MAX, 1..MAX] OF Real ;
```

Ikusten denez, arrayak kopuru errealen taula karratuak izango dira. Gehienez array batek `8x8` zenbaki erreal bil ditzake, baina une jakin batean arrayek izan dezaketen tamaina efektiboa `2` eta `8` bitartekoa izango da.

Lau eragiketa aritmetikoaz gain, array karratuek berezko dituzten operazioak onartuko dira `ArraErag` unitatearen bidez. Hots, operazio berezi hauek: arrayak datuz betetzea kopuruak teklatuz emanik, arrayen balioak taula bezala pantailaratzea, arrayaren array iraulia, sarrerako arrayaren array adjuntua, arrayaren determinantea eta arrayaren alderantzizkoa.

Zenbaki errealen array karratuak lantzen dituen unitatearen beharkizunak ondoko taulan biltzen dira, konturatu, praktikan, `ArraErag` unitatearen bitartez datu-mota berri bat eskaintzen zaiola bezero-programari:

`ArraErag` unitatearen espezifikazioa

Egitura: Kopuru errealen taula baten ideia ondoen isladatzen duen datu egiturarik, zenbaki errealen array bat litzateke. Arrayaren dimentsioak aldez aurretik zehaztu beharra dagoenez `MAX=8` konstante definitu da.

Eragiketak: Hauek dira unitateari loturik dauden operazioak:

- Array bati, teklatutik irakurririk, balioak eman
- Array baten osagaiak, taula bezala, pantailan agertu
- Sarrerako array bati dagokion array iraulia lortu
- Sarrerako array bati dagokion array adjuntua lortu
- Sarrerako array baten determinantea kalkulatu
- Sarrerako array bati dagokion alderantzizko arraya lortu

- Sarrerako bi arrayen batura kalkulatu
- Sarrerako bi arrayen kendura kalkulatu
- Sarrerako bi arrayen biderkadura kalkulatu
- Sarrerako bi arrayen zatidura kalkulatu

Zerrenda irakurtzetik ondoriotzen den bezala, ohizko lau eragiketa aritmetikoak arrayen artean burutzeko gainerako eragiketak inplementatu beharko dira. Honen zergatia hobeto ulertzearren ohizko lau eragiketa aritmetikoak defini ditzagun.

10.5.1.1 Batuketa

Demagun `T1` eta `T2` zenbaki errealen bi array ditugula, sarrerako taula biren dimentsio efektiboak berdinak izanik, taula horien batura `T3` beste array berri bat izango da. `T3` arrayaren osagaiek honela lortzen dira:

```
T3[i,j] = T1[i,j] + T2[i,j]
```

Lerro eta zutabe bakoitzeko, sarrerako balioen batura kalkulatuz erdiesten da `T3` arrayaren elementuak.

10.5.1.2 Kenketa

Era beretsuan, `T1` eta `T2` zenbaki errealen sarrerako taula biren kendura beste array karratu bat izango da. Sarreren dimentsio efektiboak berdinak izanik, taula horien batura `T3` beste array berri bat izango da. `T3` arrayaren osagaiek honela lortzen dira:

```
T3[i,j] = T1[i,j] - T2[i,j]
```

Lerro eta zutabe bakoitzeko, sarrerako balioen kendura kalkulatuz erdiesten da `T3` arrayaren elementuak.

10.5.1.3 Biderketa

Demagun `T1` eta `T2` zenbaki errealen bi array ditugula eta euren dimentsio efektiboak `3x3` direla, sarrerako taula bien balioak `1`-tik `9`-ra doazen kopuruak direla suposatuz, taula horien biderkadura `T3` beste array berri bat izango da. `T3` arrayaren osagaiek honela lortzen dira:

$$\begin{pmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \times \begin{pmatrix} \mathbf{1} & 2 & 3 \\ \mathbf{4} & 5 & 6 \\ \mathbf{7} & 8 & 9 \end{pmatrix} = \begin{pmatrix} \mathbf{1x1 + 2x4 + 3x7} & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{pmatrix}$$

`3x3` arrayen arteko biderkadura lortzeko honelako prozedura idaz genezake (ikus `ArrayenAlderantzizkoa` izeneko programa):

```
PROCEDURE Biderketa_3x3 (CONST T1,T2:DM_Taula3; VAR T3:DM_Taula3) ;
BEGIN
   T3[1,1] := T1[1,1]*T2[1,1] + T1[1,2]*T2[2,1] + T1[1,3]*T2[3,1] ;
   T3[1,2] := T1[1,1]*T2[1,2] + T1[1,2]*T2[2,2] + T1[1,3]*T2[3,2] ;
   T3[1,3] := T1[1,1]*T2[1,3] + T1[1,2]*T2[2,3] + T1[1,3]*T2[3,3] ;

   T3[2,1] := T1[2,1]*T2[1,1] + T1[2,2]*T2[2,1] + T1[2,3]*T2[3,1] ;
   T3[2,2] := T1[2,1]*T2[1,2] + T1[2,2]*T2[2,2] + T1[2,3]*T2[3,2] ;
   T3[2,3] := T1[2,1]*T2[1,3] + T1[2,2]*T2[2,3] + T1[2,3]*T2[3,3] ;

   T3[3,1] := T1[3,1]*T2[1,1] + T1[3,2]*T2[2,1] + T1[3,3]*T2[3,1] ;
   T3[3,2] := T1[3,1]*T2[1,2] + T1[3,2]*T2[2,2] + T1[3,3]*T2[3,2] ;
   T3[3,3] := T1[3,1]*T2[1,3] + T1[3,2]*T2[2,3] + T1[3,3]*T2[3,3] ;
END ;
```

`Biderketa_3x3` prozedura horri orokortasuna faltatzen zaio, horregatik `Biderketa` izeneko beste hau kodetu dugu `ArrayenAlderantzizkoa_OROKORRA_2_8` izena duen programan, non `Luz` aldagaiak arrayen dimentsio efektiboa adierazten duen:

```
PROCEDURE Biderketa (CONST T1,T2:DM_Taula3; Luz:Byte; VAR T3:DM_Taula3) ;
VAR
   Lerro, Zutabe, k : Byte ;
   Metatua : Real ;
BEGIN
   FOR Lerro:=1 TO Luz DO
   BEGIN
      FOR Zutabe:=1 TO Luz DO
      BEGIN
         Metatua := 0 ;
         FOR k:=1 TO Luz DO
         BEGIN
            Metatua := Metatua + T1[Lerro,k]*T2[k,Zutabe] ;
         END ;
         T3[Lerro,Zutabe] := Metatua ;
      END ;
   END ;
END ;
```

10.5.1.4 Zatiketa

`T1` eta `T2` zenbaki errealen bi array ditugula euren arteko zatidura burutzeko, `T2` array zatitzailearen alderantzizkoa lortuko da lehenik eta ondoren zatiketa operazioa ikusitako biderketa bezala plantea daiteke. `T1` zati `T2` beste array bat izango da, `T3` deituko duguna:

$$T3 = T1 \text{ zati } T2 = T1 \text{ bider } (T2)^{-1}$$

Beraz, array karratu baten alderantzizkoa nola lortzen den gogoratuko dugu orain. Array baten alderantzizkoa hiru urratsetan kalkulatzen da:

1. Sarrerako arrayaren iraulia
2. Array irauliaren adjuntua
3. Array irauliaren adjuntua zati sarrerako arrayaren determinantea

$$(T2)^{-1} = Adj(T2^{irauli}) \; / \; Determ(T2)$$

Horretarako array bati dagokion array iraulia definitu beharko dugu, halaber, array baten array adjuntua ere definituko dugu. Baina ezer baino lehen `T2` sarrerako arrayaren determinantea zero ez dela frogatu beharra dago, `Determ(T2)` espresioak zero balio duenean `T2` arrayak alderantzizkorik ez baitu.

10.5.1.4.1 Array karratu baten determinantea

Irakurleak jakingo duenez, `3x3` array baten determinantea lortzeko honelako hiru errutinak idaz daitezke (ikus `ArrayenAlderantzizkoa` izeneko programa):

```
FUNCTION Diagonale_a_3x3 (CONST T:DM_Taula3) : Real ;
VAR
  Lerro, Zutabe : Byte ;
  Metatua : Real ;
BEGIN
   Metatua := 1 ;
   FOR Lerro:=1 TO MAX DO
      FOR Zutabe:=1 TO MAX DO
      BEGIN
          IF Lerro=Zutabe THEN
            Metatua := Metatua * T[Lerro,Zutabe] ;
      END ;
   Diagonale_a_3x3 := Metatua ;
END ;


FUNCTION Diagonale_b_3x3 (CONST T:DM_Taula3) : Real ;
VAR
  Lerro, Zutabe : Byte ;
  Metatua : Real ;
BEGIN
   Metatua := 1 ;
   FOR Lerro:=1 TO MAX DO
      FOR Zutabe:=1 TO MAX DO
      BEGIN
          IF Lerro+Zutabe=MAX+1 THEN
            Metatua := Metatua * T[Lerro,Zutabe] ;
      END ;
   Diagonale_b_3x3 := Metatua ;
END ;


FUNCTION Determinante3 (CONST T:DM_Taula3) : Real ;
VAR
  Diagonalak, Hiruki1, Hiruki2 : Real ;
BEGIN
   Diagonalak := Diagonale_a_3x3(T) - Diagonale_b_3x3(T) ;
   Hiruki1 := T[2,1]*T[3,2]*T[1,3] + T[1,2]*T[2,3]*T[3,1] ;
   Hiruki2 := T[1,2]*T[2,1]*T[3,3] + T[3,2]*T[2,3]*T[1,1] ;
   Determinante3 := Diagonalak + Hiruki1 - Hiruki2 ;
END ;
```

`Determinante3` prozedura hori ezin daitekeenez beste dimentsioetako array karratuekin erabili orokortasuna faltatzen zaiola baiezta dezakegu, horregatik array baten determinantea kalkulatzeko beste metodoren batean oinarritu beharko gara. Izan ere, array baten determinantea lortzeko arrayaren edozein lerro edo zutabe aukeratuz ondoko garapena egin daiteke. Demagun arrayaren balioak `1`-tik `9`-ra doazen kopuruak direla, taula horri dagokion determinantea lehenengo lerrotik garatuz honela lortzen da:

$$\begin{vmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} = \mathbf{+1} \times \begin{vmatrix} 5 & 6 \\ 8 & 9 \end{vmatrix} \mathbf{-2} \times \begin{vmatrix} 4 & 6 \\ 7 & 9 \end{vmatrix} \mathbf{+3} \times \begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix}$$

Non `3x3` array baten determinantea kalkulatzeko `2x2` hiru determinanteen ebazpena lortu behar da. `2x2` hiru determinanteak ere modu beretsuan kalkula daitezke, adibidez:

$$\begin{vmatrix} \mathbf{5} & \mathbf{6} \\ 8 & 9 \end{vmatrix} = \mathbf{+5} \times 9 \quad \mathbf{-6} \times 8$$

Orokorrean, `nxn` array baten determinantea kalkulatzeko `n` aldiz `(n-1)x(n-1)` ordenako determinanteak lortuko dira, hona hemen `Luz` lerro kopurua duen array karratu baten determinantea kalkulatzen duen `DeterminanteOrokorra` funtzio errekurtsiboa:

```
FUNCTION Zeinu (Lerro, Zutabe : Byte) : Integer ;
BEGIN
   IF (Lerro+Zutabe) MOD 2 = 0 THEN
     Zeinu := 1
   ELSE
     Zeinu := -1 ;
END ;

FUNCTION DeterminanteOrokorra (CONST T:DM_Taula; Luz:Byte) : Real ;
CONST
  Lerro = 1 ;   { lehen lerroa garatuz }
VAR
  Laguntzaile : DM_Taula ;
  Zutabe, Ler, Zut, i, j : Byte ;
  Metatua, Determ : Real ;
BEGIN
   Metatua := 0.0 ;
   FOR Zutabe:=1 TO Luz DO
   BEGIN
      i := 1 ;
      FOR Ler:=Lerro+1 TO MAX DO
      BEGIN
         j := 1 ;
         FOR Zut:=1 TO MAX DO
         BEGIN
            IF (Ler<>Lerro) AND (Zut<>Zutabe) THEN
            BEGIN
               Laguntzaile[i,j] := T[Ler,Zut] ;
               j := j + 1 ;
            END ;
         END ;
         IF ((Ler<>Lerro) AND (Zut<>Zutabe)) OR (j=MAX) THEN
            i := i + 1 ;
      END ;
      IF Luz=2 THEN Determ := Laguntzaile[1,1]
               ELSE Determ := DeterminanteOrokorra(Laguntzaile, Luz-1) ;
      Metatua := Metatua + Zeinu (Lerro,Zutabe) * T[Lerro,Zutabe] * Determ ;
   END ;
   DeterminanteOrokorra := Metatua ;
END ;
```

`DeterminanteOrokorra` funtzioaren dei errekurtsiboa `IF-THEN-ELSE` sententziaren `ELSE` zatian aurkitzen da eta beltzez adierazi dugu, ikusten denez arrayaren luzera logikoa unitate bat txikagoa delako azkenean `IF-THEN-ELSE` sententziaren `THEN` zatitik igaroko da, ondorioz dei errekurtsiboen kopurua ez da infinitoa izango.

10.5.1.4.2 Array karratu baten array iraulia

Edozein array karraturen array iraulia lortzeko sarrerako arrayaren lerroak sarrerako arrayaren zutabeengatik ordezkatuko dira. Demagun sarrerako arrayaren balioak 1-tik 9-ra doazen kopuruak direla, taula horri dagokion array iraulia hauxe litzateke:

$$\begin{pmatrix} \boxed{1 & 2 & 3} \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}^{\text{iraulia}} = \begin{pmatrix} \boxed{1 \\ 2 \\ 3} & 4 & 7 \\ & 5 & 8 \\ & 6 & 9 \end{pmatrix}$$

Jarraian T1 array baten T2 array iraulia eskuratu eta itzuli egiten duen TaulaIrauli prozedura erakusten da (ikus ArrayenAlderantzizkoa_OROKORRA_2_8 delako programa):

```
PROCEDURE TaulaIrauli (CONST T1:DM_Taula; Luz:Byte; VAR T2:DM_Taula) ;
VAR
  Lerro, Zutabe : Byte ;
  Laguntzaile : Real ;
BEGIN
   T2 := T1 ;
   FOR Lerro:=1 TO Luz DO
      FOR Zutabe:=Lerro TO Luz DO
      BEGIN
         Laguntzaile := T2[Lerro, Zutabe] ;
         T2[Lerro, Zutabe] := T2[Zutabe, Lerro] ;
         T2[Zutabe, Lerro] := Laguntzaile ;
      END ;
END ;
```

10.5.1.4.3 Array karratu baten array adjuntua

Array karraturen array adjuntua lortzeko sarrerako arrayaren elementu guztiei dagozkien elementu adjuntua kalkulatuko da. Datua den arrayaren elementu jakin baten adjuntua, bere lerro eta zutabe kenduz geratzen den orden txikiko arrayaren determinantea litzateke. Demagun sarrerako arrayaren balioak 1-tik 9-ra doazen kopuruak direla, taula horri dagokion array adjuntua hauxe litzateke:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}^{\text{adjuntua}} = \begin{pmatrix} \begin{vmatrix} 5 & 6 \\ 8 & 9 \end{vmatrix} & \begin{vmatrix} 4 & 6 \\ 7 & 9 \end{vmatrix} & \begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix} \\ \begin{vmatrix} 2 & 3 \\ 8 & 9 \end{vmatrix} & \begin{vmatrix} 1 & 3 \\ 7 & 9 \end{vmatrix} & \begin{vmatrix} 1 & 2 \\ 7 & 8 \end{vmatrix} \\ \begin{vmatrix} 2 & 3 \\ 5 & 6 \end{vmatrix} & \begin{vmatrix} 1 & 3 \\ 4 & 6 \end{vmatrix} & \begin{vmatrix} 1 & 2 \\ 7 & 8 \end{vmatrix} \end{pmatrix}$$

3x3 array beten ordez nxn array bat bagenu, bere array adjuntua lortzeko algoritmoa aurrerago ikusi dugu, izan ere, funtsean, array adjuntua lortzeko algoritmoa array baten determinantea kalkulatzeko ikasi duguna da.

Jarraian aurkezten den TaulaAdjunt prozedura honela deskribatuko genuke: T1 array baten dimentsio efektiboa Luz aldagaiak zehazten du, eta T1 taula horri dagokion T2 array adjuntua lortzen du. Ikus ArrayenAlderantzizkoa_OROKORRA_2_8 delako programa:

```
FUNCTION Zeinu (Lerro, Zutabe : Byte) : Integer ;
BEGIN
   IF (Lerro+Zutabe) MOD 2 = 0 THEN
     Zeinu := 1
   ELSE
     Zeinu := -1 ;
END ;


PROCEDURE TaulaAdjunt (CONST T1:DM_Taula; Luz:Byte; VAR T2:DM_Taula) ;
VAR
  Laguntzaile : DM_Taula ;
  Lerro, Zutabe, Ler, Zut, i, j : Byte ;
  Determ : Real ;
BEGIN
   FOR Zutabe:=1 TO Luz DO
   BEGIN
      FOR Lerro:=1 TO Luz DO
      BEGIN
         i := 1 ;
         FOR Ler:=1 TO Luz DO
         BEGIN
            j := 1 ;
            FOR Zut:=1 TO Luz DO
            BEGIN
               IF (Ler<>Lerro) AND (Zut<>Zutabe) THEN
               BEGIN
                  Laguntzaile[i,j] := T1[Ler,Zut] ;
                  j := j + 1 ;
               END ;
            END ;
            IF ((Ler<>Lerro) AND (Zut<>Zutabe)) OR (j=Luz) THEN
               i := i + 1 ;
         END ;
         IF Luz=2 THEN Determ := Laguntzaile[1,1]
                  ELSE Determ := DeterminanteOrokorra(Laguntzaile, Luz-1) ;
         T2[Lerro,Zutabe] := Zeinu(Lerro,Zutabe) * Determ ;
      END ;
   END ;
END ;
```

`TaulaAdjunt` prozedurak `DeterminanteOrokorra()` funtzio errekurtsiboa deitzen du. `TaulaAdjunt` prozeduraren itxura (`FOR-DO` sententzien antolaketa) determinantea kalkulatzen duen funtzioarena da, `DeterminanteOrokorra()`-rena alegia.

10.5.2 Array karratuen aritmetikarako unitatearen interfazea

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

`ArraErag` unitatearen interfazea

```
INTERFACE

USES
    Crt ;
CONST
    MAX = 8 ;
TYPE
    DM_Taula = ARRAY[1..MAX, 1..MAX] OF Real ;
```

```
PROCEDURE TaulaTeklatutikBete (VAR T:DM_Taula; VAR Luz:Byte) ;
PROCEDURE TaulaErakutsi (CONST T:DM_Taula; Luz:Byte) ;
PROCEDURE TaulaIrauli (CONST T1:DM_Taula; Luz:Byte; VAR T2:DM_Taula) ;
PROCEDURE TaulaAdjuntua (CONST T1:DM_Taula; Luz:Byte; VAR T2:DM_Taula) ;
FUNCTION Determinante (CONST T:DM_Taula; Luz:Byte) : Real ;
PROCEDURE TaulaAlderantziz (CONST T1:DM_Taula; Luz:Byte; VAR T2:DM_Taula) ;
PROCEDURE Batuketa (CONST T1,T2:DM_Taula; Luz:Byte; VAR T3:DM_Taula) ;
PROCEDURE Kenketa (CONST T1,T2:DM_Taula; Luz:Byte; VAR T3:DM_Taula) ;
PROCEDURE Biderketa (CONST T1,T2:DM_Taula; Luz:Byte; VAR T3:DM_Taula) ;
PROCEDURE Zatiketa (CONST T1,T2:DM_Taula; Luz:Byte; VAR T3:DM_Taula) ;
```

10.5.3 Array karratuen aritmetikarako unitatearen inplementazioa

Interfazean plazaturiko datu-mota eta azpiprogramaz gain, `ArraErag` unitateak beste azpirrutina probatuak erabiltzen dituenez unitatearen inplementazioan zehaztuko dira: `Zeinu()` funtzio pribatua da, bestalde, pribatuak diren `TaulaZatiDeterminante()`, `TaulaNulua()` eta `TaulaAleatoriokiBete()` prozedurak ditugu ere.

`ArraErag` unitatearen inplementazioa

```
IMPLEMENTATION

{-------------------------------------------------------------------------}
{ Funtzio pribatua:                                                       }
{ Array baten elementuari dagokion adjuntuaren zeinua zehazten du.        }
{-------------------------------------------------------------------------}
FUNCTION Zeinu (Lerro, Zutabe : Byte) : Integer ;
BEGIN
   IF (Lerro+Zutabe) MOD 2 = 0 THEN
     Zeinu := 1
   ELSE
     Zeinu := -1 ;
END ;


{-------------------------------------------------------------------------}
{ Prozedura pribatua:                                                     }
{ Arrayaren osagaiak zati zenbaki erreal bat (determinantearen balioa). }
{-------------------------------------------------------------------------}
PROCEDURE TaulaZatiDeterminante (VAR T:DM_Taula; Luz:Byte; Delta:Real) ;
VAR
  Lerro, Zutabe : Byte ;
BEGIN
   FOR Lerro:=1 TO Luz DO
      FOR Zutabe:=1 TO Luz DO
         T[Lerro,Zutabe] := T[Lerro,Zutabe] / Delta ;
END ;


{-------------------------------------------------------------------------}
{ Prozedura pribatua:                                                     }
{ Neurri jakin bateko array karratu bat zeroz bete.                       }
{-------------------------------------------------------------------------}
PROCEDURE TaulaNulua (VAR T:DM_Taula; Luz:Byte) ;
VAR
  Lerro, Zutabe : Byte ;
BEGIN
   FOR Lerro:=1 TO Luz DO
      FOR Zutabe:=1 TO Luz DO
         T[Lerro,Zutabe] := 0 ;
END ;
```

```
{-------------------------------------------------------------------}
{ -9 eta 9 bitarteko balioak aleatoriki hartuz array bat betetzen du. }
{ Arraya karratua izango da eta bere dimentsioa kanpotik datorkio.    }
{-------------------------------------------------------------------}
PROCEDURE TaulaAleatoriokiBete (VAR T:DM_Taula; Luz:Byte) ;
VAR
  Lerro, Zutabe : Byte ;
  Zeinua : Integer ;
BEGIN
   Randomize ;
   FOR Lerro:=1 TO Luz DO
      FOR Zutabe:=1 TO Luz DO
      BEGIN
         Zeinua := Random (2) ;
         IF Zeinua=0 THEN
           Zeinua := -1 ;
         T[Lerro, Zutabe] := Zeinua * Random(10) ;
      END ;
END ;


{-------------------------------------------------------------------}
{ -9 eta 9 bitarteko balioak teklatuz emanik array bat betetzen du. }
{ Arraya karratua izango da eta bere dimentsioa teklatuz ematen da. }
{-------------------------------------------------------------------}
PROCEDURE TaulaTeklatutikBete (VAR T:DM_Taula; VAR Luz:Byte) ;
VAR
  Lerro, Zutabe : Byte ;
  Zeinua : Integer ;
  Erantz : Char ;
BEGIN
   REPEAT
      Write ('Zein da array karratuaren lerro (edo zutabe) kopurua?  (2..8)   ') ;
      ReadLn (Luz) ;
   UNTIL (Luz > 1) AND (Luz <= MAX) ;
   Write (Luz*Luz, ' zenbaki teklatuz irakurri ordez, ') ;
   Write ('nahi duzu aleatorioki lortzea? (B/E) ') ;
   ReadLn (Erantz) ;
   IF UpCase(Erantz) = 'B' THEN
     TaulaAleatoriokiBete (T, Luz)
   ELSE
   BEGIN
      FOR Lerro:=1 TO Luz DO
         FOR Zutabe:=1 TO Luz DO
         BEGIN
            Write ('Eman [', Lerro, ',', Zutabe, '] osagaia:  ') ;
            ReadLn (T[Lerro, Zutabe]) ;
         END ;
   END ;
END ;


{-------------------------------------------------------------------}
{ Array karratu baten elementuak pantailaratzeko azpiprograma.      }
{-------------------------------------------------------------------}
PROCEDURE TaulaErakutsi (CONST T:DM_Taula; Luz:Byte) ;
VAR
  Lerro, Zutabe : Byte ;
BEGIN
   FOR Lerro:=1 TO Luz DO
   BEGIN
      FOR Zutabe:=1 TO Luz DO
      BEGIN
         Write (T[Lerro, Zutabe] :9:3) ;
      END ;
      WriteLn ;
   END ;
END ;
```

```
{-------------------------------------------------------------------}
{ Array karratu baten elementuak irauliz beste array bat lortzen da. }
{-------------------------------------------------------------------}
PROCEDURE TaulaIrauli (CONST T1:DM_Taula; Luz:Byte; VAR T2:DM_Taula) ;
VAR
  Lerro, Zutabe : Byte ;
  Laguntzaile : Real ;
BEGIN
   T2 := T1 ;
   FOR Lerro:=1 TO Luz DO
      FOR Zutabe:=Lerro TO Luz DO
      BEGIN
         Laguntzaile := T2[Lerro, Zutabe] ;
         T2[Lerro, Zutabe] := T2[Zutabe, Lerro] ;
         T2[Zutabe, Lerro] := Laguntzaile ;
      END ;
END ;


{-------------------------------------------------------------------}
{ Array karratu baten elementu guztien adjuntuak lortuz sarrerako  }
{ arrayaren array adjuntua erdiesten da.                           }
{-------------------------------------------------------------------}
PROCEDURE TaulaAdjuntua (CONST T1:DM_Taula; Luz:Byte; VAR T2:DM_Taula) ;
VAR
  Laguntzaile : DM_Taula ;
  Lerro, Zutabe, Ler, Zut, i, j : Byte ;
  Determ : Real ;
BEGIN
   FOR Zutabe:=1 TO Luz DO
   BEGIN
      FOR Lerro:=1 TO Luz DO
      BEGIN
         i := 1 ;
         FOR Ler:=1 TO Luz DO
         BEGIN
            j := 1 ;
            FOR Zut:=1 TO Luz DO
            BEGIN
               IF (Ler<>Lerro) AND (Zut<>Zutabe) THEN
               BEGIN
                  Laguntzaile[i,j] := T1[Ler,Zut] ;
                  j := j + 1 ;
               END ;
            END ;
            IF ((Ler<>Lerro) AND (Zut<>Zutabe)) OR (j=Luz) THEN
               i := i + 1 ;
         END ;
         IF Luz=2 THEN Determ := Laguntzaile[1,1]
                  ELSE Determ := Determinante(Laguntzaile, Luz-1) ;
         T2[Lerro,Zutabe] := Zeinu(Lerro,Zutabe) * Determ ;
      END ;
   END ;
END ;


{-------------------------------------------------------------------}
{ Array karratu baten determinantea kalkulatzen duen funtzioa.      }
{ Funtzio errekurtsiboa denez, deiak eteteko baldintzaren bat       }
{ jarri beharra dago, kasu honetan 2x2 array bat lerroz garatzean   }
{ elementu bakoitzari dagokion orden txikiko determinantea 1x1      }
{ array bat litzateke (2x2 arrayaren beste elementu bat alegia).    }
{-------------------------------------------------------------------}
FUNCTION Determinante (CONST T:DM_Taula; Luz:Byte) : Real ;
CONST
  Lerro = 1 ;   { lehen lerroa garatuz }
VAR
  Laguntzaile : DM_Taula ;
  Zutabe, Ler, Zut, i, j : Byte ;
  Metatua, Determ : Real ;
```

```
BEGIN
   Metatua := 0.0 ;
   FOR Zutabe:=1 TO Luz DO
   BEGIN
      i := 1 ;
      FOR Ler:=Lerro+1 TO MAX DO
      BEGIN
         j := 1 ;
         FOR Zut:=1 TO MAX DO
         BEGIN
            IF (Ler<>Lerro) AND (Zut<>Zutabe) THEN
            BEGIN
               Laguntzaile[i,j] := T[Ler,Zut] ;
               j := j + 1 ;
            END ;
         END ;
         IF ((Ler<>Lerro) AND (Zut<>Zutabe)) OR (j=MAX) THEN
            i := i + 1 ;
      END ;
      IF Luz=2 THEN Determ := Laguntzaile[1,1]
               ELSE Determ := Determinante(Laguntzaile, Luz-1) ;
      Metatua := Metatua + Zeinu (Lerro,Zutabe) * T[Lerro,Zutabe] * Determ ;
   END ;
   Determinante := Metatua ;
END ;


{-------------------------------------------------------------------------}
{ Sarrerako array karratu baten alderantzizkoa lortzen duen prozedura, }
{ alderantzizkoa kalkulatzeko ezer baino lehen determinantea ezagutu   }
{ behar da, ondoren, sarrerako iaruliaren adjuntua eskuratuko da.      }
{-------------------------------------------------------------------------}
PROCEDURE TaulaAlderantziz (CONST T1:DM_Taula; Luz:Byte; VAR T2:DM_Taula) ;
VAR
   Delta : Real ;
   Taula : DM_Taula ;
BEGIN
   Delta := Determinante (T1, Luz) ;
   IF Delta<>0 THEN
   BEGIN
      TaulaIrauli (T1, Luz, Taula) ;
      TaulaAdjuntua (Taula, Luz, T2) ;
      TaulaZatiDeterminante (T2, Luz, Delta) ;
   END
   ELSE
   BEGIN
      Write ('Sarrerako arrayaren determinantea 0 denez ') ;
      WriteLn ('bere alderantzizkoa array nulua da') ;
      TaulaNulua (T2, Luz) ;
   END ;
END ;


{-------------------------------------------------------------------------}
{ Bi array karratuen arteko batura kalkulatzen duen prozedura.       }
{ Sarreako arrayen dimentsio bera izango du emaitzak.                }
{-------------------------------------------------------------------------}
PROCEDURE Batuketa (CONST T1,T2:DM_Taula; Luz:Byte; VAR T3:DM_Taula) ;
VAR
  Lerro, Zutabe : Byte ;
BEGIN
   FOR Lerro:=1 TO Luz DO
      FOR Zutabe:=1 TO Luz DO
         T3[Lerro,Zutabe] := T1[Lerro,Zutabe] + T2[Lerro,Zutabe] ;
END ;

{-------------------------------------------------------------------------}
{ Bi array karratuen arteko kendura kalkulatzen duen prozedura.      }
{ Sarreako arrayen dimentsio bera izango du emaitzak.                }
{-------------------------------------------------------------------------}
```

```
PROCEDURE Kenketa (CONST T1,T2:DM_Taula; Luz:Byte; VAR T3:DM_Taula) ;
VAR
  Lerro, Zutabe : Byte ;

BEGIN
   FOR Lerro:=1 TO Luz DO
      FOR Zutabe:=1 TO Luz DO
         T3[Lerro,Zutabe] := T1[Lerro,Zutabe] - T2[Lerro,Zutabe] ;
END ;


{--------------------------------------------------------------------}
{ Bi array karratuen arteko biderkadura kalkulatzen duen prozedura.  }
{ Sarreako arrayen dimentsio bera izango du emaitzak.                }
{--------------------------------------------------------------------}
PROCEDURE Biderketa (CONST T1,T2:DM_Taula; Luz:Byte; VAR T3:DM_Taula) ;
VAR
   Lerro, Zutabe, k : Byte ;
   Metatua : Real ;
BEGIN
   FOR Lerro:=1 TO MAX DO
   BEGIN
      FOR Zutabe:=1 TO MAX DO
      BEGIN
         Metatua := 0 ;
         FOR k:=1 TO MAX DO
         BEGIN
            Metatua := Metatua + T1[Lerro,k]*T2[k,Zutabe] ;
         END ;
         T3[Lerro,Zutabe] := Metatua ;
      END ;
   END ;
END ;


{--------------------------------------------------------------------}
{ Bi array karratuen arteko zatidura lortzeko bi urratsetan banantzen }
{ kalkulua. Lehenenik array zatitzailearen alderantzizkoa eskuratzen  }
{ da, eta ondoren, datuen zatiketa egiteko biderketa bat burutuko da. }
{--------------------------------------------------------------------}
PROCEDURE Zatiketa (CONST T1,T2:DM_Taula; Luz:Byte; VAR T3:DM_Taula) ;
VAR
  Lerro, Zutabe : Byte ;
  Taula1, Taula2 : DM_Taula ;
  Delta : Real ;
BEGIN
   Delta := Determinante (T2, Luz) ;
   IF Delta<>0 THEN
   BEGIN
      TaulaIrauli (T2, Luz, Taula1) ;
      TaulaAdjuntua (Taula1, Luz, Taula2) ;
      TaulaZatiDeterminante (Taula2, Luz, Delta) ;
   END
   ELSE
   BEGIN
      Write ('Sarrerako arrayaren determinantea 0 denez ') ;
      WriteLn ('bere alderantzizkoa array nulua da') ;
      TaulaNulua (Taula2, Luz) ;
   END ;

   Biderketa (T1, Taula2, Luz, T3) ;
END ;


BEGIN
   Writeln ('ArraErag unitatearen hasieraketa');
   Writeln ('--------------------------------');
   Readln;
END.
```

10.5.4 Array karratuen aritmetikarako unitatea erabiltzen

`ArraErag` unitatea erabiltzeko bost programa prestatu izan ditugu, guztietan array karratuen arteko kalkulu errazak egin eta emaitza erakusten da. Jarraian programa bakoitzaren izen eta deskribapen laburra ematen da:

`ArrayenEragiketaAritmetikoak1` programa honek `Taula1` array karratu baten dimentsio efektiboa eskatzen du lehenik, ondoren, taula horren balioak teklatuz eman ordez aleatorioki sortzea aukera ematen du. Datuak memorian daudelarik, `Taula1` arrayaren alderantzizko arraya lortu egiten da, eta, amaitzeko, `Taula1` eta bere alderantzizkoa den `Taula2` arrayen arteko biderkadura kalkulatzen da. Zalantzarik gabe, azken emaitza array identitatea izateak gauzak ondo joan direla frogatzen du.

Matematikoki formulatuz:

$$\begin{aligned} &\texttt{Taula1} \\ &\texttt{Taula2} = (\texttt{Taula1})^{-1} \\ &\texttt{I} = \texttt{Taula1} * \texttt{Taula2} \end{aligned}$$

`ArrayenEragiketaAritmetikoak2` programa hau eta `ArrayenEragiketaAritmetikoak1` izeneko programa berdintsuak dira. `Taula1` array karratu baten dimentsio efektiboa eskatzen du lehenik, ondoren, taula horren balioak teklatuz eman ordez aleatorioki sortzea aukera ematen du. Behin datuak memorian daudelarik, `Taula1` arraya zati sarrerako `Taula1` eginez zatidura pantailaratzen da. Zalantzarik gabe, hemen ere, azken emaitza array identitatea izateak gauzak ondo joan direla frogatzen du.

Matematikoki formulatuz:

$$\begin{aligned} &\texttt{Taula1} \\ &\texttt{I} = \texttt{Taula1} / \texttt{Taula1} \end{aligned}$$

`ArrayenEragiketaAritmetikoak3` programa hau ere `ArrayenEragiketaAritmetikoak1` eta `ArrayenEragiketaAritmetikoak2` bi programekin alderatuz berdintsuak dira. `Taula1` array karratu baten dimentsio efektiboa eskatzen du eta bere balioak aleatorioki aukeratzen dira. `Taula2` izeneko beste array batentzat aurrekoa errepikatuz bi arrayen balioak memorian izango ditugu. `Taula1` arraya zati sarrerako beste `Taula2` eginez `Zatidura1` izeneko arraya lortuko da, `Taula2` arraya zati sarrerako beste `Taula1` eginez `Zatidura2` izeneko arraya lortuko da, amaitzeko `Zatidura1` eta `Zatidura2` arrayen arteko biderketa eginez `Biderkadura` izeneko arraya lortuko da zein pantailaratzen den. Zalantzarik gabe, hemen ere, nola ez, azken emaitza array identitatea izateak gauzak ondo joan direla frogatzen digu.

Matematikoki formulatuz:

$$\begin{aligned} &\texttt{Taula1, Taula2} \\ &\texttt{Zatidura1} = \texttt{Taula1} / \texttt{Taula12} \\ &\texttt{Zatidura2} = \texttt{Taula2} / \texttt{Taula11} \\ &\texttt{I} = \texttt{Zatidura1} * \texttt{Zatidura2} \end{aligned}$$

`ArrayenEragiketaAritmetikoak4` programa honek `Taula1` array karratu baten dimentsio efektiboa eskatzen du lehenik eta bere balioak aleatorioki aukeratzen ditu, ondoren, taula horren

tamaina efektiboko `Identitatea` delako array unitarioa lortzen da. Datuak memorian daudelarik, `Taula1` arraya eta `Identitatea` arrayaren arteko biderkadura kalkulatu eta pantailaratu egiten da. Datuak memorian daudelarik, `Taula1` arraya eta `Identitatea` arrayaren arteko zatidura kalkulatu eta pantailaratu egiten da. Hemen ere gauzak ondo joan direla frogatzerik dago, emaitza diren `Biderkadura` eta `Zatidura` array biak sarrerako `Taula1` arrayarekin alderatuz berdinak izan behar baitira.

Matematikoki formulatuz:

```
Taula1, I
Taula1 = Taula1 * I
Taula1 = Taula1 / I
```

`ArrayenEragiketaAritmetikoak5` programa hau eta `ArrayenEragiketaAritmetikoak4` izeneko programa berdintsuak dira. `Taula1` array karratu baten dimentsio efektiboa eskatzen du eta bere balioak aleatorioki aukeratzen ditu, ondoren, taula horren tamaina efektiboko `Nulua` delako zeroz beteriko arraya lortzen da. Datuak memorian daudelarik, `Taula1` arraya eta `Nulua` arrayaren arteko batura kalkulatu eta pantailaratu egiten da. Datuak memorian daudelarik, `Taula1` arraya eta `Nulua` arrayaren arteko kendura erdietsi eta pantailaratu egiten da. Hemen ere, nola ez, gauzak ondo joan direla frogatzerik dago, emaitza diren `Batura` eta `Kendura` array biak sarrerako `Taula1` arrayarekin alderatuz berdinak izan behar baitira.

Matematikoki formulatuz:

```
Taula1, N
Taula1 = Taula1 + N
Taula1 = Taula1 - N
```

10.5.4.1 Ekuazio sistemak ebazten

`ArraErag` unitatea ekuzio linealen sistemak ebazteko erabil daiteke ere. Jarraian oso ezagunak diren bi metodo hauek aurkeztuko ditugu Cramer metodoa eta Gauss-Jordan metodoa.

10.5.4.1.1 Cramer

Cramer araua oso zehatza eta erraza da sistema linealak ebazteko. Sistema modu egokian antolatu ondoren Cramer metudoaren funtsa determinante biren zatidura kalkulatzea da. Esate baterako, eta hiru ekuazioko sistema bat hartuz, honako antolaketa burutu behar da Cramerren metodoa aplikatu aurretik:

$$A_1 x_1 + B_1 x_2 + C_1 x_3 = D_1$$
$$A_2 x_1 + B_2 x_2 + C_2 x_3 = D_2$$
$$A_3 x_1 + B_3 x_2 + C_3 x_3 = D_3$$

Non A_i, B_i eta C_i zenbaki errealak ekuazioen koefizienteak diren, eta D_i ekuazioen termino askea (zenbaki erreal bat ere). Sistemaren soluzioak x_i inkognitak izango dira.

Cramerren araua gogoratuz (non E_i emaitzak x_i inkogniten soluzioak diren):

$$E_1 = \frac{\begin{vmatrix} \mathbf{D_1} & B_1 & C_1 \\ \mathbf{D_2} & B_2 & C_2 \\ \mathbf{D_3} & B_3 & C_3 \end{vmatrix}}{\begin{vmatrix} A_1 & B_1 & C_1 \\ A_2 & B_2 & C_2 \\ A_3 & B_3 & C_3 \end{vmatrix}} \qquad E_2 = \frac{\begin{vmatrix} A_1 & \mathbf{D_1} & C_1 \\ A_2 & \mathbf{D_2} & C_2 \\ A_3 & \mathbf{D_3} & C_3 \end{vmatrix}}{\begin{vmatrix} A_1 & B_1 & C_1 \\ A_2 & B_2 & C_2 \\ A_3 & B_3 & C_3 \end{vmatrix}} \qquad E_2 = \frac{\begin{vmatrix} A_1 & B_1 & \mathbf{D_1} \\ A_2 & B_2 & \mathbf{D_2} \\ A_3 & B_3 & \mathbf{D_3} \end{vmatrix}}{\begin{vmatrix} A_1 & B_1 & C_1 \\ A_2 & B_2 & C_2 \\ A_3 & B_3 & C_3 \end{vmatrix}}$$

Ikus daitekeenez metodo honetan `ArraErag` unitateko determinantea kalkulatzen duen funtzioa erabili daiteke, jarraian ematen den programan egin dugun bezala:

```
PROGRAM Cramer ;                                { \TP70\10\CRAMER.PAS }
USES
   ArraErag ;
TYPE
   DM_ElementuAskeak = ARRAY[1..MAX] OF Real ;          (* zutabe bat da *)
   (* koefizienteen arraya honela definitzen da ArrErag unitate barruan: *)
   (*       DM_Taula = ARRAY[1..MAX, 1..MAX] OF Real ;                    *)


PROCEDURE SistemaIrakurri (VAR K : DM_Taula ;
                           VAR Z : DM_ElementuAskeak ;
                           VAR Kop : Integer) ;
VAR
   i, j : Integer ;
BEGIN
  REPEAT
     Write ('Zein da ekuazioen kopurua (1..', MAX, ')?  ') ;
     ReadLn (Kop) ;
  UNTIL (Kop >= 1) AND (Kop <= MAX) ;

  (* Sistema konpatiblea eta determinatua izan dadin:
        ekuazioen kopurua = ikogniten kopurua         *)

  WriteLn ('Sistemaren koefizienteak eman:') ;
  FOR i:=1 TO Kop DO
  BEGIN
      FOR j:=1 TO Kop DO
      BEGIN
          Write ('[', i, ',', j, '] koefizientea ---->  ') ;
          ReadLn (K[i, j]) ;
      END;
      Write(i, '. ekuazioaren balio askea: ') ;
      ReadLn (Z[i]) ;
  END;
END;


PROCEDURE SistemaErakutsi (CONST K : DM_Taula ;
                           CONST Z : DM_ElementuAskeak ;
                           Kop : Integer) ;
VAR
   i, j : Integer ;
BEGIN
  FOR i:=1 TO Kop DO
  BEGIN
      FOR j:=1 TO Kop DO
      BEGIN
          Write (K[i, j]:7:3) ;
      END;
      Write ('   ---> ') ;
      Write (Z[i]:7:3) ;
      WriteLn ;
  END;
END;
```

```
PROCEDURE SoluzioakBilatu (CONST K : DM_Taula ;
                           CONST Z : DM_ElementuAskeak ;
                           VAR S : DM_ElementuAskeak ;
                           Determ : Real ;
                           Kop : Integer) ;
VAR
   Lerro, Zutabe : Integer ;
   KoefBerri : DM_Taula ;
BEGIN
   FOR Zutabe:=1 TO Kop DO
   BEGIN
      KoefBerri := K ;
      FOR Lerro:=1 TO Kop DO
         KoefBerri[Lerro, Zutabe] := Z[Lerro] ;

      S[Zutabe] := Determinante (KoefBerri, Kop) / Determ ;
   END ;
END ;


PROCEDURE SoluzioakIkusi (CONST S : DM_ElementuAskeak; Kop : Integer) ;
VAR
   Kontagailu : Integer ;
BEGIN
   FOR Kontagailu:=1 TO Kop DO
      Writeln('E', Kontagailu, ' = ', S[Kontagailu]:8:3) ;
END ;


VAR
   EkuKop : Integer ;                (* ekuazioen kopurua *)
   Sistema : DM_Taula ;              (* neurriak: EkuKop x EkuKop *)
   Askeak : DM_ElementuAskeak ;      (* neurriak: EkuKop x 1 *)
   Soluzioak : DM_ElementuAskeak ;   (* neurriak: EkuKop x 1 *)
   Determ : Real ;
BEGIN
   SistemaIrakurri (Sistema, Askeak, EkuKop) ;

   WriteLn ('Jatorrizko sistemaren koefizienteak:') ;
   WriteLn ('-----------------------------------') ;
   SistemaErakutsi (Sistema, Askeak, EkuKop) ;

   Determ := Determinante (Sistema, EkuKop) ;

   IF Determ <> 0 THEN
   BEGIN
      SoluzioakBilatu (Sistema, Askeak, Soluzioak, Determ, EkuKop) ;

      WriteLn ('Sistemaren soluzioak:') ;
      WriteLn ('--------------------') ;
      SoluzioakIkusi (Soluzioak, EkuKop) ;
   END
   ELSE
      WriteLn ('Sistemaren ekuazioak konbinazio linealak dira') ;
END.
```

`Cramer` izeneko programa aztertzean ikus daiteke `MAX` konstantea eta `DM_Taula` datu-mota `ArraErag` utitatean definiturik daudela, baina ekuazio bakoitzaren termino askea bektore batean biltegitzea erabaki dugulako `DM_ElementuAskeak` delako datu-mota `Cramer` programa barnean definitu behar izan dugu.

Sistemaren ekuazio kopurua teklatuz irakurri ondoren, sistemaren koefizienteak eta elementu askeak teklatuz ematen dira. Sistema pantailan erakutsi ondoren, koefizienteen determinantea kalkulatzen da sistema ebazgarria den frogatzeko; baiezkoan soluzioak zehaztuko dira `SoluzioakBilatu` prozedurari deituz eta `SoluzioakIkusi` bitartez dagokien pantailaraketa lortuko da.

`Cramer` programak honelako irteerak ematen ditu:

```
ArraErag unitatearen hasieraketa
--------------------------------

Zein da ekuazioen kopurua (1..8)?  3
Sistemaren koefizienteak eman:
[1,1] koefizientea ---->  -1
[1,2] koefizientea ---->  -0.25
[1,3] koefizientea ---->  3.18
1. ekuazioaren balio askea: -4
[2,1] koefizientea ---->  0.2
[2,2] koefizientea ---->  4.17
[2,3] koefizientea ---->  -5
2. ekuazioaren balio askea: 5.1
[3,1] koefizientea ---->  12
[3,2] koefizientea ---->  -7.1
[3,3] koefizientea ---->  9.4
3. ekuazioaren balio askea: -14.5
Jatorrizko sistemaren koefizienteak:
-----------------------------------
 -1.000 -0.250  3.180   --->  -4.000
  0.200  4.170 -5.000   --->   5.100
 12.000 -7.100  9.400   ---> -14 500
Sistemaren soluzioak:
--------------------
E1 =   -0.370
E2 =   -0.449
E3 =   -1.410
_
```

10.5.4.1.2 Gauss-Jordan

Gauss-Jordan metodoa ekuazio linealen sistemak ebazteko biderik zuzenena izan daiteke, eta bere funtsa koefizienteen matrizea diagonalizatzean datza. Garatu dugun Gauss-Jordan metodoaren programan, ez da `ArraErag` unitatea erabiltzen baina Cramerena frogatzeko balio du.

Algoritmoa hobeto ulertzeko sistema zehatz bat erabiliko dugu, adibidez:

$$\begin{aligned} 2\,x_1 + 1\,x_2 - 1.5\,x_3 &= 3.5 \\ 4\,x_1 - 3\,x_2 + 3\,x_3 &= -2.5 \\ -8\,x_1 + 4.2\,x_2 - 1\,x_3 &= 3 \end{aligned}$$

Gauss-Jordan metodoan koefiziente eta termino askeekin array osatu bat lortuko dugu, sistema determinatua eta ebazgarria izan dadin array osatu honen dimentsioak 3x4 izango dira adibidean[23]. Transformazio linealak aplikatuz koefizienteen azpiarray karratuan array identitatea jartzen denean, elementu askeen zutabeak sistemaren soluzioak izango dira:

$$\left(\begin{array}{ccc|c} A_1 & B_1 & C_1 & \boldsymbol{D_1} \\ A_2 & B_2 & C_2 & \boldsymbol{D_2} \\ A_3 & B_3 & C_3 & \boldsymbol{D_3} \end{array}\right) \xrightarrow{\text{transformazio linealak}} \left(\begin{array}{ccc|c} 1 & 0 & 0 & \boldsymbol{E_1} \\ 0 & 1 & 0 & \boldsymbol{E_2} \\ 0 & 0 & 1 & \boldsymbol{E_3} \end{array}\right)$$

Hona hemen Gauss-Jordan metodoaren algoritmoa:

Lerroka lan eginez, lerro bakoitzeko hiru urrtas ematen dira:

[23] Orokorrean matrize osatuak edukiko dituen neurriak *EkuazioKopuru x (EkuazioKopuru+1)* izan da.

1.urratsa Lerroaren pibotea aukeratu. Lerro baten pibotea array diagonalizatuan batekoak izango direnen posizioetako elementuak dira, beraz iterazio kopuruaren arabera digonal nagusiko elementua pibotetzat hartzen da.

2.urratsa Pibotearen lerroa aldatu. Pibotearen lerroak nozitzen duen transformazio lineala eslaka aldaketa izango da, hots, pibotearen lerroa "normalizatu" egiten da bere elementu guztiak (termino askea barne) pibotearen baliogatik zatituz.

3.urratsa Pibotearen zutabea aldatu. Pibotea ez diren elementuak zero bihurtzea izango da zutabearen aldaketa, horretarako arrayaren beste lerroren baten laguntza izango dugu bere konbinazio lineal bat osatuz.

Adibideari lotuz hona hemen Gauss-Jordan metodoaren aplikazioa:

1. iterazioan lerroa: $2\,x_1 + 1\,x_2 - 1.5\,x_3 = 3.5$

1.urratsa Lerroaren pibotea aukeratu: *Pibotea = 2*

2.urratsa Pibotearen lerroa aldatu: *Pibotea = 2*

$$2/_2\,x_1 + 1/_2\,x_2 - 1.5/_2\,x_3 = 3.5/_2$$

Sistema 1. lerroa normalizatu ondoren:

$$\begin{pmatrix} \boxed{1 & 0.5 & -0.75 & 1.75} \\ 4 & -3 & 3 & -2.5 \\ -8 & 4.2 & -1 & 3 \end{pmatrix}$$

3.urratsa Pibotearen zutabea aldatu:

$$\begin{pmatrix} \boxed{1 & 0.5 & -0.75 & 1.75} \\ \mathbf{4} & -3 & 3 & -2.5 \\ -8 & 4.2 & -1 & 3 \end{pmatrix} \longleftarrow$$

*faktorea = -(**4**)*
transformazio lineala: (2. lerroa) + (1.lerroa x faktorea)

*(**4** -3 3 -2.5) + (-4 -2 3 -7)* ⟶ *(**0** -5 6 -9.5)*

Sistema 2. lerroa aldatu ondoren:

$$\begin{pmatrix} \boxed{1 & 0.5 & -0.75 & 1.75} \\ 0 & -5 & 6 & -9.5 \\ \mathbf{-8} & 4.2 & -1 & 3 \end{pmatrix} \longleftarrow$$

*faktorea = -(**-8**)*
transformazio lineala: (3. lerroa) + (1.lerroa x faktorea)

*(**-8** 4.2 -1 3) + (8 4 -6 14)* ⟶ *(**0** 8.2 -7 17)*

Sistema 3. lerroa aldatu ondoren:

$$\begin{pmatrix} \boxed{1 & 0.5 & -0.75 & 1.75} \\ 0 & -5 & 6 & -9.5 \\ 0 & 8.2 & -7 & 17 \end{pmatrix}$$

2. iterazioan lerroa: $0\ x_1 - 5\ x_2 + 6\ x_3 = -9.5$

1.urratsa Lerroaren pibotea aukeratu: *Pibotea = -5*

2.urratsa Pibotearen lerroa aldatu: *Pibotea = -5*

$$0/_{-5}\ x_1 - 5/_{-5}\ x_2 + 6/_{-5}\ x_3 = -9.5/_{-5}$$

Sistema 2. lerroa normalizatu ondoren:

$$\begin{pmatrix} 1 & 0.5 & -0.75 & 1.75 \\ 0 & 1 & -1.2 & 1.9 \\ 0 & 8.2 & -7 & 17 \end{pmatrix}$$

3.urratsa Pibotearen zutabea aldatu:

$$\begin{pmatrix} 1 & \mathbf{0.5} & -0.75 & 1.75 \\ 0 & 1 & -1.2 & 1.9 \\ 0 & 8.2 & -7 & 17 \end{pmatrix} \leftarrow$$

*faktorea = -(**0.5**)*
transformazio lineala: (1. lerroa) + (2.lerroa x faktorea)

*(1 **0.5** -0.75 1.75) + (0 -0.5 0.6 -0.95)* ⟶ *(1 **0** -0.15 0.8)*

Sistema 1. lerroa aldatu ondoren:

$$\begin{pmatrix} 1 & 0 & -0.15 & 0.8 \\ 0 & 1 & -1.2 & 1.9 \\ 0 & \mathbf{8.2} & -7 & 17 \end{pmatrix} \leftarrow$$

*faktorea = -(**8.2**)*
transformazio lineala: (3. lerroa) + (2.lerroa x faktorea)

*(0 **8.2** -7 17) + (0 -8.2 9.84 -15.58)* ⟶ *(1 **0** 2.84 1.42)*

Sistema 3. lerroa aldatu ondoren:

$$\begin{pmatrix} 1 & 0 & -0.15 & 0.8 \\ 0 & 1 & -1.2 & 1.9 \\ 0 & 0 & 2.84 & 1.42 \end{pmatrix}$$

3. iterazioan lerroa: $0\ x_1 + 0\ x_2 + 2.84\ x_3 = 1.42$

1.urratsa Lerroaren pibotea aukeratu: *Pibotea = 2.84*

2.urratsa Pibotearen lerroa aldatu: *Pibotea = 2*

$$0/_{2.84}\ x_1 + 0/_{2.84}\ x_2 + 2.84/_{2.84}\ x_3 = 1.42/_{2.84}$$

Sistema 3. lerroa normalizatu ondoren:

$$\begin{pmatrix} 1 & 0 & -0.15 & 0.8 \\ 0 & 1 & -1.2 & 1.9 \\ 0 & 0 & 1 & 0.5 \end{pmatrix}$$

3.urratsa Pibotearen zutabea aldatu:

$$\begin{pmatrix} 1 & 0 & \mathbf{-0.15} & 0.8 \\ 0 & 1 & -1.2 & 1.9 \\ 0 & 0 & 1 & 0.5 \end{pmatrix} \leftarrow$$

*faktorea = -(**-0.15**)*
transformazio lineala: (1. lerroa) + (3.lerroa x faktorea)

*(1 0 **-0.15** 0.8) + (0 0 0.15 0.075)* ⟶ *(1 0 **0** 0.875)*

Sistema 1. lerroa aldatu ondoren:

$$\begin{pmatrix} 1 & 0 & \mathbf{0} & 0.875 \\ 0 & 1 & -1.2 & 1.9 \\ 0 & 0 & 1 & 0.5 \end{pmatrix} \leftarrow$$

*faktorea = -(**-1.2**)*
transformazio lineala: (2. lerroa) + (3.lerroa x faktorea)

*(0 1 **-1.2** 1.9) + (0 0 1.2 0.6)* ⟶ *(0 1 **0** 2.5)*

Sistema 2. lerroa aldatu ondoren:

$$\begin{pmatrix} 1 & 0 & 0 & 0.875 \\ 0 & 1 & 0 & 2.5 \\ 0 & 0 & 1 & 0.5 \end{pmatrix}$$

Ebatzi nahi den sistema lineala hau izanik:

$$2\,x_1 + 1\,x_2 - 1.5\,x_3 = 3.5$$
$$4\,x_1 - 3\,x_2 + 3\,x_3 = -2.5$$
$$-8\,x_1 + 4.2\,x_2 - 1\,x_3 = 3$$

Hona hemen emaitzak Gauss-Jordan metodoaren algoritmoa sistema horri aplikatu ondoren:

$$\left(\begin{array}{ccc|c} A_1 & B_1 & C_1 & \mathbf{D_1} \\ A_2 & B_2 & C_2 & \mathbf{D_2} \\ A_3 & B_3 & C_3 & \mathbf{D_3} \end{array}\right) \xrightarrow{\text{transformazio linealak}} \left(\begin{array}{ccc|c} 1 & 0 & 0 & \mathbf{E_1} \\ 0 & 1 & 0 & \mathbf{E_2} \\ 0 & 0 & 1 & \mathbf{E_3} \end{array}\right)$$

$$\left(\begin{array}{ccc|c} 2 & 1 & -1.5 & 3.5 \\ 4 & -3 & 3 & -2.5 \\ -8 & 4.2 & -1 & 3 \end{array}\right) \xrightarrow{\text{transformazio linealak}} \left(\begin{array}{ccc|c} 1 & 0 & 0 & \mathbf{0.875} \\ 0 & 1 & 0 & \mathbf{2.5} \\ 0 & 0 & 1 & \mathbf{0.5} \end{array}\right)$$

Eta hauxe litzateke `GaussJordan` programa:

```
PROGRAM GaussJordan ;                              { \TP70\10\GAUSJORD.PAS }

{$DEFINE ArazketaEZ}

CONST
   EKUAZ_KOP_MAX = 10 ;
TYPE
   DM_Bektore = ARRAY[1..EKUAZ_KOP_MAX+1] OF Real ;
   DM_Matrize = ARRAY[1..EKUAZ_KOP_MAX] OF DM_Bektore ;


PROCEDURE Normalizatu (VAR B : DM_Bektore; Pib : Real; Kop : Integer) ;
VAR
   j : Integer ;
BEGIN
  FOR j:=1 TO Kop+1 DO         (* Pibotearen lerroa zati pibotea *)
     B[j] := B[j] / Pib ;      (* egitean, piboteak 1 balioko du *)
END ;
```

```
PROCEDURE KonbibazioLineala (VAR B1 : DM_Bektore;
                             CONST B2 : DM_Bektore;
                             Fakt : Real; Kop : Integer) ;
VAR
   Zutabe : Integer ;
BEGIN
  FOR Zutabe:=1 TO Kop+1 DO
     B1[Zutabe] := B1[Zutabe] + Fakt * B2[Zutabe];
END ;



PROCEDURE SistemaIrakurri (VAR S : DM_Matrize; VAR Kop : Integer) ;
VAR
   i, j : Integer ;
BEGIN
  REPEAT
     Write ('Zein da ekuazioen kopurua (1..', EKUAZ_KOP_MAX, ')?  ') ;
     ReadLn (Kop) ;
  UNTIL (Kop >= 1) AND (Kop <= EKUAZ_KOP_MAX) ;

  (* Sistema konpatiblea eta determinatua izan dadin:
        ekuazioen kopurua = ikogniten kopurua         *)

  WriteLn ('Sistemaren koefizienteak eman:') ;
  FOR i:=1 TO Kop DO
  BEGIN
      FOR j:=1 TO Kop DO
      BEGIN
          Write ('[', i, ',', j, '] koefizientea ---->  ') ;
          ReadLn (S[i, j]) ;
      END ;
      Write(i, '. ekuazioaren balio askea: ') ;
      ReadLn (S[i, Kop+1]) ;
  END ;
END ;



PROCEDURE SistemaErakutsi (CONST S : DM_Matrize; Kop : Integer) ;
VAR
   i, j : Integer ;
BEGIN
  FOR i:=1 TO Kop DO
  BEGIN
      FOR j:=1 TO Kop+1 DO
      BEGIN
          IF j = Kop+1 THEN
            Write ('   ---> ') ;
          Write (S[i, j]:7:3) ;
      END ;
      WriteLn ;
  END ;
END ;



PROCEDURE SoluzioakIkusi (CONST S : DM_Matrize; Kop : Integer) ;
VAR
   Lerro : Integer ;
BEGIN
   FOR Lerro:=1 TO Kop DO
      Writeln('E', Lerro, ' = ', S[Lerro, Kop+1]:8:3) ;
END ;
```



```
PROCEDURE Diagonalizatu (VAR M : DM_Matrize; Kop : Integer;
                         VAR ErroreKode : Integer) ;

(* Matrize batekoz eta zeroz osaturik uzten du, ondorioz sistemaren
   soluzioa azken zutabearen elementuak izango dira               *)

VAR
    Lerro, j, PibLerro : Integer ;
    Pibotea, Faktore : Real ;
BEGIN                                (* Diagonalizatu *)
  ErroreKode := 0 ;
  FOR PibLerro:=1 TO Kop DO          (* Lerroka lan eginez *)
    BEGIN
      Pibotea := M[PibLerro][PibLerro] ;
      IF Pibotea <> 0 THEN
      BEGIN
         Normalizatu (M[PibLerro], Pibotea, Kop) ;
         FOR Lerro:=1 TO Kop DO    (* Pibotearen zutabean zeroak jarri *)
           IF Lerro <> PibLerro THEN
           BEGIN
              Faktore := -(M[Lerro][PibLerro]) ;
              KonbibazioLineala (M[Lerro], M[PibLerro], Faktore, Kop) ;
           END ;

           {$IFDEF ArazketaBAI}
              WriteLn ;
              SistemaErakutsi (M, Kop) ;
              ReadLn ;
           {$ENDIF ArazketaBAI}

      END
      ELSE
        ErroreKode := -1 ;           (* soluzio gabeko sistema *)
    END ;
END ;


                          { PROGRAMA NAGUSIA }

VAR
    EkuKop : Integer ;     (* ekuazioen kopurua *)
    Sistema : DM_Matrize ; (* neurriak: EkuKop x EkuKop+1 *)
    ErroreKodea : Integer ;
BEGIN
  SistemaIrakurri (Sistema, EkuKop) ;

  WriteLn ('Jatorrizko sistemaren koefizienteak:') ;
  WriteLn ('-----------------------------------') ;
  SistemaErakutsi (Sistema, EkuKop) ;

  Diagonalizatu (Sistema, EkuKop, ErroreKodea) ;

  IF ErroreKodea = 0 THEN
  BEGIN
     WriteLn ('Sistema aldatuaren koefizienteak:') ;
     WriteLn ('--------------------------------') ;
     SistemaErakutsi (Sistema, EkuKop) ;

     WriteLn ('Sistemaren soluzioak:') ;
     WriteLn ('--------------------') ;
     SoluzioakIkusi (Sistema, EkuKop) ;
  END
  ELSE
     WriteLn ('Sistemaren ekuazioak konbinazio linealak dira') ;
END.
```

GaussJordan programak honelako irteerak ematen ditu hasierako {$DEFINE } klausula ArazketaEZ denean, ikus daitekeenez Cramer programaren emaitza bera ateratzen da:

```
Zein da ekuazioen kopurua (1..10)?  3
Sistemaren koefizienteak eman:
[1,1] koefizientea ---->  -1
[1,2] koefizientea ---->  -0.25
[1,3] koefizientea ---->  3.18
1. ekuazioaren balio askea: -4
[2,1] koefizientea ---->  0.2
[2,2] koefizientea ---->  4.17
[2,3] koefizientea ---->  -5
2. ekuazioaren balio askea: 5.1
[3,1] koefizientea ---->  12
[3,2] koefizientea ---->  -7.1
[3,3] koefizientea ---->  9.4
3. ekuazioaren balio askea: -14.5
Jatorrizko sistemaren koefizienteak:
-----------------------------------
 -1.000 -0.250  3.180   --->  -4.000
  0.200  4.170 -5.000   --->   5.100
 12.000 -7.100  9.400   ---> -14 500
Sistema aldatuaren koefizienteak:
--------------------------------
  1.000  0.000  0.000   --->  -0.370
  0.000  1.000  0.000   --->  -0.449
  0.000  0.000  1.000   --->  -1.140
Sistemaren soluzioak:
--------------------
E1 =   -0.370
E2 =   -0.449
E3 =   -1.410
_
```

10.6 PROGRAMAK

Hona hemen 10. kapituluaren programak orrialdeen arabera sailkaturik:

Izena	*Programaren identifikadorea*	*ORRI.*	*Ikasgaia*
ARRAY_1.PAS	ArrayBatDatuzBete	10-07	Array datu-mota
ARRAY_2.PAS	ArrayenEsleipena	10-09	Array datu-mota
ARRAY_3.PAS	ArrayenEsleipenOkerra1	10-09	Array datu-mota
ARRAY_4.PAS	ArrayenEsleipenOkerra2	10-10	Array datu-mota
ARRAY_5.PAS	ArrayBiBerdinak	10-11	Array datu-mota
ARRAY_6.PAS	MajuskulenMaiztasunak	10-13	Array datu-mota
ARRAY_7.PAS	ArrayBatenLuzeraFisikoa	10-16	Array datu-mota
ARRAY_8.PAS	ArrayBatenLuzeraLogiko1	10-17	Array datu-mota
ARRAY_9.PAS	ArrayBatenLuzeraLogiko2	10-17	Array datu-mota
ARRAY_10.PAS	R_Direktiba	10-19	Array datu-mota
ARRAY_11.PAS	ArrayBatParametroBezala	10-21	Array datu-mota
ARRAY_12.PAS	ArrayBiBerdinakOteDiren	10-22	Array datu-mota
ARRAY_13.PAS	ArrayDimentsiobakarra	10-25	Array datu-mota
HASIERAK.PAS	AldagaienHasieraketa	10-27	Array datu-mota
ARRAY_14.PAS	ArrayenHasieraketa	10-27	Array datu-mota
ARRAY_15.PAS	LabeaTenperaturazBete	10-29	Array datu-mota
ARRAY_16.PAS	ArrayDimentsioBikoitza	10-33	Array datu-mota
ARRAY_17.PAS	ArrayenHasieraketak	10-35	Array datu-mota
ARRAY_18.PAS	Asamakizuna	10-36	Array datu-mota

ALGOR_01.PAS	IbileraArrayetan	10-39	Array datu-mota
ALGOR_02.PAS	BilaketaLinealaArrayetan	10-42	Array datu-mota
ALGOR_03.PAS	BilaketaBitarraArrayetan	10-46	Array datu-mota
ALGOR_04.PAS	BilaketaBitarraArrayErrepikatuak	10-49	Array datu-mota
ALGOR_05.PAS	TartekaketaArrayetan	10-51	Array datu-mota
ALGOR_06.PAS	EzabaketaArrayetan	10-53	Array datu-mota
ALGOR_07.PAS	NahasketaArrayetan	10-57	Array datu-mota
ORDEN_01.PAS	OrdenaAukeraketa	10-59	Array datu-mota
ORDEN_02.PAS	OrdenaTartekaketaSekuentzialki	10-62	Array datu-mota
ORDEN_03.PAS	OrdenaTartekaketaBitarra	10-64	Array datu-mota
ORDEN_04.PAS	OrdenaBurbuila	10-67	Array datu-mota
ORDEN_05.PAS	OrdenaBurbuilaHobetua	10-69	Array datu-mota
ARR-AL_0.PAS	ArrayenAlderantzizkoa	10-73	Arrayen unitatea
ARR-AL_B.PAS	ArrayenAlderantzizkoa_OROKORRA_2_8	10-74	Arrayen unitatea
ARRAERAG.PAS ARRAY__1.PAS	ArraErag ArrayenEragiketaAritmetikoak1	10-84	Arrayen unitatea
ARRAERAG.PAS ARRAY__2.PAS	ArraErag ArrayenEragiketaAritmetikoak2	10-84	Arrayen unitatea
ARRAERAG.PAS ARRAY__3.PAS	ArraErag ArrayenEragiketaAritmetikoak3	10-84	Arrayen unitatea
ARRAERAG.PAS ARRAY__4.PAS	ArraErag ArrayenEragiketaAritmetikoak4	10-84	Arrayen unitatea
ARRAERAG.PAS ARRAY__5.PAS	ArraErag ArrayenEragiketaAritmetikoak5	10-84	Arrayen unitatea
CRAMER.PAS	Cramer	10-86	Arrayen aplikazioa
GAUSJORD.PAS	GaussJordan	10-91	Arrayen aplikazioa

10.7 BIBLIOGRAFIA

- Borland, *TURBO PASCAL 7.0. Erabiltzailearen gida*, Borland International, 1992
- Borland, *TURBO PASCAL 7.0 Ingurunearen laguntza*, Borland International, 1992
- Joyanes, *TURBO PASCAL 6.0 a su alcance*, McGraw-Hill, 1993

KONTZEPTUEN INDIZE ALFABETIKOA

—E—

—F—

—G—

—H—

—I—

—K—

—L—

—M—

—N—

—O—

—P—

—R—

—S—

—T—

—U—

—V—

—W—

—X—

—Z—