eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

BILBOKO INGENIARIEN GOI ESKOLA TEKNIKOA

KONPUTAGAILUEN PROGRAMAZIOA
TURBO PASCAL BITARTEZ

II

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

4. ATALA: TURBO PASCAL 7.0 LENGOAIAREN ELEMENTUAK

AURKIBIDEA

4.1 SARRERA

Lengoaia natural bat ikasten dugunean estrategia bi daude, batetik gramatika eta lengoaiaren barne arauak sistematikoki menperatu eta ondoren lengoaia erabili, bestetik hasiera-hasieratik lengoaia erabili nahiz eta lengoaiaren barne egiturak erabat ezagunak ez izan.

Programazio lengoaien ikaste prozesuan ere, aurreko lerroan esandakoak garrantzia du eta liburu honetan bigarren bidea erabiliko dugu nagusiki. Hau da ekinean ikasiko dugu programatzen, kontzeptuak adibide-programen bitartez azalduz.

4.2 LENGOAIAREN FUNTSEZKO ELEMENTUAK

Dakigunez programa bat sententzien multzo bat da, eta sententziak funtsezko elementuetan bana daitezke. Programa bateko sententzietan berezko esanahia duten elementu horiei literatura anglosaxoian *token* esaten zaie eta gainerako hizkuntzen hiztunok hitz hori inportatu egin dugu. Token arteko bereizketa egiteko, zenbaitetan, token bi artean banatzaileren bat jarri behar da, banatzailerik erabiliena zuriunea da baina tabuladorea eta orga-itzulera onargarriak dira ere.

Programa bat sententzien multzoa bada eta sententzia bat zenbait token bilduma dela onarturik, oso sinplistak izanik, programa bat hainbat token multzoa bezala ikus daiteke.

Hona hemen Turbo Pascal lengoaian emandako sententzia bat. Zertarako balio duen jakin gabe, bere itxura azter dezagun token ezberdinak identifikatuz:

```
Luzera := (Diametroa DIV 2) * PI ;
```

Sententzian agertzen diren tokenak 10 dira, banan-banan zerrendaturik hauek dira:

Luzera	:=	(	Diametroa	DIV	2	)	*	PI	;
1	2	3	4	5	6	7	8	9	10

Token sailkapena egitean, lau motatakoak ager daitezkeela jakin behar da:

1. Ikur bereziak edo sinboloak. Adibidez: `:=` `(` `)` `*` eta `;`
2. Identifikadoreak. Adibidez: `Luzera` `Diametroa` eta `PI`
3. Hitz erreserbatuak. Adibidez: `DIV`
4. Konstanteak. Adibidez: `2`

Sententzian derrigorrezkoak diren banatzaileak bi dira: `Diametroa` eta `DIV` artean dagoen zuriunea batetik, eta bestetik `DIV` eta `2` artean dagoena. Hala ere, eta irakurgarritasuna zaintzearren beste zuriune batzuk tartekatu dira.

4.2.1 Hitz erreserbatuak eta sinboloak

Hitz erreserbatuak, Turbo Pascal konpiladoreak aurredefiniturik dituen elementuak dira. Hitz erreserbatuek ingelesetik hartutako terminoak dira eta konpiladoreak esanahia zehatz eta berezia ezagutzen die. Hitz erreserbatuak lengoaiaren primitiboak liratekeenez programak egiteko ezinbestekoak dira.

Hitz erreserbatuak, banatzaile gabeko karaktere-kateak dira. Konpiladoreak ez du maiuskula eta minuskula artean bereizketa egiten baina ohitura da hitz erreserbatuak maiuskuletan idaztea. Hitz erreserbatuen aplikazioak lau dira:

- `BEGIN, END, FUNCTION, PROGRAM, ...` blokeak definitzeko
- `DIV, MOD, AND, OR, NOT` eta `XOR` operazioak egiteko
- `FILE, ARRAY, STRING, RECORD, ...` datu-egiturak definiteko
- `IF, REPEAT, WHILE, DO, FOR, ...` kontrol egiturak definiteko

Turbo Pascal 7.0 konpiladoreak ezagutzen dituen hitz erreserbatu guztiak jarraian ematen den taulan biltzen dira:

```
AND          ARRAY        ASM          BEGIN        CASE         CONST
CONSTRUCTOR  DESTRUCTOR   DIV          DO           DOWNTO       ELSE
END          FILE         FOR          FUNCTION     GOTO         IMPLEMENTATION
IF           IN           INHERITED    INLINE       INTERFACE    LABEL
MOD          NIL          NOT          OBJECT       OF           OR
PACKED       PROCEDURE    PROGRAM      RECORD       REPEAT       SET
SHL          SHR          STRING       THEN         TO           TYPE
UNIT         UNTIL        USES         VAR          WHILE        WITH
XOR
```

Ondoren agertzen diren karaktereak ikur bereziak dira konpiladorerako, eta hitz erreserbatuen antzera aurredefinituriko zereginak dituzte.

+	Batuketa
-	Kenketa edo minus zeinua
*	Biderkaketa
/	Zatiketa erreala
=	Berdintasuna
<	Txikiago baino
>	Handiago baino
[	Array datu-motaren dimentsioaren muga
]	Array datu-motaren dimentsioaren muga
.	Erregistroetan eremuen arteko banatzailea
,	Identifikadore arteko banatzailea
(	Prozedura eta funtzioetan parametro zerrendaren muga
)	Prozedura eta funtzioetan parametro zerrendaren muga
:	Identifikadore eta datu-mota arteko banatzailea
;	Agindu amaieraren marka
‘	Karaktere-katearen muga
^	Erakuslea
@	Eragigai baten helbidea
{	Iruzkinaren muga edo konpiladorearen direktiba
}	Iruzkinaren muga edo konpiladorearen direktiba
$	Jarraian doazen karaktereak kode hamaiseitarran daude
#	Jarraian doazen karaktereak ASCII kodean daude

Badira zenbait sinbolo bi karakterez adierazten direnak, ondokoak:

<=	Txikiado edo berdin
>=	Handiago edo berdin
:=	Esleipena
..	Balioen heina
(*	Iruzkinaren muga edo konpiladorearen direktiba, { bezalakoa
*)	Iruzkinaren muga edo konpiladorearen direktiba, } bezalakoa
(.	Array datu-motaren dimentsioaren muga, [bezalakoa
.)	Array datu-motaren dimentsioaren muga,] bezalakoa

4.2.2 Identifikadoreak

Lengoaiaren elementuak izendatzeko erabiltzen diren etiketak dira identifikadoreak. Identifikadoreak edozein luzerako karaktere-kateak dira baina lehenengo hirurogeitahiruak esanguratsuak dira. Derrigorrez letra batez edo _ karakterez hasiko dira. Identifikadoreetan maiuskulak eta minuskulak ez dira bereizten.

Identifikadoreek izendatzen dituzten elementuak hauek dira: aldagaiak, konstanteak, datu-motak, programak, prozedurak, funtzioak, unitateak eta erregistroen eremuak.

4.2.2.1 Identifikadore estandarrak

Turbo Pascal lengoaiak identifikadore batzuk ezagutzen ditu, aurredefiniturtik daudelako. Horiei identifikadore estandarrak esaten zaie eta edozein programatan erabil daitezke.

Konstanteak	`MAXINT, PI, MAXLONGINT, TRUE, ...`
Datu-motak	`Integer, Real, Boolean, Byte, Char, ...`
Prozedurak	`ReadLn, WriteLn, New, Assign, Delete, ...`
Funtzioak	`Cos, Abs, Eof, Pi, Length, ...`
Unitateak	`Crt, Graph, Dos, System, ...`

4.2.2.2 Erabiltzailearen identifikadoreak

Identifikadore estandarrak ez bezala, erabiltzailearen identifikadoreak programadore batek programa jakin baterako asmatu eta definitu duen etiketa da. Identifikadoreak finkatzeko arauak berriro gogoratuz:

1. Identifikadorearen luzera edozein izan daiteke
2. Esanguratsuak lehenengo 63 karaktereak dira
3. Lehenengo karakterea letra bat edo azpimarra karakterea _ izango da
4. Bigarren eta hurrengo karaktereak ondokoak izan daitezke:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
_
0 1 2 3 4 5 6 7 8 9
```

5. Maiuskulak = Minuskulak
6. Hitz erreserbatuak ezin dira identifikadore bezala erabili

Ikus ditzagun programa batetik hartutako identifikadoreak, batzuk baliagarriak dira beste batzuk berriz programa konpilatzerakoan ez dira onartuko:

Onargarriak	Onartezinak	
`Adina`	`Begin`	(hitz erreserbatua da)
`Deitura_1`	`1_Deitura`	(zenbakiz hasten da)
`BaturaLortu`	`Batura Lortu`	(zuriunea banatzailea da)
`Batura_Lortu`	`Batura-Lortu`	(- token bat da)

4.2.3 Konstanteak

Programa batean aldatzen ez diren balioak konstanteak deitzen dira. Konstanteak programetan bi modutan ager daitezke, balioak zuzenki ager daitezke edo konstanteari lotu zaion identifikadore baten bitartez. Hurrengo adibideetan konstanteak ematen dira:

`109`	zenbakizko konstante bat (zenbaki osoa)
`4.082`	zenbakizko konstante bat (zenbaki erreala)
`7E-2`	zenbakizko konstante bat (zenbaki erreala)
`'M'`	karaktere konstante bat
`'kaixo !'`	karaktereen kate konstantea

Hona hemen konstanteen erabilpena programa[1] batean:

```
PROGRAM KonstanteaPantailaratzen ;          { \TP70\04\MEZUA.PAS }
BEGIN
     WriteLn ('kaixo !') ;
END.
```

Programa honek, pantailaraketa bat eskatzen duen, agindu bakarra du. Programa exekutatzean ondokoa agertuko da pantailan:

```
kaixo !
_
```

Baina konstanteen balioak zuzenki erabili beharrean, askotan identifikadore batez izendatzen dira horretarako ondoko programa azal dezagun:

```
PROGRAM KonstanteakPantailaratzen ;          { \TP70\04\MEZUAK.PAS }
CONST
     ADINA = 37 ;
     ABIZEN_1 = 'Salazar' ;
BEGIN
     WriteLn (ABIZEN_1, ' ===> ', ADINA, ' urte') ;
END.
```

Aurreko programarekin alderatuz bigarren honek kontzeptu berri bi gehitzen dizkio lehenengoari. Batetik `WriteLn` bakar batek gauza bat baino gehiago pantailara dezakeela baldin eta parentesi barnean parametroak jarri eta komaz banatzen badira (honetaz gehiago luzatuko gara **4.5.1** puntuan). Eta bestetik, nahiz eta `WriteLn` prozedurak dituen lau parametroak konstanteak izan modu ezberdinean erabili dira, `ADINA` eta `ABIZEN_1` konstanteak explizitoki identifikadore banaz definitu dira (ondorioz, memorian `37` eta `'Salazar'` balioak gorde dira).

Bigarren programaren pantailaraketa hau litzateke:

```
Salazar ===> 37 urte
_
```

Konstanteak definitzeko blokea `CONST` hitz erreserbatuaren bitartez markatzen da eta programaren hasieran jartzen da. Bloke horren bidez `KonstanteakPantailaratzen` izeneko programan `37` eta `'Salazar'` balioak memorian gordetzen dira. Geroago, programan balioak erabili behar direnean bakoitzari dagokion identifikadoreaz erreferentzia egitea aski da. Kontutan izan, ohituraz konstanteen identifikadoreak maiuskuletan jartzen direla.

[1] Nahiz eta Turbo Pascal programa baten egitura eta sarrera/irteerako prozedurak oraindik ezagutu ez, eta guztia erabat ulertu ez, une honetan hutsuneak geratu arren adibidearen komenigarritasunen sinesturik gaude.

4.2.4 Aldagaiak

Programa baten exekuzioan datu batzuk ez dira aldatzen baina beste batzuek balio ezberdinak har ditzakete. Alda daitezkeen datuak aldagaien bitartez adierazten dira, eta funtsean aldagai batek memoriaren zati zehatz bat erabiltzea ahalbideratzen du.

Aldagaiak definitzeko ondoko lerroak idazten dira, `VAR` bloke batean aldagaien identifikadoreak eta aldagaien datu-motak agertuko dira:

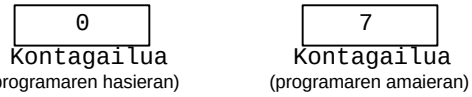
```
VAR
      Kontagailua : Integer ;
      Erantzuna : Char ;
      Aurkitua, Amaiturik : Boolean ;
      Emaitza, ErroreMaximoa : Real ;
```

Ikus daitekeenez aldagaien identifikadoreak aukeratzerakoan, programadoreak maiuskulak eta minuskulak erabil ditzake. Adibidean dauden `Integer`, `Char`, `Boolean` eta `Real` identifikadore estandarrak datu-motak dira eta **4.3** puntuan aztertuko ditugu.

Hona hemen aldagai bat darabilen gure lehenengo programa:

```
PROGRAM AldagaiaPantailaratzen ;                    { \TP70\04\ALDAGAI.PAS }
VAR
      Kontagailua : Integer ;
BEGIN
      Kontagailua := 0 ;
      WriteLn ('Hasieran Kontagailua-ren balioa = ', Kontagailua) ;
      Kontagailua := 7 ;
      WriteLn ('Amaieran Kontagailua-ren balioa = ', Kontagailua) ;
END.
```

`VAR` blokearen bitartez aldagaia definitu ondoren, programan hasierako balio bat ematen zaio eta balio hori alda daiteke programa exekutatzen den bitartean. Aldagai baten nozioa ez da ahaztu behar: aldagai oro datuak gordetzeko "kutxa" bat da, eta askotan kontzeptu bi nahastu egiten dira; batetik aldagaiaren identifikadorea eta bestetik aldagaiak memorian gordetzen duen datua; hau da, adibidearen lehenengo aldagaiari dagokion izena `Kontagailua` da eta hasieran gordetzen duen balioa `0` izan daiteke, baina programaren amaieran (aldagaieren identifikadorea berdina izanik) memoriko toki horretan kokaturik dagoen balioa `7` da. Balioak emateko `:=` esleipen operazioa **4.2.6** puntuan azaltzen da.



Datuaren balioa memorian pilatzen dena da, hots, aldagaiaren edukia, eta esan den bezala programaren ibileran memoriko gelaska horietan gorde daitekeena balio ezberdinak izan daitezke. Aldagaiaren identifikadoreak berriz, memoriko helbidearen ideia ematen digu eta datua memorian gordetzeko (zein memoriatik eskuratzeko) lana betetzen du, dakigunez memoriko gelaska baten helbidea aldaezina da. Horrela ulertzen da programaren exekuzioa:

```
Hasieran Kontagailua-ren balioa = 0
Amaieran Kontagailua-ren balioa = 7
_
```

`WriteLn` prozedurak bi parametro ditu konstante bat lehena eta aldagai bat bigarrena, konstanteak idatzirik dauden bezala agertzen dira baina aldagaia pantailaratzerakoan aldagaiaren balioa erakutsi behar du `WriteLn` prozedurak. Aldagaia bera da, lehenengo zein bigarren `WriteLn` prozeduratan daudenak (helbide bera memorian, baina ez edukiera berdina: hasieran `0` eta `7` amaieran).

4.2.5 Iruzkinak

Programa baten sententziak egiten dutena hobeto ulertzeko oharrak edo iruzkinak tartekatzen dira. Iruzkinak testuak izango dira eta konpiladoreak aintzat har ez ditzan testuaren hasieran giltza karakterea `{` jartzen da eta amaieran bere simetrikoa `}`. Sinboloak aipatu ditugunean esandakoa gogoratuz (**4.2.1** puntua), giltzen ordez `(*` eta `*)` karaktere bikoteak erabil daitezke.

```
(* Iruzkin baten adibidea *)
{ Beste iruzkin baten adibidea }
{ Iruzkin kabiatuak { bat bestearen barnean } ez dira onartzen }
(* Iruzkin kabiatuak (* bat bestearen barnean *) ez dira onartzen *)
{ Iruzkinak kabiatzeko (* bat bestearen barnean *) sinboloak konbinatu }
(* Iruzkinak kabiatzeko { bat bestearen barnean } sinboloak konbinatu *)
```

4.2.6 Esleipena

Esleipen sententzia oinarrizko agindua da, honen bitartez aldagai batek duen balioa espresio baten emaitzagatik ordezkatzen da. Esleipen batean beraz, alde bi izango dira ezkerraldean balio berria hartuko duen aldagaia, eskuinaldean espresioa (espresioaren datu-mota helburuko aldagaiarekiko koherentea izango da). Zati bien artean `:=` sinboloa agertuko da.

Esleipen agindua exekutatzean eskuinetik ezkerrera egiten da, hasteko eskuinean dagoen espresioa ebaluatzen da, eta bere emaitza ezkerraldean dagoen aldagaiak erreferentziatzen duen memoriko txokoan gordetzen da (memoriko posizio horietan aurretik zegoena galduz). Horra hor zenbait esleipen:

```
Kontagailua := 0 ;                          { 1. esleipena }
Kopurua := Kontagailua ;                    { 2. esleipena }
Kontagailua := Kontagailua + 1 ;            { 3. esleipena }
Salneurria := Kostua + Kostua * bez /100 ;  { 4. esleipena }
Izena := 'Eguzkine' ;                       { 5. esleipena }
y := sin(x) ;                               { 6. esleipena }
```

`1.` eta `5.` adibideetan eskuineko adierazpenak konstanteak dira, konstante bat bere baliora ebaluatzen denez `0`-ren emaitza zero izango da eta horixe gordetzen da `Kontagailua` identifikadoreak markatzen duen memoriko gelasketan.

`2.` adibidean espresioa aldagai bat da, eta aldagai bat duen baliora ebaluatzen da, horregatik eta `Kontagailua` aldagaiak zero balio duenez `Kopurua` aldagaiak ere zero balioko du `2.` esleipen hori exekutatzean.

`3.` eta `4.` adibideetan eskuineko espresioak adierazpen aritmetikoak dira, horiek ebaluatu ondoren euren emaitza ezkerreko aldagaiek erreferentziatzen dituzten gelasketan gordeko dira, `3.` adibidean agertzen den moldea askotan agertzen denez astiro azter dezagun: `1.` esleipena kontutan izanik `Kontagailua` aldagaiak zero balio du, eta `Kontagailua+1` espresioaren emaitza bat izango da, bateko hori gordetzeko `Kontagailua` aldagaiak adierazten duen posizioan egiten denez, azkenean honek izango duen balioa `1` izango da eta `Kontagailua` aldagaiaren inkrementua litzateke.

`6.` adibidean Turbo Pascal lengoaiak aurredefiniturik duen `sin()` funtzio estandarra agertzen da, `x` aldagaia berak gordetzen duen baliora ebaluatu ondoren `sin()` funtzio trigonometrikoak `x` balioari dagokion sinua itzultzen du, zein zenbakizko balio bat den eta `y` aldagaiak markatzen duen memoriko posizioetan gordetzen den.

4.3 AURREDEFINITURIKO DATU-MOTAK

Bigarren kapituluan ordenadorearen memoria eta informazioaren biltegitzea memorian aztertu genituen. Konkretuki, sinboloen adierazpidea eta kopuruen adierazpidea ikusi genituen, azkeneko honetan zenbaki osoen eta zenbaki errealen adierazpideak bereiziz. Bigarren kapituluan esandakoa kontutan izanik, orain Turbo Pascal 7.0 programazio lengoaiak eskaintzen dituen biltegitze mekanismoak ikasiko ditugu.

Edozein lengoaiak aurredefinituriko dituen datu-motak bi zatiz osatuak agertzen dira:

1. Aurrez determinaturiko memorian biltegitzeko sistema. Adibidez eta kurtsoaren bigarren kapitulua gogoratuz, kopuru frakzionatuak adierazteko konputazio sistema jakin batean “koma higikorra” metodoa erabil daiteke.
2. Datu-mota horrentzat egokiak diren eragiketak definiturik egongo dira. Ongi mugatutako eragiketak kolekzio bat osatzen dute, operaziok sintaxia naturala gordetzen dutenez euren erabilpen erraza ziurtaturik dago. Esate baterako, kopuru frakzionatuen datu-motarako definiturik izango diren eragiketak batuketa, kenketa, biderkaketa eta zatiketa izan daitezke.

4.3.1 Datu-mota osoak

Bost dira aurredefiniturik dauden datu-mota osoak: `Shortint`, `Integer`, `Longint`, `Byte` eta `Word`. Datu-mota bakoitza zenbaki osoen multzo bat dela onar daiteke, eta ondoko taulan datu-mota bakoitzeko multzoaren behemuga eta goimuga biltzen dira:

Datu-mota	Heina[2]	Formatua
`Shortint`	-128 .. 127	8 bit eta zeinuarekin
`Integer`	-32768 .. 32767	16 bit eta zeinuarekin
`Longint`	-2147483648 .. 2147483647	32 bit eta zeinuarekin
`Byte`	0 .. 255	8 bit eta zeinurik gabe
`Word`	0 .. 65535	16 bit eta zeinurik gabe

Gure ariketa gehienetan `Integer`, `Longint` eta `Byte` datu-motak erabiliko ditugu eta goimugak gogoratu behar izatea ekidin asmoz, Turbo Pascal lengoaiak aurredefiniturik ditu `MAXINT` eta `MAXLONGINT` konstanteak.

Programazio lengoaia gehienetan zenbaki osoen adierazpiderako, bigarren kapituluan esan zen bezala, *birako osagarria* delako notazioa erabiltzen da, Turbo Pascalak ere *birako osagarria* darabil eta hori experimentalki baieztatzeko programa hau exekuta daiteke:

```
PROGRAM ZenbakiOsoenBarneAdierazpidea ;              { \TP70\04\BARNE1.PAS }
VAR
   Zenbakia, i : Integer ;
   Konta : Word ;
BEGIN
    Write ('Zenbaki osoa eta positiboa sartu: ') ;
    ReadLn (Zenbakia) ;
    Konta := MAXINT+1 ;
    Write (' ', Zenbakia, 'ren barne adierazpidea:  ');
```

[2] Datu-mota bakoitzari dagozkion behemuga eta goimugaren balioen zergatia bigarren kapituluan azaldu zen.

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

    Zenbakia := -Zenbakia ;
    Konta := MAXINT+1 ;
    Write (Zenbakia, 'ren barne adierazpidea:  ');
    FOR i:= 16 DOWNTO 1 DO
    BEGIN
        IF (Konta AND Zenbakia) = 0 THEN Write ('0')
        ELSE Write ('1') ;
        Konta := Konta DIV 2 ;
        IF (i+3) MOD 4 = 0 THEN Write (' ')
    END;
    Writeln
END.
```

Programa exekutatzean, adibidez hiru zenbaki osoa sartzen badugu `3` eta `-3` zenbakien barne adierazpidea pantailaratuko ditu. Ikus daitekeenez bien arteko erlazioa *birako osagarria* da:

```
Zenbaki osoa eta positiboa sartu: 3
 3ren barne adierazpidea:  0000 0000 0000 0011
-3ren barne adierazpidea:  1111 1111 1111 1101
_
```

`ZenbakiOsoenBarneAdierazpidea` programa osorik ulertzeko Turbo Pascal nahikorik ez dakaigu oraindik. Aurrerago, une egokia izango denean, berriro ekingo diogu.

4.3.1.1 Zenbaki osoen eragileak

Datu-mota osoek ezagutzen dituzten operazioak hamabi dira (sei operadore aritmetiko eta erlaziozko beste sei operadore).

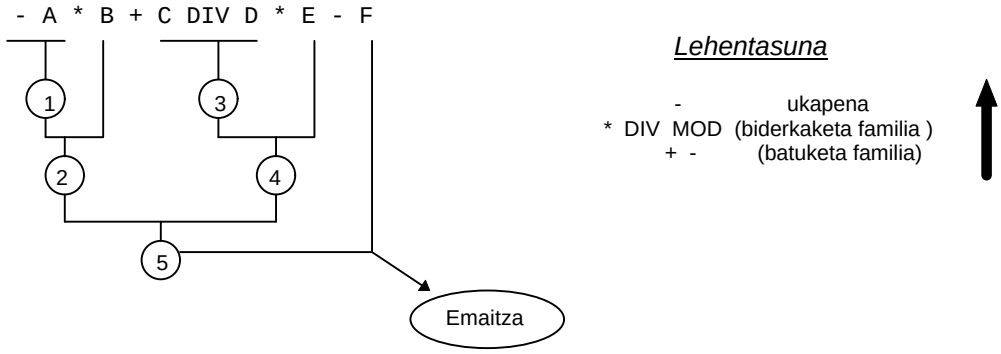
Operadore aritmetikoak, aurreratu denez, sei dira eta adierazpen aritmetikoak sortzeko ezinbestekoak dira. Adierazpen aritmetiko batean konstanteak, aldagaiak eta funtzioen emaitzak eragile aritmetikoen bitartez konbinatzen dira, esate baterako ondoko adibidean ematen den adierazpen aritmetikoan agertzen diren tokenak 12 dira (aldagai biren identifikadoreak, konstante bi, funtzio baten identifikadorea eta horiek konbinatzen dituzten hiru eragile aritmetiko eta parentesi sinboloak).

```
(Diametroa DIV 2) * PI + abs(Tentsioa)
```

Hurrengo taulan zenbaki osoekin erabil daitezkeen sei eragile aritmetikoak biltzen dira:

Eragilea	Eragiketa	Adibidea
+	Batuketa	zbk + 1
-	Kenketa	zbk - 2
-	Ukapena	- zbk
*	Biderkaketa	zbk * 3
DIV	Zatiketa osoa	zbk DIV 4
MOD	Zatiketa osoaren hondarra	zbk MOD 5

Sei eragile horien artean hierarkia bat dago, lehenengo biak batuketaren familiakoak dira eta azken hirurak biderkaketa familia osatzen dute, ukatzaile berezeiagoa da eta isolaturik bezala kontsideratuko dugu. Adierazpen aritmetikoak konplexuak eta luzeak izan daitekeela kontutan izan dezagun, horrelakoetan biderkaketa familiako eragileak lehentasuna daukate batuketa familiako eragileen aurrean eta ukatzailea guztien aurretik dago. Horregatik ondoko adierazpen aritmetikoan eragiketak zenbakien ordenez burutuko dira:



Dagoen ukatzailea aplikatuko da aurrena, beraz `A` identifikadorerari dagokion balioaren ukapena egiten da. Gero biderkaketak egingo dira, ukapenaren bitarteko emaitza hori eta `B` identifikadorerari dagokion balioen arteko biderkadura lortzen da; ondoren `C` eta `D` arteko zatiketa osoa kalkulatzen da eta honi `E`-k balio duena biderkatzen zaio. Amaitzeko batuketak osatuko dira, 2 gehi 4 alde batetik eta 5 ken `F` bestetik eginez.

Adibidean ikus daitekeenez familia bereko eragiketak ezkerretik eskuinerantz burutzen dira, eta azkenean adierazpen aritmetiko batek emaitza bakar bat eskainiko du (eragigai guztiak zenbaki osoak direnez eta eragileak zenbaki osoen operadoreak direnez, emaitza ere zenbaki osoa izango da).

Familien arteko lehentasuna aldatzeko parentesiak erabiltzen dira. Hona hemen eragileen lehentasun arauak:

1. Lehenengo parentesi arteko azpiadierazpenak ebaluatzen dira. Parentesiek beste parentesi kabiatuak barneratzen badituzte, sakonago dauden azpiadierazpenak lehenik ebaluatuz kanporantz abiatuko gara.
2. Eragilearen ordena, dakigunez, familiaren arabera ordenatzen da (ukapena aurrena, biderkaketak ondoren, eta, batuketak amaitzeko).
3. Lehentasun bereko eragileak aurkitzean, ezkerrean daudenak ebaluatzen dira lehenago eta ondoren eskuinean daudenak ebaluatuko dira.

Jarraian, `OsoenAdierazpenAritmetikoak` programaren bitartez `Emaitza` aldagaiaren balioa kalkulatzen da. Programa hori oinarritzat harturik eta ondoko aldaketak eginez, kasu bakoitzean aterako litzatekeen emaitzaren balioa lehenik igarri eta ostean kalkulatu:

```
-(Diametroa DIV 2) * R - (T MOD R + (4 * S)) - Diametroa DIV 3

- Diametroa DIV 2 * R - (T MOD R + 4 * S) - Diametroa DIV 3

-(Diametroa DIV (2 * R) - (T MOD (R + 4) * S) - Diametroa DIV 3

-(Diametroa DIV 2) * R - (T MOD R + 4 * S)) - Diametroa (DIV 3)

-(Diametroa DIV 2) * R - (T MOD R + (4 * S)) - Diametroa DIV 3

-(Diametroa / 2) * R + (T MOD R + (4 * S)) - Diametroa DIV 3

- Diametroa DIV 2 * R + (-T MOD R + 4 * -S) + Diametroa DIV -3
```

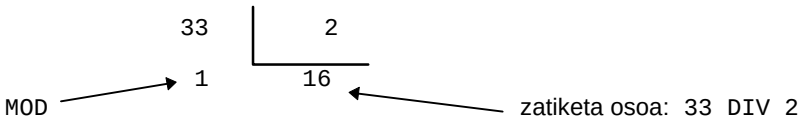
```
PROGRAM OsoenAdierazpenAritmetikoak ;                { \TP70\04\ESPRE1.PAS }
CONST
      R = 4 ;
      S = 5 ;
      T = 27 ;
VAR
      Diametroa, Emaitza : Integer ;
BEGIN
      Diametroa := 33 ;
      Emaitza := -(Diametroa DIV 2) * R - (T MOD R + 4 * S) - Diametroa DIV 3 ;
      WriteLn ('Emaitza = ', Emaitza) ;
END.
```

`Diametroa` eta konstanteen balioak aintzat harturik `Emaitza` aldagaian gordetzen dena adierazpen aritmetikoa ebaluatuz lortzen da. Honela:

Bi parentesi daude eta biak maila berekoak, beraz ezkerrean dagoena ebatziko da lehenago.

```
-(16) * R - (T MOD R + 4 * S) - Diametroa DIV 3
```

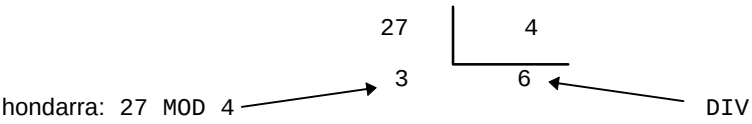
`DIV` eragileak zenbaki osoen zatiketa ematen duenez `33` zati `2` hamasei eta erdi da, baina emaitza osoa izan behar duenez `16`arekin geratzen gara. Gainerako hondarra lortzeko `MOD` eragilea aplikatuko litzateke.



Eta bigarrena gero. Bigarren parentesia ebaluatzean, biderkaketa familiako eragiketak batuketa familiakoaren aurretik burutzen dira azkenean `23` emanez

```
-(16) * R - (3+20) - Diametroa DIV 3
```

`MOD` eragileak zenbaki osoen zatiketaren hondarra ematen duenez `27` zati `4` sei baino gehiago da. Geratzen den hondarra `27-(4*6) = 3` litzateke.



Ondoren ukapena burutuko da.

```
-16 * R - 23 - Diametroa DIV 3
```

Orain biderkaketa familiako eragiketak egingo dira. Ezkerragora dagoen biderkaketa lehenik eta zatiketa osoa geroago.

```
-64 - 23 - Diametroa DIV 3
-64 - 23 - 11
```

Amaitzeko, batuketa familiako eragiketak (ezkerretik eskinera ere).

```
-87 - 11
-98
```

Zenbaki osoekin adierazpen aritmetikoak sortzeaz gain, adierazpen boolearrak ere sor daitezke. Adierazpen boolear baten emaitzak bi balioen artean bat izango du (horregatik adierazpen boolearrei adierazpen logiko esaten zaie), Turbo Pascal lengoaian adierazpen

boolear batek FALSE ala TRUE balioak har ditzake. Erlaziozko eragileak aritmetikoen ostean ebaluatzen dira.

Hurrengo taulan osoekin erabil daitezkeen erlaziozko sei eragileak biltzen dira:

Eragilea	Eragiketa	Adibidea
>	Handiago baino	zbk > 1
<	Txikiago baino	zbk < 2
>=	Handiago edo berdin	zbk >= 3
<=	Txikiago edo berdin	zbk <= 4
=	Berdin	zbk = 5
<>	Desberdin	zbk <> 6

Erlaziozko eragileak aurrerago erruz erabiliko ditugu.

4.3.1.2 Zenbaki osoen gainezkada

Sistema fisikoak, ordenadorea kasurako, finitoak eta mugatuak direlako zenbakien adierazpidean gainezkadaren arazoa suertatzen dela bigarren kapituluan behin eta berriz aipatu genuen. Orain eta Turbo Pascal lengoaiaren daukagun hastapeneko ezagutzarekin experimentalki frogako dugu. Baina lehenago arazoaren zergaitia bergogora dezagun.

Turbo Pascaleko Byte datu-motarako memorian 8 bit erreserbatzen dira eta 2^8=256 bit konbinazio guztiak zenbaki osoak eta positiboak adierazteko erabiltzen dira. Beraz eta zeroa adieraztea erabakitzen bada, zenbaki oso handiena 255 izango da. Hurrengo taula begira ezazu:

Zenbakia	Memorian	Balioa
0	0 0 0 0 0 0 0 0	0 + 0 + 0 + 0 + 0 + 0 + 0 + 0
1	0 0 0 0 0 0 0 1	0 + 0 + 0 + 0 + 0 + 0 + 0 + 1
2	0 0 0 0 0 0 1 0	0 + 0 + 0 + 0 + 0 + 0 + 2 + 0
3	0 0 0 0 0 0 1 1	0 + 0 + 0 + 0 + 0 + 0 + 2 + 1
254	1 1 1 1 1 1 1 0	128 + 64 + 32 + 16 + 8 + 4 + 2 + 0
255	1 1 1 1 1 1 1 1	128 + 64 + 32 + 16 + 8 + 4 + 2 + 1

Byte datu-motako bi zenbakien arteko eragiketa aritmetikoa egitean lortzen den emaitza, muga den 255 balioa baino handiagoa izan daiteke (gainezkada gertatu dela esaten da horrelakoetan). Adibidez, hurrengo programan pantailaratzen den emaitza zein litzateke?

```
PROGRAM BytenGainezkada ;                                { \TP70\04\GAINEZ1.PAS }
VAR
      zbk_1, zbk_2, Emaitza : Byte ;
BEGIN
      zbk_1 := 250 ;
      zbk_2 := 9 ;
      Emaitza := zbk_1 + zbk_2 ;
      WriteLn ('Emaitza = ', Emaitza) ;
END.
```

Aldagai biak aritmetikoki batuz gero 259 aterako litzateke, baina hori ezinezkoa da. Ordenadoreak zbk_1 eta zbk_2 balioak modu bitarrean batuko ditu:

```
                 1  1  1  1
250              1  1  1  1  1  0  1  0      128 + 64 + 32 + 16 + 8 + 0 + 2 + 0
  9              0  0  0  0  1  0  0  1        0 +  0 +  0 +  0 + 8 + 0 + 0 + 1
____             ________________________
  3              0  0  0  0  0  0  1  1        0 +  0 +  0 +  0 + 0 + 0 + 2 + 1
```

Beraz, emaitza lortzean, `(250+9)` batura batugaiak baino txikiagoa ateratzen da, programak inolako errore mezurik erakutsi gabe.

`Byte` datu-motarako esandakoak beste datu-mota osoetarako hedagarria da, adibidez `Integer` datu-motako zenbaki baterako 16 bit erreserbatzen dira, ezkerreko lehenengo bitak zeinua markatzen du eta gainerako 15ak modulua (2^{15}=`32768` bit konbinazio).

Zenbaki negatiboen lehenengo bitak `1` balio du, eta modulua adierazteko bit konbinazioak `32768` direnez, zenbaki negatibo handiena `-32768` izango da (`10000000 00000000` kode bitarrean) eta zenbaki negatibo txikiena `-1` izango da (`11111111 11111111` kode bitarrean).

`0`tik hasten diren zenbaki osoak eta positiboak `32768` izango dira ere, zeroa bera euren artean (`00000000 00000000` kode bitarrean). Horregatik, `Integer` datu-motako zenbaki positibo handiena `32767` izango da (`01111111 11111111` kode bitarrean).

```
PROGRAM IntegerrenGainezkada ;                          { \TP70\04\GAINEZ2.PAS }
VAR
      zbk_1, Emaitza : Integer ;
BEGIN
      zbk_1 := MAXINT ;   { MAXINT = 32767 }
      Emaitza := zbk_1 + 1 ;
      WriteLn (zbk_1, ' inkrementatuz ', Emaitza, ' ateratzen da') ;
END.
```

Gainezkadaren fenomenoa zenbakien adierazpidean inplizitoki erroturik dagoenez ezin da ahaztu, eta gaur egun bizitzako hainbat zeregin programek kontrolatzen dituztenez gainezkadak direla eta, benetako ezbeharrak suertatu izan dira. Esate baterako europearren ARIANE 5 satelite jaurtigailuak 1996ko Ekainaren 4an eztanda egin zuenean, software arazo bategatik izan zen eta zehazkiago aldagai baten balioa (oso maila handia zenez) zenbakien heinetik kanpo geratu zelako, hots, gainezkada gertatu zelako.

Hau guztiagatik, gogora dezagun berriro zenbaki osoentzat Turbo Pascal lengoaiak dituen mugak, hona hemen **4.3.1** puntuan agertu genuen taula:

Datu-mota	Heina	Formatua
`Shortint`	-128 .. 127	8 bit eta zeinuarekin
`Integer`	-32768 .. 32767	16 bit eta zeinuarekin
`Longint`	-2147483648 .. 2147483647	32 bit eta zeinuarekin
`Byte`	0 .. 255	8 bit eta zeinurik gabe
`Word`	0 .. 65535	16 bit eta zeinurik gabe

Non, datu-mota bakoitzari dagokion heinaren mugak nola lortzen direla testuaren bigarren kapitulua justifikatu zen teorikoki.

Baina, Turbo Pascal lengoaiak muga horien balioak lortzeko baliabideak eskaintzen ditu, `High()` eta `Low()` funtzioen bitartez lor daitezke osoa den zenbaki bati dagozkion goimuga eta behemuga.

Ikus `High_eta_Low_funtzioak` izeneko programa hau:


```
PROGRAM High_eta_Low_funtzioak;           { \TP70\04\HIGH_LOW.PAS }

VAR
  s : ShortInt;
  i : Integer;
  l : LongInt;
  b : Byte;
  w : Word;

BEGIN
   WriteLn ('    BEHEMUHA    ALDAGAIA    GOIMUGA');
   WriteLn ('------------    --------    ------------');
   WriteLn (Low(s):12, ' <= ShortInt <= ', High(s));
   WriteLn (Low(i):12, ' <= Integer  <= ', High(i));
   WriteLn (Low(l):12, ' <= LongInt  <= ', High(l));
   WriteLn (Low(b):12, ' <=   Byte   <= ', High(b));
   WriteLn (Low(w):12, ' <=   Word   <= ', High(w));
END.
```

`Low()` funtzioa edozein datu-motatako aldagai bati aplikatzen bazaio, aldagai horrek har dezakeen baliorik txikiena itzultzen du. Gauza bera egiten du `High()` funtzioak baina baliorik handiena itzuliz.

Honezkero, hona hemen `High_eta_Low_funtzioak` izeneko programa exekutatzean lortzen den irteera:

```
    BEHEMUGA    ALDAGAIA    GOIMUGA
------------    --------    ------------
        -128 <= ShortInt <= 127
      -32768 <= Integer  <= 32767
 -2147483648 <= LongInt  <= 2147483647
           0 <=   Byte   <= 255
           0 <=   Word   <= 65535
_
```

`Low()` eta `High()` funtzioak ezin dira zenbaki errealekin erabili. Dakigunez, zenbaki errealak dezimalak dituztenak dira eta hurrengo puntuan ikasten dira.

4.3.2 Datu-mota errealak

80x86 familiako prozesadoreak zenbaki osoak tratatzeko diseinaturik daude, baina zenbaki frakzionatuak[3] lantzeko erraztasunik ez dute. Hori dela eta, 80x86 familiako mikroprozesadoreek koprozesadore matematiko laguntzaileak izan ditzakete, 80x87 izenez ezagutzen diren koprozesadoreak alegia.

Edozein ordenadoretan (koprozesadorerik duen ala ez duen kontutan izan gabe), zenbaki frakzionatuak lantzeko Turbo Pascal lengoaiak `Real` izeneko datu-mota eskaintzen du. Datu-mota honetako aldagaiek 6 byte hartzen dituzte memorian, eta horietan berretzailea eta mantisa (zeinua eta modulua) gordeko dira. `Real` batek duen heina $2.9x10^{-39}$-tik $1.7x10^{38}$-ra doana da, eta dituen digitu esaguratsuak 11 edo 12 dira.

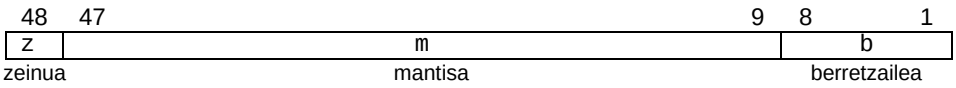
[3] Kopuru frakzionatuak adieraztean balioa eta komaren posizioa biltegitu behar dira memorian, horretarako (eta berriro ere bigarren kapituluan esandakoak gogoratuz) *koma higikorrezko formato bitarra* erabiltzen da, zeinean mantisa eta berretzailea bereizten diren.

Zenbait programetan, dointasuna edo/eta moduluaren mugak `Real` izeneko datu-motarenak baino handiagoak behar izanez gero Turbo Pascal lengoaiak datu-mota erreal hedatuak ditu. Horiek erabiltzeko ordenadoreak koprozesadorea edukiko du, edo duela simulatu beharko da. Hona hemen Turbo Pascal 7.0 datu-mota errealen taula:

Datu-mota	Heina		Tamaina (byte)	Digitu esanguratsuak
	behemuga	goimuga		
`Real`	$2.9x10^{-39}$	$1.7x10^{38}$	6	11 edo 12
`Single`	$1.5x10^{-45}$	$3.4x10^{38}$	4	7 edo 8
`Double`	$5.0x10^{-324}$	$1.7x10^{308}$	8	15 edo 16
`Extended`	$3.4x10^{-4932}$	$1.1x10^{4932}$	10	19 edo 20

Zenbaki errealak idazteko Turbo Pascal lengoaiak modu bi onartzen ditu. Lehenengoan alde osoa puntua eta alde zatigarria idazten da, adibidez hamahiru eta erdi jartzeko `13.5` jarriko litzateke. Bigarren idazkeran (modu zientifikoa delakoan) hamahiru eta erdi jartzeko `1.35E+01` idatziko litzateke, non `E` ostean dagoena hamarreko berretzailea den eta aurrean dagoena mantisa den.

Ordenadorearen memorian ***K*** kopuru frakzionatu bat adierazteko kode bitarrean egingo da. Horretarako eta `Real` datu-mota adibidetzat harturik, `Real` bati dagozkion 6 bytak (48 bit) eremuka banatzen dira:



K kopuruaren balioa ondoko formulaz kalkula daiteke:

$$K = (-1)^{z} \times 1.m \times 2^{(b-129)}$$

Gure programa guztietarako `Real` datu-mota nahikoa izango da, baina inoiz `Extended` izeneko datu-mota erabili behar bada ondoko programa biak eta dagozkien irteerak aztertu:

```
PROGRAM KoprorikEz ;

CONST
   ZENBAKIA = 0.1234567890123456789 ;
BEGIN
    WriteLn ('ZENBAKIA=', ZENBAKIA) ;
END.
```

```
PROGRAM Koprorekin ;
{$N+}
CONST
   ZENBAKIA = 0.1234567890123456789 ;
BEGIN
    WriteLn ('ZENBAKIA=', ZENBAKIA) ;
END.
```

Programak izendatzeko aukeratu diren izenak alde batera utzirik, programa biren artean dagoen desberdintasuna `{$N+}` iruzkina litzateke, eta iruzkinak ez direnez konpilatzen irteera berdina izan beharko litzateke batean zein bestean. Baina ez, irteerei begirada bat emanez `KoprorikEz` izeneko programan `ZENBAKIA` konstantearen adierazpiderako `Real` datu-mota erabili da eta beste programan aldiz `Extended` datu-motaz baliatu da ordenadorea. Eta horra hor `{$N+}` iruzkinaren zergatia, izan ere iruzkin itxura izan arren konpiladoreari ematen zaion agindua[4] da `{$N+}`, horren bitartez koprozesadorea duen sistema batetarako kodea sortzea eskatzen zaio (ahalmen hau desaktibaturik dago eta 6 byteko `Real` datu-mota zenbaki errealak adierazteko modu lehenetsia da).

```
ZENBAKIA= 1.2345678901E-01
_
```

```
ZENBAKIA= 1.23456789012346E-0001
_
```

[4] Horrelako aginduei direktiba esaten zaie, *konpiladorearen direktibak* aurrerago ikusiko ditugu.

4.3.1.1 Zenbaki errealen eragileak

Datu-mota errealek ezagutzen dituzten operazioak aritmetikoak eta erlaziozkoak dira. Lehenengoek, adierazpen aritmetikoak, formula matematikoak idazteko balio dute, eta erlaziozko eragileak berriz, adierazpen logikoak sortzeko erabiltzen dira. Ondoko tauletan biltzen dira operadore horiek:

Eragilea	Eragiketa	Adibidea
+	Batuketa	`zbk + 1.2`
-	Kenketa	`zbk - 0.42E2`
-	Ukapena	`- zbk`
*	Biderkaketa	`zbk * 31.6E-1`
/	Zatiketa erreala	`zbk / 0.04`

Eragilea	Eragiketa	Adibidea
>	Handiago baino	`zbk > 1.07`
<	Txikiago baino	`zbk < 2.9`
>=	Handiago edo berdin	`zbk >= 3.45E-1`
<=	Txikiago edo berdin	`zbk <= 4.88E4`
=	Berdin	`zbk = 5.6723E+3`
<>	Desberdin	`zbk <> 6.7`

Eragile horiek dituzten lehentasun arauak zenbaki osoek dituztenak berberak dira.

Operadore aritmetikoekin, konstanteekin, aldagaiekin eta parentesiekin adierazpen aritmetikoak sortuko dira, eta Turbo Pascal lengoaiak sintaxi zehatz bat exigitzen du. Ondoko zerrendan formula matematikoak eta dagozkien Turbo Pascal token egokiaz osaturiko adierazpenak agertzen dira (lehentasun arauek eskatu ez arren, zenbaitetan eta irakurgarritasuna errazteko asmoz, soberan egon daitezkeen parentesiak jartzen dira):

$\frac{y^2 - 42}{x + 3,1}$ `(y*y - 42) / (x + 3.1)`

$(y^2 + x)\, z$ `(y*y + x) * z`

$\frac{-1}{x + 1}$ `( - 1) / (x + 1)`

4.3.1.2 Eragile aritmetiko eta eragigaien arteko bateragarritasuna

Dagoenekoz zenbaki osoak eta zenbaki frakzionatuak adierazteko datu-motak ikasi ditugu, eta datu-motarekin batera memoriko biltegitze sistema bat eta onargarriak diren eragiketak definiturik daude. Orain arteko espresio aritmetikoetan zenbaki osoak eta zenbaki errealak erabili ditugu baina talde bakoitza bere aldetik eta nahastu gabe. Adierazpen aritmetiko batek, ikusi dugunez, balio bakar batera ebaluatzen da, adierazpen aritmetikoaren eragigaietan datu-mota ezberdinak agertzen direnean espresioak hartzen duen balio bakar hori adierazpen aritmetikoaren emaitza izango da eta honen datu-mota ere bakarra izango da.

Eragigaien konpatibilitatea edo bateragarritasuna lege batez gidatzen da: bi eragigai arteko operazio batean eragigai bat datu-mota batekoa izanik eta beste eragigaia beste datu-

mota batekoa izanik, emaitzari dagokion datu-mota eragigaien bi datu-motetatik garaiena[5] izango da.

Lege horren ondorio bezala, ondokoa baiezta daiteke: bi eragigai arteko operazio bat egitean eragigai biak datu-mota berekoak direnean, emaitzaren datu-mota eragigaiena izango da; baldin eta operazioak berak kontrakorik behartzen ez badu[6].

Eragile aritmetikoen eta euren eragigaien konpatibilitatea taula bezala jarririk hau litzateke:

Eragilea	Eragiketa	Eragigaien datu-mota	Emaitzaren datu-mota
-	Ukapena	Osoa	Osoa
-	Ukapena	Erreala	Erreala
+	Batuketa	Osoa, Osoa	Osoa
+	Batuketa	Osoa, Erreala	Erreala
+	Batuketa	Erreala, Erreala	Erreala
-	Kenketa	Osoa, Osoa	Osoa
-	Kenketa	Osoa, Erreala	Erreala
-	Kenketa	Erreala, Erreala	Erreala
*	Biderkaketa	Osoa, Osoa	Osoa
*	Biderkaketa	Osoa, Erreala	Erreala
*	Biderkaketa	Erreala, Erreala	Erreala
/	Zatiketa erre.	Osoa, Osoa	**Erreala**
/	Zatiketa erre.	Osoa, Erreala	Erreala
/	Zatiketa erre.	Erreala, Erreala	Erreala
DIV	Zatiketa osoa	Osoa, Osoa	Osoa
MOD	Zatiketaren h.	Osoa, Osoa	Osoa

Eragile eta eragigaien bateragarritasuna lantzeko asmoz hona hemen ariketa bat:

```
PROGRAM Datu_motenBateragarritasuna ;                { \TP70\04\ESPRE2.PAS }
VAR
      Eragigai1, Eragigai2 : Integer ;
BEGIN
      Eragigai1 := 32 ;
      Eragigai2 := 17 ;
      WriteLn ('Batuketaren emaitza = ', Eragigai1 + Eragigai2) ;
END.
```

`Datu_motenBateragarritasuna` programa honetan zenbaki osoak diren aldagai bi definitu dira, `WriteLn` egitean adierazpen aritmetiko bat (batuketa bat) burutu eta dagokion emaitza pantailaratuko da, eragigaiek dituzten balioetarako pantailaraketa `49` izango da (zein eragigaiak bezala `Integer` datu-mota den). Programa hori abiapuntuz harturik ondoko aldaketak egin eta emaitzak (edo errore mezuak) interpretatu:

1. Eragiketa aldatu. Batuketa kalkulatu beharrean gainerako beste eragileak aplikatu (`-`, `*`, `/`, `DIV` eta `MOD`).
2. Eragigai biren datu-mota `Real` izan dadila. Eragigai biak errealak izanik, zenbaki errealen eragile guztiak aplikatu.
3. Eragigai bat `Real` bezala mantendu bestea `Integer` izan dadila. Emaitzak aztertzean bigarren puntukoak berberak izango dira.

[5] Adibidez, `Integer` eta `Real` datu-mota artean garaiena `Real` da.
[6] Adibidez, edozein eragigai bikoterako / zatiketaren emaitzari dagokion datu-mota beti erreala izango da.

4.3.3 Datu-mota boolearrak

Aintzinako Grezian arrazonamenduaren sistematizazioa lortu zen. Grezia klasikoan geometria aztertzeko axiomak erabiliz egiten zen. Beraz, esate baterako zirkuluaren propietateak ikasteko esperimentuak eta neurketak egin beharrean, grekoek zirkuluaren definizio formala eginez dagokion geometriaren dedukzio-sistema bat eraiki zuten.

Era berean, garai hartako pentsatzaileek baliozko arrazonamendua nola egin zitekeen sakonki ikertu zuten. Aristotele filosofoak logika formala sortu zuen eta bere lanek XIX. mendean jarraipena izan zuten. Hona hemen Aristotelen tratatuetan irakur daitekeen adibidea:

> Baldin eta zerbait pertsona bat bada, orduan hilkorra da.
> Sokrates pertsona bat da.
> Ondorioz, Sokrates hilkorra da.

Arrazonamendu mota honi *modus ponens* esaten zaio eta ikus daitekeenez Baldin eta - orduan gakoak erabiltzen ditu. Matematika eta gainerako zientzietan agertzen da arrazonamendu mota hau eta oso emankorra da, baina bada beste arrazonamendu era bat asko erabiltzen dena eta *modus tollens* bezala ezagutzen dena. Aristotelerekin jarraituz:

> Baldin eta zerbait pertsona bat bada, orduan hilkorra da.
> Zeus ez da hilkorra.
> Ondorioz, Zeus ez da pertsona.

Aristotelen lanei jarraipena emanez, George Boole matematikaraiak, 1847 urtean, proposizio-logikaren hedapena lortu zuen. Boolek adierazpen aritmetikoen eta adierazpen logikoen arteko paralelismoa aurkitu zuen, hori dela eta AND eta OR eragile logikoek x eta + eragile aritmetikoekin duten parekotasuna handia dela frogatu zuen. Adierazpen aritmetikoek zenbakiak eta kopuruak maneiatzeko ahalmena dute, modu beretsuan Boole-ren Algebrak proposizioak (esaldiak) zuzentasunez eta doitasunez maneiatzeko ahalmena du.

Booleren funtsezko ideia argumentazio logikoak sinboloen bitartez ordezkatzea izan zen. Jarraian ematen diren esaldiak balio bi onartzen dituzte **gezurra** ala **egia**, horregatik proposizioak izango dira:

- Zenbaki naturalen artean negatiborik ez dago
- Auto gehienen erregaia alkohola da
- 4 + 6 = 15
- Hiru bider lau hamabi dira
- Esaldi hau proposizio bat da

Goian aipaturiko esaldiak proposizioak dira, baina horien maneiaketa errazteko proposizioak sintaxi jakin bat beteko duten aldagai sinbolikoen bitartez ordezkatzen dira, aldagai horien datu-mota Turbo Pascal goimailako lengoaian Boolean deitzen da.

Datu-mota boolearren bitartez definituriko konstanteek edo/eta aldagaiek bi balioetatik bat har dezakete. Aldagai edo konstante boolear batek FALSE ala TRUE balio dezake, baina ez besterik. Beraz datu-mota boolearrek duten heina 2 da, eta zentzu fisikorik izan ez arren FALSE balioa TRUE balioa baino txikiagoa da.

Nahiz eta bit batekin aski izan, Turbo Pascalek byte bat erreserbatzen du memorian aldagai edo konstante boolear baterako. Horra hor konstante eta aldagai boolearrak definitzeko lerroak:

```
CONST
       OK = TRUE ;
       GEZURRA = FALSE ;
VAR
       Aurkitua, Amaiturik : Boolean ;
```

Boolearrak diren datu-motek onartzen dituzten eragiketak bi taldetan sailkatzen dira, batetik eragiketa logikoak deitzen direnak, eta bestetik erlaziozko eragiketa ezagunak. Berriak diren eragile logikoei begirada bat eman diezaiegun.

Eragilea		Eragiketa	Adibidea
NOT	ez	Ukapen logikoa	NOT Aurkitua
AND	eta	Biderkaketa logikoa. Konjuntzioa	Aurkitua AND OK
OR	edo	Batuketa logikoa. Disjuntzioa	Aurkitua OR OK
XOR	ala	OR elkarbaztertzailea	Aurkitua XOR OK

Besteek, erlaziozko eragileek, esan dugunez zentzu fisikorik ez dute eragigai boolearrekin. Edozein kasutan ere, erlaziozko eragileen bitartez sortzen diren adierazpenak ebaluatzeko FALSE balioa TRUE balioa baino txikiagoa dela gogoratu.

Eragilea	Eragiketa	Adibidea
>	Handiago baino	Aurkitua > OK
<	Txikiago baino	OK < TRUE
>=	Handiago edo berdin	OK >= Aurkitua
<=	Txikiago edo berdin	Aurkitua <= OK
=	Berdin	OK = OK
<>	Desberdin	OK <> OK

Datu-mota boolearrekin erabiliko ditugun erlaziozko eragileak azken biak izango dira, = eta <> alegia.

4.3.3.1 Adierazpen boolearrak

Lau dira Turbo Pascal lengoaiaren eragile logikoak, eta aritmetikan eragile aritmetikoak bezala, eragigaiak elkar lotuz adierazpenak eraikitzeko balio dute (adierazpen logikoak[7] eragile logikoekin, adierazpen aritmetikoak eragile aritmetikoekin).

A eta B eragigai boolearrak izanik, eragile logikoak aplikatu eta ondorioztatzen den adierazpen bakoitzari dagokion emaitza taula batean antolantzen da, horri egi taula esaten zaio. Egi tauletan agertuko diren emaitzak egiazkoak (e) ala gezurrezkoak (g) izan daitezke.

A	B	A OR B
e	e	e
e	g	e
g	e	e
g	g	g

A	B	A AND B
e	e	e
e	g	g
g	e	g
g	g	g

A	B	A XOR B
e	e	g
e	g	e
g	e	e
g	g	g

A	NOT A
e	g
g	e

Ikusten denez, NOT eragileak A-ren balioari dagokion aurkakoa lortzen du. Beste eragileek eragigai bi darabiltzate, A AND B adierazpena egia izan dadin eragigai biek TRUE balio behar dute. Bestalde, A OR B disjuntzioa egia izan dadin nahikoa da eragigai bietarik batek TRUE balio izatea.

[7] Adierazpen logiko edo boolear batek FALSE ala TRUE balioetatik bat hartuko du.

Disjuntzio arruntarekin batera bada eguneroko hizkuntzan erabiltzen den beste disjuntzio mota bat, *disjuntzio elkarbaztertzailea* alegia. Amak umeari honela esaten dionean "piszina edo zinema", bien artean bat aukeratu behar duela ulertzen da, hots, piszinara joan ala bestela zinemara. Argi dagoenez disjuntzio logiko arruntaren egi taulak ez dio disjuntzio elkarbaztertzailearen egoera horri egoki erantzuten, horretarako `A XOR B` disjuntzioa dago eta duen emaitza egia izango da sarrera biak desberdinak direnean.

Lau eragile logiko horien artean hierarkia bat dago, eta operadore aritmetikoen antzera familietan sailka daitezke: ukatzailea logikoa alde batetik, disjuntzio biek batuketaren familia osatzen dute eta konjuntzioa biderkaketa familia litzateke. Adierazpen logikoetan eragileak konbinatzen direnean, ukatzailea ebaluatzen da lehen, ondoren, eta ordena zainduz, konjuntzio eragilea eta batuketa logiko familiako disjuntzioekin amaitzeko. Horregatik ondoko programan ez da parentesirik behar:

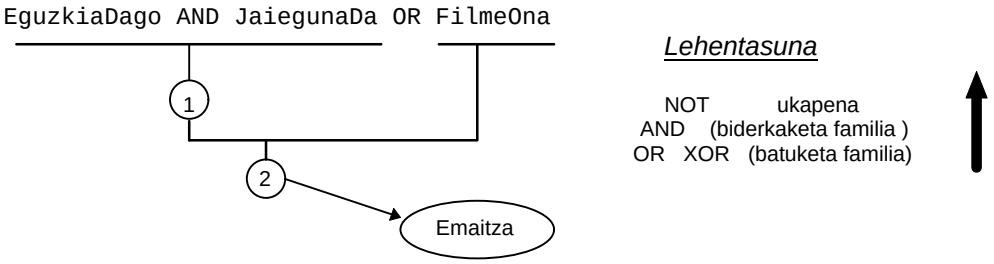
```
PROGRAM BolearrenLehentasuna ;                          { \TP70\04\BOOLEHE.PAS }
VAR
      Eragigai1, Eragigai2 : Boolean ;
BEGIN
      Eragigai1 := FALSE ;
      Eragigai2 := TRUE ;
      WriteLn ('NOT Eragigai1 OR Eragigai2 = ', NOT Eragigai1 OR Eragigai2) ;
      WriteLn ('Eragigai1 OR NOT Eragigai2 = ', Eragigai1 OR NOT Eragigai2) ;
END.
```

Parentesiak soberan egon arren, argitasunagatik ipintzea komeniko litzateke programa errazago irakurri ahal izateko horrela idatziz: `Eragigai1 OR (NOT Eragigai2)`. `BolearrenLehentasuna` izeneko programa baliatuz aldaketak egin eta emaitzak interpretatu.

Zenbaitetan kalkulu logikoak zuzentasunez burutzeko parentesiak derrigorrezkoak dira. Adibidez, ondoko proposizioekin kalkulu logiko bat egin nahi da:

Proposizioa	Aldagaia
Filme ona ematen dute	`FilmeOna`
Eguraldi eguzkitsua	`EguzkiaDago`
Jaieguna denez lanik ez	`JaiegunaDa`

Proposizio horiek egia direnean, dagozkien aldagaietan `TRUE` gordeko dugu, eta gezurra direnean berriz `FALSE`. Egun jakin batean, hiru aldagaien balioak ezagunak direlarik zer nolako plana izango dugun ondoko adierazpen logikoaren bitartez kalkula daiteke (plan ona denean adierazpenak `TRUE` balio du, eta kontrakoan espresioaeren balioa `FALSE` izango da).



Egun eguzkitsua bada kanpora irten gaitezke (eginkizun hori plan on bat dela kontsidera dezakegu) baina horretarako lanera ez joatea ezinbestekoa denez, proposizio biak konjuntzioz lotuko ditugu. Baina bada beste posibilitate bat eguna galdutzat ez jotzeko, pelikula on bat ikustea alegia; eta hori egiteko gaba edo arratsaldea aprobetxa dezakegunez ez dago jaia izatearen baldintzarik.

Baina parentesiak jarriz, interpretazioa erabat aldatzen da:

```
EguzkiaDago AND (JaiegunaDa OR FilmeOna)
```

Plan ona izateko derrigorrez egun eguzkitsua behar da, eta gainera bi egoera hauetatik bat egia izango da (edo biak): jaieguna izatea, filme ona programaturik egotea.

Egoerak subjetiboak direnez eta norberaren gogoarekin zerikusia dutenez, espresio logiko biak egokiak izan daitezke baina bakoitzak pentsakera desberdina isladatzen du. Arau orokor bezala, eragileen lehentasun arauen arabera soberan egon arren, irakurketa erraztuko luketen parentesiak erabiltzea gomendatzen da. Jarraian ematen den programa azter ezazu:

```
PROGRAM UrtebetetzeaAsmatu ;                          { \TP70\04\URTEBETE.PAS }
CONST
      NIRE_HILA = 3 ;
      NIRE_EGUNA = 26 ;
VAR
      Eguna, Hila : Byte ;
BEGIN
      Write ('Egun bat sartu: ') ;
      ReadLn (Eguna) ;
      Write ('Hilabete bat sartu: ') ;
      ReadLn (Hila) ;
      Write (Hila, '/' ,Eguna) ;
      WriteLn (' nire urtebetetzea: ', (Eguna=NIRE_EGUNA) AND (Hila=NIRE_HILA)) ;
END.
```

`UrtebetetzeaAsmatu` programa honen ezaugarria erabiltzareari datuak eskatzen dizkiola eta horiek landu ondoren informazio bat erantzuten duela, orain artekoak ez bezala elkarreragilea da. Programa honetan lehenengo aldiz `ReadLn` agindua agertzen zaigu, aurrerago (**4.5.1** puntuan) ikasiko dugu baina orain aldagai bati teklatuaren bitartez balioren bat emateko dela jakin dezagun; egun eta hilabete bat zehaztu ondoren emaitza lortzeko adierazpen boolear bat ebaluatu behar da:

```
Egun bat sartu: 26
Hilabete bat sartu: 12
12/26 nire urtebetetzea: FALSE
_
```

Orain arte aipatu ditugun eragileen lehentasuna jarraian laburbiltzen da:

Eragilea	Lehentasuna
- NOT	1.
* / DIV MOD AND	2.
+ - OR XOR	3.
< <= = <> >= >	4.

Ez da ahaztu behar familien arteko lehentasuna aldatzeko parentesien bitartez egiten dela. Eta lehentasun bereko eragileak adierazpen batean aurkitzean, adierazpenean ezkerragora agertzen den eragiletik hasten da ebaluazioak egiten Turbo Pascal lengoaia.

Jarraian agertzen diren adierazpen logikoen balioa iruzkin bezala jarri da. Baldin eta `x=12.4`, `y=37.0`, `z=0.1` eta `ordenaturik=FALSE` orduan:

```
(z < x) AND (y >= 21)          { Egia }
(z < x) AND ordenaturik        { Gezurra }
z < (x AND y) >= 21            { Errorea }
z < x AND y >= 21              { Aurreko errore berdina }
NOT ordenaturik                { Egia }
(x >= y) OR ((y-4) >= z)       { Egia }
x > 12.4                       { Gezurra }
ordenaturik AND TRUE           { Gezurra }
```


4.3.4 Karaktere datu-mota

Kopuruaz gain, programetan izenak erabiltzen direlako ordenadorearen memorian letrak ere gorde behar izaten dira. Letrak edo karaktereak era bitarrean idazterakoan kodetze sistema ezberdinak daude, Turbo Pascal lengoaiak ASCII kode hedatua erabiltzen du, eta karaktere datu-motaren marka `Char` hitz erreserbatua da.

Karaktere bat memorian zeinu gabeko 8 biteko formatuan biltegitzen denez, `Char` datu-motak `Byte` datu-motarekin lotura eta antza du. Esan denez, `Char` aldagai batean gorde daitekeen balioa ASCII taulako sinboloetatik bat izango da, ondoko lerroetan karaktere konstante bat eta bi aldagai definitzen dira:

```
CONST
       GORA = ‘G‘ ;
VAR
       Aukera, Erantzuna : Char ;
```

Turbo Pascaleko identifikadoreak letrak direnez, hauek eta `Char` datu-motaren balioak desberdintzeko komatxoak erabiltzen dira. Bestalde, programa batean adibidez `A` sinboloa agertzean paper bi joka ditzake, batetik aldagai bat izan daiteke eta bestetik alfabeto majuskularen lehenengo letra. Egoera hori ekidin nahirik karaktereen balioak komatxoen artean mugatzen dira, beraz programa batean `A` sinboloak aldagai bat adierazten du (zehatzago identifikadore bat) eta ‘`A`‘ moldeak berriz karaktere baten balioa litzateke.

Aurrekoa kontutan izanik, ondoko programan agertzen diren konstante eta aldagaiak identifikatu ondoren euren balioak zeintzu diren asmatu:

```
PROGRAM Karaktere_KonstanteEtaAldagaiak ;           { \TP70\04\KARAK1.PAS }
CONST
       IRTEN = ‘I‘ ;
       MIN = 3 ;
VAR
       Aukera1, Aukera2, B : Char ;
       Zenbat, K, X, Y : Integer ;
BEGIN
       Write (‘Zein da zure sarrera: ‘) ;
       ReadLn (B) ;
       K := 7 ;
       Zenbat := 2;
       X := MIN * Zenbat - K ;
       Y := Y + X ;                      { Hau ondo dago? }
       Aukera1 := B ;
       Aukera2 := ‘B‘ ;
END.
```

Datu-mota bat definitzeak, biltegitze sistema batekin batera, onargarriak diren eragiketak dakartza. Karaktere datu-motak onar ditzakeen eragileak erlaziozkoak dira, erlaziozko eragileak aplikatzean `Char` delakoa datu-mota ordinala dela kontutan izango da.

`Char` datu-motaren balioak mugatuak dira kopuruan (256 sinbolo dira) eta gainera balioen arteko orden jakin eta aldaezina dago (ASCII taularen arabera ordenaturik daude karaktereak). Orden horri zor zaio ondoko konparaketak egin ahal izatea:

```
PROGRAM KaraktereenEragiketak ;                     { \TP70\04\KARAK2.PAS }
CONST
       IRTEN = ‘I‘ ;
       MIN = ‘7‘ ;
VAR
       Karak1, A, B : Char ;
```

```
BEGIN
      A := 'I' ;
      B := 'b' ;
      Karak1 := 'Z' ;
      WriteLn ('IRTEN = A       ', IRTEN = A) ;
      WriteLn ('IRTEN <> ''Q''     ', IRTEN <> 'Q') ;
      WriteLn ('MIN < A         ', MIN < A) ;
      WriteLn ('B > Z           ', B > Z) ;
      WriteLn ('Karak1 >= ''Z''    ', Karak1 >= 'Z') ;
      WriteLn ('Karak1 <= B     ', Karak1 <= B) ;
END.
```

Karaktereak konparatzerakoan ASCII taularen arabera egiten da, eta taula horren antolaketa nolakoa den ezagutzea komeni da. Deskriba dezagun ASCII taula: Lehenengo 32 karaktereak (0-tik 31 bitartera) bereziak dira, 33. karakteretik 47. bitartera eragiketak adierazteko sinboloak daude (zuriunea, parentesiak, batuketa eragilea, kenketa eragilea e.a.), ondoren digituak datoz 0-tik 9-ra, gero letra majuskulak (A-tik Z-ra ordenaturik), majuskulen ostean minuskulak (hauek ere alfabetikoki ordenaturik) eta jarraian zenbait sinbolo guztira 256 karaktere multzoa osatu arte.

Ondorioz, edozein karakterek bere posizioa du ASCII taulan eta dagokion ordena berezkoa izanik aldaezina da; adibidez alfabeto majuskularen lehenengo letrari dagokion posizio ordinala 65 da, 'A' karakterea memorian kode bitarrean gordetzen da eta dagokion bit segida 65 zenbaki osoaren berbera da. `Char` eta `Byte` datu-motak arteko aldaketak burutzeko `Ord()` eta `Chr()` funtzio estandarrak daude.

Funtzioak aurrerago ikasiko ditugu, dagoenekoz `sin()` eta `abs()` funtzioak aipatu ditugu, lehenengoak angelua adierazten duen zenbakia hartzen du eta dagokion sinuaren balioa itzultzen du, bigarrenak ere zenbakizko parametro bat behar du baina bere emaitza zenbakiaren balio absolutua da. `Ord()` funtzioari ematen zaion datua karaktere bat da eta eskaintzen duen irteera parametro horri dagokion ordinala (karaktereak ASCII taulan duen posizioa). `Chr()` funtzioak berriz, 0 eta 255 bitarteko zenbaki bati dagokion ASCII karakterea itzultzen du.

```
PROGRAM Ord_eta_Chr ;                                  { \TP70\04\KARAK3.PAS }
VAR
      Karak : Char ;
      Zbk : Byte ;
BEGIN
      Write ('Karaktere bat eman: ') ;
      ReadLn (Karak) ;
      WriteLn (Karak, ' karaktereari dagokion ordena ' , Ord(Karak), ' da') ;
      WriteLn ;
      Write ('0 eta 255 arteko zenbaki bat eman: ') ;
      ReadLn (Zbk) ;
      WriteLn (Zbk, '. karakterea ' , Chr(Zbk), ' da') ;
END.
```

`Ord_eta_Chr` programa exekutatzen ondokoa atera daiteke:

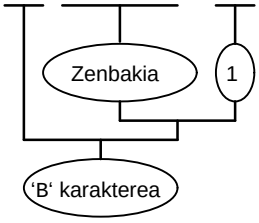
```
Karaktere bat eman: M
M karaktereari dagokion ordena 77 da

0 eta 255 arteko zenbaki bat eman 104
104. karakterea h da
_
```

Interesgarria izan daiteke karaktere aldagai baten edukia inkrementatzea (`Karak` aldagaian ‘A‘ gorderik badago, ‘B‘ gorde izatea), baina karaktereekin ezin direnez batuketak egin ondokoa gaizki egongo litzateke:

```
Karak := ‘A‘ ;
Karak := Karak + 1 ;
```

Horretarako karaktereak zenbaki bihurtu, batu, eta emaitzaren datu-mota berriro aldatu beharko litzateke:

```
Karak := ‘A‘ ;
Karak := Chr(Ord(Karak) + 1) ;
```



Baina badaude datu-mota ordinalekin erabil daitezkeen funtzio estandarrak, antzeko eginkizunak errazten dituztenak: `Pred()` eta `Succ()` funtzio estandarrak. Karaktere datu-motara mugatuz, `Pred()` eta `Succ()` funtzioak honela erabiltzen dira:

`Pred()` sarrera: karaktere jakin bat
irteera: karaktere horren aurrekoa ASCII taulan

`Succ()` sarrera: karaktere jakin bat
irteera: karaktere horren ondokoa ASCII taulan

Ondorioz, agindu hauek baliokideak izango dira:

```
Karak := ‘A‘ ;                          |       Karak := ‘A‘ ;
Karak := Chr(Ord(Karak) + 1) ;          |       Karak := Succ(Karak) ;
```

`ASCII_taula` izendatu dugun ondoko programak karaktereak pantailaratzeko balio du, karaktereak lerroka ateratzen ditu eta bakoitzeko bere ordinala erakusten du. Hasierako 32 karaktereen artean denak inprimagarriak ez direnez, hogeita hamabigarrenetik (zuriunetik) hasiko gara.

```
PROGRAM ASCII_taula ;                                          { \TP70\04\ASCII.PAS }
USES
      Crt ;
VAR
      Itxoin : Char ;
      Zbk : Byte ;
BEGIN
      ClrScr ;
      WriteLn (‘       Hona hemen karaktere inprimagarrien ASCII kodea        ‘) ;
      WriteLn (‘-------------------------------------------------------------‘) ;
      WriteLn ;
      FOR Zbk:=32 TO 255 DO
      BEGIN
          Write (‘   [‘ , Zbk:3, ‘]‘ , Chr(Zbk) ) ;       { pantailaraketa }
          IF (Zbk+1) MOD 8 = 0 THEN
                              BEGIN
                                  WriteLn ;
                                  Itxoin := ReadKey ;
                              END ;
      END ;
      WriteLn ;
END.
```

`ASCII_taula` programan gauza berriak agertzen dira, batzuk ikasiko ditugu eta beste batzuk aurrerago landuko ditugu. Programaren helburua ASCII karaktereak erakustea da (ordinala eta karakterea pantailaratzea hain zuzen), eta helburu hori `{ pantailaraketa }` iruzkinez markaturik dagoen aginduak betetzen du, non `Zbk` aldagaiak 32 eta 255 arteko balio bat hartzen duen. Karaktereak 223 direnez, denak pantailaratzeko 223 `Write` beharko genituzke, edo errepikatzen den `Write` bakarra[8]. Pantailaraketak taula itxura izan dezan, zortzikoten lerroak osatzen dira eta zortzikoteak zehazteko `MOD` eragilea erabiltzen da, zortzikotea osatzen denean `WriteLn` bitartez lerroa aldatu eta tekla bat sakatu arte programa geldiarazten[9] da.

Turbo Pascal lengoaiak programadoreak erabil ditzan hainbat funtzio eta prozedura aurredefiniturik du. Adibidez, `WriteLn`, `Write`, eta `ReadLn` prozedura estandarrak dira (**4.5.1** puntuan sakonki ikasiko ditugunak); `Sin`, `Abs`, `Ord`, `Chr`, `Succ` eta `Pred` funtzio ezagunak estandarrak direnez programadoreak arazorik gabe bere kodean tarteka ditzake.

Prozedura eta funtzio estandarrak aurreidatzirik daude eta `UNIT` deituriko software elementu batzuetan bildurik aurkitzen dira. Oraintsu aipatu ditugun prozedura eta funtzioak `System` izeneko oinarrizko unitatean daude eta edozein programetan eskuragarri egongo dira, baina badira beste funtzio eta prozedura asko beste unitate batzuetan aurkitzen direnak (`ClrScr` eta `ReadKey` adibidez, `Crt` unitatean daude) horiek erabili ahal izateko konpiladoreari `Crt` unitatea karga dezala eskatu behar zaio. Horretarako programaren izenburuaren ostean `USES` hitz erreserbatua jarriko da.

```
PROGRAM Crt_unitatea ;                                  { \TP70\04\CRT_UNIT.PAS }
USES
      Crt ;
VAR
      Itxoin : Char ;
      Zbk : Byte ;
      Izena : String ;
BEGIN
      ClrScr ;
      GotoXY (20, 10) ;
      Write ('Zure izena mesedez: ') ;
      GotoXY (40, 10) ;
      ReadLn (Izena) ;
      HighVideo ;
      GotoXY (20, 14) ;
      TextBackGround (5) ;
      WriteLn ('Kaixo ', Izena, '. Zer moduz zaude?') ;
      Itxoin := ReadKey ;
END.
```

Zer gertatzen da `Crt_unitatea` izeko azken programa honetan `USES` agindua kentzen bada?. Eta tokiz aldatzen bada?.

4.4 PROGRAMA BATEN EGITURA

Turbo Pascal lengoaiaz programatzean ebatzi nahi den arazoa modulutan banatzen da, moduluak[10] parametroen bitartez erlazionatu eta komunikatzen dira. Programa bat idaztean, moduluzko programazio teknikak erabiliko direnez programa berak egitura jakin eta finko bat behar du.

[8] Horixe da `FOR` aginduaren zioa, errepikapena burutzea eta `Zbk` aldagaia inkrementatzea.

[9] Geldiketa `ReadKey` funtzio estandarraz lortzen da.

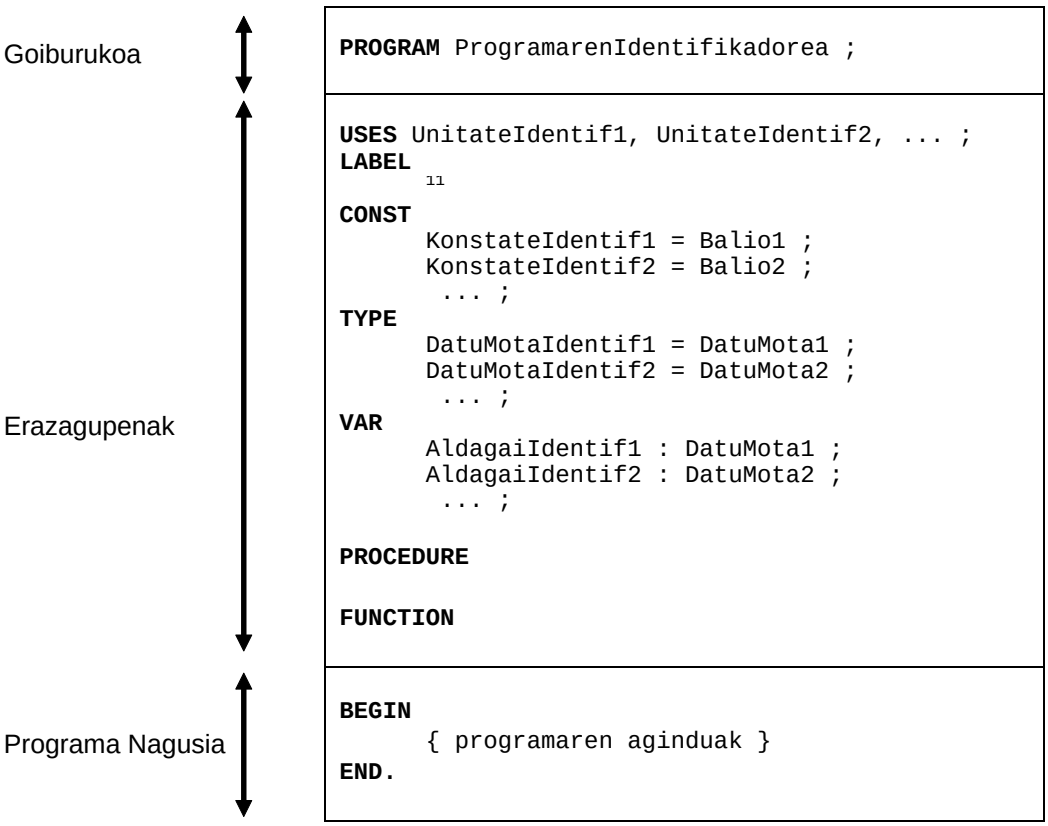
[10] Moduluei azpiprograma esaten zaie, eta Pascal lengoaiak bi motatako azpiprogramak onartzen ditu: prozedurak eta funtzioak.

Programa exekutatzen denean, moduluetatik batek kontrola hartzen du eta hortik abiaturik beste modulu batzuetara alda daiteke, hasieran kontrola hartzen duen moduluari *programa nagusia* esaten zaio. Bestalde, lehenengo agindu bezala programari izen bat jartzen zaio *programa goiburukoa* alegia. Bloke bien artean deklarazioak (edo erazagupenak) idazten dira.

Turbo Pascal Programa batek duen egituraren atal nagusiak hauek lirateke:

- Goiburukoa
- Erazagupen edo deklarazioak
- Programa Nagusia

Ondoko taulan Turbo Pascal programa baten egitura erakusten da:



Hiru ataletatik derrigorrezkoa dena Programa Nagusia da, gogoratu kapitulu honetan idatzi dugun lehenengo programa (`KonstanteaPantailaratzen` deitu izan duguna) non goiburukoa eta programa nagusia besterik ez den agertzen. Baina programa gehienetan deklarazioak behar izaten dira eta praktikan ez da programarik atal hau falta zaionik.

`TYPE` blokeak programadoreak sortuko dituen datu-mota berriekin zerikusia du eta zortzigarren kapituluak ikusiko dugu.

`PROCEDURE` eta `FUNCTION` blokeek programa bat modulutan zatitzeko balio dute eta seigarren kapituluko ikasgaiak izango dira.

[11] `LABEL` bitartez `GOTO` agindurako etiketak definitzen dira, guk ez dugunez `GOTO` agindua erabiliko ez da bloke hori azalduko.

4.4.1 Goiburukoa: PROGRAM hitz erreserbatua

Aukerakoa da, idazten ez bada ere programa konpila daiteke. Edozein kasutan, goiburukoa jartzen denean PROGRAM hitz erreserbatuaz hasiko da, ondoren programadoreak hautatutako etiketa bat jartzen da eta amaitzeko aginduen bukaera markatzen duen ; sinboloa ipintzen da. Orain arte erakutsi diren programa guztien hasierako lerroa goiburukoa izan da.

4.4.2 Erazagupen atala

Programa batean erabiltzen diren elementu guztiak deklaratu egin behar dira, adibidez konstanteak eta aldagaiak. Gauza bera esan daiteke datu mota berriak erabiltzeari buruz, definitu behar direla alegia (ikus **4.4.2.2** puntua). Esan dugunez programa bat idaztean, programadorearen FUNCTION eta PROCEDURE izango diren moduluak egiten dira, horiek ere erazagupen atal honetan garatzen dira.

4.4.2.1 Unitateak

Turbo Pascalek konpilaketa banatua onartzen du, hau da, programa batek beste programadoreren batek idatziriko eta konpilaturiko prozedurak eta funtzioak har ditzake. Horretarako, **4.3.4** puntuko ASCII_taula eta Crt_unitatea izeneko programetan egin den bezala USES sententzia jarriko da.

USES sententzia ez da derrigorrezkoa, baina programa batean agertu behar bada goiburukoaren ostean ipiniko da.

4.4.2.2 Datu-motak

Turbo Pascalaren oinarrizko datu-motak abiapuntu bezala harturik norberaren datu-motak antola daitezke TYPE hitz erreserbatuaren bitartez. Esate baterako, Char datu-mota jatorrizkoa izanik Karaktere datu-mota berria sortuko dugu:

```
PROGRAM DatuMotaBerria ;                              { \TP70\04\DATUMOTA.PAS }
TYPE
      Karaktere = Char ;
VAR
      Letra1 : Char ;
      Letra2 : Karaktere ;
BEGIN
      Write ('Tekla bat sakatu ondoren RETURN eman: ') ;
      ReadLn (Letra1) ;
      Letra2 := Letra1 ;
      WriteLn ('Letra1=' , Letra1, '      Letra2=' , Letra2 ) ;
      WriteLn ;
END.
```

DatuMotaBerria programan aldagai bi agertzen dira, bat Char datu-motakoa eta bestea Karaktere datu-motakoa, ikusten den bezala biak modu berean irakurri, asignatu eta pantailaratu egiten dira. Izan ere, TYPE hitz erreserbatuaren bidez DatuMotaBerria programa zehatz horrek Karaktere datu-mota ezagutuko du.

Programa batean datu-mota espezifikoak erabili nahi badira ondoko sintaxia zainduko dugu: `TYPE` blokearen barruan, = sinboloaren ezkerraldean datu-mota berria identifikatuko duen etiketa jartzen da (**4.2.2.2 Erabiltzailearen identifikadoreak** puntuan esandakoa gogoratu), = sinboloaren eskuinean `DatuMotaIdentif` datu-mota berri hori definitzeko balio duen datu-mota bat (adibidez, `DatuMota`[12]).

```
TYPE
      DatuMotaIdentif = DatuMota ;
```

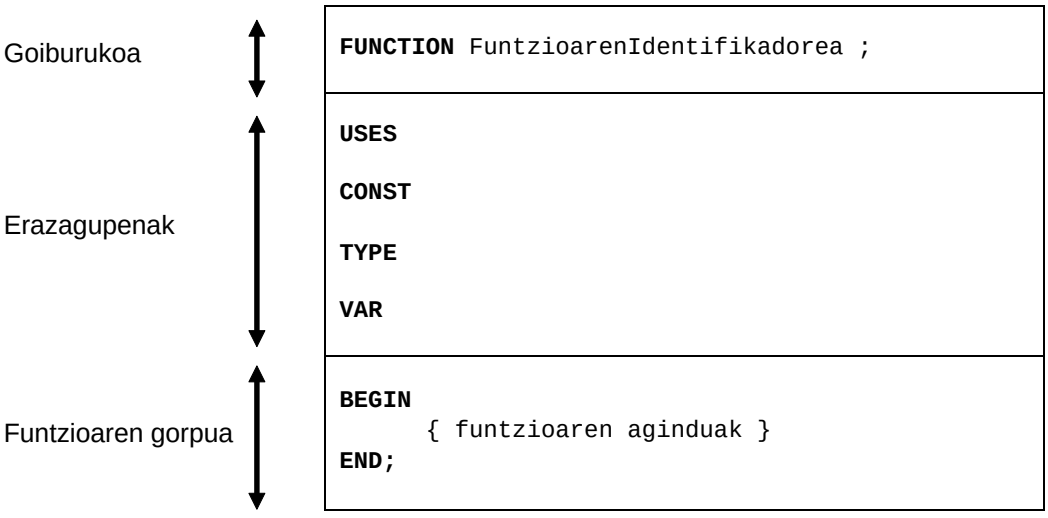
4.4.2.3 Konstante eta aldagaiak

Konstanteak eta aldagaiak **4.2.3** eta **4.2.4** puntuetan aztertu dira.

4.4.2.4 Prozedura eta funtzioak

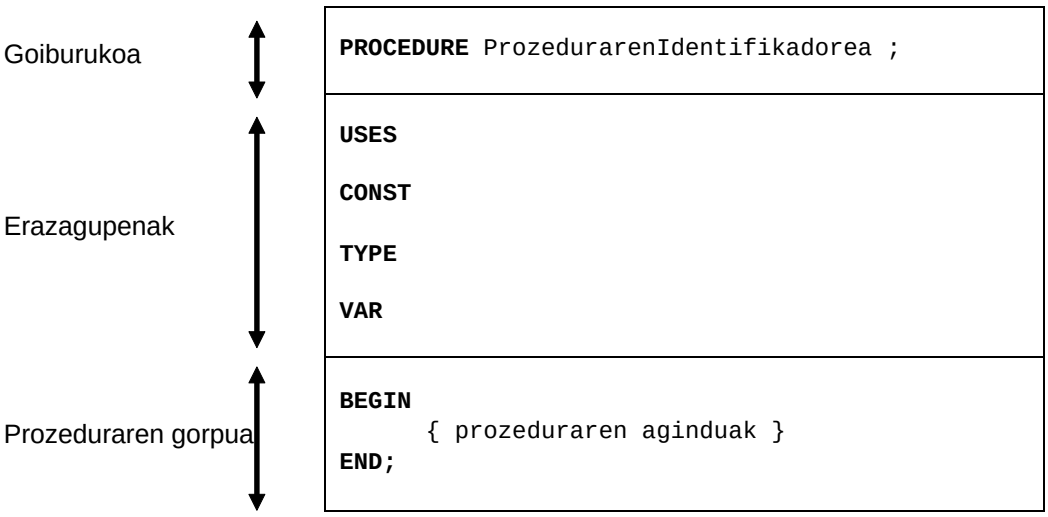
Programa bat modulutan zatitzen denean, modulu horiek Programa Nagusia baino lehen idatzi behar dira. Moduluek programa oso baten egitura dutenez azpiprogramak esaten zaie, bi motatako moduluak daude Pascal lengoaian funtzioak eta prozedurak.

Funtzio batek programa batek duen egitura dauka baina `PROGRAM` hitz erreserbatuaz hasi beharrean `FUNCTION` batez hasten da, eta puntu batez amaitu beharrean puntu eta komaz amaitzen da. Funtzioak, dagozkion aginduaz gain bere datu-mota, konstante, aldagai eta azpiprogramak izan ditzake.



Era berean, prozedura batek ere programaren egitura dauka baina `PROGRAM` hitz erreserbatuaz hasi beharrean `PROCEDURE` batez hasi eta bukatzeko puntu eta koma jartzen da. Honek ere, dagozkion aginduaz gain bere datu-mota, konstante, aldagai eta azpiprogramak izan ditzake.

[12] `DatuMota` Turbo Pascal lengoaiak aurredefiniturik duen datu-mota bat izango da, edo bestela lerro hori baino lehenago programadoreak definitu duen datu-mota espezifikoa izango da.



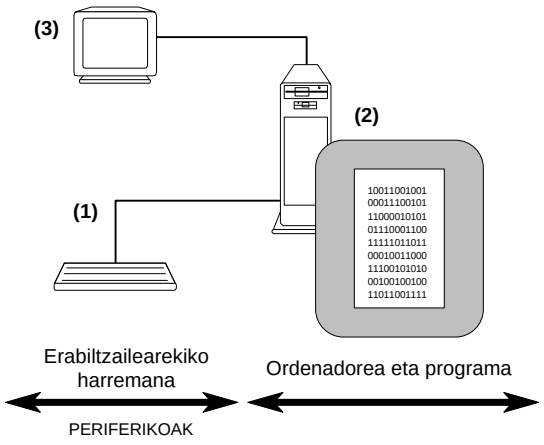
4.4.3 Programa Nagusia

Erazagunenak eta gero Programa Nagusia dator eta bertan sententziak idazten dira, atal hau `BEGIN` eta `END` batez mugatzen da. `END` ostean puntua ipintzen da, bestela konpilaketa prozesuan ondoko errorea ematen du: `Error 10: Unexpected end of file.`

4.5 IRTEERA/SARRERA

Ordenadorea eta erabiltzailearen arteko harremanak periferikoen bitartez gauzatzen dira. Izan ere, ordenadore baten funtzionamendua programa batez gidatzen da, eta programa hori ordenadorearen memorian kokatu behar denez operadore/ordenadore lotura izan behar da. Baina ez bakarrik programa bera, programak landuko dituen datuek eta programak eskainiko dituen emaitzek ere operadore/ordenadore komunikazio horren beharra daukate.

Hurrengo eskeman periferikorik arruntenak agertzen dira (teklatua eta monitorea), Turbo Pascal lengoaiak programa bat teklatutik hartutako datuaz elikatzeko prozedurak ditu, modu berean emaitzak pantailaratzeko ahalmena duten prozedurak ere aurredefiniturik ditu.



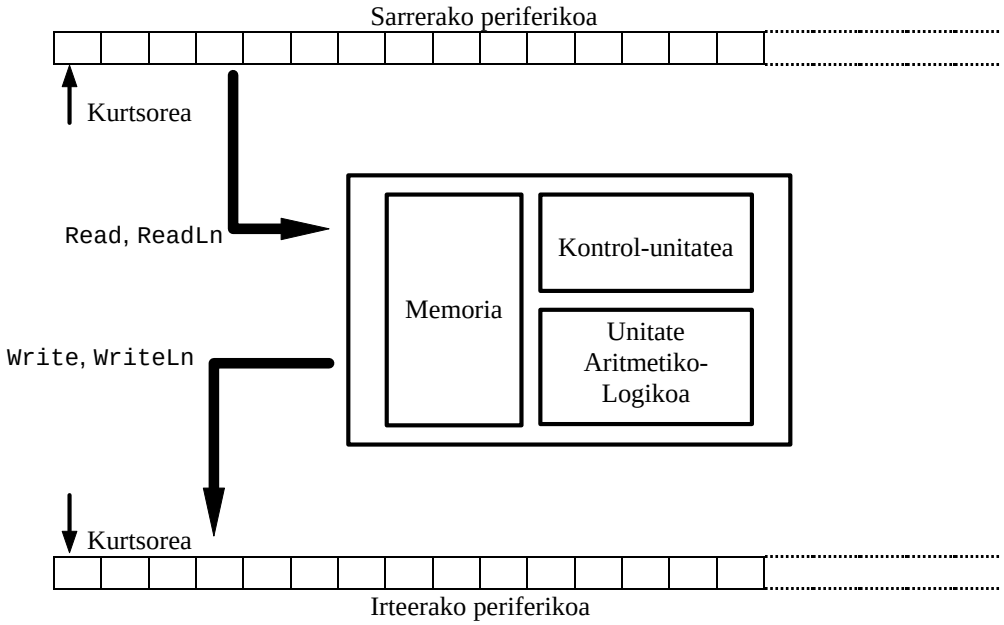
Ordenadore-programa baten bitartez informazio prozesaketa bat egiten denean erabiltzaileari datuak sartzea eskatuko zaio (1), datuak teklatutik zein fitxategitik ordenadorearen memorian barneratzeko `Read` eta `ReadLn` prozeduraz egingo da. Hortik aurrera, ordenadoreak kalkuluak zirkuitu digitalen bitartez burutzen ditu programaren algoritmoa jarraituz (2), eta emaitzak memorian (kode bitarrean, noski) gordez. Emaitzak memoriatik pantailara kanporatzeko kode bitarretik hamartarrera bihurtzen dira (3), hori lortzeko Turbo Pascal lengoaiak `Write` eta `WriteLn` irteerako prozedurak ditu.

Periferikoak sailkatzean sarrerako, irteerako edo sarrera/irteerakoak diren aintzat hartzen da. Adibidez:

	Periferikoa
Sarrera	Teklatu, eskaner, mikrofono, bideo kamera
Irteera	Monitore, inprimagailu, bozgoragailu
Sarrera/Irteera	Disketera, modem, CD-ROM

Nahiz eta periferikoak asko izan programazio lengoaia batek, funtsean, bi operazio onartzen ditu: datuen sarrerak eta emaitzen irteerak. Turbo Pascal lengoaian `Read` eta `ReadLn` prozeduraz egingo dira lehenak, eta `Write` eta `WriteLn` prozeduren bitartez bigarrenak.

Gure programetarako ordenadoreak eskema hau jarraitzen duela onar daiteke:



4.5.1 `Write` eta `WriteLn` prozedurak

`Write` eta `WriteLn` aurredefinituriko prozedurak dira eta programa batetik informazioa operadoreari komunikatzeko balio dute. `Write` eta `WriteLn` berdintsuak dira baina lehenengoak kurtsorea irteeraren hurrengo posizioan uzten du, eta besteak (`WriteLn` delakoak) kurtsorea hurrengo lerroaren hasieran lagatzen du.

`Write` eta `WriteLn` prozedurak diren aldetik pantailaraketak egitea ahalbideratzen dute, baina zer pantailaratu parametro bezala parentesi artean zehazten da. Sarrerako prozedura estandar hauek ez dute parametroen kopurua mugatzen eta `WriteLn` kasurako

parametrorik ez jartzean hurrengo lerroaren hasierara eramaten du kurtsorea ezer pantailaratu gabe. Parametroak adierazpenak dira eta komaz banaturik datoz, adibidez:

```
WriteLn (parametro1, parametro2, parametro3, ...) ;
WriteLn ;
Write (parametro1, parametro2, parametro3, ...) ;
```

Non parametroak, adierazpenak izanik, ondoko itxura izan dezaketen:

- `Adierazpena` zenbakizko edo karakteredun espresioa izango da, zenbakizko espresioa bada osoa edo erreala izan daiteke
- `Adierazpena:n` zenbakizko edo karakteredun espresioa izango da, zenbakizko espresioa bada osoa edo erreala izan daiteke
- `Adierazpena:n:m` zenbaki errealen espresioa izango da

Adierazpenekin batera datozen `n` eta `m` formatua zehazteko balio duten zenbaki osoak dira, ondoko adibideetan formatodun irteerak erakusten dira:

```
PROGRAM IrteerakoFormatua1_n ;                          { \TP70\04\OUT1_N.PAS }
VAR
      ZenbakiOsoa : Integer ;
      ZenbakiErreala : Real ;
BEGIN
      Write (‘Zenbaki oso bat sartu:‘) ;
      ReadLn (ZenbakiOsoa) ;
      Write (‘Zenbaki erreala sartu:‘) ;
      ReadLn (ZenbakiErreala) ;
      WriteLn ;
      WriteLn (‘ n=1 izatean|‘, ZenbakiOsoa:1);
      WriteLn (‘ n=3 izatean|‘, ZenbakiOsoa:3);
      WriteLn (‘ n=0 izatean|‘, ZenbakiErreala:0);
      WriteLn (‘n=14 izatean|‘, ZenbakiErreala:14);
      WriteLn ;
      WriteLn (‘ n=1 izatean|‘, -ZenbakiOsoa:1);
      WriteLn (‘ n=3 izatean|‘, -ZenbakiOsoa:3);
      WriteLn (‘ n=0 izatean|‘, -ZenbakiErreala:0);
      WriteLn (‘n=14 izatean|‘, -ZenbakiErreala:14);
END.
```

`IrteerakoFormatua1_n` izeneko programan zenbakiekin lan egiten da, definituriko bi aldagaiei teklatuz balioak eman ondoren pantailaraketak egiten dira (zenbakien balioak eta dagozkien alderantzizkoak pantailaratzen dira), adibidez `12` eta `12.345` teklatuz emanez gero irteera honakoa litzateke:

```
Zenbaki oso bat sartu:12
Zenbaki erreala sartu:12.345

 n=1 izatean|12
 n=3 izatean| 12
 n=0 izatean| 1.2E+01
n=14 izatean| 1.2345000E+01
n=24 izatean|        1.2345000000E+01

 n=1 izatean|-12
 n=3 izatean|-12
 n=0 izatean|-1.2E+01
n=14 izatean|-1.2345000E+01
n=24 izatean|       -1.2345000000E+01
_
```


`ZenbakiOsoa` osoekin ari garela `n` zenbakiak irteerako erabiltzen den eremuaren zabalera zehazten du, `n`-k 1 balio duenean `12` eta `-12` irteerak ez direnez kabitzen pantailaraketan `n` hori ez da kontutan izango, `n`-k 3 balio duenean `12`-ren aurrean zuriunea tartekatzen da.

`ZenbakiErreala` errealekin ari garela, osoekin bezala, `n` zenbakiak irteerako erabiltzen den eremuaren zabalera zehazten du, kontutan izanik Turbo Pascalak errealentzat idazkera zientikoa darabilela. Hori dela eta, `n`-k 0 balio duenean ez da inolako zenbaki errealik kabituko eta horrelakoetan zenbaki positiboentzat zuriune bat tartekatzen da eta mantisan dezimal bakarra erakusten da, `n`-ren balioa nahiko handia denean dezimal guztiak azalduko dira eta hauen ostean zeroak gehitzen dira `n`-k adierazten duena bete arte, gehienez 10 dezimale erakusten direnez `n`-k oso balio handia duenean aurretik zuriuneak tartekatzen dira.

Zenbakiak izan beharrean karaktereak direnean `n`-k berdin jokatzen du, pantailaraketarako eremu baten luzera definitzea alegia, irteera eremuaren eskubira justifikaturik agertuko delarik:

```
PROGRAM IrteerakoFormatua2_n ;                        { \TP70\04\OUT2_N.PAS }
USES
      Crt ;
VAR
      KaraktereBat : Char ;
      KaraktereKatea : String ;
BEGIN
      ClrScr ;
      Write (‘Karaktere bat sartu:‘) ;
      ReadLn (KaraktereBat) ;
      Write (‘Esaldi bat sartu:‘) ;
      ReadLn (KaraktereKatea) ;
      WriteLn ;
      WriteLn (‘ n=0 izatean|‘, KaraktereBat:0);
      WriteLn (‘ n=3 izatean|‘, KaraktereBat:3);
      WriteLn (‘ n=5 izatean|‘, KaraktereKatea:5);
      WriteLn (‘n=15 izatean|‘, KaraktereKatea:15);
END.
```

Hona hemen programa honi dagokion irteeraren bat:

```
Karaktere oso bat sartu:J
Esaldi bat sartu:Turbo Pascal

 n=0 izatean|J
 n=3 izatean|  J
 n=5 izatean|Turbo Pascal
n=15 izatean|   Turbo Pascal
_
```

Aurreko `IrteerakoFormatua1_n` adibide-programaren irteera aztertuz zenbaki errealak idazkera zientifikoan agertzen dira. Zenbaki errealak idazkera zientifikoan[13] erakustea ekidin nahi badugu, `n` eta `m` formatuaz lor daiteke. Orain arte bezala `n`-k eremu osoaren luzera adieraziko luke, eta `m` dezimalen kopurua litzateke.

Aurredefiniturik dagoen `PI` zenbakia honela ager daiteke `IrteerakoFormatua_nm` izeneko programaren arabera:

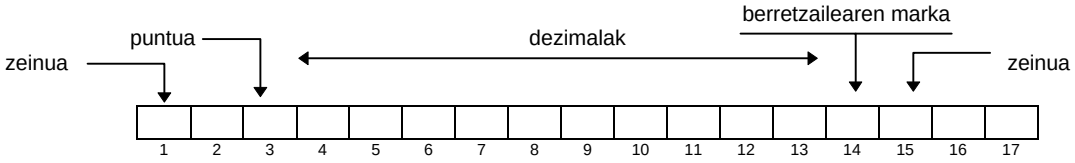
[13] Idazkera zientifikoan Turbo Pascalek 17 luzerako eremu bat jartzen du; hasierako karakterea zuriunea zenbaki positiboetarako eta minus zeinua negatiborako, ondoren mantisa dator zein beti unitatea puntua eta hamar dezimale diren, 14. karakterea exponentea adierazten duen E bat da eta ondokoak berretzailearen zeinua eta balioak dira.

```
PROGRAM IrteerakoFormatua_nm ;                        { \TP70\04\PI.PAS }
USES
      Crt ;
BEGIN
      ClrScr ;
      WriteLn (PI);
      WriteLn (-PI);
      WriteLn ;
      WriteLn (PI:10);
      WriteLn (PI:5);
      WriteLn ;
      WriteLn (PI:17:5);
      WriteLn (PI:10:5);
      WriteLn (PI:0:5);
END
```

Hauxe litzateke `IrteerakoFormatua_nm` izeneko programa honi dagokion irteera:

```
 3.1415926536E+00
-3.1415926536E+00

 3.142E+00
 3.1E+00

          3.14159
   3.14159
3.14159
_
```

Laburbilduz, zenbaki errealentzat Turbo Pascalek darabilen idazkera mantisa normaldua eta berretzailea da (mantisaren dezimal kopurua 10 delarik). Eremuaren zabalera `n` bitartez mugatuz dezimalak ezkutatuko lituzke, `n` oso txikia balitz dezimal bakar bat erakutsiz pantailaratuko luke zenbakia. Formatua `n` eta `m` zenbakiaz zehaztean, `m`-k erakutsiko diren dezimal kopurua da.



4.5.2 Read eta ReadLn prozedurak

Aurreko puntuan ikusi den `Write` eta orain ikasiko dugun `Read` prozedura estandarrak, teklatu-pantaila periferikoz gain, fitxategiekin lan egiteko erabiltzen dira ere.

`Read` eta `ReadLn` prozeduraz teklatuaren bitartez idatziko balioak aldagaietan gordetzen dira, balioak zer aldagaietan biltegitzea parametro bezala zehazten da. Sarrerako prozedura estandar hauek ez dute parametroen kopurua mugatzen, baina gure programa gehienetan balio bat memorizatzeko `ReadLn` bana aplikatuko zaio.

```
ReadLn (parametro1) ;    { gehienetan }
ReadLn (parametro1, parametro2, parametro3, ...) ;
```

Parametro bat baino gehiago irakurri egiten bada, balioak zuriunez banatzen dira. Teklatuaren bitartez programari emango zaion datu oro aldagaiari dagokion datu-motakoa izango da (datu-mota bera edo bestela datu-mota bateragarria). `Read` eta `ReadLn` prozedurak egikaritzean datuak itxaroten programa gelditzen da, horregatik sarrerako prozedura horien

bitartez operadoreak zer datu sartuko duen azaltzen duen `Write` edo `WriteLn` bat idazten da lehenago:

```
Write ('Urtea, hilea eta eguna eman: ') ;
ReadLn (zbkUU, zbkHH, zbkEE) ;
```

`Read` eta `ReadLn` prozedurak antzekoak dira eta euren arteko diferentzia ikusteko `Read_ReadLn` izeneko programa hau aztertuko dugu:

```
PROGRAM Read_ReadLn ;                                          { \TP70\04\IN1.PAS }
USES
      Crt ;
VAR
      zbk1, zbk2, zbk3, zbk4 : Integer ;
BEGIN
      ClrScr ;
      WriteLn ('zbk1=', zbk1, 'zbk2=':8, zbk2, 'zbk3=':8, zbk3, 'zbk4=':8, zbk4 ) ;
      WriteLn ;
      WriteLn ('Hiru aldagaietan gordetzeko, osoak diren lau zenbaki eman:') ;
      Read (zbk1, zbk2, zbk3) ;        { bufferra bete }
      WriteLn ('Laugarren zenbakia buffer memorian dagoenez handik hartu:',);
      Read (zbk4) ;
      WriteLn ('zbk1=', zbk1, 'zbk2=':8, zbk2, 'zbk3=':8, zbk3, 'zbk4=':8, zbk4 ) ;
      WriteLn ;
      WriteLn ('Hiru aldagaietan gordetzeko, beste lau zenbaki oso eman:') ;
      ReadLn (zbk1, zbk2, zbk3) ;      { bufferra ustu }
      WriteLn ('Buffer memorian ezer ez dagoenez, eskatu:',);
      Read (zbk4) ;
      WriteLn ('zbk1=', zbk1, 'zbk2=':8, zbk2, 'zbk3=':8, zbk3, 'zbk4=':8, zbk4 ) ;
END.
```

Programa honen gakoa `Read(zbk1,zbk2,zbk3)` eta `ReadLn(zbk1,zbk2,zbk3)` aginduetan dago. Exekuzio batean `Read`-ari lau zenbaki emanez gero (adibidez `11 22 33` eta `44`) lehenengo hiru balioak `zbk1 zbk2 zbk3` aldagaietan gordetzen dira eta `44` balioa memoriko bufferrean kokatzen da, hurrengo `Read` edo `ReadLn` bateko lehenengo aldagaiari balio hori esleitzen zaio. Bestalde, `ReadLn(zbk1,zbk2,zbk3)` aginduari `111 222 333` eta `444` balioak emanez, bufferra ez da betetzen eta ondorioz azkeneko `444`-a galtzen da eta hurrengo `Read(zbk4)` sarreran programa gelditzen da datu bat eman arte (adibidez `555`):

```
zbk1=0   zbk2=0   zbk3=0   zbk4=0

Hiru aldagaietan gordetzeko, osoak diren lau zenbaki eman:
11 22 33 44
Laugarren zenbakia buffer memorian dagoenez handik hartu:
zbk1=11   zbk2=22   zbk3=33   zbk4=44

Hiru aldagaietan gordetzeko, beste lau zenbaki oso eman:
111 222 333 444
Buffer memorian ezer ez dagoenez, eskatu:
555
zbk1=111   zbk2=222   zbk3=333   zbk4=555
_
```

Laburbilduz, `Read` eta `ReadLn` prozeduretan agertzen diren parametro kopurua, adibidez hiru izanik erabiltzaileak bi modutan erantzun dezake:

1. Parametro kopuruak adierazten duena baino datu gutxiago emanez. Aurreko adibidean datu bat edo bi eman eta gero `RETURN` sakatzen bada, ez da igaroko hurrengo sententzia exekutatzera eta programa geldirik geratuko da falta den datua (edo diren datuak) itxaroten. `Read` eta `ReadLn` prozedurek berdin lan egiten dute.

2. Parametro kopuruak adierazten duena baino datu gehiago emanez. Kasu honetan `Read` eta `ReadLn` prozeduren portaera ez da berdina, aurreko adibidean lau datu eman eta gero `RETURN` sakatzen bada:

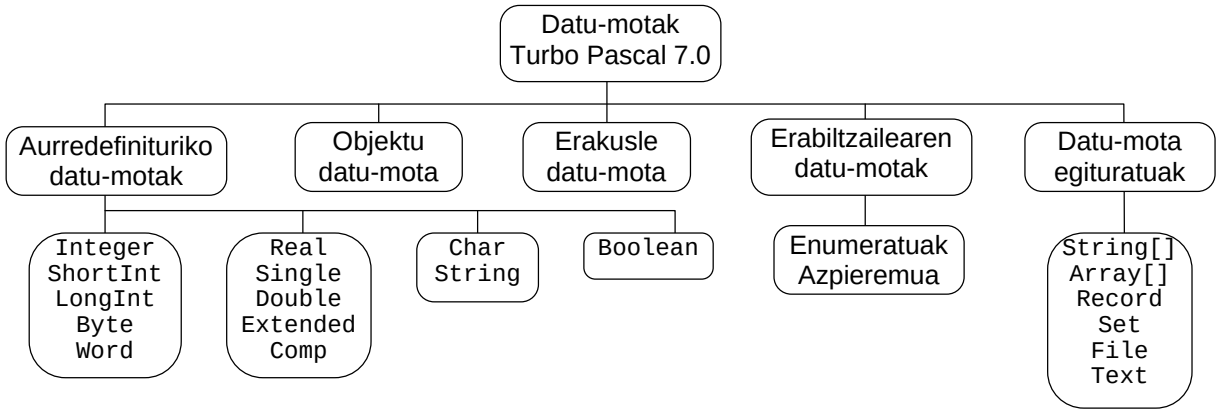
   `Read` prozedura erabiltzean, soberan dagoen datua teklatuaren bufferean mantentzen da, eta hurrengo irakurketa operazioan hartuko da programa gelditu gabe.

   `ReadLn` prozedura erabiltzean, soberan dagoen datua teklatuaren bufferetik kentzen da, eta hurrengo irakurketa operazioan hartuko diren datuak operadoreak emanikoak izango dira, programa gelditu ko da horretarako beta eskainiz.

4.6 DATU-MOTAK EGITURATUAK

Bigarren kapituluan ordenadorearen memoria eta informazioaren biltegitzea memorian aztertu genituen. Konkretuki, sinboloen adierazpidea eta kopuruen adierazpidea ikusi genituen, azkeneko honetan zenbaki osoen eta zenbaki errealen adierazpideak bereiziz. Bigarren kapituluan esandakoa kontutan izanik, orain Turbo Pascal 7.0 programazio lengoaiak eskaintzen dituen biltegitze mekanismoak ikasiko ditugu.

Baina ezer baino lehen testu honetan landuko ditugun datu-moten taula bat erakus dezagun:



Edozein lengoaiak aurredefinituriko dituen datu-motak bi zatiz osatuak agertzen dira:

1. Aurrez determinaturiko memorian biltegitzeko sistema, beste modu batean esanda aldagai batek memoria zenbat toki hartzen duen (memori posizioak finkaturik ondorioz aldagai baten balioen esparrua definiturik geratzen da). Adibidez eta kurtsoaren bigarren kapitulua gogoratuz, kopuru frakzionatuak adierazteko konputazio sistema jakin batean “koma higikorra” metodoa erabil daiteke.
2. Datu-mota horrentzat egokiak diren eragiketak definiturik egongo dira. Ongi mugatutako eragiketak kolekzio bat osatzen dute, operaziok sintaxia naturala gordetzen dutenez euren erabilpen erraza ziurtaturik dago. Esate baterako, kopuru frakzionatuen datu-motarako definiturik izango diren eragiketak batuketa, kenketa, biderkaketa eta zatiketa izan daitezke.

Hauek dira ikasiko direnak: `String`, `Array`, `Record`, `Set`, `File`, `Text` erakusleak eta objektuak.

4.6.1 STRING datu-mota

`String` datu-motako aldagaiak hitzak edo esaldiak gordetzeko balio du, string bat zenbait karakterez osaturiko katea izango da. Horregatik `String` datu-motako 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 `Esaldia`, `KateLabur`, `KateLuze` eta `KateLuzeena` karaktere-kateek zenbat byte hartzen duten memorian azter daiteke:

```
PROGRAM StringAldagaienLuzera ;                    { \TP70\04\KATEAK.PAS }
USES
    Crt ;
VAR
   Esaldia : String ;          { 255 karaktere gorde ahal izateko aldagaia }
   KateLabur : String[10] ;    {  10 karaktere gorde ahal izateko aldagaia }
   KateLuze : String[100] ;    { 100 karaktere gorde ahal izateko aldagaia }
   KateLuzeena : String[255] ; { 255 karaktere gorde ahal izateko aldagaia }
   Itxaron : Char ;
BEGIN                    { Programa Nagusia }
   ClrScr ;
   WriteLn ('STRING ALDAGAIEN MEMORI POSIZIOEN BEHARRAK') ;
   WriteLn ('=========================================') ;
   WriteLn ('Esaldia -----------> ', SizeOf (Esaldia):3, ' byte') ;
   WriteLn ('KateLabur ---------> ', SizeOf (KateLabur):3, ' byte') ;
   WriteLn ('KateLuze ----------> ', SizeOf (KateLuze):3, ' byte') ;
   WriteLn ('KateLuzeena -------> ', SizeOf (KateLuzeena):3, ' byte') ;
   Itxaron := ReadKey ;
END .
```

Hona hemen `StringAldagaienLuzera` programaren ireera:

```
STRING ALDAGAIEN MEMORI POSIZIOEN BEHARRAK
=========================================
Esaldia -----------> 256 byte
KateLabur ---------->  11 byte
KateLuze ----------> 101 byte
KateLuzeena -------> 256 byte
_
```

`String` datu-motako aldagai batean karaktereak gorde daitezke eta katearen osagai bakoitzaren atzipena indibidualki egiteko karakterearen posizioa zehaztu beharra dago. Adibidez, `Izena` aldagaiaren lehen karakterea pantailaratzeko sententzia hau da:

```
WriteLn (Izena[1]) ;
```

Karaktere-katea baten osagaien banan-banako atzipen kontzeptua `Karaktereka` izeneko programa honetan ikus daiteke:

```
PROGRAM Karaktereka ;                          { \TP70\04\KATEAK1.PAS }
CONST
   MAX = 7 ;
VAR
   Izena : String[MAX] ;   { 7 karaktere gordetzeko aldagaiak }
   Kont : Byte ;
BEGIN
   Write ('Zure izena eman: ') ;
   ReadLn (Izena) ;
   WriteLn (Izena) ;
   FOR Kont:=1 TO MAX DO
      Write (Izena[Kont], '-') ;
   WriteLn ;
END .
```

4.6.2 ARRAY datu-mota

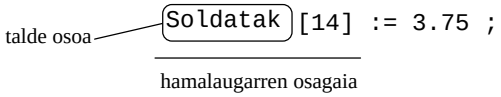
Zenbaitetan datuek taldeak osatzen dituzte, esate baterako enpresa baten langileen soldatak zenbakien talde bat litzateke, nahiz eta taldearen osagai bakoitzak bere balio propioa izan denak “elkarrekin” memorian biltegitzea komeni daiteke. Horretarako, array datu-mota erabil daiteke.

Array bat definitzeko osagai guztien kopurua indize baten bitartez zehaztu beharra dago, osagai bakoitzaren atzipena (`String` datu-motaren kasuaren antzera) array aldagaiaren identifikadoreaz eta taldean duen posizioaz lortzen da. Hurrengo adibidean `Soldatak` izena duen zenbaki errealen array bat definituko da, ikusten denez taldean gehienez ehun zenbaki erreal gorde ahal izango dira:

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

Indizeak zeregin bikoitza du array-etan, batetik memoria erreserba gauzatzeko ezin bestekoa da eta bestetik taldeko datu indibidualak eskuratzeko erabiltzen da. Adibidea gogoan izanik lehenengo soldatari dagokion indizea `1` da eta azkenekoari `100`, beraz guztira ehun zenbaki gorde daitezke `Soldatak` aldagaian.

Hurrengo sententziak `Soldatak` aldagaiaren osagai bati (hamalaugarrenari) balioa emateko balio du:



4.6.3 RECORD datu-mota

Array datu-motaren ezaugarri bat osagaien homogeneotasuna izanik, batzutan datu talde baten adierazpiderako array-ak erabiltzea ezinezkoa bihurtzen da. Adibidez, lantegi bateko langilea kontzeptualizatzeko bere ezaugarrietan oinarritzen bagara, azkar konturatzen gara ezaugarriok datu-mota ezberdinez definitzen direla:

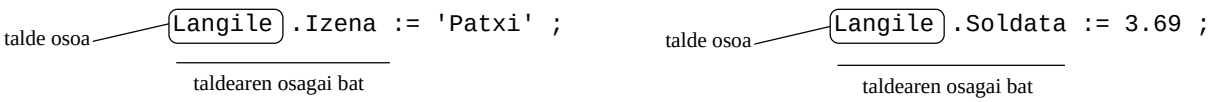
Langilearen izena:	`String` datu-mota
Langilearen kategoria:	`Char` datu-mota
Langilearen soldata:	`Real` datu-mota
Langilearen adina:	`Byte` datu-mota

Halakoetan `Record` datu-motako aldagai bakar batek informazio guzti hori “elkarturik” biltegi dezake, `Record` edo erregistroaren osagai bakoitzari **eremu** esaten zaio eta kasu honetan indizerik erabili beharrean eremuaren identifikadoreaz erreferentzia daiteke `Record`-aren edozein osagai. Hau litzateke `Langile` erregistroaren definizioa:

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

Eremuetan datuak gordetzeko edo bertatik datuen atzipena gauzatzeko erabiliko den moldea honako hau da: erregistroaren identifikadorea eta eremuaren identifikadorea puntu kualifikadorez banaturik.

`Langile` erregistroaren datuak gorde ditzagun:



`Langile` erregistroarekin lan egiten ikasteko asmoz `ErregistroAldagaia` programa prestatu dugu, zeinek honelako kodea duen:

```
PROGRAM ErregistroAldagaia ;                    { \TP70\04\ERREGI.PAS }
VAR
  Langile : RECORD
               Izena : String ;
               Maila : Char ;
               Dirua : Real ;
               Adina : Byte ;
            END ;
BEGIN
   Write ('Langilearen izena eman: ') ;
   ReadLn (Langile.Izena) ;
   Write (Langile.Izena, '-(r)en kategoria eman: ') ;
   ReadLn (Langile.Maila) ;
   Write (Langile.Izena, '-(r)en soldata eman: ') ;
   ReadLn (Langile.Dirua) ;
   Write (Langile.Izena, '-(r)en adina eman: ') ;
   ReadLn (Langile.Adina) ;
   WriteLn ;
   WriteLn ('RECORD ALDAGAIAREN ETA EREMUEN MEMORI POSIZIOEN BEHARRAK') ;
   WriteLn ('=======================================================') ;
   WriteLn ('Langile.Izena -----> ', SizeOf (Langile.Izena):3, ' byte') ;
   WriteLn ('Langile.Maila -----> ', SizeOf (Langile.Maila):3, ' byte') ;
   WriteLn ('Langile.Dirua -----> ', SizeOf (Langile.Dirua):3, ' byte') ;
   WriteLn ('Langile.Adina -----> ', SizeOf (Langile.Adina):3, ' byte') ;
   WriteLn ('Langile -----------> ', SizeOf (Langile):3, ' byte') ;
END.
```

Hauxe `ErregistroAldagaia` programaren irteera bat:

```
Langilearen izena eman: Koldo
Koldo-(r)en kategoria eman: C
Koldo-(r)en soldata eman: 4.67
Koldo-(r)en adina eman: 39

RECORD ALDAGAIAREN ETA EREMUEN MEMORI POSIZIOEN BEHARRAK
=======================================================
Langile.Izena -----> 256 byte
Langile.Maila ----->   1 byte
Langile.Dirua ----->   6 byte
Langile.Adina ----->   1 byte
Langile -----------> 264 byte
_
```

4.6.4 SET datu-mota

`Set` datu-motari multzoa esaten zaio eta `Byte` edo `Char` elementuz osaturik dago.

`Set` datu-mota azaldu aurretik erabiltzailearen datu-motak sortzeko `Type` klausula azaldu beharra dagoenez hamaikagarren kapitulurako utziko ditugu `Set` datu-motak merezi dituen azalpenak.

4.6.5 FILE eta TEXT datu-motak

Fitxategiek duten ezaugarririk garrantzitsuena gordetzen duten informazioa diskoan kokatzen dela eta ez memorian, horregatik informazioo iraunkorra dela kontsidera daiteke. `File` edo fitxategi datu-mota, array-ak bezala, hainbat elementu homogeneoz osaturiko taldea litzateke[14]. Hona hemen fitxategi aldagai biren erazagupenak:

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

Fitxategiaren oinarrizko elementuaren datu-mota karakterea denean testu-fitxategiak sor daitezke, Turbo Pascal lengoaiak horientzat `Text` datu-mota aurredefiniturik du. Adibidez:

```
VAR
   Datuak : TEXT ;
```

Informazio iraunkorra tratatzen dituzten fitxategiak hamabigarren kapituluan garatzen dira.

4.6.6 Erakusle datu-mota

Erakuslea[15] memoria dinamikoa erabiltzeko datu-mota aproposa da, memoriaren banaketa eta memoria dinamikoaren kontzeptuak hamahirugarren kapituluan ikusiko dira.

Erakusle datu-mota ikasi ondoren, garrantzi handikoak diren egitura dinamikoak (zerrenda kateatuak, pilak, ilarak eta zuhaitzak) lantzeko aukera izango dugu.

4.6.7 Objektu datu-mota

Turbo Pascal lengoaiak Turbo Pascal lengoaiak objektu edo `Object` datu-mota onartzen du. `Object` datu-motek erregistroen antza dute haiek bezala eremutan banatzen direlako, objektuek eremuaz gain metodoak ere barneratzen dituzte.

`Object` datu-motaren interesa handia da eta kapitulu oso bat, hamalaugarrena alegia, dedikatu diogu.

[14] Ikusten denez fitxategiak deklaratzean ez da euren muga markatzen duen indizerik jartzen, izan ere fitxategi baten elemntuen kopurua ez dago definiturik.
[15] Erakusle datu-motari pointer esaten zaio ere.

4.7 PROGRAMAK

Hona hemen 4. kapituluaren programak orrialdeen arabera sailkaturik:

Izena	*Programaren identifikadorea*	*ORRI.*	*Ikasgaia*
MEZUA.PAS	KonstanteaPantailaratzen	4-06	Pantailaraketa bat
MEZUAK.PAS	KonstanteakPantailaratzen	4-06	Pantailaraketa bat
ALDAGAI.PAS	AldagaiaPantailaratzen	4-07	Pantailaraketa bat
BARNE1.PAS	ZenbakiOsoenBarneAdierazpidea	4-09	Memoriaren edukia
ESPRE1.PAS	OsoenAdierazpenAritmetikoak	4-12	Espresioak
GAINEZ1.PAS	BytenGainezkada	4-13	Memoriaren edukia
GAINEZ2.PAS	IntegerrenGainezkada	4-14	Memoriaren edukia
HIGH_LOW.PAS	High_eta_Low_funtzioak	4-15	Datu-moten esparruak
KOPRO.PAS	Koprorekin	4-16	Konpilazio direktibak
ESPRE2.PAS	Datu_motenBateragarritasuna	4-18	Espresioak
BOOLEHE.PAS	BolearrenLehentasuna	4-21	Espresio boolearrak
URTEBETE.PAS	UrtebetetzeaAsmatu	4-22	Espresio boolearrak
KARAK1.PAS	Karaktere_KonstanteEtaAldagaiak	4-23	Karaktereak
KARAK2.PAS	KaraktereenEragiketak	4-23	Karaktereak
KARAK3.PAS	Ord_eta_Chr	4-24	Karaktereak
ASCII.PAS	ASCII_taula	4-25	Karaktereak
CRT.PAS	CRT_unitatea	4-26	Unitateak
DATUMOTA.PAS	DatuMotaBerria	4-28	Karaktereak
OUT1_N.PAS	IrteerakoFormatua1_n	4-32	`WriteLn()`, formatuak
OUT2_N.PAS	IrteerakoFormatua2_n	4-33	`WriteLn()`, formatuak
PI.PAS	IrteerakoFormatua_nm	4-34	`WriteLn()`, formatuak
IN1.PAS	Read_ReadLn	4-35	`Read()` eta `ReadLn()`
KATEAK.PAS	StringAldagaienLuzera	4-37	`String` datu-mota
KATEAK1.PAS	Karaktereka	4-37	`String` datu-mota
ERREGI.PAS	ErregistroAldagaia	4-39	`Record` datu-mota

4.8 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

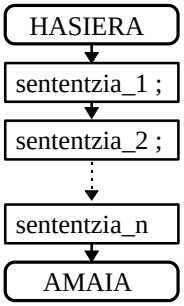
5. ATALA: BALDINTZAK ETA ERREPIKAPENAK

AURKIBIDEA

5.1 SARRERA

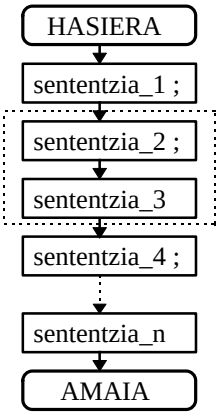
Orain arteko programa guztietan agertzen ziren sententzi guztiak exekutatu egiten ziren programa egikaritzean. Sententzia guztiak sekuentzialki exekutatzen ziren eta behin bakarrik exekutatzen ziren programa martxan jartzean.

Hau litzateke sententzia sekuentzialen fluxu diagrama:




```
sententzia_1 ;
sententzia_1 ;
   .
   .
   .
sententzia n
```

Orain arte erabilitako sententziak bakunak izan dira (sententziek ez dute multzorik osatzen), sententzien taldeak osatzeko Turbo Pascal lengoaian `BEGIN` eta `END` hitz erreserbatuen bidez egiten da. Multzoa osatzen duten sententzia taldea sententzi konposatu bakarra bezala uler daiteke, ondoko irudian bigarren eta hirugarren sententziak multzo batean elkartu dira:




```
sententzia_1 ;
BEGIN
    sententzia_2 ;
    sententzia_3
END ;
sententzia_4 ;
   .
   .
   .
sententzia n
```

5.2 BALDINTZAZKO AGINDUAK

Turbo Pascal lengoaian baldintzazko aginduak hiru dira:

- `IF-THEN`
- `IF-THEN-ELSE`
- `CASE-OF`

Hurrengo puntuetan banan-banan ikusiko ditugu.

5.2.1 IF-THEN baldintzazko sententzia

Gehienetan aginduak sekuentzialak izaten dira, baina zenbait kasutan programaren sententzia bat (edo sententzia multzo bat) ez exekutatzea interesatzen zaigu. Adibidez, ondoko ekuazioan `zatikizuna` eta `zatitzailea` aldagaiak teklatutik irakurtzen badira `x` emaitza kalkula daiteke.

```
PROGRAM ErroreakAurreikusiGabe ;              { \TP70\05\IF1.PAS }
VAR
      zatikizuna, zatitzailea, x : Integer ;
BEGIN
      Write (‘Zatikizunaren balioa eman: ‘) ;
      ReadLn (zatikizuna) ;
      Write (‘Zatitzailaren balioa eman: ‘) ;
      ReadLn (zatitzailea) ;
      x := zatikizuna DIV zatitzailea ;
      WriteLn (zatikizuna, ‘ zati ‘, zatitzailea, ‘ = ‘, x) ;
      WriteLn (‘------------------------------‘) ;
END.
```

Iturburu programa konpilatu ondoren exekutatzen badugu, honako hau azal daiteke monitorearen pantailan:

```
Zatikizunaren balioa eman: 19
Zatitzailearen balioa eman: 3
18 zati 3 = 6
------------------------------
_
```

Zatiketa osoa aplikatzen ari garenez, lortzen den emaitza `6` da eta ondo egongo litzateke. Baina zer gertatuko litzateke zatitzailearen balioa `3` izan beharrean `0` balitz?. Ikus dezagun:

```
Zatikizunaren balioa eman: 19
Zatitzailearen balioa eman: 0
Runtime error 200 at 0BF2:00BD.
_
```

Hau litzateke irteerako pantailaren adibide bat, eta ingurune integratuan ari bagara `ErroreakAurreikusiGabe` programaren edizio-leihora itzuliko ginateke eta bertan errorea non suertatu den agertuko litzateke. Argi dagoenez, ordenadoreak ezin du zero arteko zatiketa burutu eta gure programa ahula da halako kasuak aintzat hartzen ez dituelako.

Gure arazoa konponduko litzateke `x := zatikizuna DIV zatitzailea ;` sententzia exekutatu baino lehen zatitzailearen balioa zero ez dela testatzeko ahalmena izango bagenu. Horretarako hain zuzen ere, baldintzazko aginduak definiturik daude goimailako edozein lengoaiatan. Gure programak honelako itzura edukiko luke:

```
PROGRAM ErroreakAurreikusten ;                { \TP70\05\IF2.PAS }
VAR
      zatikizuna, zatitzailea, x : Integer ;
BEGIN
      Write (‘Zatikizunaren balioa eman: ‘) ;
      ReadLn (zatikizuna) ;
      Write (‘Zatitzailaren balioa eman: ‘) ;
      ReadLn (zatitzailea) ;
      (* zatitzailea 0 ez denean hau egin: *)
            x := zatikizuna DIV zatitzailea ;
            WriteLn (zatikizuna, ‘ zati ‘, zatitzailea, ‘ = ‘, x) ;
      WriteLn (‘------------------------------‘) ;
END.
```

Lehengoarekin alderatuz, ErroreakAurreikusten programa honetan ez dugu gauza larregirik egin: iruzkin bat tartekatu eta azken sententziaren aurreko biak eskubirago idatzi. Horregatik ErroreakAurreikusten programaren exekuzioa eta ErroreakAurreikusiGabe programarena berdinak lirateke, izan ere iruzkinek programa konpilatuan ez dute eraginik eta sententziak multzoka jartzeko BEGIN eta END markaz mugatu behar dira. Baina aldaketa horiek IF-THEN sententzia ulertzeko balio digute, hona hemen ErroreakAurreikusten programa erabat idatzirik:

```
PROGRAM ErroreakAurreikusten ;                { \TP70\05\IF2.PAS }
VAR
       zatikizuna, zatitzailea, x : Integer ;
BEGIN
       Write ('Zatikizunaren balioa eman: ') ;        (* sententzia_1 *)
       ReadLn (zatikizuna) ;                          (* sententzia_2 *)
       Write ('Zatitzailaren balioa eman: ') ;        (* sententzia_3 *)
       ReadLn (zatitzailea) ;                         (* sententzia_4 *)
       IF zatitzailea <> 0 THEN                       (* sententzia_5 *)
         BEGIN
              x := zatikizuna DIV zatitzailea ;
              WriteLn (zatikizuna, ' zati ', zatitzailea, ' = ', x)
         END ;
       WriteLn ('------------------------------') ;   (* sententzia_6 *)
END.
```

ErroreakAurreikusten programak sei sententzia ditu, baldintzazko agindua den bosgarren sententzia IF hitz erreserbatuan hasten da eta END osteko puntu eta koman amaitzen da. Iturburu programa berria konpilatu eta lehen eman diren datuak sartzean, honako hau azal daiteke monitorearen pantailan:

```
Zatikizunaren balioa eman: 19
Zatitzailearen balioa eman: 3
18 zati 3 = 6
------------------------------
_
```

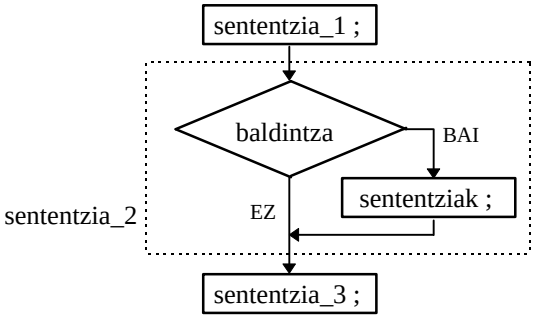
Izan ere zatitzailea 3 balio denez IF eta THEN artean dagoen espresio boolearrak hartzen duen balio TRUE da. Ondorioz, THEN ostean dagoen blokeko sententziak exekutatzen dira (zatiduraren kalkulua eta pantailaraketa). Programa bukatu aurretik 6. sententzia (marratxoak idaztea) exekutatuko da ere.

19 eta 0 datuak sartzean programaren irteera desberdina da. Hasierako lau sententziak (eta azkeneko seigarrena) lehenago bezala exekutatuko dira, ez da gauza bera gertatuko bosgarrenarekin. IF eta THEN artean dagoen galdera zatitzailea <> 0 izanik, eta sarreren balioen arabera honela planteaturik dago 0 <> 0 eta hori beti gezurra izango da, eta ondorioz THEN ostean dagoen blokea baztertu egiten da exekutatu gabe geratuz.

```
Zatikizunaren balioa eman: 19
Zatitzailearen balioa eman: 0
------------------------------
_
```

IF-THEN baldintzazko aginduak galdera[1] bat eta sententzia multzo bat integraturik ditu. Galderaren erantzuna egia izatean (TRUE Turbo Pascalean) multzoaren sententziak egikaritu eta IF-THEN aginduaren jarraian datorren hurrengoari txanda egokituko zaio. Galderaren erantzuna gezurra bada (FALSE Turbo Pascalean) multzoaren barneko sententziak ez dira exekutatzen eta besterik gabe IF-THEN agindua amaitutzat jo daiteke, txanda bere jarraian datorren sententziak hartuko luke. Grafikoki:

[1] Gogoratu laugarren kapituluan **4.3.3 Datu-mota boolearrak** eta **4.3.3.1 Adierazpen boolearrak** izenburuko puntuetan esandakoa.




```
sententzia_1 ;

IF baldintza THEN
  BEGIN
    sententziak ;
  END ;

sententzia_3 ;
```

5.2.1.1 Adibidea

Demagun bigarren graduko ekuazio bat dugula. Eta ekuazioari dagozkion erroak zer motatakoak diren jakin nahi dugula.

$A\ x^2 + B\ x + C = 0$ datuak: A, B eta C

emaitza: `erroak errealak dira`
`erroa erreala da`
`erroak irudikariak dira`

Algoritmoa:

1. Diskriminantea kalkulatu $Diskr = B^2 - 4\ A\ C$
2. Diskriminantearen arabera pantailaratu:

`Diskr > 0`	hau idatzi	`erroak errealak dira`
`Diskr = 0`	hau idatzi	`erroa erreala da`
`Diskr < 0`	hau idatzi	`erroak irudikariak dira`

```
PROGRAM NolakoErroakDiren ;                    { \TP70\05\ERROAK1.PAS }
VAR
      A, B, C, Diskr : Real ;
BEGIN
      Write (‘A koefizientearen balioa eman: ‘) ;
      ReadLn (A) ;
      Write (‘B koefizientearen balioa eman: ‘) ;
      ReadLn (B) ;
      Write (‘C koefizientearen balioa eman: ‘) ;
      ReadLn (C) ;

      Diskr := sqr(B) - 4*A*C ;

      IF Diskr > 0 THEN
            WriteLn (‘erroak errealak dira‘) ;
      IF Diskr = 0 THEN
            WriteLn (‘erroa erreala da‘) ;
      IF Diskr < 0 THEN
            WriteLn (‘erroak irudikariak dira‘) ;
END.
```

Koefizienteen arabera diskriminantea kalkulatzean `sqr()` funtzio estandarra erabiltzen da, honek parametroaren karratua itzultzen du (xehetasun gehiagorako kurtsora `sqr` azpian kokatu eta laguntza eskatu `Ctrl+F1` teklen bitartez).

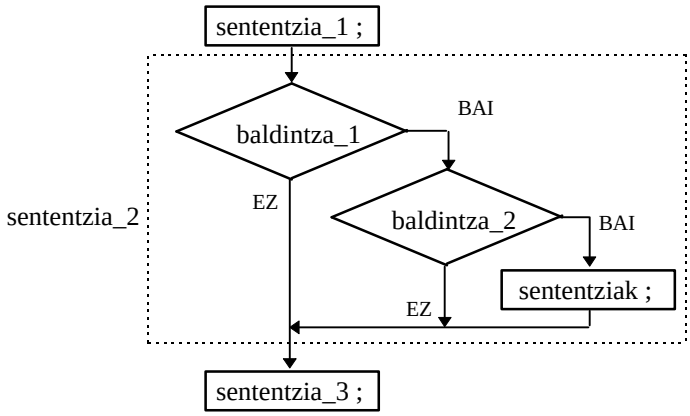
Diskriminantearen balioa finkaturik dagoela, ondoko hiru `IF-THEN` sententzietatik batek bakarrik betetzen du egindako galderak, beraz horri dagokion mezua agertuko da eta beste biak saihesturik eta pantailaratu gabe geratuko dira. `IF-THEN` sententzia bakoitzarekin batera agindu bakarra dagoenez ez dago zertan `BEGIN-END` mugarriz multzorik eratu behar.

5.2.1.2 IF-THEN kabiatuak

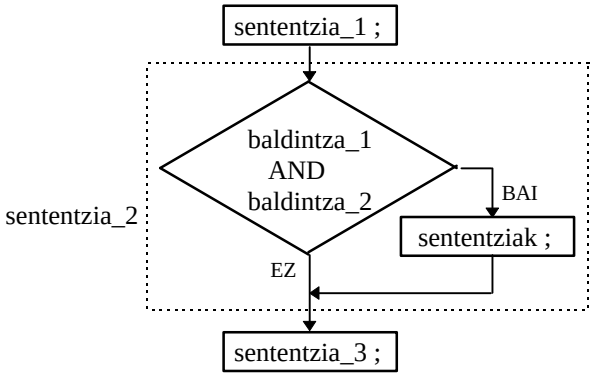
`IF-THEN` sententziaren batek beste `IF-THEN` sententzia bat barneratzen duenean kabiaturik daudela esaten da. Dakigunez, egikatzarapen-denboran eta sarturiko datuen arabera `IF` biren menpeko sententzia exekutatuko da ala ez.

Kabiatuak diren baldintzazko sententziek honako hau adierazten dute: "baldin *baldintza1* egia den eta beste baldintza (*baldintza2*) ere egia den, orduan *sententziak* egikaritu.

Argi dagoenez baldintza biak egia izan behar dira menpeko agindu multzora iritsi ahal izateko, horregatik `IF-THEN` kabiatuak `IF-THEN` sententzia bakar batek ordezka ditzake, ikus ondoko eskemak:




```
sententzia_1 ;

IF baldintza_1 THEN
  IF baldintza_2 THEN
    BEGIN
         sententziak ;
    END ;

sententzia_3 ;
```

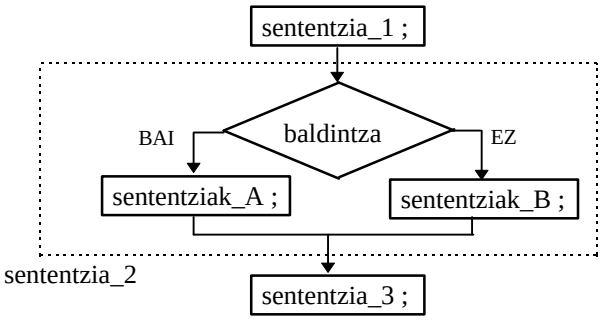



```
sententzia_1 ;

IF (baldintza_1) AND (baldintza_1) THEN
  BEGIN
       sententziak ;
  END ;

sententzia_3 ;
```

Baldintzazko sententzia kabiatu biren kondizioak `AND` eragilearen bitartez elkar daitezke adierazpen boolear konposatu bat lortuz, (eragigai biak `TRUE` izatean baldintza konposatua beteko da). Programatzeko modu biak ondo daude baina aitortu beharrra dago `AND` eragilearen bidea geihago erabiltzen dela.

5.2.2 IF-THEN-ELSE baldintzazko sententzia

`IF-THEN-ELSE` sententzia batek, elkar baztertzaileak diren bi aukeren artean bat hautatzeko balio du. Sententzia hau ere adierazpen boolear batez kontrolatzen da, baina irudiak argitzen duen bezala, eta datuen arabera, baldintza eta geroko bi bideetatik bat derrigorrez egikaritzen da bestea saihestuta geratzen delarik:




```
sententzia_1 ;

IF baldintza THEN
  BEGIN
    sententziak_A ;
  END
ELSE
  BEGIN
    sententziak_B ;
  END ;

sententzia_3 ;
```

5.2.2.1 Adibidea

5.2.1 puntuan idatzi den `ErroreakAurreikusten` izeneko programa `IF-THEN-ELSE` sententzia batez osa dezagun. Izan ere, sarrerako datuaren arabera `zatitzailea` etiketaz zehazturiko aldagaiak zero izango da ala bestela zero ez den balioren bat hartuko du. Beraz, `IF-THEN-ELSE` adibide aproposa litzeteke:

```
PROGRAM ErroreakAurreikustenIfThenElse ;            { \TP70\05\IF3.PAS }
VAR
      zatikizuna, zatitzailea, x : Integer ;
BEGIN
      Write ('Zatikizunaren balioa eman: ') ;       (* sententzia_1 *)
      ReadLn (zatikizuna) ;                         (* sententzia_2 *)
      Write ('Zatitzailaren balioa eman: ') ;       (* sententzia_3 *)
      ReadLn (zatitzailea) ;                        (* sententzia_4 *)
      IF zatitzailea = 0 THEN                       (* sententzia_5 *)
        BEGIN
            WriteLn (zatikizuna, ' zati ', zatitzailea, ' = INFINITO')
        END
      ELSE
        BEGIN
            x := zatikizuna DIV zatitzailea ;
            WriteLn (zatikizuna, ' zati ', zatitzailea, ' = ', x)
        END ;
      WriteLn ('------------------------------') ;   (* sententzia_6 *)
END.
```

`ErroreakAurreikustenIfThenElse` programak ere sei sententzia ditu, baldintzazko agindua den bosgarren sententziada eta `IF` hitz erreserbatuan hasten da eta bigarren `END` osteko puntu eta koman amaitzen da, `ELSE` baino lehen dagoen `END` ostean ez da puntu eta komarik jarri behar, bestela konpilatzerakoan `IF-THEN-ELSE` sententzia gaizki interpretatuko litzateke (`ELSE` baino lehen dagoen `END` ostean puntu eta komarik jartzean `IF-THEN` agindu bat lortuko genuke eta jarraian datorren `ELSE` hitzak ez luke zentzurik izango konpilazio-errorea eraginez).

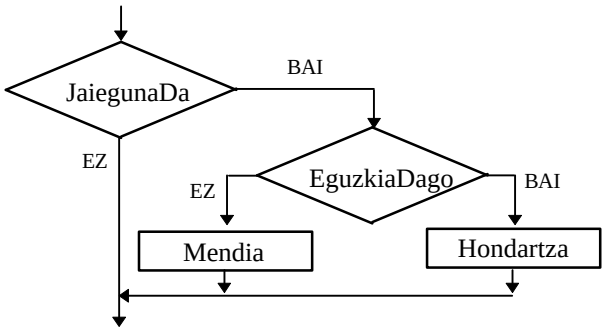
5.2.2.2 IF-THEN-ELSE kabiatuak

Elkar baztertzaileak diren bi aukeren artean bat hautatzeko `IF-THEN-ELSE` sententzia aproposena da. Zenbaitetan, sententzia hau eta `IF-THEN` sententzia kabiatu egiten dira eta halakoetan ez da argi geratzen `ELSE` zatia zein `IF`-ekin doan. Adibidez, demagun ondoko egoera Turbo Pascal lengoaiak darabilen sintaxiaz baliaturik programatu nahi dugula: *Jaieguna bada mendira edo hondartzara joango gara, baina hondartzara joateko eguna eguzkitsua izango da*.

Horretarako aldagai boolear bikote bat daukagu:

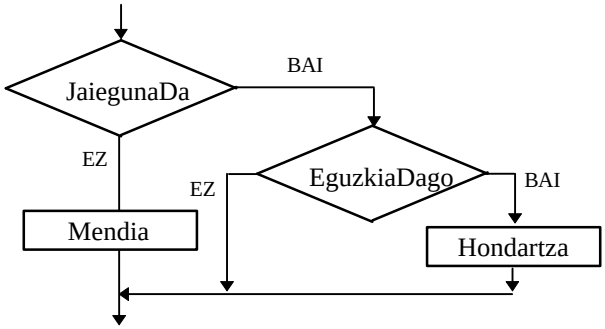
Proposizioa	Aldagaia
Eguraldi eguzkitsua	`EguzkiaDago`
Jaieguna denez lanik ez	`JaiegunaDa`

Programaren eskema hauxe litzeteke:




```
IF JaiegunaDa THEN
  IF EguzkiaDago THEN
    BEGIN
         WriteLn (‘ Hondartza ‘) ;
    END
  ELSE
    BEGIN
         WriteLn (‘ Mendia ‘) ;
    END ;
```

Eta ondorizta daitekeena hauxe da: Bi `IF` jarraian daudenean eta `ELSE` bakar bat agertzen denean, `ELSE` hori barneko (azkeneko) `IF`-arekin loturik dago. Arau hau beti beteko da eta `ELSE` zatia kanpoko (lehenengo) `IF`-arekin elkartu nahi badugu ezin da puntu eta komarik jarri, `BEGIN-END` bitartez moldatu behar da. Adibidez, hau programatzeko: *Jaieguna bada eta egun eguzkitsua bada hondartzara joango gara, bestela (jaieguna ez bada) mendira joko dugu*.




```
IF JaiegunaDa THEN
  BEGIN
    IF EguzkiaDago THEN
      BEGIN
        WriteLn (‘ Hondartza ‘) ;
      END ;
  END
ELSE
  BEGIN
    WriteLn (‘ Mendia ‘) ;
  END ;
```

Demagun bigarren graduko ekuazio bat dugula. Eta ekuazioari dagozkion erroak zer motatakoak diren eta zenbat balio duten jakin nahi dugula.

$A\ x^2 + B\ x + C = 0$ datuak: A, B eta C

emaitza:
```
erroak errealak dira: Erro1 eta Erro2
erroa erreala da: Erro1
erroak irudikariak dira
```

Algoritmoa:

1. Diskriminantea kalkulatu $Diskr = B^2 - 4\ A\ C$
2. Diskriminantearen arabera pantailaratu:

`Diskr > 0`	hau idatzi	`erroak errealak dira: Erro1 eta Erro2`
`Diskr = 0`	hau idatzi	`erroa erreala da: Erro1`
`Diskr < 0`	hau idatzi	`erroak irudikariak dira`

```
PROGRAM ErroakKalkulatzen ;                    { \TP70\05\ERROAK2.PAS }
VAR
      A, B, C, Diskr, Erro1, Erro2 : Real ;
BEGIN
      Write ('A koefizientearen balioa eman: ') ;
      ReadLn (A) ;
      Write ('B koefizientearen balioa eman: ') ;
      ReadLn (B) ;
      Write ('C koefizientearen balioa eman: ') ;
      ReadLn (C) ;

      Diskr := sqr(B) - 4*A*C ;

      IF Diskr < 0 THEN
            WriteLn ('erroak irudikariak dira')
      ELSE
         BEGIN
            IF Diskr > 0 THEN
              BEGIN
                   Erro1 := (-B + sqrt(Diskr)) / (2*A) ;
                   Erro2 := (-B - sqrt(Diskr)) / (2*A) ;
                   Write ('erroak errealak dira: ') ;
                   WriteLn (Erro1:0:2,' eta ', Erro2:0:2) ;
              END
            ELSE
              BEGIN
                   Erro1 := -B / (2*A) ;
                   WriteLn ('erroa erreala da: ', Erro1:0:2) ;
              END ;
         END ;
END.
```

Koefizienteen arabera diskriminantea kalkulatu ondoren hiru posibilitatetik unekoa aukeratzen da. `IF-THEN-ELSE` baten bitartez ebaluatu ondoren, diskriminantea negatiboa izatean `erroak irudikariak dira` mezua agertzen da, bestela (0 edo positiboa izatean) berriro erabaki behar da zein bidetatik joan, galdera hau `IF-THEN-ELSE` egituraz gauzatu da ere. Diskriminantearen erro karratua lortzeko `sqrt()` funtzio estandarra erabiltzen da, honek parametroaren erro karratua itzultzen du (xehetasun gehiagorako kurtsora `sqrt` azpian kokatu eta laguntza eskatu `Ctrl+F1` teklen bitartez).

5.2.3 CASE-OF baldintzazko sententzia

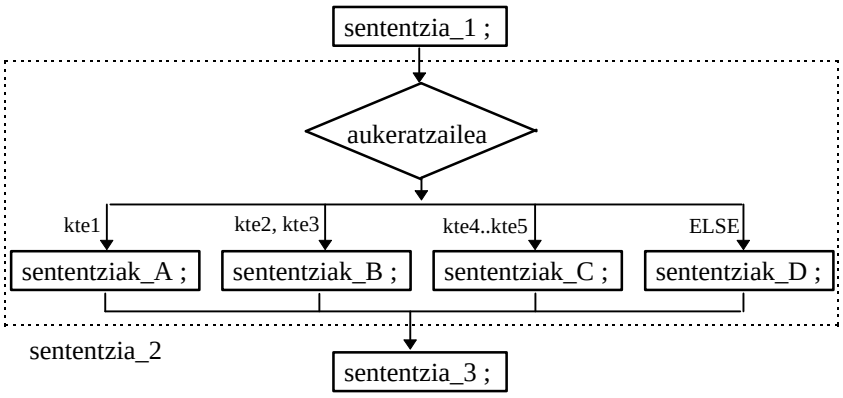
`IF-THEN` edo `IF-THEN-ELSE` bezalako sententziak adierazpen boolear batez kontrolatzen dira, eta datu-mota boolearrak balio bakar bi har ditzakeenez (`TRUE` eta `FALSE`) horrelako egiturek gehinez bi auken artean hautapena egiteko balioko dute. Askotan aukera gehiagoren artean hautapena egin ahal izatea gertatzen da, eta orduan anitz `IF-THEN` konbinatuz ebatziko genuke arazoa. Baina Turbo Pascal lengoaiak badu horrelakoetarako oso egokia den egitura, `CASE-OF` delako egitura hain zuzen ere.

`IF-THEN` eta `IF-THEN-ELSE` sententzietan ez bezala, `CASE-OF` batek aldagai batez kontrolatzen da. `CASE-OF` egituraren kontrol-aldagai horren datu-mota ezin izango da `Real` izan, egia esan kontrol-aldagaiaren datu-mota edozein ordinala izango da (`Integer` familiakoak, `Char` eta `Boolean`).

`CASE-OF` sententziak `BEGIN` -ik ez du baina bai bere `END` propioa, non amaitzen den.

`CASE-OF` sententzia exekutatu aurretik kontrol-aldagaiak balioren bat hartuko du, eta duen balioa `CASE-OF` sententzian agertzen diren konstanteekin konparatzen du, eta horren arabera izan daitezkeen bide guztietatik bat edo beste aukeratuko da. Parekotasunik aurkitzen ez bada ezer egin gabe hurrengo sententzia igaroko da.

Hona hemen `CASE-OF` egituraren logika azaltzen duen irudia:




```
sententzia_1 ;

CASE aukeratzailea OF
  kte1: BEGIN
          sententziak_A ;
        END ;
  kte2, kte3: BEGIN
                sententziak_B ;
              END ;
  kte4..kte5: BEGIN
                sententziak_C ;
              END ;
  ELSE
       sententziak_D ;
END ;

sententzia_3 ;
```

Demagun ondoko programan nota bat adierazten duen zenbaki oso bat sartzen dugula, programaren ireteera notari dagokion testua izango da. Lehenago esan den bezala kontrol-aldagaia erreala ez da izango beraz `NotakPantailaratzen` programan aukeratzaileren lana betetzen duen aldagaia `Byte` datu-motatakoa da:

```
PROGRAM NotakPantailaratzen ;                  { \TP70\05\CASE1.PAS }
VAR
       Nota : Byte ;
BEGIN
       Write ('Notari dagokion zenbaki osoa eman: ') ;
       ReadLn (Nota) ;

       CASE Nota OF
         0..2 : BEGIN
                     WriteLn ('Oso gaizki') ;
                END ;
         3, 4 : BEGIN
                     WriteLn ('Gaizki') ;
                END ;
         5 :  BEGIN
                     WriteLn ('Nahiko') ;
              END ;
         6 :  BEGIN
                     WriteLn ('Ondo ') ;
              END ;
         7..9 : BEGIN
                     WriteLn ('Oso ondo') ;
                END ;
         10 : BEGIN
                     WriteLn ('Bikain') ;
              END ;
         ELSE
              WriteLn ('Nota 0 eta 10 artean egongo da!') ;
       END ;

       WriteLn ('------------------------------') ;
END.
```

`Nota` teklatutik irakurri ondoren `CASE-OF` egitura egikarituko du programak, `Nota`-k gordetzen duen balioa konstanteekin konparatzen da. Adibidez `0` eta `2` artean badago (`0`, `1` edo `2` denean) `Oso gaizki` mezua agertu eta programaren azken sententzia (marratxoen idazketa) exekutatuko da. `Nota`-k gordetzen duen balioa `3` edo `4` bada `Gaizki` mezua agertu eta marratxoak idatziko ditu. `5` eta `6` balioterako lehen bezala bakoitzari dagokion mezua eta gero marratxoak idatziko lirateke. `Nota`-k gordetzen duen balioa `7` eta `9` bitartekoa bada (`7`, `8` edo `9` denean) mezua `Oso ondo` izango da. `Bikain` pantailaratzeko sarturiko zenbakia `10` izango da.

Beraz, `0` eta `10` arteko notari dagokion irteera pantailaratuko da, baina beste zenbakiren bat teklatutik irakurriz gero, ez litzateke konstanteekin parekatuko eta ondorioz `ELSE` kasua beteko litzateke laguntza-mezu hau bistaratuz: `Nota 0 eta 10 artean egongo da!`

Laburbilduz, hiru modutan adierazten dira `CASE-OF` egituraren kasuak:

1. Konstante isolatu baten bitartez (`Nahiko`, `Ondo` eta `Bikain` adibideak)
2. Balioak enumeratuz (`Gaizki` adibidea). Balio konstanteak elkarrekiko komaz banatu behar dira
3. Balioen azpieremu bat definituz (`Oso gaizki` eta `Oso ondo` adibideak). Behemuga eta goimuga diren balio konstanteen artean puntu bi jartzen dira

`CASE-OF` sententzia menuren bat duen programetan asko erabiltzen denez, hona hemen menu batez gidatzen den adibide-programa bat:

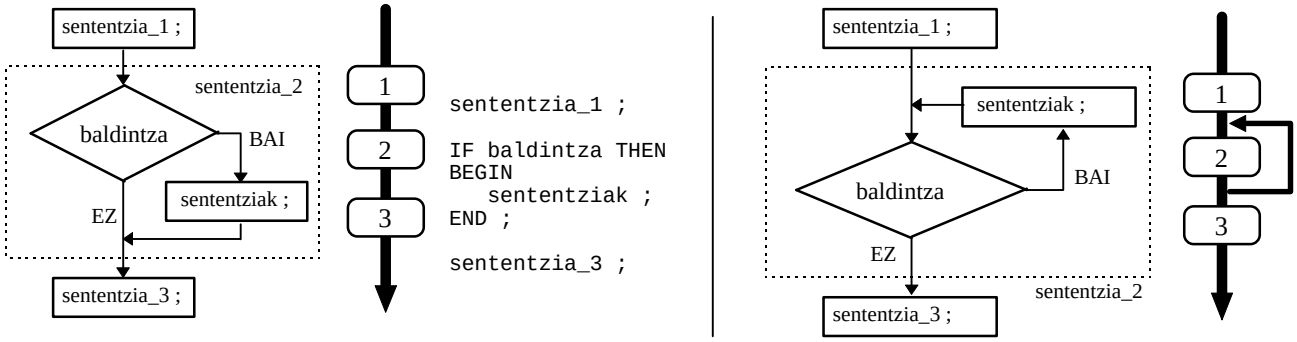
```
PROGRAM Aukerak_MenuBatezHautatzen ;                { \TP70\05\MENUA.PAS }
USES
    Crt ;
VAR
       Aukera, Itxoin : Char ;
       Erag1, Erag2 : Real ;
BEGIN
       ClrScr ;
       Write ('Lehenengo Eragigaiaren balioa eman: ') ;
       ReadLn (Erag1) ;
       Write ('Bigarren eragigaiaren balioa eman:  ') ;
       ReadLn (Erag2) ;

       WriteLn ('===============Menua===============') ;
       WriteLn ('    +     Batura lortzeko') ;
       WriteLn ('    -     Kendura lortzeko') ;
       WriteLn ('    *     Biderkadura lortzeko') ;
       WriteLn ('    /     Zatiketa burutzeko') ;
       WriteLn ('==================================') ;
       Write ('              Zure aukera: ') ;
       ReadLn (Aukera) ;

       CASE Aukera OF
         '+' : WriteLn (Erag1:0:2, ' + ', Erag1:0:2, ' = ', Erag1 + Erag2:0:2) ;
         '-' : WriteLn (Erag1:0:2, ' - ', Erag1:0:2, ' = ', Erag1 - Erag2:0:2) ;
         '*','x','X': WriteLn (Erag1:0:2, ' * ', Erag1:0:2, ' = ', Erag1 * Erag2:0:2) ;
         '/' : WriteLn (Erag1:0:2, ' / ', Erag1:0:2, ' = ', Erag1 / Erag2:0:2) ;
         ELSE
              WriteLn ('Eragiketa gaizki aukeratuta!') ;
       END ;

       Itxoin := ReadKey ;
END.
```

Hona hemen `Aukerak_MenuBatezHautatzen` programaren exekuzio-adibide bat:

```
Lehenengo Eragigaiaren balioa eman: 91.231
Bigarren Eragigaiaren balioa eman:  72.404
===============Menua==============
    +     Batura lortzeko
    -     Kendura lortzeko
    *     Biderkadura lortzeko
    /     Zatiketa burutzeko
==================================
              Zure aukera: +
91.231 + 72.404 = 163.64
_
```


5.3 AGINDU ERREPIKAKORRAK

Ikusi ditugun baldintzazko sententziekin agindu bat (edo agindu multzo bat) exekuta daiteke, erabakia balio jakin batzuen arabera hartzen delarik. Aginduak exekutatzea edo saihestea erabakitzen delarik fluxua aurrerantz doa, `IF-THEN`, `IF-THEN-ELSE` eta `CASE-OF` egituren logika adierazteko marraztu ditugun irudietan aginduak goitik behera exekutatzen dira:



Agindu errepikakorren zioa sententzia bat (edo sententzia multzo bat) errepikatu ahal izatea da. Hortaz, agindu errepikakor batean hiru ezaugarri ezagutuko dira:

1. Eskuineko irudiak adierazten duen sententzia errepikakorrean *bigizta* bat dago (aurrerantz linealki egin beharrean, atzerantz egiten du)
2. Errepikatu beharreko sententzia (edo sententzia multzoa) bigarren ezaugarria litzateke. Horri *iterazioa* esaten zaio
3. Azkenik, bigiztatik noiz irten behar den zehazten duen *baldintza* aldez aurretik definiturik egongo da. Baldintza horrek, uneoro, aditzera ematen du programak beste iterazio bat burutu behar duen, ala bestela bigiztatik irtetea dagokion

Turbo Pascal lengoaian agindu errepikakorrak hiru dira:

- `WHILE-DO`
- `REPEAT-UNTIL`
- `FOR-DO`

Hurrengo puntuetan banan-banan ikusiko ditugu.

5.3.1 WHILE-DO sententzia errepikakorra

Demagun konstante oso bati dagokion faktoriala kalkulatzen duen programa idatzi nahi dugula. Orain arte ikasi dugunarekin honako zerbait idatziko genuke:

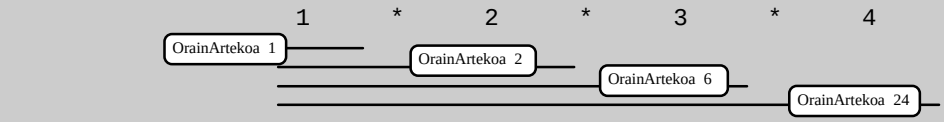
```
PROGRAM Faktoriala5BigiztarikGabe ;          { \TP70\05\WHILE0.PAS }
CONST
      ZBK = 5 ;
VAR
      emaitza : Integer ;
BEGIN
      emaitza := 1 * 2 * 3 * 4 * ZBK ;
      WriteLn (ZBK, '! = ', emaitza) ;
END.
```

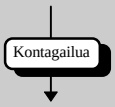
`Faktoriala5BigiztarikGabe` izeneko iturburu programa konpilatu ondoren exekutatzen badugu, beti aterako luke emaitza berdina. Izan ere, ez duenez sarrera aldatzeko posibilitaterik honako hau azalduko da monitorearen pantailan:

```
5! = 120
_
```

Konstantearen balioa `5` izan beharrean adibidez `4` balitz, orain ikusi izan dugun `Faktoriala5BigiztarikGabe` programa aldatu eta berri bat sortu beharko genuke. Baina, nola liteke `ZBK` konstante bat izan gabe teklatutik irakurtzen den aldagaia balitz?. Egia esan agindu errepikakorrik gabe ezin da egoera hori programatu, hona hemen `ReadLn(Zbk)` ondorengo algoritmoa:

Demagun teklatutik irakurtzen den zenbakia `4` dela: `Zbk ← 4` (lau errepikapen)





1. iterazioan lortzen dena: `1 ← OrainArtekoa * 1`
2. iterazioan lortzen dena: `2 ← OrainArtekoa * 2`
3. iterazioan lortzen dena: `6 ← OrainArtekoa * 3`
4. iterazioan lortzen dena: `24 ← OrainArtekoa * 4`

`4! = 24` lortzeko algoritmo horretan kontzeptu bi kontutan izanik errepikapenak burutzen dira. Batetik, zenbat iterazio egingo diren kontrolatzen duen kontagailu bat dago, aldakorra eta zenbaki oso bat denez `Integer` datu-motako aldagai baten bitartez ordezkatuko dugu (kontagailua den aldagai horren izenerako, nola ez, `Kontagailua` etiketa aukeratu dugu). Goian ikus daitekeenez, `Kontagailua` aldagaiak `1` balio izatetik `4` balio izatera igaroko da iterazio bakoitzean inkrementu bat jasanik.

Beste kontzeptua, emaitza partzialak[2] gordetzen duen metagailua litzateke, metagailua diogu iterazio bakoitzak erakartzen duena metatuz edo pilatuz doalako, metagailua zenbaki oso bat izango da baina gainezkadak ekidin nahiez `LongInt` datu-mota egokituko diogu; metagailua izendatzeko etiketari buruz, honako hau: iterazioa jakin batetik irtetean, metagailuak orain artean emaitzak zenbat balio duen adierazten duenez `OrainArtekoa` etiketa jarri diogu. Algoritmoan erreparatuz, iterazio jakin batean `OrainArtekoa` aldagaiari dagokion balioa honela kalkulatzen da: Aurreko Iterazioan Lortutakoa bider Uneko Iterazioaren Zenbakia.

Beraz, `Zbk` aldiz errepikatuko diren sententziak hauek lirateke:

```
OrainArtekoa := OrainArtekoa * Kontagailua ;        (1)
Kontagailua := Kontagailua + 1 ;                    (2)
```

`Zbk` iterazio ondoren `OrainArtekoa` etiketadun aldagaiak `Zbk`-ren faktoriala gordetzen duenez, bigizta bukatzen dela ziurtatu beharko da, eta ondoren emaitza den `OrainArtekoa` pantailan idatziko da.

Baina iterazio bakoitzean burutzen diren kalkuluak (1) eta (2) itxurakoak dira. Non aldagai bati dagokion balio berria lortzeko daukan balioaz eta beste konstante edo (aldagaiaz)

[2] Kontutan izan `Faktoriala5BigiztarikGabe` programan sarrerako datua konstantea zela eta emaitza esleipen bakar batean kalkula zitekeela, baina orain datua den zenbakiaren faktoriala zatika kalkulatuz lortuko dugu azken emaitza.

baliatzen den. Esleipenetan eskuinekoa ebaluatu ondoren ezkerreko aldagaian gordetzen da emaitza (gogoratu **4.2.6 Esleipena** puntua), adibidez, baldin: `OrainArtekoa ← 6`
`Kontagailua ← 4`

Orduan,

```
OrainArtekoa := OrainArtekoa * Kontagailua ;
      24     ←       6       *       4
```

Hau da, esleipena egiteko eskuineko elementuek balio ezagunak behar dituzte. Gure algoritmoan beharkizun hau ez da hasieran betetzen (ez baitakigu `WHILE` baino lehen `Kontagailua` eta `OrainArtekoa` aldagaiek zer gordetzen duten), ondorioz aldagai horien hasieraketak derrigorrezkoak dira. Hona hemen `FaktorialaWhileBigiztarekin` izeneko programaren kodifikazioa:

```
PROGRAM FaktorialaWhileBigiztarekin ;         { \TP70\05\WHILE1.PAS }
VAR
      Kontagailua, Zbk : Integer ;
      OrainArtekoa : LongInt ;
BEGIN
      Write ('Zenbaki osoa eta positiboa eman: ') ;
      ReadLn (Zbk) ;

      OrainArtekoa := 1 ;       (* lehenengo hasieraketa *)
      Kontagailua := 1 ;        (* bigarren hasieraketa *)

      WHILE Kontagailua <= Zbk DO
         BEGIN
            OrainArtekoa := OrainArtekoa * Kontagailua ;
            Kontagailua := Kontagailua + 1 ;
         END ;

      WriteLn (Zbk, '! = ', OrainArtekoa) ;
END.
```

`Zbk` zenbakirako `4` datua emanez gero, programaren aldagaiek hartzen dituzten balioak taula honetan biltzen dira. Zeinek egitura errepikakor baten funtzionamendua erakusten duen:

Kontagailua	Kontagailua + 1
1	1 + 1
2	2 + 1
3	3 + 1
4	4 + 1
5	

OrainArtekoa	OrainArtekoa * Kontagailua
1	
1	1 * 1
2	1 * 2
6	2 * 3
24	6 * 4

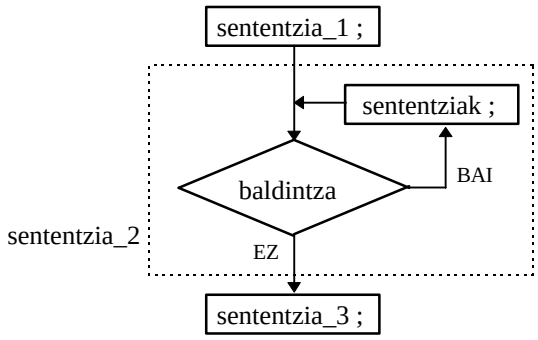
lehenengo hasieraketa
bigarren hasieraketa
1. iteraziotik irtetean
2. iteraziotik irtetean
3. iteraziotik irtetean
4. iteraziotik irtetean

Hona hemen `FaktorialaWhileBigiztarekin` programaren irteera bat, ikus daitekeenez emaitza `Faktoriala5BigiztarikGabe` programaren bezalakoa da, baina kasu honetan datua aukeragarria da:

```
Zenbaki osoa eta positiboa eman: 5
5! = 120
_
```

`WHILE-DO` agindu errepikakorrak, baldintzazkoek bezala, galdera bat integraturik du eta ondoren errepikatu behar den sententzia multzo bat. Galderaren erantzuna egia izatean (`TRUE` Turbo Pascalean) multzoaren sententziak egikaritu egingo dira, multzoaren azken sententzia exekutatu eta gero galdera berriro ebaluatuko da bigizta bat sortuz, eta egia izatean berriro egikarituko dira `WHILE-DO` aginduari loturiko sententziak. Adierazpen

boolearraren erantzuna gezurra izango denean (FALSE Turbo Pascalean) multzoaren barneko sententziak ez dira gehiagotan exekutatzen eta besterik gabe WHILE-DO agindua amaitutzat jo daiteke, txanda bere jarraian datorren sententziak hartuko luke. Grafikoki:




```
sententzia_1 ;

WHILE baldintza DO
  BEGIN
    sententziak ;
  END ;

sententzia_3 ;
```

5.3.1.1 Adibidea

Demagun ondoko segidaren balioa lortu nahi dugula:

$$1/1^2 - 1/2^2 + 1/3^2 - 1/4^2 + 1/5^2 - 1/6^2 + \ldots$$

Ikus daitekeenez, batetik batugaiak gero eta txikiagoak dira balio absolutuaz, eta bestetik zeinua aldatuz doa. Segidaren balioa batugaien batura algebraikoa dela kontsideraturik, arazoa determinaturik geratuko da batugaien kopurua ezagutzean, baina demagun datua zeharka ematen dela, demagun tekladutik sartzen den datua prezisioa dela. Adibidez segidaren kalkuluak milakoen[3] doitasuna izango duela.

Algoritmoa:

1. Doitasuna tekladutik irakurri
2. Hasieraketak. Lau izango dira, alde batetik emaitza partzialak gordeko dituen Emaitza aldagaiak behar duena, eta bestetik bigiztan sartzea bermatzen duen uneko batugaiaren balioa eta berari dagokion zeinua, eta laugarrenik batugaiak lortzeko erabili behar den Kontagailua kontagailua
3. Bigiztaren iterazio bakoitzeko: uneko batugaiaren balioa kalkulatu eta emaitzari metatu, hurrengo bigiztarako prestakizunak burutu (kontagailua inkrementatu, zeinua aldatu). Azken batugaia (balio absolutuan) doitasuna baino txikiago izan bada bigiztatik irten
4. Metagailuak gordetzen duena pantailaratu

```
PROGRAM SegidaWhileBigiztarekin1 ;           { \TP70\05\WHILE2_1.PAS }
VAR
      Kontagailua, Zeinua : Integer ;
      Batugaia, Emaitza, Doitasuna : Real ;
BEGIN
      Write ('Doiatsuna eman: ') ;
      ReadLn (Doitasuna) ;

      Emaitza := 0 ;           (* lehenengo hasieraketa  *)
      Batugaia := 1 ;          (* bigarren hasieraketa   *)
      Zeinua := 1 ;            (* hirugarren hasieraketa *)
      Kontagailua := 1 ;       (* laugarren hasieraketa  *)
```

[3] Horren arabera segidari metatu egingo zaion azken batugaiaren balio absolutua 0.001 baino txikiagoa izango da, eta horrez gero bigizta etengo da.

```
      WHILE Abs(Batugaia) > Doitasuna DO
         BEGIN
             Batugaia := Zeinua * (1 / Sqr(Kontagailua)) ;
             Emaitza := Emaitza + Batugaia ;


             WriteLn (Kontagailua, '. batugaia = ', Batugaia:8:5,
                      'segida = ':15, Emaitza:0:5) ;

             Kontagailua := Kontagailua + 1 ;
             Zeinua := Zeinua * (-1) ;
         END ;

      WriteLn ('Segida = ', Emaitza:0:5) ;
END.
```

Hauxe litzateke `SegidaWhileBigiztarekin1` programaren irteera bat:

```
Doitasuna eman: 0.05
1. batugaia =  1.00000      segida = 1.00000
2. batugaia = -0.25000      segida = 0.75000
3. batugaia =  0.11111      segida = 0.86111
4. batugaia = -0.06250      segida = 0.79861
5. batugaia =  0.04000      segida = 0.83861
Segida = 0.83861
_
```

Irteera azter dezagun, lehenengo lerroa sarrerako datuari dagokio eta azkenekoa bigiztatik irten eta gero emaitzaren pantailaraketa litzateke; zenbakiz hasten diren bost mezuak `Wile` egitura barruan lortu dira (iterazio bakoitzeko bana).

`SegidaWhileBigiztarekin1` programa honetan lau hasieraketa egin dira. Eta iterazio bakoitzeko batugai berria kalkulatu eta metagailuari (`Emaitza`-ri) pilatzen zaio, horrekin batera hurrengo iterazioa prestatzeko asmoarekin zeinua kontrolatzen duen aldagaia berrizten da eta batugaiaren balioa emango duen kontagailua ere inkrementatzen da.

Ariteka bezala saia gaitezen ondoko aldaketak egiten, eta eraginak azter ditzagun:

Aldaketa	**Eragina**
Agindu hau kendu[4] `Emaitza := 0 ;`	Nahiz eta aldaketa hau akats bat izan, Turbo Pascal 7.0 bertsioan emaitza berdina izango da. Hori horrela da 7.0 bertsioan programa nagusiaren zenbakizko aldagai guztiak berez, eta automatikoki, `0` balioz hasieratzen direlako.
Agindu hau kendu `Batugaia := 1 ;`	Hasieraketa hau ez bada jartzen, aurrekoa gogoratuz, `Batugaia` aldagaiak Turbo Pascal 7.0 bertsioan `0` balioko du, ondorioz fluxua bigizta barrura ez da sartuko. Izan ere, irakurri den `Doitasuna` positiboa denez adierazpen boolear `Abs(0) > Doitasuna` beti `FALSE` izango da. Beraz, hasieraketa hori kentzean segidaren balioa `Emaitza`-ren hasierakoa izango da (`0` izango da).

[4] Aginduak kentzeko aski da iruzkin bezala jartzea. Esate baterako: `(* Emaitza := 0 ; *)`

Agindu hau kendu `Zeinua := 1 ;`	Aurrekoa bezala, hasieran `Zeinua` aldagaiak (Turbo Pascal 7.0 bertsioan) `0` balioko du. Hasieraketa dela eta `Batugaia`-k `1` balio duenez, fluxua bigizta barrura sartuko dela bermatuta dago. Baina `WHILE` barnean, `Batugaia`-k hartzen duen balioa bider `0` egiten denez bigarren iteraziorik ez da izango. Beraz, hasieraketa hori kentzean segidaren balioa `0` izango da, baina oraingoan iterazio bat burutu egin da.
`Kontagailua := 1 ;` agindu hau kendu	Aurrekoak bezala, `Kontagailua` aldagaiak (Turbo Pascal 7.0 bertsioan) hasieran `0` balioko du. Beraz bigizta barruko lehenengo aginduak hau eragiten du: `Error 200: Division by zero.` Ondorioz, hasieraketa hori kentzean programa ezingo da egikaritu.
Bigizta barruko agindu hau kendu `Kontagailua := Kontagailua + 1 ;`	`WHILE` bigiztan sartu ondoren, `Batugaia`-rako kalkulatzen den lehenengo balioa `1` da, eta datua den `Doitasuna`-ren balioa adibidez `0.05` baldin bada, bigarren iterazioa egitera sartuko da `Abs(1) > Doitasuna` adierazpen boolearra egia bait da. Bigarren iterazioan `Batugaia`-k hartzen duen balioa `-1` da, eta berriro ere `Abs(-1) > Doitasuna` baldintza egia denez beste iterazio bat egingo da. Ondorioz, kontagailua inkrementatzen ez bada eta ohi denez `Doitasuna` zero eta bat artean aukeratuz, programa `WHILE` bigiztan kateaturik[5] geratzen da.
Bigizta barruko `WriteLn` agindua kendu	Programa ondo lebilke baina `WHILE` iterazio bakoitzean kalkulatzen diren balio horien informazioa ez litzateke pantailaratuko.
Bigizta barruko agindu hau kendu `Zeinua := Zeinua * (-1) ;`	Batugaien batuketa eta kenketak txandaka egin ordez, batuketak soilik egingo lirateke eta lortu litzatekeen emaitzaren balioa proposatu den segidarekin ez litzateke bat etorriko. Adibidez, `Doitasuna`-ren irakurketan `0.05` emanik, lortzen den segidaren emaitza `1.46361` da.

Argi dagoenez `Zeinua` eta `Kontagailua` arteko aldagaien artean erlazioa dago, horregatik euretariko bat kenduz `SegidaWhileBigiztarekin1` programaren bigarren bertsio bat egingo dugu, bigarren bertsio honetan zehazki derrigorrezkoak diren aginduak idatziko ditugu.

Hona hemen, `SegidaWhileBigiztarekin2` izeneko programari dagokion kodifikazioa. Gauza bera egiten duenez, pantailatik irtengo dena, `WHILE` barneko idazketak izan ezik, lehengokoa izango litzateke:

[5] Bigizta infinito batetik irteteko `CONTROL-Pause` teklak sakatu. Agindu errepikakorrekin ari garenean, programa bat exekutatu aurretik diskoan gorde, horretarako `ALT-File-Save` edo `F2` teklak sakatu.

```
PROGRAM SegidaWhileBigiztarekin2 ;            { \TP70\05\WHILE2_2.PAS }
VAR
      Kontagailua : Integer ;
      Batugaia, Emaitza, Doitasuna : Real ;
BEGIN
      Write (‘Doiatsuna eman: ‘) ;
      ReadLn (Doitasuna) ;

      Emaitza := 0 ;            (* lehenengo hasieraketa *)
      Batugaia := 1 ;           (* bigarren hasieraketa *)
      Kontagailua := 1 ;        (* hirugarren hasieraketa *)

      WHILE Abs(Batugaia) > Doitasuna DO
         BEGIN
             Batugaia := Zeinua * (1 / Sqr(Kontagailua)) ;
             IF Kontagailua MOD 2 = 0 THEN
                    Batugaia := (-1) * Batugaia ;
             Emaitza := Emaitza + Batugaia ;
             Kontagailua := Kontagailua + 1 ;
         END ;

      WriteLn (‘Segida = ‘, Emaitza:0:5) ;
END.
```

5.3.1.2 Adibidea

Zenbaki osoekin lan eginez, prozesu errepikakor batean zenbakiak irakurriko dira. Programak, irakurritako zenbaki positiboen eta negatiboen kopuruak kalkulatuko ditu. Errrepikapenak eteteko, `0` zenbakiaren bitartez egingo da.

```
PROGRAM PosNegWhileBigiztarekin1 ;              { \TP70\05\WHILE3_1.PAS }
CONST
      AMAITU = 0 ;
VAR
      PositiboKont, NegatiboKont : Integer ;
      Zenbakia : LongInt ;
BEGIN
      PositiboKont := 0 ;             (* positiboen kopurua hasieran *)
      NegatiboKont := 0 ;             (* negatiboen kopurua hasieran *)

      Write ('Zenbaki bat eman (amaitzeko ', AMAITU, ' sakatu): ') ;
      ReadLn (Zenbakia) ;

      WHILE Zenbakia <> AMAITU DO
         BEGIN
             IF Zenbakia > 0 THEN
                    PositiboKont := PositiboKont + 1
             ELSE
                    NegatiboKont := NegatiboKont + 1 ;

             Write ('Zenbaki bat eman (amaitzeko ', AMAITU, ' sartu): ') ;
             ReadLn (Zenbakia) ;
         END ;

      WriteLn ('Positiboen kopurua = ', PositiboKont) ;
      WriteLn ('Negatiboen kopurua = ', NegatiboKont) ;
END.
```

`PosNegWhileBigiztarekin1` izeneko programan bigiztara iritsi aurretik datuaren irakurketa bat egiten da. Horrela, `WHILE` egiturak duen baldintza balio ezagunekin (`Zenbakia` aldagaian irakurritakoa eta `AMAITU` konstantea) ebaluatu ahal izango da. Hau da horri dagokion irteera bat:

```
Zenbaki bat eman (amaitzeko 0 sakatu): 18
Zenbaki bat eman (amaitzeko 0 sartu): 5
Zenbaki bat eman (amaitzeko 0 sartu): -11
Zenbaki bat eman (amaitzeko 0 sartu): 0
Positiboen kopurua = 2
Negatiboen kopurua = 1
_
```

Baina demagun `WHILE` kanpoan eginiko irakurketa eta pantailaraketa kendu nahi ditugula, `PosNegWhileBigiztarekin1` beste modu batean programatuko dugu, mezua eta datuen sarrera behin bakarrik idatziz. Programa honi `PosNegWhileBigiztarekin2` esango diogu eta duen berezitasunik nabarmenena, `WHILE`-ren baldintza balio ezagunekin ebaluatzeko derrigorrezkoa zaigun hasieraketa litzateke.

`WHILE`-an sarrera ziurtatzeko `Zenbakia` aldagaiari zero ez den balioren bat esleitu behar zaio, izan ere `AMAITU` konstantea `0` da. Beste hasieraketa biak (`PositiboKont` eta `NegatiboKont` aldagaiena) soberan egon daitezkeela pentsa daiteke, Turbo Pascal 7.0 bertsioak zenbakizko aldagai orokorrak zeroz hasieratzen dituelako, baina edozein konpiladorerako suposaketa hau egitea onartezina denez, hasieraketok ezinbestekotzat joko ditugu ere.

Hona hemen `PosNegWhileBigiztarekin2` programa, non derrigorrezkoak zaizkigun hiru hasieraketak agertzen diren:

```
PROGRAM PosNegWhileBigiztarekin2 ;                { \TP70\05\WHILE3_2.PAS }
CONST
      AMAITU = 0 ;
VAR
      PositiboKont, NegatiboKont : Integer ;
      Zenbakia : LongInt ;
BEGIN
      PositiboKont := 0 ;             (* positiboen kopurua hasieran *)
      NegatiboKont := 0 ;             (* negatiboen kopurua hasieran *)

      Zenbakia := 7 ;                 (* AMAITU ez den edozein balio *)

      WHILE Zenbakia <> AMAITU DO     (* sartzea ziurtaturik dago *)
         BEGIN
            Write ('Zenbaki bat eman (amaitzeko ', AMAITU, ' sartu): ') ;
            ReadLn (Zenbakia) ;

            IF Zenbakia > 0 THEN
                  PositiboKont := PositiboKont + 1 ;

            IF Zenbakia < 0 THEN
                  NegatiboKont := NegatiboKont + 1 ;
         END ;

      WriteLn ('Positiboen kopurua = ', PositiboKont) ;
      WriteLn ('Negatiboen kopurua = ', NegatiboKont) ;
END.
```

`PosNegWhileBigiztarekin2` programaren `WHILE` barruan aldaketa bi egin dira: `IF-THEN-ELSE` egitura `IF-THEN` bikote batez ordezkatu izan da, eta datuaren irakurketa bigiztaren hasieran dago. Zer dela eta aldaketa horiek?

Bigiztak datuak irakurri eta prozesatzeko erabiltzen dira askotan, baina bada zailtasun bat gainditu behar izaten dena, bigiztatik noiz irtetea zehaztearena. Zenbaitetan erabiltzen den teknika *zelatari*arena izaten da. *Zelataria* datua balitz bezala irakurtzen den balioa izango da, baina kontuz zelatari-balioa aukeratzekoan (zelataria ez da prozesatzen, zelataria bigizta eteteko jartzen da). Esate baterako, adibide honetan zero balioak zelatari izateko ahalmena du, baina zenbaki positiboak eta negatiboak zenbatu ordez, bakoitien eta bikoitien kopuruak

lortu nahi baditugu zelataria besteren bat izan beharko luke, edo bestela emaitza pantailaratzean `0` bikoitia dela kontutan izan.

Aurrekoerekin loturik `PosNegWhileBigiztarekin1` programa errepikatuko dugu, baina oraingoan zelataria zero izan beharrean karaktere bat izango da. Programa berri honek hartuko duen izena `PosNegWhileBigiztarekin3` izango da:

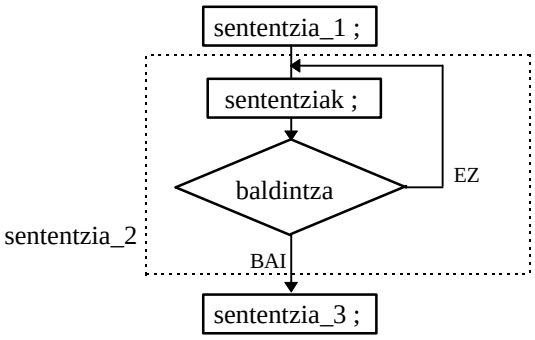
```
PROGRAM PosNegWhileBigiztarekin3 ;                { \TP70\05\WHILE3_3.PAS }
USES
       Crt ;
VAR
       PositiboKont, NegatiboKont : Integer ;
       Zenbakia : LongInt ;
       Erantzuna : Char ;              (* zelataria *)
BEGIN
       PositiboKont := 0 ;             (* positiboen kopurua hasieran *)
       NegatiboKont := 0 ;             (* negatiboen kopurua hasieran *)

       Write ('Zenbakirik sartu nahi duzu? (B/E) ') ;
       Erantzuna := ReadKey ;
       WriteLn (Erantzuna) ;
       Erantzuna := UpCase (Erantzuna) ;

       WHILE Erantzuna = 'B' DO
          BEGIN
              Write ('Zenbaki bat eman: ') ;
              ReadLn (Zenbakia) ;

              IF Zenbakia >= 0 THEN            (* zero positibotzat hartuz *)
                     PositiboKont := PositiboKont + 1
              ELSE
                     NegatiboKont := NegatiboKont + 1 ;

              Write ('Zenbaki gehiagorik? (B/E) ') ;
              Erantzuna := ReadKey ;
              WriteLn (Erantzuna) ;
              Erantzuna := UpCase (Erantzuna) ;
          END ;

       WriteLn ('Positiboen kopurua = ', PositiboKont) ;
       WriteLn ('Negatiboen kopurua = ', NegatiboKont) ;
END.
```

Zelataria berezia darabilen `PosNegWhileBigiztarekin3` programak izan dezakeen irteera bat hauxe litzateke:

```
Zenbakirik sartu nahi duzu? (B/E) B
Zenbaki bat eman: 0
Zenbaki gehiagorik? (B/E) B
Zenbaki bat eman: -5
Zenbaki gehiagorik? (B/E) E
Positiboen kopurua = 1
Negatiboen kopurua = 1
_
```

5.3.2 REPEAT-UNTIL sententzia errepikakorra

5.3.1.2 Adibidea puntuan ikusi diren `PosNegWhileBigiztarekin1,2,3` programetan `WHILE` aginduaren barrura sartzea bermatu behar izan da, beharkizun hori betetzeko inoiz era bortxatuan egin dugu (`Zenbakia:=7;` hasieraketa gogoratu). Halako ardurarik ez izateko `REPEAT-UNTIL` agindu errepikakorra oso aproposa da. Ikus dezagun `REPEAT-UNTIL` aginduari dagokion eskema:




```
sententzia_1 ;

REPEAT
    sententziak ;
UNTIL baldintza ;

sententzia_3 ;
```

`REPEAT-UNTIL` agindu errepikakorrari, `WHILE-DO` aginduari bezala, errepikatu behar den sententzia multzo bat dagokio eta horren ondoren galdera bat integraturik dauka. Galderaren erantzuna gezurra izatean (`FALSE` Turbo Pascalean) multzoaren sententziak berriro egikarituko dira bigizta bat sortuz. Adierazpen boolearraren erantzuna egia izango denean (`TRUE` Turbo Pascalean) multzoaren barneko sententziak ez dira gehiagotan exekutatzen eta besterik gabe `REPEAT-UNTIL` agindua amaitutzat jo daiteke, txanda bere jarraian datorren sententziak hartuko luke.

Ekar dezagun gogora `IF-THEN` erabiliz idatzi izan dugun lehenengo programa, 5-4 orrialdeko `ErroreakAurreikusten` deiturikoa. Orduan, datu bat irakurri ondoren bere balioaren arabera erabakiak egiten ikasi genuen, adibide-programa horretan `zatitzailea` izeneko balioa teklatuz eman ondoren zatiketa burutzea ala ez erabakitzen zen. `ErroreakAurreikusten` alda dezagun aldagai kritikoari iragazki jarriz, programa berriari `ErroreakSaihesten` esango diogu:

```
PROGRAM ErroreakSaihesten ;                    { \TP70\05\REPEAT1.PAS }
VAR
       zatikizuna, zatitzailea, x : Integer ;
BEGIN
    Write ('Zatikizunaren balioa eman: ') ;                (* sententzia_1 *)
    ReadLn (zatikizuna) ;                                  (* sententzia_2 *)

    REPEAT                                                 (* sententzia_3 *)
       Write ('Zatitzailaren balioa eman: ') ;
       ReadLn (zatitzailea) ;
    UNTIL zatitzailea <> 0 ;

    x := zatikizuna DIV zatitzailea ;                      (* sententzia_4 *)
    WriteLn (zatikizuna, ' zati ', zatitzailea, ' = ', x) ;
    WriteLn ('------------------------------') ;           (* sententzia_6 *)
END.
```

`REPEAT` gakoak ez duenez inolako oztoporik jartzen bigiztaren aurreneko errepikapena ziurtatutik dago, `zatitzailea` aldagaiak hartzen duen balioaren arabera 3. sententzia ostera egikaritu (beste zatitzale berri bat eskatuz) ala 4. sententziarekin jarraitzea erabakiko da. `WHILE-DO` aginduak behar izaten dituen hasieraketak `REPEAT-UNTIL` sententziak ez ditu behar, honetaz baliatuz **5.3.1.2 Adibidea** puntuko programa berridatz dezagun:

5.3.2.1 Adibidea

Zenbaki osoekin lan eginez, prozesu errepikakor batean zenbakiak irakurriko dira. Programak, irakurritako zenbaki positiboen eta negatiboen kopuruak kalkulatuko ditu. Errrepikapenak eteteko, `0` zenbakiaren bitartez egingo da.

```
PROGRAM PosNegRepeatBigiztarekin ;                  { \TP70\05\REPEAT2.PAS }
CONST
      AMAITU = 0 ;
VAR
      PositiboKont, NegatiboKont : Integer ;
      Zenbakia : LongInt ;
BEGIN
      PositiboKont := 0 ;             (* positiboen kopurua hasieran *)
      NegatiboKont := 0 ;             (* negatiboen kopurua hasieran *)

      REPEAT                          (* sartzea ziurtaturik dago *)
         Write ('Zenbaki bat eman (amaitzeko ', AMAITU, ' sartu): ') ;
         ReadLn (Zenbakia) ;
         IF Zenbakia > 0 THEN
                   PositiboKont := PositiboKont + 1 ;
         IF Zenbakia < 0 THEN
                   NegatiboKont := NegatiboKont + 1 ;
      UNTIL Zenbakia = AMAITU ;

      WriteLn ('Positiboen kopurua = ', PositiboKont) ;
      WriteLn ('Negatiboen kopurua = ', NegatiboKont) ;
END.
```

`PosNegRepeatBigiztarekin` programa lehenagoko `PosNegWhileBigiztarekin2` programarekin alderatzen badugu, aldaketak bi direla konturatzen gara. Batetik, eta garrantzitsuena, `Zenbakia:=7;` bezalako hasieraketarik ez dela behar. Bigarrenik, bigiztatik irteteko galdera ezberdina dela, izan ere `WHILE` aginduak duen logika eta `REPEAT` aginduak duena kontrakoak[6] dira.

5.3.2.2 Adibidea

Zenbaki baten erro karratua Newtoren bitartez honela kalkulatzen da:

$$Erro_i = \frac{Zenbakia / Erro_{i-1} + Erro_{i-1}}{2}$$

non,

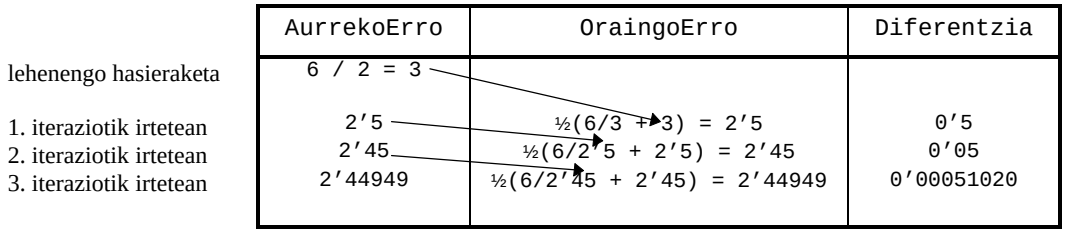
Zenbakia sarrerako zenbakia den
$Erro_i$ i-garren iterazioko erroa, zein aurreko iterazioaren arabera lortzen den
$Erro_o$ edozein konstante izan daiteke adibidez zenbakiaren erdia

Programan *Zenbakia* eta soluziorako nahi den *Epsilon* prezisioa irakurtzen dira, eta bigizta batean $Erro_1$ $Erro_2$ $Erro_3$ $Erro_4$ ... $Erro_{n-1}$ $Erro_n$ kalkulatuko dira. Bigiztatik irteteko baldintza bi erro jarraien arteko diferentzia irakurritako *Epsilon* prezisioa baino txikiagoa izatea izango da:

$$| Erro_{n-1} - Erro_n | < Epsilon$$

Algoritmoa ondoko taulan biltzen da, suposatuz `Zenbakia` aldagaian `6` datua gorde dela, eta prezisioa milakoaren mailakoa izango dela (`Epsilon` aldagaian `0.001` balioa gorde dela). Iterazio bakoitzeko erroen arteko diferentzia aurkitzen da, horretarako aurreko iterazioko erroaren balioa eta oraingoan kalkulatutakoa erabiltzen dira. Diferentzia `Epsilon` baino handiagoa den bitartean prozesu errepikakorra martxan ariko da:

[6] Bakoitzari dagokion eskeman ikus daitekeenez, `WHILE-DO` batean errepikapena burutu dadin adierazpen boolearrak egia balio behar du, `REPEAT-UNTIL` aginduan berriz errepikapena gauzatzeko adierazpen boolearra gezurra izango da.



Hona hemen `ErroKarratuaNewtonFormulaz` programaren kodea. Ikus daitekeenez soluzioa `OraingoErroa` aldagaian lortzen da. Baina, zertarako da `Laguntzaile` izeneko aldagaia?.

```
PROGRAM ErroKarratuaNewtonFormulaz ;                    { \TP70\05\NEWTON.PAS }
VAR
      Zenbakia, Epsilon, AurrekoErro, OraingoErro, Laguntzaile : Real ;
BEGIN
      Write ('Zenbaki bat eman: ') ;
      ReadLn (Zenbakia) ;
      Write ('Kalkuluaren prezisioa zehaztu: ') ;
      ReadLn (Epsilon) ;

      AurrekoErro := Zenbakia / 2 ;          (* beste balio bat emanik ere dabil *)

      REPEAT                                 (* sartzea ziurtaturik dago *)
        Laguntzaile := AurrekoErro ;
        OraingoErro := ((Zenbakia / AurrekoErro) + AurrekoErro) / 2 ;
        AurrekoErro := OraingoErro ;

        Write ('Aurrekoa: ', Laguntzaile:0:8, '     Oraingoa: ', OraingoErro:0:8) ;
        WriteLn ('     Diferentzia: ', Abs (OraingoErro - Laguntzaile):0:8) ;
      UNTIL Abs (OraingoErro - Laguntzaile) < Epsilon ;

      WriteLn (Zenbakia:0:3, '-ren Newton erroa -------> ', OraingoErro:0:8) ;
      WriteLn (Zenbakia:0:3, '-ren benetako erroa -----> ', Sqrt (Zenbakia):0:8) ;
END.
```

Programa hau `WHILE-DO` kontrol-egituraz kodetuz gero `ErroKarratuaNewtonWhile` izeneko programa lor daiteke, eta ez da `Laguntzaile` delako aldagairik. Aipatutako datuekin `ErroKarratuaNewtonFormulaz` programaren irteera:

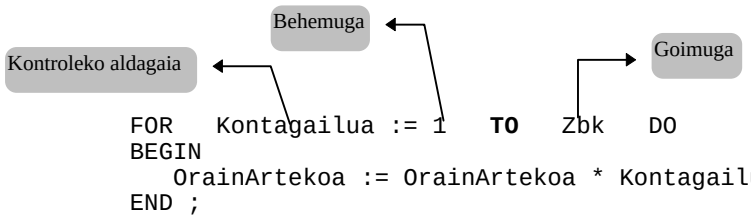
```
Zenbaki bat eman: 6
Kalkuluaren prezisioa zehaztu: 0.001
Aurrekoa: 3.00000000      Oraingoa: 2.50000000      Diferentzia: 0.50000000
Aurrekoa: 2.50000000      Oraingoa: 2.45000000      Diferentzia: 0.05000000
Aurrekoa: 2.45000000      Oraingoa: 2.44948980      Diferentzia: 0.00051020
6.000-ren Newton erroa --------> 2.44948980
6.000-ren benetako erroa ------> 2.44948974
_
```

5.3.3 FOR-DO sententzia errepikakorra

Agindu hau, sententzia bat edo sententzien multzo bat zenbait aldiz errepikatu behar denean erabiltzen da. `FOR-DO` aginduan errepikapen kopurua aldez aurretik ezaguna da, eta `FOR-DO` agindua exekutatzen den unean errepikapenak zenbat izango diren ezaguna da. Adibiderik argiena faktorial baten kalkulua litzateke, zenbaki oso bat emanik bere faktoriala lortzeko behar diren biderkaketa partzialak definiturik daude (gogoratu **5.3.1** puntuko `FaktorialaWhileBigiztarekin` izeneko programa). Hauxe da eginkizun bera betetzen duen programa `FOR-DO` sententzia erabiliz:

```
PROGRAM FaktorialaForBigiztarekin ;          { \TP70\05\FOR1.PAS }
VAR
      Kontagailua, Zbk : Integer ;
      OrainArtekoa : LongInt ;
BEGIN
      REPEAT
         Write ('Zenbaki osoa eta positiboa eman: ') ;
         ReadLn (Zbk) ;
      UNTIL Zbk >= 0 ;

      OrainArtekoa := 1 ;       (* metagailuaren hasieraketa *)

      FOR Kontagailua:=1 TO Zbk DO
         BEGIN
            OrainArtekoa := OrainArtekoa * Kontagailua ;
         END ;

      WriteLn (Zbk, '! = ', OrainArtekoa) ;
END.
```

Programaren gakoa `FOR-DO` aginduan dagoenez, isolaturik eta atalka azter dezagun. Hau da `FaktorialaForBigiztarekin` adibide-programatik ateratako `FOR-DO` agindua:



Hasteko, esan beharra dago `FOR-DO` sententziak kontroleko aldagai bat darabilela, adibidean `Kontagailua` deitzen dena. `FOR` eta `DO` hitz erreserbatuaz gain, `TO` hitza agertzen da ere, azken honek iterazio bakoitzeko kontroleko aldagaiaren balioa inkrementatu egingo dela adierazten du. Beraz, iterazioen kopurua `Kontagailua`-k balio `1` duenetik `Zbk` balio izan arte lortzen da.

Nabaria denez agindu errepikakor hau planteatzerakoan, kontroleko aldagaiak egingo duen tartea definiturik egon behar du, hots behemuga eta goimuga ezagunak izango direla. Goimugaren balioa behemugarena baino txikiagoa balitz, ez litzateke errorerik suertatuko iteraziorik burutu gabe amaituko litzateke `FOR-DO` sententzia.

Kontroleko aldagaiaren datu-mota ordinala[7] izango da: zenbaki osoa (Integer, Byte, LongInt, ShortInt eta Word), Char, Boolean, enumeratua edo azpieremua. Goimuga eta behemuga zehazten dituzten aldagai, konstante edo adierazpenak kontroleko aldagaiarekin konpatibleak izango dira.

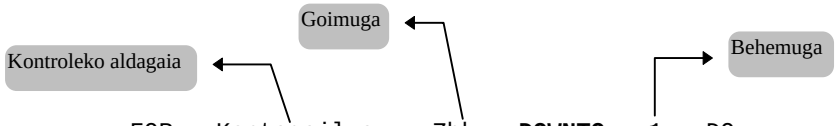
Aipatzekoa da ere, kontroleko aldagaia automatikoki inkrementatzen dela, eta ondorioz ez litzatekeela onargarria izango honelakorik `FOR-DO` sententziaren barruan egitea:

```
Kontagailua := Kontagailua + 1 ;
```

`WHILE` eta `REPEAT` beste bi agindu errepikakorrekin alderatuz gero, `FOR-DO` sententziak berezitasunak ditu, batetik hasieraketa integraturik duela eta bestetik kontroleko aldagaiaren inkrementuak automatikoak direla. Desabantailak, berezitasun horien kontrako aurpegia lirateke, hots hasieraketak integraturik ditu baina aldez aurretik iterazioen kopurua ezagutzea ezinbestekoa da, eta inkrementuak automatikoki egiten dira baina beti banan banan kontroleko aldagaia handituz. Ahal izanez gero `FOR` batez programa daitekeena ez da `WHILE` edo `REPEAT` bezala planteatuko.

[7] Onargarria ez den oinarrizko datu-mota bakarra `Real` zenbaki errealena da.

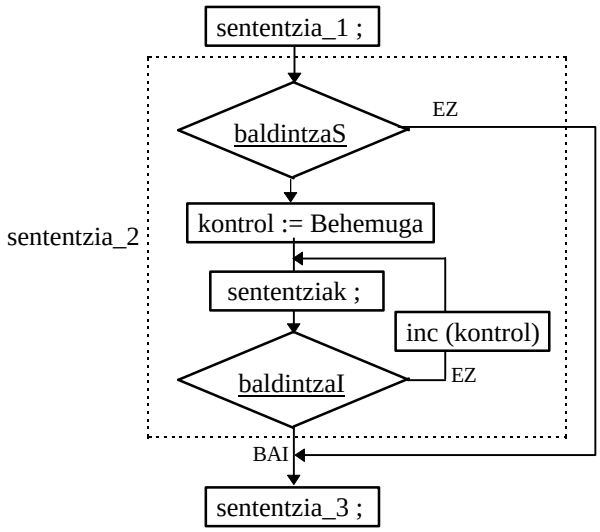
`FOR-DO` aginduak `DOWNTO` hitz erreserbatua onartzen du ere, kasu horretan faktoriala kalkulatzen duen `FaktorialaForBigiztarekin` adibide-programak itxura hau izango luke:




```
FOR   Kontagailua := Zbk   DOWNTO   1   DO
BEGIN
   OrainArtekoa := OrainArtekoa * Kontagailua ;
END ;
```

Lehen bezala kontroleko aldagaia `Kontagailua` izanik, `DOWNTO` hitza agertzen delako, iterazio bakoitzeko kontroleko aldagaiaren balioa unitate batean dekrementatu egingo da.

Hona hemen `FOR-TO-DO` aginduari dagokion eskema:




```
sententzia_1 ;

FOR kontrol:=Behemuga TO Goimuga DO
BEGIN
    sententziak ;
END ;

sententzia_3 ;
```

baldintzaS → `Behemuga <= Goimuga`

baldintzaI → `Kontrol = Goimuga`

`FOR-TO-DO` eskemaren aurrean ondoko baieztapenak egin daitezke:

1. `FOR-TO-DO` aginduak sarrerako baldintza bat du, horretan `WHILE-DO` baten antza dauka. Behemuga goimuga baino handiagoa denean ez da bigizta exekutatuko eta txanda hurrengo sententziari pasatzen zaio
2. Iterazio baten ostean bigarren konprobazio bat egiten da. Horren arabera kontroleko aldagaiak goimugaren balioa hartu izan badu, bigiztatik irten eta hurrengo sententzia exekutatuko da. Baina kontroleko aldagaiaren balioa goimuga baino txikiagoa bada, kontroleko aldagaia inkrementatu eta beste iterazio bat hasiko da
3. Kontroleko aldagaia inkrementatzean unitate batean handitzen da
4. Goimuga behemuga baino handiagoa denean, iterazioak exekutatuko ondoren kontroleko aldagaiak `Goimuga` balioko du. Eta iterazioen kopurua `Goimuga-Behemuga+1` izango da
5. Behemuga goimuga baino handiagoa denean, ez da iteraziorik exekutatuko eta ondorioz kontroleko aldagaiak `FOR-DO` agindu errepikakorrera heldu baino lehen izandako balioa mantenduko du

`FOR-DOWNTO-DO` eskema aurrekoaren berbera litzateke, ondoko aldaketak aintzat harturik noski. Sarrerako baldintza: baldintzaS → `Goimuga >= Behemuga`. Irteerako baldintza: baldintzaI → `Kontrol = Behemuga`. Iterazio bakoitzeko, kontroleko aldagaia inkrementatu beharrean dekrementatu egiten da.

5.3.3.1 Adibidea

Behemuga eta goimuga lanak beteko dituzten bi zenbaki irakurri, eta haien artean aurkitzen diren bakoitien batura lortu:

```
PROGRAM N_M_artekoZenbakiBakoitienBatura ;          { \TP70\05\FOR2.PAS }
VAR
      Kont, BeheM, GoiM, Batura : Integer ;
BEGIN
      REPEAT
            Write ('Behemuga eman: ') ;
            ReadLn (BeheM) ;
            Write ('Goimuga eman: ') ;
            ReadLn (GoiM) ;
      UNTIL GoiM > BeheM ;

      WriteLn ('Batura     Kont') ;
      WriteLn ('---------------') ;
      Batura := 0 ;
      FOR Kont:=BeheM TO GoiM DO
      BEGIN
         IF Kont MOD 2 <> 0 THEN
         BEGIN
            WriteLn (Batura:4, '   +', Kont:5) ;
            Batura := Batura + Kont  ;
         END ;
      END ;

      WriteLn ('Bakoitien batura -----> ', Batura) ;
END.
```

`REPEAT-UNTIL` baten bitartez `BeheM` eta `GoiM` datuen iragazketa egiten da, ondoren `FOR-DO` prozesu errepikakorra hasten da. Honelako irteera gerta daiteke:

```
Behemuga eman: 4
Goimuga eman: 11
Batura     Kont
---------------
   0   +    5
   5   +    7
  12   +    9
  21   +   11
Bakoitien batura -----> 32
_
```

5.3.3.2 Kontra adibidea

Zenbaki naturalen artean (osoak eta positiboak) bereizketa bat egiten da, zatigarriak direnak eta zatigarriak ez direnak. Azken hauek zenbaki lehenen multzoa osaten dute. Honela definitzen da zenbaki lehena: *"Z zenbaki bat lehena izateko, 1-gatik eta Z-gatik zatigarria izango da baina 2 eta Z-1 artekoen zatiketak egin ondoren hondarra lortzen da"*.

`ZenbakiLehenaOteDenGAIZKI` programan `Znbk` zenbaki natural bat teklatuaren bitartez irakurriko da, eta programak zenbakia lehena ala zatigarria den erantzungo du.

Horretarako algoritmoa hauxe da: `Znbk` datuari zenbait zatiketa aplikatu, zatiketa osoen zatitzailea `2` eta `Znbk-1` tartean dauden zenbaki osoak izango dira, eta ateratzen diren hondarrei so eginez baten bat `0` bada, `Znbk` datua zatigarria izango denez laguntzailea den `LehenaDa` aldagai boolearrak `FALSE` balioko du.

ZenbakiLehenaOteDenGAIZKI programa exekutatzen bada, ondo dabilela ikus daiteke. Baina hala ere ezin dugu ZenbakiLehenaOteDenGAIZKI kodeketa ontzat eman; izan ere, errepikapenak eteteko FOR-DO barnean Kont kontroleko aldagaia aldatzen da, Turbo Pascal lengoaiak sintaktikoki hori onartu arren guk gaitzetsiko dugu, eta irteerako baldintza (Kont=Znbk-1) gertatu baino lehen FOR-DO agindua amaitzea nahi badugu WHILE-DO edo REPEAT-UNTIL bat erabiliko dugu. Ikus ZenbakiLehenaOteDenONGI izeneko programa.

```
PROGRAM ZenbakiLehenaOteDenGAIZKI ;                  { \TP70\05\FOR3.PAS }
VAR
      Znbk, Kont : Integer ;
      LehenaDa : Boolean ;
BEGIN
      REPEAT
            Write ('Osoa eta positiboa den zenbaki bat eman: ') ;
            ReadLn (Znbk) ;
      UNTIL Znbk > 0 ;

      LehenaDa := TRUE ;

      FOR Kont:=2 TO Znbk-1 DO
      BEGIN
            WriteLn (Znbk:6, ' zati ', Kont) ;
            IF Znbk MOD Kont = 0 THEN
            BEGIN
                  LehenaDa := FALSE ;
                  Kont := Znbk-1 ;
            END ;
      END ;

      IF LehenaDa THEN
            WriteLn (Znbk, ' zenbakia lehena da')
      ELSE
            WriteLn (Znbk, ' zenbakia zatigarria da') ;
END.
```

ZenbakiLehenaOteDenONGI programak jarraitzen duen algoritmoa hauxe da: hasteko Znbk datua zenbaki lehena dela suposatuko dugu (datua zenbaki zatigarria ez dela suposatzen ari gara), WHILE-DO baten bitartez 2 eta Znbk-1 artean zatitzailerik dagoen aztertu eta baiezkoan bigizta eteteko duen LehenaDa aldagai boolearrean FALSE gorde, bestela hasieraketaren TRUE balioak jarraituko du:

```
PROGRAM ZenbakiLehenaOteDenONGI ;                    { \TP70\05\FOR4.PAS }
VAR
      Znbk, Kont : Integer ;
      LehenaDa : Boolean ;
BEGIN
      REPEAT
            Write ('Osoa eta positiboa den zenbaki bat eman: ') ;
            ReadLn (Znbk) ;
      UNTIL Znbk > 0 ;

      LehenaDa := TRUE ;
      Kont := 2 ;

      WHILE (Kont <= Znbk-1) AND LehenaDa DO
      BEGIN
            WriteLn (Znbk:6, ' zati ', Kont) ;
            IF Znbk MOD Kont = 0 THEN
                  LehenaDa := FALSE
            ELSE
                  Kont := Kont + 1 ;
      END ;
```



```
      IF LehenaDa THEN
            WriteLn (Znbk, ' zenbakia lehena da')
      ELSE
            WriteLn (Znbk, ' zenbakia zatigarria da') ;
END.
```

Programa biek algoritmo bera darabilte, baina lehenak hori gauzatzeko egitura desegokia aukeratu du. Programen irteera honelako zerbait izan daiteke:

```
Osoa eta positiboa den zenbaki bat eman: 5
     5 zati 2
     5 zati 3
     5 zati 4
5 zenbakia lehena da
_
```

5.3.3.3 Adibidea

Zenbait funtzio matematiko segida jakin baten bitartez definitzerik dago. Esate baterako, kosinu funtzio trigonometrikoa Taylorren bitartez honelaxe kalkulatzen da:

$$\text{Kos}(x) = {x^0}/{0!} - {x^2}/{2!} + {x^4}/{4!} - {x^6}/{6!} + \ldots$$

Non x zenbaki erreal bat den eta radianetan emaniko angelua adierazten duen.

`KosinuaTaylorBitartez` lortzen duen programan, sarrerako datua, `rAng` angelua, gradutan irakurriko dugunez, ezer baino lehen radianetara igaroko dugu. Eta ondoren segidaren balioa kalkulantzen jardungo gara `EPSILON` doitasun konstante eta zehatz batetarako. Gure programaren zuzentasuna frogatzearren lorturiko emaitza, eta `cos()` funtzio estandarrak eskaintzen duena alderatu egiten dira:

```
PROGRAM KosinuaTaylorBitartez ;                          { \TP70\05\FOR5.PAS }
USES
      Crt ;
CONST
      EPSILON = 0.0005 ;
VAR
      iKont, iFaktoriala, iZenbakia, iZeinua, iKontagailu : Integer ;
      liBatuketa : LongInt ;
      rAng, rX, rKosinua, rBatugai, rBerreketa : Real ;
BEGIN
      ClrScr ;
      REPEAT
            Write ('Lehenengo koadranteko angelua graduetan: ') ;
            ReadLn (rAng) ;
      UNTIL (rAng <= 90) AND (rAng >= 0) ;

      rX := rAng * 2 * Pi / 360 ;  (* sarrerako angelua radianetara igaro *)

      iKont := 0 ;
      rKosinua := 0 ;
      iZeinua := 1 ;
      rBatugai := 1 ;
      WHILE Abs(rBatugai) > EPSILON DO
      BEGIN
            rKosinua := rKosinua + rBatugai ;
            iKont := iKont + 2 ;
            iZeinua := (-1) * iZeinua ;      (* zeinuaren kontrola *)

            liBatuketa := 1 ;            (* faktorialaren kalkulua *)
            FOR iKontagailu:=1 TO iKont DO
                  liBatuketa := liBatuketa * iKontagailu ;
```

```
            rBerreketa := 1 ;             (* berreketaren kalkulua *)
            FOR iKontagailu:=1 TO iKont DO
                  rBerreketa := rBerreketa * rX ;

            rBatugai := iZeinua * rBerreketa / liBatuketa ;
            WriteLn (iKont DIV 2, '. iterazioan =====> ', Abs(rBatugai):0:5) ;
      END ;

      WriteLn ('kosinua[', rAng:0:3, 'º] -----> ', rKosinua:0:5) ;
      WriteLn ('cosinus(', rAng:0:3, 'º) -----> ', cos(rX):0:5) ;
END.
```

Azter dezagun `KosinuaTaylorBitartez` programaren kodeketa. *Segidaren osagaiak kalkulatzen eta metatzen jarraitu*, horixe litzateke esaldi bakar batean programak egiten duena. Zeregin errepikakorra denez, bigiztatik baldintza bat ezarri beharra dago, esate baterako kalkulatu den azken batugaiaren garrantzia txikia izatea[8] (`EPSILON` konstanteak duen esanahia horixe da).

Hasieraketa egokiak egin ondoren `WHILE` bigizta barruko funtzezko aginduak hauek lirateke:

```
            rKosinua := rKosinua + rBatugai ;
            iKont := iKont + 2 ;
```

Baina noski `rBatugai` aldagaiari dagokion balioa kosinuari metatu aurretik kalkulatu beharra dago, eta ondoren garrantziduna izango den frogatu beharra dago (hots, jarritako doitasuna baino handiagoa dela). Aipatutako `rBatugai` aldagaiaren balioa lortzeko hiru eginkizun bete behar dira:

1. zeinua kontrolatu
2. faktoriala kalkulatu
3. berreketa zehaztu

Hona hemen `30` gradutako angeluari dagokion kosinua:

```
Lehenengo koadranteko angelua graduetan: 30
1. iterazioan =====> 0.13708
2. iterazioan =====> 0.00313
3. iterazioan =====> 0.00003
kosinua[30.000º] -----> 0.86605
cosinus[30.000º] -----> 0.86603
_
```

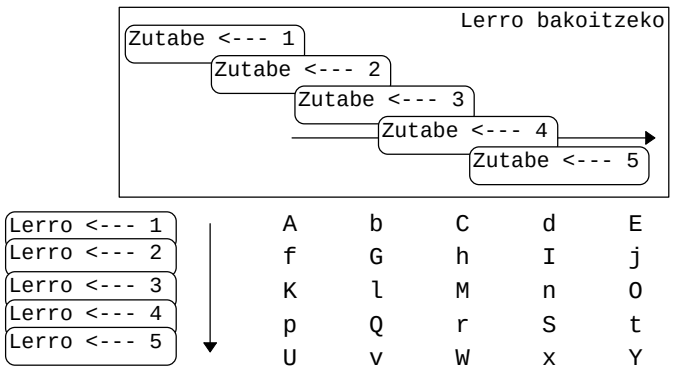
5.3.3.4 Adibidea

Egitura errepikakorretan oinarrituta, ondoko karaktereen taula erakusten duen programa idatzi behar da:

```
    A    b    C    d    E
    f    G    h    I    j
    K    l    M    n    O
    p    Q    r    S    t
    U    v    W    x    Y

_
```

[8] Konturatu Taylor izeneko segidaren osagaiak, balio absolutuan, gero eta txikiagoak direla.

Horretarako, eta taula bat dela kontsideratuz, bi `FOR-DO` beharko ditugu. `FOR-DO` bat bestearen barnean kabiaturik aurkituko da, kanpoko bigiztak lerroak kontrolatzen ditu, eta barrukoak zutabeak. Horregatik lerroak zenbatzen dituen `Lerro` kontagailuak `1`-tik `5` bitartera aldatuko da, modu beretsuan `Zutabe` kontagailuaren ibiltartea `1`-tik `5` bitartera doana izango da.



`FOR Lerro:=1 TO 5 DO` aginduaren arabera, hasiera batean `Lerro` kontagailuak `1` balio duelarik, kabiaturik dagoen bigarren bigizta hasten da. Hots, `FOR Zutabe:=1 TO 5 DO` agindua. Azken bigarren honek amaiera lortuko duenean `Zutabe` kontagailuak `5` balioko du, eta kanpoko bigiztaren bigarren iterazioa (`Lerro` kontagailuari `2` dagokionaren iterazioa) hasiko da.

`KaraktereenTaula1` izeneko programan ikus daitekeenez, barneko bigiztan `Write` prozedura erabiltzen da, eta bost karaktereak idatzi ondoren (kanpoko bigiztan) `WriteLn` bitartez lerroz aldatzen da:

```
PROGRAM KaraktereenTaula1 ;                          { \TP70\05\FOR6.PAS }
VAR
      Lerro, Zutabe : Integer ;
      Karaktere : Char ;
      Nagusia : Boolean ;
BEGIN
      Karaktere := 'a' ;
      Nagusia := TRUE ;
      FOR Lerro:=1 TO 5 DO
      BEGIN
            FOR Zutabe:= 1 TO 5 DO
            BEGIN
                  IF Nagusia THEN
                    BEGIN
                         Write (UpCase(Karaktere):5) ;
                         Nagusia := FALSE ;
                    END
                  ELSE
                    BEGIN
                         Write (Karaktere:5) ;
                         Nagusia := TRUE ;
                    END ;

                  Karaktere := Succ (Karaktere) ;
            END ;
            WriteLn ;
      END ;
END.
```

`KaraktereenTaula1` programari aldaketa txiki bat egitea proposatzen da. Beti 5x5 taula bera atera beharrean, 1 eta 25 bitarteko zenbaki oso bat teklatuz irakurri eta zenbaki horrek adierazten duen kopuruen letrak idatzi, azken lerroa amaitu gabe geratzen bada `%` sinboloz osatu. Programa honek `KaraktereenTaula2` izena izango du eta bigarren bertsio

honetan ezin dira FOR-DO egiturak erabili, horregatik kanpoko bigiztarako WHILE-DO agindua jarri da. KaraktereenTaula2 programak honelako irteerak izan ditzake:

```
Zenbaki oso bat eman: 17
    A    b    C    d    E
    f    G    h    I    j
    K    l    M    n    O
    p    Q    %    %    %
_
```

Hona hemen KaraktereenTaula2 programa:

```
PROGRAM KaraktereenTaula2 ;                          { \TP70\05\FOR7.PAS }
VAR
       Lerro, Zutabe, Kopurua : Integer ;
       Karaktere : Char ;
       Nagusia : Boolean ;
BEGIN
       REPEAT
              Write ('Zenbaki oso bat eman: ') ;
              ReadLn (Kopurua) ;
       UNTIL (Kopurua >= 1) AND (Kopurua <= 25) ;

       Karaktere := 'a' ;
       Nagusia := TRUE ;
       Lerro := 0 ;
       WHILE Lerro*5 < Kopurua DO
       BEGIN
              FOR Zutabe:= 1 TO 5 DO
              BEGIN
                IF Nagusia THEN
                  BEGIN
                     IF Lerro*5+Zutabe <= Kopurua THEN
                            Write (UpCase(Karaktere):5)
                     ELSE
                            Write ('%':5) ;
                     Nagusia := FALSE ;
                  END
                ELSE
                  BEGIN
                     IF Lerro*5+Zutabe <= Kopurua THEN
                            Write (Karaktere:5)
                     ELSE
                            Write ('%':5) ;
                     Nagusia := TRUE ;
                  END ;

                Karaktere := Succ (Karaktere) ;
              END ;

              WriteLn ;
              Lerro := Lerro + 1 ;
       END ;
END.
```

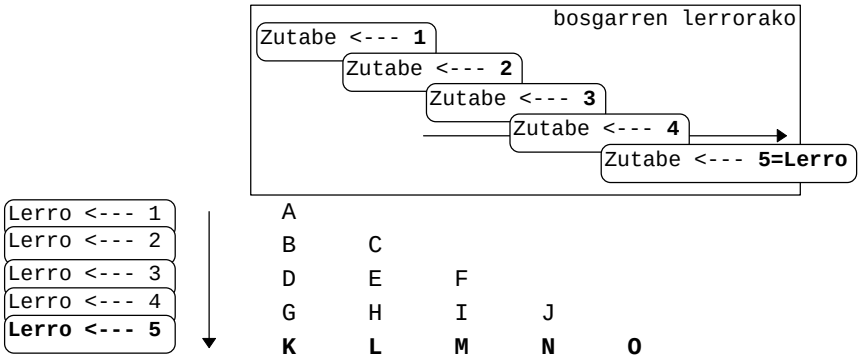
5.3.3.5 Adibidea

5.3.3.4 puntuan idatzi den KaraktereenTaula1 izeneko programan FOR-DO bikote batez ebatzi da, A eta Y artean dauden karaktereak bost zutabetan antolatu direnez taula 5x5 karratua atera izan da. Egindako hori kontutan izanik (hots, FOR-DO kabiatuak jarriz) ondoko karaktereen piramidea erakusten duen programa idatzi:

```
    A
    B    C
    D    E    F
    G    H    I    J
    K    L    M    N    O
    P    Q    R    S    T    U
_
```

Lehen bezalaxe bi `FOR-DO` beharko ditugu, `FOR-DO` bat bestearen barnean kabiaturik aurkituko delarik. Lerroak kontrolatzen dituen kanpoko bigiztari dagozkion behemuga eta goimuga finkoak eta ezagunak dira, lerroak zenbatzen dituen `Lerro` kontagailuak `1`-tik `5` bitartera doa. Zutabeak berriz, aldakorrak dira, lehenengo lerroan adibidez `Zutabe` izeneko kontagailuaren ibiltartea `1`-tik `1` bitartera doa, eta bigarrenean `Zutabe` kontagailu horri `1`-tik `2` bitartera joatea egokituko zaio.

Orokorki, lerroen kontagailuak `Lerro` balio duenean, zutabeen kontagailuak `1`-tik `Lerro` bitartera joan beharko du:



Hona hemen `KaraktereenPiramidea1` programaren kodeketa:

```
PROGRAM KaraktereenPiramidea1 ;                          { \TP70\05\FOR8.PAS }
VAR
      Lerro, Zutabe : Integer ;
      Karaktere : Char ;
BEGIN
      Karaktere := 'A' ;
      FOR Lerro:=1 TO 5 DO
      BEGIN
            FOR Zutabe:= 1 TO Lerro DO
            BEGIN
                  Write (Karaktere:5) ;
                  Karaktere := Succ (Karaktere) ;
            END ;
            WriteLn ;
      END ;
END.
```

`KaraktereenPiramidea1` programa aldatu, 1 eta 25 arteko zenbaki osoa eman eta honelako irteerak eskainiko ditu. Bertsio berriari `KaraktereenPiramidea2` deitu:

```
Zenbaki oso bat eman: 17
    A
    B    C
    D    E    F
    G    H    I    J
    K    L    M    N    O
    P    Q    %    %    %    %
_
```

5.3.3.6 Adibidea

Zenbaki bat teklatuz irakurri eta zenbaki-erdigunea den ala ez zehaztu. Zenbaki-erdiguneek ondoko baldintza betetzen dute: *"Z zenbaki-erdiguneak zenbaki osoak bi zerrendetan banatzen ditu (lehenengo zerrenda 1-tik Z-1 bitartera, bigarren zerrenda Z+1 zenbakian hasten da). Z zenbakia zenbaki-erdigunea izateko zerrenden zenbakiak batuz, batura biak berdinak izango dira"*.

Oso guztien artetik lehendabiziko zenbaki-erdigunea 6 da, zeinek (1,2,3,4,5) eta (7,8) zerrendak banatzen dituen; ikus daitekeenez zerrenda biren batura 15 da. Bigarren zenbaki-erdigunea 35 da, dagozkion zerrendak (1, ... ,34) eta (36, ... ,49) izanik edozeinaren batura 595 da.

`Muga` teklatuz irakurri ondoren, `ZenbakiErdiguneakBilatu` programan `1` eta `Muga` artean dauden zenbaki-erdiguneak bilatzen dira:

```
PROGRAM ZenbakiErdiguneakBilatu ;                    { \TP70\05\FOR10.PAS }
USES
   Crt ;
VAR                                                (* aldagai orokorrak *)
  Muga, Kont, k  : Integer ;
  AurrekoBatura, AtzekoBatura : LongInt ;
BEGIN
   ClrScr ;
   REPEAT
     Write ('Muga eman: ') ;
     ReadLn (Muga) ;
   UNTIL Muga > 0 ;

   FOR Kont:=2 TO Muga DO
   BEGIN
      AurrekoBatura := 0 ;
      FOR k:=1 TO Kont-1 DO
           AurrekoBatura := AurrekoBatura + k ;

      AtzekoBatura := 0 ;
      k := Kont + 1 ;
      WHILE AtzekoBatura < AurrekoBatura DO
      BEGIN
           AtzekoBatura := AtzekoBatura + k ;
           k := k + 1 ;
      END ;

      IF AtzekoBatura = AurrekoBatura THEN
          Write (Kont:10) ;
   END ;
END.
```

`ZenbakiErdiguneakBilatu` programan `Kont` balio guztietarako[9] erdigunea den ala ez frogatzen da. Horretarako, `AurrekoBatura` eta `AurrekoBatura` aldagaietan aurreko eta atzeko zerrenden baturak kalkulatu eta gordetzen dira, baina `Kont=1` denean (hasieraketen balioak direla eta) `1` zenbakia zenbaki-erdigune ateratzen da, hori ekiditeko kanpoko bigizta nagusia `2`-tik hasten da.

Hona hemen `ZenbakiErdiguneakBilatu` programaren irteera bat:

```
Muga eman: 5000
         6        35       204      1189
_
```

[9] Logikoki `Kont` aldagaiak dituen behe eta goi mugak `1` eta `Muga` izan beharko lirateke.

5.3.3.7 Adibidea

ZenbakiLehenakBilatu izeneko programa idatzi **5.3.3.2** eta **5.3.3.6** puntuetan egindako adibideak kontutan izanik. Muga teklatuz irakurri ondoren, FOR-DO baten barnean Kont kontagailua zenbaki lehena ote den frogatu:

```
PROGRAM ZenbakiLehenakBilatu ;                        { \TP70\05\FOR11.PAS }
VAR
      Muga, Kont, k : Integer ;
      LehenaDa : Boolean ;
BEGIN
      REPEAT
            Write ('Osoa eta positiboa den zenbaki bat eman: ') ;
            ReadLn (Muga) ;
      UNTIL Muga > 0 ;

      FOR Kont:=1 TO Muga DO
      BEGIN
            LehenaDa := TRUE ;
            k := 2 ;
            WHILE (k <= Kont-1) AND LehenaDa DO
            BEGIN
                  IF Kont MOD k = 0 THEN
                        LehenaDa := FALSE
                  ELSE
                        k := k + 1 ;
            END ;
            IF LehenaDa THEN
                  Write (Kont:10) ;
      END ;
      WriteLn ;
END.
```

Hauxe litzateke ZenbakiLehenakBilatu programaren irteera bat:

```
Muga eman: 15
         1         2         3         5         7        11        13
_
```

5.3.3.6 eta **5.3.3.7** puntuetako programa bietan egitura berbera erabiltzen da, hots FOR-DO batek zeregin errepikakor oso zehatza kontrolatzen du. Batean oso bat zenbaki-erdigunea oten den frogatzen da, eta bestean osoa zenbaki lehena den ala ez aztertzen da. Edozein kasutan errepikatzen diren eginkizunak bereziak eta berezko nortasunak dituzte, beraz FOR-DO barruan honelako zerbait egitea oso interesgarria litzateke:

```
FOR Kont:=2 TO Muga DO
BEGIN
    ErdiguneaDa := Zehaztu (Kont) ;
    IF ErdiguneaDa THEN
            Write (Kont:10) ;
END ;
```

```
FOR Kont:=1 TO Muga DO
BEGIN
      LehenaDa := Frogatu (Kont) ;
      IF LehenaDa THEN
            Write (Kont:10) ;
END ;
```

Zehaztu() eta Frogatu() izeneko elementuak funtzioak dira, orain arte erabili diren Cos(), UpCase(), Lengths() ... bezalakoak[10]. Lehenengoak, Zehaztu() izenekoak zenbaki oso bat hartu eta erdigunea bada TRUE itzultzen du, bestela FALSE. Bigarrenak, Frogatu() izenez ezagutzen dena funtzio boolearra da ere, hartutako zenbaki osoa lehena bada TRUE eta zatigarria bada FALSE itzultzen du.

Hurrengo kapituloan honelako azpiprogramak erabiltzen ikasiko dugu.

[10] Hau da duten berezitasun bakarra da: estandarrak izan beharrean erabiltzaileak asmatutakoak dira.

5.4 PROGRAMAZIO ARIKETAK EBAZTEKO URRATSAK

Lehenengo kapituluan esan zen bezala planteaturiko arazo bat soluzionatzeko lau urrats betetzen dira, problema bati erantzuten dion programa lortzeko lau urratsak hauek dira:

1. *Arazoaren definizioa*
2. *Algoritmoa asmatu*
3. *Algoritmoa programa bezala idatzi*
4. *Soluzioa ebaluatu*

Esate baterako, Charles Babbage (1792-1871) matematikariak, 1822. urtean, *Kenketa-makina* eraiki zuen eta makina hau Diferentzia Finituen printzipio matematikoa zuen oinarritzat. Demagun Diferentzia Finituen printzipioa ordenadore programa batean gauzatu nahi dugula.

Lehenengo kapituluan aipatu zen polinomien taulak osatzeko asmoz erabil daitekeen Diferentzia Finituen printzipioak zati bi ditu, batetik polinomio jakin bati dagokion diferentzia finutua kalkulatu behar da, eta, bestetik, diferentzia finitua aintzat hartuz polinomioa puntu ezberdinetan ebaluatu.

5.4.1 eta **5.4.2** puntuetan arazo biak soluzionatuko dira urratsez urrtas. **5.4.1** puntuan hirugarren ordenako polinomio baten koefizienteak datutzat hartuz dagokion diferentzia finitua lortuko da. **5.4.2** puntuan aurreko ariketa osatuko dugu, izan ere hirugarren ordenako polinomio baten koefizienteak datuetatik abiatuz, dagokion diferentzia finitua lortu ondoren, abzisa jakin bat eman eta polinomioaren balio kalkulatuko da Diferentzia Finituen printzipioa aplikatuz.

5.4.1 Diferentzia Finituen metodoa (zenbaki osoekin)

Helburua: hirugarren ordenako polinomio baten A, B, C eta D koefizienteak teklatutik hartu ondoren, funtzio horri[11] dagokion diferentzia finitua lortu.

Metodologia: Lau urratsetan banatuko dugu lana.

5.4.1.1 Arazoaren definizioa.
5.4.1.2 Algoritmoa asmatu
5.4.1.3 Algoritmoa programa bezala idatzi
5.4.1.3 Soluzioa ebaluatu

5.4.1.1 Arazoaren definizioa

Hona hemen diferentzia finituen metodoa zertan den:

Demagun, diferentzia finituen metodoaren bitartez $y=x^3-3x^2+2$ funtzioari dagokion taula lortu nahi dela. Bost zutabeko taula eraiki beharko litzateke, non zutabe bakoitzaren

[11] 10. kapituluan, array edo matrize kontzeptua ikasi ondoren, edozein *N* ordenako polinomioekin lan egingo dugu. Ikus `DIFFINIT.PAS` fitxategiko `DiferentziaFinituak` izena duen programa, non funtzioa *N* ordenakoa izanik a_n, a_{n-1}, a_{n-2}, ... a_1, a_0 koefizienteak zenbaki osoak diren.

deskribapena hau litzatekeen: lehenengo zutabea x, bigarren zutabea y, hirugarren zutabea y_n eta y_{n+1} arteko kendura (lehenengo diferentzia), laugarren zutabea u_n eta u_{n+1} arteko kendura (bigarren diferentzia), eta, bosgarren zutabea konstantea den v_n eta v_{n+1} arteko kendura (hirugarren diferentzia):

		lehenengo diferentzia	*bigarren diferentzia*	*hirugarren diferentzia*
x	*y = x^3-3x^2+2*	*u*	*v*	*w*
1	0			
		-2		
2	-2		6	
		4		6
3	2		12	
		16		6
4	18		18	
		34		6
5	52		24	
		58		
6	110			
7	**198**			

Ikus daitekeenez, lehenengo diferentziari dagokion zutabea honelaxe kalkulatzen da:

$u_n = y_{n+1} - y_n$

```
-2 = -2 - 0
 4 = 2 - (-2)
16 = 18 - 2
34 = 52 - 18
58 = 110 - 52
...
```

Bigarren diferentziari dagokion zutabea:

$v_n = u_{n+1} - u_n$

```
 6 = 4 - (-2)
12 = 16 - 4
18 = 34 - 16
24 = 58 - 34
...
```

Hirugarren diferentziari dagokion zutabea:

$w_n = v_{n+1} - v_n$

```
6 = 12 - 6
6 = 18 - 12
6 = 24 - 18
...
```

y=x^3-3x^2+2 funtzioa hirugarren ordenakoa delako, hirugarren diferentzia konstantea izango da beti, hau da hain zuzen ere diferentzia finituen printzipio matematikoa eta edozein n ordenako polinomiotan ere betetzen da (n-garren diferentzia konstantea izango da).

Demagun y=x^3-3x^2+2 funtzioaren balioa ezagutu nahi dela x=7 denean, 7^3-3(7^2)+2 egingo bagenu 198 aterako litzaiguke. Emaiza bera eskura genezake difirentzia finituen metodoari esker, behar diren datuak aurreko taulan daude, hots, x=1 eta x=6 arteko taula izanez gero, ez dugu 7-ren kuboa kalkulatu behar nahikoa baita gezi beltzak adierazten duen zenbakiak batzea, euren batura 198 izango da.

$7^3-3(7^2)+2 = 198 = 6 + 24 + 58 + 110$

w (x=6) w (x=6) w (x=6) y (x=6

Zenbaki osoekin lan egiten duen `HasieraketaTaula3O` programak hirugarren ordenako polinomioak ebatzi ahal izango ditu, eta bete behar dituen espezifikazioak datorren puntuan zehazten dira.

5.4.1.2 Algoritmoa asmatu

Diferentzi Finituen taula osatzen duen `HasieraketaTaula3O` programaren algoritmoa.

Arazo orokorra:

Hirugarren ordenako funtzioak ebazteko Diferentzi Finutuen taula osatu.

Sarrerako datuak:

Polinomioaren `A`, `B`, `C` eta `D` lau koefizienteak (positiboak edo negatiboak izan daitezkeen zenbaki osoak).

Prozesua:

Algoritmoak lau urrats ditu, eta bosgarren bat soluzioaren zuzentasuna ebaluatu ahal izateko balioko diguna.

1. Polinomioa aplikatuz funtzioaren lehen lau balioak kalkulatu, hau da, $x=1$ denean y_1 lortu, $x=2$ denean y_2 lortu, $x=3$ denean y_3 lortu eta $x=4$ izatean y_4 lortu. Lehen urratsean taulako lehen zutabea kalkulatu egiten da.
2. Taulako ezkerreko zutabea dugularik, *lehenengo diferentzia* izenburuko zutabearen balioak eskuratuko dira horretarako $u_n = y_{n+1} - y_n$ formula hiru aldiz aplikatuko da u_1, u_2 eta u_3 zehaztuz.
3. Taulako bigarren zutabean oinarriturik, hirugarren zutabeko elementuak, *bigarren diferentzia* liratekeenak, kalkulatu ahalko dira $v_n = u_{n+1} - u_n$ formula bi aldiz aplikatuz v_1 eta v_2 lortzeko.
4. Taulako hirugarren eta azken zutabea konstantea den diferentzi finitua litzateke, hots, hirugarren zutabeko v_1 eta v_2 balioen arteko kenketa eginez w_1 erdiesten da.
5. Diferentzi Finituen taula osaturik dagoela, x ezberdinak emanik dagozkien y-ak lor daitezke polinomioaren bidez eta Diferentzi Finituen printzipioaren bitartez.

Adibidea:

Datuak: `A` = 1
`B` = -2
`C` = 0
`D` = 3

y_n	u_n	v_n	w_n
0			
-2	-2		
2	4	6	
18	16	12	6

5.4.1.3 Algoritmoa programa bezala idatzi

Hauxe da Diferentzi Finituen taula osatzen duen programa, ikusten denez hirugarren ordenako funtzioak ebaluatzeko asmatuta dago. Eta funtzioaren diferentzia finituak lortzeko funtzioa lau alditan kalkulatu beharra dago, hona hemen bere balioak biltegitzen dituzten aldagaien izenak: `FuntzioaJoandako`, `FuntzioaLehenago`, `FuntzioaOraintsu`, eta `FuntzioaOraintxe`.

```
PROGRAM HasieraketaTaula3O ;                    { \TP70\05\DIF_FIN1.PAS }
USES                                            { Hasieraketa-Taula     }
   Crt ;                                        { zenbaki osoekin       }

VAR
   A, B, C, D, Znbk, i, Emaitza : LongInt ;
   Dif1, AurrekoDif1, AintzinDif1, Dif2, AurrekoDif2, Dif3 : Integer ;
   FuntzioaJoandako, FuntzioaLehenago, FuntzioaOraintsu, FuntzioaOraintxe : LongInt ;
   Itxaron : Char ;

BEGIN
  ClrScr ;
  WriteLn ('"Diferentzia Finitua" metodoan urrats bi daude. Batetik diferentzien') ;
  WriteLn ('taula osatu behar da (programa honek egiten duen gauza bakarra). Eta') ;
  WriteLn ('bestetik, funtzio jakin bati dagokion diferentzien taula hori dugula') ;
  WriteLn ('funtzioaren balioak kalkula daitezke diferentzien batuketak eginez. ') ;
  WriteLn ;
  WriteLn ('Programa honetan zenbaki osoekin lan egiten dela aintzat hartu.') ;
  Itxaron := ReadKey ;
  ClrScr ;
  WriteLn ('Polinomioaren A, B, C eta D koefizienteak zehaztu') ;
  WriteLn ('-------------------------------------------------') ;
  Write ('Polinomioaren 3. ordenako koefizientea eman: ') ;
  ReadLn (A) ;
  Write ('Polinomioaren 2. ordenako koefizientea eman: ') ;
  ReadLn (B) ;
  Write ('Polinomioaren 1. ordenako koefizientea eman: ') ;
  ReadLn (C) ;
  Write ('Polinomioaren koefiziente askea eman:       ') ;
  ReadLn (D) ;

  WriteLn ;
  WriteLn ('Funtzioaren lehen lau balioak, polinomioa aplikatuz') ;
  WriteLn ('---------------------------------------------------') ;
  FuntzioaJoandako := A*(1*1*1) + B*(1*1) + C*1 + D ;
  FuntzioaLehenago := A*(2*2*2) + B*(2*2) + C*2 + D ;
  FuntzioaOraintsu := A*(3*3*3) + B*(3*3) + C*3 + D ;
  FuntzioaOraintxe := A*(4*4*4) + B*(4*4) + C*4 + D ;

  WriteLn ('Joandako=', FuntzioaJoandako) ;
  WriteLn ('Lehenago=', FuntzioaLehenago) ;
  WriteLn ('Oraintsu=', FuntzioaOraintsu) ;
  WriteLn ('Oraintxe=', FuntzioaOraintxe) ;

  WriteLn ;
  WriteLn ('"Diferentzia Finituen" metodoaren hasieraketa-taula') ;
  WriteLn ('--------------------------------------------------') ;
  AintzinDif1 := FuntzioaLehenago - FuntzioaJoandako ;
  AurrekoDif1 := FuntzioaOraintsu - FuntzioaLehenago ;
  Dif1 := FuntzioaOraintxe - FuntzioaOraintsu ;

  AurrekoDif2 := AurrekoDif1 - AintzinDif1 ;
  Dif2 := Dif1 - AurrekoDif1 ;

  Dif3 := Dif2 - AurrekoDif2 ;
  WriteLn ('Joandako=', FuntzioaJoandako:4) ;
  WriteLn ('Lehenago=', FuntzioaLehenago:4, '    AintzinDif1=', AintzinDif1:4) ;
  Write ('Oraintsu=', FuntzioaOraintsu:4, '    AurrekoDif1=', AurrekoDif1:4) ;
  WriteLn ('    AurrekoDif2=', AurrekoDif2) ;
  Write   ('Oraintxe=', FuntzioaOraintxe:4, '           Dif1=', Dif1:4        ) ;
  WriteLn ('           Dif2=', Dif2, '    Dif3=', Dif3) ;

  FuntzioaJoandako := FuntzioaLehenago ;
  FuntzioaLehenago := FuntzioaOraintsu ;
  FuntzioaOraintsu := FuntzioaOraintxe ;


  { Hasieraketa hemen bukatzen da. Diferentzi Finituaren metodoa frogatzeko }
  { funtzioaren hurrengo balioa kalkula daiteke diferentziak batuz.         }
```

```
  WriteLn ;
  WriteLn ('Hurrengo balioa "Diferentzia Finituen" metodoa bitartez:') ;
  Emaitza := Dif3 + Dif2 + Dif1 + FuntzioaOraintxe ;
  WriteLn ('Hurrengoa=', Emaitza, '   <---- Dif3 + Dif2 + Dif1 + FuntzioaOraintxe') ;

  WriteLn ('Funtzioaren hurrengo balioa polinomioa aplikatuz:') ;
  Emaitza := A*(5*5*5) + B*(5*5) + C*5 + D ;
  WriteLn ('Hurrengoa=', Emaitza, '   <---- Ax^3 + Bx^2 + Cx + D') ;
END.
```

5.4.1.4 Soluzioa ebaluatu

`HasieraketaTaula3O` programaren azken urratsean algoritmoaren zuzentasuna frogatzeko funtzioaren hurrengo balioa (*x=5* denerako) kalkulatzen da bi modutan, batetik polinomioa puntu horretan ebaluatuz, eta bestetik Diferentzi Finituen taulako elementuen batura eginez. Ikusten denez gauza bera lortzen da eta algoritmoa ongi dagoela baiezta daiteke.

Hala ere, algoritmoaren portaera puntu bakar batean frogatu ordez geihagotan ikertzeko asmoz beste programa bat osatu dugu, horren izena `DiferentziaFinituak3O` da eta aurreko `HasieraketaTaula3O` programan oinarritzen da. Hauxe litzateke soluzioa ebaluatzen duen `DiferentziaFinituak3O` programaren irteera bat:

```
Polinomioaren A, B, C eta D koefizienteak zehaztu
-------------------------------------------------
Polinomioaren 3. ordenako koefizientea eman: 1
Polinomioaren 2. ordenako koefizientea eman: -3
Polinomioaren 1. ordenako koefizientea eman: 0
Polinomioaren koefiziente askea eman:        2

4 edo handiagoa den zenbaki natural bat eman ondoren
F = (1)*X^3 + (-3)*X^2 + (0)*X + (2)   funtzioaren
balioak "Diferentzia Finitua" metodoz kalkulatuko dira:  12

Dif.Finitu. F[ 5] =  52          Polinom. F"[ 5] =  52
Dif.Finitu. F[ 6] = 110          Polinom. F"[ 6] = 110
Dif.Finitu. F[ 7] = 198          Polinom. F"[ 7] = 198
Dif.Finitu. F[ 8] = 322          Polinom. F"[ 8] = 322
Dif.Finitu. F[ 9] = 488          Polinom. F"[ 9] = 488
Dif.Finitu. F[10] = 702          Polinom. F"[10] = 702
Dif.Finitu. F[11] = 970          Polinom. F"[11] = 970
Dif.Finitu. F[12] =1298          Polinom. F"[12] =1298
_
```

5.4.2 Diferentzia Finituen metodoa (zenbaki errealekin)

Azaldu izan ditugun `HasieraketaTaula3O` eta `DiferentziaFinituak3O` programak zenbaki osoekin lan egiten zuten. Datu-motak aldatuz, zenbaki errealak onartzen dituzten `HasieraketaTaula3E` eta `DiferentziaFinituak3E` programak idatzi ditugu ere .

5.5 PROGRAMAK

Hona hemen 5. kapituluaren programak orrialdeen arabera sailkaturik:

Izena	*Programaren identifikadorea*	*ORRI.*	*Ikasgaia*
IF1.PAS	ErroreakAurreikusiGabe	5-04	Baldintzazko aginduak
IF2.PAS	ErroreakAurreikusten	5-05	Baldintzazko aginduak
ERROAK1.PAS	NolakoErroakDiren	5-06	Baldintzazko aginduak
IF3.PAS	ErroreakAurreikustenIfThenElse	5-08	Baldintzazko aginduak
ERROAK2.PAS	ErroakKalkulatzen	5-10	Baldintzazko aginduak
CASE1.PAS	NotakPantailaratzen	5-11	Baldintzazko aginduak
MENUA.PAS	Aukerak_MenuBatezHautatzen	5-12	Baldintzazko aginduak
WHILE0.PAS	Faktoriala5BigiztarikGabe	5-13	Agindu errepikakorrak
WHILE1.PAS	FaktorialaWhileBigiztarekin	5-15	Agindu errepikakorrak
WHILE2_1.PAS	SegidaWhileBigiztarekin1	5-16	Agindu errepikakorrak
WHILE2_2.PAS	SegidaWhileBigiztarekin2	5-19	Agindu errepikakorrak
WHILE3_1.PAS	PosNegWhileBigiztarekin1	5-19	Agindu errepikakorrak
WHILE3_2.PAS	PosNegWhileBigiztarekin2	5-20	Agindu errepikakorrak
WHILE3_3.PAS	PosNegWhileBigiztarekin3	5-21	Agindu errepikakorrak
REPEAT1.PAS	ErroreakSaihesten	5-22	Agindu errepikakorrak
REPEAT2.PAS	PosNegRepeatBigiztarekin	5-23	Agindu errepikakorrak
NEWTON.PAS	ErroKarratuaNewtonFormulaz	5-24	Agindu errepikakorrak
NEWTON_.PAS	ErroKarratuaNewtonWhile	5-24	Agindu errepikakorrak
FOR1.PAS	FaktorialaForBigiztarekin	5-25	Agindu errepikakorrak
FOR2.PAS	N_M_artekoZenbakiBakoitienBatura	5-27	Agindu errepikakorrak
FOR3.PAS	ZenbakiLehenaOteDenGAIZKI	5-28	Agindu errepikakorrak
FOR4.PAS	ZenbakiLehenaOteDenONGI	5-28	Agindu errepikakorrak
FOR5.PAS	KosinuaTaylorBitartez	5-29	Agindu errepikakorrak
FOR6.PAS	KaraktereenTaula1	5-31	Agindu errepikakorrak
FOR7.PAS	KaraktereenTaula2	5-32	Agindu errepikakorrak
FOR8.PAS	KaraktereenPiramidea1	5-33	Agindu errepikakorrak
FOR9.PAS	KaraktereenPiramidea2	5-33	Agindu errepikakorrak
FOR10.PAS	ZenbakiErdiguneakBilatu	5-34	Agindu errepikakorrak
FOR11.PAS	ZenbakiLehenakBilatu	5-35	Agindu errepikakorrak
DIF_FIN1.PAS	HasieraketaTaula3O	5-39	Algoritmika
DIF_FIN2.PAS	DiferentziaFinituak3O	5-40	Algoritmika
DIF_FIN3.PAS	HasieraketaTaula3E	5-40	Algoritmika
DIF_FIN4.PAS	DiferentziaFinituak3E	5-40	Algoritmika

5.6 BIBLIOGRAFIA

- Wirth, N., *Algoritmos + Estructuras de Datos = Programas*, Ed. Castillo, 1986
- Decker R., Hirshfield S., *Pascal's Triangle*, PWS-Kent Publishing Company, Boston, 1992
- Borland, *TURBO PASCAL 7.0. Erabiltzailearen gida*, Borland International, 1992
- Borland, *TURBO PASCAL 7.0 Ingurunearen laguntza*, Borland International, 1992

6. ATALA: AZPIPROGRAMAK, FUNTZIOAK ETA PROZEDURAK

AURKIBIDEA

6.1 SARRERA

Eguneroko bizitzari begiraturik, agertzen zaigun lan eta arazo guztiak norberak konpontzea ezinezkoa dela onartu beharra dago. Horrela, espezialista beteriko mundu honetan, autoaren mantenua eta konponketak tailerrean egiten dizkigute, jaten ditugun elikagaiak ez ditugu guk produzitzen, ...

Bizitza arruntean, besteek egiten dituzten anitz zerbitzuez baliatzen garen bezala, programazioan ere programa gehienek beste espezialista batzuen beharra izaten dute. Zeregin espezifikoak betetzen dituzten espezialista horiei azpiprograma edo azpirrutina esaten zaie, eta dagoenekoz, besteak beste, `ReadLn()` eta `Cos()` erabili ditugu. Lehenengoaren zeregina teklatuz harturiko balioren bat aldagai bati esleitzea da, eta bigarrenaren betebeharra radianetan emaniko angelu baten kosinua kalkulatzea da.

`ReadLn()` eta `Cos()` azpiprogramak ezagunak dira konpiladorearentzat eta edozein programatan erabiltzeko gaitasuna du programadoreak, nahiz eta azpiprograma horiek barneratzen[1] dituzten sententziak ezagutu ez.

6.2 AZPIPROGRAMA BATEN HELBURUA

Azpiprogramen erabilpena justifikatzeko arrazoi ezberdinak daude, ondoko puntuetan garrantzitsuenak aipatuko ditugu: ahalik eta kode gutxiago idaztea, programa antolaturik egon dadila eta kodearen berrerabilpena.

6.2.1 Kodearen errepikapena ekiditea

Seguruenik hau izan zen azpiprogramak idazteko lehenengo arrazoia. Zenbaitetan programa baten toki ezberdinetan kalkulu berbera burutu izaten da, ondorioz kalkulua betetzen dituzten aginduak toki horietan errepikatu egin beharko dira. Adibidez zenbaki oso baten faktoriala lortzeko bosgarren kapituluko `FaktorialaForBigiztarekin` programa ezagutzen dugu, horren arabera zenbaki oso bat teklatutik irakurri eta dagokion faktoriala pantailaratzen da.

```
PROGRAM FaktorialaForBigiztarekin ;           { \TP70\06\FOR1.PAS }
VAR
      Kontagailua, Zbk : Integer ;
      OrainArtekoa : LongInt ;
BEGIN
      Write (‘Zenbaki osoa eta positiboa eman: ‘) ;
      ReadLn (Zbk) ;

      OrainArtekoa := 1 ;       (* metagailuaren hasieraketa *)

      FOR Kontagailua:=1 TO Zbk DO
            OrainArtekoa := OrainArtekoa * Kontagailua ;

      WriteLn (Zbk, ‘! = ‘, OrainArtekoa) ;
END.
```

[1] `Cos()` azpiprogramaren kasuan, dagokion barne funtzionamenduaren ideia hartzeko, bosgarren kapituluko `KosinuaTaylorBitartez` programa gogoratu. Ez dugu esan nahi Turbo Pascal lengoaiak ezagutzen duen `Cos()` azpiprograma derrigorrez segida horren bitartez egina egon behar denik, baina bai `Cos()` azpiprogramaren kodifikazioa `KosinuaTaylorBitartez` programaren antzekoa izango dela.

Baina demagun bi kopuru osoen arteko zenbaki konbinatorioa lortu nahi dugula. Zenbaki konbinatorioa, kopuru osoak diren *m* eta *n* faktorialetan oinarritzen da eta honela formulatzen da[2]:

$$\binom{m}{n} = \frac{m!}{n!(m-n)!}$$

Zenbaki konbinatorioa kalkulatuko lukeen programa hauxe litzateke:

```
PROGRAM ZenbakiKonbinatorioa ;                { \TP70\06\KONBINAT.PAS }
VAR
       Kontagailua, ZbkM, ZbkN, ZbkM_N : Integer ;
       FaktM, FaktN, FaktM_N : LongInt ;
BEGIN
       REPEAT
          Write (‘m zenbaki osoa eta positiboa eman: ‘) ;
          ReadLn (ZbkM) ;
          Write (‘n zenbaki osoa eta positiboa eman: ‘) ;
          ReadLn (ZbkN) ;
       UNTIL (ZbkM >= 0) AND (ZbkN >= 0) AND (ZbkM > ZbkN) ;

       FaktM := 1 ;                           (* lehengo kalkulua *)
       FOR Kontagailua:=1 TO ZbkM DO
              FaktM := FaktM * Kontagailua ;

       FaktN := 1 ;                           (* bigarren kalkulua *)
       FOR Kontagailua:=1 TO ZbkN DO
              FaktN := FaktN * Kontagailua ;

       ZbkM_N := ZBKM - ZbkN ;
       FaktM_N := 1 ;                         (* hirugarren kalkulua *)
       FOR Kontagailua:=1 TO ZbkM_N DO
              FaktM_N := FaktM_N * Kontagailua ;

       WriteLn (‘Zenbaki konbinatorioa = ‘, FaktM DIV (FaktN * FaktM_N) ) ;
END.
```

Ikus daitekeenez `ZenbakiKonbinatorioa` programan zeregin berdina hiru aldiz egiten da: lehenengo kalkuluan `ZbkM` aldagaiak duena aintzat harturik bere faktoriala lortzen da, bigarrenean eta hirugarrenean gauza bera egiten da baina `ZbkN` eta `ZbkM_N` aldagaietarako. Ondorioz, faktoriala kalkulatzen duen kodea hiru aldiz agertzen da `ZenbakiKonbinatorioa` izeneko programa horretan.

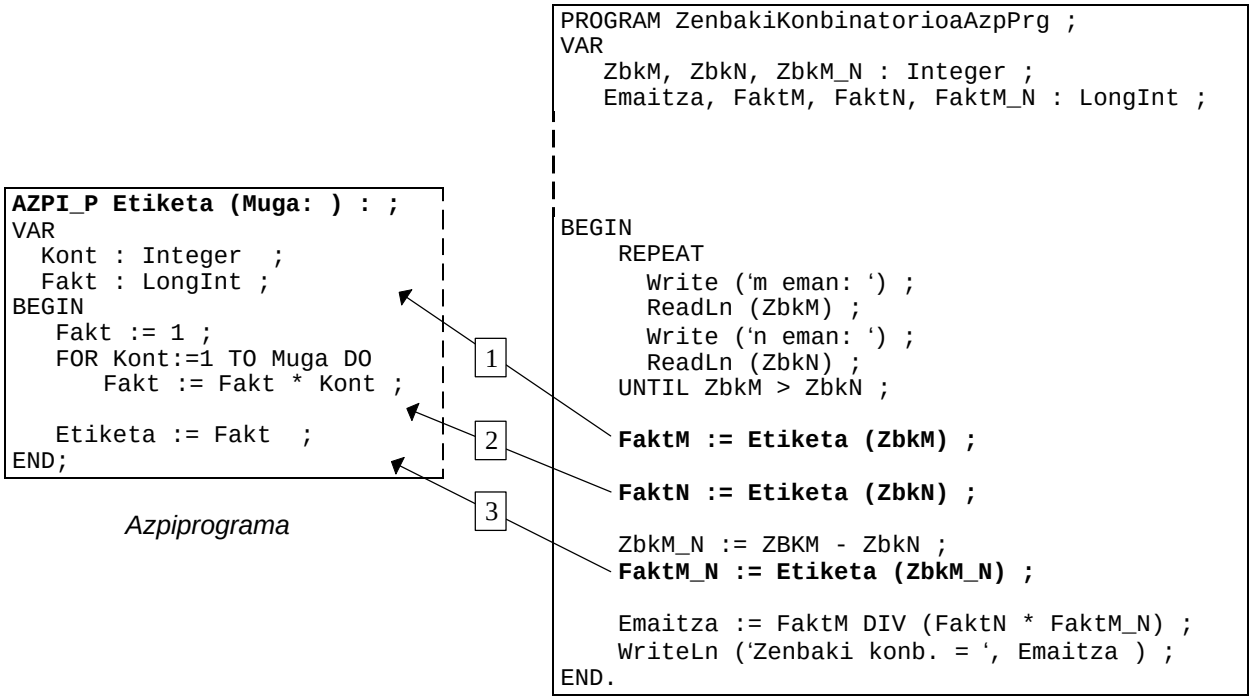
Azpiprogramaren kontzeptuak kodea ez errepikatzea dakar. Horretarako faktoriala ematen digun sententzien multzoa etiketatu eta toki bakar batean idatziko dugu, gero zerbitzu hori behar den unean eskatu (deitu) ahal izateko. Honelako bidea aukeratzean mailaketa[3] bat agertzen da azpiprogramaren kontzeptuak eragiten duena.

Jarraian lehenago egin den `ZenbakiKonbinatorioa` izeneko programa berbera baina azpiprograma batez grafikoki planteaturik erakusten da. `ZenbakiKonbinatorioaAzpPrg` izeneko programak darabilen azpiprograma eskuin aldean idatzi da, eta programa nagusia (azpipromaren erabilpena egiten duen kodea) ezker aldean idatzirik dago, marren estiloak kode guztia fitxategi bakar batean joango litzatekeela adierazten du.

Hauxe litzateke eskema:

[2] Non $m > n$ den.

[3] Azpiprogramaren maila, eta azpiprograma deitu ondoren bere eginkizunaz baliatuko den programaren maila (azken honi programa nagusia esango diogu hemendik aurrera).



Azpiprograma

Programa nagusia

`ZenbakiKonbinatorioaAzpPrg` programak ez du Turbo Pascal lengoaiaren sintaxia erabat zaintzen, baina azpiprogramak erabiltzeko abantaila azaltzeko balio du. Hots, errepikatu beharko litzatekeen `Etiketa()` kodea[4] behin bakarrik idaztea eta nahi den bestetan deitzea eta bere zerbitzuaz baliatzea. Adibidean dei bakoitzeko zenbaki bat jarri dugu, eta ikus daitekeenez edozein deitan zer zenbakiren faktoriala kalkulatu nahi den datu bezala adierazi behar zaio azpiprogramari.

6.2.2 Programaren antolaketa lortzea

Kodearen errepikapena ekiditea interesgarria izanik, azpiprogramak idazteak beste eragin garrantziago bat dakar: programa bera antolatzea. Hau da, programa bat eginkizun zehatzetan banatzea posiblea balitz eta eginkizun bakoitzeko azpiprograma bat idatziko balitz, orduan programaren antolatua suertatzen da ondoko abantailak dituelarik:

- Programa bakarra eta monolitikoa izan beharrean, moduluz osaturik egongo da
- Programaren erroreak bilatzea errazagoa izango da, errorea zer modulutan agertzen den identifikatzea posible delako
- Programa garatzerakoan programadore bakar batek egin beharrean, programadore talde batek egin dezake
- Programaren mantenua errazten da azpiprogramak erabili izan badira. Azken finean programa baten aldaketak programa bera ulertzea eskatzen du, programa bakarra eta txikia denean posible litzateke baina programa handituz doan heinean ulergaitza bihurtzen da. Ondorioz, programa modularra izatean aldaketak eta hobekuntzak egitea askoz errazagoa izango da

[4] Laugarren kapituluko **4.4 PROGRAMA BATEN EGITURA** izenburua duen puntua gogoratuz, ikus daiteke `Etiketa` azpiprogramak Turbo Pascal programa batek duen egitura berbera duela: goiburukoa - erazagupenak - programa nagusia.

Hona hemen `ZenbakiKonbinatorioa3AzpPrg` programari ezagutzen zaion barne antolaketa, programa hau lehenago aztertu den hemen `ZenbakiKonbinatorioaAzpPrg` programan oinarritzen da baina beste bi azpiprograma gehitu ditugu (bat datuak lortzeko, eta bestea emaitza bistaratzeko):

```
AZPI_P Etiketa1 (VAR ZenbM,ZenbN: ) ;
BEGIN
   Write ('m eman: ') ;
   ReadLn (ZenbM) ;
   Write ('n eman: ') ;
   ReadLn (ZenbM) ;
END;
```

```
AZPI_P Etiketa2 (Muga: ) : ;
VAR
  Kont : Integer  ;
  Fakt : LongInt ;
BEGIN
   Fakt := 1 ;
   FOR Kont:=1 TO Muga DO
       Fakt := Fakt * Kont ;

   Etiketa2 := Fakt  ;
END;
```

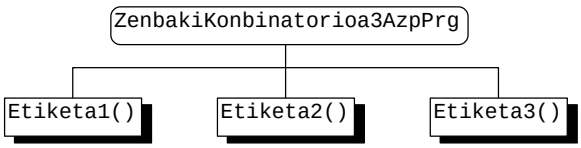
```
AZPI_P Etiketa3 (FktM, FktN, FktM_N: ) ;
VAR
  Ema : LongInt ;
BEGIN
 Ema := FktM DIV (FktN * FktM_N) ;
 WriteLn ('Zenb. konb. = ', Ema ) ;
END;
```

```
PROGRAM ZenbakiKonbinatorioa3AzpPrg ;
VAR
   ZbkM, ZbkN, ZbkM_N : Integer ;
   FaktM, FaktN, FaktM_N : LongInt ;



BEGIN
    Etiketa1 (ZbkM, ZbkN) ;

    FaktM := Etiketa2 (ZbkM) ;

    FaktN := Etiketa2 (ZbkN) ;

    ZbkM_N := ZBKM - ZbkN ;
    FaktM_N := Etiketa2 (ZbkM_N) ;

    Etiketa3 (FaktM, FaktN, FaktM_N) ;
END.
```

Programa nagusia

← *Azpiprogramak*

`ZenbakiKonbinatorioa3AzpPrg` programa honek ez du Turbo Pascal lengoaiaren sintaxia erabat zaintzen (aurrerago ikasiko dugu), baina orain gauza bi aipatuko ditugu. Batetik programa nagusiko azpiprogramen deiak bi eratakoak izan daitezkeela konturatzen gara (`Etiketa1()` eta `Etiketa3()` moduko deiak eta `Etiketa2()`-ri dagokion deia). Bestetik, programa nagusia laburragoa dela eta programa bera ondoko irudiaz eskematiza daitekeela:



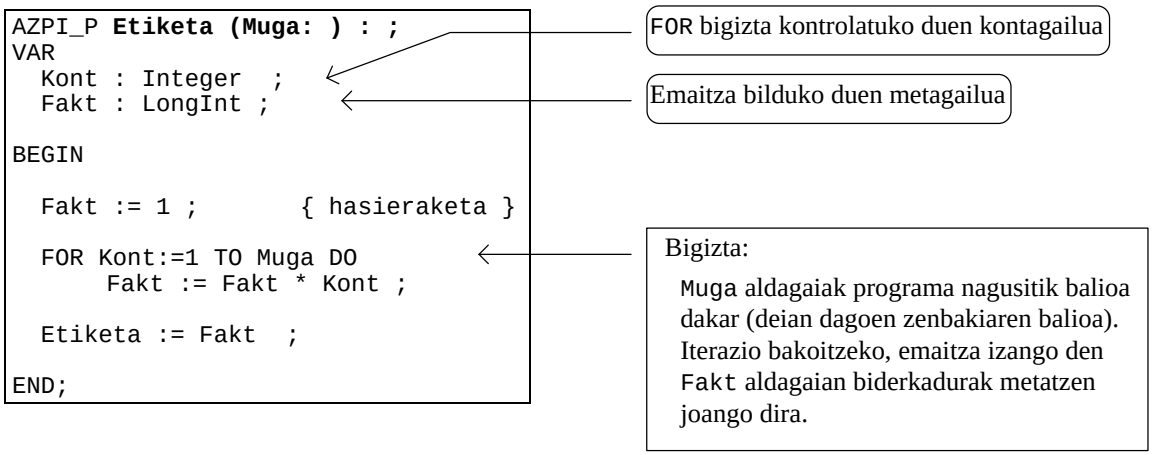
6.2.3 Kodearen independentzia

Programa jakin batean nortasun bereziko eginkizun bat identifikatzen denean azpiprograma bezala kode daiteke (adibidez faktoriala kalkulatzearena), horrela programa modulutan zatituz. Argi dago faktoriala lortzen duen modulua programa nagusitik deitua dela eta nolabaiteko lotura[5] dutela elkar, baina berdin esan daiteke aurkakoa ere. Hots, faktoriala kalkulatzen duen kode zatia inolako aldaketarik egin gabe beste edozein programatan erabil daitekeela.

Honetaz gehiago sakontzeko asmoz `ZenbakiKonbinatorioaAzpPrg` programari beste begirada bat eman diezaiogun, ikusten denez `Etiketa()` izeneko moduluaren azken

[5] Izan ere faktoriala kalkulatzeko zenbakia programa nagusian ezagutzen da eta azpiprogramara datu bezala transferitu egiten da.

helburua zenbait biderkaketaren metatua lortu eta emaitza programa nagusira itzultzea da. Momentuz programa nagusira emaitzak nola itzultzen diren ez dugu aipatuko, horregatik helburuaren lehenengo zatiari buruz arituko gara, hots kalkulua lortzeko egiten dena. Hona hemen `Etiketa()` deritzon azpiprograma:



Ikus daitekeenez, `Etiketa()` azpiprogramak datu bat eta aldagai bi beharrezkoak ditu, datua `Muga` aldagaian[6] gorderiko kopuru osoa litzateke, eta aldagai laguntzaileak berriz `Kont` eta `Fakt` lirateke. `Kont` eta `Fakt` aldagaiak azpiprogramarenak dira (`Etiketa()` azpiprogramaren aldagai pribatuak dira), programa nagusiak ezagutzen ez dituena.

Hortan datza hain zuzen ere kodearen independentzia, azpiprograma batean behar izan daitezkeen aldagai eta gainerako elementuak defini daitezkeela eta denak pribatuak direnez, ezin izango dira atzitu beste azpiprograma edo programa nagusitik. Honek izugarrizko abantaila suposatzen du, izan ere ez dugu aldagaien izendatzaileak asmatzen ibili beharko, azpiprograma ezberdinetan aldagaien identifikadore berberak erabiltzeko gaitasuna izango dugu euren arteko talkarik ez baita egongo.

6.3 AZPIPROGRAMEEN ARTEKO KOMUNIKAZIOA

Programa nagusia eta azpiprogrameen arteko komunikaziorako bi elementu kontutan izango dira, batetik azpiprograma martxan jartzeko *deia* egin behar dela eta bestetik informazio trukaketa *parametro*en bidez egiten dela. Elementu biak elkar erlazionaturik egoten dira gehienetan, baina ez beti. Esate baterako pantaila garbitzen duen `ClrScr` prozedura estandarraren deiak parametrorik ez du behar.

6.3.1 Azpiprogramaren deia

Azter dezagun azpiprograma deitzeko zer egin behar den. Horretarako, ezer baino lehen deiaren helburua (**6.3.1.1** puntua) ikus dezagun, ondoren **6.3.1.2 Parametroen ordena azpiprogramaren deian** gaia jorratuko dugu.

[6] Aldagaia baino, programa nagusitik emaniko datuari parametro esaten zaio (**6.3 AZPIPROGRAMEEN ARTEKO KOMUNIKAZIOA** puntuan ikusiko dugu).

6.3.1.1 Deiaren Helburua

`ZenbakiKonbinatorioaAzpPrg` programa gogora ekarriz, esan berri daukagu programa nagusia eta `Etiketa()` izeneko moduluaren arteko harremana datua den zenbaki oso baten bitartez gauzatzen dela. Kopuru oso hori programa nagusian balio ezberdinak hartzen ditu dei bakoitzaren arabera (`ZnbkM` lehenengo deian, `ZnbkN` bigarrenean eta `ZnbkM_N` hirugarrenean), deietan kanpotik datorkion kopuru osoa ezberdinak izan arren azpiprograma barruko identifikadorea beti `Muga` da.

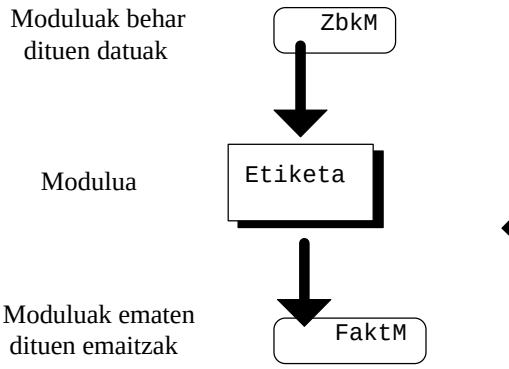
`Etiketa()` izeneko moduluari begiratzen badiogu honako hau esan dezakegu: azpiprograma batek, programa batek bezala, emaitzak lortzeko asmoz datuen aldaketa bat egiten du, baina datuak teklatutik irakurri beharrean kanpotik heltzen zaizkio, eta emaitzak pantailara igorri beharrean programa nagusira bidaltzen ditu.

Beraz, azpiprograma bat erabiltzeko gauza bi ezagutzea derrigorrezkoa da:

1. Azpiprogramari dagokion identifikadorea edo izena, zeinek modulua aktibatzeko balio duen
2. Azpiprogramak zer nolako datuak hartzen dituen eta zer nolako emaitza itzultzen duen

Kontzeptu bi horiek jakitearekin aski da modulua erabili ahal izateko, eta kontura gaitezan bien artean deia zehazten dutela.

Lehenego kontzeptuak programak erabilgarri izan ditzakeen modulu guztien artean interesatzen zaiguna zehazten du, eta moduluak *"zer egiten du?"* galderari erantzuten dio. Bigarren kontzeptuarekin programa nagusia eta azpiprogramaren arteko informazio trukaketa zehazten da (moduluak behar dituen datuak eta itzultzen dituen emaitzak), kontzeptu honek modulua *"nola erabilzen da?"* galderari erantzuten dio. Eskematikoki:




```
PROGRAM ZenbakiKonbinatorioaAzpPrg ;
VAR
   ZbkM, ZbkN, ZbkM_N : Integer ;
   Emaitza, FaktM, FaktN, FaktM_N : LongInt ;
BEGIN
    Write ('m eman: ') ;
    ReadLn (ZbkM) ;
    Write ('n eman: ') ;
    ReadLn (ZbkN) ;

    FaktM := Etiketa (ZbkM) ;

    FaktN := Etiketa (ZbkN) ;

    ZbkM_N := ZBKM - ZbkN ;
    FaktM_N := Etiketa (ZbkM_N) ;

    Emaitza := FaktM DIV (FaktN * FaktM_N) ;
    WriteLn ('Zenbaki konb. = ', Emaitza ) ;
END.
```

Laburbilduz, azpiprograma bat erabiliko duen programadoreak gauza bi jakin behar ditu: moduluaren izena[7] eta zer egiten duen. Zehatzago esanik azpiprogramari dagokion identifikadorea eta dituen parametroak.

[7] Azpiprograma baterako aukeratuko dugun identifikadorea ez da inolaz ere `Etiketa` izango, askoz esanguratsoagoa baizik.

Baina bada azpiprogramaren beste ikuspegi bat, modulua erabili ezezik modulua kodetu behar duen programadorearen ikuspegia, aurreko **6.2.3 Kodearen independentzia** puntuan azaldu duguna hain zuzen ere. Kasu honetan, moduluak *"zer egiten du?"* eta modulua *"nola erabiltzen da?"* galderei beste hirugarren bat gehitzen zaie: moduluak *"bere zeregina nola egiten du?"*.

Kapitulu hau ikasten hasi aurretik lehenengo bi galderak planteaitzen dituzten zailtasunak gainditzen ikasi dugu, adibiderako `WriteLn()` eta `Cos()` azpiprogramak ezagunen kasuak; izan ere **4.2.3 Konstanteak** puntuan idatzi zen lehenengo Turbo Pascal programa agindu bakar batez osatzen zen (pantailaraketa egiten zuen azpiprograma baten honako dei hau: `WriteLn ('kaixo !') ;`). Hemendik aurrera norberaren azpiprogramak garatzen hasiko garenez hirugarren galderarekin topo egingo dugu.

6.3.1.2 Parametroen ordena azpiprogramaren deian

`ZenbakiKonbinatorioa3AzpPrg` programarekin jarraituz, dituen hiru moduluetatik hirugarrena, `Etiketa3()` delakoa, aztertuko dugu jarraian.

```
AZPI_P Etiketa3 (FktM, FktN, FktM_N: ) ;
VAR
   Ema : LongInt ;
BEGIN
   Ema := FktM DIV (FktN * FktM_N) ;
   WriteLn ('Zenb. konb. = ', Ema ) ;
   FktM := 79 ;
END;
```

Azpiprograma

```
VAR
   FaktM, FaktN, FaktM_N : LongInt ;


   { deia baino lehen }

   Etiketa3 (FaktM, FaktN, FaktM_N) ;

   { deiaren ondoren }
```

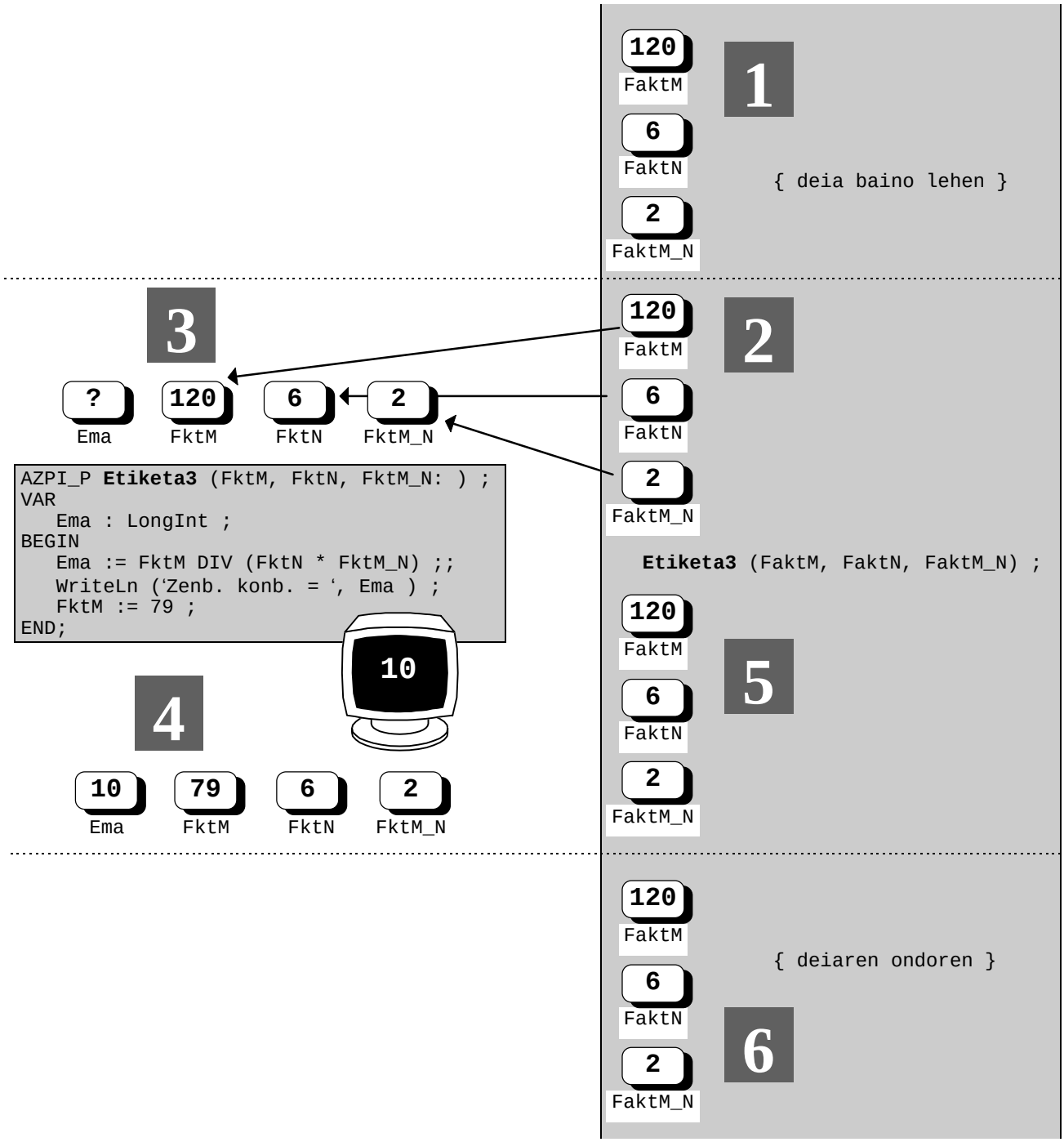
Programa nagusiaren zatia

`Etiketa3()` moduluaren aginduak ikusirik bere helburua erraz asma daiteke, lehenago kalkulaturiko `FaktM`, `FaktN` eta `FaktM_N` programa nagusiko aldagaien balioak `Etiketa3()` azpiprogramara bidaltzen dira eta bertan eragiketa aritmetikoak egin ondoren zenbaki konbinatorioari dagokion emaitza pantailaratzen da. Adibidez, hurrengo orrialdearen eskeman `ZbkM` eta `ZbkN` aldagaietan `5` eta `3` gorde izan direla suposatuz, `Etiketa3()` modulua nola dabilen erakusten da:

1 `Etiketa2()` moduluaren bitartez `5` eta `3` faktorialen kalkuluak egiten direlarik `FaktM` eta `FaktN` aldagaien balioak ezagunak dira, modu beretsuan `FaktM_N` zehazteko `(5-3)!` eragiketa burutu izan da. Beraz, programa nagusiko `ZbkM` eta `ZbkN` aldagaiek dituzten balioak `5` eta `3` izanik `FaktM`, `FaktN` eta `FaktM_N` aldagaietan `120`, `6` eta `2` egongo lirateke. Horregatik `Etiketa3()` moduluaren deia baino lehen, adibide honetan, `FaktM`, `FaktN` eta `FaktM_N` aldagaiei dagozkien memori kutxa bakoitzean `120`, `6` eta `2` idatzi dugu.

2 Aurreko puntuan esandakoaren arabera `Etiketa3()` prozedura deitu aurretik programa nagusiko `FaktM`, `FaktN` eta `FaktM_N` aldagaiek dituzten balioak `120`, `6` eta `2` izango dira.

3 `Etiketa3()` prozeduraren deia gertatzen denean programa nagusiko `FaktM`, `FaktN` eta `FaktM_N` aldagaien eta azpiprogramaren `FktM`, `FktN` eta `FktM_N` parametroen artean elkarketa bat suertatzen da. Horren arabera



`Etiketa3()` deiaren parentesi barruan dagoen lehenengo aldagaiaren balioa azpiprogramaren lehenengo parametroari transferitzen zaio, bigarren aldagaiarena azpiprogramaren bigarrenari, eta, hirugarren aldagaiarena azpiprogramaren hirugarren parametroari.

Ondorioz, programaren kontrola hartuz `Etiketa3()` delako prozedura exekutatzen hasten denenan `FktM`, `FktN` eta `FktM_N` parametroei dagozkien balioak programa nagusitik orden honetan erantsi zaizkien `120`, `6` eta `2` izango dira. Baina `Etiketa3()` moduluaren aldagai pribatua den `Ema`-k balio ezagunik ez du izango, horregatik eta dagokion memori kutxan edozer gauza egon daitekeenez galdera ikur bat idatzi dugu.

4 `Etiketa3()` prozedurak dituen hiru sententzien eraginak hauek dira: `Ema` aldagaiak `10` balio zehatza jasoko du, emaitza den balio hori pantailaratzen da, eta azkenik, `FktM`-ri `79` konstantea esleitzen zaio.

Beraz, `Etiketa3()` azpiprogramaren amaieran dagoen egoera aurreko orrialdeko irudian adierazi da, monitorean `10` emaitza agertuz eta `Etiketa3()` moduluak darabilzkien memori kutxak marraztuz. Kontutan izan `FktM`-ren edukia aldatu izan arren, horrek eraginik ez duela programa nagusiko `FaktM` aldagaiaren gainean.

5 Izan ere, `Etiketa3()` moduluak kontrola programa nagusiari itzultzen dionean, bere aldagaiek gordetzen dutena azpiprograma deitu aurreko balio berberak izango dira. Izan ere, `FaktM`, `FaktN`, `FaktM_N` eta `FktM`, `FktN`, `FktM_N` dituzten memori posizioak ezberdinak izanik azpiprograma barruan egindako aldaketek ez dute programa nagusiarengan eraginik izango.

6 Horregatik `Etiketa3()` izeneko prozeduraren deiaren ondoren programa nagusiko `FaktM FaktN FaktM_N` hiru aldagaien edukiak hasierakoak izango dira.

Puntu honetan esandakoa baieztapen honen bidez laburbiltzen da: modulu baten deian parametroen posizioek erabateko garrantzia dute, eta gertatzen den lotura ordenez suertatzen denez bikote bakoitzaren datu-motak berdinak, ala behintzat koherenteak, izango direla.

6.3.2 Parametro motak

Aurreko `ZenbakiKonbinatorioa3AzpPrg` delako programaren `Etiketa2()` izeneko moduluari begiratzen badiogu arestian esandakoa errepika dezakegu: azpiprograma batek datuen aldaketa bat egiten du emaitzaren bat lortzeko, baina `Muga` datua teklatutik irakurri beharrean programa nagusitik heltzen zaio, eta emaitza pantailaratua izan dadin programa nagusira bidaltzen du (zeinek `Etiketa3()` moduluari pasatuko dion, pantailaraketa prozedura horrek egiten baitu).

Baina `ZenbakiKonbinatorioa3AzpPrg` programaren moduluen izenak esanguratsuagoak izan daitezen alda ditzagun. Moduluen goiburukoak ere ezberdindu ditugu batetik `AZPI_P` eta bestetik `AZPI_F` jarri ditugu[8]. `ZenbakiKonbinatorioa3AzpPrg` programan, modulu bakoitzeko, dauden parametroak honako taula honetan biltzen dira:

Azpiprograma		**Parametroa**		
Identifikadorea	**Mota**	**Izena**	**Datu-mota**	**Jokaera**
`Sarrerak`	Prozedura	`ZbkM` `ZbkN`	`Integer` `Integer`	irteerakoa irteerakoa
`Faktoriala`	Funtzioa	`Muga`	`Integer`	sarrerakoa

`Irteera`	Prozedura	`FktM`
		`FktN`
		`FktM_N`
			`LongInt`
			`LongInt`
			`LongInt`
				sarrerakoa
				sarrerakoa
				sarrerakoa

[8] Turbo Pascal lengoaian azpiprogramak **p**rozedurak edo **f**untzioak izan daitezke (ikus **6.4 AZPIPROGRAMA MOTAK TURBO PASCAL LENGOAIAN** deituriko puntua).

Hauxe litzateke `ZenbakiKonbinatorioa3AzpPrg` programa berritua:

```
AZPI_P Sarrerak (VAR ZbkM,ZbkN: ) ;
BEGIN
   Write ('m eman: ') ;
   ReadLn (ZbkM) ;
   Write ('m eman: ') ;
   ReadLn (ZbkM) ;
END;
```

```
AZPI_F Faktoriala (Muga: ) : ;
VAR
  Kont : Integer  ;
  Fakt : LongInt ;
BEGIN
   Fakt := 1 ;
   FOR Kont:=1 TO Muga DO
      Fakt := Fakt * Kont ;

   Faktoriala := Fakt  ;
END;
```

```
AZPI_P Irteera (FktM, FktN, FktM_N: ) ;
VAR
  Ema : LongInt ;
BEGIN
 Ema := FktM DIV (FktN * FktM_N) ;
 WriteLn ('Zenb. konb. = ', Ema ) ;
END;
```

```
PROGRAM ZenbakiKonbinatorioa3AzpPrg ;
VAR
   ZbkM, ZbkN, ZbkM_N : Integer ;
   FaktM, FaktN, FaktM_N : LongInt ;



BEGIN
    Sarrerak (ZbkM, ZbkN) ;

    FaktM := Faktoriala (ZbkM) ;

    FaktN := Faktoriala (ZbkN) ;

    ZbkM_N := ZBKM - ZbkN ;
    FaktM_N := Faktoriala (ZbkM_N) ;

    Irteera (FaktM, FaktN, FaktM_N) ;
END.
```

Programa nagusia

← *Azpiprogramak*

Azpiprogramak hiru dira eta izendatzeko asmatu diren izenak taularen lehenengo zutabean jarri dira `Sarrerak`, `Faktoriala` eta `Irteera`, azpiprogramen mota bigarren zutabean zehazturik dago (bi prozedura eta funtzio bat), Pascal lengoaiak `PROCEDURE` eta `FUNCTION` hitz erreserbatuak ezagutzen ditu horiek ezberdintzeko. Azpiprograma definitzeko balio duen goiburukoan elementu bi horiek agertuko dira:

```
AZPIPROGRAMA_MOTA Identifikadorea ( _________________ ) ;
```

Parentesien artean parametroak agertuko dira. Adibide bera aintzat harturik, parametroen ezaugarriak hiru zutabetan bildu dira: batetik parametroaren izendatzailea, bestetik parametroari dagokion datu-mota eta azkenik parametroak duen portaera (orokorrean hiru portaeretatik bat izan dezake parametro jakin batek *irteerakoa*, *sarrerakoa* eta *irteera-sarrerakoa*).

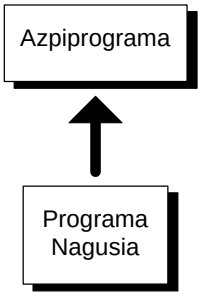
Parametroen jokaerak banan banan aztertuko ditugu hurrengo **6.2.3.1-6.2.3.2-6.2.3.3** ataletan.

6.3.2.1 Sarrerako parametroak

Azpiprograma batek, dagokion zeregina betetzeko, behar dituen datuei sarrerako parametroak deritze. Sarrerako parametroak, programa nagusiak azpiprogramari komunikatzen dion informazioa litzateke, eta euren balioak azpiprograma deitzean zehazturik egongo dira programa nagusian.

Azpiprogramak sarrerako parametroen balioak erabil eta alda ditzake, baina aldaketak ez dira programa nagusiari itzultzen.

Komunikazioa zentzu bakar batean ematen da. Ikus irudia:



Demagun azpiprogramaren helburua bete ahal izateko daturen bat behar dela, informazio hori programa nagusiak ezagutzen duenez azpiprogramari *sarrerako parametro* bezala pasatuko dio deia egiten denean.

Azpiprograma parametroaren bidez heldu zaion datuaz baliatuko da, dagokion lana betetzeko. Baldin eta parametroaren balioa azpiprograman aldatzen bada programa nagusiarengan eraginik ez du izango. Sarrerako parametroetan komunikazioaren zentzua bakarra da (deitzen duen modulutik deitua den modulurakoa).

6.3.2.1.1 Adibideak

Adibide bezala `SarrerakoParametroa1` programa ikus dezagun. Programa honek bi modulu ditu `GidoienLerroaIdatzi` eta `TestuBatIdatzi` deiturikoak, biak prozedurak dira eta argi dagoenez ez batak ez besteak parametrorik (informaziorik) ez dute behar. Izan ere euren zereginak uneoro konstanteak dira, eta horregatik programa nagusiak ez du zertan daturik igorri behar deiak suertatzen direnean.

```
PROGRAM SarrerakoParametroa1 ;                { \TP70\06\PARAM1S.PAS }
USES
   Crt ;

PROCEDURE GidoienLerroaIdatzi ;
VAR
  Kont : Integer ;
BEGIN
    FOR Kont:= 1 TO 12 DO
       Write ('=') ;
    WriteLn ;
END;

PROCEDURE TestuBatIdatzi ;
BEGIN
    WriteLn ('=  Kaixo!  =') ;
END;

BEGIN
    ClrScr ;                  (* Programa Nagusia *)
    GidoienLerroaIdatzi ;
    TestuBatIdatzi ;
    GidoienLerroaIdatzi ;
END.
```

`SarrerakoParametroa1` programaren exekuzioaren hasieran `ClrScr` prozedura estandarra aktibatzen da (ondorioz pantaila garbituko da, eta kurtsorea monitorearen goiko ezker erpinan kokatuko da). Ondoren `GidoienLerroaIdatzi` prozedura pizten da, zeinek `Kont` aldagai laguntzaile baten bitartez 12 gidoi elkar jarraian idatziko dituen, amaitzean `WriteLn()` prozedura estandarren bitartez lerro jauzi bat egiten da. Programa nagusiaren jarraituz, gidoiak eta lerro aldaketa egin ondoren `TestuBatIdatzi` deituriko moduluaren txanda dator, honek 12 karakteredun testu bat idazten du `WriteLn()` prozedura estandarrari testua parametro bezala pasatuz. Programa nagusia bukatu aurretik `GidoienLerroaIdatzi` prozedura berriro deitzen denez, beste 12 gidoi eta lerro jauzia lortuko dira pantailan.

Hona hemen, `SarrerakoParametroa1` programaren edozein exekuziotan suertatuko den irteera. Programa nagusiak azpiprogramei daturik ez dionez bidaltzen, eta azpiprograma horien barnean daturik ez denez irakurtzen, irteera edonoiz hau izango da:

```
============
=  Kaixo!  =
============
_
```

Datuak behar ez dituen azpiprogramen adibideekin jarraituz, `SarrerakoParametroa2` programaren exekuzioa azalduko dugu. Programa ulertzeko gogoratu laugarren kapituluko **4.2.1 Hitz erreserbatuak eta sinboloak** puntuan karaktere batekin lan egiteko bere ASCII kodea erabil daitekeela aipatu zela, horretarako # sinboloa jarri behar zaio aurretik ASCII kodeari.

ASCII taularen zazpigarren karakteera ordenadorearen txistua dela jakinik, zarata hori lortzeko sententzia hau aski da: `Write (#7) ;`

```
PROGRAM SarrerakoParametroa2 ;                { \TP70\06\PARAM2S.PAS }
USES
   Crt ;

PROCEDURE Txistuak ;
BEGIN
  Write (#7) ;     (* ASCII taularen zazpigarren karakterea txistua da *)
  Delay (2000) ;   (* 2000 milisegundoz { 2 seg. } exekuzioa eteten da *)
  Write (#7) ;
END;

VAR
  MezuaIrakurri : Char ;
BEGIN
  ClrScr ;                  (* Programa Nagusia *)
  Txistuak ;
  Write ('Txistu biak berriro entzuteko tekla bat sakatu. ') ;
  MezuaIrakurri := ReadKey ;
  Txistuak ;
END.
```

`SarrerakoParametroa1` eta `SarrerakoParametroa2` programa biak berdintsuak dira, dituzten modulutan sarrerarik egon ez arren eginkizunak betetzen dituzteelako. Hau litzateke prozeduran ezaugarri bat funtzioek ez dutena: prozedurek, horrela behar izanez gero, sarrera edo irteera parametrorik gabe lan egin dezakete (funtzioek berriz, irteerako emaitza itzultzen diote beti programa nagusiari).

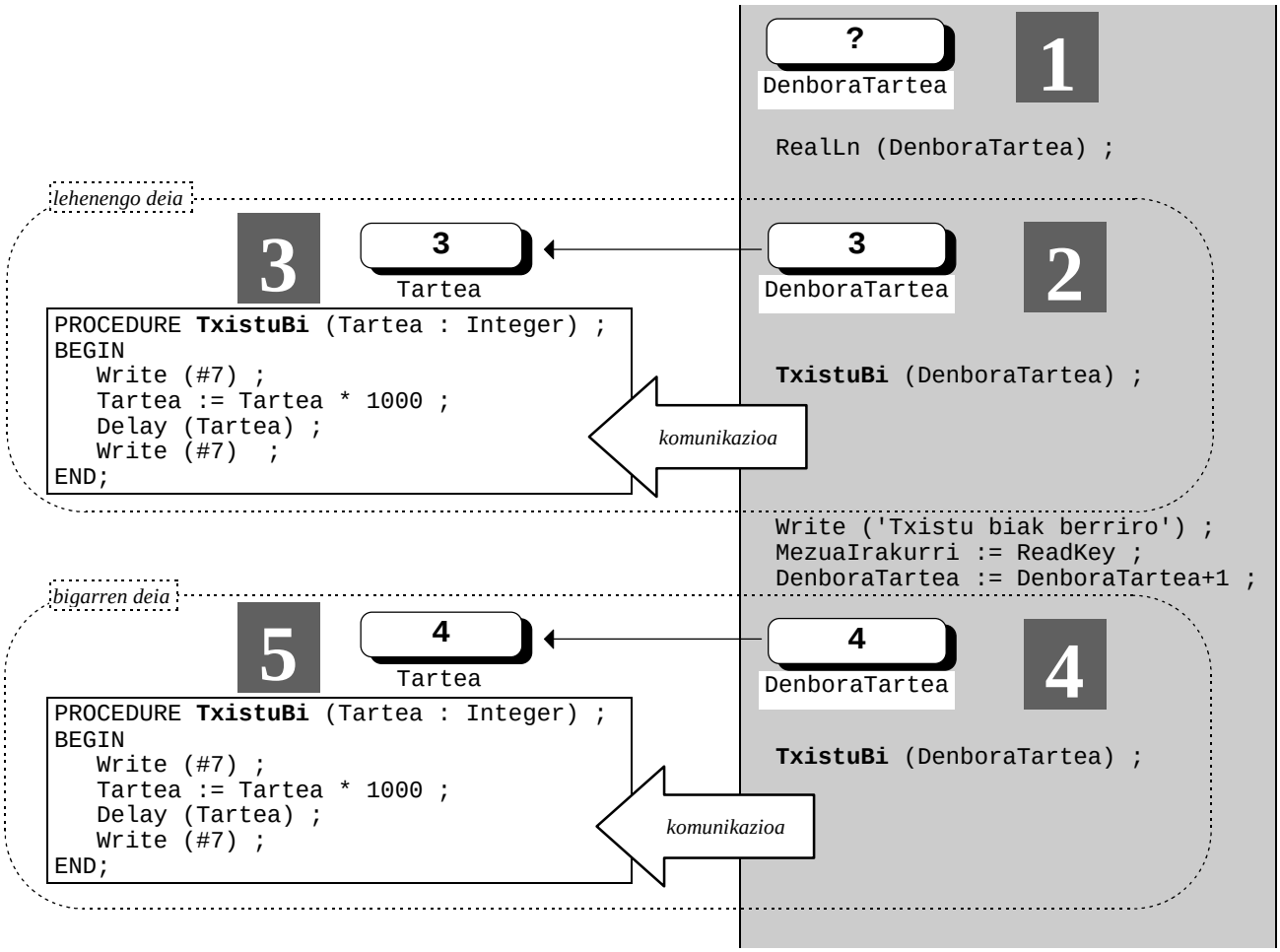
Hurrengo adibidean programa nagusitik datuak bidaliko dira azpiprogramari. Oraintsu erabilitako `Txistuak` prozedura aldatu egin da, txistu biren artean dagoen denbora tartea derrigorrez bi segundotakoa izan beharrean aukeragarria izan dadin. Programa nagusian ezagutzen den `DenboraTartea` aldagaiaren bitartez etenaren luzera zehazten da, baina programa geldi arazten duen `Delay()` prozedura estandarra `TxistuBi()` moduluan aurkitzen denez, `DenboraTartea` aldagaiak gordetzen duen datua igorri behar zaio:

```
PROGRAM SarrerakoParametroa3 ;                { \TP70\06\PARAM3S.PAS }
USES
   Crt ;

PROCEDURE TxistuBi (Tartea : Integer) ;
BEGIN
  Write (#7) ;
  Tartea := Tartea * 1000 ;    (* Tartea milisegundotara iragan *)
  Delay (Tartea) ;
  Write (#7) ;      (* ASCII taularen zazpigarren karakterea txistua da *)
END;
```

```
VAR
  MezuaIrakurri : Char ;
  DenboraTartea : Integer ;

BEGIN
  ClrScr ;                    (* Programa Nagusia *)
  REPEAT
    Write ('Txistuen arteko denbora tartea segundotan (1-5): ') ;
    ReadLn (DenboraTartea) ;
  UNTIL (DenboraTartea >= 1) AND (DenboraTartea <= 5) ;
  TxistuBi (DenboraTartea) ;
  Write ('Txistu biak berriro entzuteko tekla bat sakatu. ') ;
  MezuaIrakurri := ReadKey ;
  DenboraTartea := DenboraTartea + 1 ;
  TxistuBi (DenboraTartea) ;
END.
```

`SarrerakoParametroa3` izeneko programa hau `SarrerakoParametroa2` programa baino malguagoa da, izan ere txistu biren arteko denbora tartea konpilazio denboran (programa garatzaileak) hautatu beharrean, exekuzio denboran (programaren erabiltzaileak) aukeratzen da.

`TxistuBi()` azpiprogramak `Tartea` parametroaren bidez heldu zaion datuaz baliatuko da, txistu biren arteko denbora etena betetzeko. `Tartea` parametroak programa nagusiko `DenboraTartea` aldagaiaren balioa hartzen du, horretarako kontutan izan biak `Integer` erazagutu izan direla eta parentesi barneko `Tartea` parametroaren aurrean ezer ez dela idazten:



1 Programa nagusiaren hasieran `DenboraTartea` aldagaiak edozer balio izan dezakeenez, berari dagokion memori kutxan galdera ikur bat idatzi dugu. Teklatuaren bitartez `1` eta `5` arteko kopuru oso bat irakurri ondoren, adibidez `3`, bigarren urratsera joan gaitezke.

2 `TxistuBi()` prozeduraren lehenengo deia. Modulu honek parametro bat behar du, programa nagusiko `DenboraTartea` azpiprograman `Tartea` izendatu dena. Modulu deitzailearen parametroari *parametro erreal*[9] esaten zaio, eta modulu deituaren parametroari aldiz *parametro formal*[10]. Sarrerako parametroa izatean parametro formalak parametro errealaren balioa jasotzen du, eta komunikazioaren norabideak zentzu bakarra duenez parametro formalaren balioa azpiprograman aldatuko balitz programa nagusiarengan eraginik ez luke izango.

3 `TxistuBi()` lehenengo deian `Tartea` parametro formalak `3` balioko du eta txistu biren arteko itxarote denbora hirumila milisegundotakoa izango da.

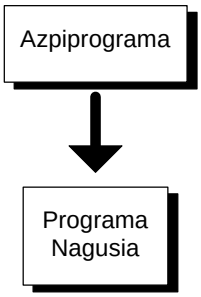
4 `TxistuBi()` prozeduraren bigarren deia. Azpirrutina deitu aurretik `DenboraTartea` aldagaiaren balioa inkrementatu egiten denez parametro erreal horrek `4`ko balioa du eta ...

5 ... `Tartea` parametro formalak jasotzen duena `4` hori izango da, txistu biren arteko itxarote denbora oraingoan lau segundotakoa izanik. `TxistuBi()` prozedura amaitzean, programaren kontrola programa nagusiak hartzen du berriro eta `SarrerakoParametroa3` programa bukatu egiten da.

6.3.2.2 Irteerako parametroak

Azpiprograma batek, modulu deitzaileari balioren bat itzuli behar dionenean irteerako parametro baten bitartez gauzatu behar da. Irteerako parametroak, azpirrutinak programa nagusiari komunikatzen dion informazioa litzateke, eta euren balioak azpiprograma barnean zehaztu egingo dira.

Komunikazioa zentzu bakar batean ematen da. Ikus irudia:



Demagun azpiprogramaren helburua kalkulu bat egitea dela, eta lortzen duen emaitza programa nagusiak behar duela. Beraz, azpiprogramak lan egin dezan *sarrerako parametro* hartzeaz gain, *irteerako parametro*ren bat beharko du.

Irteerako parametro baten balioa azpirrutina barruan aldatuz gero, sarrerako parametroak ez bezala, programa nagusiarengan eragina izango du. Irteerako parametroetan ere komunikazioaren zentzua bakarra da (deitua den modulutik deitzen duen modulurakoa).

[9] Azpirrutina (prozedura edo funtzio) bat abiatzean erabiltzen den benetako aldagaia. Parametro erreal honi *uneko parametro* ere esaten zaio.

[10] Azpirrutina baten goiburukoan agertzen den aldagai generikoa identifikatzeko balio duen sinboloa. Azpiprograma deitzean, aldagai honek beroni dagokion aldagai erreala ordezkatuko du.

6.3.2.2.1 Adibideak

Irteerako parametroen adibide gisa `IrteerakoParametroa1` programa ikus dezagun, zeinek denbora jakin bat segundotan jartzen duen. Programaren lehenengo bertsio honek modulu bakarra du, `SegundoenKalkulua()` deiturikoa, eta bertan sarrerako parametro bi erabiltzen dira (denboraren ordua eta minutua). Kalkulua lortu ondoren, emaitza programa nagusiak behar duenez, informazio horren komunikazioa irteerako parametro baten bitartez burutu da.

```
PROGRAM IrteerakoParametroa1 ;                { \TP70\06\PARAM1I.PAS }

PROCEDURE SegundoenKalkulua (Ord, Min : Byte; VAR Seg : LongInt) ;
VAR
  Metagailu : LongInt ;
BEGIN
  Metagailu := Ord*60 + Min ;    (* Denbora minututara iragan  *)
  Metagailu := Metagailu * 60 ;  (* Denbora segundotara iragan *)

  Seg := Metagailu ;   (* Emaitza programa nagusira itzultzeko *)
END;

VAR
  Orduak, Minutuak : Byte ;
  Segundoak : LongInt ;
BEGIN
  REPEAT                                    (* Programa Nagusia *)
    Write ('Orduak eman (0-23):   ') ;
    ReadLn (Orduak) ;
  UNTIL (Orduak >= 0) AND (Orduak <= 23) ;
  REPEAT
    Write ('Minutuak eman (0-59): ') ;
    ReadLn (Minutuak) ;
  UNTIL (Minutuak >= 0) AND (Minutuak <= 59) ;

  SegundoenKalkulua (Orduak, Minutuak, Segundoak) ;

  WriteLn (Orduak, ':', Minutuak, ' ---> ', Segundoak, ' segundo') ;
END.
```

`IrteerakoParametroa1` programari dagokion programa nagusiari so eginez, ondo bereziturikо hiru atal nabarmentzen dira:

1. Datuen sarrera (`Orduak` eta `Minutuak`)
2. Kalkulu bat (`Segundoak`)
3. `Segundoak` emaitzaren pantailaraketa

Hiru atalak banan-banan azterturik:

`IrteerakoParametroa1` programaren exekuzioaren hasieran denboraren ordua eta minutuak teklatuz irakurtzen dira (zenbaki osoak, positiboak eta txikiak izango direnez `Byte` datu-mota hautatu dugu). Bilatzen ari garen emaitza `Segundoak` aldagaian pilatuko dugu gero pantailaratzeko, segundoen kopurua nahiko handia izan daitekeenez `LongInt` datu-mota aukeratu dugu.

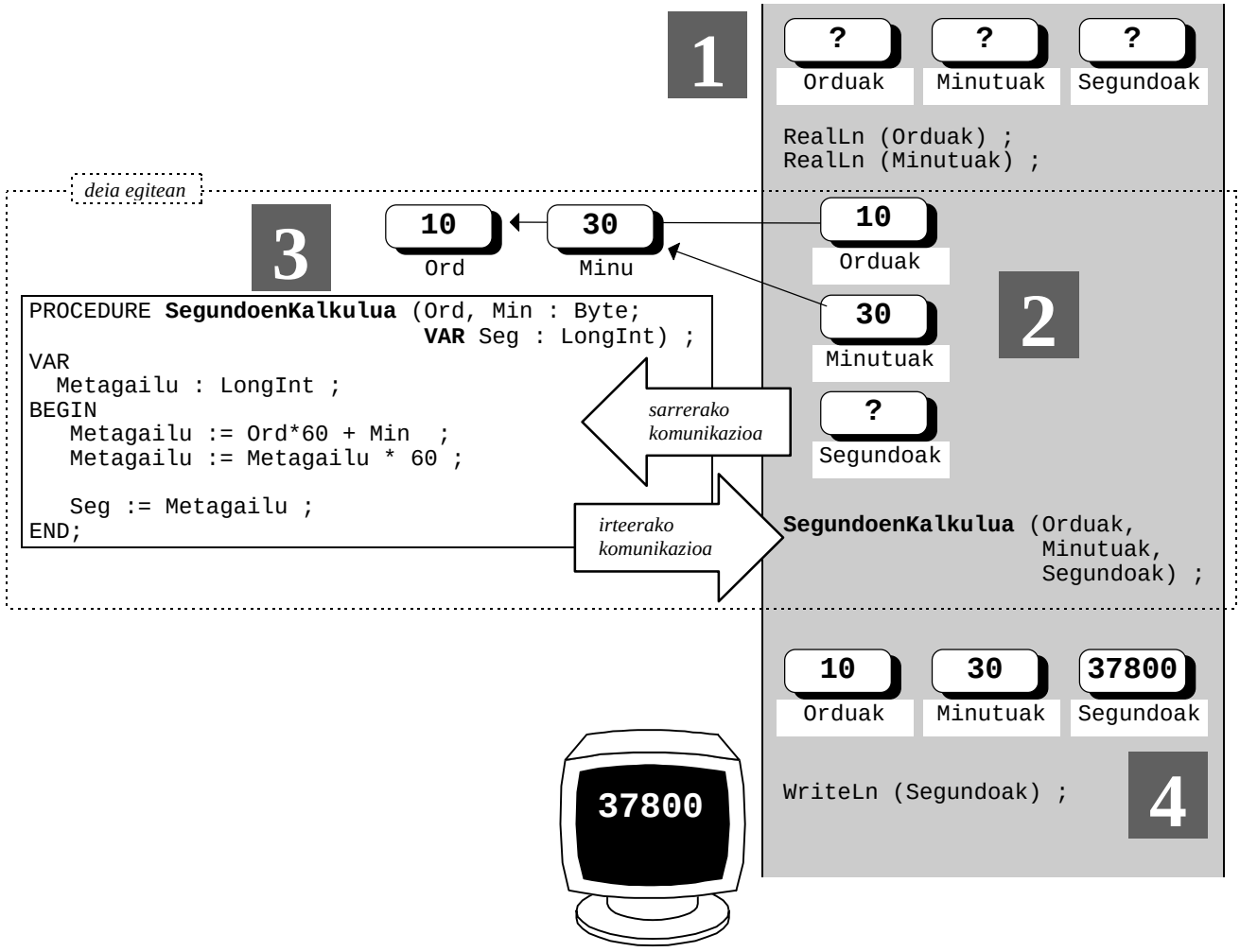
Ondoren `SegundoenKalkulua()` prozedura pizten da; zeinek `Orduak` eta `Minutuak` aldagaiak sarrerako parametro errealak erabiltzen dituen eta `Segundoak` irteerako parametro erreala den. Lehenengo biek `SegundoenKalkulua()` deia egiten denerako balio ezagunak izango dituzte, baina hirugarrenari balioa prozedura barnean ematen zaionez hasieran edozer gauza izan dezake.

`SegundoenKalkulua()` modulutik irtetean, programa nagusiko `Segundoak` aldagaiak emaitza jasotzen du eta `WriteLn()` prozedura estandarren bitartez pantailaratzen da.

Hona hemen, `IrteerakoParametroa1` programa exekutatu ondoren monitorean ager daitekeena:

```
Orduak eman (0-23):   10
Minutuak eman (0-59): 30
10:30 ---> 37800 segundo
_
```

`SegundoenKalkulua()` azpiprograma `Ord` eta `Min` parametroen bidez heltzen zaizkion datuaz baliatzen da, eragiketa matematiko bati esker, `Seg` parametroan emaitza gordetzeko. Argi dago `Metagailu` aldagai laguntzailea soberan egon daitekeela:



1 Programa nagusia hasten denean `Orduak`, `Minutuak` eta `Segundoak` aldagaiak balio ezagunik gabe aurkituko dira. Lehenengo biak, `Orduak` eta `Minutuak`, teklatuaren bitartez irakurtzen dira, adibidez, `10` eta `30` kopuru osoak gordez.

2 `SegundoenKalkulua()` prozeduraren deia gertatzen denean, **6.3.2.1** eta **6.3.2.1.1** puntuetan ikusi den bezala, sarrerako parametro errealak diren `Orduak` eta `Minutuak` dituzten balioak `Ord` eta `Min` parametro formaletara transmititzen dira. Aurrekoekin alderatuz `Segundoak` izeneko aldagaiaren jokaera ezberdina da (ikus berari dagokion `Seg` parametro formalaren aurrean `VAR` hitz erreserbatua dagoela).

3 Izan ere, `SegundoenKalkulua()` prozedurak duen goiburukoan hiru parametro behar direla adierazten da, areago hirutatik lehenengo biak sarrerakoak eta hirugarrena irteerakoa direla zehazten da. Sarrerako parametro formaletan parametro errealen balioak kopiatzen dira, eta irteerakoan gordetzen dena **6.3.2.2 Irteerako-parametroak** puntun ikusi izan dugu.

Dagoenekoz irteerako `Seg` parametro formala azpirrutina barnean aldatuz gero, berari dagokion `Segundo` parametro erreala aldatu egingo dela onar dezagun.

4 `SegundoenKalkulua()` prozeduraren exekuzioa amaitzean kontrola programa nagusira itzultzen da eta `Segundo` parametro errealak `Seg`-ek izan duena balioko du. Hartutako adibiderako `Segundo` aldagaiak `37800` kopurua gordeko du (`37800 = 60*60*10 + 60*30`) eta hori pantailan idatzi ahal izateko `WriteLn()` prozedura estandarra erabiltzen da.

Irteerako parametroen bigarren adibiderako `IrteerakoParametroa1` programa alda dezagun, programa beraren bigarren bertsioak hiru modulu izango ditu: ezagutzen dugun `SegundoenKalkulua()` deiturikoa eta datuen irakurketaz arduratzen diren `OrduIrakurketa()` eta `MinutuIrakurketa()` prozedurak.

Hona hemen `IrteerakoParametroa2` programa:

```
PROGRAM IrteerakoParametroa2 ;                { \TP70\06\PARAM2I.PAS }

PROCEDURE OrduIrakurketa (VAR Ord : Byte) ;
BEGIN
  REPEAT
    Write ('Orduak eman (0-23):   ') ;
    ReadLn (Ord) ;
  UNTIL (Ord >= 0) AND (Ord <= 23) ;
END;

PROCEDURE MinutuIrakurketa (VAR Min : Byte) ;
BEGIN
  REPEAT
    Write ('Minutuak eman (0-59): ') ;
    ReadLn (Min) ;
  UNTIL (Min >= 0) AND (Min <= 59) ;
END;

PROCEDURE SegundoenKalkulua (Ord, Min : Byte; VAR Seg : LongInt) ;
VAR
  Metagailu : LongInt ;
BEGIN
  Metagailu := Ord*60 + Min ;    (* Denbora minututara iragan *)
  Metagailu := Metagailu * 60 ;  (* Denbora segundotara iragan *)

  Seg := Metagailu ;   (* Emaitza programa nagusira itzultzeko *)
END;

VAR
  Orduak, Minutuak : Byte ;
  Segundoak : LongInt ;
BEGIN                           (* Programa Nagusia *)
  OrduIrakurketa (Orduak) ;
  MinutuIrakurketa (Minutuak) ;
  SegundoenKalkulua (Orduak, Minutuak, Segundoak) ;
  WriteLn (Orduak, ':', Minutuak, ' ---> ', Segundoak, ' segundo') ;
END.
```

`IrteerakoParametroa2` programaren gauzarik aipagarriena, nola ez, parametroen jokaera litzateke. Programaren helburua (aurreko bertsiorena bezala) denbora bi daturen bitartez teklatutik irakurri eta segundoen kalkulua burutu ondoren pantailaraketa egitea da. Hurrengo taulan, `IrteerakoParametroa2` programan agertzen diren parametroen portaera prozedura bakoitzeko azaltzen da:

		Parametro Errealak		
		`Orduak`	`Minutuak`	`Segundoak`
Deiak	`OrduIrakurketa()`	irteerakoa	~~X~~	~~X~~
	`MinutuIrakurketa()`	~~X~~	irteerakoa	~~X~~
	`SegunduenKalkulua()`	sarrerakoa	sarrerakoa	irteerakoa

Nabaria denez parametro erreal bat, `Orduak` adibidez, dei batzutan irteerakoa izan daiteke eta beste dei batzutan sarrerakoa eta horren arrazoia azpirrutina bakoitzean duen portaeran datza. Izan ere, `Orduak` aldagaiak `OrduIrakurketa()` prozeduran balioa hartzen duenez irteerakoa izan beharko du derrigorrez; baina `Orduak` aldagai berbera datua da `SegunduenKalkulua()` prozedurarako (ondorioz sarrerakoa dela adierazi beharra dago).

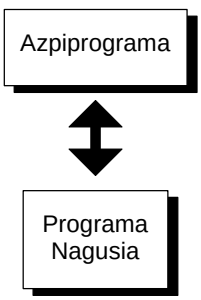
Non adierazten da parametro batek sarrerako ala irteerako portaera izango duen?. Zeharka bada ere **6.3.2.1.1** puntuan eta **6.3.2.2.1** puntu honetan ikusi dugu; *parametro batek duen jokaera azpiprogramaren goiburukoan, parametro formalaren bitartez zehazten da*. Adibide berdinaren bigarren bertsioarekin jarraituz parametro formalen taula bete dezagun:

		Parametro Formalak		
		`Ord`	`Min`	`Seg`
Goiburukoak	`OrduIrakurketa()`	`VAR`	~~X~~	~~X~~
	`MinutuIrakurketa()`	~~X~~	`VAR`	~~X~~
	`SegunduenKalkulua()`			`VAR`

Ondorioz, parametro bat irteerakoa izan dadin azpiprogramaren goiburukoan `VAR` hitz erreserbatua jarriko da; aldiz, parametro bat sarrerakoa izan dadin azpirrutinaren goiburukoan dagokion parametro formalaren aurrean ez da ezer jarriko.

6.3.2.3 Sarrera/Irteerako parametroak

Parametro batek, dei berean, sarrerako eta irteerako jokaera izan dezake. Beraz komunikazioa zentzu bietan ematen da. Ikus irudia:



Azpiprogramaren helburua bete ahal izateko daturen bat behar da (*sarrerako parametro* bezala pasatuko zaio deia egitean), eta azpirrutinan lorturiko emaitza parametro beraren bitartez (orain *irteerako parametroa* izango da) programa nagusiari itzultzen zaio.

Sarrera/Irteerako parametro batek balio ezaguna du azpiprogramara sartzean, eta azpirrutina barruan aldatu ondoren programa nagusiarengan eragina izango du.

6.3.2.3.1 Adibideak

Sarrera/Irteerako parametroen adibidea egin aurretik ariketa bat planteatuko dugu orain arte ikasi dugunarekin burutzeko:

> *"Osoak eta positiboak diren kopuru bi teklatuz irakurri ondoren, azpiprograma baten bitartez aldagai biren balioak trukatu (programaren azkenean batak zuen balioa besteak izango du, eta bigarren aldagaiak zuena lehenengoak izango du)"*

Askorik pentsatu gabe honelako hiru azpiprogramak erabil genitzake:

```
PROCEDURE DatuBatIrakurri (VAR ___)
```

```
PROCEDURE DatuakTrukatu ( ___; VAR ___)
```

```
PROCEDURE DatuBatErakutsi ( ___)
```

Goiburukuen hurbilpenak

```
DatuBatIrakurri (Zahar1) ;
DatuBatIrakurri (Zahar2) ;

DatuakTrukatu (Zahar1, Zahar2,
               Berri1, Berri2) ;

DatuBatErakutsi (Berri1) ;
DatuBatErakutsi (Berri2) ;
```

Programa nagusiko deiak

Bi osoen arteko trukateta egiten duen programari `IrteerakoParametroa3` deitzen badiogu, eta goiko hiru prozedurak aintzat hartzen badira honelako kodea eta exekuzio adibidea izango lituzke:

```
PROGRAM IrteerakoParametroa3 ;                { \TP70\06\PARAM3I.PAS }

PROCEDURE DatuBatIrakurri (VAR Znbki : Byte) ;
BEGIN
  Write ('Zenbaki oso bat eman: ') ;
  ReadLn (Znbki) ;
END;

PROCEDURE DatuBatErakutsi (Znbki : Byte) ;
BEGIN
  WriteLn ('Zenbakia = ', Znbki) ;
END;

PROCEDURE DatuakTrukatu (Datu1, Datu2 : Byte;
                         VAR Emaitza1, Emaitza2 : Byte) ;
BEGIN
  Emaitza1 := Datu2 ;
  Emaitza2 := Datu1 ;
END;

VAR
  Zahar1, Zahar2, Berri1, Berri2 : Byte ;
BEGIN
  DatuBatIrakurri (Zahar1) ;                    (* Programa Nagusia *)
  DatuBatIrakurri (Zahar2) ;
  DatuakTrukatu (Zahar1, Zahar2, Berri1, Berri2) ;
  Zahar1 := Berri1 ;
  Zahar2 := Berri2 ;
  DatuBatErakutsi (Zahar1) ;
  DatuBatErakutsi (Zahar2) ;
END.
```

```
Zenbaki oso bat eman: 7
Zenbaki oso bat eman: 28
Zenbakia = 28
Zenbakia = 7
_
```

Adibidean `DatuakTrukatu()` prozedurari `7` eta `28` balioak (orden horretan) sartzen zaizkionez, irteerako parametroetan itzuliko diren `28` eta `7` (orden horretan) kopuruak `Zahar1` eta `Zahar2` aldagaien hasierako balioak ordezkatzeko erabiliko dira. Baina zeregin berdina beste modu batez egiterik bada, `DatuakTrukatu()` prozeduran sarrera/irteera parametro bakar bi erabiliz:

```
PROGRAM SarreraIrteerakoParametroa1 ;            { \TP70\06\PARAM1SI.PAS }

PROCEDURE DatuBatIrakurri (VAR Znbki : Byte) ;
BEGIN
  Write ('Zenbaki oso bat eman: ') ;
  ReadLn (Znbki) ;
END;

PROCEDURE DatuBatErakutsi (Znbki : Byte) ;
BEGIN
  WriteLn ('Zenbakia = ', Znbki) ;
END;

PROCEDURE DatuakTrukatu (VAR Zbk1, Zbk2 : Byte) ;
VAR
  Laguntzaile : Byte ;
BEGIN
  Laguntzaile := Zbk1 ;         (* Zbk1-ren balioa gorde *)
  Zbk1 := Zbk2 ;
  Zbk2 := Laguntzaile ;
END;

VAR
  Zahar1, Zahar2 : Byte ;
BEGIN                              (* Programa Nagusia *)
  DatuBatIrakurri (Zahar1) ;
  DatuBatIrakurri (Zahar2) ;
  DatuakTrukatu (Zahar1, Zahar2) ;
  DatuBatErakutsi (Zahar1) ;
  DatuBatErakutsi (Zahar2) ;
END.
```

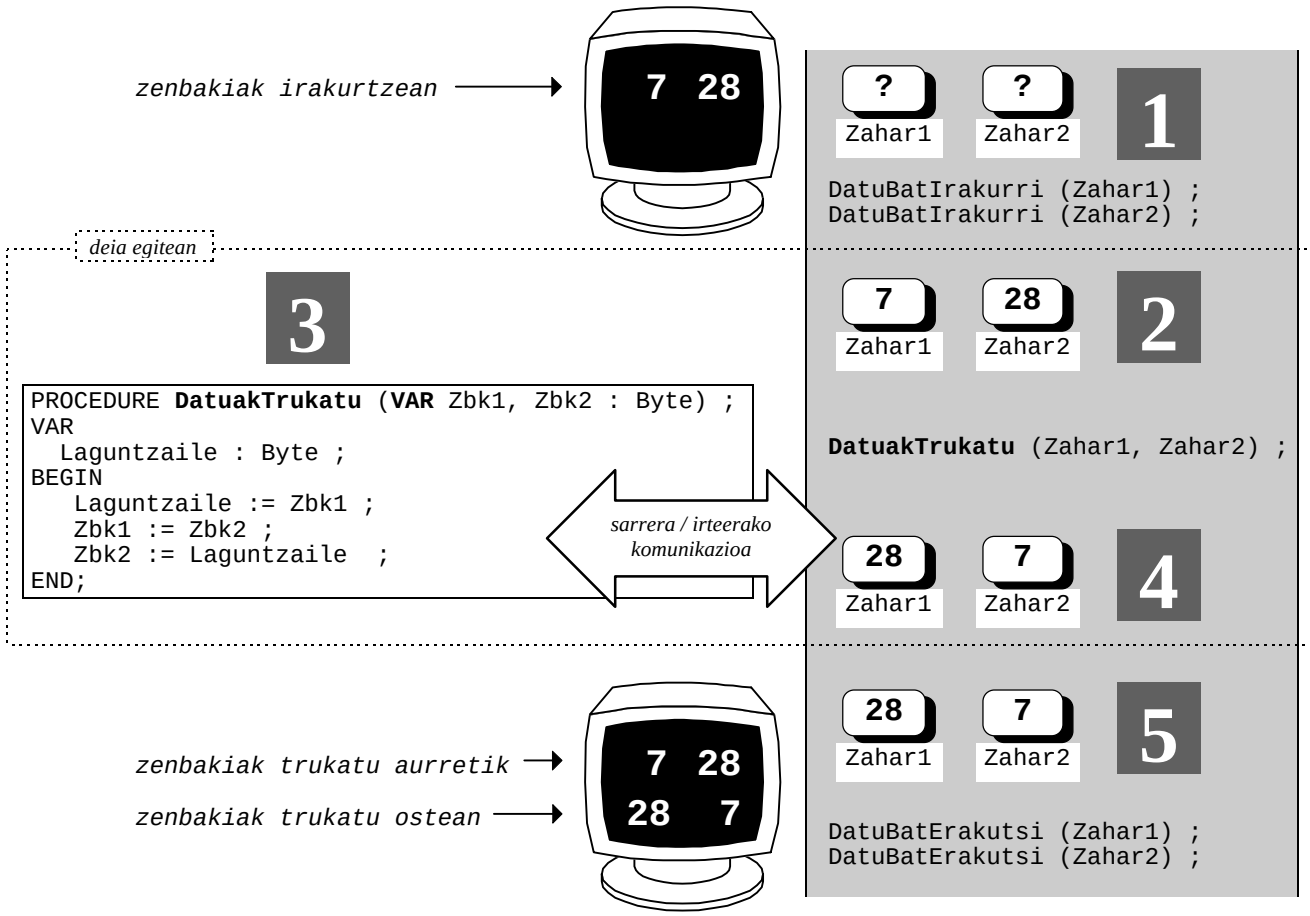
`SarreraIrteerakoParametroa1` izeneko programan dagoen desberdintasun bakarra `DatuakTrukatu()` prozeduran sarrera/irteera parametroak agertzen direla da. Horren deia egiten denean `Zahar1` eta `Zahar2` programa nagusiko aldagaiek derrigorrez balio ezagunak izango dituzte eta `Zbk1` zein `Zbk2` parametro formalei pasatzen zaizkie. Baina `Zbk1` eta `Zbk2` parametroak irteerakoak direnez euren aldaketak programa nagusiko `Zahar1` eta `Zahar2` aldagaietan somatzen dira.

`SarreraIrteerakoParametroa1` programaren eskema bat eginez:

Suposa daitekeenez, programa nagusia hasten denean `Zahar1` eta `Zahar2` aldagaiak balio ezagunik gabe aurkituko dira. Datuok lortzeko `DatuBatIrakurri()` prozedurari birritan deituko zaio, adibidez `7` eta `28` kopuru osoak `Zahar1` eta `Zahar2`-an gordez.

Ondorioz, `DatuakTrukatu()` prozedura deitu aurretik sarrera/irteerako `Zahar1` eta `Zahar2` uneko parametroek hurrenez hurren `7` eta `28` balioko dute. Parametrook sarrerakoak direnen aldetik `7` eta `28` kopuru osoak `DatuakTrukatu()` moduluari pasatuko dizkio eta ...

3 ... bertan aldaketak gertatzen dira. Hots, aldagai laguntzaile baten bitartez egiten den trukaketa: `7` eta `28` direnak `28` eta `7` geratzen dira. Baina `Zbk1` eta `Zbk2` parametro formalak irteerakoak dira ere (kontutan izan aurretik `VAR` hitza daukatela), beraz `DatuakTrukatu()` prozeduraren exekuzio bukatzean balio berriak parametro errealei eransten zaizkie.



4 Esan den bezala `DatuakTrukatu()` prozeduratik irtetean `Zahar1` eta `Zahar2` aldagaien balioak, hurrenez hurren, `28` eta `7` izango direnez `DatuBatErakutsi ()` modulua deitzean bosgarren puntuan azaltzen den pantailaraketa agertuko da.

5 `DatuBatIrakurri(Zahar1)` prozedura deitzean pantailan, besteak beste, `7`-koa agertuko da; `DatuBatIrakurri(Zahar2)` deia egitean pantailan teklatuz sartu den `28`-koa agertuko da.

Zenbaki biak elkar trukatu ondoren, `DatuBatErakutsi(Zahar1)` izeneko azpirrutinak eragiten duen irteera `28` da; eta `DatuBatErakutsi(Zahar2)` deiari dagokiona `7` da.

Konturatzen bagara `SarreraIrteerakoParametroa1` izeneko programa osatzen duten hiru azpirrutinetan parametroen jokamoldeak posible diren hirurak daude. Izan ere `Zahar1` edo `Zahar2` parametro erreala `DatuBatIrakurri()` prozeduran *irteerakoa* da baina `DatuBatErakutsi()` prozeduran *sarrerakoa*, eta `DatuakTrukatu()` prozeduran `Zbk1` eta `Zbk2` parametroak *sarrera/irteerakoak* dira.

Hiru azpirrutinetan lan egiten duten parametroen taula eraikiz, honako hau izango genuke. Non irteerako zein sarrera/irteerako komunikazioetarako parametro formaletan VAR hitz erreserbatua agertzen den.

	Parametroak				
	Errealak		**Formalak**		
	`Zahar1`	`Zahar1`	`Znbki`	`Zbk1`	`Zbk2`
`DatuBatIrakurri()`	irteerakoa	irteerakoa	VAR	X	X
`DatuakTrukatu()`	sarrera irteerakoa	sarrera irteerakoa	X	VAR	VAR
`DatuBatErakutsi()`	sarrerakoa	sarrerakoa		X	X

Beraz non dago irteerako eta sarrera/irteerako komunikazio moten arteko diferentzia, parametro formalak berdinak badira?. Beste era batera galdetuz, `DatuBatIrakurri()` prozeduraren `Znbki` parametro formala eta `DatuakTrukatu()` moduluaren `Zbk1` parametroa berdintsuak dira, bata irteerakoa eta bestea sarrer/irteerakoa izanik, zerk ezberdintzen ditu?. Egia esan, diferentzia prozedura bakoitzaren deiaren aurrean dago, hau da, prozedurak elikatzeko behar diren parametro errealetan.

Hots, irteerako komunikazioa darabilen `DatuBatIrakurri()` prozedurak emaitza lortu ahal izateko ez duenez programa nagusiko daturik behar `Zahar1` edo `Zahar2` parametro errealen balioak hasieran ezezagunak izan daitezke. Baina `DatuakTrukatu()` moduluan berriz, `Zahar1` eta `Zahar2` parametro errealak ezagunak eta baliodunak izango dira deia egin baino lehenago (datu horietaz oinarritzen baita emaitzak kalkulatzeko).

6.3.3 Parametroen erabilpena Turbo Pascal lengoaian

Turbo Pascal lengoaiak parametroen erabilpena ezagutzen dituen mekanismoak hiru dira, lehenengo biak Pascal lengoaia estandarretik datozkienak eta hirugarrena Turbo Pascal 7.0 bertsiotik aurrera gehitu dena. Parametroen erabilpen ekintzari parametro pasatze esaten zaio, ondorioz hiru parametro pasatzerekin topo egingo dugu (jarraian datozen **6.3.3.1** eta **6.3.3.3** bitarteko puntuetan ikusiko direnak):

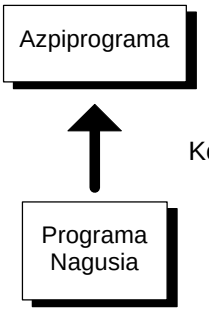
1. Baliozko parametroa
2. Aldagai-parametroa
3. Konstante-parametroa

6.3.3.1 Baliozko parametroa puntuan aztertzen den pasatze modua, kontzeptualki, **6.3.2.1 Sarrerako parametroak** esandakoarekin konektaturik dagoenez hura gogoan izatea garrantzizkoa da. Berdintsu esan daiteke **6.3.2.2 Irteerako parametroak** ikasi dugunarekin, hots lotura duela **6.3.3.2 Aldagai-parametroa** puntuan aztertzen den pasatze moduarekin. Lehenengo biak ikusitakoan hirugarrena, konstante-parametroa, erraz ulertzen da.

6.3.3.1 Baliozko parametroa

Parametro pasatze modu hau, modulu nagusia eta menpekoaren arteko *sarrerako* komunikazioa lortu nahi denean aplikatzen da. Hau da, **6.3.2.1.1 Adibideak** puntuan ikusitako `TxistuBi(DenboraTartea)` deia egitean, `DenboraTartea` parametro erreala eta dagokion

`Tartea` parametro formalaren arteko erlazioa lehenengoaren balioan datza. Programa nagusitik informazioa azpirrutinari eskaintzen zaio, komunikazioa sarrerako parametroen bitartez gauzatuz.



Nolabait esateko, moduluen arteko *sarrerako* komunikazioa antolatzeko *baliozko parametroak* pasatuko dira batetik bestera.

Azpirrutinari dagokion lana bete ahal izateko, parametroen bidez heltzen zaizkion datuetan oinarrituko da. Baina datu horiek aldatuz gero, aldaketok modulu deitzailearengan eraginik ez dute izango.

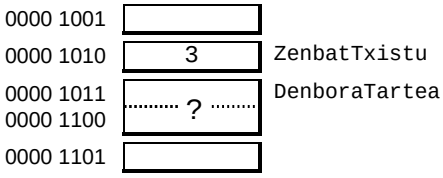
Baina baliozko parametroaren pasatze era adibide baten bitartez ikas dezagun. Demagun lehenago landutako `SarrerakoParametroa3` programaren beste bertsio bat egin nahi dugula, eta desberdintzeko `SarrerakoParametroa4` izendatuko dugula. Programa berri honek egingo duena honela definitzen da:

> *"Azpiprograma batean txistuak lortuko dira. Txistuen kopurua eta txistu biren arteko denbora tartea, zenbaki osoak eta positiboak, programa nagusian teklatuz irakurriko dira"*

`SarrerakoParametroa4` programaren muina `HainbatTxistu()` prozeduran dago, zeinek bi datu behar dituen. Lehenengo datua txistuen kopurua izango da, eta bigarrena euren arteko denbora tartea milisegundotan emanik, bakoitzari programa nagusian dagokion aldagaia `ZenbatTxistu` eta `DenboraTartea` izango da. Hona hemen `SarrerakoParametroa4` programaren kodea:

```
PROGRAM SarrerakoParametroa4 ;                { \TP70\06\PARAM4S.PAS }
USES
  Crt ;

PROCEDURE HainbatTxistu (Kopurua : Byte; Tartea : Integer) ;
VAR
  Kont : Byte ;
BEGIN
  FOR Kont:=1 TO Kopurua DO
  BEGIN
     Write (#7) ;         (* txistua ASCII taulan zazpigarren karakterea *)
     Delay (Tartea) ;
  END ;
END;

VAR
  ZenbatTxistu : Byte ;
  DenboraTartea : Integer ;
BEGIN                              (* Programa Nagusia *)
  REPEAT
    Write ('Txistuen kopurua (1-10): ') ;
    ReadLn (ZenbatTxistu) ;
  UNTIL (ZenbatTxistu >= 1) AND (ZenbatTxistu <= 10) ;
  REPEAT
    Write ('Txistuen arteko denbora tartea milisegundotan (1000-5000): ') ;
    ReadLn (DenboraTartea) ;
  UNTIL (DenboraTartea >= 1000) AND (DenboraTartea <= 5000) ;

  HainbatTxistu (ZenbatTxistu, DenboraTartea) ;
END.
```

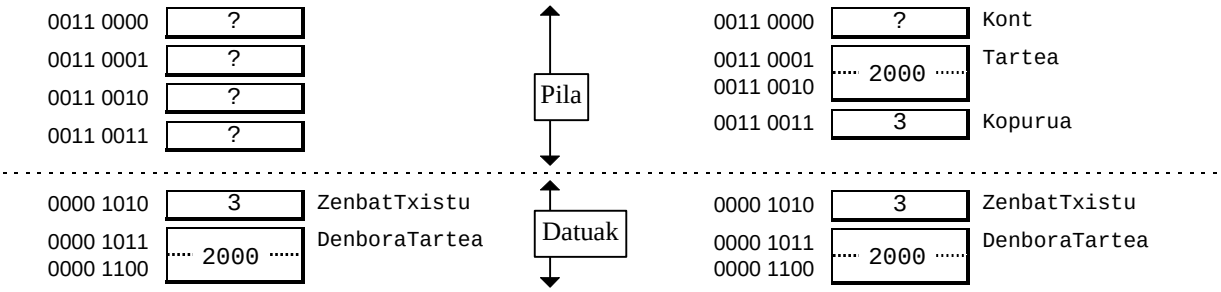
Ikus daitekeenez `SarrerakoParametroa4` programa nagusian `ZenbatTxistu` eta `DenboraTartea` aldagaiak deklaratu dira, lehenengoa `Byte` eta bigarrena `Integer` direlarik[11]. Demagun aldagai bakoitzaren memoria helbideak ondoko irudian emandakakoak direla, `SarrerakoParametroa4` programaren exekuzioa hasten denenan memoria posizio horietan balio ezezagunak aurkituko dira (zer esanik ez, datuok ere kode bitarrean gordetzen direla memorian), baina errazagoa gertatzen zaigulako aldagaien edukiak idazkera hamartarrean jarriko ditugu. Horrela, `ZenbatTxistu` txistuen kopurua zehaztean, adibidez, `3` tekleatu ondoren, memorian hau izango genuke:




```
REPEAT
   Write ('Txistuen kopurua (1-10): ') ;
   ReadLn (ZenbatTxistu) ;
UNTIL (ZenbatTxistu >= 1) AND (ZenbatTxistu <= 10) ;
```

Bigarren irakurketa, `DenboraTartea` aldagaiarena, burutu eta gero `0000 1011` helbidean hasten diren hurrengo bi kutxak beteko lirateke. Baina zergatik ez dugu `HainbatTxistu()` prozeduraren aldagai eta parametroi dagozkien memoria erreserbak adierazi?. Nolabait esateko, azpirrutinen bertako aldagaiek daukaten kategoria eta programa nagusiaren aldagaiek daukatena ezberdina delako. Izan ere, `HainbatTxistu()` prozedura deitu aurretik bere aldagaien erreserbak egin gabe egongo dira, eta, deia suertatzen denean memoriaren pila delakoan dagozkion erreserbak egingo dira (pilari stack esaten zaio ere).

`SarrerakoParametroa4` programaren exekuzioa hasten denean `ZenbatTxistu` eta `DenboraTartea` aldagaierako erreserbak egiten dira, eta programa amaitu arte memoria posizio horiek erabiliak izango dira. Baina, `HainbatTxistu()` prozedurak dituen `Kopurua`, `Tartea` eta `Kont` aldagaien erreserbak `SarrerakoParametroa4` programa horren exekuzio hasieran ez dira egiten, azpirrutinaren deian baizik. Hona hemen `HainbatTxistu()` prozedura deitu aurretik eta deitu ondoren memoriaren balizko mapa bat:



Programa nagusiko deia aurretik *Programa nagusiko deia egitean*

Eskuineko zutabean ikusten den bezala, programa nagusian `HainbatTxistu()` prozeduraren deia burutzean, parametro erreal bakoitzeko erreserba bat egiten da pilan. Horrez gain, eta pasatze modua baliozko parametroena delako, erreserbaturiko posizio berri horietan `ZenbatTxistu` eta `DenboraTartea` parametro errealen balioak kopiatzen dira.

Baliozko parametroak pasatzean azpirrutinaren deian parametro errealak espresioak izan daitezke: konstanteak, aldagaiak, adierazpenak (aritmetikoak, boolearrak, ...), edo funtzioak. Horra hor baliozko parametro formalak dituen prozedura baten zenbait dei onargarri:

[11] `Byte` datu-motak memorian 8 bit (byte bat) hartzen du, eta `Integer` datu-motak 16 bit.


```
HainbatTxistu (3, DenboraTartea) ;              { konstantea eta aldagaia }
HainbatTxistu (ZenbatTxistu, DenboraTartea) ;   { bi aldagai }
HainbatTxistu (3, DenboraTartea + 2) ;          { konstantea eta adierazpena }
HainbatTxistu ( sqr(ZenbatTxistu), 2) ;         { funtzioa eta konstantea }
```

Baliozko parametroak pasatzean, esan den bezala, uneko parametroak orokorrean espresioak izan daitezke. Baina edozein kasutan, parametro pasatze era honetan 6 urrats ezberdintzen dira:

1 Dagoenekoz memorian banaketa bakarra egin dugu, zeinen arabera programa nagusiko datuen gelaskak eta pilarako gelaskak bereizten ziren. Baina programaren sententziak ere ordenadorearen memorian pilaturik daude, eta horren ondorioz programaren kodeari memori posizio jakin batzuk dagozkio.

Azpiprograma baten deia egiteak sententzien jauzi bat suposatzen du, eta azpiprogramaren aginduak bete ondoren programa nagusiaren deiaren ondoko sententziarekin jarraitu beharra dago. Horregatik deia egitean lehenengo urratsa, programa nagusiaren hurrengo sententziaren helbidea pilan gordetzen da.

2 Aldagai berriak sortuko dira pilan, parametro formalak izango direnak. Eta horiekin batera azpirrutinak behar izan ditzakeen bertako aldagaientzat ere erreserbak egingo dira pilan.

Esate baterako, `HainbatTxistu()` prozeduraren parametro formal biak eta `Kont` aldagai lagungarria pilan sortzen dira, baina euren arteko diferentzia hasieraketan dago ...

3 ... hots, `Kont` aldagaiak ez du balio ezagunik izango baina `Kopurua` eta `Tartea` aldagaiek parametro errealen balioak izango dituzte. Horretarako `HainbatTxistu()` prozeduraren deian parametro erreal bakoitzak balio konkretu bat suposatzen du (konstantea, aldagaia, adierazpena edo funtzioa[12]).

4 Parametro errealetatik hartutako balioak kontutan izanik azpirrutinaren sententziak exekutatzen dira. Bainan, **6.3.2.1 Sarrerako parametroak** puntuan esandakoa gogoratuz, parametro formalen edo azpirrutinaren bertako aldagaien balioak aldatuz gero pograma nagusiarengan eraginik ez da ezagutuko, eta ondorioz parametro errealak aldatu gabe geratuko dira.

5 Azpiprogramaren azken sententzia exekutatu eta gero, parametro formal eta bertako aldagaiei pilan dagokien memoria askatu egiten da. Une honetan azpirrutinaren modulu deitzaileak hartu beharko luke programa exekutatzeko ardura, horretarako ...

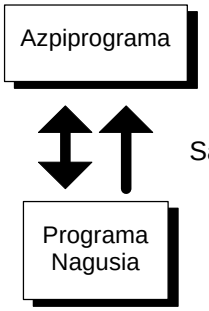
6 ... lehenengo urratsean aipatu dugun itzultzeko helbideaz baliatuko da. Beste modu batean azaldurik, azpiprogramaren deia egitean pilan grabaturik utzi den programa nagusiaren (modulu deitzailearen) hurrengo sententziaren helbidera jauzi egiten da, kontrola modulu deitzaileak berreskuratuz.

[12] Funtzioa Pascal lengoaiaren azpirrutina mota bat da eta duen ezaugarria balio bakar bat itzultzen duela da, zortzigarren oinoharra errepikatuz ikus **6.4 AZPIPROGRAMA MOTAK TURBO PASCAL LENGOAIAN** deituriko puntua.

6.3.3.2 Aldagai-parametroa

Parametro pasatze modu hau, modulu nagusia eta menpekoaren arteko *irteerako* edo *sarrera/irteerako* komunikazioa lortu nahi denean aplikatzen da. Egitan, sarrera/irteera eta irteera komunikazioak berdintsuak dira, bietan parametro errealak aldatuak izango baitira azpiprograma exekutatu ondoren (irteerako komunikazioan parametro errealek duten balioa ez da azpirrutina barnean erabiltzen).

Gogora ekar dezagun orain **6.3.2.2.1 Adibideak** puntuan ikusitako `IrteerakoParametroa1` programa, zeinean `SegundoenKalkulua()` prozedura erabiltzen den. `SegundoenKalkulua()` prozedurak hiru parametro ditu, alde batetik `Orduak` eta `Minutuak` (baliozko pasatzearekin sarrerakoak direlako), eta irteerakoa den `Segundoak` aldagai-parametro bezala pasatuko dena azpiprogramara.



Azpiprograma eta modulu deitzailearen arteko *sarrera/irteerako* komunikazioa edo *irteerako* komunikazioa gauzatzeko *aldagai-parametroak* pasatuko dira batetik bestera.

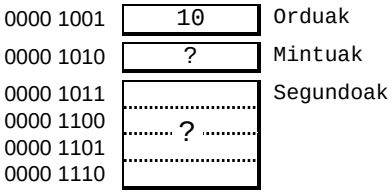
Sarrera/Irteerako Komunikazioa | Irteerako Komunikazioa | *Aldagai Parametroak*

Azpirrutinak modulu deitzaileari itzuli behar dion emaitza azpiprograma barnean zehaztu egiten da, eta parametro horren aldaketak modulu deitzailean iraungo du.

Aldagai-parametroaren pasatze era adibide batez azaltzeko `IrteerakoParametroa1` programa idaz dezagun berriro, eta `SegundoenKalkulua()` prozeduraren deia gertatzean memoriaren mapa marraz dezagun pila eta datuen arloak bereiziz:

```
PROGRAM IrteerakoParametroa1 ;                { \TP70\06\PARAM1I.PAS }

PROCEDURE SegundoenKalkulua (Ord, Min : Byte; VAR Seg : LongInt) ;
VAR
  Metagailu : LongInt ;
BEGIN
  Metagailu := Ord*60 + Min ;    (* Denbora minututara iragan  *)
  Metagailu := Metagailu * 60 ;  (* Denbora segundotara iragan *)

  Seg := Metagailu ;   (* Emaitza programa nagusira itzultzeko *)
END;

VAR
  Orduak, Minutuak : Byte ;
  Segundoak : LongInt ;
BEGIN                                      (* Programa Nagusia *)
  REPEAT
    Write ('Orduak eman (0-23):   ') ;
    ReadLn (Orduak) ;
  UNTIL (Orduak >= 0) AND (Orduak <= 23) ;
  REPEAT
    Write ('Minutuak eman (0-59): ') ;
    ReadLn (Minutuak) ;
  UNTIL (Minutuak >= 0) AND (Minutuak <= 59) ;

  SegundoenKalkulua (Orduak, Minutuak, Segundoak) ;

  WriteLn (Orduak, ':', Minutuak, ' ---> ', Segundoak, ' segundo') ;
END.
```


Dakigunez `IrterakoParametroa1` programa nagusian `Orduak`, `Minutuak` eta `Segundoak` aldagaiak deklaratu dira, lehenengo birako `Byte` datu-mota aski izan daiteke eta hirugarrenako `Word` edo `LongInt` aproposak izan daitezke[13]. Demagun aldagai bakoitzaren memoria helbideak ondoko irudian emandakakoak direla, programaren exekuzioa hasten denenan memoria posizio horietan balio ezezagunak aurkituko dira. Baina `Orduak` datua zehaztean, adibidez, `10` tekleatu ondoren, memorian hau izango genuke:




```
REPEAT
   Write ('Orduak eman (0-23):   ') ;
   ReadLn (Orduak) ;
UNTIL (Orduak >= 0) AND (Orduak <= 23) ;
```

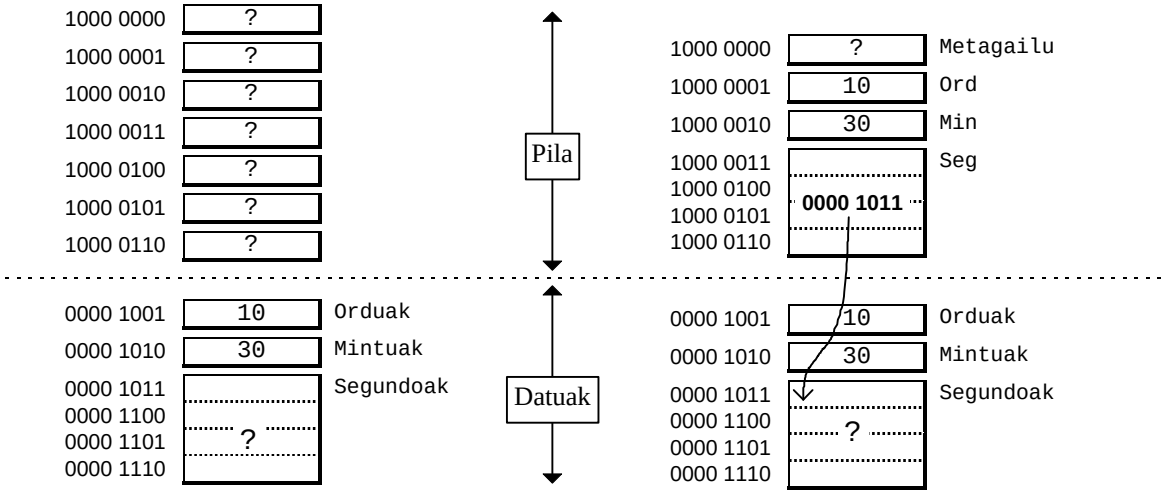
Bigarren irakurketa, `Minutuak` aldagaiarena, egin eta gero `0000 1010` helbidedun gelaska beteko litzateke. Baina `Segundoak` aldagaiak `SegundoenKalkulua()` prozeduraren barruan balioa hartzen duenez aldagai-parametro bezala pasatu beharko zaio, oraingoan ere parametro pasatze hau 6 urratsetan banatuko dugu.

Aldagai-parametroen erabiltzean:

1 Dakiguna errepikatuz azpiprograma baten deia egiteak, sententzien jauzi bat suposatzen duelako modulu deitzailera itzuli ahal izateko azpirrutinari jarraitzen dion sententziaren helbidea pilan gordetzen da.

2 Aldagai berriak sortuko dira pilan, parametro formalak izango direnak. Eta horiekin batera azpirrutinak behar izan ditzakeen bertako aldagaientzat ere erreserbak egingo dira pilan.

Adibidez `SegundoenKalkulua()` prozeduraren hiru parametro formalak eta `Metagailu` aldagai lagungarria pilan sortzen dira, baina bakoitzaren hasieraketa desberdina da ...



Programa nagusiko deia aurretik *Programa nagusiko deia egitean*

3 ... hots, `Metagailu` aldagaiak ez du balio ezagunik izango, baina baliozko pasatze era erabili denez `Ord` eta `Min` aldagaiek dagozkien parametro errealen balioak izango dituzte; ez da gauza bera `Seg` aldagaiarekin gertatzen irteerakoa delako. Pilan sorturiko `Seg` aldagaiak bere parametro

[13] `LongInt` datu-mota aukeratu izan da, zeinek memorian 32 bit (lau byte) hartzen duen.

errealaren helbidea gordetzen du, hori dela eta `SegundoenKalkulua()` prozedura exekutatzen den bitartean pilako `Seg` bitartez programa nagusiko `Segundoak` aldagaiaren memori posizioak erreferentziatzen dira, ondorioz ...

... `SegundoenKalkulua()` prozedura barnean `Seg` aldagaia aldatzean benetan aldatzen dena `Segundoak` aldagaia da. Horregatik da hain zuzen ere irteerakoa helbidearen bitartez erreferentziaturik egotean, gertatutako aldaketak `Segundoak` aldagaiari bideratzen zaizkiolako.

Azpiprogramaren azken sententzia exekutatu eta gero, parametro formal eta bertako aldagaiei pilan dagokien memoria askatu egiten da. Une honetan `SegundoenKalkulua()` azpirrutinaren modulu deitzaileak (programa nagusiak gure adibidean) hartu beharko luke programa exekutatzeko ardura, horretarako ...

... lehenengo urratsean aipatu dugun itzultzeko helbideaz baliatuko da. Beste modu batean azaldurik, azpiprogramaren deia egitean pilan grabaturik utzi den programa nagusiaren (modulu deitzailearen) hurrengo sententziaren helbidera jauzi egiten da, kontrola modulu deitzaileak berreskuratuz.

Moduluen arteko komunikaziorako helbide edo erreferentzia bat erabiltzean (aldagai-parametro pasatze eran) `VAR` hitz erreserbatua jarri behar da parametro formalaren aurrean, eta deia egiten den uneko parametroa espresioa izan ezik aldagaia izango da (dagokion parametro formalaren datu-mota bereko aldagaia hain zuzen ere). Hona hemen hiru dei desegoki:

```
SegundoenKalkulua (Orduak, Minutuak, 7) ;
Error 20: Variable identifier expected.

SegundoenKalkulua (Orduak, Minutuak, Segundoak+4) ;
Error 89: ")" expected.

SegundoenKalkulua (Orduak, Minutuak, Sqr(Segundoak)) ;
Error 122: Invalid variable reference.
```

6.3.3.3 Konstante-parametroa

Parametro pasatze mota hau Turbo Pascal lengoaiaren bertsio berrienetan onartzen da soilik (Turbo Pascal 7.0 bertsiotik aurrerakoak). Aldagai-parametroen kasuan bezala, konstante-parametro batek modulu deitzailearen datuari erreferentzia bat zuzentzen dio. Baliozko parametroen kasuan bezalaxe, konstante-parametroa *sarrerako* komunikazioa gauzatzeko erabiltzen da, hots, menpeko moduluak nagusitik informazioa hartzen duenean baina irteerako daturik ez dagoenean.

Baina konstante-parametro eta baliozko parametroaren arteko desberdintasunik bada. Dakigunez, azpirrutina barnean baliozko parametro batek duen portaera aldagai batena da, hasieraketa modulu deitzailean jasan duen aldagai baten portaera du. Bestalde konstante-parametroa aldagai-parametroarekiko alderik badu, izan ere konstante-parametro batek azpirrutina barnean konstante baten portaera du (modulu deitzaileak emaniko balio konstante batena hain zuzen).

Ondorioz, konstante-parametro bat ezin izango da azpiprograma barruan aldatu, eta, azpiprogramatik beste azpirrutinaren bat deitzen badu ezin daiteke aldagai-parametroaren pasatze era erabili (konpilatzean detektatzen diren erroreak). Jarraian ematen den adibidean

ikus daitekeenez, konstante-parametroaren pasatze modua adierazteko parametro formalaren aurrean `CONST` hitz erreserbatua erabiltzen da:

```
PROGRAM KonstanteParametroa ;                  { \TP70\06\PARAKONS.PAS }
USES
   Crt ;
PROCEDURE GidoienLerroaIdatzi (CONST KopuruA : Byte; KopuruG : Byte) ;
VAR
  Kont : Integer ;
BEGIN
    (* KopuruA := KopuruA DIV 2 ;      ezinezkoa da *)
    FOR Kont:= 1 TO KopuruA DO
       Write ('*') ;
    WriteLn ;
    KopuruG := KopuruG DIV 2 ;
    FOR Kont:= 1 TO KopuruG DO
       Write ('=') ;
    WriteLn ;
END;

VAR
  ZenbatA, ZenbatG : Byte ;
BEGIN
    ClrScr ;                              (* Programa Nagusia *)
    Write ('Zenbat asterisko? ') ;
    ReadLn (ZenbatA) ;
    Write ('Zenbat gidoi? ') ;
    ReadLn (ZenbatG) ;

    WriteLn ('Deiaren baino lehen') ;
    WriteLn ('ZenbatA = ', ZenbatA, '        ZenbatG = ', ZenbatG) ;

    GidoienLerroaIdatzi (ZenbatA, ZenbatG) ;

    WriteLn ('Dei ostean') ;
    WriteLn ('ZenbatA = ', ZenbatA, '        ZenbatG = ', ZenbatG) ;
END.
```

`KonstanteParametroa` programak honelako irteraren bat izan dezake. Non programa nagusian irakurriko datu bietatik `GidoienLerroaIdatzi()` prozedura barruan bakar bat alda daitekeen:

```
Zenbat asterisko? 7
Zenbat gidoi? 9
Deiaren baino lehen
ZenbatA = 7        ZenbatG = 9
*******
====
Dei ostean
ZenbatA = 7        ZenbatG = 9
_
```

Programa aztertzean aldaketak egiten baditugu, konturatuko gara `KopuruA` aldagaia `KopuruG` aldagaia baino babestuago dagoela. Izan ere, konstante-parametro pasatze era aukeraturik ezinezkoa zaigu `GidoienLerroaIdatzi()` prozeduraren barruan honelakorik egitea:

`KopuruA := 17 ;`	parametroari balio berririk ematea
`ReadLn (KopuruA) ;`	parametroa beste azpirrutina bati (aldagai-parametro bezala) pasatzeak konstantea aldatzea suposatuko bailuke

Sarrerako parametroen kasuan esan den bezalaxe, konstante-parametroren bat pasatzean azpirrutinaren deian parametro errealak espresioak izan daitezke: konstanteak,

aldagaiak, adierazpenak (aritmetikoak, boolearrak, ...), edo funtzioak. Horra hor, errepikaturik, konstante-parametro formal bat darabilen prozeduraren dei onargarriak:

```
GidoienLerroaIdatzi (7, ZenbatG) ;                { konstantea }
GidoienLerroaIdatzi (ZenbatA, ZenbatG) ;          { aldagaia }
GidoienLerroaIdatzi (ZenbatA + 3, ZenbatG) ;      { adierazpena }
GidoienLerroaIdatzi ( abs(ZenbatA), ZenbatG) ;    { funtzioa }
```

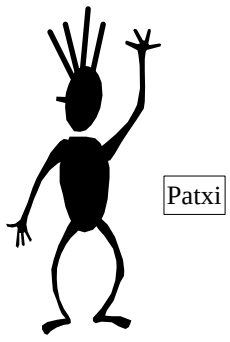
6.3.4 Azpiprogrameen arteko komunikazioa. Laburpena

Komunikazio parametroen bitartez lortzen denez parametroen pasatze moduak gogora ditzagun. Parametro pasatze moduak zehazten du menpeko moduluak nola lan egingo duen, hots, menpeko moduluak definiturik dauzkan parametro formalek definitzen dute modulu deitzailearen eta azpirrutinaren arteko komunikazioa zein izango den. Horren arabera parametroak balioak ala erreferentziak (memori posizioen helbideak) izan daitezke. Jarraian luzatzen den taulan horixe da zutabeak bereizteko irizpide nagusia:

	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

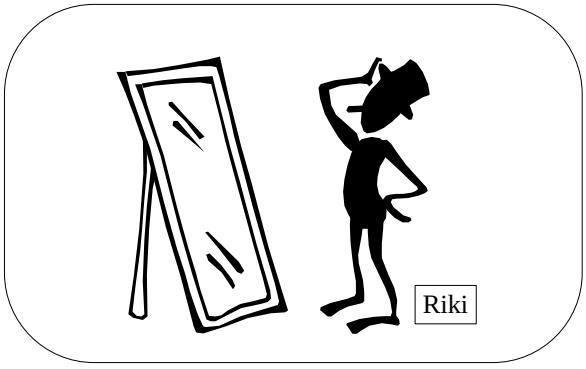
Taularen informazioa biltzen duen istorio bat kontatuko dugu orain, Patxi Punki-ren alegia. **Patxi Punki** bere herriko kale nagusitik dabilen laguna dugu, eta programa nagusiko aldagaitzat hartuko dugu. Bere herriko kale nagusian zenbait zerbitzu (zenbait azpirrutina) eskuragarri du Patxik, besteak beste arropak egiteko jostundegia, tabernak eta ileapaindegia. Dendetako zerbitzuak nola lortzen dituen Patxik azter dezagun:

Kale nagusitik doan bitartean arropa erostea erabakitzen du Patxik, horregatik egokia den prozedurari dei egin beharko zaio (aproposa den dendara sartu). Hona hemen Patxi Punki jostundegira sartu aurretik:

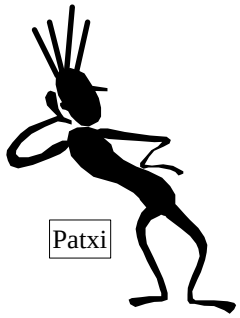


Patxi Punki kale nagusian

Jaka bat egin araztera doa jostunarengana, eta, neurriak hartzeko orduan, jostundegian Patxi sartu beharrean bere anai bikoitza sartzen da. Patxiren anaia Riki da, eta horrelaxe ezagutzen du jostunak dendan dagoen bitartean (argi dago, nolabait, une jakin batean bi pertsona daudela Patxi kanpoan eta Riki jostundegian, eta biak neurri eta itxura berdina dutela). Jostundegiaren azpirrutina martxan dagoen bitartean:

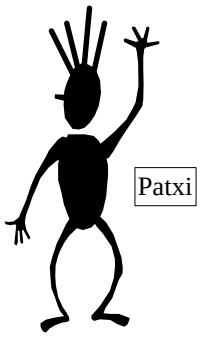


Riki Punki jostundegaiaren barruan



Patxi Punki kale nagusian itxaroten

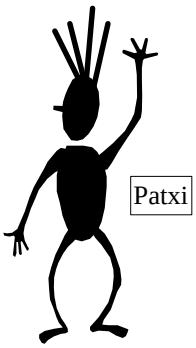
Jostundegiaren azpirrutina bukatu egiten denean Riki desagertzen da, eta kale nagusiko Patxik ez du arropa berririk jantziko.



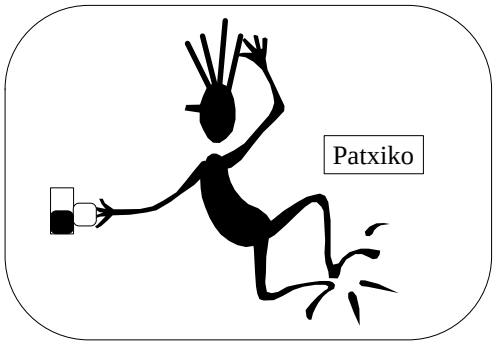
Patxi Punki berriro kale nagusian

B

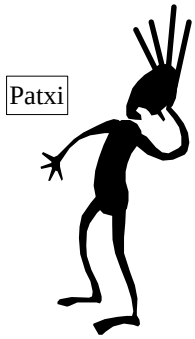
Gure Patxi egarri da eta oraingoan tabernara doa. Taberna barruan tabernariak, konfidantza duenez, Patxiko deitu egiten dio. Pertsona berbera izan arren kalean Patxi bezala ezaguna da eta tabernan Patxiko deritzo gure lagunari. Konturatzen gara Patxik edandakoa berak jasan beharko duela (Riki anairik ez baitago), Patxikok larregi edan ondoren horra hor kalera irtetean Patxik erakutsiko duen egoera.



Patxi kale nagusian tabernan sartu aurretik

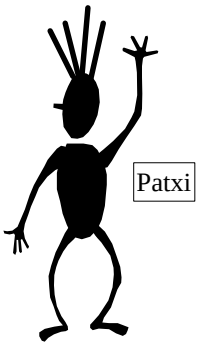


Patxiko (Patxi Punki) taberna barruan

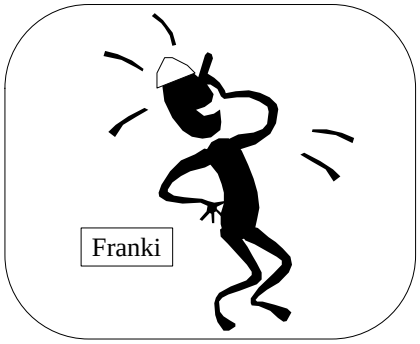


Patxi kale nagusian horrdituta

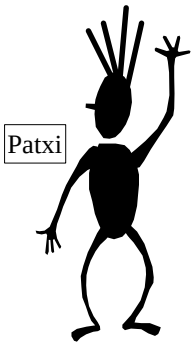
Kale nagusian dagoela ilea moztera erabaki du Patxik, horregatik Patxi Panki ileapaindegirantz abiatzen da; oraingoan ere Patxi bera doa Riki anaiaz baliatu gabe. Ileapaintzaileak Patxi Punki Franki bezala ezagutzen du. Tabernara ez bezala Patxi ileapaindegi barnera sartzean burua kasko batez babesten du, eta ondorioz pertsona bera den Patxi eta Frankiri ezin zaio ilerik moztu eta kalera itzultzean sartu aurretik zeukan itxura izango du.



Patxi kale nagusian gandor eta guzti



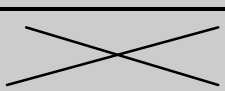
Franki (Patxi Punki) ileapaindegian gandorra babesturik



Patxi kale nagusian gandorrarekin ere

Patxi Punki-ren ibilerek parametroen hiru pasatze erak azaltzen dituzte. Aurrenean, jostundegian, baliozko parametro bat da (datuaren kopia kloniko bat ematen diona eta aldaketarik jasatzen ez duena). Bigarrenean, benetako Patxiren erreferentzia bat dago taberna barruan eta horren ondorioz mozkorra. Hirugarrenean ere Patxiren erreferentzi bat dago ileapaindegian, baina konstante-parametroa denez aldaezina agertzen zaigu Patxi Punki.

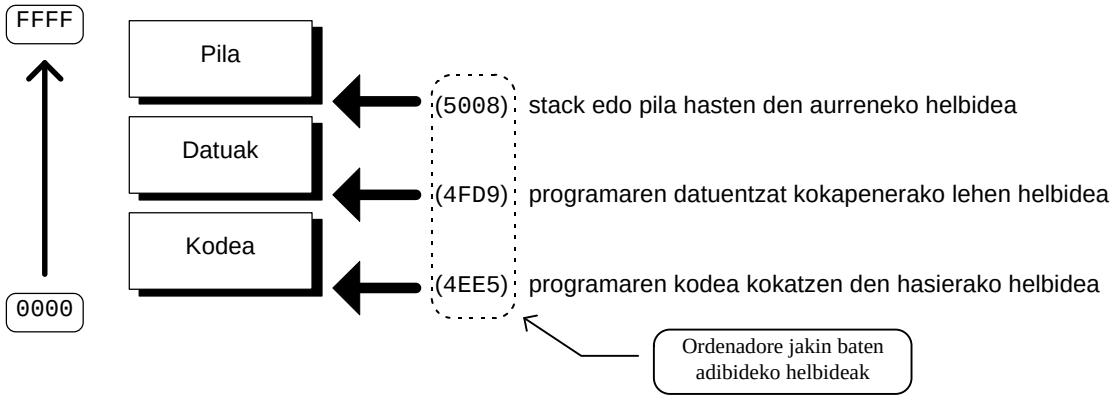
Patxi Punki-ren alegia ikusi ondoren hauxe da bere laburpena:

	Azpiprograma (zerbitzua)	**Turbo Pascal markatzailea**	**Efektua**
Baliozko parametroa	Jostundegia		pertsona desberdinak
Aldagai-parametroa	Taberna	`VAR`	pertsona bera babesik gabe
Konstante-parametroa	Ileapaindegia	`CONST`	pertsona bera babesarekin

6.4 PARAMETRO MOTAK ETA MEMORI HELBIDEAK

Programa bat exekutatzen denean ordenadorearen memorian hiru atal bereiz daitezke. Lehenengo zatian derrigorrezkoa den Sistema Eragilea aurkituko litzateke, eta berarekin batera programaren sententzia konpilatuak (programa makina-kodera itzulita). Memoriaren bigarren zatia programak erabiltzen dituen datuentzat da, programa nagusiko aldagaientzat tokia litzateke bigarren zati hau. Memoriaren hirugarren atalari pila edo stack esaten zaio eta programaren exekuzio hasieran hutsik legoke.

Bereizi ditugun hiru zati horiei segmentu esaten zaie, Turbo Pascal lengoaiak funtzio estandarrak[14] ditu segmentu horiek zer helbidetan hasten diren ezagutzeko. Hau litzateke memoriaren hiru atalen (kode-segmentua, datu-segmentua eta pila) eskema bat:



Helbideak bistaratzeko `AldagaiNagusienHelbideak` programa egikari daiteke, zeinek `Ikus` izeneko unitate bat darabilen `IntToHex()` funtzioa garatzeko:

```
PROGRAM AldagaiNagusienHelbideak ;                  { \TP70\06\HELBID1.PAS }
USES
    Crt, Ikus ;      (* IKUS.TPU  unitatea darabil *)

VAR
   ZbkOsoa : Integer ;
   Karakterea : Char ;
   ZbkErreala : Real ;
   Boolearra : Boolean ;
   OsoaLuzea : LongInt ;
   ZbkByte : Byte ;
   BesteErreala : Real ;

BEGIN              (* Programa Nagusia *)
   ClrScr ;

   WriteLn ('PROGRAMA NAGUSIAN EZAGUTZEN DIREN ALDAGAIEN 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 ;
   Write   ('Pilaren erakusleari dagokion memoria:  ') ;
   Write   (IntToHex (Seg (HeapPtr)),':') ;
   WriteLn (IntToHex (Ofs (HeapPtr))) ;
   WriteLn ; WriteLn ;
```

[14] Parametrorik behar ez dituzten `cSeg`, `dSeg` eta `sSeg` funtzioak.

```
   WriteLn ('ALDAGAIA':30, '        HELBIDEA', '   TAMAINA') ;
   WriteLn ('-----------------------------------------':60) ;

   WriteLn ('BesteErreala --->   ':38,IntToHex (Seg (BesteErreala)),':',
           IntToHex (Ofs (BesteErreala)), '    (', SizeOf (BesteErreala), ')') ;
   WriteLn ('ZbkByte --->   ':38,IntToHex (Seg (ZbkByte)),':',
           IntToHex (Ofs (ZbkByte)), '    (', SizeOf (ZbkByte), '+1)') ;
   WriteLn ('OsoaLuzea --->   ':38,IntToHex (Seg (OsoaLuzea)),':',
            IntToHex (Ofs (OsoaLuzea)), '    (', SizeOf (OsoaLuzea), ')') ;
   WriteLn ('Boolearra --->   ':38,IntToHex (Seg (Boolearra)),':',
           IntToHex (Ofs  (Boolearra)), '    (', SizeOf (Boolearra), '+1)') ;
   WriteLn ('ZbkErreala --->   ':38,IntToHex (Seg (ZbkErreala)),':',
           IntToHex (Ofs (ZbkErreala)), '    (', SizeOf (ZbkErreala), ')') ;
   WriteLn ('Karakterea --->   ':38,IntToHex (Seg (Karakterea)),':',
           IntToHex (Ofs (Karakterea)), '    (', SizeOf (Karakterea), '+1)') ;
   WriteLn ('ZbkOsoa --->   ':38,IntToHex (Seg (ZbkOsoa)),':',
            IntToHex (Ofs (ZbkOsoa)), '    (', SizeOf (ZbkOsoa), ')') ;
END .
```

`AldagaiNagusienHelbideak` programa exekutatu ondoren jarraian erakusten den irteera ager daiteke, kontutan izan ordenadore batetik bestera kode-segmentu, datu-segmentu eta pilaren helbideak desberdinak izan daitezkeela. Baina zalantza gabekoa programa nagusiko aldagaien kokamena izango da, denak datu-segmentuan aurkituko dira:

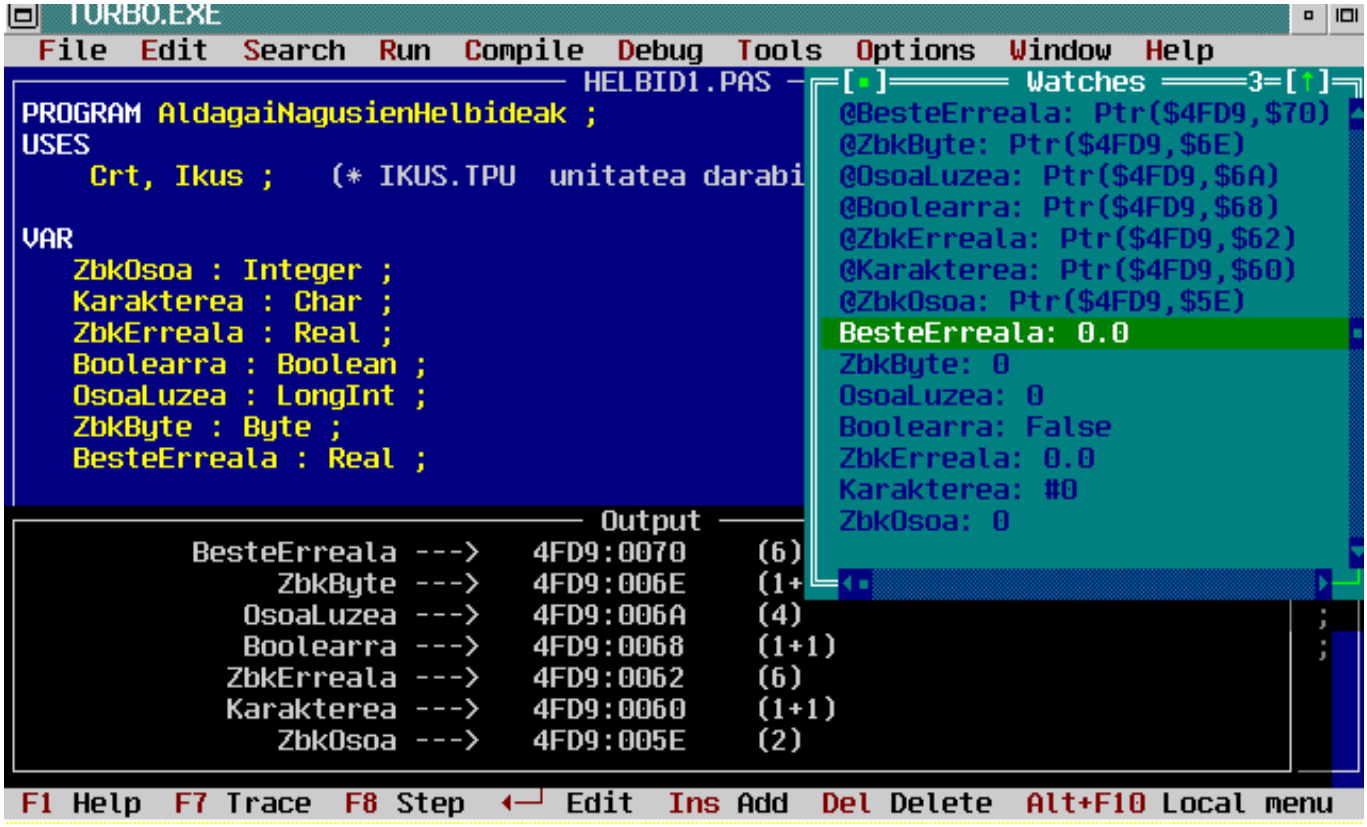
```
PROGRAMA NAGUSIAN EZAGUTZEN DIREN ALDAGAIEN MEMORI HELBIDEAK
============================================================

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)
_
```

Irteera adibide honetan ikus daitekeenez, lehenago deklaraturiko programa nagusiko aldagaiek helbide baxuagoak dituzte azkenean erazagututakoek baino. Bestalde, zortzikote bakar bat behar dituzteen datu-motatako aldagaientzat (adibidez, `ZbkByte`, `Boolearra` eta `Karakterea`) egitan 2 zortzikote hartzen dira memorian, nahiz eta bigarrena erabili ez, honetaz zerbait gehiago esango dugu zortzigarren kapituluan konpilazio-direktibak aipatzean.

Turbo Pascal 7.0 konpiladoreak duen inguruneaz balia gaitezke memori posizioak bistaratzeko, zehazki, menu nagusian agertzen den `Debug` izeneko komandoaren bitartez (hirugarren kapituluko **3.5.6 DEBUG komandoa** izeneko puntua gogoratu).

Kasu honetan programaren zorriketa edo errore detektatze prozesua baino, memoriaren banaketa frogatu nahi da. Horretarako `Debug | Watch` aukeraren bitartez arazketa leihoa ireki eta `Debug | Add watch` bidez kontrolatu nahi diren elementuak (aldagaien helbideak eda edukiak) bertaratuko dira.

Demagun `AldagaiNagusienHelbideak` programaren aldagaiaei dagozkien helbideak eta edukiak erakustea nahi dela, honelakoxe `Watch` leihoa izango genuke:



Non aldagai baten edukia adierazteko aldagaiaren identifikadorea erabiltzen den, eta aldagai horren helbidea adierazteko `@` operadorea aurretik jarri behar zaion identifikadoreari. Nabaria denez emaitza berera iritsten gara: `4FD9:0070` helbidean lehenengo lau digituak segmentua da eta azkeneko lauak desplazamendua, `watch` leihoan helbide hori `($4FD9,$70)` bezala agertzen da.

6.4.1 Baliozko parametroak eta memori helbideak

Helbideak bistaratzeko `AldagaiNagusienHelbideak` programak datuak non kokatzen diren erakusten digu, hurrengo urratsa azpirrutina bat osatzea eta bere bertako aldagai eta parametroei dagozkien helbideak aztertzea litzateke. Hori da `ParametroFormalenHelbideakA` programak duen helburua, parametro formalak eta azpirrutinaren bertako aldagaien helbideak bistaratzea.

```
PROGRAM ParametroFormalenHelbideakA ;                { \TP70\06\HELBID2A.PAS }
USES
    Crt, Ikus ;      (* IKUS.TPU  unitatea darabil *)

PROCEDURE AzpirrutinaBaliozko (Oso1, Oso2, Oso3 : INTEGER) ;
VAR
  BertakoA, BertakoB, BertakoC : INTEGER ;
BEGIN
   WriteLn ; WriteLn ;
   WriteLn ('ALDAGAIA':29, '       HELBIDEA', '   TAMAINA') ;
   WriteLn ('-------------------------------------------':60) ;
```

```
   WriteLn ('BertakoC -->   ':36, IntToHex (Seg (BertakoC)),':',
            IntToHex (Ofs (BertakoC)), '    (', SizeOf (BertakoC), ')') ;
   WriteLn ('BertakoB -->   ':36, IntToHex (Seg (BertakoB)),':',
            IntToHex (Ofs (BertakoB)), '    (', SizeOf (BertakoB), ')') ;
   WriteLn ('BertakoA -->   ':36, IntToHex (Seg (BertakoA)),':',
            IntToHex (Ofs (BertakoA)), '    (', SizeOf (BertakoA), ')') ;
   WriteLn ('Oso3 ------>   ':36, IntToHex (Seg (Oso3)),':',
            IntToHex (Ofs (Oso3)), '    (', SizeOf (Oso3), ')') ;
   WriteLn ('Oso2 ------>   ':36, IntToHex (Seg (Oso2)),':',
            IntToHex (Ofs (Oso2)), '    (', SizeOf (Oso2), ')') ;
   WriteLn ('Oso1 ------>   ':36, IntToHex (Seg (Oso1)),':',
            IntToHex (Ofs (Oso1)), '    (', SizeOf (Oso1), ')') ;

END ;

VAR
   ZbkOso1, ZbkOso2, ZbkOso3 : Integer ;

BEGIN              (* Programa Nagusia *)
   ClrScr ;

   WriteLn ('PARAMETRO ERREALEN ETA FORMALEN HELBIDEAK (BALIOZKO PASATZE ERAN)') ;
   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 ('ZbkOso3 --->   ':36, IntToHex (Seg (ZbkOso3)),':',
            IntToHex (Ofs (ZbkOso3)), '    (', SizeOf (ZbkOso3), ')') ;
   WriteLn ('ZbkOso2 --->   ':36, IntToHex (Seg (ZbkOso2)),':',
            IntToHex (Ofs (ZbkOso2)), '    (', SizeOf (ZbkOso2), ')') ;
   WriteLn ('ZbkOso1 --->   ':36, IntToHex (Seg (ZbkOso1)),':',
            IntToHex (Ofs (ZbkOso1)), '    (', SizeOf (ZbkOso1), ')') ;

   AzpirrutinaBaliozko (ZbkOso1, ZbkOso2, ZbkOso3) ;

END .
```

```
PARAMETRO ERREALEN ETA FORMALEN HELBIDEAK (BALIOZKO PASATZE ERAN)
==================================================================

PILARI dagokion segmentuaren hasiera:  5E78
DATUEI dagokien segmentuaren hasiera:  5E4A
KODEARI dagokion segmentuaren hasiera: 5D33


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

                    ZbkOso3 --->   5E4A:0062    (2)
                    ZbkOso2 --->   5E4A:0060    (2)
                    ZbkOso1 --->   5E4A:005E    (2)


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

                    BertakoC -->   5E78:3DEE    (2)
                    BertakoB -->   5E78:3DF0    (2)
                    BertakoA -->   5E78:3DF2    (2)
                    Oso3 ------>   5E78:3DF8    (2)
                    Oso2 ------>   5E78:3DFA    (2)
                    Oso1 ------>   5E78:3DFC    (2)
_
```

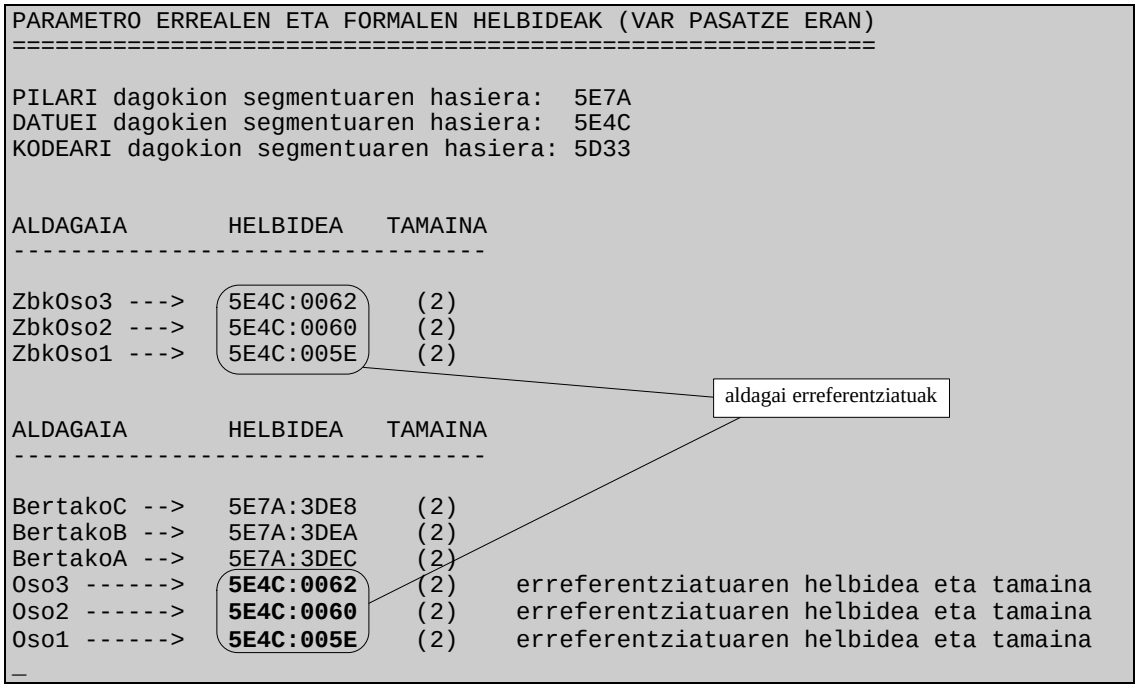
Programa nagusiko `ZbkOso1`, `ZbkOso2` eta `ZbkOso3` izeneko aldagaiak zenbaki osoak dira eta datuen segmentuan daude (adibide honetan `5E4A` helbideko segmentuan daude, eta bertan bi byteko desplazamendua dute). `AzpirrutinaBaliozko()` prozedurako `Oso1`, `Oso2` eta `Oso3` parametroak eta `BertakoA`, `BertakoB` zein `BertakoC` izeneko aldagaiak pilan aurkitzen dira (adibidean `5E78` helbideko segmentuan hain zuzen ere).

6.4.2 Aldagai-parametroak eta memori helbideak

`ParametroFormalenHelbideakA` programaren azpirrutinan aldagai-parametroak erabiliz honako kode hau izango genuke:

```
PROGRAM ParametroFormalenHelbideakVAR_A ;           { \TP70\06\HELBID3A.PAS }
USES
    Crt, Ikus ;      (* IKUS.TPU  unitatea darabil *)

PROCEDURE AzpirrutinaVAR (VAR Oso1, Oso2, Oso3 : INTEGER) ;
VAR
  BertakoA, BertakoB, BertakoC : INTEGER ;
BEGIN
   WriteLn ; WriteLn ;
   WriteLn ('ALDAGAIA', '       HELBIDEA', '   TAMAINA') ;
   WriteLn ('---------------------------------') ;
   WriteLn ('BertakoC -->   ':6, IntToHex (Seg (BertakoC)),':',
            IntToHex (Ofs (BertakoC)), '    (', SizeOf (BertakoC), ')') ;
   WriteLn ('BertakoB -->   ':6, IntToHex (Seg (BertakoB)),':',
            IntToHex (Ofs (BertakoB)), '    (', SizeOf (BertakoB), ')') ;
   WriteLn ('BertakoA -->   ':6, IntToHex (Seg (BertakoA)),':',
            IntToHex (Ofs (BertakoA)), '    (', SizeOf (BertakoA), ')') ;
   WriteLn ('Oso3 ------>   ':6, IntToHex (Seg (Oso3)),':',
            IntToHex (Ofs (Oso3)), '    (', SizeOf (Oso3),
            ')    erreferentziatuaren helbidea eta tamaina') ;
   WriteLn ('Oso2 ------>   ':6, IntToHex (Seg (Oso2)),':',
            IntToHex (Ofs (Oso2)), '    (', SizeOf (Oso2),
            ')    erreferentziatuaren helbidea eta tamaina') ;
   WriteLn ('Oso1 ------>   ':6, IntToHex (Seg (Oso1)),':',
            IntToHex (Ofs (Oso1)), '    (', SizeOf (Oso1),
            ')    erreferentziatuaren helbidea eta tamaina') ;
END ;

VAR
   ZbkOso1, ZbkOso2, ZbkOso3 : INTEGER ;

BEGIN              (* Programa Nagusia *)
   ClrScr ;
   WriteLn ('PARAMETRO ERREALEN ETA FORMALEN HELBIDEAK (VAR PASATZE ERAN)') ;
   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', '       HELBIDEA', '   TAMAINA') ;
   WriteLn ('---------------------------------') ;

   WriteLn ('ZbkOso3 --->   ':6, IntToHex (Seg (ZbkOso3)),':',
            IntToHex (Ofs (ZbkOso3)), '    (', SizeOf (ZbkOso3), ')') ;
   WriteLn ('ZbkOso2 --->   ':6, IntToHex (Seg (ZbkOso2)),':',
            IntToHex (Ofs (ZbkOso2)), '    (', SizeOf (ZbkOso2), ')') ;
   WriteLn ('ZbkOso1 --->   ':6, IntToHex (Seg (ZbkOso1)),':',
            IntToHex (Ofs (ZbkOso1)), '    (', SizeOf (ZbkOso1), ')') ;

   AzpirrutinaVAR (ZbkOso1, ZbkOso2, ZbkOso3) ;

END .
```

Irteeran ikus daitekeenez, `VAR` parametroen helbidea pantailaratu nahi denean Turbo Pascal lengoaiak programa nagusiko aldagai erreferentziaturen helbidea erakusten du nahiz eta bera pilan erreserbaturik izan. `AzpirrutinaVAR()` prozeduraren bertako aldagaientzat aldiz, lehengo `AzpirrutinaBaliozko()` bezala, pilako helbideak bistaratzen dira:



Inguruneaz baliatuz gero irteera baliokide aterako litzateke.

6.4.3 Konstante-parametroak eta memori helbideak

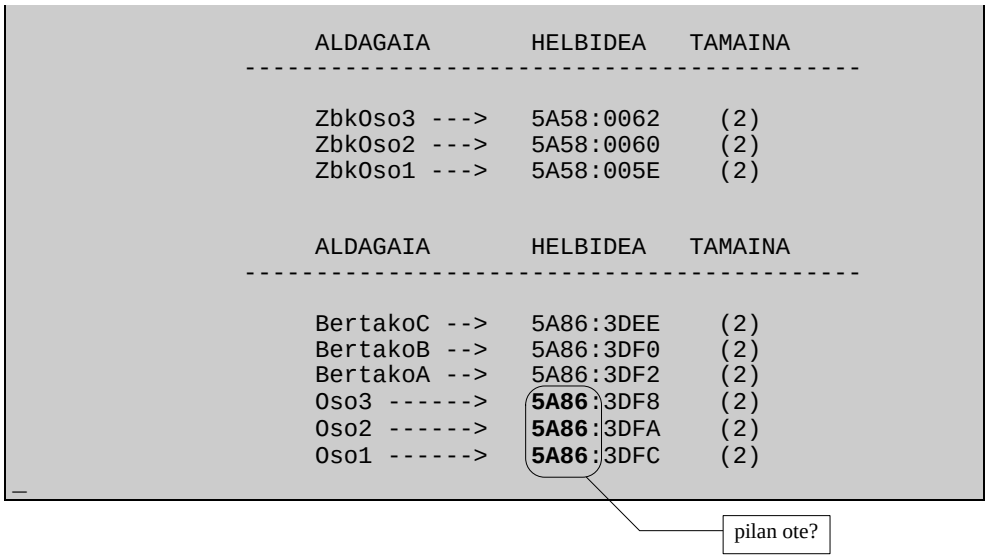
`ParametroFormalenHelbideakVAR_A` izeneko programan `VAR` pasatze era `CONST` pasatze moduagatik ordezkatuko bagenu, orain arte ikusitakoarekin erraza litzaiguke asmatzea programaren irteera zein izango litzatekeen. Konstante-parametro eta aldagai-parametro programen irteerak funtsean berdinak izan beharko lukete, izan ere kasu bietan parametro formal bakoitzean dagokion parametro errealaren erreferentzia bat gordetzen baita. Diferentzia parametro errealen babes mailan dago, kasu batean parametro errealak azpirrutina barruan alda daitezke eta bestean (konstante-parametroaren kasuan) guztiz babesturik daudela.

Ondorioz, `CONST` pasatze era erabiliz `ParametroFormalenHelbideak`**`CONST`**`_A` izeneko programa lortuko genuke, eta azpirrutinaren goiburukoa hau izango litzateke:

```
PROCEDURE AzpirrutinaCONST (CONST Oso1, Oso2, Oso3 : INTEGER) ;
```

Ondorioz irteera honelako zerbait atera beharko litzaiguke:

```
PARAMETRO ERREALEN ETA FORMALEN HELBIDEAK (CONST PASATZE ERAN)
============================================================

PILARI dagokion segmentuaren hasiera:  5A86
DATUEI dagokien segmentuaren hasiera:  5A58
KODEARI dagokion segmentuaren hasiera: 5942
```



Aproba eginez gero bertako aldagaiak eta parametro formalak pilan aurkitzen direla konturatuko gara, (`Oso1`, `Oso2` eta `Oso3` parametroak erreferentziak izan beharrean baliozkoak dirudite). Ateratzen dena esandakoarekin bat ez dator baina datu-mota egituratuak[15] ikasten jarri artean ezin izango dugu zergatiaren arrazoia argitu. Utz dezagun irekita arazoa, onarturik ileapaindegiko Franki Panki eta kaleko Patxi Panki pertsona bera direla eta `CONST` pasatze erako `ParametroFormalenHelbideakCONST_A` izena duen programaren irteera horrek beste interpretazioren bat duela (sakontzeko ikus **10.1.6 Arrayak parametro bezala** puntua).

6.5 ALDAGAIEN IRAUPENA ETA ALDAGAIEN ESPARRUA

Azpirrutinen kontzeptua azaldu dugunetik programa bat blokeka idatz daitekeela ikasi dugu, eta bloke batek (azpirrutina batek) beste azpiprograma bat barnera dezakeela aipatu izan da. Ondorioz bi motatako aldagaiak agertu zaizkigu, batetik programa nagusiko *aldagai orokorrak*, eta bestetik azpiprogrametan definiturik aurkitzen diren *bertako aldagaiak*.

Azter dezagun **6.5** puntu nagusi honetan aldagaiek dituzten ezaugarri bi: aldagaien iraupena eta aldagaien esparrua.

6.5.1 Aldagaien iraupena

Aldagaien iraupena kontzeptua aldagaia “bizirik” irauten duen denborarekin loturik dago, hots, memori erreserba egiten zaionetik memoria hori libre geratzen den arteko tartea.

Aldagaien iraupenari begiraturik programa nagusiko aldagaiek programaren iraupena izango dute (programa hasieran sortzen dira eta programaren exekuzioa amaitu arte bizirik irauten dute). Gainerako edozein aldagai, azpirrutinako edozein aldagai, menpeko bloke jakin bati loturik egongo da eta bloke hori exekutatzen hasten denean baino ez da sortuko bertako aldagaia, blokearen exekuzioa amaitzean bertako aldagaia memoriatik desagertuko da.

Iraupenak eragin handia du aldagaien hasieraketetan, programa nagusiko aldagaiek hasieraketa behin bakarrik duten bitartean, bertako aldagaiek berriz azpiprograma deitzen den bakoitzean hasieratuko dira.

[15] Ikasiko ditugun datu-mota egituratuak `String`, `Array`, `Set` eta `Record` dira.

Esate baterako ondoko programan bi azpirrutina ditugu, prozedura bat eta funtzio bat, programa exekutatzean memoriaren datu-segmentuan 6 byte erreserbaturik izango dira (bi `ZnbkNag`-rako eta lau `Emaitza`-rako). Gainerako aldagaiak (`ZnbkAzp`, `Kont`, `Fakt`, eta `Muga`) memoriaren pilan aurkituko dira baina ez denak aldiberean eta etengabe programaren exekuzioaren bitartean.

```
PROGRAM AldagaienIraupena ;              { \TP70\06\IRAUPEN.PAS }
VAR
   ZnbkNag : Integer ;
   Emaitza : LongInt ;

PROCEDURE DatuaIrakur (VAR ZnbkAzp : Integer) ;
BEGIN
   { 3. iruzkina }
   Write ('Zenbaki osoa eman: ') ;
   ReadLn (ZnbkAzp) ;
END;                          { DatuaIrakur()-ren amaiera }

FUNCTION Faktoriala (Muga : Integer) : LongInt ;
VAR
   Kont : Integer ;
   Fakt : LongInt ;
BEGIN
   Fakt := 1 ;
   { 5. iruzkina }
   DatuaIrakur (Muga) ;
   { 6. iruzkina }

   FOR Kont:=1 TO Muga DO
      Fakt := Fakt * Kont ;
   Faktoriala := Fakt ;
END;                           { Faktoriala()-ren amaiera }

BEGIN
   { 1. iruzkina }
   WriteLn ('Programa nagusia hastean  ZnbkNag=', ZnbkNag) ;
   { 2. iruzkina }
   DatuaIrakur (ZnbkNag) ;
   { 4. iruzkina }
   Emaitza := Faktoriala (ZnbkNag) ;
   { 7. iruzkina }
   WriteLn ('Programa nagusia amaitzean  ZnbkNag=', ZnbkNag) ;
   WriteLn ('Faktoriala--------------->  Emaitza=', Emaitza) ;
END.            { AldagaienIraupena-ren amaiera }
```

`AzpirrutinaBanatuak_Esparrua` programaren ondoko exekuzioaren irteera hau kontutan izanik, ikus dezagun pilaren edukia programaren mugarriak diren iruzkin bakoitzeko.

```
Programa nagusia hastean  ZnbkNag=0
Zenbaki osoa eman: 17
Zenbaki osoa eman: 5
Programa nagusia amaitzean  ZnbkNag=17
Faktoriala--------------->  Emaitza=120
_
```

Programaren lehen eginkizuna mezu bat pantailaratzea da, `ZnbkNag` zenbakizko aldagai orokorra delako `0`-ra hasieratuta egongo da. Jarraian `DatuaIrakur()` prozeduraren bitartez `ZnbkNag` aldagaian `17` gordetzen da, eta `Faktoriala()` funtzioaren deia burutzen da. `Faktoriala()` funtzioak berriro deitzen dio `DatuaIrakur()` prozedurari eta horren bitartez `Muga` aldagaiak `5` balioa hartzen du, kalkuluak burutu ondoren `Faktoriala()` funtzioak `120` itzultzen dio programa nagusiari. Programa bukatu aurretik `ZnbkNag` eta `Emaitza` aldagai banaren pantailaraketak egiten dira.

ikus dezagun pilaren edukia programaren mugarriak diren iruzkin bakoitzeko.

`1. iruzkina`

Programaren hasiera.

`2. iruzkina`

`DatuaIrakur()` deitu baino lehentxoago.

`3. iruzkina`

2. iruzkinan piztutako `DatuaIrakur()`-en.

`ZnbkAzp`
4. iruzkinaren helbidea

`4. iruzkina`

`DatuaIrakur()` deitu ondoren.

`5. iruzkina`

4. iruzkinan piztutako `Faktoriala()`-n.

`Fakt`
`Kont`
`Muga`
7. iruzkinaren helbidea

`3. iruzkina`

5. iruzkinan piztutako `DatuaIrakur()`-en.

`ZnbkAzp`
6. iruzkinaren helbidea
`Fakt`
`Kont`
`Muga`
7. iruzkinaren helbidea

`6. iruzkina`

4. iruzkinan piztutako `Faktoriala()`-n.

`Fakt`
`Kont`
`Muga`
7. iruzkinaren helbidea

`7. iruzkina`

`Faktoriala()` deitu ondoren.

1. eta 2. iruzkinak exekutatzen direnean pila hutsik dago.

3. iruzkina `DatuaIrakur()` prozeduraren deitu eta gero dago, beraz pilan bi datu egongo dira batetik zer agindutara (kode-segmentuaren helbideren bat) itzuli behar duen kontrola, eta, `ZnbkAzp` bertako aldagaiarentzat erreserba (aldagai-parametro bat denez gero bere edukia `ZnbkNag` erreferentziatuaren helbidea izango da). Laugarren iruzkinean pila hutsik egongo da ere.

5. iruzkinan pilan ondoko datuak gordeko dira: programaren sekuentzia mantentzen dela segurtatzeko 7. iruzkinaren helbidea, `Muga` parametroa (`ZnbkNag`-k duen balioaren kopia gordeko da pilan) eta bertako bi aldagaiak (`Kont` eta `Fakt`).

3. iruzkinan oraindik ez da `Faktoriala()` funtzioa amaitu eta `DatuaIrakur()` deitzen da bigarrenez. Ondorioz pilan dagoena ezabatu gabe lehenagoko 3. iruzkinan egindako erreserba biak burutzen dira baina gordetzen diren datuak hauek direla: programaren sekuentziak `Faktoriala()` funtzioan jarraitu behar duenez 6. iruzkinaren helbidea, eta, `ZnbkAzp` aldagai-parametroak `Muga`-ren helbidea izango du.

6. iruzkina gertatzen denean pilaren egoera 5. iruzkinako berbera izango da.

7. iruzkinan pila hutsik dago.

6.5.2 Aldagaien esparrua

Aldagai baten esparrua bestalde, aldagaiaren izenari dagokion ezagutza-eremuarekin lotuta dago, hau da, aldagaiaren identifikadorea programaren zein bloketan ezaguna den konpiladorearentzat.

Esate baterako, ondoko programan `ZnbkAzp` eta `ZnbkNag` identifikadoreak dituzten bi aldagai daude, bat azpirrutinakoa eta bestea programa nagusikoa (biak `Integer` datu-motatakoak). Iraupenari buruz esan dezakeguna hau da, datu-segmentuan bi byte hartzen dira programa nagusiko `ZnbkNag` aldagaiarentzat eta ez da memoria gehiago beharko `Azpirrutina` prozedura aktibatu arte. Programaren kontrola `Azpirrutina` prozeduran dagoenean aldagaientzat lau zortzikote hartzen dira, datu-segmentuko lehengo biak gehi pilako beste bi byte `ZnbkAzp`-rentzat. Ondorioz, `ZnbkAzp` bertako aldagaiaren iraupena prozedurarena da, eta `ZnbkNag` aldagaierena `EsparruaAztertzen0` programarena.

```
PROGRAM EsparruaAztertzen0 ;          { \TP70\06\ESPARRU0.PAS }
VAR
   ZnbkNag : Integer ;

PROCEDURE Azpirrutina ;
VAR
   ZnbkAzp : Integer ;
BEGIN
   WriteLn ('Azpirrutina barruan  ZnbkNag=', ZnbkNag) ;
   WriteLn ('Azpirrutina barruan  ZnbkAzp=', ZnbkAzp) ;
END;

BEGIN
   WriteLn ('0 programa nagusian  ZnbkNag=', ZnbkNag) ;
 { WriteLn ('0 programa nagusian  ZnbkAzp=', ZnbkAzp)  ; }

   Azpirrutina ;
END.
```

derrigorrez

Esparruari buruz aritu aurretik, konturatu gaitezen `EsparruaAztertzen0` programan `ZnbkNag` aldagaia azpirrutina baino lehenago erazagututa dagoela. Aldagai jakin bat non erazagutzen den horrek mugatzen du bere esparrua, arau bezala aldagai bat erazagutu den blokean eta bere barneko blokeetan ezaguna izango da. Horrela `ZnbkNag` aldagaia programaren hasieran deklaratu delako `Azpirrutina` prozeduran eta programa nagusian ezaguna da, bere esparrua programa osoarena delako aldagai globala edo *orokorra* esaten zaio. Baina azpirrutinaren blokean definituriko `ZnbkAzp` aldagaia ezin da bloke horren kanpoan atzitu, ezezaguna delako bere esparrua `Azpirrutina` prozedurarena delarik.

Hauxe litzateke `EsparruaAztertzen0` programaren irteera bat:

```
0 programa nagusian  ZnbkNag=0
Azpirrutina barruan  ZnbkNag=0
Azpirrutina barruan  ZnbkAzp=22782
_
```

Non, gauza bi azpimarratuko genituzke. Bat, aldagai orokorrarentzat konpiladoreak hasieraketa bat jartzen duela (zenbakizko aldagaiak zeroz hasieratzen dira); baina bertako aldagaiarentzat ez da horrelakorik egiten, eta bere edukia pantailaratzean aldagaiari dagokion memoriko posizioan une horretan dagoena agertuko da (esate baterako `22782`). Bigarren oharra `ZnbkNag` aldagai orokorra edonon atzitzeko aukera dugula, ondorioz moduluen arteko komunikazioa parametroen bitartez ezezik aldagai orokorren bidez egiterik ere posible[16] da.

[16] Posible izateak ez du esan nahi komenigarria denik. Areago, ez dugu testu honetan aldagai orokorretan oinarritzen den moduluen arteko komunikazioa onartuko.

`EsparruaAztertzen0` programan `ZnbkNag` aldagaiaren erazagupen tokia aldatuz gero `EsparruaAztertzen1` programa lortuko genuke, zeinean `ZnbkAzp` aldagaiaren esparrua lehen bezala azpiprogramarena den baina `ZnbkNag` aldagaiarena programa nagusiko sententzien gorputzera (`BEGIN` eta `END.` arteko tartera) mugatzen den:

```
PROGRAM EsparruaAztertzen1 ;        { \TP70\06\ESPARRU1.PAS }

PROCEDURE Azpirrutina ;
VAR
   ZnbkAzp : Integer ;
BEGIN
 { WriteLn ('Azpirrutina barruan  ZnbkNag=', ZnbkNag) ; }
   WriteLn ('Azpirrutina barruan  ZnbkAzp=', ZnbkAzp) ;
END;

VAR
   ZnbkNag : Integer ;
BEGIN
   WriteLn ('1 Programa nagusian  ZnbkNag=', ZnbkNag)  ;
 { WriteLn ('1 Programa nagusian  ZnbkAzp=', ZnbkAzp) ; }

   Azpirrutina ;
END.
```

derrigorrez

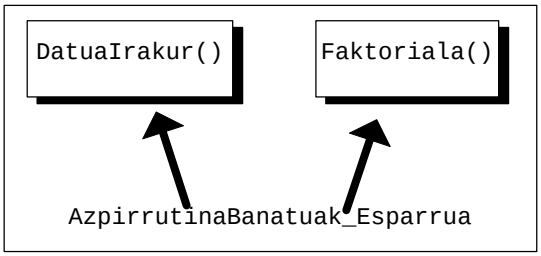
Kasu honetan `ZnbkNag` aldagaia ezezaguna da `Azpirrutina` izeneko prozeduran, ondorioz programaren irteera honelako zerbait izango litzateke:

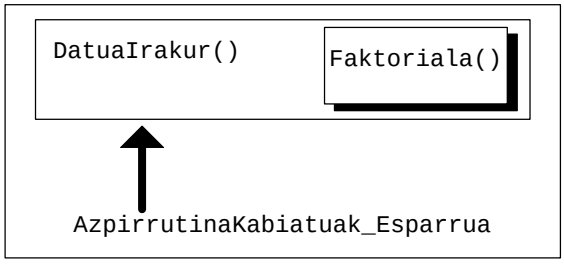
```
1 programa nagusian  ZnbkNag=0
Azpirrutina barruan  ZnbkAzp=22782
_
```

Aldagaiak tokiz mugitzea ezezik azpirrutinak ere programaren leku ezberdinetan defini daitezke. Ondoko bi adibideetan emaitza berdina lortzen duen programak ematen dira, baina batean azpirrutinak banaturik dauden bitartean, bigarrenean bat bestearen barnean kabiatu egin ditugu. `AzpirrutinaBanatuak_Esparrua` eta `AzpirrutinaKabiatuak_Esparrua` programen irteeraren bat honako hau izan daiteke, non datua den zenbakiaren irakurketa eta faktorialaren kalkulua azpirrutina bitan egiten diren:

```
Programa nagusia hastean    ZnbkNag=0
Zenbaki osoa eman: 7
7! = 5040
Programa nagusia amaitezan  ZnbkNag=7
_
```

`AzpirrutinaBanatuak_Esparrua` eta `AzpirrutinaKabiatuak_Esparrua` programen desberdintasuna azpirrutinen barne antolaketan datza. Izan ere, lehenengoan `DatuaIrakur()` eta `Faktoriala()` zuzenki programa nagusiaren menpekoak dira, eta, bigarrenean berriz `DatuaIrakur()` prozedura da programa nagusitik atzi daitekeen bakarra eta `Faktoriala()` izeneko funtzioaren pizteak `DatuaIrakur()` prozeduraren bitartekaritza behar du.





Hauek dira `AzpirrutinaBanatuak_Esparrua` eta `AzpirrutinaKabiatuak_Esparrua` programen kodeak:

```
PROGRAM AzpirrutinaBanatuak_Esparrua ;
VAR
   ZnbkNag : Integer ;
   Emaitza : LongInt ;

PROCEDURE DatuaIrakur (VAR ZnbkAzp : Integer) ;
BEGIN
   Write ('Zenbaki osoa eman: ') ;
   ReadLn (ZnbkAzp) ;
END;               { DatuaIrakur()-ren amaiera }

FUNCTION Faktoriala (Muga : Integer) : LongInt ;
VAR
   Kont : Integer ;
   Fakt : LongInt ;
BEGIN
   Fakt := 1 ;
   FOR Kont:=1 TO Muga DO
      Fakt := Fakt * Kont ;
   Faktoriala := Fakt ;
END;                { Faktoriala()-ren amaiera }

BEGIN
   WriteLn ('Hastean    ZnbkNag=', ZnbkNag) ;
   DatuaIrakur (ZnbkNag) ;
   Emaitza := Faktoriala (ZnbkNag) ;
   WriteLn (ZnbkNag,'! = ', Emaitza) ;
   WriteLn ('Amaitzean  ZnbkNag=', ZnbkNag) ;
END.            { AzpirrutinaBanatuak_Esparrua }
```

```
PROGRAM AzpirrutinaKabiatuak_Esparrua ;
VAR
   ZnbkNag : Integer ;

PROCEDURE DatuaIrakur (VAR ZnbkAzp : Integer) ;
VAR
   Emaitza : LongInt ;
FUNCTION Faktoriala (Muga : Integer) : LongInt ;
VAR
   Kont : Integer ;
   Fakt : LongInt ;
BEGIN
   Fakt := 1 ;
   FOR Kont:=1 TO Muga DO
      Fakt := Fakt * Kont ;
   Faktoriala := Fakt ;
END;                { Faktoriala()-ren amaiera }

BEGIN
   Write ('Zenbaki osoa eman: ') ;
   ReadLn (ZnbkAzp) ;

   Emaitza := Faktoriala (ZnbkAzp) ;
   WriteLn (ZnbkNag,'! = ', Emaitza) ;
END;               { DatuaIrakur()-ren amaiera }

BEGIN
   WriteLn ('Hastean    ZnbkNag=', ZnbkNag) ;
   DatuaIrakur (ZnbkNag) ;
   WriteLn ('Amaitzean  ZnbkNag=', ZnbkNag) ;
END.           { AzpirrutinaKabiatuak_Esparrua }
```

`AzpirrutinaBanatuak_Esparrua` eta `AzpirrutinaKabiatuak_Esparrua` programen aldagaiek dituzten ezagutza-eremuak desberdinak dira, ondoko tauletan biltzen dira modulu bakoitzeko identifikadore guztien esparrua. Aldagai baten esparruaren adierazpidea gezi batez gauzatu dugu, geziaren punta gabeko muturrak aldagaia non deklaratu den adierazten du eta geziaren puntak aldagaiaren ezagutza-eremua noraino zabaltzen adierazten du.

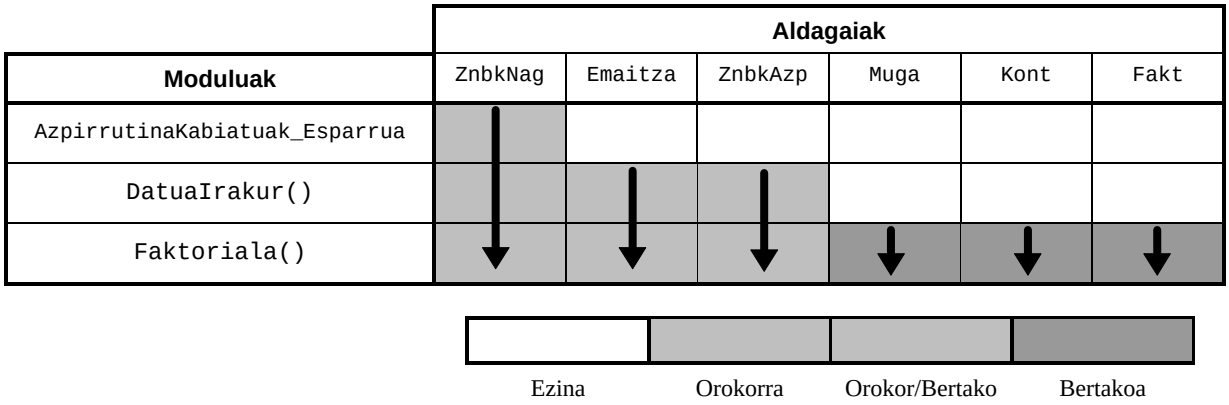
`AzpirrutinaBanatuak_Esparrua` programari dagokion esparru-taula:

	Aldagaiak					
Moduluak	`ZnbkNag`	`Emaitza`	`ZnbkAzp`	`Muga`	`Kont`	`Fakt`
`AzpirrutinaBanatuak_Esparrua`	↓	↓				
`DatuaIrakur()`			↓			
`Faktoriala()`	↓	↓		↓	↓	↓

Ezina Orokorra Bertakoa

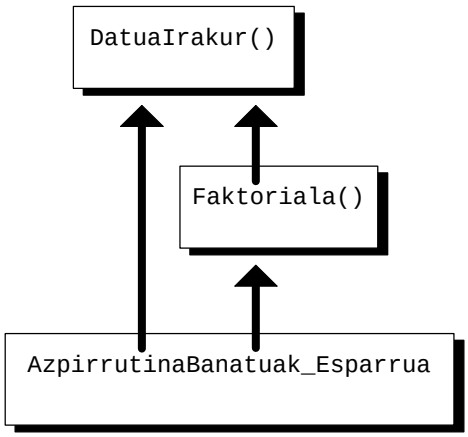
`AzpirrutinaBanatuak_Esparrua` programaren aldagaiak bi mailetan antolatzen dira, programaren hiru modulutan erabil daitezkeen `ZnbkNag` eta `Emaitza` aldagai orokorrak, eta, azpirrutina bakoitzaren bertako edo pribatuak diren bertako aldagaiak. Aldagairen bat erreferentzia modulu batean ezinezkoa denean taularen laukia hutsik agertzen da.

`AzpirrutinaKabiatuak_Esparrua` programaren aldagaien mailaketa konplexuagoa da, lehen bezala programaren hiru modulutan erabil daitekeen `ZnbkNag` aldagaia orokorra izango da, baina `DatuaIrakur()` prozeduran bertakoa den `Emaitza` aldagaia `Faktoriala()` funtzioan orokorra izango litzateke. Lehen bezala `Faktoriala()` funtzioak bertako hiru aldagai berberak ditu (`Muga`, `Kont` eta `Fakt`):

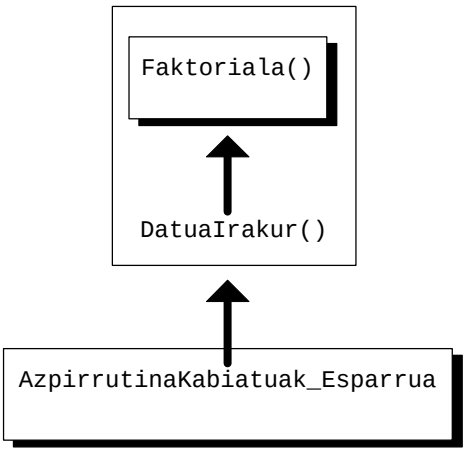


Ikus daitekeenez azpirrutinak kabiatzen direnean gerta daiteke aldagairen bat aldiberean pribatua eta orokorra izatea, `Emaitza` aldagaia esate baterako, kabiaturiko modulu edo moduluetako orokorra izango da eta kanpoko moduluek ezin izango dute atzitu pribatua delako eurentzat.

Aldagaientzat esandakoa moduluen identifikadorei zabal daiteke. Horrela, azpirrutina banatuetan komunikazioa ondoko eskeman erakusten da, non, azpiprogramaren bat deitu ahal izateko deiaren aurkitzen den modulua baino lehen definiturik egon behar den:



`AzpirrutinaBanatuak_Esparrua` moduluak, behar izanez gero, `DatuaIrakur()` prozedura eta `Faktoriala()` funtzioa dei ditzake. Modu berean, `Faktoriala()` funtzioak bera baino lehenago definiturik dagoen `DatuaIrakur()` modulua dei dezake.



`AzpirrutinaKabiatuak_Esparrua` deituriko moduluak `DatuaIrakur()` prozedura dei dezake, baina `Faktoriala()` funtzioarekin ezin da zuzenki komunikatu `DatuaIrakur()`-ren modulu pribatua delako. `DatuaIrakur()` izeneko moduluak `Faktoriala()` dei dezake.

Beraz modulu bat deitua ahal izateko gaitasuna, bere ezagutza-esparrua edo eremua, programaren iturburu-kodean modulua non idatzi denarekin lotuta dago. Arau orokor bezala azpirrutina bat eskuragarri izateko, deia gertatzen den baino lehen definiturik egongo da eta modulu deitzailea eta modulu deituaren arteko kabiaketa maila jauzi bakar batez neurtu ahal izango da.

6.5.3 Identifikadoreen lehentasuna eta ustegabeko gertaerak

Esparruaren kontzeptuarekin batera zalantza bat sor daiteke bloke batean identifikadre berdina duten aldagai bi aurkitzen direnean. Demagun adibidez, honako programa daukagula, non programa nagusiak definituriko `Kont` aldagaia agertzen den baina `Kont` identifikadore berbera azpirrutinan ere deklaratu den:

```
PROGRAM IdentifikadoreenLehentasuna ;
VAR
   Kont : Integer ;

   PROCEDURE Pantailaraketa ;
   VAR
      Kont : Integer ;
   BEGIN
      Write ('Azpirrutinan-->  ') ;
      FOR Kont:=1 TO 3 DO
        Write ('Kont=', Kont, '  ') ;
   END;  { Pantailaraketa-ren amaiera }

BEGIN
   Kont := 7 ;
   WriteLn ('Hastean------->  Kont=', Kont) ;
   Pantailaraketa ;
   WriteLn ;
   WriteLn ('Amaitzean----->  Kont=', Kont) ;
END.
```

Posible ote da horrelako identifikadore bikoiztuak erabiltzerik?. Bai, eta memori helbide ezberdin bi izango lirateke.

`Kont` aldagai biak atzigarriak dinenez gero, `Write` horrek zein hartzen du aintzat?

Programa nagusiaren esleipen eta `WriteLn`-etan zalantzarik ez dago, azpirrutinako `Kont` ezezaguna baita.

Aurreneko galderari dagokion erantzuna baiezkoa da, hots, modulu kabiatu batean edozein identifikadore definitzerik bada (kanpoko moduluan erabiltzen ari bada ere). Horrez gero, `Pantailaraketa` azpirrutinaren `FOR` aginduari loturik aurkitzen diren esleipena eta `Write` zalantzagarriak izango lirateke, izan ere `Kont` aldagai orokorra eta `Kont` bertako aldagaia biak atzigarriak baitira. Lehentasuna bertako aldagaiak balu ezkerreko irteera izan beharko litzateke, lehentasuna aldagai orokorrari balegokio aldiz eskuinekoa:

```
Hastean------->  Kont=7
Azpirrutinan-->  Kont=1  Kont=2  Kont=3
Amaitzean----->  Kont=7
_
```

```
Hastean------->  Kont=7
Azpirrutinan-->  Kont=1  Kont=2  Kont=3
Amaitzean----->  Kont=3
_
```

Dagoen bezala `IdentifikadoreenLehentasuna` programa exekutatzean ezkerreko irteera ateratzen dela kontura gaitezke, eta `Pantailaraketa` prozedurari dagokion `Kont` aldagaiaren erazagupena kentzean eskuinekoa. Beraz, modulu ezberdinetan berberak diren bi identifikadore definiturik daudenean (modulu berean etiketa berdinak erabiltzea ezinezkoa baita), moduluak banatuak badira zalantzarik ez dago eta identifikadore bakoitzak duen esparrua bere moduluarena da, baina modulu kabiatuak izatean barnerago dagoen identifikadoreak lehentasuna izango du kanporago dagoenaren aurrean.

Esandako honekin batera programazio arazo latzak gerta daitezke eta euren kalteak ekidin nahirik aldagai orokorren erabilpena mugatzen saiatuko gara. Esate baterako, aipatu bezala `IdentifikadoreenLehentasuna` programan, `Pantailaraketa` prozeduraren `Kont` aldagaiaren definizioa kenduko bagenu, programa konpilagarria izango litzateke eta aldaketa txiki horrek eragingo lukeen irteera lehenago eskuinean erakutsi duguna izango litzateke.

Demagun `AldagaiOrokorra` izeneko programa batean `Znbk` aldagai orokor bat dugula eta bere balioa teklatuz irakurri egiten dela, ondoren eta konturatu gabe `Azpirrutina` prozeduran aldagai horren balioa aldatu egiten dela, eragina programa nagusian nozitzen

denez aurrerantzean `Znbk` aldagaiak balio desegokia izango du. Oso arriskutsua izan daiteke orokorra den aldagaia azpirrutina batean atzitzea bertan, ustegabe, aldatua izan daitekeelako:

```
PROGRAM AldagaiOrokorra ;              { \TP70\06\ESPARRU5.PAS }
VAR
   Znbk : Integer ;

   PROCEDURE Azpirrutina ;
   BEGIN
      WriteLn ('Azpirrutinan sartzean---->  Znbk=', Znbk) ;
      Znbk := -123 ;   { Konturatu gabe }
      WriteLn ('Azpirrutinatik irtetean-->  Znbk=', Znbk) ;
   END;  { Azpirrutina-ren amaiera }

BEGIN
   Write ('Zenbaki oso bat eman: ') ;
   ReadLn (Znbk) ;
   WriteLn ('Hastean------------------>  Znbk=', Znbk) ;
   Azpirrutina ;
   WriteLn ('Sarrerako zenbakia------->  Znbk=', Znbk) ;
END.
```

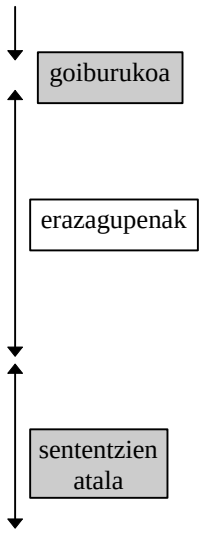
Hauxe da `AldagaiOrokorra`-ri dagokion irteeraren bat:

```
Zenbaki oso bat eman: 69
Hastean------------------>  Znbk=69
Azpirrutinan sartzean---->  Znbk=69
Azpirrutinatik irtetean-->  Znbk=-123
Sarrerako zenbakia------->  Znbk=-123
_
```

Horren sinplea ematen duen errore honek buruhauste handiak ekartzen dizkio programatzen duen orori, eta horregatik arau bezala aldagai orokorrak azpirrutinen barneetan ez erabiltzea aholkatzen da (azpirrutina batean programa nagusiko balioren bat behar izanez gero parametro bezala pasatuko zaio). Gero programa behar bezala ibiltzen ez bada eta aldagai baten errua izan daitekeela somatzen badugu, bere arazketa askoz errazago egin ahal izango da azpirrutina guztiak ez direlako aztertu beharko.

Arazoak baztertu nahirik programa nagusiren aldagaiak orokorrak ez izatea proposatzen dugu, (programa nagusian ezagunak baina ez azpiprogrametan). Horretarako aski da aldagaiak azpirrutinen ostean eta aginduen gorputza baino lehen erazagutzea. `AldagaiOrokorra` izeneko programa honela berridatziz ezinezkoa litzateke ustegabeko `Znbk`-ren aldaketa suertatzea:

```
PROGRAM AldagaiOrokorra ;        { \TP70\06\ESPARRU5.PAS }

   PROCEDURE Azpirrutina ;

VAR
   Znbk : Integer ;

BEGIN
   Write ('Zenbaki oso bat eman: ') ;
   ReadLn (Znbk) ;
   WriteLn ('Hastean------------------>  Znbk=', Znbk) ;
   Azpirrutina ;
   WriteLn ('Sarrerako zenbakia------->  Znbk=', Znbk) ;
END.
```



6.6 AZPIPROGRAMA MOTAK TURBO PASCAL LENGOAIAN

Programen modulaketa lantzen duen seigarren kapitulu honentzat aukeratu dugun izenburua **Azpiprogramak: funtzioak eta prozedurak** izan da, horiek baitira Turbo Pascal lengoaiak onartzen dituen azpirrutina motak. Funtzio eta prozeduraren arteko ezberdintasuna kontzeptuan baino praktikan ematen da, funtzio bat emaitza bakar bat duen kalkulu bati loturik doan modulua litzateke, eta, prozedura aldiz kalkulua baino eginkizuntzat hartuko dugu zeinek modulu deitzaileari emaitzik itzul diezaiokeen ala ez.

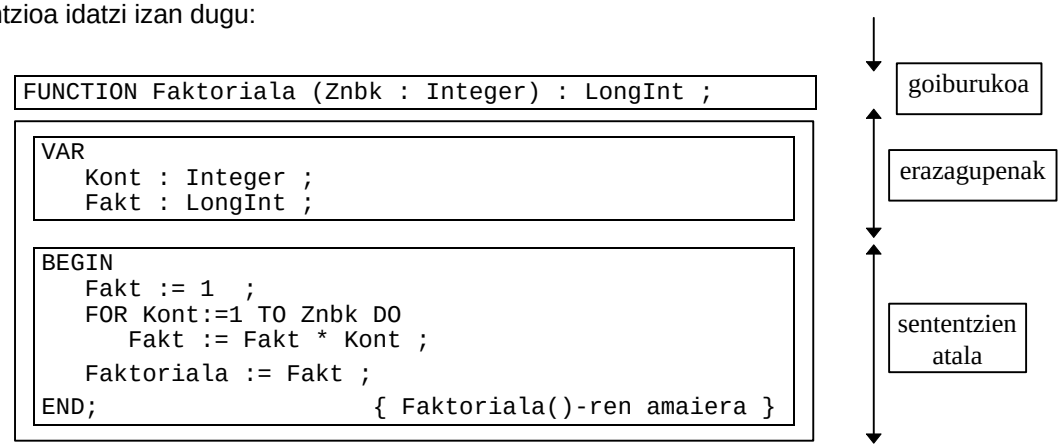
Prozedura bat, beraz, eginkizun jakin bat burutzeko sententzien multzoa litzateke (pantaila garbitzea, balioak bistaratzea, datuak aldagaietan gordetzea, ...). Prozedura batek behar izan dezake sarrerako parametrorik (baliozko parametroak edo eta konstante-parametroak), eta irteerarik kanporatu behar izanez gero aldagai-parametroak eduki ditzake ere.

Funtzio batek izango duen helburua balio zehatz eta bakar bat modulu deitzaileari itzultzea izango da (faktorialaren kalkulua, karaktere baten ordinala, menu baten aukera ...). Askotan funtzioak kanpoko datuak behar izango ditu, baliozko parametro edo konstante-parametroak izango direnak, bainan gure irizpidearen arabera ez du aldagai-parametrorik edukiko eta bere emaitza itzultzeko berezko mekanismo propioaz baliatuko da.

Bainan Turbo Pascal-aren azpiprograma mota biak banan-banan azter ditzagun.

6.6.1 Funtzioak

Gure prometan funtzioak dagoenekoz erabili izan ditugu, esate baterako duela gutxi **6.5.1 Aldagaien iraupena** puntuan zenbaki oso baten faktoriala lortzen zuen honako funtzioa idatzi izan dugu:



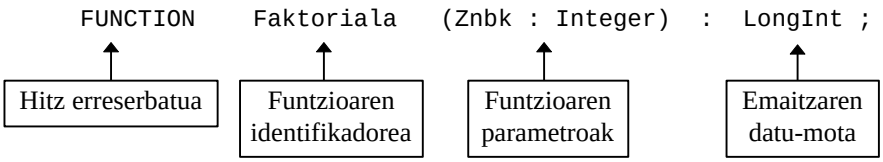
Funtzioaren egitura, ikus daitekeenez, Turbo Pascal programa baten egitura berbera da (laugarren kapituluko **4.4 PROGRAMA BATEN EGITURA** gogoratu), kasu honetan deklarazioak `Faktoriala()` funtzioaren sententzien atalan ezagunak izango dira (funtzioaren sententzien atalan eta izango lituzkeen beste bloke kabiatuetan).

6.6.1.1 Funtzioaren atalak

Edozein funtziok dituen hiru zatiak jarraian zehaztasunez azaltzen dira.

6.6.1.1.1 Funtzioaren goiburukoa

`Faktoriala()` funtzioa adibidetzat harturik bere goiburukoan lau elementu agertzen dira:



Funtzio baten blokea hasten dela adierazteko Pascal lengoaian `FUNCTION` marka jarri behar da, eta bere ondoren funtzioa etiketatzen duen izen bat dator (funtzioaren identifikadore bat finkatzerakoan **4.2.2.2 Erabiltzailearen identifikadoreak** puntuan emaniko arauak gogora ekar).

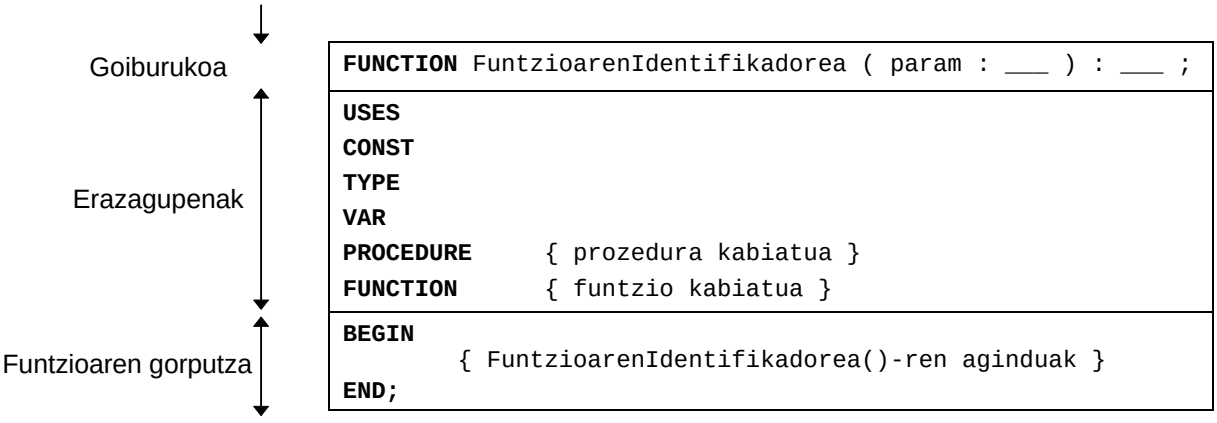
Funtzioaren identifikadoreak berebiziko garrantzia izango du, eta bi zereginetarako balio duela esan genezake. Batetik funtzioa bera deitu ahal izateko (dagoenekoz ez da guretzeat berria edozein azpiprograma pizteko bere etiketa behar dela); baina bestetik badu etiketa horrek funtzioentzat berezia den helburua, hots, funtzioak modulu deitzaileari itzuliko dion emaitza bideratzeko tresna izatea.

Funtzioak kanpoko daturik behar izango balu, modulu deitzaileak parametroen bitartez hornituko dio, eta, parametroaren datu-mota zehazten delarik parentesi artean mugatuko da goiburukoan. Adibidez `Faktoriala()` funtzioak `Znbk` kopuru osoaren faktoriala lortu behar duenez zenbakiaren balioa jaso behar du.

Funtzioaren goiburukoa puntu eta komaz bukatu baino lehen funtzioak itzultzen duen emaitzaren datu-mota idazten da. Pascal lengoaiak aldagai eta parametroen erazagupenetan daukan ohitura bera mantenduz datu-motaren baino lehentxoago bi puntuak jarriko dira. Gure adibideko funtzioaren emaitza `Faktoriala` etiketarekin elkartuko dugu zein `LongInt` bezala definitu den.

6.6.1.1.2 Funtzioaren erazagupenak

Funtzio gehinek aldagai laguntzaileak behar izaten dituzte euren zereginak bete ahal izateko, halakoetan ez dira aldagai orokorrak erabiliko bertako aldagai pribatuak baizik. Faktoriala kalkulatzen duen gure adibidean `FOR` sententziak behar duen `Kont` kontagailua atal honetan erazagutuko da, gauza bera egingo da derrigorrezkoa den `Fakt` aldagai metatzailearekin. Funtzioaren bigarren atal honetan aldagaiak ezezik unitateak, konstanteak, datu-motak eta azpirrutina kabiatuak ere erazagut daitezke:



Funtzioaren goiburukoa esplikatzean parametroei buruzkoa azaletik aipatu duguna berriro errepikatuko dugu orain, hots, Turbo Pascal lengoaiari dagokiola funtzio batek onar ditzakeen parametro motak ikasitako hirurak dira (baliozko parametroak, aldagai-parametroak eta konstante-parametroak) baina gure iritsirako, eta beste autore zein programadere batzuk ez dira behar bada bat etorriko, funtzioetan sarrerako parametroak[17] erabiliko dira eta sekulan aurretik VAR bereizgarririk duen aldagai-parametrorik.

Funtzioaren emaitza modulu nagusiari itzultzeko, irteerako parametroaren ordez, funtzioak berezia duen mekanismoaz baliatuko gara, funtzioaren identifikadorea inplizitoki aldagai-parametrotzat hartzea alegia. Ondorioz, eta ongi ezagutzen dugun, `Faktoriala()` funtzioaren adibideari begiraturik `Faktoriala` identifikadorea bi modutan uler daiteke:

1. Faktoriala kalkulatzen duen modulua pizteko erabili behar den izena, halakotan `Faktoriala` identifikadorearekin batera datua den zenbaki osoa zehaztu beharko da. `Faktoriala()` funtzioaren aktibaketa burutu ahal izateko bere deia programa nagusian aurkituko da, edo bestela `Faktoriala()` definitu den tokitik beherago dauden azpirrutinetan (funtzio batek bere buruari errekurtsiboki dei diezaiokeenez bere aktibaketa `Faktoriala()` funtzioan bertan egon daiteke ere)
2. `Faktoriala` identifikadorea parentesi gabe agertzen denean ez da azpirrutinaren deia[18] egiten ari. Kasu horretan `Faktoriala` identifikadore hori, balio bat pilatzen duen aldagaia litzateke, `Faktoriala()` izeneko funtzioak itzuliko duen emaitzaren lau[19] zortzikoteko gordelekua hain zuzen ere. Modulu deitzaileari emaitza itzultzen dion `Faktoriala` identifikadorea parentesi gabe dator eta `Faktoriala()` funtzio barruan baino ez da ezagutzen.

Programa batean `Faktoriala` identifikadorearen agerpenari dagozkion interpretazio biak hauek lirateke:

```
WriteLn ( Faktoriala(5) ) ;
```

`5` zenbakiaren faktoriala kalkulatu eta pantailaratu

```
Faktoriala := 120 ;
```

Dagoenekoz `Faktoriala()` funtzioak `120` itzuliko du

Eskuineko kasuan `Faktoriala()` funtzioaren exekuzioa hor bertan amaituko balitz modulu deitzaileak `120` hartuko luke emaitzat, kontutan izan `Faktoriala()` funtzioaren barruan erabiltzen den `Faktoriala` izeneko aldagaia berezia dela eta berarekin zentzu bateko esleipenak baino ezin daitezkeela egin (balioren bat hartzen duen zentzuko esleipena).

6.6.1.1.3 Funtzioaren sententzien atala

Funtzioak bete behar duen helburua mamitzeko derrigorrezkoak diren aginduak idatziko dira atal honetan. `Faktoriala()` izeneko funtzioaren harira joz, berresan eta errepikatuko dugu funtzioaren atalan, besteak beste, honelako agindu bat behintzat izanen dela:

```
Faktoriala := [          ] ;
```

Funtzioak irteera bat du eta onartzen dugun komunikazio bide bakarra hori baita.

[17] Datua balio bakar bat denean baliozko parametroa izango da eta datuak balio bat baino gehiago biltzen duenean (aurrerago zehaztuko ditugun datu-mota egituratuak) konstante-parametroz elikatuko zaio informazioa funtzioari.

[18] `Faktoriala()` adibidean gauzak horrela dira, baina orokorrean salbuespenak egon daitezke: funtzioren batek parametrorik ez badu orduan funtzio horren deian ez da parentesirik idatziko.

[19] Goiburukoan definitu den emaitzaren datu-mota `LongInt` denez memorian 4 byte beharko dira.

6.6.1.2 Funtzioaren deia

Funtzio baten helburua balio bat lortzea denez gero, funtzioaren deia balio hori ager daitekeen tokietan kokatu ahal izango da. Funtzio baten deia egiteko funtzioa identifikatzen duen etiketa idatziko da, eta bere ostean funtzioak behar dituen parametroak parentesi artean jarriko dira (funtzioak parametrorik ez duenean ez dira parentesiak idatziko).

Funtzioaren deian dauden parametroak uneko parametro edo parametro errealak lirateke, uneko parametroak parametro formalekin elkartzen direnez **6.6.1.1.1 Funtzioaren goiburukoa** eta **6.6.1.1.2 Funtzioaren erazagupenak** puntuetan aipatu duguna gogora ekarriko dugu. Hots, funtzio baten parametro formalak baliozko parametroak edo/eta konstante-parametroak izango direla baina ez aldagai-parametroak. Baliozko parametroei dagozkien parametro errealak konstanteak, aldagaiak, adierazpenak edo funtzioak izango dira **6.3.3.1** puntua gogoratu. Erreferentziazkoak diren konstante-parametroak ez ditugu oraindik kurtsoan sistematikoki aplikatu datu-mota egituratuetarako erabiltzen direlako.

Adibide baten bitartez `Faktoriala()` funtzioaren dei ezberdinak ikusi ahal izateko jarraian erakusten den `FuntzioarenDeiak` programa prestatu izan dugu:

```
PROGRAM FuntzioarenDeiak ;                    { \TP70\06\FUNTZDEI.PAS }

FUNCTION Faktoriala (Znbk : Integer) : LongInt ;
VAR
   Kont : Integer ;
   Fakt : LongInt ;
BEGIN
   Fakt := 1  ;
   FOR Kont:=1 TO Znbk DO
      Fakt := Fakt * Kont ;
   Faktoriala := Fakt ;
END ;                  { Faktoriala()-ren amaiera }

VAR
  Datua : Integer ;
  Emaitza : LongInt ;
BEGIN
   WriteLn ('1. deian   4!=', Faktoriala(4) ) ;              { 1. deia }

   Write ('Zenbaki natural bat eman: ') ;
   ReadLn (Datua) ;
   Emaitza := Faktoriala(Datua) ;                            { 2. deia }
   WriteLn ('2. deiaren ondoren   ', Datua, '!=', Emaitza) ;

   Write ('3. deian   ', Emaitza DIV (Datua*Datua), '!-->') ;
   IF Faktoriala(Emaitza DIV (Datua*Datua)) < 0 THEN         { 3. deia }
        WriteLn ('gainezkada') ;
   ELSE WriteLn ('zalantzazkoa') ;

   Write ('Zenbaki negatibo bat eman: ') ;
   ReadLn (Datua) ;
   Emaitza := Faktoriala( Abs(Datua) ) ;                     { 4. deia }
   WriteLn ('4. deiaren ondoren   ', Abs(Datua), '!=', Emaitza) ;
END.
```

`Faktoriala()` funtzioak duen parametro bakarrerako baliozko pasatze era aukeratu da `Znbk` sarrerako datua delako, horrez gero programa nagusian egin daitekeen edozein deitan agertuko den parametro erreala ondoko lau aukeretatik bat izan ahalko da.

1. deian uneko parametroa `4` konstantea da eta `Faktoriala(4)` funtzioaren deia `WriteLn` prozedura estandarren parametro erreala litzateke. Honez gero, lehenik `1. deian  4!=` karaktereen kate konstantea pantailaratzen da eta ondoren `4`-ren faktoriala kalkulatu eta emaitza den `24` kopurua erakusten da

2. deian uneko parametroa `Datua` aldagaia da eta sarrerakoa izan behar duenez `Faktoriala(Datua)` funtzioaren deia baino lehen baliorik izan behar duenez teklaturen bitartez irakurtzen da. Bigarren dei honen moldea aldatu egin da funtzioaren emaitza `LongInt` zenbaki bat denez datu-mota horretako `Emaitza` izeneko aldagaian jaso dugu

3. deian uneko parametroa adierazpena da (adierazpen aritmetikoa funtzioaren parametro erreala zenbaki bat delako). Hirugarren dei honetarako moldea aldatu dugu ere, funtzioaren emaitza `LongInt` datu-motatako zenbaki bat denez `0` konstantearekin konparatzea zilegi da eta bere ondorioz `IF-THEN-ELSE` kontrol-egituraren mezu bat ala bestea agertuko da monitorean

4. deian uneko parametroa `Abs()` funtzio estandarren emaitza da. `Faktoriala()` funtzioaren parametroa sarrerakoa denez bere balioa beste funtzio batek eman diezaioke, gordeko den kontu bakarra datu-moten konpatibilitatearena izango da. Adibide honetan `Abs()` funtzio estandarrak itzultzen duen emaitzaren datu-mota bere parametroarena denez gero, eta `Datua` aldagaia `Integer` definitu delako `Abs(Datua)`-ren emaitza `Integer` izango da ere, prezeski `Faktoriala()`-k behar duen sarreraren datu-mota

Hauxe duzue `FuntzioarenDeiak` programari dagokion exekuzioaren adibide bat:

```
1. deian   4!=24
Zenbaki natural bat eman: 6
2. deiaren ondoren   6!=720
3. deian   20!-->gainezkada
Zenbaki negatibo bat eman: -5
4. deiaren ondoren   5!=120
_
```

6.6.1.3 Funtzioen adibideak

Jarraian funtzioen hiru adibide-programa erakusten dira.

6.6.1.3.1 Kosinua Taylor bitartez

Testuaren bosgarren kapituluan kontrol-egitura errepikakorrak ikasi genituen eta **5.3.3.3 Adibidea** puntuan angelu baten kosinua Taylorren formulaz kalkulatzera iritsi ginen. Baina orduan, halabeharrez, `KosinuaTaylorBitartez` izeneko programa bakar eta monolitiko baten bidez egiten zen, orain azpirrutinetan oinarrituko den programa baliokide osatuko dugu.

Ezer baino lehen Taylorren formula oroitzea derrigorrezkotzat jotzen dugu. Angelu baten kosinu kontzeptu trigonometrikoa Taylorri dagokion segida honen bitartez lor daiteke, non x zenbaki erreal bat den eta radianetan emaniko angelua adierazten duen:

$$Kos\ (x) = x^0/_{0!} - x^2/_{2!} + x^4/_{4!} - x^6/_{6!} + \ldots$$

Idatziko dugun `TaylorFuntzioz` programaren exekuzioak izango duen pantailaraketa aspaldiko `KosinuaTaylorBitartez` programak zuen berbera izango da, `rAng` aldagaian teklatuz irakurtzen den balioa berdina denean:


```
Lehenengo koadranteko angelua graduetan: 30
1. iterazioan =====> -0.13708
2. iterazioan =====>  0.00313
3. iterazioan =====> -0.00003
kosinua[30.000º] -----> 0.86605
cosinus(30.000º) -----> 0.86603
_
```

`TaylorFuntzioz` programak ezagutuko dituen hiru funtzioak, segidaren batugai bakoitzat dituen hiru elementu edo ezaugarriak aintzat hartzen dituzteenak izango dira. Programa nagusiko `rBatugai` aldagaiari dagokion balioa kalkulatzean ondoko hiru eginkizun bereiztuak bete behar dira:

1. zeinua kontrolatu
2. faktoriala kalkulatu
3. berreketa zehaztu

Batugai bati dagokion zeinua kontrolatzeko `-1` balioaren berreketaz egin daitekeenez hiru eginkizunak beteko dituzten bi funtzio idatzi dira. Hauxe izango da `TaylorFuntzioz` programaren kodea `EPSILON=0.0005` doitasun konstanterako. Gure programaren zuzentasuna frogatzearren, lorturiko emaitza eta `cos()` funtzio estandarrak eskaintzen duena alderatu egiten dira:

```
PROGRAM TaylorFuntzioz1 ;                    { \TP70\06\FUNADIB1.PAS }
USES
  Crt ;
CONST
  EPSILON = 0.0005 ;

FUNCTION Berreketa (rOinarria:Real; iAldiak:Integer) : Real ;
VAR
  rMetagailua : Real ;
  iKontagailua : Integer ;
BEGIN
     rMetagailua := 1 ;                 (* berreketaren kalkulua *)
     FOR iKontagailua:=1 TO iAldiak DO
             rMetagailua := rMetagailua * rOinarria ;

     Berreketa := rMetagailua ;
END ;

FUNCTION Faktoriala (iMuga:Integer) : LongInt ;
VAR
  liMetagailua : LongInt ;
  iKontagailua : Integer ;
BEGIN
     liMetagailua := 1 ;                 (* faktorialaren kalkulua *)
     FOR iKontagailua:=1 TO iMuga DO
             liMetagailua := liMetagailua * iKontagailua ;

     Faktoriala := liMetagailua ;
END ;

VAR
  iKont : Integer ;
  liFaktoreak : LongInt ;
  rAng, rX, rKosinua, rBatugai, rBerredura, rZeinua : Real ;

BEGIN
  ClrScr ;
  REPEAT
     Write ('Lehenengo koadranteko angelua graduetan: ') ;
     ReadLn (rAng) ;
  UNTIL (rAng <= 90) AND (rAng >= 0) ;

  rX := rAng * 2 * Pi / 360 ;     (* sarrerako angelua radianetara igaro *)
```

```
  iKont := 0 ;
  rKosinua := 0 ;
  rBatugai := 1 ;
  WHILE Abs(rBatugai) > EPSILON DO
  BEGIN
     rKosinua := rKosinua + rBatugai ;
     iKont := iKont + 2 ;

     liFaktoreak := Faktoriala(iKont) ;        (* faktorialaren deia *)
     rBerredura := Berreketa(rX, iKont) ;      (* berreketaren deia *)
     rZeinua := Berreketa(-1, iKont DIV 2) ;   (* berreketaren deia *)

     rBatugai := rZeinua * rBerredura / liFaktoreak ;
     WriteLn (iKont DIV 2, '. iterazioan =====> ', rBatugai:8:5) ;
  END;

  WriteLn ('kosinua[', rAng:0:3, 'º] -----> ', rKosinua:0:5) ;
  WriteLn ('cosinus(', rAng:0:3, 'º) -----> ', cos(rX):0:5) ;
END.
```

`TaylorFuntzioz1` programa hau eta bosgarren kapituluko **5.3.3.3 Adibidea** puntuan azaltzen den `KosinuaTaylorBitartez` programa berdintsuak dira, biak konparatzean orain azpimarratuko genukeena `Berreketa()` eta `Faktoriala()` funtzioek eskaintzen duten abantaila litzateke, alegia euren zereginak modulu banatuetan egotean programa nagusia asmatzeko eta garatzeko askeago aurkitzen garela.

Programa nagusiak egin behar duena honako esaldiaz definitzen da: *segidaren osagai baten kalkulua agindu eta metatzen jarraitu*. Zeregin errepikakorra denez, bigiztatik irteteko baldintza bat ezarri beharra dago, esate baterako kalkulatu den azken batugaiaren garrantzia `EPSILON` konstante bat baino txikiagoa izatea.

`WHILE` bigizta programa nagusiko elementua denez berari dagozkion kontrol-aldagaien hasieraketa egokiak bertan egin behar dira, `WHILE` bigizta eteteko baldintza programa nagusian kontrolatzen da ere. Baina programa nagusiak ez du gehiagoren ardurarik izango, izan ere iterazio bakoitzean lortzen den osagaiaren kalkulua `Berreketa()` eta `Faktoriala()` funtzioek burutzen dutelako eta osagaia eskuratzeko euren deiak egitea aski da.

6.6.1.3.2 Angeluen bihurketa

Bigarren adibide honetan angeluekin jarraituko dugu. Demagun angelu baten balioa radianetan teklatuz irakurtzen dugula, eta programak angeluaren graduak minutuak eta segundoak lortu behar dituela.

```
PROGRAM RadianakBihurtu1 ;                          { \TP70\06\BIHURK1.PAS }
USES
   Crt ;

FUNCTION GraduakKalkulatu (Ang:Real) : Integer ;
BEGIN
   GraduakKalkulatu := Trunc ( (Ang*360) / (2*Pi) ) ;
END;

FUNCTION MinutuakKalkulatu (Ang:Real) : Integer ;
VAR
   GraduenHondarra : Real ;
BEGIN
   GraduenHondarra := Frac ( (Ang*360) / (2*Pi) ) ;
   GraduenHondarra := GraduenHondarra * 60 ;
   MinutuakKalkulatu := Trunc (GraduenHondarra) ;
END;
```

```
FUNCTION SegundoakKalkulatu (Ang:Real) : Integer ;
VAR
   GraduenHondarra, MinutuenHondarra : Real ;
BEGIN
   GraduenHondarra := Frac ( (Ang*360) / (2*Pi) ) ;
   GraduenHondarra := GraduenHondarra * 60 ;
   MinutuenHondarra := Frac (GraduenHondarra) ;
   MinutuenHondarra := MinutuenHondarra * 60 ;
   SegundoakKalkulatu := Trunc (MinutuenHondarra) ;
END;


VAR                              (* programa nagusiko aldagaiak *)
  Radianak : Real ;
  Graduak, Minutuak, Segundoak : Integer ;

BEGIN
   ClrScr ;
   REPEAT
     Write ('Sarrerako datua eman (1. koadranteko angelu bat radianetan) ') ;
     ReadLn (Radianak) ;
   UNTIL (Radianak < 2*Pi/4) AND (Radianak > 0) ;

   Graduak := GraduakKalkulatu (Radianak) ;
   Minutuak := MinutuakKalkulatu (Radianak) ;
   Segundoak := SegundoakKalkulatu (Radianak) ;

   Write (Radianak:0:5,' radian ----> ') ;
   WriteLn (Graduak,'º ', Minutuak,''' ', Segundoak,'"') ;
END.
```

Adibide honen `RadianakBihurtu1` izeneko programan dauden `GraduakKalkulatu()`, `MinutuakKalkulatu()` eta `SegundoakKalkulatu()` hiru funtzioak berdintsuak direnez gero lehenengo biak azalduko ditugu soilik. Hiruretan `Trunc()` funtzio estandarra erabiltzen da, honek zenbaki erreal bat hartzen du, eta bere helburua sarrera moztu eta lortzen den kopuru osoa modulu deitzeileari itzultzea da. `MinutuakKalkulatu()` eta `SegundoakKalkulatu()` beste azpirrutinetan `Frac()` funtzio estandarra erabiltzen da ere, honek zenbaki erreal bat hartu eta bere dezimalak (kopuru erreal bezala noski) itzultzen ditu.

`GraduakKalkulatu()` funtzioan sarrerako `Radianak` angelua gradutara bihurtzen da, kopuru erreal horrek zati bi ditu osoa den aldea (graduak) eta dezimalak (minutuak eta segundoak). Beraz, graduak erdiesteko aski da `Trunc()` funtzio estandarraren bitartez alde osoa moztea eta zenbaki oso hori da itzultzen dena.

`MinutuakKalkulatu()` funtzioaren hasiera berdina da, hots, sarrerako `Radianak` angelua gradutara bihurtzen da ere, ondoren kopuru erreal horren alde dezimalarekin lan egingo denez `Frac()` funtzioa aplikatzen da graduen hondarra lortzen delarik. Hondarra minututara igarotzeko bider `60` eta horren zati osoa minutuak izanik dezimalak segundoak izango dira.

Hona hemen `RadianakBihurtu1` izeneko programaren irteera bat:

```
Sarrerako datua eman (1. koadranteko angelu bat radianetan) 0.75
0.75000 radian ----> 42º 58' 18"
_
```

Ikusten denez `SegundoakKalkulatu()` funtzioaren kodean segundoak lortzeko graduak eta minutuak kalkulatu beharra dago (beste bietan egiten dena egin beharra dago), nolabait esateko hiru funtzioen bitarteko bihurketan kalkukuluak errepikatzen dira. Horregatik hiru funtzioen ordez hiru emaitzak itzultzen dituen modulu bakarra idaz daiteke (aurrerago ikusiko dugun `RadianakBihurtu2` izeneko programan erakusten den prozedura).

6.6.1.3.3 Funtzio boolearra

Funtzio baten emaitza `Boolean` datu-motatako denean, funtzioari boolearra esaten zaio. Askotan funtzio boolerren deiak `IF`, `WHILE-DO` edo `REPEAT-UNTIL` egituren baldintzetan egiten dira.

Hirugarren adibidean zenbaki bat lehena den ala zatigarria den erabakitzen duen funtzio boolearra erabiltzen da. Horren bitartez `ZenbakiLehenakPantailaratzen` programan muga bat irakurtzen da eta `1`-tik hasita dauden zenbaki lehenak pantailaratzen dira.

```
PROGRAM ZenbakiLehenakPantailaratzen ;              { \TP70\06\LEHEN3.PAS }
USES
   Crt ;

FUNCTION LehenaDa (Zenbakia:Integer) : Boolean ;
VAR
   Lehena : Boolean ;        (* bertako aldagaia *)
   Kont : Integer ;          (* bertako aldagaia *)
BEGIN
   Lehena := TRUE ;
   Kont := 2 ;
   WHILE (Kont < Zenbakia) AND Lehena DO
   BEGIN
     IF Zenbakia MOD Kont = 0 THEN
            Lehena := FALSE ;
     Kont := Kont + 1 ;
   END;
   LehenaDa := Lehena ;
END;

VAR
  Muga, Kont : Integer ;        (* programa nagusiko aldagaiak *)

BEGIN
   ClrScr ;
   REPEAT
     Write ('Muga izango den zenbaki osoa eman: ') ;
     ReadLn (Muga) ;
   UNTIL Muga > 0 ;

   WriteLn ;
   WriteLn ('Hona hemen ', Muga, ' baino txikiagoak diren zenbaki lehenak:');
   FOR Kont:=1 TO Muga-1 DO
   BEGIN
     IF LehenaDa(Kont) THEN                         (* Funtzioaren deia *)
                          Write (Kont:5) ;
   END;
END.
```

`LehenaDa ()` funtzioak zenbaki natural bat hartzen du eta dituen zatitzaileak bilatzen jartzen da, zaititzaileren bat aurkitzean zenbakia lehena ez dela frogatzen da eta ondorioz `FALSE` konstantea itzuliko dio modulu deitzaileari. Programa nagusia informazio hortaz baliatzen da `Write()` pantailaraketak burutzeko.

Hauxe da `ZenbakiLehenakPantailaratzen` programari dagokion irteera bat:

```
Muga izango den zenbaki osoa eman: 17

Hona hemen 17 baino txikiagoak diren zenbaki lehenak:
    1    2    3    5    7   11   13
_
```


6.6.1.4 Zenbait funtzio estandar

Hona hemen Turbo Pascal lengoaiak ezagutzen dituen zenbait[20] funtzio estandar:

	Funtzioa	Deskribapen laburra	x argumentu datu-mota	Emaitzaren datu-mota
Aritmetikoak	**Abs (x)**	Balio absolutua	Osoa edo Erreala	x bezalakoa
	Sqr (x)	Karratua	Osoa edo Erreala	x bezalakoa
	Sqrt (x)	Erro karratua (x >= 0)	Osoa edo Erreala	Erreala
	Exp (x)	E ber x (E = 2.7182818)	Osoa edo Erreala	Erreala
	Ln (x)	Logaritmo Nepertarra[21] (x > 0)	Osoa edo Erreala	Erreala
Trigonometrikoak	**Pi**	Pi zenbakia		Erreala
	Sin (x)	x-en sinua (x radianetan)	Osoa edo Erreala	Erreala
	Cos (x)	x-en cosinua (x radianetan)	Osoa edo Erreala	Erreala
	ArcTan (x)	Arku tangentea	Osoa edo Erreala	Erreala (radianetan)
Datu-mota bihurketak	**Round (x)**	Erreal bat osora biribiltzen du	Erreala	LongInt
	Trunc (x)	Erreal bat osora mozten du	Erreala	LongInt
	Int (x)	Erreal baten alde osoa	Erreala	Erreala
	Frac (x)	Erreal baten zatikia	Erreala	Erreala
Karakterekoak	**Ord (x)**	Karaktere baten ordinala	Char	LongInt
	Chr (x)	Oso bati dagokion karakterea	Byte	Char
	Pred (x)	Argumentuaren aurrekoa	Char	Char
	Succ (x)	Argumentuaren hurrengoa	Char	Char
	ReadKey	Karaktere bat teklatuz irakurri		Char
	UpCase (x)	Karaktere baten majuskula	Char	Char
Grafikoak	**GetX**	Kurtsorearen x koordenatua	-	Integer
	GetY	Kurtsorearen y koordenatua	-	Integer
	GetPixel (x,y)	(x,y) pixelaren balioa	Integer	Word
	GraphResult	Errore kodea itzultzen du	-	Integer
	TextHeight (k)	k karaktere katearen altuera	String	Wordr
	TextWidth (k)	k karaktere katearen zabalera	String	Wordr
Besterik	**Odd (x)**	Oso bat bakoitia den ala ez	Osoa	Boolean
	Random	0 eta 0.99999 arteko zenbaki aleatorioa itzultzen du		Erreala
	Random (x)	0 eta x-1 arteko zenbaki aleatorioa itzultzen du	Osoa eta positiboa	Osoa eta positiboa

[20] Hau zerrenda laburtu bat besterik ez da, Turbo Pascal lengoaiak ehundaka funtzio aurredefiniturik baitu

[21] Zenbaki baten logaritmoa edozein oinarritan, logaritmo neparterren arteko eragiketa bezala plantea daiteke. $\log_z x$ = (logaritmo, z oinarrian, x) = Ln x / Ln z

6.6.2 Prozedurak

Programen moduluak osatzeko, funtzioekin batera, Turbo Pascal eta Pascal arruntak prozedurak ezagutzen ditu. Aspalditik erabili izan ditugu prozedurak, idatzi genuen gure lehen programa `WriteLn()` prozeduran oinarritzen zen adibidez.

Esan dugun bezala funtzio eta prozeduraren arteko ezberdintasuna kontzeptuan baino praktikan ematen da. Komenioz, funtzio bat emaitza bakar bati loturik doan bitartean, prozedura kalkulua baino eginkizuna izango da guretzat. Ondorioz prozedurak emaitza bat edo gehiago itzul diezaioke modulu deitzaileari, edo beharbada ez dio inolako emaitzik itzuliko.

Modulu deitzaileari emaitzik itzultzen ez dion prozedura baten kasua ikus dezagun, horretarako kapitulu honen hasierako `ZenbakiKonbinatorioa3AzpPrg` programa gogora ekar dezagun:

```
PROCEDURE DatuakHar (VAR ZenbM,ZenbN:
                         Integer) ;
BEGIN
   Write ('m eman: ') ;
   ReadLn (ZenbM) ;
   Write ('n eman: ') ;
   ReadLn (ZenbM) ;
END;
```

```
FUNCTION Faktoriala (Muga:Interger) :
                              LongInt ;
VAR
  Kont : Integer  ;
  Fakt : LongInt ;
BEGIN
   Fakt := 1 ;
   FOR Kont:=1 TO Muga DO
   Fakt := Fakt * Kont ;
   Faktoriala := Fakt  ;
END;
```

```
PROZEDURE Erakuts (FktM, FktN, FktM_N:
                   LongInt) ;
VAR
  Ema : LongInt ;
BEGIN
 Ema := FktM DIV (FktN * FktM_N) ;
 WriteLn ('Zenb. konb. = ', Ema ) ;
END;
```

```
PROGRAM ZenbakiKonbinatorioa3AzpPrg ;




VAR
   ZbkM, ZbkN, ZbkM_N : Integer ;
   FaktM, FaktN, FaktM_N : LongInt ;

BEGIN

    DatuakHar (ZbkM, ZbkN) ;

    FaktM := Faktoriala (ZbkM) ;

    FaktN := Faktoriala (ZbkN) ;

    ZbkM_N := ZBKM - ZbkN ;
    FaktM_N := Faktoriala (ZbkM_N) ;

    Erakuts (FaktM, FaktN, FaktM_N) ;

END.
```

Programa nagusia

← *Azpiprogramak*

`ZenbakiKonbinatorioa3AzpPrg` programa honetan ezaguna zaigun `Faktoriala()` funtzioarekin batera bi prozedura azaltzen zaizkigu: bat `DatuakHar()` izenekoa zeinek bi oso irakrri ondoren programa nagusira bueltatzen dituen, eta bestea `Erakuts()` deituriko zeinek ez dion modulu deitzaileari ezer itzultzen. Hona hemen, hiru moduluetan, bakoitzari dagokion helburua:

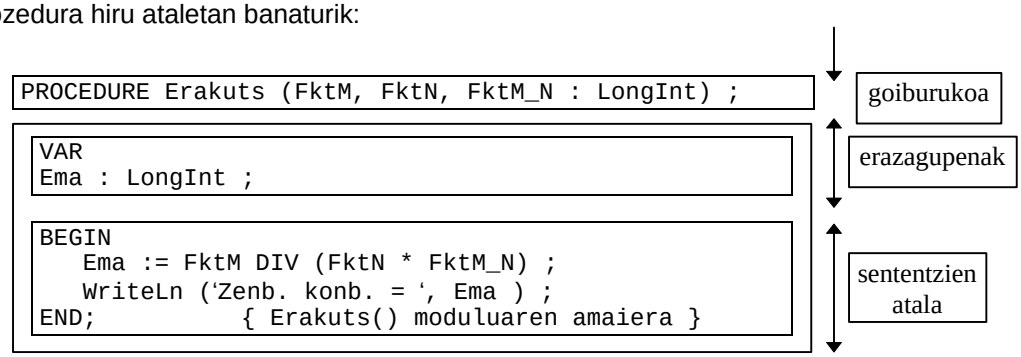
Modulua	Mota	Helburua	Itzulitakoa
`Faktoriala()`	Funtzioa	Parametroaren faktoriala kalkulua lortzea	Zenbaki oso bat
`DatuakHar()`	Prozedura	Bi balio oso teklatuz irakurri ahal izatea	Bi zenbaki oso
`Erakuts()`	Prozedura	Balio oso baten pantailaraketa	

Nahiz eta `Erakuts()` prozedurak ezer ere ez itzuli programa nagusiari, badu helburu eta zeregin zehatza. Halako kasuetan prozedura batek kanpoko daturen bat behar izango balu baliozko parametroa edo konstante-parametroa izango da.

Bestalde, `DatuakHar()` deituriko prozedurak balio bi itzuli behar dizkio modulu deitzailea den programa nagusiari. Irteera dagoen halako kasuetan prozedura batek aldagai-parametroa erabiliko du. Izan daiteke aldi berean prozedura batek kanpoko daturen bat behar izatea eta emaitzaren bat bueltatu behar izatea, orduan parametro formalak holakoak izango dira: sarrerako datuak baliozko parametroak edo konstante-parametroak izango diren bitartean, irteerako emaitzek aldagai-parametroen jokaera izango dute.

6.6.2.1 Prozeduraren atalak

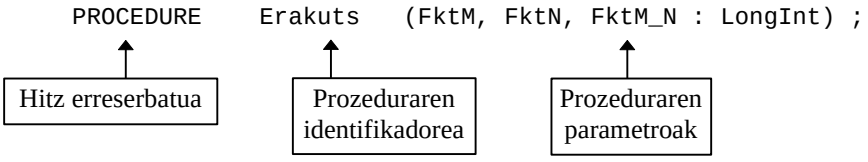
Edozein prozedurak, ikasi ditugun funtzio baten hiru zatiak izango ditu, jarraian datozen puntuetan zehaztasunez azaltzen direnak. Hau litzateke, esate baterako, `Erakuts()` prozedura hiru ataletan banaturik:



Prozeduraren egitura, espero genukeenez, Turbo Pascal programa baten egitura berbera da.

6.6.2.1.1 Prozeduraren goiburukoa

`Erakuts()` prozedura adibidetzat harturik bere goiburukoan hiru elementu agertzen dira:



Prozedura baten blokea hasten dela adierazteko Pascal lengoaian `PROCEDURE` marka jarri behar da, eta bere ondoren prozedura etiketatzen duen izen bat dator (prozeduraren identifikadore bat finkatzerakoan **4.2.2.2 Erabiltzailearen identifikadoreak** puntuan emaniko arauak gogora ekar).

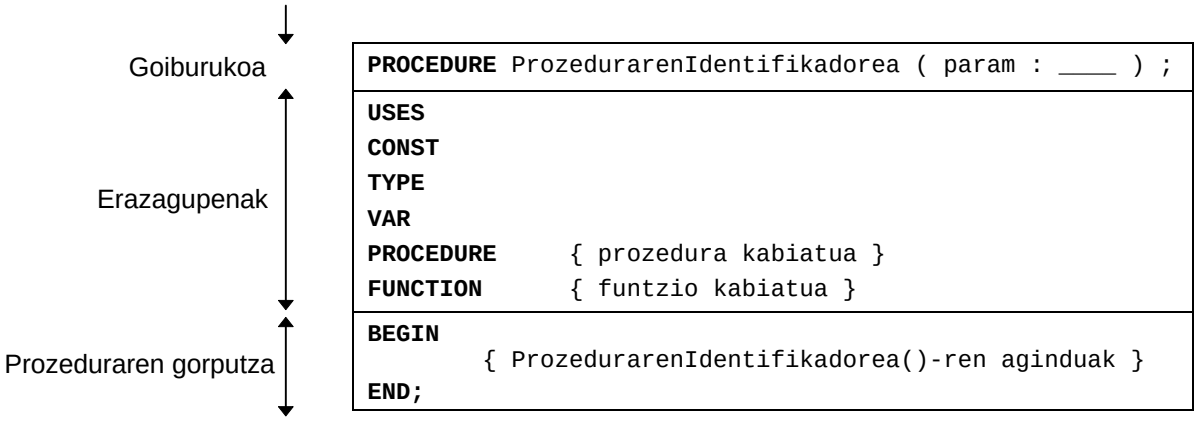
Modulu nagusiak prozedura bat aktibatzeko dei bat egin beharko dio eta horretarako prozeduraren identifikadorea erabiliko du, eta hori izango da prozedura baten identifikadoreak izango duen zeregin bakarra (funtzioetan azpirrutinaren identifikadoreak dituen ardurak bi direla ez ahaztu).

Esan dugun bezala, prozedurak lan egiteko kanpoko daturik behar izango balu, modulu deitzaileak parametroen bitartez hornituko dio, eta, parametroaren datu-mota zehazten delarik parentesi artean mugatuko da goiburukoan. Modu beretsuan prozedurak emaitzik itzuliko balio modulu deitzaileari, kanporako komunikazioa parametroen bitartez gauzatuko da, irteerako aldagai-parametroen bitartez alegia.

Prozeduraren goiburukoa puntu eta komaz bukatzen da.

6.6.2.1.2 Prozeduraren erazagupenak

`DatuakHar()` prozedura salbuespen bat izan arren prozedura gehienek aldagai laguntzaileak behar izaten dituzte euren zereginak bete ahal izateko, halakoetan ez dira aldagai orokorrak erabiliko bertako aldagai pribatuak baizik eta horretarako prozeduraren erazagupenen atala dago. Prozeduraren bigarren atal honetan aldagaiak ezezik unitateak, konstanteak, datu-motak eta azpirrutina kabiatuak ere erazagut daitezke:



Prozeduraren goiburukoa esplikatzean esandakoa errepikatuz, prozedura batek onar ditzakeen parametro motak ikasitako hirurak dira (aldagai-parametroak, baliozko parametroak eta konstante-parametroak) eta komunikazioa sarrerakoa denean azken biak erabiliko diren bitartean, komunikazioa irteerakoa izatean aldagai-parametroak erabiliko dira.

6.6.2.1.3 Prozeduraren sententzien atala

Suposa daitekeenez, prozeduaren helburua mamitzen duten aginduak idatziko dira hirugarren atal honetan.

6.6.2.2 Prozeduraren deia

Funtzio baten helburua balio bat lortzea zela gogoratuz, funtzioaren deia balio hori ager daitekeen tokietan kokatu ahal izango dela esan dugu; adibidez espresioetan eta beste moduluren bat deitzen duen uneko parametroetan. Baina prozedurak ezin daitekeenez kalkulu batekin (kalkuluaren emaitzarekin) ordezkatu, ezin izango dira funtzioak bezala deitu.

Prozedurek duten deiaren formatua bakarra da: prozeduraren identifikadorea eta parametro errealak parentesi artean eta komaz banaturik. Prozeduren deia instrukzio edo agindu bat denez puntu eta komaz amaituko da.

6.6.2.3 Prozeduren adibideak

Jarraian prozeduren hiru adibide-programa erakusten dira.

6.6.2.3.1 Kosinua Taylor bitartez

Kosinua Taylorren formulaz kalkulatzen duen `TaylorFuntzioz1` programa **6.6.1.3.1** puntuan aztertu da, eta bertan programa nagusiko angelua gradutan irakurri eta `rAng` izeneko aldagaian gordetzen da. Honelako zerbait egiten da:

```
REPEAT
   Write ('Lehenengo koadranteko angelua graduetan: ') ;
   ReadLn (rAng) ;
UNTIL (rAng <= 90) AND (rAng >= 0) ;
```

Baina zein da `rAng` sarrera daturako soilik lehenengo koadrantea onartzeko baldintza mugatuaren arrazoia?. Experimenta dezagun `REPEAT-UNTIL` egitura kendu eta `rAng` sarrera angelu handiak emanik, honelako zerbait agertuko zaigu:

```
Lehenengo koadranteko angelua graduetan: 160
1. iterazioan =====> -3.89910
2. iterazioan =====>  2.53383
3. iterazioan =====> -0.65865
4. iterazioan =====>  0.09172
5. iterazioan =====> -0.00795
6. iterazioan =====>  0.00047
kosinua[160.000º] -----> -0.94014
cosinus(160.000º) -----> -0.93969
_
```

Bigarren koadranteko `160` gradutako angelua ematean ez dirudi ezer arrarorik ateratzen denik, izan ere batugaia alternatiboki positiboa eta negatiboa da eta balio absolutuan gero eta txikiagoa. Baina `160` gradutako angelu hori gainditzen duten sarrera handiagorekin hona hemen `TaylorFuntzioz1` programaren emaitza:

```
Lehenengo koadranteko angelua graduetan: 390
1. iterazioan =====> -23.16615
2. iterazioan =====> 89.44512
3. iterazioan =====> -138.13997
4. iterazioan =====> 114.29185
5. iterazioan =====> -58.83784
6. iterazioan =====> 20.65222
7. iterazioan =====> -358.37333
8. iterazioan =====> 10595.77872
9. iterazioan =====> 1095140.48970
10. iterazioan =====> -21685995.18000
11. iterazioan =====> 4040716356.60000
12. iterazioan =====> -126117617480.00000
13. iterazioan =====> 2445733690500.00000
14. iterazioan =====> -152700880390000.00000
15. iterazioan =====> -6906532447600000.00000
16. iterazioan =====> -209997112020000000.00000
Runtime error 200 at 0C12:02D0
_
```

Zergatik suertatu da errorea exekuzioan? Zer dela eta hartzen ditu batugaiak horrelako balioak iterazio bakoitzeko?

Galdera horiei dagokien erantzuna ulertzeko `TaylorFuntzioz1` programatik abiatuta beste programa bat idatzi dugu, `TaylorFuntzioz2` izendatu dena. `TaylorFuntzioz2` izeneko programa berrituan egin diren aldaketak txikiak dira:

1. Sarrerako `rAng` angeluaren irakurketa mugatzen duen `REPEAT-UNTIL` egitura kendu egin da (eta dagokion mezua ere)
2. Programa nagusiko `WHILE-DO` bigiztara sartu aurretik `WriteLn()` bi tartekatu dira, lehenengoan `rAng` angelua radianetan erakusten da eta bigarrenari esker karakteren konstante bat idatzi da pantailan
3. Programa nagusiko `WHILE-DO` bigiztara barruko mezuan, iterazio bakoitzeko, `Faktoriala()` eta `Berreketa()` funtzioen emaitzak eta batugaia aurkezten dira

`TaylorFuntzioz1` programari aldaketa horiek aplikaturik `TaylorFuntzioz2` programa lortzen da eta honen exekuzioa `160` gradutako angelu batetarako ez luke inolaz errorerik eragingo. Baina `160` gradu izan beharrean, sarreran irakurtzen den angelua `161` gradutako baldin bada, orduan `TaylorFuntzioz2` programaren exekuzioa ez da zuzentasunez amaitzen errore bat suertatzen delako eta pantailarakada honelako zerbait izango da:

```
Edozein koadranteko angelua graduetan: 161
Angelua radianetan: 2.80998
                        Faktoriala              Berredura       Batugaia
 1. iterazioan =====>             2                7.89599       -3.94799
 2. iterazioan =====>            24               62.34663        2.59778
 3. iterazioan =====>           720              492.28824       -0.68373
 4. iterazioan =====>         40320             3887.10211        0.09641
 5. iterazioan =====>       3628800            30692.51214       -0.00846
 6. iterazioan =====>     479001600           242347.71175        0.00051
 7. iterazioan =====>    1278945280          1913574.65730       -0.00150
 8. iterazioan =====>    2004189184         15109562.79600        0.00754
 9. iterazioan =====>    -898433024        119304928.61000        0.13279
10. iterazioan =====>   -2102132736        942030301.17000       -0.44813
11. iterazioan =====>    -522715136       7438260083.90000       14.23005
12. iterazioan =====>    -775946240      58732413392.00000      -75.69134
13. iterazioan =====>   -1853882368     463750439480.00000      250.15095
14. iterazioan =====>   -1375731712    3661767969200.00000    -2661.68755
15. iterazioan =====>    1409286144   28913276450000.00000   -20516.25681
16. iterazioan =====>   -2147483648  228298887890000.00000  -106309.95403
Runtime error 200 at 0C12:02D0
_
```

gainezkada

Zazpigarren iterazioan faktoriala kalkulatzean gure funtzioak `1278945280` itzultzen du eta hori ez da zuzena `14! = 87178291200` delako. Egia esan `Faktoriala()` funtzioan balioak metatzean gainezkada suertatu da 87178291200 kopurua `LongInt` datu-motatako aldagai batean ezin delako biltegitu. Gainezkadaren ondorioz zazpigarren iterazioan lortzen den batugaiaren balioa `-0.00150` da, hots, txikiagoa izan beharrean aurrekoaren baino balio absolutu handiagoa du eta programa nagusiko `WHILE-DO` bigizta ez da eteten. Une jakin batean, 16 iterazioan, faktorialak `0` ematen du eta hamaseigarren batugaiak infinitu balioko du eta hortik `TaylorFuntzioz2` programaren abortatzea.

Gainezkadak konponbiderik ez duenez, arazoak ekiditeko bi bide daude:

1

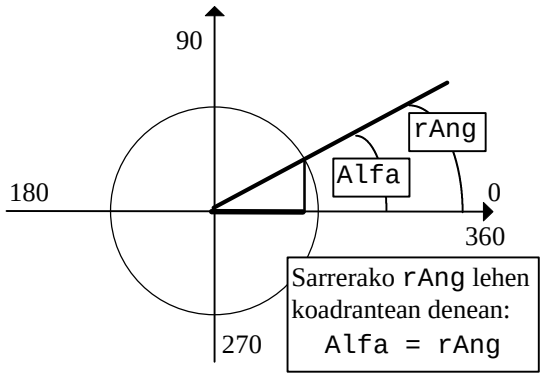
Programari eskatzen zaion doitasuna lasatzea, hau da `EPSILON` konstantea `0.0005` izan beharrean handiagoa izatea. Adibidez doitasuna honela definituz `EPSILON=0.0006` seigarren iterazioan batugaiaren balioa azpitik geratzen da eta `WHILE-DO` bigiztaren balditza beteko litzateke, `161`-ren kosinua zuzentasunez lortzen delarik.

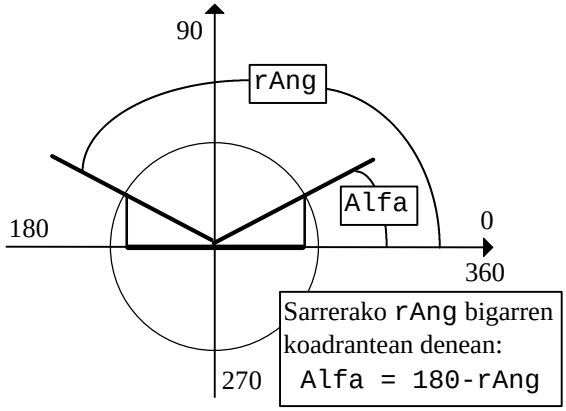
Tamalez, `EPSILON=0.0006` aldaketa horrekin ez da modu orokorrean arazoa konpontzen (froga daiteke orain gainezkada `164` gradutako angelu batentzat gertatzen dela).

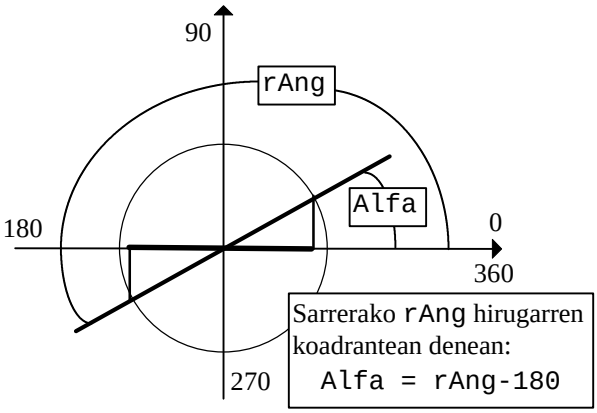
2

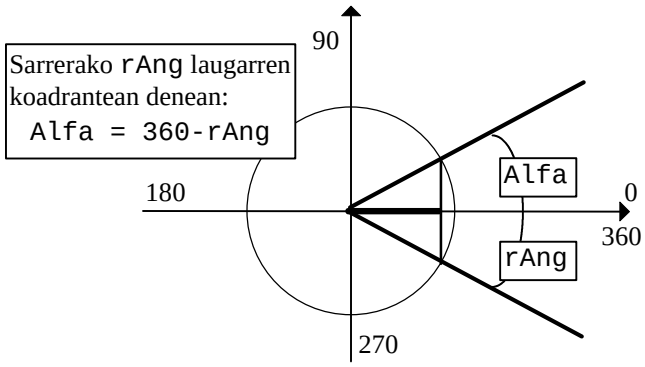
Programatik itxaroten den doitasuna finkoa eta aldaezina bada soluzioa sarrera angeluaren balioa aldatzetik etorriko da. Izan ere, doitasuna `0.0005` izanik gainezkadaren muga `160` graduetan dago, hots, kodrante oso bat eta bigarrenaren zati handi bat. Programaren azken helburua angelu baten kosinua lortzea dela kontutan izanik, edozein angelu teklaturik irakurri arren sarrerari "dagokion" lehenengo koadranteko angeluaren kosinua kalkulatuko dugu (ikus ondoko irudiak).

Datua den `rAng` sarrerako angeluari dagokion lehenengo koadranteko angelua zehazten duen azpirrutina berri bat idatziko da, zeini `Moldaketa()` deituko diogun.









Sarreran teklatuz irakurtzen den `rAng` angeluari dagokion `Alfa` moldatua lortzeaz gain `Moldaketa()` deituriko azpirrutinak jatorrizko datuaren koadrantea itzuli beharko du ere. Izan ere, jatorrizko angeluaren eta angelu moldatuaren kosinuen balio absolutuak berdinak izan arren bigarren eta hirugarren kodranteetako angeluek kosinu negatiboak eragiten dituzte eta beste bi kodranteen angeluek aldiz positiboak. Ondoriz, `Moldaketa()` azpirrutinak gauza bi itzuliko dizkio bere modulu deitzaileari: angelu moldatua (kopuru erreal bat) eta jatorrizko angeluaren koadrantea (1, 2, 3 edo 4 balio osoetatik bat).

Beraz, funtzio batek guretzat emaitza bakar bat baino ezin duenez itzuli, `Moldaketa()` azpiprograma ondoko goiburukoa duen prozedura izango da:

```
PROCEDURE Moldaketa (VAR rAngMoldatua : Real; VAR bKoadrantea : Byte) ;
```

Hau litzateke `Moldaketa()` prozedura barneratzen duen `TaylorFuntzioz3` programa, kontutan izan programa nagusiko bigizta `WHILE-DO` izan beharrean `REPEAT-UNTIL` dela baina horrek ez du garrantzirik ebatzi nahi dugun arazoan.

Funtsezkoena `Moldaketa()` prozedura da, bere kodean sarrerako angeluak zenbat aldiz biratzen duen zehaztu ondoren `Frac()` funtzio estandarraren bitartez biraketaren hondarra lortzen da (zein lau koadranteetatik edozeinean egon daitekeen):

```
PROGRAM TaylorFuntzioz3 ;                              { \TP70\06\FUNADIB3.PAS }
USES
  Crt ;
CONST
  EPSILON = 0.0005 ;


FUNCTION Berreketa (rOinarria:Real; iAldiak:Integer) : Real ;




FUNCTION Faktoriala (iMuga:Integer) : LongInt ;




PROCEDURE Moldaketa (VAR rAngMoldatua : Real; VAR bKoadrantea : Byte) ;
VAR
  rHondar : Real ;
BEGIN
  Write ('Moldatzean ') ;
  rAngMoldatua := Abs (rAngMoldatua) ;
  rHondar := Frac (rAngMoldatua / (2*PI)) ;     { Bira kopuruaren hondarra }
  rAngMoldatua := 2 * PI * rHondar ;            { Hondarra radianetan }

  IF (rAngMoldatua < PI/2) AND (rAngMoldatua >= 0) THEN
  BEGIN
    WriteLn (rAngMoldatua:0:5, ' radian   (1. koadrantea)') ;
    rAngMoldatua := rAngMoldatua;
    bKoadrantea := 1 ;
  END ;

  IF (rAngMoldatua < PI) AND (rAngMoldatua >= PI/2) THEN
  BEGIN
    WriteLn (PI-rAngMoldatua:0:5, ' radian   (2. koadrantea)') ;
    rAngMoldatua := PI - rAngMoldatua ;
    bKoadrantea := 2 ;
  END ;

  IF (rAngMoldatua < 3*PI/2) AND (rAngMoldatua>=PI) THEN
  BEGIN
    WriteLn (rAngMoldatua-PI:0:5, ' radian   (3. koadrantea)') ;
    rAngMoldatua := rAngMoldatua - PI ;
    bKoadrantea := 3 ;
  END ;

  IF (rAngMoldatua < 2*PI) AND (rAngMoldatua >= 3*PI/2) THEN
  BEGIN
    WriteLn (2*PI-rAngMoldatua:0:5, ' radian   (4. koadrantea)') ;
    rAngMoldatua := 2*PI - rAngMoldatua ;
    bKoadrantea := 4 ;
  END ;
END ;

VAR
  iKont : Integer ;
  liFaktoreak : LongInt ;
  rX, rEmaitza, rZeinua, rBatugai, rAngelua, rBerredura : Real ;
  bKoadrantea : Byte ;
```



```
BEGIN
  Clrscr ;
  Write ('Edozein koadranteko angelua graduetan:  ') ;
  Read (rX) ;
  rX := rX * 2 * PI / 360 ;     (* sarrerako angelua radianetara igaro *)
  WriteLn ('Angelua radianetan: ', rX:0:5) ;

  rAngelua := rX ;
  Moldaketa (rAngelua,bKoadrantea) ;
  iKont := 1 ;
  rEmaitza := 1 ;

  REPEAT
     liFaktoreak := Faktoriala (2*iKont) ;
     rBerredura := Berreketa (rAngelua, 2*iKont) ;
     rZeinua := Berreketa (-1, iKont) ;

     rBatugai := rZeinua * rBerredura / liFaktoreak ;
     WriteLn (iKont,'. iterazioan batugaia: ', rBatugai:8:5) ;
     rEmaitza := rBatugai + rEmaitza ;

     iKont:=iKont + 1 ;
  UNTIL Abs(rBatugai) < EPSILON ;

  IF (bKoadrantea=2) OR (bKoadrantea=3) THEN
    rEmaitza:=(-1)*rEmaitza ;

  Write ('gure programaz lortutako kosinua [',rAngelua:8:4,'] -----> ') ;
  WriteLn (rEmaitza:8:5) ;
  Writeln ('sarrerako angeluaren kosinua     (',rX:8:4,') -----> ',cos(rX):8:5) ;
  Write ('angelu moldatuaren kosinua       (',rAngelua:8:4,') -----> ') ;
  WriteLn (cos(rAngelua):8:5) ;
END.
```

`TaylorFuntzioz3` programa egikaritzean honako zerbait agertuko da:

```
Edozein koadranteko angelua graduetan: 570
Angelua radianetan: 9.94838
Moldatzean 0.52360 radian   (3. koadrantea)
1. iterazioan batugaia: -0.13708
2. iterazioan batugaia:  0.00313
3. iterazioan batugaia: -0.00003
gure programaz lortutako kosinua [  0.5236] -----> -0.86603
sarrerako angeluaren kosinua     (  9.9484) -----> -0.86603
angelu moldatuaren kosinua       (  0.5236] ----->  0.86603
_
```

6.6.2.3.2 Angeluen bihurketa

Prozeduren bigarren adibide honen eta funtzioen bigarren adibidearen helburuak berdinak dira, **6.6.1.3.2** puntuan egindako planteamendua errepikatuz: demagun angelu baten balioa radianetan teklatuz irakurtzen dugula, eta programak angeluaren graduak minutuak eta segundoak lortu behar dituela.

Kasu honetan `RadianakBihurtu2` programan unitate bihurketak burutzeko erabiltzen den azpirrutina prozedura bat da, eta bertan hiru emaitzak zehaztu ondoren programa nagusiari itzultzen zaizkio.

`RadianakBihurtu1` eta `RadianakBihurtu2` programak alderatuz gero, bigarrenak dei bakar bat duenez bizkorrago lan egingo duela pentsa daiteke eta hortik egokiagoa dela, nahiz eta emaitzak bietan berdinak izan.

```
PROGRAM RadianakBihurtu2 ;                              { \TP70\06\BIHURK2.PAS }
USES
   Crt ;


PROCEDURE Kalkuluak (Ang : Real; VAR Gradu, Minutu, Segundo : Integer) ;
VAR
   GraduenHondarra, MinutuenHondarra : Real ;
BEGIN
   Gradu := Trunc ( (Ang*360) / (2*Pi) ) ;
   GraduenHondarra := Frac ( (Ang*360) / (2*Pi) ) ;
   GraduenHondarra := GraduenHondarra * 60 ;
   Minutu := Trunc (GraduenHondarra) ;
   MinutuenHondarra := Frac (GraduenHondarra) ;
   MinutuenHondarra := MinutuenHondarra * 60 ;
   Segundo := Trunc (MinutuenHondarra) ;
END;


VAR                                (* programa nagusiaren aldagaiak *)
  Radianak : Real ;
  Graduak, Minutuak, Segundoak : Integer ;

BEGIN
   ClrScr ;
   REPEAT
     Write ('Sarrerako datua eman (1. koadranteko angelu bat radianetan) ') ;
     ReadLn (Radianak) ;
   UNTIL (Radianak < 2*Pi/4) AND (Radianak > 0) ;

   Kalkuluak (Radianak, Graduak, Minutuak, Segundoak) ;

   Write (Radianak:0:5,' radian ----> ', Graduak,'º ') ;
   WriteLn (Minutuak,''' ', Segundoak,'"') ;
END.
```

`RadianakBihurtu2` programaren irteera, funtzioz osaturiko `RadianakBihurtu1`-ren berbera izango litzateke.. Adibidez:

```
Sarrerako datua eman (1. koadranteko angelu bat radianetan) 0.75
0.75000 radian ----> 42º 58' 18"
_
```

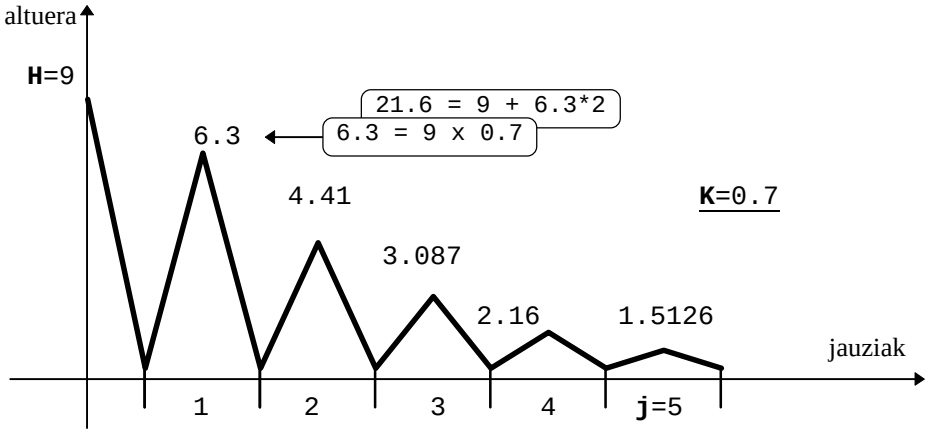
6.6.2.3.3 Pilota jauzika

Demagun pilota elastiko bat dugula eta `H` altuera batetik erortzen uzten dugula, behean jo ondoren pilotak gorantz egingo du berriro altuera maximo bat lortuz. Pilotak jauzi batetik bestera lurraren aurkako talkan galtzen duen energia berezkoa duen ezaugarri bat da, `K` errebotearen ahalmena deituko duguna.

Jarraian erakusten den irudian pilotak bost jauzi eman ditu, jauzi bakoitzaren altuera eta ibilitako distantzia metatua pantailaratu egin behar ditu programak. Adibe-programan parte hartzen duten elementuen zerrenda:

- `H` pilota jaurtitzen den altuera (irudian `H=9.0`)
- `K` errebotearen ahalmena 1.0 > K > 0.0 (irudian `K=0.7`)
- `j` jauzi kopurua (irudian `j=5`)

Programan `H` eta `K` teklatuz irakurriko dira.



`PilotaJauzika` programa nagusian `FOR-DO` bigizta bat izango dugu eta bere barruan `EmaitzaBi()` prozedurari dei bat egingo zaio. `EmaitzaBi()` prozedurak jauzi bati dagokion altuera maximoa itzuliko du, horretarako azken jauziaren altuera bider errebotearen ahalmena lortzea aski da. `EmaitzaBi()` prozedurak jauzi jakin baten altuera maximoan hasieratik ibilitako distantzia metatua itzuliko du ere, horretarako prozedurak behar duen datu bakarra ordurarte ibilitako distantzia da eta barruan altuera maximoa kalkulatzeko gai denez goiko eskeman jarri den eragiketa burutzea aski da soluzioa lortzeko.

Hona hemen `PilotaJauzika` programaren kodea:

```
PROGRAM PilotaJauzika ;                                  { \TP70\06\JAUZIKA.PAS }
USES
   Crt ;
CONST
   JAUZIKOPMAX = 5 ;

PROCEDURE EmaitzaBi (rElast : Real; VAR rAlt, rDist : Real) ;
BEGIN
   rAlt := rAlt * rElast ;
   rDist := rDist + rAlt*2 ;
END;


VAR                                (* programa nagusiaren aldagaiak *)
  rH, rK, rDistantzia, rAltuera : Real ;
  Jauzia : Byte ;

BEGIN
   ClrScr ;
   REPEAT
     Write ('Hasierako altuera eman (1.0 eta 10.0 bitartekoa) ') ;
     ReadLn (rH) ;
   UNTIL (rH >= 1) AND (rH <= 10) ;

   REPEAT
     Write ('Pilotaren errebote ahalmena eman (0.0 eta 1.0 bitartekoa) ') ;
     ReadLn (rK) ;
   UNTIL (rK >= 0) AND (rK <= 1) ;

   rDistantzia := rH ;
   rAltuera := rH ;
   FOR Jauzia:=1 TO JAUZIKOPMAX DO
   BEGIN
      EmaitzaBi (rK, rAltuera, rDistantzia) ;
      Write (Jauzia, '. jauzian --->   Altuera maximoa: ', rAltuera:0:3) ;
      WriteLn ('    Distantzia metatua: ', rDistantzia:0:3) ;
   END ;
END.
```

Eta hauxe da, adibide datuekin, `PilotaJauzika` programarri dagokion irteera:

```
Hasierako altuera eman (1.0 eta 10.0 bitartekoa) 9
Pilotaren errebote ahalmena eman (0.0 eta 1.0 bitartekoa) 0.7
1. jauzian --->   Altuera maximoa: 6.300    Distantzia metatua: 21.600
2. jauzian --->   Altuera maximoa: 4.410    Distantzia metatua: 30.420
3. jauzian --->   Altuera maximoa: 3.087    Distantzia metatua: 36.594
4. jauzian --->   Altuera maximoa: 2.161    Distantzia metatua: 40.916
5. jauzian --->   Altuera maximoa: 1.513    Distantzia metatua: 43.941
_
```

6.6.2.4 Zenbait prozedura estandar

Hona hemen Turbo Pascal lengoaiak ezagutzen dituen zenbait[22] prozedura estandar:

	Prozedura	Deskribapen laburra
Sarrera/Irteerakoak	**Read ()**	Teklatu edo fitxategitik irakurri
	ReadLn ()	Teklatutik irakurri
	Write ()	Pantailan edo fitxategian idatzi
	WriteLn ()	Pantailan idatzi
Karakterekoak	**Str ()**	Zenbaki bat karaktere kate bihurtu
	Val ()	Karaktere kate bat zenbaki bihurtu
	Delete ()	Karaktere kate baten zatia ezabatu
	Insert ()	Karaktere kate batean beste karaktere batzuk tartekatu
Grafikoak	**DetectGraph ()**	Grafiko kontrolagailua eta lan modua detektatu
	InitGraph ()	Pantaila grafikoa ireki
	CloseGraph ()	Pantaila grafikoa itxi
	Rectangle ()	Rektangulu bat marraztu
	Circle ()	Zirkulu bat marraztu
	Line ()	Lerro bat marraztu
Besterik	**ClrScr ()**	Pantaila garbitu
	Randomize ()	Zenbaki aleatorioen hazia hartu

6.7 ERREKURTSIBITATEA

Identifikadore jakin bat Turbo Pascal programa batean ezaguna izan dadin non kokatuko den **6.5.2 Aldagaien Esparrua** puntuan ikasi da. Azpirrutina batek bere gainean definiturik aurkitzen diren azpirrutinak dei ditzake, bere gaineko azpirrutinak eta bere burua ere dei dezake edozein azpiprogramak. Azpiprograma batek bere buruari deitzen dionean dei errekurtsiboa egin dela esaten da.

Azpiprograma baten barruan dei errekurtsibo bat egitean, berriro exekutarzen da azpiprograma bera eta berriro egingo da dei errekurtsiboa zeinek azpiprograma piztuko duen ere. Ondorioz errekurtsibitatea erabiltzean azpiprogramen exekuzio kate bat sortzen da.

[22] Hau zerrenda laburtu bat besterik ez da, Turbo Pascal lengoaiak ehundaka prozedura aurredefiniturik baitu

Dei errekurtsiboak egingo direnean, azpirrutinak baldintzazko sententzia bat izango du eta horren bitartez kontrolatu ahalko da noiz moztu dei errekurtsiboen katea. Kontutan izan dei bakoitzeko, ordenadorearen pilan azpirrutinaren itzul-helbidea, parametroak eta bertako aldagaial biltegitu behar direla.

Zenbaki natural baten faktoriala itzultzen duen funtzioa `FOR-DO` egitura iteratiboaren bitartez programatu izan dugu behin baino gehiagotan, errekurtsibitatea erabiliz helburu bera duen beste funtzio bat idatziko dugu orain. Izan ere, edozein `n` zenbakiren faktoriala F_n izendatzen badugu hau betetzen da:

$$F_n = n \times F_{n-1}$$

Non, F_{n-1} sinboloak `n-1` zenbakiaren faktoriala adierazten duen. Beste modu batez esanik `n` zenbakiaren faktoriala lortzeko bere aurreko `n-1` zenbakiaren faktorialan oinarri gaitezke, eta `n-1` zenbakiarena kalkulatzeko bere aurreko `n-2` faktorialan. Prozesua ezin da infinitoraino luzatu eta hura moztuko duen baldintzaren bat jarri beharko da. Adibidez, `4`-ren faktoriala honelaxe plantea daiteke:

```
4! = 4 x 3!
3! = 3 x 2!
2! = 2 x 1!
1! = 1
```

`1` zenbakiaren faktoriala lortzeko ez dugu aurreko formula erabiltzen. Zuzenki `1` dela baitakigu.

6.7.1 Funtzio errekurtsiboaren adibidea

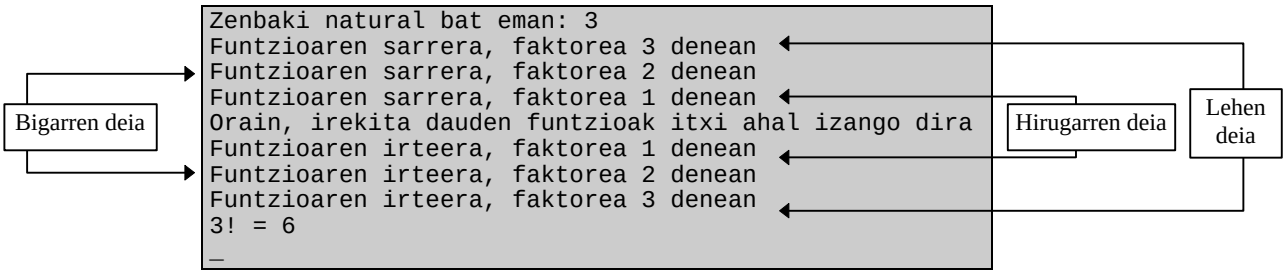
Zenbaki natural baten faktoriala kalkulatzen duen `FaktorialaErrekF()` funtzio errekurtsiboa ondoko programan idatzi da, zeinean zenbaki oso bat teklatuz irakurri ondoren bere faktoriala lortzeko `FOR-DO` bigizta iteratiborik ez dagoen:

```
PROGRAM FaktorialaErrekurtsiboa1 ;              { \TP70\06\ERREKUR1.PAS }
USES
   Crt ;

FUNCTION FaktorialErrekF (iMuga : Integer) : LongInt ;
BEGIN
   WriteLn ('Funtzioaren sarrera, faktorea ', iMuga, ' denean') ;
   IF iMuga = 1 THEN
   BEGIN
     WriteLn ('Orain, irekita dauden funtzioak itxi ahal izango dira') ;
     FaktorialErrekF := 1 ;
   END
   ELSE
     FaktorialErrekF := iMuga * FaktorialErrekF (iMuga-1) ;
   WriteLn ('Funtzioaren irteera, faktorea ', iMuga, ' denean') ;
END ;

VAR
   iZnbk : Integer ;
   liFakt : LongInt ;
BEGIN
    ClrScr ;
    REPEAT
       Write ('Zenbaki natural bat eman: ') ;
       ReadLn (iZnbk) ;
    UNTIL iZnbk > 0 ;

    liFakt := FaktorialErrekF (iZnbk) ;
    WriteLn (iZnbk, '! = ', liFakt) ;
END.
```

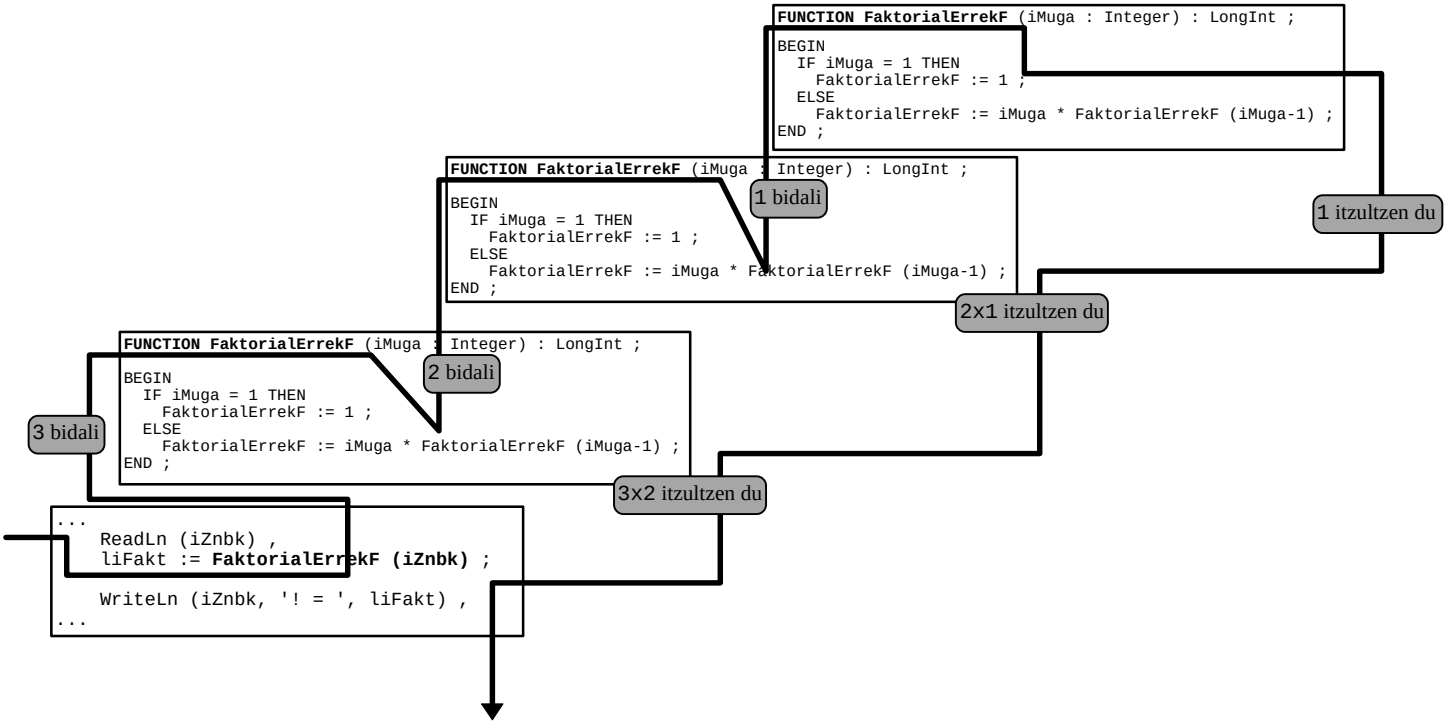
`FaktorialaErrekF()` funtzio errekurtsiboaren barruan dauden laguntza mezuak kenduz hau geratzen zaigu. Eta honela ulertzen da, `iMuga` parametroa `1` denean funtzioak `1` itzuliko du, bestela funtzioak bere buruari berriro deituko dio `iMuga-1` parametro errealarekin:

```
FUNCTION FaktorialErrekF (iMuga : Integer) : LongInt ;
BEGIN
   IF iMuga = 1 THEN
     FaktorialErrekF := 1 ;
   ELSE
     FaktorialErrekF := iMuga * FaktorialErrekF (iMuga-1) ;
END ;
```

`FaktorialaErrekurtsiboa1` programa `3` sarrerarekin exekutatzean honako irteera agertuko zaigu:

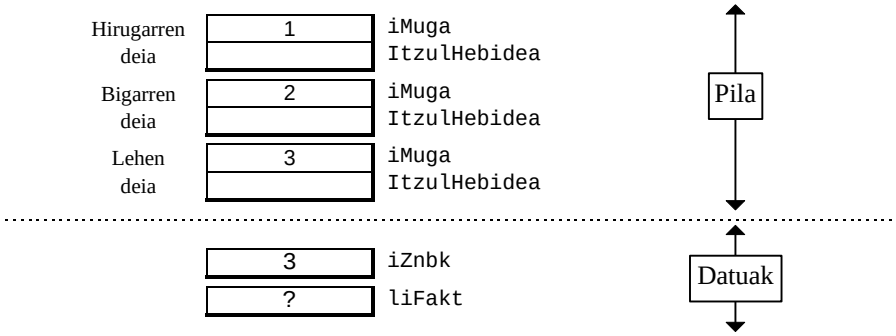


`FaktorialaErrekF()` funtziori hiru aldiz deituko zaio, behin programa nagusian eta ondoren errekurtsiboki beste bi aldiz. Hirugarren deia oraindik prozesatzen ari denean beste biak oraindik amaitu gabe egonen dira, hauxe da programaren fluxuarentzat marraz daitekeen eskema:



Azpirrutina bat deitzen denean ordenadoreak memoriaren pila darabil azpirrutinaren zeregina burutzeko. Horregatik dei errekurtsiboetan irteteko baldintzarik ez badago `202 stack overflow error` exekuzio errorea gertatzen da (pila erabat saturatu egin dela alegia).

Gure kasuan dei errekurtsiboak eteteko baldintza dagoenez ez da horrelakorik gertatuko, sarrerako `iZnbk` zenbakia `1` baino handiagoa den bitartean behintzat. Ondoriz `FaktorialaErrekF()` funtzioaren hirugarren deia oraindik prozesatzen ari denean ordenadorearen memoria mapak honako itzura izango du:



Zer gertatuko litzateke `FaktorialaErrekurtsiboa1` programaren datu sarreran `REPEAT-UNTIL` iragazkia kendu eta `0` (edo zenbaki negatibo baten) faktoriala kalkulatzen saiatuko bagina?.

6.7.2 Prozedura errekurtsiboaren adibidea

Zenbaki natural baten faktoriala kalkulatzen duen `FaktorialaErrekF()` funtzioa alda dezagun prozedura bezala eratuz, honi `FaktorialaErrekP()` deituko diogu eta bera barneratzen duen programaren identifikadorea `FaktorialaErrekurtsiboa2` izango da.

```
PROGRAM FaktorialaErrekurtsiboa2 ;              { \TP70\06\ERREKUR2.PAS }
USES
   Crt ;

PROCEDURE FaktorialErrekP (iMuga : Integer; VAR liEmaitza : LongInt) ;
VAR
  liLaguntzailea : LongInt ;
BEGIN
   WriteLn ('Prozeduraren sarrera, faktorea ', iMuga, ' denean') ;
   IF iMuga = 1 THEN
   BEGIN
     WriteLn ('Orain, irekita dauden prozedurak itxi ahal izango dira') ;
     liEmaitza := 1 ;
   END
   ELSE
   BEGIN
     FaktorialErrekP (iMuga-1, liLaguntzailea) ;
     liEmaitza := iMuga * liLaguntzailea ;
   END ;
   WriteLn ('Prozeduratik irteera, faktorea ', iMuga, ' denean') ;
END ;

VAR
   iZnbk : Integer ;
   liFakt : LongInt ;
BEGIN
    ClrScr ;
    REPEAT
       Write ('Zenbaki natural bat eman: ') ;
       ReadLn (iZnbk) ;
    UNTIL iZnbk > 0 ;
    FaktorialErrekP (iZnbk, liFakt) ;
    WriteLn (iZnbk, '! = ', liFakt) ;
END.
```

Kontzeptualki `FaktorialaErrekF()` eta `FaktorialaErrekP()` azpirrutinak berdinak dira bigarrenak konplexuagoa ematen badu ere, ez dugu horregatik ezer berririk gehituko.

10. kapituluan array edo matrizeak ikasiko ditugunean, emaniko array karratu baten determinantearen kalkulurako berriro erabiliko ditugu teknika errekutsiboak.

Ariketa bezala beste problema bat proposatuko dugu orain:

> *"Demagun `B = 1+ 3 + 6 + 9 + ...` segida daukagula, eta goimuga ezagun batetarako batugaien kopura kalkulatu nahi dela. Programa nagusian osoa den `Muga` zenbakia teklatuz irakurriko da eta `B` baturak lehen aldiz goimuga gainditzeko behar dituen batugaien kopurua, prozedura errekurtsibo baten bitartez lortuko da"*

6.8 PROGRAMAK

Hona hemen 6. kapituluaren programak orrialdeen arabera sailkaturik:

Izena	*Programaren identifikadorea*	*ORRI.*	*Ikasgaia*
FOR1.PAS	FaktorialaForBigiztarekin	6-05	Agindu errepikakorrak
KONBINAT.PAS	ZenbakiKonbinatorioa	6-06	Agindu errepikakorrak
AZPIPRG1.PAS	ZenbakiKonbinatorioa1AzpPrg	6-07	Azpirrutinak
AZPIPRG2.PAS	ZenbakiKonbinatorioa2AzpPrg	6-08	Azpirrutinak
AZPIPRG3.PAS	ZenbakiKonbinatorioa3AzpPrg	6-09	Azpirrutinak
PARAM1S.PAS	SarrerakoParametroa1	6-15	Azpirrutinak, parametroak
PARAM2S.PAS	SarrerakoParametroa2	6-16	Azpirrutinak, parametroak
PARAM3S.PAS	SarrerakoParametroa3	6-16	Azpirrutinak, parametroak
PARAM1I.PAS	IrteerakoParametroa1	6-19	Azpirrutinak, parametroak
PARAM2I.PAS	IrteerakoParametroa2	6-21	Azpirrutinak, parametroak
PARAM3I.PAS	IrteerakoParametroa3	6-23	Azpirrutinak, parametroak
PARAM1SI.PAS	SarreraIrteerakoParametroa1	6-24	Azpirrutinak, parametroak
PARAM4S.PAS	SarrerakoParametroa4	6-27	Azpirrutinak, parametroak
PARAKONS.PAS	KonstanteParametroa	6-33	Azpirrutinak, parametroak
IKUS.PAS	Ikus	6-37	Unitateak
HELBID1.PAS	AldagaiNagusienHelbideak	6-37	Azpirrutinak, helbideak
HELBID2A.PAS	ParametroFormalenHelbideakA	6-39	Azpirrutinak, helbideak
HELBID2B.PAS	ParametroFormalenHelbideakB	6-39	Azpirrutinak, helbideak
HELBID3A.PAS	ParametroFormalenHelbideakVAR_A	6-41	Azpirrutinak, helbideak
HELBID3B.PAS	ParametroFormalenHelbideakVAR_B	6-41	Azpirrutinak, helbideak
HELBID4A.PAS	ParametroFormalenHelbideakCONST_A	6-42	Azpirrutinak, helbideak
HELBID4B.PAS	ParametroFormalenHelbideakCONST_B	6-42	Azpirrutinak, helbideak
IRAUPEN.PAS	AldagaienIraupena	6-44	Azpirrutinak
ESPARRU0.PAS	EsparruaAztertzen0	6-46	Azpirrutinak
ESPARRU1.PAS	EsparruaAztertzen1	6-47	Azpirrutinak
ESPARRU2.PAS	AzpirrutinaBanatuak_Esparrua	6-48	Azpirrutinak
ESPARRU3.PAS	AzpirrutinaKabiatuak_Esparrua	6-48	Azpirrutinak
ESPARRU4.PAS	IdentifikadoreenLehentasuna	6-50	Azpirrutinak
ESPARRU5.PAS	AldagaiOrokorra	6-51	Azpirrutinak
FUNTZDEI.PAS	FuntzioarenDeiak	6-55	Azpirrutinak, funtzioak
FUNADIB1.PAS	TaylorFuntzioz1	6-57	Azpirrutinak, funtzioak
BIHURK1.PAS	RadianakBihurtu1	6-58	Azpirrutinak, funtzioak
LEHEN3.PAS	ZenbakiLehenakPantailaratzen	6-60	Azpirrutinak, funtzioak
FUNADIB2.PAS	TaylorFuntzioz2	6-66	Azpirrutinak, prozedurak

FUNADIB3.PAS	TaylorFuntzioz3	6-68	Azpirrutinak, prozedurak
BIHURK2.PAS	RadianakBihurtu1	6-70	Azpirrutinak, prozedurak
JAUZIKA.PAS	PilotaJauzika	6-71	Azpirrutinak, prozedurak
ERREKUR1.PAS	FaktorialaErrekurtsiboa1	6-73	Errekurtsibitatea
ERREKUR2.PAS	FaktorialaErrekurtsiboa2	6-75	Errekurtsibitatea
ERREKUR3.PAS	BatuketaErrekurtsiboa	6-76	Errekurtsibitatea

6.9 BIBLIOGRAFIA

- Salmon, W., *Introducción a la computación con Turbo Pascal. Estructuta y abstracciones*, Addison-Wesley Iberoamericana, Wilmington, 1993
- Wirth, N., *Algoritmos + Estructuras de Datos = Programas*, Ed. Castillo, 1986
- Decker R., Hirshfield S., *Pascal's Triangle*, PWS-Kent Publishing Company, Boston, 1992
- Borland, *TURBO PASCAL 7.0. Erabiltzailearen gida*, Borland International, 1992
- Borland, *TURBO PASCAL 7.0 Ingurunearen laguntza*, Borland International, 1992
- Leetsma, S., Nyhoff, L., *Programación en Pascal*, Prentice Hall, Madrid, 1999

7. ATALA: UNITATEAK

AURKIBIDEA

7.1 SARRERA

Azpirrutinek programen modulaketa eragiten duten bezala, unitateak ere programen antolaketarako tresna dira.

Turbo Pascal lengoaiak aurredefiniturik dituen konstante, datu-mota, aldagai, funtzio eta prozedura unitate deituriko software elementu batzuetan bildurik aurkitzen dira. Asko direnez eta norberaren programa batean denak erabili behar izatea arraroa izango denez, une bakoitzean benetan behar direnak erabil daitezen, taldeka eskaintzen ditu konpiladoreak.

Turbo Pascal lengoaiak aurredefiniturik dituen konstante, datu-mota, aldagai, funtzio eta prozedura darabilten gure programak aplikazio-programak edo bezero-programak deituko ditugu. Aplikazio-programek azpirrutina estandar batetaz baliatzeko dagokion unitatearen izena `USES` klausula ostean jarriko du bere hasieran.

Esan dugunez, aurredefinituriko azpirrutinen helburuak zeharo ezberdinak dira, adibidez: funtzio matematikoak (`Sin()` , `Exp()`, `Ln()`, ...); kurtsore eta pantailaren kontrola (`ClrScr` , `GotoXY()`, `CloseGraph`, ...); karaktere kateen prozesaketa (`Copy()`, `Delete()`, `Concat()`, ...); sistema eragiletik informazioa jasotzea (`DiskFree()`, `DiskSize()`, `GetDir()`, ...). Horregatik azpirrutinak konpilatu ondoren *unitateak* deituriko software elementuetan gordetzen dira, beraz unitate bat konpilaturiko azpirrutinen bilduma bat da. Turbo Pascal lengoaiaren unitateak `.TPL` eta `.TPU` luzapenak dituzten fitxategietan gordetzen dira diskoan.

Jarraian Turbo Pascal-aren unitate estandarrak enumeratzen dira:

Unitatea	Fitxategia diskoan	Deskribapen laburra
`System`	TURBO.TPL	Behemailako errutinak
`Dos`	TURBO.TPL	DOS-erekiko interfazerako errutinak
`Crt`	TURBO.TPL	Sarrera/irteera errutinak
`Printer`	TURBO.TPL	Inprimagailurako irteera
`Overlay`	TURBO.TPL	Estalpe kudeaketa
`Strings`	STRINGS.TPL	Karaktere hutsean amaituriko kateak
`WinDos`	WINDOS.TPU	DOS-erekiko interfazerako errutinak
`Graph`	GRAPH.TPU	Interfaze grafikorako errutinak
`Graph3`	GRAPH3.TPU	3.0 bertsioaren azpirrutina grafikoak
`Turbo3`	TURBO3.TPU	3.0 bertsioarekiko bateragarritasuna

Sistema informatikoaren domeinu ezberdinetan aproposak diren azpirrutina bereziak unitate estandarrek barneratzen dituzte. Aurreko zerrendaren lehenengo bost unitateak TURBO.TPL liburutegi-fitxategian aurkitzen dira, eta memorian automatikoki kokatzen dira Turbo Pascal kargatzean. Azken unitate biek, konpiladorearen bertsio zaharren programak integratzeko balio dute.

Arestian aipatu den bezala, konpilaturik dagoen unitatea gordetzen duen fitxategiak `.TPU` (Turbo Pascal Unit) luzapena izango du. TPU fitxategi batek unitate bakat bat barneratzen du, eta aplikazio-programari eztekaketa (linkaketa) prozesuan gehituko zaio. Konpilaturiko unitate ezberdinak fitxategi bakar batean gordetzean errutinen liburutegi bat sortzen da, eta fitxategiak izango duen luzapena `.TPL` (Turbo Pascal Library) izango da.

Horrela, pantaila garbitzen duen `ClrScr` prozedura edo `ReadKey` funtzio estandarrak aplikatu ahal izateko lehenago `Crt` unitatea deklaratu behar da, euren kode konpilatua bertan baitago; gero, erabiltzailearen iturburu-programatik makina-lengoaia lortzeko itzulpen prozesuan, programadorearen sententzia konpilatuak eta `Crt` unitatean daudenak linkatu egingo dira programa exekutagarria lortzeko. Gauza bera esan daiteke `ReadLn()`, `WriteLn()`, `Cos()` eta horrelako beste zenbait azpiprogramei buruz, alegia, azpirrutina horiek barneratzen

dituzten sententziak unitateren batean daudela, `System` izeneko unitatean aurkitzen dira baina `System` unitatea beti linkatzen denez ez da `USES` klausula ostean deklaratzen.

Zazpigarren kapitulu honetan, erabiltzen ezezik, gure unitateak sortzen ikasiko dugu.

7.1.1 Turbo Pascal eta grafikoak

Esandakoaren adibide gisa Turbo Pascal lengoaiak grafikoak lantzeko zenbait funtzio eta prozedura garaturik ditu eta horiek erabiltzeko `Graph` izeneko unitate deklaratu behar da. Une honetan helburua ez da azpirrutina grafikoak sakonki ezagutzea, gutxi batzuk ikasiko ditugu eta geuk beste hainbat sortuko dugu. Geure azpirrutina grafikoak `Grafiko` deituko dugun unitate batean bilduko ditugu.

Beraz, gure helburua `Grafiko` izena izango duen unitatea sortzea izango da.

7.1.1.1 Grafikoak irekitzen eta ixten

Programa grafikoekin hasteko `GrafikoakLantzen0` izena duen adibide-programa aurkeztuko dugu. Bertan agertzen diren kontzeptu berriak bost azpirrutina dira: `InitGraph()`, `CloseGraph` eta `OutTextXY()` prozedurak, eta, `GraphResult` eta `GraphErrorMsg()` funtzioak. Kontutan izan, azpirrutinok konpiladorearentzat ezagunak izan daitezen `Graph` unitatearen erabilpena aplikazio-programaren hasieran erazagutu behar dela.

`CloseGraph` prozedurak pantaila grafikoa ixteko balio du eta ez du inolako parametrorik behar.

`VGA`, `VGAMed` konstanteak dira eta `Graph` unitatean definiturik daude.

`InitGraph()` prozedurak pantaila grafikoa irekitzeko da. Bereziak eta espezifikoak ez diren ordenadoreek ezin dute grafikoak marrazteko behar diren seinaleak sortu, horretarako aparteko zirkuitu elektroniko bat derrigorrezkoa dute adaptodore grafiko edo txartel grafiko deitzen dena. Monitore bakoitza txartel grafiko jakin bati loturik izaten da adibidez VGA monitorei dagokien txartel grafikoa VGA txartela da; baina edozein txartel grafikok bera baino apalagoak diren grafiko motak emula ditzakeenez, VGA txartela duen ekipo batek CGA eta EGA txartelak emula (imita) ditzake.

Txartel grafikoetan memoria txipak daude, eta memoria kopuruak markatzen du txartelaren ahalmena (CGA txartelak 16 Kbyte ditu, EGA-k 64 Kbyte, ...), memoria kopuruak mugatzen du zenbat pixel[1] eta zenbat kolore biltegi daitekeen txartelan. Txartelan memoria nahikorik dagoenean, posible da pantailarakada bat baino geihago aldiberean gorderik edukitzea, hurrengo taulan bereizmenaren[2] (pixelen kopurua) eta kolorearen araberako sailkapena egiten da memoria beharra erakutsiz:

[1] Pixela ingelesezko “picture element” hitzen laburtzapen fonetikoa da eta, definizioz , iduri baten elementurik txikiena litzateke, indibidualki har daitekeena kolorea edo intentsitatea aplikatzeko edo besteengandik bereizteko.

[2] Bi neurriren artean bereiz daitekeen saltorik txikiena bereizmena litzateke. Pantailaren bereizmena elementu txikienak (pixelak) adieraziko du, zenbat eta pixel gehiago izan (monitorearen neurri fisikoak aldatzen ez direnez) irudian xehetasun gehiago nabarituko da.

	Koloreak			
Bereizmena (zutabe x lerro)	2	4	16	256
320 x 200	8 Kb	16 Kb	32 Kb	64 Kb
640 x 200	16 Kb	32 Kb	64 Kb	128 Kb
640 x 350	32 Kb	64 Kb	128 Kb	256 Kb
640 x 480	64 Kb	128 Kb	256 Kb	512 Kb

Gure programek txartel grafikoaz balia daitezen ordenadorearen eta txartelaren artean interfaz-softwarea behar da. Turbo Pascal 7.0 lengoaiak zenbait adaptodore grafikoekin lan egin ahal izateko kontrolagailu[3] programak eskaintzen ditu, bakoitzari zenbaki bat dagokio (`Graph` unitatean izenak ere definiturik daude) eta ondoko taulan biltzen dira:

Identifikadorea	Balioa	Txartela	Kontrolagailua
`Detect`	`0`	-	-
`CGA`	`1`	Color Graphics Adapter	`CGA.BGI`
`MCGA`	`2`	Multicolor Graphics Adapter	`CGA.BGI`
`EGA`	`3`	Enhaced Graphics Adapter	`EGAVGA.BGI`
`EGA64`	`4`	Enhaced Graphics Adapter	`EGAVGA.BGI`
`EGAmomo`	`5`	Enhaced Graphics Adapter	`EGAVGA.BGI`
`IBM8514`	`6`	IBM PC 8514 Graphics	`IBM8514.BGI`
`HERCmomo`	`7`	Hercules monocromo	`HERC.BGI`
`ATT400`	`8`	AT & T 6300 Adapter	`ATT.BGI`
`VGA`	`9`	Video Graphics Adapter	`EGAVGA.BGI`
`PC3270`	`10`	IBM 3270 PC	`PC3270.BGI`

[4]

`InitGraph()` prozedurak behar duen datuetatik bat, ekipoan zer txartel instalaturik dagoen izango da. Beraz, aurreko taularen lehendabiziko bi zutabeetan erakusten dena eman behar zaio (kasurik kasu), instalaturiko txartela jakin ezean `0` edo `Detect` eman diezaiokegu berak identifika eta dagokion kontrolagailua karga dezan.

Txartel grafikoak *modu* ezberdinetan erabil daitezke. Modu jakin batek bereizmena, kolore kopurua eta grafikoetarako memoria kantitatea zehazten ditu. Adibidez, VGA adaptadore grafikoak ondoko hiru moduak definiturik ditu:

VGA adaptodere grafikoaren moduak				
Identifikadorea	**Balioa**	**Bereizmena**	**Kolor.**	**Orriak**
`VGAlo`	`0`	640 x 200	16	4
`VGAmed`	`1`	640 x 350	16	2
`VGAhi`	`2`	640 x 480	16	1

Txartel baten moduak balio oso batez izendatzen dira, baina Turbo Pascal 7.0 lengoaiak VGA adaptadore grafikoari dagozkion moduak aukeratzeko aurreko taularen lehenengo zutabeko konstanteak aurredifiniturik ditu `Graph` unitate estandarrean.

[3] Pixela ingelesezko “picture element” hitzen laburtzapen fonetikoa da eta, definizioz , iduri baten elementurik txikiena litzateke, indibidualki har daitekeena kolorea edo intentsitatea aplikatzeko edo besteengandik bereizteko.
[4] Pixela inglesezko “picture element” hitzen laburtzapen fonetikoa da eta, definizioz , iduri baten elementurik txikiena litzateke, indibidualki har daitekeena kolorea edo intentsitatea aplikatzeko edo besteengandik bereizteko.

`InitGraph()` prozedurak behar duen bigarren datua modua litzatekeenez, kontrolagailuak onartzen duen bat eman beharko zaio bigarren parametro bezala. VGA adaptodore batekin lan eginez gero aurreko taulako lehen bi zutabeetan erakusten diren konstanteetatik bat aukeratu beharko da.

`InitGraph()` prozedurak behar duen hirugarren, eta azken, datua interfaz programen kokamena diskoan izango da. Turbo Pascal konpiladorea diskoan instalatzen denean direktorio jakin batean gordetzen da, adibidez `TP70` izena duen direktorioan, adaptadore grafikoen kontrolagailuak `TP70` direktorioaren menpeko den `BGI` direktorio batean aurkituko dira. Ondorioz, kontrolagailuen irispidea `'C:\TP70\BGI'` delarik, CGA txartel batekin `0` moduan lan egiteko honelako sententzia idatziko genuke:

```
InitGraph (CGA, 0, 'C:\TP70\BGI')
```

`GraphResult` funtzioak azken eragiketa grafikoaren errore-kodea itzultzen du, errorerik ez dagoenean `0` itzultzen du.

`GraphErrorMsg()` funtzioak errore-kode bat jasotzen du eta erroreari dagokion mezu testuala itzultzen du.

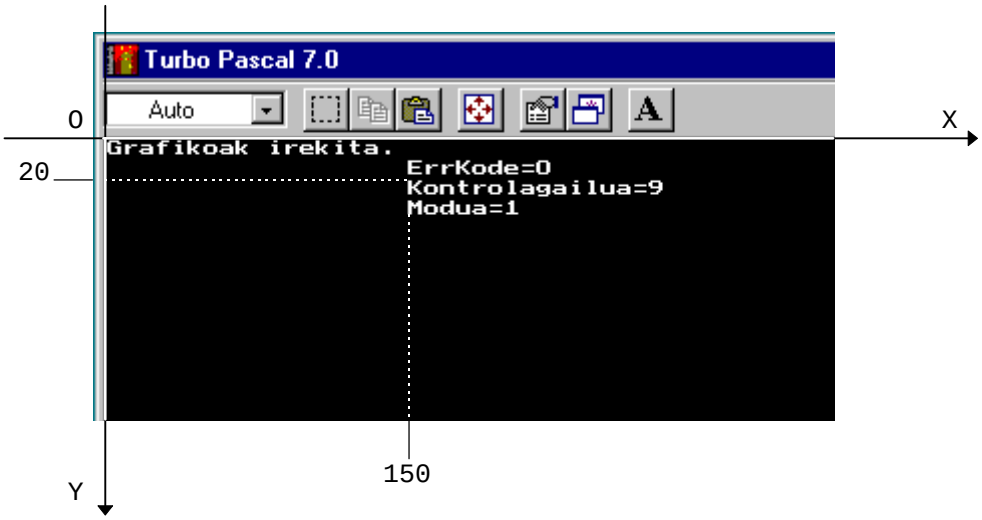
`OutTextXY()` prozeduraren bitartez pantaila grafikoan karaktere kate bat marrazten du. Dagozkion lehen bi parametroak pantailaren puntu baten koordenatuak dira, eta marrazketa non hasiko den adierazten dute. Hirugarren parametroa marraztu nahi den esaldia da, zenbakizko aldagai baten balioa pantailaratu ahal izateko zenbakia string bihurtu beharko da (horretarako `Str(zbk,kate)` prozedura estandarra existitzen da).

Esandako guztia kontutan izanik erraz ulertzen da `GrafikoakLantzen0` izena duen adibide-programa honen kodea:

```
PROGRAM GrafikoakLantzen0 ;                    { \TP70\07\GRAF_0.PAS }
USES
   Crt, Graph ;
VAR
   Kontrolagailua, Modua : Integer ;
   Mezua : String ;
   Itxaron : Char ;
   ErrKode : Integer ;

BEGIN
   Kontrolagailua := VGA ;
   Modua := VGAmed ;
   InitGraph (Kontrolagailua, Modua, 'C:\tp70\bgi') ;
   ErrKode := GraphResult ;

   IF ErrKode = 0 THEN
   BEGIN
      OutTextXY (0, 0, 'Grafikoak irekita.') ;
      Str (ErrKode, Mezua) ;
      OutTextXY (150, 10, 'ErrKode=' + Mezua) ;
      Str (Kontrolagailua, Mezua) ;
      OutTextXY (150, 20, 'Kontrolagailua=' + Mezua) ;
      Str (Modua, Mezua) ;
      OutTextXY (150, 30, 'Modua=' + Mezua) ;
      Itxaron := ReadKey ;
      CloseGraph ;
   END
   ELSE
   BEGIN
      Writeln ('Errore grafikoaren kodea ---> ErrKode=', ErrKode) ;
      WriteLn ('Errore grafikoaren mezua ---> ', GraphErrorMsg (ErrKode)) ;
   END ;
END.
```


`InitGraph()` prozedurak arazorik gabe lan egiterik badu `GraphResult` funtzioak `0` itzultzen digu, bestela zenbaki oso bat emanen du zein `GraphErrorMsg()` funtzioari pasa diezaiokegun.

`GrafikoakLantzen0` programa exekutzean honelako zerbait aterako litzateke. Non, ikusten denez, `OutTextXY()` prozedurari esker karaktere kateak pantailaren koordenatu zehatzetan agertzen diren, pantailaren goi ezkerreko erpina `(0,0)` pixela da eta `OX` ardatza horizontala da:



Baina gerta daiteke gure ordenadorearen Turbo Pascal konpiladorea `C:\TP70` ez den beste toki batean instalaturik egotea (beste disko unitate batean edo beste direktorio batean), horrelakoetan `InitGraph()` prozedurak ezin izango du adaptadore grafikoaren kontrolagailua aurkitu eta `IT-THEN-ELSE` egituran `ELSE`-ren bidea aukeratuko litzateke honelako zerbait erakutsiz:

```
Errore grafikoaren kodea ---> ErrKode=-3
Errore grafikoaren mezua ---> Device driver file not found (EGAVGA.BGI)
_
```

`GrafikoakLantzen0` programa horren bitartez kontrolagailu eta modu ezberdinak experimenta daitezke, horretarako `InitGraph()` prozeduraren aurrean dauden esleipen bietan konstante egokiak jar daitezke, adibidez:

```
Kontrolagailua := CGA ;
Modua := 0 ;
```

Hona hemen `GrafikoakLantzen0` programak erabiltzen dituen azpirrutina grafikoak:

Azpirrutina	Mota	Deskribapen laburra
`InitGraph()`	Proze.	Makinan instalaturik dagoen hardwarearen arabera sistema grafikoa hasieratzen du
`CloseGraph`	Proze.	Sistema grafikoa ixten du
`GraphResult`	Funtz.	Azken eragiketa grafikoari dagokion errore-kodea itzultzen du
`GraphErrorMsg()`	Funtz.	Errore-kode baten mezu testuala itzultzen du
`OutTextXY()`	Proze.	String bat irteerako periferikora igortzen du

7.1.1.2 Pantaila testuala vs. pantaila grafikoa

Aurreko adibidean, kontrolagailua CGA eta modua 0 ariketa egitean askoz pixel handiagoak ateratzen direla konturatzen gara, kasu horri dagokion bereizmena 320x200 baita. Programa grafikoen bigarren adibidean pantaila testual eta grafiko artean konmutazioa ikasiko dugu (bide batez adaptadore kontrolagailuak eta moduak lantzen jarraituko dugu), horretarako `GrafikoakLantzen1` eta `GrafikoakLantzen2` izeneko programak prestatu ditugu. Bertan agertzen diren elementu berriak jarraian agertzen den taulan ematen dira:

Azpirrutina	Mota	Deskribapen laburra
`GetModeRange()`	Proze.	Adaptadore grafikoaren kontrolagailuari dagozkion modurik apalena eta garaiena itzultzen ditu
`GetDriverName`	Funtz.	Adaptadore grafikoaren kontrolagailuari dagokion izena string bezala itzultzen du
`GetMaxX`	Funtz.	Indarrean aurkitzen diren kontrolagailu eta moduari dagokien eskuineko zutabearen zenbakia
`GetMaxY`	Funtz.	Indarrean aurkitzen diren kontrolagailu eta moduari dagokien beheko lerroaren zenbakia
`RestoreCrtMode`	Proze.	Sistema grafikoa ireki baino lehenago egon den pantaila mota berreskuratzen du
`SetGraphMode()`	Proze.	Pantaila garbitu eta sistema grafikoan ezartzen du

`GrafikoakLantzen1` eta `GrafikoakLantzen2` izeneko programen hasierak aurreko puntuan azaldu den `GrafikoakLantzen0` programarekin alderatuz, antzekoak direla ohartzen gara. Kontutan izan `grOK` eta `Detect` konstanten balioak zero direla eta haiek erabiltzeko ahalmena `Graph` unitatea deklaratu izateak ematen digula.

`GrafikoakLantzen1` programak ordenadorean instalaturik dagoen txartel grafikoa detektatu ondoren dagokion kontrolagailua eta modurik garaiena ezartzen ditu. Gero modu maximo eta minimoak lortzen ditu, eta, jarraian pantailaren bereizmena kalkulatzen da.

`RestoreCrtMode` prozedurak sistema grafikotik testu-pantailara igarotzeko aukera ematen digu, eta berriro sistema grafikoa ezartzeko `SetGraphMode()` aplika daiteke. Sistema biren artean konmutazioa lortzeko hobe da `SetGraphMode()`-`RestoreCrtMode` bikoteaz egitea eta ez `InitGraph()`-`CloseGraph` prozeduren bitartez.

Hauxe da `GrafikoakLantzen1` programaren kodea:

```
PROGRAM GrafikoakLantzen1 ;                          { \TP70\07\GRAF_1.PAS }
USES
 Crt, Graph ;
VAR
 Kontrolagailua, Modua, ModuApala, ModuGaraia, Altuera, Zabalera : Integer ;
 Mezua : String ;
 Itxaron : Char ;
BEGIN
   Kontrolagailua := Detect ;               { Detect=0    detekta dezala }
   InitGraph (Kontrolagailua, Modua, 'C:\tp70\bgi') ;

   IF GraphResult <> grOk THEN
        Writeln (#7, 'Erroren bat grafikoak irekitzean') ;

   Str (Kontrolagailua, Mezua) ;
   OutTextXY (0, 10, 'Kontrolagailuaren kodea ' + Mezua) ;
   Mezua := GetDriverName ;
   OutTextXY (0, 20, 'Kontrolagailuaren izena ' + Mezua) ;
   Str (Modua, Mezua) ;
   OutTextXY (0, 30, 'Modua ' + Mezua) ;
```

```
   GetModeRange (Kontrolagailua, ModuApala, ModuGaraia) ;
   Str (ModuApala, Mezua) ;
   OutTextXY (0, 50, 'Modurik apalena ' + Mezua) ;
   Str (ModuGaraia, Mezua) ;
   OutTextXY (0, 60, 'Modurik altuena ' + Mezua) ;

   Zabalera := GetMaxX ;    { lehen pixela (0,0) delako }
   Altuera := GetMaxY ;     { lehen pixela (0,0) delako }

   Str (Zabalera, Mezua) ;
   OutTextXY( 0, 70, 'Zabalera, X maximoa = ' + Mezua) ;
   Str (Altuera, Mezua) ;
   OutTextXY (0, 80, 'Altuera,  Y maximoa  = ' + Mezua) ;

   OutTextXY (0, 110, 'TESTU-PANTAILA KONMUTATU') ;
   OutTextXY (0, 115, '------------------------') ;

   Itxaron := ReadKey ;

   RestoreCrtMode ;

   REPEAT
     Write ('0, 1 edo 2 moduetatik zein nahi duzu? ') ;
     ReadLn (Modua) ;
   UNTIL (Modua >= 0) AND (Modua <= 2) ;

   SetGraphMode (Modua) ;

   OutTextXY (0, 10, 'PANTAILA GRAFIKOA BERRRESKURATUTA') ;
   OutTextXY (0, 15, '---------------------------------') ;

   Zabalera := GetMaxX ;    { lehen pixela (0,0) delako }
   Altuera := GetMaxY ;     { lehen pixela (0,0) delako }

   Str (Modua, Mezua) ;
   OutTextXY (0, 40, 'Modua ' + Mezua) ;
   Str (Zabalera, Mezua) ;
   OutTextXY( 0, 50, 'Zabalera, X maximoa = ' + Mezua) ;
   Str (Altuera, Mezua) ;
   OutTextXY (0, 60, 'Altuera,  Y maximoa = ' + Mezua) ;

   Itxaron := ReadKey ;
   CloseGraph ;
END.
```

`GrafikoakLantzen1` programan kontrolagailua detektatu eta sistema grafikoaren modua alda daiteke. Bestean, `GrafikoakLantzen2` deritzon programan, ordenadorean instalaturiko adaptadore grafikoaren kontrolagailua eta modua alda daitezke, kodeak oso berdintsuak direnez ez ditugu `GrafikoakLantzen2` programaren sententziak testuan tartekatuko, norberak egin dezan ariketa bezala honela planteada daiteke:

> *“Sistema grafikoa detektatzeko gai den programa bat idatzi. Testu-pantailara konmutatu eta kontrolagailu bat eta modu bat aukeratzeko posibilitatea eman. Hartutako datuekin sistema grafikora itzuli”*

7.1.1.3 Pixelak

Programa grafikoen hirugarren adibidean, `GrafikoakLantzen3` deritzonean, aipatu dugun pixel kontzeptua lantzen aurkezten da `PutPixel()` prozedura estandarren bitartez. `PutPixel()` prozeduraren zeregina pixelak pantailan marraztea da, horretarako pantaila grafikoaren puntu baten koordenatuak eta puntuaren kolorea behar dira.

Jarraian `GrafikoakLantzen3` programaren sententziak erakusten dira:

```
PROGRAM GrafikoakLantzen3 ;  {pixelak}              { \TP70\07\GRAF_3.PAS }
USES
   Crt, Graph ;
VAR
   Kontrolagailua, Modua, Altuera, Zabalera : Integer ;
   Mezua : String ;
   Itxaron : Char ;
BEGIN
   Kontrolagailua := Detect ;                { Detect=0    detekta dezala }
   InitGraph (Kontrolagailua, Modua, 'C:\tp70\bgi') ;

   IF GraphResult <> grOk THEN
        Writeln (#7, 'Erroren bat grafikoak irekitzean') ;

   Zabalera := GetMaxX ;
   Altuera := GetMaxY ;

   PutPixel (0, 0, 11) ;
   PutPixel (Zabalera, 0, 12) ;
   PutPixel (0, Altuera, 13) ;
   PutPixel (Zabalera, Altuera, 14) ;
   PutPixel (Zabalera DIV 2, Altuera DIV 2, 15) ;

   Itxaron := ReadKey ;
   CloseGraph ;
END.
```

Ikusten denez `PutPixel()` prozeduraren hirugarren parametroa zenbaki oso bat da, zeinek koloreren bat adierazten duen. Jarraian datorren **7.1.1.4 Koloreak** puntuan koloreen taula lortzen da.

7.1.1.4 Koloreak

Sistema grafiko batean testuen eta marrazkien kolorea, adaptadorearen arabera, aukeragarria da. Esate baterako, VGA txartel batek (dauzkan edozein hiru modutan) onartzen dituen koloreen kopurua 16 da baina CGA adapatadore batek bakarrik lau kolorekin lan egiteko ahalmena du 0, 1, 2, eta 3 moduetan eta bi koloreekin 4 moduan. Taula honetan agertzen den informazioren zuzentasuna `GrafikoakLantzen4` programaz experimentalki froga daiteke:

VGA eta CGA adaptodere grafikoen koloreak					
Kontrolag.		**Modua**		**Bereizmena**	**Kolor.**	**Paleta**
VGA	9	VGAlo	0	640 x 200	16	-
VGA	9	VGAmed	1	640 x 350	16	-
VGA	9	VGAhi	2	640 x 480	16	-
CGA	2	CGAco	0	320 x 200	4	C0
CGA	2	CGAc1	1	320 x 200	4	C1
CGA	2	CGAc2	2	320 x 200	4	C2
CGA	2	CGAc3	3	320 x 200	4	C3
CGA	2	CGAhi	4	640 x 200	2	-

`GrafikoakLantzen4` izena duen programan dauden azpirrutina grafiko berriak lau hauek dira: `GetMaxColor`, `SetBkColor()`, `SetColor()` eta `GraphDefaults`.

`GetMaxColor` funtzio honek, instalaturiko hardwarearen arabera, `SetColor()` edo/eta `SetBkColor()` prozedurei pasa diezaiekegun baliorik garaiena itzultzen du.

`SetBkColor()` prozedura honek hondoaren kolorea finkatzen du.

`SetColor()` prozedurak marrazkiaren kolorea finkatzen du.

`GraphDefaults` kurtsorearen erakuslea hasieran kokatzen du eta sistema grafikoaren balio lehenetsiak hartzen ditu.

Jarraian ematen den `GrafikoakLantzen4` programaren sententzietan hasierako esleipenek sistema grafikoa nola irekiko den zehazten dute, aurreko taula aintzat harturik horiek aldatuz irteera ezberdinak ikus daitezke:

```
PROGRAM GrafikoakLantzen4 ;                                    { \TP70\07\GRAF_4.PAS }
USES
   Crt, Graph ;
VAR
   Kontrolagailua, Modua, Altuera, Zabalera : Integer ;
   Mezua : String ;
   Kol : Integer ;
   Itxaron : Char ;
BEGIN
   Kontrolagailua := Detect ;   { VGA edo CGA konstanteak jar daitezke }
   Modua := 0 ;                 { konstante hau ere alda daiteke }
   InitGraph (Kontrolagailua, Modua, 'C:\tp70\bgi') ;

   IF GraphResult <> grOk THEN
        Writeln (#7, 'Erroren bat grafikoak irekitzean') ;

   FOR  Kol:=0 TO GetMaxColor DO
   BEGIN
     IF Kol=0 THEN
     BEGIN
         SetBkColor (Kol+1) ;
         SetColor (GetMaxColor) ;
         OutTextXY (100, 20, 'Benetako 0 kolorea beltza da') ;
     END
     ELSE
         SetBkColor(0) ;

     SetColor (Kol) ;
     Str (Kol, Mezua) ;
     Mezua := 'Testu hau ' + Mezua + '. kolorean dago' ;
     OutTextXY (100, 10*(Kol+3), Mezua) ;
     Itxaron := ReadKey ;
   END ;
   Itxaron := ReadKey ;

   GraphDefaults ;
   OutTextXY (0, GetMaxY-30, 'Hau da kolore lehenetsia.    Tekla bat sakatu amaitzeko') ;
   Itxaron := ReadKey ;
   CloseGraph ;
END.
```

7.1.1.5 Letra-tipoak eta estiloak

Orain arteko adibide grafikoetan agertu den idazmoldea beti berdina izan da, baina Turbo Pascal-ek testuak kudeatzeko zenbait azpirrutina ditu eta horiek erabiltzen ikasiko dugu. Testuaren moldea zehazteko funtsezko prozedura `SetTextStyle()` deitzen da eta hiru parametro behar ditu (letra-tipoa, testuaren norabidea eta letra tamaina), `GrafikoakLantzen5`

izeneko programan horrekin batera `ClearViewPort` prozedura agertzen da zein pantaila grafikoa garbitzeko balio duen. Taula honetan `SetTextStyle()` prozedurak onartzen dituen konstanteak eta balioak biltzen dira:

SetTextStyle (Font, Norabide, Tamaina : Word)				
Font		**Norabide**		**Tamaina**
Identif.	**Balioa**	**Identif.**	**Balioa**	**Balioa**
DefaultFont	0	HorizDir	0	1..10
TriplexFont	1	VertDir	1	
SmallFont	2			
SansSerifFont	3			
GothicFont	4			

Taularen datu horiekin `FOR-DO` egitura batean `GrafikoakLantzen5` programak molde ezberdineko mezuak aterako ditu.

```
PROGRAM GrafikoakLantzen5 ;                    { \TP70\07\GRAF_5.PAS }
USES
   Crt, Graph ;
VAR
   Kontrolagailua, Modua, Altuera, Zabalera : Integer ;
   Mezua : String ;
   Font : Integer ;
   Itxaron : Char ;
BEGIN
   Kontrolagailua := 0 ;                          { Detekta dezala }
   InitGraph (Kontrolagailua, Modua, 'C:\tp70\bgi') ;

   IF GraphResult <> grOk THEN
        Writeln (#7, 'Erroren bat grafikoak irekitzean') ;

   FOR Font:=0 TO 4 DO
   BEGIN
      ClearViewPort ;
      SetTextStyle (Font, 0, 5) ;
      SetColor (Font+1) ;

      Str (Kontrolagailua, mezua) ;
      OutTextXY (0, 40, 'Kontrolagailua ' + Mezua) ;
      Str (Modua, mezua) ;
      OutTextXY (0, 80, 'Modua ' + Mezua) ;

      Zabalera := GetMaxX ;    { lehen pixela (0,0) delako }
      Altuera := GetMaxY ;     { lehen pixela (0,0) delako }

      Str (Zabalera, mezua) ;
      OutTextXY (0, 120, 'X maximoa edo zabalera ' + Mezua) ;
      Str (Altuera, mezua) ;
      OutTextXY (0, 160, 'Y maximoa edo altuera  ' + Mezua) ;

      SetTextStyle (Font, 1, 5) ;               { testu bertikala }
      OutTextXY (0, 220, 'Bertikalki') ;

      Itxaron := ReadKey ;
   END ;
   CloseGraph ;
END.
```

Testuak justifika daitezke, hau da, `OutTextXY()` prozedurak `X` eta `Y` koordenatuetan testua ateratzerakoan erreferentziak zehaz daitezke. Adibidez, puntutik eskuinetara ala ezkerretara ala zentraturik, edo, puntuaren gainean ala behean ala zentraturik. Horretarako `SetTextJustify()` prozedura dago, hona hemen bere bi parametroen sarrerak:

SetTextJustify (Horizontal, Bertikal : Word)			
Horizontal		**Bertikal**	
Identif.	**Balioa**	**Identif.**	**Balioa**
LeftText	0	BottomText	0
CenterText	1	CenterText	1
RightText	2	TopText	2

Azken taularen datuak erabiltzen dituen `GrafikoakLantzen6` programa jarraian ematen da, zeinean hiru mezuren kokapen ezberdinak eskatzen diren:

```
PROGRAM GrafikoakLantzen6 ;          { \TP70\07\GRAF_6.PAS }
USES
   Crt, Graph ;
VAR
   Kontrolagailua, Modua : Integer ;
   NonX, NonY : Integer ;
   Itxaron : Char ;
BEGIN
   Kontrolagailua := VGA ;
   Modua := VGAmed ;
   InitGraph (Kontrolagailua, Modua, 'C:\tp70\bgi') ;

   IF GraphResult <> grOk THEN
        Writeln (#7, 'Erroren bat grafikoak irekitzean') ;

   NonX := 200 ;
   NonY := 200 ;
   PutPixel (NonY, NonY, GetMaxColor) ;

   SetTextStyle (SansSerifFont, 0, 3) ;
   SetColor (4) ;

   SetTextJustify (LeftText, TopText) ;
   OutTextXY (NonX, NonY, 'left-top') ;

   SetTextJustify (CenterText, CenterText) ;
   OutTextXY (NonX, NonY, 'center-center') ;

   SetTextJustify (RightText, BottomText) ;
   OutTextXY (NonX, NonY, 'right-bottom') ;

   NonX := 300 ;
   NonY := 300 ;

   SetTextStyle (SansSerifFont, 0, 3) ;
   SetColor (2) ;
   PutPixel (NonY, NonY, GetMaxColor) ;

   SetTextJustify (RightText, TopText) ;
   OutTextXY (NonX, NonY, 'right-top') ;

   SetTextJustify (RightText, CenterText) ;
   OutTextXY (NonX, NonY, 'right-center') ;

   SetTextJustify (RightText, BottomText) ;
   OutTextXY (NonX, NonY, 'right-bottom') ;

   Itxaron := ReadKey ;

   CloseGraph ;
END.
```

7.1.1.6 Lerro zuzenak

Pixelak marrazten diren bezala, Turbo Pascal lengoiak lerro zuzenak marraz ditzake. Horretarako `Graph` unitatean bi prozedura aurredifiniturik daude `Line()` eta `LineTo()` izenak dituztenak (`GrafikoakLantzen8` deritzon programan lehenengoa erakusten da, eta bigarrena `GrafikoakLantzen9` deitu dugun programan).

`Line()` eta `LineTo()` prozeduraz lerroak marrazten dira baina lehendik lerroaren itxura zehaz daiteke `SetLineStyle()` prozeduraren bitartez, hauek lirateke `SetLineStyle()` azpirrutinak behar dituen parametroak:

SetLineStyle (Estilo, Eredu, Lodiera : Word)				
Estilo		**Eredu**	**Lodiera**	
Identif.	**Balioa**	**Balioa**	**Identif.**	**Balioa**
SolidLn	0		NormWidth	1
DottedLn	1		ThickWidth	3
CenterLn	2			
DashedLn	3			
UserBitLn	4			

`SetLineStyle()` prozedurak lerroen estiloa finkatzen du, behar dituen hiru parametroetatik lehenengoa estiloa litzateke (bere balioa `4` denean programadorearen bit molde berezia erabiliko dela adierazteko da). Bigarren parametroa aurreko `UserBitLn` estiloarekin loturik dago eta ez da kontutan izaten baldin eta estiloa `0` eta `3` bitartean aurkitzen bada. Hirugarren parametroak lerroaren lodiera zehazten du.

`MoveTo()` testu-pantailan bezala sistema grafikoan kurtsorearen erakuslerik bada, gertatzen dena grafikoekin lan egitean kurtsorea ikustezina dela. Prozedura honen bitartez erakuslea toki jakin batean kokatzen da.

`GetX`, `GetY` erakuslea non aurkitzen den ezagutzeko funtzioak.

`Line()` prozedurak lerroa marrazteko mutur biren koordenatuak behar ditu (osoak diren lau zenbaki). Prozedura honek ez du erakuslea tokiz aldatzen, parametroen ordena: hasierako x, hasierako y, amaierako x, amaierako y.

`LineTo()` prozedurak lerroa marrazteko, erakuslea non aurkitzen den hasierako muturtzat joz pasatzen zaizkion bi zenbaki osoen bitartez adierazten punturaino lerroa marrazten du. Prozedura honek erakuslearen posizioa berrizten du lerroaren azken muturrean kokatuz.

Ondoren ematen diren hiru programak aipatutako azpirrutinen erabilpen adibideak lirateke. `GrafikoakLantzen7` eta `GrafikoakLantzen8` programetan ikurrina marrazten da `Line()`, `PutPixel()` eta `SetLineStyle()` prozeduretan oinarriturik. `GrafikoakLantzen9` izena duen programa `LineTo()`, `MoveTo()`, `GetX` eta `GetY` azpirrutinen erabilpena ikasteko asmatu da.

Hona hemen `GrafikoakLantzen8` programa:

```
PROGRAM GrafikoakLantzen8 ;                          { \TP70\07\GRAF_8.PAS }
USES
   Crt, Graph ;
VAR
   Kontrolagailua, Modua, HasiX, HasiY, AmaiX, AmaiY, i, j : Integer ;
   Itxaron : Char ;
```

```
BEGIN
   Kontrolagailua := VGA ;
   Modua := VGAhi ;
   InitGraph (Kontrolagailua, Modua, 'C:\tp70\bgi') ;

   IF GraphResult <> grOk THEN
        Writeln (#7, 'Erroren bat grafikoak irekitzean') ;

   SetBkColor (7) ;

   FOR i:=(GetMaxX DIV 4) TO 3*(GetMaxX DIV 4) DO        {oihal berdea }
      FOR j:=(GetMaxY DIV 4) TO 3*(GetMaxY DIV 4) DO
         PutPixel (i, j, 2) ;

   HasiX := GetMaxX DIV 4 ;                              { diagonal bat }
   HasiY := GetMaxY DIV 4 ;
   AmaiX := 3 * (GetMaxX DIV 4) ;
   AmaiY := 3 * (GetMaxY DIV 4) ;
   SetColor (12) ;
   Line (HasiX, HasiY, AmaiX, AmaiY) ;

   HasiX := 3 * (GetMaxX DIV 4) ;                        { beste diagonala }
   HasiY := GetMaxY DIV 4 ;
   AmaiX := GetMaxX DIV 4 ;
   AmaiY := 3 * (GetMaxY DIV 4) ;
   SetColor (12) ;
   Line (HasiX, HasiY, AmaiX, AmaiY) ;

   HasiX := GetMaxX DIV 2 ;                              { Gurutz zuria }
   HasiY := GetMaxY DIV 4 ;
   AmaiX := GetMaxX DIV 2 ;
   AmaiY := 3 * (GetMaxY DIV 4) ;
   SetLineStyle (SolidLn, 0, ThickWidth) ;
   SetColor (GetMaxColor) ;
   Line (HasiX, HasiY, AmaiX, AmaiY) ;

   HasiX := GetMaxX DIV 4 ;
   HasiY := GetMaxY DIV 2 ;
   AmaiX := 3 * (GetMaxX DIV 4) ;
   AmaiY := GetMaxY DIV 2 ;
   SetLineStyle (SolidLn, 0, ThickWidth) ;
   SetColor (GetMaxColor) ;
   Line (HasiX, HasiY, AmaiX, AmaiY) ;

   Itxaron := ReadKey ;
   CloseGraph ;
END.
```

Eta hau `GrafikoakLantzen9` programa:

```
PROGRAM GrafikoakLantzen9 ;                          { \TP70\07\GRAF_9.PAS }
USES
   Crt, Graph ;
VAR
   Kontrolagailua, Modua, HasiX, HasiY, AmaiX, AmaiY : Integer ;
   Mezu1, Mezu2 : String ;
   Itxaron : Char ;
BEGIN
   Kontrolagailua := VGA ;
   Modua := VGAmed ;
   InitGraph (Kontrolagailua, Modua, 'C:\tp70\bgi') ;

   IF GraphResult <> grOk THEN
        Writeln (#7, 'Erroren bat grafikoak irekitzean') ;

   SetBkColor (7) ;
```

```
   HasiX := 100 ;
   HasiY := 100 ;

   MoveTo (HasiX, HasiY) ;                          { Koordenatuak }
   Str (GetX, Mezu1) ;
   Str (GetY, Mezu2) ;
   OutTextXY (200, 10, 'Lehen koordenatuak: (' + Mezu1 + ',' + Mezu2 + ')') ;

   MoveTo (195, 15) ;                               { Gezi zuria }
   SetLineStyle (DashedLn, 0, NormWidth) ;
   LineTo (100, 55) ;
   LineTo (100, 97) ;
   Line (100, 97, 96, 90) ;
   Line (100, 97, 104, 90) ;

   HasiX := 100 ;                                   { Gezi urdina }
   HasiY := 100 ;
   MoveTo (HasiX, HasiY) ;
   SetColor (1) ;
   SetLineStyle (SolidLn, 0, ThickWidth) ;
   LineTo (150, 150) ;
   LineTo (125, 150) ;
   LineTo (125, 220) ;
   LineTo (75, 220) ;
   LineTo (75, 150) ;
   LineTo (50, 150) ;
   LineTo (HasiX, HasiY) ;

   HasiX := 100 ;                                   { 3. dimentsioa }
   HasiY := 100 ;
   MoveTo (HasiX, HasiY) ;
   SetColor (8) ;
   SetLineStyle (SolidLn, 0, NormWidth) ;
   LineTo (130, 90) ;
   LineTo (180, 140) ;
   LineTo (150, 150) ;

   HasiX := 125 ;
   HasiY := 220 ;
   MoveTo (HasiX, HasiY) ;
   LineTo (155, 210) ;
   LineTo (155, 149) ;

   Itxaron := ReadKey ;
   CloseGraph ;
END.
```

`GrafikoakLantzen9` programan sistema grafikoa ireki ondoren lau eginkizun bete egiten dira:

1. Kurtsore grafikoaren koordenatuak lortu eta mezu baten bitartez pantailaratu, hasieran `(100,100)` puntuan kokatzen da kurtsorea `MoveTo()` prozeduraz, gero `GetX` eta `GetY` funtzioen bitartez koordenatu horiek berreskuratu eta pantailan bistarazteko.
2. Aurreko mezuarekin loturik doan gezi zuria marrazten da bigarren urratsean, kolore lehenetsia zuria denez ez da zertan `SetColor()` prozedurari deitu behar. Baina lerroaren estiloa ez denez betea izango `SetLineStyle()` prozeduraren deia egiten da.
3. Gezi urdinaren marrazketa `LineTo()` bitartez egiten da.
4. Hirugarren dimentsioa gauzatzeko `LineTo()` bitartez egiten da ere, zeinek kurtsorea dagoen tokitik parametroek adierazten duten tokiraino lerroa marraztu ondoren kurtsorea azken muturrean kokatzen duen.

7.1.1.7 Oinarrizko azpirrutina grafiko estandarrak

Hona hemen oinarrizko programa grafikoek erabiltzen dituzten azpirrutina grafikoen zerrenda laburtua:

Azpirrutina	Mota	Deskribapen laburra
`InitGraph()`	Proze.	Makinan instalaturik dagoen hardwarearen arabera sistema grafikoa hasieratzen du
`CloseGraph`	Proze.	Sistema grafikoa ixten du
`GraphResult`	Funtz.	Azken eragiketa grafikoari dagokion errore-kodea itzultzen du
`GraphErrorMsg()`	Funtz.	Errore-kode baten mezu testuala itzultzen du
`OutTextXY()`	Proze.	String bat irteerako periferikora igortzen du
`GetModeRange()`	Proze.	Adaptadore grafikoaren kontrolagailuari dagozkion modurik apalena eta garaiena itzultzen ditu
`GetDriverName`	Funtz.	Adaptadore grafikoaren kontrolagailuari dagokion izena string bezala itzultzen du
`GetMaxX`	Funtz.	Indarrean aurkitzen diren kontrolagailu eta moduari dagokien eskuineko zutabearen zenbakia
`GetMaxY`	Funtz.	Indarrean aurkitzen diren kontrolagailu eta moduari dagokien beheko lerroaren zenbakia
`RestoreCrtMode`	Proze.	Sistema grafikoa ireki baino lehenago egon den pantaila mota berreskuratzen du
`SetGraphMode()`	Proze.	Pantaila garbitu eta sistema grafikoan ezartzen du
`PutPixel()`	Proze.	Pantailaren puntu batean kolore jakin bateko pixela marrazten du
`GetMaxColor`	Funtz.	Indarrean aurkitzen diren kontrolagailu eta moduari dagokien kolore garaienaren zenbakia itzultzen du
`SetColor()`	Proze.	Marrazkiaren kolorea ezartzen du
`SetBkColor()`	Proze.	Hondoaren kolorea ezartzen du
`GraphDefaults`	Proze.	Sistema grafikoaren balio lehenetsiak ezartzen ditu
`ClearViewPort`	Proze.	Sistema grafikoaren `ClrScr`
`SetTextStyle()`	Proze.	Marraztuko den letra-tipoa, norabidea eta tamaina zehazten ditu
`SetTextJustify()`	Proze.	Hondoaren kolorea ezartzen du
`SetLineStyle()`	Proze.	Marraztuko den lerroaren itxura zehazten ditu
`MoveTo()`	Proze.	Pantailaren toki batera kurtsorea eraman
`GetX`	Funtz.	Kurtsore grafikoaren X posizioa itzultzen du
`GetY`	Funtz.	Kurtsore grafikoaren Y posizioa itzultzen du
`Line()`	Proze.	Emandako bi punturen arteko lerroa marrazten du
`LineTo()`	Proze.	Kurtsorearen posizioa eta emandako puntu baten arteko lerroa marrazten du, kurtsorea higitzen du

Horietaz sakontzeko Turbo Pascal Lengoaiak eskaintzen duen laguntzaz baliatzea gomendatzen da.

7.1.2 Geure azpirrutina grafikoak

Zazpigarren kapitulu honen sarreran esan den bezala, gure helburua `Grafiko` izena duen unitatea sortzea da, horretarako **7.1.1.7 Oinarrizko azpirrutina grafiko estandarrak** puntuan bildu den taularen funtzio eta prozedurak erabiltzea posible izango zaigu baina ez Turbo Pascal lengoaiak definiturik izan dezakeen besterik.

7.1.2.1 Elementu geometrikoak banaka

Demagun gure zereginetarako poligonoak (laukiak, karratuak, lakizuzenak eta hirukiak) behar ditugula, demagun halaber forma borobilak behar ditugula zirkunferentziak, arkuak eta elipseak alegia. Jarraian datozen puntuetan elementu bakoitzari loturiko programa idatziko dugu eta ondoren guztiak biltzen dituen `Grafiko` unitatea lortuko dugu.

7.1.2.1.1 Hirukia

`Hiruki()` deituko dugun prozedurak hirukiaren hiru erpinak (sei zenbaki oso) hartuko ditu sarrerako parametro bezala, eta pantaila grafikoan `Line()` bitartez triangelua marraztuko du.

Jarraian ematen den `HirukiaMarrazten` programan `Hiruki()` prozedura garaturik ikus daiteke:

```
PROGRAM HirukiaMarrazten ;                    { \TP70\07\GRF_HIRU.PAS }
USES
   Crt, Graph ;

PROCEDURE Hiruki (Erpin1X, Erpin1Y,
                  Erpin2X, Erpin2Y,
                  Erpin3X, Erpin3Y : Integer) ;
BEGIN
   Line (Erpin1X, Erpin1Y, Erpin2X, Erpin2Y) ;
   Line (Erpin2X, Erpin2Y, Erpin3X, Erpin3Y) ;
   Line (Erpin3X, Erpin3Y, Erpin1X, Erpin1Y) ;
END ;

VAR
   Kontrolagailua, Modua : Integer ;
   Erpin1X, Erpin1Y, Erpin2X, Erpin2Y, Erpin3X, Erpin3Y : Integer ;
   Itxaron : Char ;
BEGIN
   Write ('1. erpinaren X eman: ') ;
   ReadLn (Erpin1X) ;
   Write ('1. erpinaren Y eman: ') ;
   ReadLn (Erpin1Y) ;

   Write ('2. erpinaren X eman: ') ;
   ReadLn (Erpin2X) ;
   Write ('2. erpinaren Y eman: ') ;
   ReadLn (Erpin2Y) ;

   Write ('3. erpinaren X eman: ') ;
   ReadLn (Erpin3X) ;
   Write ('3. erpinaren Y eman: ') ;
   ReadLn (Erpin3Y) ;

   Kontrolagailua := Detect ;
   InitGraph (Kontrolagailua, Modua, 'C:\tp70\bgi') ;
   IF GraphResult <> grOk THEN
        Writeln (#7, 'Erroren bat grafikoak irekitzean') ;

   Hiruki (Erpin1X, Erpin1Y, Erpin2X, Erpin2Y, Erpin3X, Erpin3Y) ;

   Itxaron := ReadKey ;
   CloseGraph ;
END.
```

7.1.2.1.2 Laukia

`Lauki()` deituko dugun prozedurak aurrekoak egiten duena gauza berbera egingo du baina hiru erpin izan beharrean lau izango dira (osoak diren zortzi zenbaki). Beraz, ez dugu `LaukiaMarrazten` programa idatziko baina bai bertan dagoen `Lauki()` prozedurari dagokion goiburukoa:

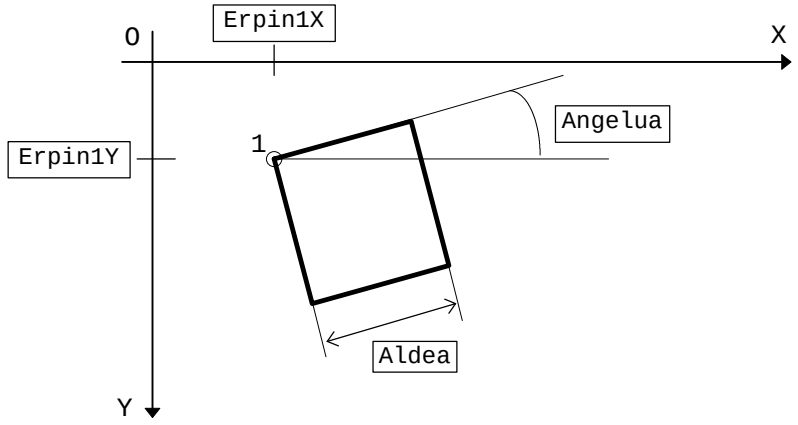
```
PROCEDURE Lauki (Erpin1X, Erpin1Y, Erpin2X, Erpin2Y,
                 Erpin3X, Erpin3Y, Erpin4X, Erpin4Y : Integer) ;
```

Hirukiaren prozeduraz baliatuko den erabiltzaileak erpinen koordenatuak jakitea normaltzat jo genezake. Laukiaren kasuan egoera bestelakoa da, lauki erregularra bada eta bere orientazioa `OX` eta `OY` ardatzena izatean erpinak aldez aurretik ezagutzea normala da ere, baina laukia erregularra izanik bere orientazioa ardatzena ez bada erpinak kalkulatu[5] beharko lirateke eta ondorioz `Lauki()` prozedura deserosoa suerta daiteke.

Horregatik hurrengo bi puntuetan edozein orientazioko karratuak eta lauki zuzenak marrazteko aproposagoak izango diren prozedurak garatuko ditugu.

7.1.2.1.3 Karratua

Esan bezala, demagun karratu baten erpina ezagutzen dugula (`OY` ardatzarekiko erpinik hurbilena ezagutzen da), suposa dezagun karratuaren aldea datua dela ere eta karratua biraturik badago angelua ezaguna da (angelua gradutan ematen da). Ikus irudia:



Honezkero, `Karratu()` deituko dugun prozedurari legokion goiburukoa hau izango litzateke:

```
PROCEDURE Karratu (Erpin1X, Erpin1Y, Aldea, Angelua : Integer) ;
```

Datuak hartu ondoren `Karratu()` prozedurak beste hiru erpinak kalkulatuko ditu, gero `Line()` edo `LineTo()` azpirrutina estandarren bitartez eskaturiko karratua marraztu ahal izateko. Kontutan izan erpinak kakulatzean, osoak ez diren koordenatuak atera daitezkeela eta pantaila grafikoan balio frakzionatuen zati osoarekin lan egingo delako erroreak[6] gerta daitezkeela.

[5] Jeneralean laukia definitzen dituzten datuak bere neurriak, erpin baten posizioa eta `OX` ardatzarekiko angelua izaten dira.

[6] Erroreek maila pixel batekoa izango denez bereizmen handiko pantailetan ez da igarriko.

Hona hemen `Karratu()` prozedura darabilen `KarratuaMarrazten` programa:

```
PROGRAM KarratuaMarrazten ;                    { \TP70\07\GRF_LAUK.PAS }
USES
   Crt, Graph ;

PROCEDURE Karratu (Erpin1X, Erpin1Y, Aldea, Angelua : Integer) ;
VAR
   DesplazX, DesplazY : Integer ;
   Radianak : Real ;
   Erpin2X, Erpin2Y, Erpin3X, Erpin3Y, Erpin4X, Erpin4Y : Integer ;
BEGIN
   Radianak := PI * Angelua / 180 ;
   DesplazX := Trunc (Aldea * cos (Radianak) ) ;
   DesplazY := Trunc (Aldea * sin (Radianak) ) ;

   Erpin2X :=  Erpin1X + DesplazX ;
   Erpin2Y :=  Erpin1Y - DesplazY ;
   Erpin3X :=  Erpin1X + DesplazX + DesplazY ;
   Erpin3Y :=  Erpin1Y + DesplazX - DesplazY ;
   Erpin4X :=  Erpin1X + DesplazY ;
   Erpin4Y :=  Erpin1Y + DesplazX ;

   MoveTo (Erpin1X, Erpin1Y) ;
   LineTo (Erpin2X, Erpin2Y) ;
   LineTo (Erpin3X, Erpin3Y) ;
   LineTo (Erpin4X, Erpin4Y) ;
   LineTo (Erpin1X, Erpin1Y) ;
END ;

VAR
   Kontrolagailua, Modua : Integer ;
   Erpin1X, Erpin1Y, Aldea, Angelua : Integer ;
   Itxaron : Char ;
BEGIN
   Write ('1. erpinaren X eman: ') ;
   ReadLn (Erpin1X) ;
   Write ('1. erpinaren Y eman: ') ;
   ReadLn (Erpin1Y) ;

   Write ('1-2 aldearen eta horizontalaren arteko angelua: ') ;
   ReadLn (Angelua) ;

   Write ('Karratuaren aldea eman: ') ;
   ReadLn (Aldea) ;

   Kontrolagailua := Detect ;
   InitGraph (Kontrolagailua, Modua, 'C:\tp70\bgi') ;
   IF GraphResult <> grOk THEN
        Writeln (#7, 'Erroren bat grafikoak irekitzean') ;

   Karratu (Erpin1X, Erpin1Y, Aldea, Angelua) ;

   Itxaron := ReadKey ;
   CloseGraph ;
END.
```

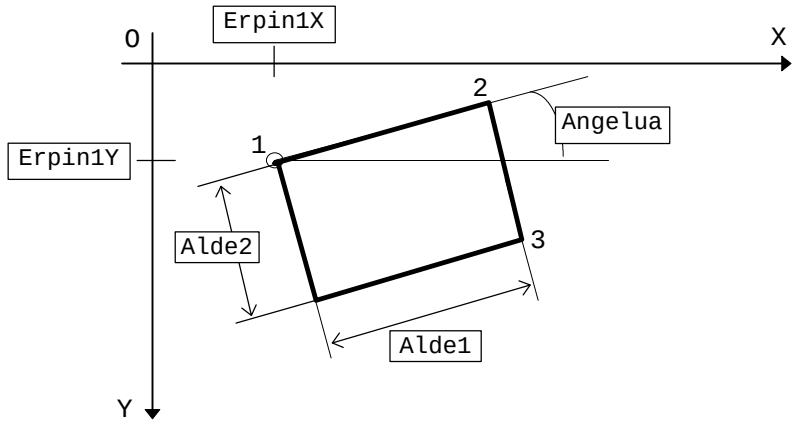
Ikusten denez `KarratuaMarrazten` adibidean datuak teklatuz irakurri eta `Karratu()` prozedurari pasatzen zaizkio. Honen barruan, ezer baino lehen angelua radianetara bihurtzen da eta ondoren `DesplazX` eta `DesplazY` aldagai laguntzailetan datua den `1` erpinarekiko desplazamenduak gordetzen dira, hots, kalkulatu behar diren beste hiru erpinen koordenatuak pantailako `(Erpin1X, Erpin1Y)` puntuarekiko desplazamenduak biltegitzen dira. Amaitzeko, `MoveTo()` bitartez hasierako erpinean kokatu eta `LineTo()` azpirrutinan oinarriturik lau lerroak marrazten dira.

7.1.2.1.4 Laukizuzena

`Karratu()` prozedura ikusi eta gero ez zaigu zail egingo `Laukizuzen()` prozedura ulertzea. Izan ere, duen ezberdintasun bakarra irudiaren neurrira zehazteko bigarren parametro bat eman behar zaiola da. `Laukizuzen()` prozedurak honelako goiburukoa izango du:

```
PROCEDURE Laukizuzen (Erpin1X, Erpin1Y, Alde1, Angelua, Alde2 : Integer) ;
```

Non, lehen bezala, `Erpin1X` eta `Erpin1Y` parametroek `0Y` ardatzarekiko erpinik hurbilenaren koordenatuak diren, eta `Alde1` balioak `1` eta `2` erpinen arteko distantzia den. Ikus irudia:



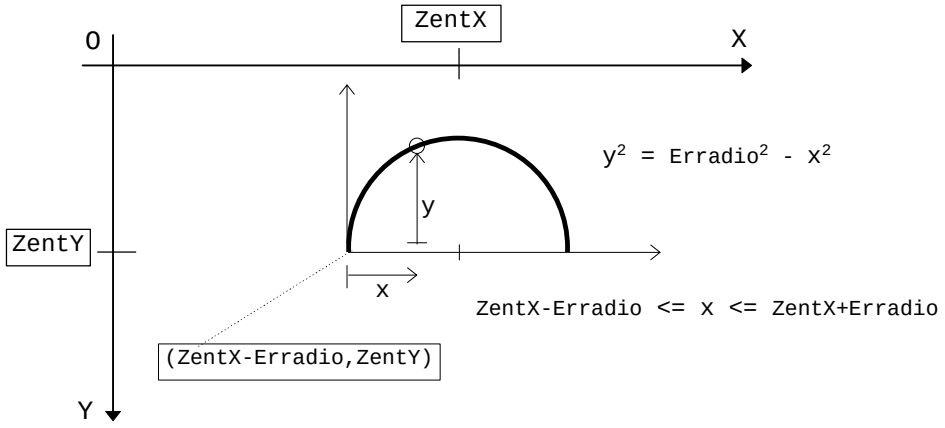
Hona hemen `LaukizuzenaMarrazten` programan aurkitzen den `Laukizuzen()` prozedurari dagozkion sententziak:

```
PROCEDURE Laukizuzen (Erpin1X, Erpin1Y, Alde1, Angelua, Alde2 : Integer) ;
VAR
   Desplaz1X, Desplaz1Y, Desplaz2X, Desplaz2Y : Integer ;
   Radianak : Real ;
   Erpin2X, Erpin2Y, Erpin3X, Erpin3Y, Erpin4X, Erpin4Y : Integer ;
BEGIN
   Radianak := PI * Angelua / 180 ;
   Desplaz1X := Trunc (Alde1 * cos (Radianak) ) ;
   Desplaz1Y := Trunc (Alde1 * sin (Radianak) ) ;
   Desplaz2X := Trunc (Alde2 * sin (Radianak) ) ;
   Desplaz2Y := Trunc (Alde2 * cos (Radianak) ) ;

   Erpin2X :=  Erpin1X + Desplaz1X ;
   Erpin2Y :=  Erpin1Y - Desplaz1Y ;
   Erpin3X :=  Erpin1X + Desplaz1X + Desplaz2X ;
   Erpin3Y :=  Erpin1Y + Desplaz2Y - Desplaz1Y ;
   Erpin4X :=  Erpin1X + Desplaz2X ;
   Erpin4Y :=  Erpin1Y + Desplaz2Y ;

   MoveTo (Erpin1X, Erpin1Y) ;
   LineTo (Erpin2X, Erpin2Y) ;
   LineTo (Erpin3X, Erpin3Y) ;
   LineTo (Erpin4X, Erpin4Y) ;
   LineTo (Erpin1X, Erpin1Y) ;
END ;
```

7.1.2.1.5 Zirkunferentzia

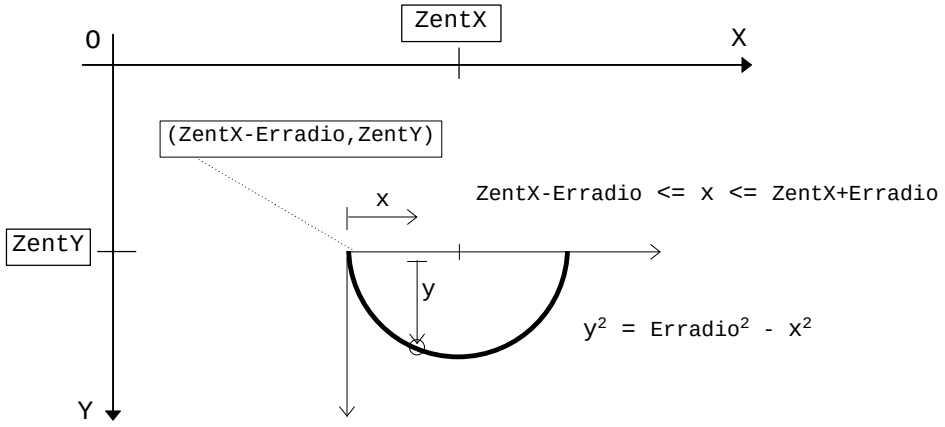
Zirkunferentziak marraztea oraintsu azaldu diren triangeluak eta laukiak baino zailagoa denez hiru urratsetan esplikatuko dugu.

Hasteko `Zirk1Marrazten` programaren bitartez koloredun bi semizirkunferentzia eta horien ardatz nagusiak marrazten ikasiko dugu. Hara, demagun zirkunferentzia zehazteko ezagutzen diren datuak zentrua eta erradioa direla (hiru zenbaki oso), `Zirk1Marrazten` programan aurkitzen den `Zirkunferentzi1()` prozedurak zentrua eta erradioa hartuko ditu `FOR-DO` egitura batean ondoko irudiak aurkezten den kalkulua burutzeko:



Non, `x` aldagaiak `ZentX-Erradio` puntutik `ZentX+Erradio` punturaino balioak hartuko dituen eta `y` altura bertikala $x^2 + y^2 = Erradio^2$ formulaz ateratzen den, `FOR-DO` bigiztaren iterazio bakoitzeko zirkunferentziaren puntua den `(x,y)` koordenatuak kalkulatu ditugunez, `LineTo(ZentX+x,ZentY-y)` aplika daiteke iterazioari dagokion lerro txikitxoa lortuz (kontutan izan bigizta aurretik `MoveTo(ZentX-Erradio,ZentY)` prozeduraren bitartez kurtsorea goiko semizirkunferentziaren ezkerreko bazterrean kokatu beharra dagoela).

Behin goiko semizirkunferentzia lortu ondoren behekoa falta zaigu, eta horretarako `FOR-DO` prozesu berriro errepikatzen da `Zirkunferentzi1()` prozeduran. Beheko semizirkunferentzia lortzeko `MoveTo(ZentX-Erradio,ZentY)` hasieraketa egiten da eta iterazio bakoitzeko `y` kalkulatzen da eta lerro txikitxoa marrazten da (oraingoan `LineTo(ZentX+x,ZentY+y)` eginez behekoa baita).



Hona hemen `Zirk1Marrazten` programa osoa:

```
PROGRAM Zirk1Marrazten ;                        { \TP70\07\GRF_ZIR1.PAS }
USES
   Crt, Graph ;

PROCEDURE Zirkunferentzi1 (ZentX, ZentY, Errad : Integer) ;
VAR
   Kont, HasiX, HasiY : Integer ;
   x, y, Erradio : LongInt ;
BEGIN
   SetColor (GetMaxColor) ;
   Erradio := Errad ;           { Integer sarrera LongInt bihurtzeko }

   Line (ZentX-Erradio, ZentY, ZentX+Erradio, ZentY) ;
   Line (ZentX, ZentY-Erradio, ZentX, ZentY+Erradio) ;

   SetColor (GetMaxColor-1) ;
   OutTextXY (ZentX, ZentY-Erradio-15, 'Goiko zirkunferentzi erdia') ;
   MoveTo (ZentX-Erradio, ZentY) ;

   FOR Kont:=-Erradio TO Erradio DO
   BEGIN
      x := Kont ;
      y := Trunc (Sqrt (Erradio*Erradio - x*x) ) ;
      LineTo (ZentX+x, ZentY-y) ;
   END ;

   SetColor (GetMaxColor-2) ;
   OutTextXY (ZentX, ZentY+Erradio+10, 'Beheko zirkunferentzi erdia') ;
   MoveTo (ZentX-Erradio, ZentY) ;
   FOR Kont:=-Erradio TO Erradio DO
   BEGIN
      x := Kont ;
      y := Trunc (Sqrt (Erradio*Erradio - x*x) ) ;
      LineTo (ZentX+x, ZentY+y) ;
   END ;
END ;

VAR
   Kontrolagailua, Modua : Integer ;
   ZentruX, ZentruY, Erradio : Integer ;
   Itxaron : Char ;
BEGIN
   Write ('Zentruaren X eman: ') ;
   ReadLn (ZentruX) ;
   Write ('Zentruaren Y eman: ') ;
   ReadLn (ZentruY) ;
   REPEAT
      Write ('Zirkunferentziaren erradioa eman: ') ;
      ReadLn (Erradio) ;
   UNTIL Erradio > 0 ;

   Kontrolagailua := Detect ;
   InitGraph (Kontrolagailua, Modua, 'C:\tp70\bgi') ;
   IF GraphResult <> grOk THEN
        Writeln (#7, 'Erroren bat grafikoak irekitzean') ;

   Zirkunferentzi1 (ZentruX, ZentruY, Erradio) ;

   Itxaron := ReadKey ;
   CloseGraph ;
END.
```

`Zirk1Marrazten` programa exekutatzean semizirkunferentzi biak eta ardatz nagusiak kolore ezberdinetan agertuko zaizkigu, honelako zerbait ateratzen da zentrutzat eta erradiotzat `(110,80)` eta `45` emanik:

Zirkunferentzien marrazketaren bigarren urratsean planteamendu berbera mantentzen da baina lehenengoan egin diren kalkulu bikoiztuak ekiditen dira; hau da, `FOR-DO` egitura bakar batean iterazio bakoitzeko `x` eta `y` zirkunferentziaren koordenatuak lortzen dira eta lerrotxo simetriko bi marraztuko dira goiko eta beheko semizirkunferentziarenak hain zuzen ere. Bigarren urratseko `Zirk2Marrazten` programan `Zirkunferentzi2()` prozedura aldatu beharra dago honelako zerbait idatziz, non `LineTo()` ordez `Line()` prozeduran oinarritzen garen:

```
PROCEDURE Zirkunferentzi2 (ZentX, ZentY, Errad : Integer) ;
VAR
   Kont, HasiX, HasiY : Integer ;
   x, y, AurrekoX, AurrekoY1, AurrekoY2, Erradio : LongInt ;
BEGIN
   SetColor (GetMaxColor) ;
   Erradio := Errad ;           { Integer sarrera LongInt bihurtzeko }

   Line (ZentX-Erradio, ZentY, ZentX+Erradio, ZentY) ;
   Line (ZentX, ZentY-Erradio, ZentX, ZentY+Erradio) ;

   SetColor (GetMaxColor-1) ;
   OutTextXY (ZentX, ZentY-Erradio-15, 'Goiko zirkunferentzi erdia') ;

   MoveTo (ZentX-Erradio, ZentY) ;
   AurrekoX := -Erradio ;
   AurrekoY1 := 0 ;
   AurrekoY2 := 0 ;
   FOR Kont:=-Erradio TO Erradio DO
   BEGIN
      x := Kont ;
      y := Trunc (Sqrt (Erradio*Erradio - x*x) ) ;
      SetColor (GetMaxColor-1) ;
      Line (ZentX+AurrekoX, ZentY+AurrekoY1, ZentX+x, ZentY-y) ;
      SetColor (GetMaxColor-2) ;
      Line (ZentX+AurrekoX, ZentY+AurrekoY2, ZentX+x, ZentY+y) ;
      AurrekoX := x ;
      AurrekoY1 := -y ;
      AurrekoY2 := y ;
   END ;
   OutTextXY (ZentX, ZentY+Erradio+10, 'Beheko zirkunferentzi erdia') ;
END ;
```


Amaitzeko, hirugarren programa bat garatu dugu `ZirkunfMarrazten` deitu duguna, eta berak barneratzen duen `Zirkunferentzi()` izeneko prozedurak zentrua eta erradioa harturik zirkunferentzi zuri bat marrazten du inolako ardatz, mezu edo kolore lagungarririk gabe. Hona hemen `Zirkunferentzi()` prozedurari dagokion goiburukoa:

```
PROCEDURE Zirkunferentzi (ZentX, ZentY, Errad : Integer) ;
```

7.1.2.1.6 Elipsea

Elipsea marrazteko **7.1.2.1.5 Zirkunferentzia** puntuan esandakoak balio digu baldin eta `y` kalkulatzeko elipseari dagokion formula erabiltzen badugu, sarrerako datuak kasu honetan zentruaren koordenatuak eta elipsearen ardatz nagusi biren neurriak izango dira. Bi programa prestatu ditugu:

1. `Elip1Marrazten` delakoan koloreak ardatzak eta mezuak agertzen dira, bere barnean dagoen `Elipse1()` prozedurak `y` kalkulatu ondoren `Line()` pare baten bitatez goiko eta beheko semielipse biren lerrotxoak marrazten ditu iterazio bakoitzeko.
2. `ElipseaMarrazten` programan berriz, koloreak eta mezuak alde batera utzirik elipse zuria pantailaratzen da. `Elipse()` prozedurak `y` kalkulatzeko darabilen formula honako hau da:

$$\frac{x^2}{\left(ArdatzHoriz/2\right)^2} + \frac{y^2}{\left(ArdatzBerti/2\right)^2} = 1$$

Hona hemen `Elipse()` prozedura:

```
PROCEDURE Elipse (ZentX, ZentY, ArdatzHoriz, ArdatzBerti : Integer) ;
VAR
   Kont, HasiX, HasiY : Integer ;
   x, y, AurrekoX, AurrekoY1, AurrekoY2, ErdiHoriz, ErdiBerti : LongInt ;
BEGIN
   SetColor (GetMaxColor) ;

   ErdiHoriz := ArdatzHoriz DIV 2 ;   { Integer sarrera LongInt bihurtzeko }
   ErdiBerti := ArdatzBerti DIV 2 ;

   MoveTo (ZentX-ErdiHoriz, ZentY) ;
   AurrekoX := -ErdiHoriz ;
   AurrekoY1 := 0 ;
   AurrekoY2 := 0 ;
   FOR Kont:=-ErdiHoriz TO ErdiHoriz DO
   BEGIN
      x := Kont ;
      y := Trunc (ErdiBerti * Sqrt ( (ErdiHoriz*ErdiHoriz - x*x) /
                                     (ErdiHoriz*ErdiHoriz)) ) ;
      Line (ZentX+AurrekoX, ZentY+AurrekoY1, ZentX+x, ZentY-y) ;
      Line (ZentX+AurrekoX, ZentY+AurrekoY2, ZentX+x, ZentY+y) ;

      AurrekoX := x ;
      AurrekoY1 := -y ;
      AurrekoY2 := y ;
   END ;
END ;
```

7.1.2.1.7 Arkua

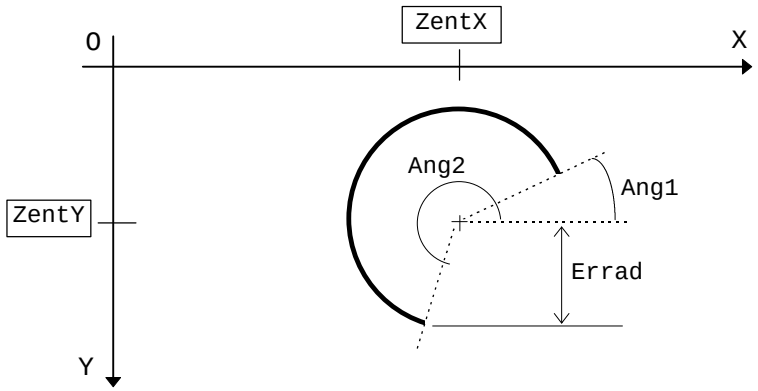
Demagun zirkunferentzien arkuak marraztu nahi direla ere, arku bat definitzeko bere kokamena eta neurriaz gain, zirkunferentziaren erradioa eta zentruaz gain, bere hasierako eta amaierako puntuak zehaztu behar dira. Arku bat pantailaratzen duen programa lortzeko zazpi bertsio eberdin idatzi dugu, hona hemen bakoitzaren deskribapen laburra:

- `Arku0Marrazten` eta `Arku1Marrazten` deitu ditugun programak sinpleenak dira, datutzat arkuaren erradioa eta 90 gradu baino txikiagoa den angelu bat onartzen dituzte.
- `Arku2Marrazten` programan, alde batetik zentrua eta erradioa eman daitezke, baina aurrekoen bezala arkuaren hasiera eskuineko puntua denez amaierako angelu bat onartzen du soilik (gehienez 180 gradutakoa angelua hain zuzen).
- `Arku3Marrazten` programak aurrekoak burutzen duena bikoiztu egiten du eta 360 graduen mugaraino edozein angelu onartzen du.
- `Arku4Marrazten` programa `Arku3Marrazten` bezalakoa da baina ez du ardatz kolore eta mezurik pantailaratzen.
- `Arku5Marrazten` deituriko programa garatuena da, sarreratzeat zirkunferentziaren zentrua eta erradioarekin batera arkuaren hasierako eta amaierako angeluak behar ditu. Angelu bi horiei, horrela balegokie, `Trukaketa()` prozeduraren bitartez moldaketa bat egiten zaie eta ondoren `Arku5()` prozeduran arkua marrazten da. Jarraian `Arku5Marrazten` programa osoa erakusten da.
- `ArkuaMarrazten` azken programak `Arku5Marrazten` delako programarekiko duen diferentzia ardatz, kolore eta mezurik pantailaratzen ez dituela da.

`Arku5Marrazten` programan oinarriturik arkuak marrazten duen programa idatzi dugu zein `ArkuaMarrazten` deitu dugun. `ArkuaMarrazten` programaren `Arkua()` prozedurak laister batean erakutsiko dugun `Arku5()` azpiprogramak behar dituen datu berdinak hartu ondoren arku zuri bat marrazten du pantailan ardatzik eta mezurik gabe. Beraz, kapitulu honetan helburutzat dugun `Grafiko` unitatean arkuak marraztuko dituen azpirrutinaren goiburukoa honako hau izango da:

```
PROCEDURE Arku (ZentX, ZentY, Errad, Ang1, Ang2 : Integer) ;
```

Non lehenengo hiru parametroek arkuaren kokamena eta tamaina adierazten duten eta azken biek arku zirkularraren hasierako eta amaierako puntuak zehazteko angeluak diren.



`ArkuaMarrazten` programa ulertzeko `Arku5Marrazten` programaren sententziak ikus ditzagun:

```
PROGRAM Arku5Marrazten ;                        { \TP70\07\GRF_ARK5.PAS }
USES
   Crt, Graph ;

PROCEDURE Trukaketa (VAR Ang1, Ang2 : Integer) ;
VAR
   Laguntzaile : Integer ;
BEGIN
   Laguntzaile := Ang1 ;
   Ang1 := Ang2 ;
   Ang2 := Laguntzaile ;
END ;

PROCEDURE Arku5 (ZentX, ZentY, Errad, Ang1, Ang2 : Integer) ;
VAR
   Kont, HasiX, HasiY, AmaiX, AmaiY : Integer ;
   x, y, Erradio : LongInt ;
   Mezu1, Mezu2 : String ;
BEGIN
     IF Ang1 > Ang2 THEN
         Trukaketa (Ang1, Ang2) ;

     Str (Ang1, Mezu1) ;
     Str (Ang2, Mezu2) ;

     SetColor (GetMaxColor) ;

     Erradio := Errad ;           { Integer sarrera LongInt bihurtzeko }

     Line (ZentX-Erradio, ZentY, ZentX+Erradio, ZentY) ;
     Line (ZentX, ZentY-Erradio, ZentX, ZentY+Erradio) ;

     HasiX :=   Trunc (Erradio * Cos (Ang1*PI/180) ) ;
     HasiY := - Trunc (Erradio * Sin (Ang1*PI/180) ) ;

     AmaiX :=   Trunc (Erradio * Cos (Ang2*PI/180) ) ;
     AmaiY := - Trunc (Erradio * Sin (Ang2*PI/180) ) ;

     SetColor (GetMaxColor-1) ;
     Line (ZentX, ZentY, ZentX+AmaiX, ZentY+AmaiY) ;
     MoveTo (ZentX, ZentY) ;
     LineTo (ZentX + HasiX, ZentY + HasiY) ;
     MoveTo (ZentX + AmaiX, ZentY + AmaiY) ;

     IF Ang2 <= 180 THEN
     BEGIN
        OutTextXY (ZentX, ZentY-Erradio-15, Mezu1 + ' ---> ' + Mezu2) ;
        MoveTo (ZentX + AmaiX, ZentY + AmaiY) ;
        SetColor (GetMaxColor-2) ;
        FOR Kont:=AmaiX TO HasiX DO
        BEGIN
           x := Kont ;
           y := Trunc (Sqrt (Erradio*Erradio - x*x) ) ;
           LineTo (ZentX + x, ZentY - y) ;
        END ;
        LineTo (ZentX + HasiX, ZentY + HasiY) ;
     END ;

     IF (Ang1 < 180) AND (Ang2 > 180) THEN
     BEGIN
        OutTextXY (ZentX, ZentY-Erradio-15, Mezu1 + ' ---> ' + Mezu2) ;
        MoveTo (ZentX + AmaiX, ZentY + AmaiY) ;
        SetColor (GetMaxColor-3) ;
        FOR Kont:=-AmaiX TO Erradio DO
        BEGIN
           x := Kont ;
           y := Trunc (Sqrt (Erradio*Erradio - x*x) ) ;
           LineTo (ZentX - x, ZentY + y) ;
        END ;
```

```
        FOR Kont:=Erradio DOWNTO -HasiX DO
        BEGIN
           x := Kont ;
           y := Trunc (Sqrt (Erradio*Erradio - x*x) ) ;
           LineTo (ZentX - x, ZentY - y) ;
        END ;
        LineTo (ZentX + HasiX, ZentY + HasiY) ;
     END ;

     IF Ang1 >= 180 THEN
     BEGIN
        OutTextXY (ZentX, ZentY+Erradio+15, Mezu1 + ' ---> ' + Mezu2) ;
        MoveTo (ZentX + HasiX, ZentY + HasiY) ;
        SetColor (GetMaxColor-4) ;
        FOR Kont:=HasiX TO AmaiX DO
        BEGIN
           x := Kont ;
           y := Trunc (Sqrt (Erradio*Erradio - x*x) ) ;
           LineTo (ZentX + x, ZentY + y) ;
        END ;
        LineTo (ZentX + AmaiX, ZentY + AmaiY) ;
     END ;
END ;

VAR
   Kontrolagailua, Modua : Integer ;
   ZentruX, ZentruY, Erradio, Angelu1, Angelu2 : Integer ;
   Itxaron : Char ;
BEGIN
   Write ('Zentruaren X eman: ') ;
   ReadLn (ZentruX) ;
   Write ('Zentruaren Y eman: ') ;
   ReadLn (ZentruY) ;

   REPEAT
      Write ('Arkuaren erradioa eman: ') ;
      ReadLn (Erradio) ;
   UNTIL Erradio > 0 ;

   REPEAT
      Write ('Horizontalarekiko hasierako angelua eman (0..360): ') ;
      ReadLn (Angelu1) ;
   UNTIL (0 <= Angelu1) AND (Angelu1 <= 360) ;

   REPEAT
      Write ('Horizontalarekiko amaierako angelua eman (0..360): ') ;
      ReadLn (Angelu2) ;
   UNTIL (0 <= Angelu2) AND (Angelu2 <= 360) AND (Angelu2 <> Angelu1) ;

   Kontrolagailua := Detect ;
   InitGraph (Kontrolagailua, Modua, 'C:\tp70\bgi') ;
   IF GraphResult <> grOk THEN
        Writeln (#7, 'Erroren bat grafikoak irekitzean') ;

   Arku5 (ZentruX, ZentruY, Erradio, Angelu1, Angelu2) ;

   Itxaron := ReadKey ;
   CloseGraph ;
END.
```

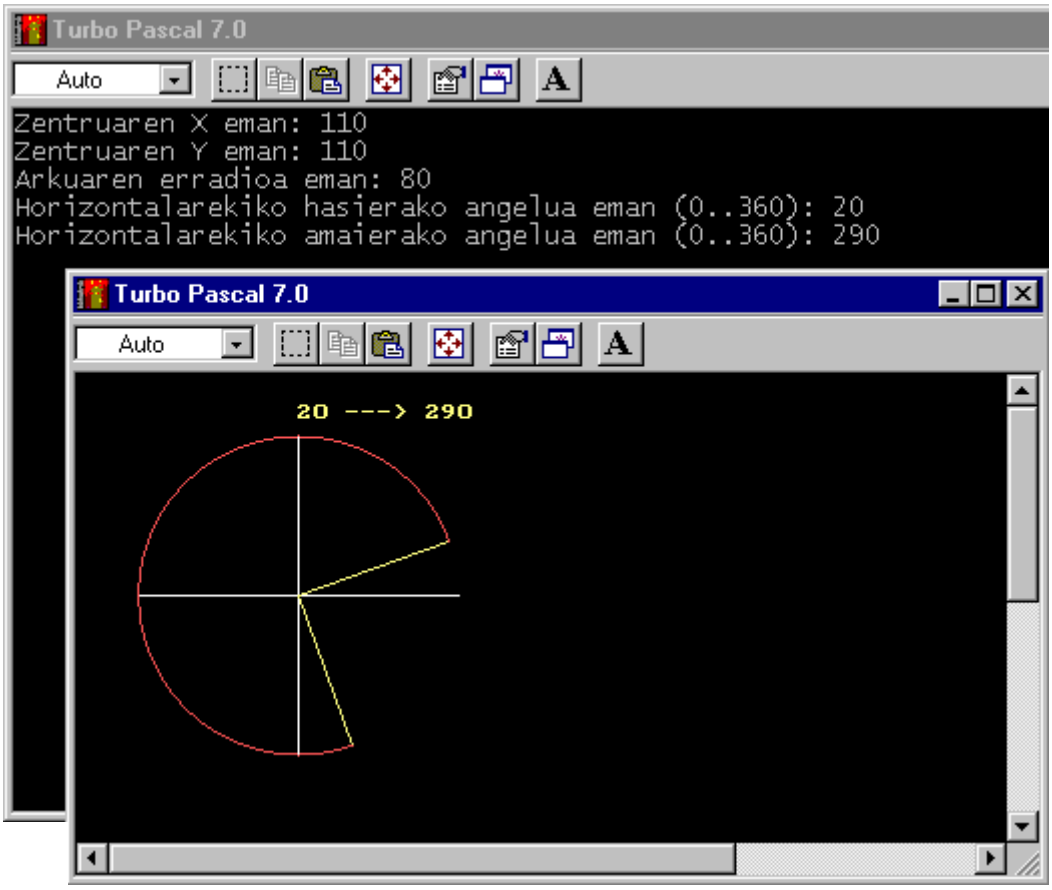
Programa nagusian arkua marrazteko datuak hartzen dira eta `ZentruX`, `ZentruY`, `Erradio`, `Angelu1` eta `Angelu2` aldagaietan biltegitzen dira. Ondoren sistema grafikoa ireki eta `Arku5()` prozeduraren deia egiten da, arkua ikusteko geldiune bat tartekatu eta programa amaitu aurretik sistema grafikoa itxi.

Azter dezagun `Arku5()` prozedura. Ikusten denez baliozko parametroak erabili dira programa nagusitik hartzen dituen datuak sarrerakoak baitira (parametro horien izenak

prozeduran hauek dira: `ZentX`, `ZentY`, `Errad`, `Ang1` eta `Ang2`). Arkuaren hasierako angelua `Ang1` dela suposatuz `Ang2` baino txikiagoa izango da, baina erabiltzaileak eman dituen balioak suposizio honekin bat ez datozean `Trukaketa(Ang1,Ang2)` deiari esker angelu biren balioak elkar trukatzen dira. Ardatz nagusiak marraztu ondoren, arkuaren hasiera eta amaiera finkatzen duten `(HasiX,HasiY)` eta `(AmaiX,AmaiY)` puntuen koordenatuak kalkulatzen dira, hortik aurrera hiru kasu bereizten dira:

1. `Ang2` angeluak gehienez 180 gradu balio duenean (halakoetan `Ang1` angelua 0 eta 179 bitartean aurkituko da), arkua `FOR-DO` bakar batez marraz daiteke.
2. `Ang1` angelua 180 gradutara iristen ez denean eta `Ang2`-k berriz 180 baino handiagoa denean dagokien arkua bi zatitan (bi `FOR-DO` bitartez) marrazten da.
3. `Ang1` angeluak 180 graduak gainditzen dituenean (`Ang2` angeluak 180 baino gehiago balioko du), dagokien arkua `FOR-DO` bakar batez marraz daiteke.

Hona hemen datu zehatz batzuen sarrera pantaila eta horiekin `Arku5()` prozedurak pantailaratzen duena, ikusten denez arkuarekin batera ardatzak eta sarrerako angeluen balioak agertzen dira:



7.1.2.1.8 Funtzio trigonometrikoak

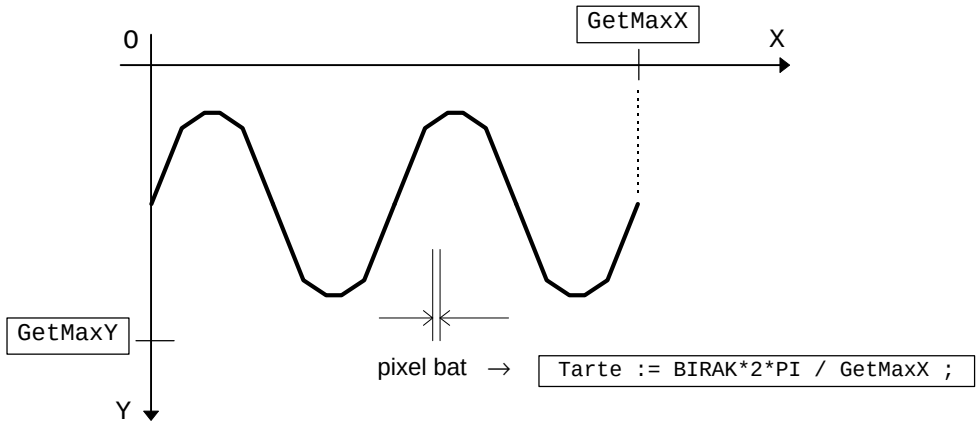
Nahiz eta `Grafiko` unitateak ez dituen funtzio trigonometrikoak barneratuko, sinu funtzio nola marraztuko litzatekeen erakustea interesgarria iruditu zaigu eta horregatik puntu honen zergatia. Demagun `Sin()` funtzio estandarraren emaitza puntuz puntu kalkulatu eta pantailaratu nahi dugula uhina marraztuz.

Beharko genituzkeen datu biak uhinaren anplitudea eta frekuentzia lirateke, gure sistema grafikoaren bereizmena ezaguna denez `x` bakoitzeko dagokion `y` lortu eta pixeleka edo lerroka marraztea litzateke lan guztia:

$$y = Anplitudea\ Sin(w\ x)$$

$$w = 2p\ Frekuentzia = \frac{2p}{T}$$

Prestatu dugun `Sinu1Marrazten` programan uhinaren frekuentzia konstantea da (`2.0` bira ematen dira pantaila osoan), txartel grafikoak onartzen duen bereizmena `640x480` izanik, horrek esan nahi du bi birak emateko ardatz horizontalaren koordenatuak `0`-tik `639` bitartera aldatu behar dela. Beraz, pixel bakoitzari dagokion tartea hau da:



Pixel bati dagokion tartea kalkulatu ondoren, pantaila osoa horizontalki hartzen duen `FOR-DO` egitura batean `x` bakoitzeko sinua lortzeko `Sin()` funtzio estandarra aplikatzea aski da. Ikus `Sinu2Marrazten` programaren sententziak zeinean birak `2.0` konstantea izan beharrean (`Sinu1Marrazten` programaren kasua) erabiltzaileak teklatuz aukeratzen duen balioren bat den:

```
PROGRAM Sinu2Marrazten ;                        { \TP70\07\GRF_SIN2.PAS }
USES
   Crt, Graph ;
VAR
   Kontrolagailua, Modua : Integer ;
   x, y, Kont, ErdiaY : Integer ;
   Altuera : Integer ;
   Birak : Real ;
   Ang, Tarte : Real ;
   Mezu1, Mezu2 : String ;
   Itxaron : Char ;
BEGIN
   REPEAT
     Write ('Altuera eman (10..200): ') ;
     ReadLn (Altuera) ;
   UNTIL (Altuera >= 10) AND (Altuera <= 200) ;
   REPEAT
     Write ('Birak pantailan ( > 0.0): ') ;
     ReadLn (Birak) ;
   UNTIL (Birak > 0) ;

   Kontrolagailua := Detect ;
   InitGraph (Kontrolagailua, Modua, 'C:\tp70\bgi') ;
   IF GraphResult <> grOk THEN
        Writeln (#7, 'Erroren bat grafikoak irekitzean') ;
```



```
   ErdiaY := GetMaxY DIV 2 ;

   Line (0, ErdiaY, GetMaxX, ErdiaY) ;
   Line (0, 0, 0, GetMaxY) ;
   SetColor (GetMaxColor - 1) ;
   MoveTo (0, ErdiaY) ;

   Ang := 0.0 ;
   Tarte := 2*PI*Birak / GetMaxX ;
   FOR Kont:=1 TO GetMaxX DO
   BEGIN
     x := Trunc (Kont) ;
     y := ErdiaY - Trunc (Altuera * sin(Ang)) ;
     LineTo (x, y) ;
     Ang := Ang + Tarte ;
   END;
                                          { y = Anplitudea * sin (omega*x) }
   Str (Altuera, Mezu1) ;                 { omega = 2*PI*f = 2*PI/T }
   Str (2*PI*Birak/GetMaxX:0:3, Mezu2) ;
   Mezu1 := 'y = ' + Mezu1 +' sin (' + Mezu2 +' x)' ;
   OutTextXY (GetMaxX DIV 2, ErdiaY-Altuera-10, Mezu1) ;

   Itxaron := ReadKey ;
   CloseGraph ;
END.
```

Anplitudea eta frekuentzia (zenbat bira pantailan) eman ondoren pixel bakoitzari dagokion tartea kalkulatzen da, eta horrekin `FOR-DO` egitura barruan `x` aldatzen joango gara eta dagokion `y` eskuratzen da (kontutan izan `Sinu2Marrazten` programan derrigorrezkoak ez diren aldagai laguntzaileak agertzen direla, adibidez `Kont` eta `Ang`).

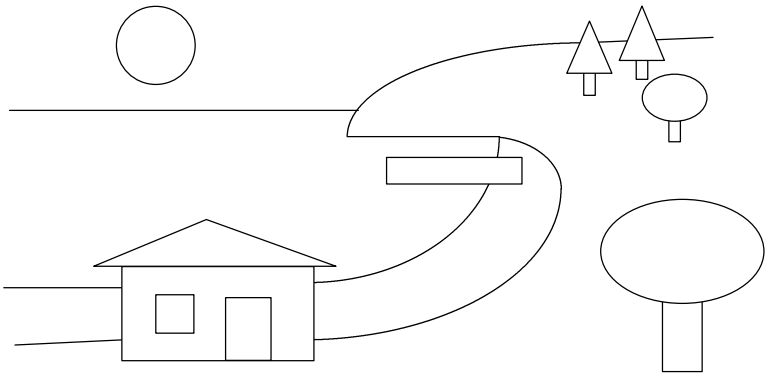
7.1.2.2 Elementu geometrikoak bildurik

7.1.2.1.1 eta **7.1.2.1.8** puntuen artean garatu diren azpirrutinek `Graph` izeneko unitatea behar dute, eta esan dugun bezala `Grafiko` deituko dugun unitate batean bilduko ditugu. Baina zergatik denak batera bildu? Programaren batek hirukia, karratua edo beste irudi bat beharrez gero dagokion azpirrutina bertan txertatzea aski baita.

Ez, ez da bide zuzena elementu geometrikoak erabiliko dituzten bezero-programa guztietan berriro kopiatzea. Arrazoiak hiru dira:

1. Lehen arrazoia unitateekin lortzen den programen mailaketa eta antolaketa oso onuragarria dela izango litzateke. Programa handi bat egitean oso espezifikoak diren zereginak agertzen dira (adibidez elementu geometrikoak marraztea) eta horiek toki batean bildurik izateak eta unitate bezala erabiltzeak programaren ulergarritasuna dakar, bezero-programa askoz txikiagoa baita
2. Bigarren arrazoiak kodearen hobetzearekin zerikusia du, izan ere zeregin bereziak unitateetan bildurik daudenean toki bakar batean aurkitzen dira, eta ondorioz bezero-programa batean ezin dira aldatu (horrek funtzio beraren bertsio ezberdinak programetan sakabanaturik edukitzea ekiditen du). Baina bestalde, inoiz unitate barnean aurkitzen den azpirrutina bat hobetzeko aukera izanez gero bere eragina bezero-programa guztietara automatikoki zabaltzen da (banan-banan aldaketa horiek bakoitzari egin beharrean)
3. Hirugarren, eta azken, arrazoia iturburu-programa eta programa konpilatuaren arteko jokuan datza. Unitate jakin bat erabiltzeko bezero-programak ez du zertan unitatearen jatorrizko iturburu-programa eduki behar, unitate konpilatua baizik. Gehienetan norberarenak ez diren unitateak konpilaturik heltzen zaizkigu

Demagun jarraian agertzen den irudia lortuko dugun bezero-programa idatz nahi dugula, zeinek `Grafiko` deituriko gure unitatea darabilen:



Bezero-programaren izena `HondartzaMarrazten` izanik honelako hasiera izango du:

```
PROGRAM HondartzaMarrazten ;
USES
   Grafiko ;

   { Bezero-programaren funtzio eta prozedurak }

VAR
   { Bezero-programaren aldagaiak }

BEGIN     { Bezero-programaren programa nagusia }

   { Besteak beste, deituko dituen prozeduren zerrenda:       }
   {       Hiruki(), Lauki(), LaukiZuzen(), Zirkunferentzi()  }
   {       Elipse(), Arku(), Line(), LineTo(), MoveTo(), ...  }

END.
```

Ikusten denez `Grafiko` unitatea darabilen bezero-programa batek horren erabilpena programaren hasieran deklaratu behar du; ondorioz hirukiak, karratuak, laukizuzenak eta gainerako elementuak marrazten dituzten azpirrutinak eskura ditzake. Kontura gaitezkeenez `Line()`, `LineTo()` eta `MoveTo()` prozedurak deitzen dira ere, baina ez da dagokien `Graph` unitatea hasieran deklaratu, horrek esan nahi du `Graph` unitatearen erazagupena `Grafiko` unitatearen barnean egiten dela.

Aurrerago azalduko den **7.3 UNITATEEN ADIBIDEA: GRAFIKOAK** izenburuko puntuan `Grafiko` delako unitatea sortuko dugu, baina lehenago edozein unitatearen barne egitura ikas dezagun.

7.2 UNITATE BAT ERAIKITZEN

Unitate bat sortzeko zenbait gauza kontutan izatea beharrezkoa da. Hurrengo puntuetan ondoko hiru hauek azaltzen dira:

1. Unitate baten barne egitura
2. Unitate baten sorrera, konpilazioa eta gaurkotzea
3. Uste gabeko gertaerak

7.2.1 Unitate baten barne egitura

Unitate bat bi zatitan banaturik dago: bat `Interface` delakoa (ikusgai eta plazaraturik dagoen zatia), eta bi `Implementation` deiturikoa (zati ezkutua edo pribatua). Interfazea eta inplementazioaz gain, unitate batek bere erazagupen eta hasieraketa atal propioak izan ditzake.

Hona hemen Turbo Pascal-aren unitate batek duen egitura:

```
UNIT UnitateIzena ;                                  { Unitatearen izenburua }

INTERFACE                                          { Unitatearen zati publikoa }
   { Hemen jarritakoa esportagarria da.    }
   { USES, CONST, TYPE, VAR, PROCEDURE eta }
   { FUNCTION erazagupenak egin daitezke   }

IMPLEMENTATION                                     { Unitatearen zati pribatua }
   { Interfazean plazaturiko prozedura eta }
   { funtzioei dagokien kodea hemen idatzi }
   { egiten da. Unitateak berak behar izan }
   { ezkero USES, CONST, TYPE, PROCEDURE,  }
   { VAR eta FUNCTION erazagupenak berriro }
   { errepika daitezke baina horiek bezero-}
   { programatik eskuraezinak izango dira  }

BEGIN                                               { Unitatearen hasieraketa }
   { Zati hau aukeraka da, behin exekutatzen }
   { da, eta unitatearen elemntuek behar izan }
   { dezaketeen prestaketak egiteko balio du  }
END.
```

`Izenburua` Unitatearen izenburua derrigorrezkoa da, eta unitate konpilatuari dagokion objektu-fitxategia sortzean kontutan izango den izena bertan aukeratzen da. Unitatearen zati hau, programa baten izenburua bezalakoa da baina honetan `UNIT` hitz erreserbatua agertzen da.

`Interfazea` Kanpora begira dagoen zatia da. Beste programa batzuek, bezero-programek, erabiliko dituzten datu-egiturak, konstanteak eta aldagaiak deklaratzen dira bertan; berdin kanpotik dei daitezkeen prozedura eta funtzioen goiburukoak (prozedura eta funtzio hauen aginduak inplementazioaren zatian aurkituko dira). Zati honetan unitateak eskaintzen dituen ezaugarriak plazaratzen dira eta `INTERFACE` hitzez markaturik dator.

`Inplementazioa` Unitatearen zati hau erabat pribatua da. Interfazean plazaratu diren prozedurak eta funtzioak hemen garaturik idazten dira. Baina askotan, gainerako beste prozedura eta funtzioak idazten dira ere, azken hauek ezin izango dira unitatearen kanpotik deitu. Derrigorrezkoa den zati hau `IMPLEMENTATION` eta `END` hitz erreserbatuez mugatzen da, eta `END` baino lehenago `BEGIN` hitz erreserbatua agertzen baldin bada, euren artekoa hasieraketa atala izango da.

`Hasieraketa` Zati hau ez da derrigorrezkoa. Unitatearen aldagai, array edo erakusleen hasieraketak egiteko balio du. Ezer baino lehenago, eta bakarrik behin batean, exekutatzen denez Turbo Pascal programa nagusi baten antza du.

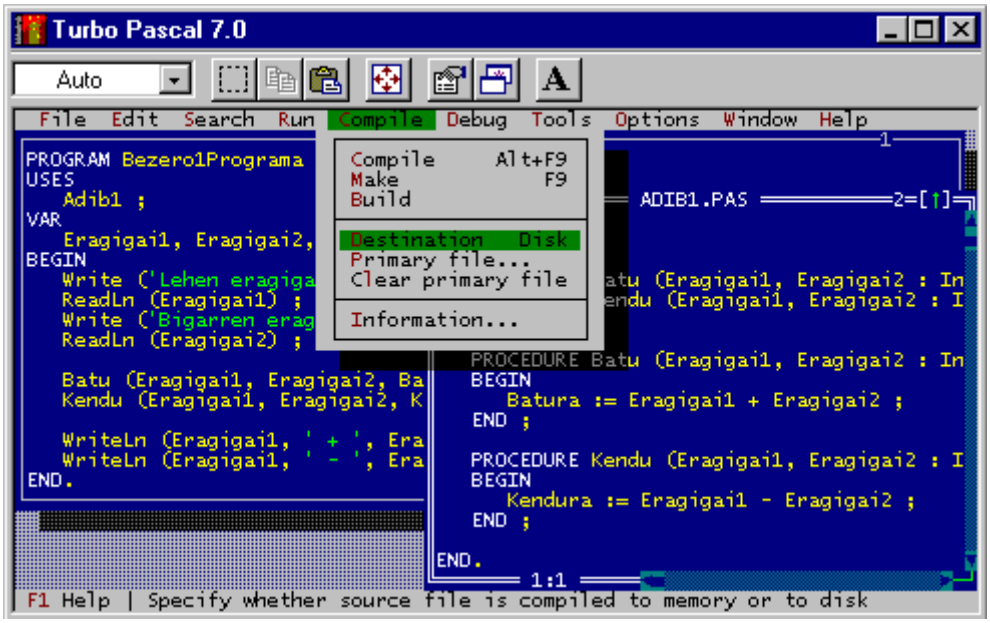
7.2.2 Unitate baten sorrera, konpilazioa eta gaurkotzea

Gainerako programak bezala unitate bat sortzeko testu-editore baten bidez egingo da. Unitatea editatu ondoren fitxategi batean gordetzen da, fitxategiaren izena unitatearen izenburuan agertzen dena izango da, luzapena berriz Turbo Pascal iturburu-programak bezala `.PAS` izango da.

Unitate baten konpilazio prozesua ohizko programa baten konpilaketa bezalakoa da; hau da, menu baten bitartez gidatzen den Turbo Pascal 7.0 bertsioan menuaren bosgarren aukera `Compile` da zeinek edizio-leihoan dagoen iturburu-programa itzultzeko balio duen. `Compile` komandoaren menpean azpiaukerak daude, esate baterako `Compile | Destination` delakoaren bitartez konpilaketaren emaitza non kokatuko den erabakitzen da (programa konpilatua ordenadorearen memorian ala diskoan gordeko den erabakitzen da[7]).

Esan den bezala, unitate baten konpilazio prozesua ohizko programena bezalakoa da, baina lorturiko fitxategi konpilatua berez ez da exekutagarria. Izan ere, ez ahaztu unitate bat lagungarri eta osagarri den software zati bat dela eta beste programa batek erabiliko duela. Ondorioz unitate konpilatu bat diskoan gordetzean Turbo Pascal inguruneak ez dio `.EXE` luzapena jartzen `.TPU` luzapena baizik (Turbo Pascal Unit).

Turbo Pascal 7.0 bertsioan `Compile | Make` eta `Compile | Build` azpiaukerak ditu iturburu-programa fitxategi anitzetan banaturik dagoenean kudeatzeko, aukera horiek aproposak dira ere unitateak independienteki sortu eta konpilatu egiten direnean. Hona hemen azpiaukero horien funtzionamendua:



`Compile | Primary file...` izeneko azpiaukerak fitxategi nagusi bat definitzeko beta ematen digu, (fitxategi nagusia baligabetzeko `Compile | Clear primary file` dago). Turbo Pascal lengoaiaren ingurune integratuan dagoen edizio-leihoan aurkitzen den iturburu-programa birkonpilatzeko `Compile | Make` azpiaukeraren bitartez egin daiteke. Fitxategi nagusirik definitu bada `Compile | Make` azpiaukera fitxategi nagusiaren konpilaketarekin hasten da, eta fitxategi nagusiaren menpeko diren gaurkotu[8] gabeko unitate guztiak konpilatuko ditu ere. `Compile` komandoaren `Compile | Build` azpiaukeraren bitartez unitate guztiak birkonpilatzen dira eguneratuta egon ala egon ez.

---

[7] `Destination Disk` hauta daiteke ala bestela `Destination Memory`.

[8] Unitate baten (`.PAS`) iturburu-programak duen data, unitateari dagokion (`.TPU`) objektu-programak duen data baino zaharragoa denean, unitatea gaurkoratua dagoela kontsideratzen da.

Turbo Pascal lengoaiak eskaintzen duen ingurune integratuaz baliaturik jarraian erakusten diren bi programak idatziko dira, leiho ezberdinetan. Adibidez aplikazio-programa `1` leihoan eta unitatea `2` leihoan idatzi ondoren `Compile | Destination Disk` aukeratu eta konpilazioari ekingo diogu, konpilatu ondoren lau fitxategi hauek edukiko ditugu diskoan:

1. `Bezero1Programa`-ri dagokion iturburu-fitxategia (`BEZERO1.PAS`)
2. Izen bereko unitatearen `ADIB1.PAS` iturburu-fitxategia
3. `Bezero1Programa`-ri dagokion `BEZERO1.EXE` fitxategi exekutagarria
4. `ADIB1.TPU` unitate konpilatua

Hauek lirateke `Bezero1Programa` eta `Adib1` unitatearen iturburu-programak:

```
PROGRAM Bezero1Programa ;                              { \TP70\07\BEZERO1.PAS }
USES
   Adib1 ;
VAR
   Eragigai1, Eragigai2, Batura, Kendura : Integer ;
BEGIN
   Write ('Lehen eragigaia eman: ') ;
   ReadLn (Eragigai1) ;
   Write ('Bigarren eragigaia eman: ') ;
   ReadLn (Eragigai2) ;

   Batu (Eragigai1, Eragigai2, Batura) ;      { Adib1 unitatean garaturik }
   Kendu (Eragigai1, Eragigai2, Kendura) ;    { Adib1 unitatean garaturik }

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

Ikusten denez `Batu` eta `Kendu` prozeduren deiak egin ahal izateko, lehendik `Adib1` unitatea eraezagutu behar da. `Adib1` unitateak bere interfazean `Batu` eta `Kendu` prozedurak jarrita ditu, beraz aplikazio-programek erabil ditzaten esportagarriak izango dira:

```
UNIT Adib1 ;                                           { \TP70\07\ADIB1.PAS }

INTERFACE
   PROCEDURE Batu (Eragigai1, Eragigai2 : Integer; VAR Batura : Integer) ;
   PROCEDURE Kendu (Eragigai1, Eragigai2 : Integer; VAR Kendura : Integer) ;

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

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

END.
```

Hona hemen `Bezero1Programa`-ren exekuzio bat:

```
Lehen eragigaia eman: 5
Bigarren eragigaia eman: 8
5 + 8 = 13
5 + 8 = -3
_
```

7.2.3 Uste gabeko gertaerak

Unitateen erabiltzea dela eta, erroreak suerta daitezke. Horregatik, hona hemen **7.2.3.1**, **7.2.3.2** eta **7.2.3.3** puntuetan ustegabeko hiru gertaeren deskribapena.

7.2.3.1 Unitate kabiatuak

Behar izanez gero, inplementazio atal pribatuan `USES` deklarazio bat jar daiteke. Horrelakoetan, `USES` hitz erreserbatua `IMPLEMENTATION` hitz erreserbatuaren ondo-ondoan idatziko da, gauza bera egiten da interfaze zatian (`INTERFACE` eta `USES` jarraian doazela alegia). Gogoratu ohizko programetan ere `USES` deklarazioa egitekotan `PROGRAM izena;` goiburukoaren ostean jartzen dela derrigorrez.

Inplementazioan ezarritako `USES` batek, unitatearen barne zehaztasunak gehiago ezkutatzen ditu. Izan ere, aplikazio-programa batek `Maila1` unitatea erabiltzen badu bere interfazea ezagutuko du nahiz eta `Maila1` -ren inplementazioaren (barne zehaztasunen) berri eduki ez; baina `Maila1` unitatearen bezero-programa horrek ez du `Maila2` unitateari buruzko ezer ezagutzen (zeharka erabiltzen duenez bere existentziaren berri dauka baina interfazea ezezaguna zaio):

```
UNIT Maila1 ;
INTERFACE
       USES Maila2 ;
       ...
IMPLEMENTATION
       ...
END.
```

```
UNIT Maila2 ;

       ...


END.
```

`Maila1` unitatea erabili nahi duen edozein aplikazio-programak (edo beste unitateren batek) derrigorrez `Maila2` aipatu behar du, honelaxe:

```
USES Maila1, Maila2 ;  { Aplikazio-programaren erazagupena }
```

Azter dezagun jarraian ematen diren bi `Maila1` eta `Maila2` unitateak eta horietan oinarritzen den `Bezero2Programa` izeneko aplikazio-programa. Iturburu-programei begirada bat emanez `Bezero2Programa` lehentxoago ikusi dugun `Bezero1Programa`-ren berdintsua da, gertatzen dena `Adib1` unitateak zituen bi prozedurak `Maila1` eta `Maila2` unitateen artean banatu direla. `Bezero2Programa` idaztean ezaguna dena `Maila1` unitatearen interfazea baldin bada, honako hau ezagutzen da:

```
INTERFACE
   USES Maila2 ;
   PROCEDURE Batu (Eragigai1, Eragigai2 : Integer; VAR Batura : Integer) ;
```

Ondorioz, `Bezero2Programa`-k publikoa den `Batu()` prozedura bati dei egin diezaioke (horretarako bi zenbaki oso sarreratzat eman eta irteera beste zenbaki oso bat izango da); horrez gain `Maila2` unitateak barneratzen duena `Bezero2Programa`-k eskura dezake `USES` bitartez deklaraturik dagoelako baina, tamalez, `Bezero2Programa` idazten duenak ez daki `Maila2` unitatearen interfazea zein den, horretarako `Maila1` ezezik `Maila2` unitatearen iturburu-programa aztertu beharra baitago.

`Maila1` unitatearen bitartez `Maila2` unitatearen zati publikoan dagoena eskura daiteke, baina `Maila1` unitatearen interfazeak ez digu `Maila2` interfazearen berri ematen.

Hauek lirateke `Bezero2Programa`, `Maila1` eta `Maila2`-ren iturburu-programak:

```
PROGRAM Bezero2Programa ;                              { \TP70\07\BEZERO2.PAS }
USES
   Maila1, Maila2 ;
VAR
   Eragigai1, Eragigai2, Batura, Kendura : Integer ;
BEGIN
   Write ('Lehen eragigaia eman: ') ;
   ReadLn (Eragigai1) ;
   Write ('Bigarren eragigaia eman: ') ;
   ReadLn (Eragigai2) ;

   Batu (Eragigai1, Eragigai2, Batura) ;     { Maila1 unitatean garaturik }
   Kendu (Eragigai1, Eragigai2, Kendura) ;   { Maila2 unitatean garaturik }

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

```
UNIT Maila1 ;                                          { \TP70\07\MAILA1.PAS }

INTERFACE
   USES Maila2 ;
   PROCEDURE Batu (Eragigai1, Eragigai2 : Integer; VAR Batura : Integer) ;

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

END.
```

```
UNIT Maila2 ;                                          { \TP70\07\MAILA2.PAS }

INTERFACE
   PROCEDURE Kendu (Eragigai1, Eragigai2 : Integer; VAR Kendura : Integer) ;

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

END.
```

7.2.3.2 Unitateen arteko erreferentzia gurutzatuak

`Adib1` izeneko unitateak `Adib2` unitatea badarabil, eta azken honek aurreko `Adib1` unitatearen deia egiten badu. Dagozkien deklarazioan ezin dira interfazetan egin, konpilazio denboran errorea[9] suertatzen baita:

Hau onartezina da:

```
UNIT Adib1 ;
INTERFACE
      USES Adib2 ;
...
```

```
UNIT Adib2 ;
INTERFACE
      USES Adib1 ;
...
```

[9] Mezuak honela dio → `Error 68: circular unit reference.`

Baina Turbo Pascal-ek unitateen erreferentzia gurutzatuak onartzen ditu, baldin eta inplementazio zati pribatuetan `USES` bana jartzen bazaie:

```
UNIT Adib1 ;
INTERFACE
      ...
IMPLEMENTATION
      USES Adib2 ;
      ...
END.
```

```
UNIT Adib2 ;
INTERFACE
      ...
IMPLEMENTATION
      USES Adib1 ;
      ...
END.
```

7.2.3.3 Unitate ezberdinetan dagoen identifikadore bera

Programa baten sententziak idaztean, kodifikazioaren toki ezberdinetan izen bereko etiketak aurkitzea erraza da. Programa nagusiaren etiketariko identifikadore homonimoa prozedura edo funtzio baten barruan dagoen bitartean arazorik ez da sortzen, azpirrutinaren identifikadoreak bertako elementuak dira eta seigarren kapituluko **6.5.3 Identifikadoreen lehentasuna eta ustegabeko gertaerak** izenburuko puntua gogora ekartzea komeniko litzateke (bertako aldagaien eta aldagai orokorren arteko harremanak argitzen baitu).

Pascal estandarrean eta Turbo Pascal bertsio zaharretan unitaterik ez dago eta ondorioz identifikadore homonimoekin arazorik ez dago, urrezko arau hau aplikatzen baita: maila jakin batean definituko diren identifikadoreak ez dira errepikatuko. Baina Turbo Pascal lengoiaren bosgarren bertsiotik aurrera unitateak existitzen direnez arau horrek ez du ondoko egoera honetan sortzen den arazoa ekiditen:

```
PROGRAM Gatazka;
USES Adib1 ;
VAR x : Integer ;
      ...
```

```
UNIT Adib1 ;
INTERFACE
      VAR x : Integer ;
      ...
```

Irakurleak asmatuko lukeenez gerta daitekeen arazoa `x` aldagaiarekin loturik dago. Izan ere, batetik `x` aldagaia `Gatazka` programaren aldagai orokorra da, eta bestetik `Adib1` unitatearen zati publikoan beste `x` bat deklaraturik dago. Unitatearen azken `x` aldagai hau `Gatazka` bezero-programak orokorra izango balitz bezala erabil dezake.

`Gatazka` programarentzat identifikadore biak orokorrak dira, eta etiketa homonimoen honelako kasuetan azkenak irauten du lehena zapaltzen duelako. Gure adibidean, `USES` agindua (ezinbestean) `Gatazka` programaren hasieran jarrita dago, eta ondorioz `Gatazka` programan aurkitzen den `x` aldagai orokorra bien artean berriena da; hau da, `Adib1` unitateko `x` aldagaia zapaldurik geratuko litzatekeenez ezingo genuke inolaz ere eskuratu.

`Adib1` unitateko `x` aldagaia `Gatazka` programatik atzitu behar izanez gero, aldagai hori puntu kualifikadorez zehaztuko litzateke. Esate baterako, `Adib1` unitateko `x` aldagaiak gordetzen duena `Gatazka` programan pantailaratzeko honela egingo litzateke:

```
WriteLn (Adib1.x) ;
```

kualifikadore

Identifikadore homonimoen arazoa esplikatzeko bezero-programa bat eta berak darabilen unitate bat aztertu ditugu, baina gertaera berbera identifikadore homonimoak dituzten bi unitateekin gerta daiteke. Hemen ere, ebazpidea arazoak eragin ditzaketen aldagaiak puntu kualifikadorez estuki identifikatzean datza:

```
BlokearenEtiketa.AldagaiarenEtiketa
```

7.3 UNITATEEN ADIBIDEA: GRAFIKOAK

Sarrerako **7.1** puntuan aipatutako `Grafiko` izena izango duen unitatea sortzeko unea heldu da. Horretarako hiru urratsetako metodologia zehatz bat jarraituko dugu, kontutan izan unitateekin lan egitean bi jokamolde izan daitezkeela: alde batetik unitatea sortzen duen programadorea, eta, bestetik, tresna bezala unitatea erabiliko duen bezero-programadorea.

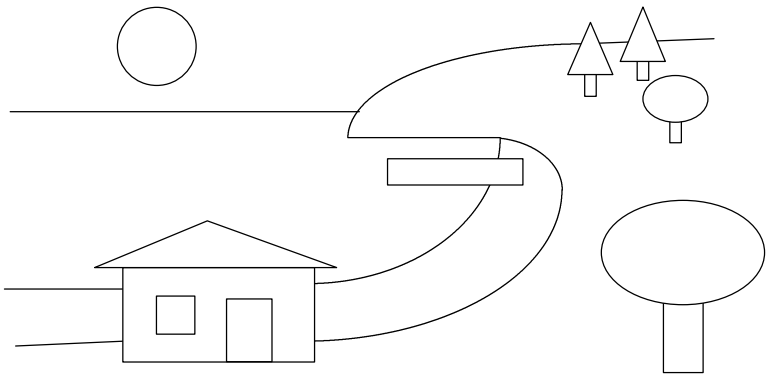
Nahiz eta azkenean aplikazio-programa bat egin, gure interesa, orain, utitatea sortzen duen programadorearen zereginean kokatzen da, esan dugun bezala, hiru urratsetan bana daitekeena:

1. Lehenik, unitateak bete beharko dituen baldintzak definituko dira
2. Aurrekoarekin lotuta unitatearen interfazea lortuko da
3. Azkenean unitatea gorpuzten duen inplementazioa idatziko da

Unitatea ezin daiteke amaitutzat eman zenbait aplikazio-programek erabili eta frogatu arte, beraz, inplementazioa idatzi ondoren sistematikoki testatu egiten da. Urrats bakoitzatik dokumentu bana ondoriotzen da, izan ere, unitatea ez da bakarrik software programa bat izango baizik eta software elementuak eta horiek erabili ahal izateko derrigorrezkoa den dokumentazio teknikoa.

7.3.1 Unitate grafikoaren beharkizunak

Sortuko den `Grafiko` unitateak **7.1.2.2 Elementu geometrikoak bildurik** puntuan arestian erakutsi izan den honelako irudiak balioko du, zeini dagokion aplikazio-programaren izena `HondartzaMarrazten` den:



Ikusten denez `HondartzaMarrazten` aplikazio-programak zirkunferentziak, hirukiak, karratuak, laukizuzenak eta gainerako elementuak marrazten dituzten azpirrutinak behar ditu. Lerroak marrazteko eta pantailan tokiz aldatzeko `Line()`, `LineTo()` eta `MoveTo()` prozedurak deitzeko ahalmena izan behar duenez `Graph` unitate estandarraren erazagupena egingo da ere.

Honezkero, `HondartzaMarrazten` bezero-programak darabiltzan azpirrutinen zerrenda idaz daiteke. Zerrenda horretan agertzen diren azpirrutinek helburua den `Grafiko` unitatearen beharkizunak osatzen dute eta interfazean aurkituko dira.

`Grafiko` unitatearen beharkizunak ondoko taulan biltzen dira:

`Grafiko` unitatearen espezifikazioa

Egitura: Unitate honek ez du datu-mota berezirik erabiltzen. Zenbaki osoak, zenbaki errealak eta karaktere-kateak lantzen dira.

Eragiketak: Hauek dira unitateari loturik dauden operazioak:

- Hiru erpin eman eta triangelua marraztu
- Lau erpin eman ondoren laukia marraztu
- Erpin baten koordenatua, angelua eta aldea emanik karratua marraztu
- Erpin baten koordenatua, angelua eta aldeak emanik laukizuzena marraztu
- Zentruaren kordenatuak eta erradioa eman ondoren zirkunferentzia marraztu
- Zentruaren kordenatuak eta ardatz biak eman ondoren elipsea marraztu
- Zentrua eta hasiera zein amaiera angeluak emanik arkua marraztu
- Pantailan kurtsorea higitu
- Hasierako eta amaierako muturrak emanik lerroa marraztu
- Amaierako muturra eman uneko posiziotik hasita lerroa marraztu

7.3.2 Unitate grafikoaren interfazea

Unitatearen beharkizunak edo espezifikazioa zehaztu ondoren dagokion interfazea erraza da. Izan ere, eragiketak burutzen dituzten azpirrutinen goiburukoek osatuko dute `Grafiko` unitatearen interfazearen zati nagusia.

`Grafiko` unitatea erabiliko duen programadoreak interfazea baino ez du ezagutuko eta eskainiko zaion dokumentazio teknikoa honelako zerbait izango da:

`Grafiko` unitatearen interfazea

```
INTERFACE
   VAR
     Irispide : String ;
     ErrKode : Integer ;
   PROCEDURE Hiruki (Erpin1X, Erpin1Y, Erpin2X, Erpin2Y,
                     Erpin3X, Erpin3Y : Integer) ;
   PROCEDURE Lauki (Erpin1X, Erpin1Y, Erpin2X, Erpin2Y,
                    Erpin3X, Erpin3Y, Erpin3X, Erpin3Y : Integer) ;
   PROCEDURE Karratu (Erpin1X, Erpin1Y, Aldea, Angelua : Integer) ;
   PROCEDURE Laukizuzen (Erpin1X, Erpin1Y, Alde1, Angelua, Alde2 : Integer) ;
   PROCEDURE Zirkunferentzi (ZentX, ZentY, Errad : Integer) ;
   PROCEDURE Elipse (ZentX, ZentY, ArdatzHoriz, ArdatzBerti : Integer) ;
   PROCEDURE Arku (ZentX, ZentY, Errad, Ang1, Ang2 : Integer) ;
   { Line(), LineTo() eta MoveTo()     interfazean ez dira berdefinituko }
```

`Irispide` eta `ErrKode` aldagaiak publikoak dira, eta hasieraketarekin eralazionaturik daudenez inplementazioari dagokion **7.3.3** hurrengo puntuan aipatuko ditugu. Garrantzitsuena honako hau da: aplikazio-programak idazteko `GRAFIKO.TPU` fitxategi konpilatua eta aurreko taula besterik ez dugu behar.

7.3.3 Unitate grafikoaren inplementazioa

Dakigunez, unitatearen atal honek interfazean plazaturiko prozedura eta funtzioak kodeturik biltzen du (ageriko prozedura eta funtzioz gain, unitateak behar dituen azpirrutina pribatuak kodetzen dira ere[10]). `Grafiko` unitatearen inplementazioa luzeegia geratzen delako hemen ez idaztea erabaki dugu, hala ere testu honekin batera iturburu-programak banatzen direnez bertan aurki eta azter daiteke `Grafiko`-ren inplementazio osoa.

`Grafiko` unitateari dagokion inplementazioaren hasieraketa azalduko dugu jarraian. Aplikazio-programak grafikoak izango dira eta `Grafiko`-ren azpirrutinak VGA txartelerako eta 2 modurako (640x480 pixeletako bereizmena) prestaturik daudenez, unitatearen hasieraketan sistemaren ezaugarriak aztertuko dira. Hasieraketaren betebeharra, laburki esanda, sistema grafikoa martxan jartzea da horretarako ezagutzen dugun `InitGraph()` prozedura aplikatuko du, kontrolagailua eta modua finkoak baldin badira ezinezkoa zaigu jakitea bere hirugarren parametroa zein den. Dakigunez txartelak kontrolatzeko programak `BGI` izena duen direktorio batean aurkitzen dira, ordenadore batetik bestera direktorio horren kokamena ezberdina izan daiteke horregatik unitatearen erabiltzaileak teklatuz sartuko du datu hori hasieraketan.

`Grafiko` unitatearen inplementazioa

```
IMPLEMENTATION
USES
   Graph, Crt ;
VAR
   Kontrolagailua, Modua : Integer ;
   Mezua : String ;
   Itxaron : Char ;




BEGIN
   Write ('BGI fitxategien irispidea eman: (adibidez:  C:\TP70\BGI) ') ;
   ReadLn (Irispidea) ;

   Kontrolagailua := 9 ;                              { VGA = 9 }
   Modua := 2 ;                                       { VGAhi = 2 }
   InitGraph (Kontrolagailua, Modua, Irispidea) ;
   ErrKode := GraphResult ;

   IF ErrKode = grOk THEN                             { grOk = 0 }
   BEGIN
      OutTextXY (0,  0, 'Unitatearen hasieraketa') ;
      OutTextXY (0, 10, '-----------------------') ;
      Str (ErrKode, Mezua) ;
      OutTextXY (0, 30, 'Errore-kodea: ' + Mezua) ;
      Itxaron := ReadKey ;
      CloseGraph ;
   END
   ELSE
   BEGIN
      Writeln (#7, 'Erroren bat grafikoak irekitzean') ;
      Writeln ('Errore grafikoaren kodea ---> ErrKode=', ErrKode) ;
      WriteLn ('Errore grafikoaren mezua ---> ', GraphErrorMsg (ErrKode)) ;
      Itxaron := ReadKey ;
   END ;
END.
```

`9`, `2` eta `Irispidea` hiru parametro horiekin `InitGraph()` prozedurak sistema grafikoa irekitzen du eta errorerik suertatuz gero `ErrKode` aldagaian jasoko dugu. Aldagai bi horiek gordetzen dutena bezero-programak erabiliko ahal izango du publikoak baitira.

[10] Esate baterako, `Arku()` prozedurak darabilen `Trukaketa()` prozedura pribatua.

7.3.4 Unitate grafikoa erabiltzen

Lehentxoago esan den bezala, unitatearen `Irispide` eta `ErrKode` aldagai publikoak bezero-programarako aldagai orokorrak dira (**6.5.1 Aldagaien iraupena** eta **6.5.2 Aldagaien esparrua** puntuak gogoratu). Horregatik `HondartzaMarrazten` aplikazio-programak duen `IF` egiturako baldintza ebaluatzeko `ErrKode`-k hasieraketaren balioaz oinarrituko da, bigarren aldagai orokorrak, `Irispidea`-k, sistema grafikoa irekitzeko balio du.

Hona hemen irudia lortzen duen `HondartzaMarrazten` aplikazio-programa:

```
PROGRAM HondartzaMarrazten ;                    { \TP70\07\HONDARTZ.PAS }
USES
   Grafiko, Graph, Crt ;
VAR
   Kontrolagailua, Modua : Integer ;
   Mezua : String ;
   Itxaron : Char ;
BEGIN
   IF ErrKode = grOk THEN
   BEGIN
      Kontrolagailua := 9 ;                                 { VGA = 9 }
      Modua := 2 ;                                          { VGAhi = 2 }
      InitGraph (Kontrolagailua, Modua, Irispidea) ;

      Zirkunferentzi (100, 50, 40) ;                        { Eguzkia }
      Line (0, 107, 350, 107) ;                             { Itxasoa }

      Hiruki (40, 230, 240, 230, 140, 190) ;                { Etxea }
      Lauki (55, 230, 225, 230, 225, 300, 55, 300) ;
      Lauki (140, 300, 140, 250, 190, 250, 190, 300) ;
      Karratu (75, 250, 35, 0) ;

      Arku (345, 150, 40, 105, 170) ;                       { Mendia }
      Arku (400, 300, 200, 100, 110) ;
      Line (306, 145, 400, 145) ;
      Line (364, 103, 510, 90) ;
      MoveTo (529, 88) ;
      LineTo (555, 86) ;
      MoveTo (582, 84) ;
      LineTo (640, 80) ;

      Laukizuzen (335, 160, 80, 0, 10) ;                    { Portua }
      Arku (380, 160, 16, 80, 0) ;

      Arku (401, 184, 40, 0, 90) ;                          { Hondartza }
      Arku (361, 179, 80, 355, 290) ;
      Arku (325, 86, 180, 290, 273) ;
      Line (333, 266, 225, 272) ;
      Line (55, 276, 0, 280) ;
      Arku (318, 157, 80, 350, 270) ;
      Line (318, 237, 225, 240) ;
      Line (55, 242, 0, 244) ;

      Elipse (560, 260, 130, 100) ;                         { Zuhaitza }
      Karratu (545, 309, 30, 0) ;

      Hiruki (500, 110, 540, 110, 520, 70) ;                { Basoa }
      Karratu (514, 110, 12, 0) ;
      Hiruki (550, 100, 590, 100, 570, 60) ;
      Karratu (564, 100, 12, 0) ;
      Elipse (610, 120, 52, 32) ;
      Karratu (605, 135, 12, 0) ;

      Itxaron := ReadKey ;
      CloseGraph ;
   END;
END.
```

`HondartzaMarrazten` aplikazio-programak hiru unitate erazagutzen ditu, bakoitzaren zergatia ondoko taulan laburbiltzen da:

Unitatea	Ahalbideratzen duen erreferentzia
`Grafiko`	`ErrKode, Irispidea, Zirkunferentzi(), Arku(), Lauki(), Elipse(), Karratu(), Hiruki(), Laukizuzen()`
`Graph`	`grOk, InitGraph(), Line(), LineTo(), MoveTo(), CloseGraph`
`Crt`	`ReadKey()`

`HondartzaMarrazten` aplikazio-programan `ErrKode` aldagaiak `0` balio duela frogatu ondoren, sistema grafikoa ireki eta goiko taulan agertzen diren azpirrutinak egoki aplikatuz hondartzeren irudian osatzen dituzten elementuak marrazten ditu. Aplikazio-programa idatzi duenari ez dio, esate baterako, `Arku()` prozeduraren kodeketa buruhauste handirik sortzen, hori `Grafiko` unitatearen inplementazioan beste programadoreren batek idatzi baitu (hots gu geuk **7.1.2.1.7 Arkua** izenburuko puntuan izan genituen buruhausteak).

7.4 UNITATEEN ARIKETA: KOORDENATU-TRANSFORMAZIOAK

Koordenatu-transformazio ohizkoenak hiru dira: biraketa, traslazioa eta eskalatua. Jarraian bakoitzeko programa bi emango ditugu, eta guztien artean programa-liburutegi bat osa daiteke unitate bezala berridatziz gero.

7.4.1 Biraketa

Koordenatu-sistemaren jatorria zentrutzat harturik, pixel bat α angelu bat biratu nahi bada ondoko formulak aplikatuko zaizkio:

$$EmaitzaX = DatuX \cos \alpha + DatuY \sin \alpha$$

$$EmaitzaY = - DatuX \sin \alpha + DatuY \cos \alpha$$

`\TP70\07\KOORD_1A.PAS` fitxategian gorde den `KoordenatuAldaketak1A` izeneko programan lerro baten biraketak egiten dira. `KoordenatuAldaketak1B` izeneko programan berriz, hiruki bat da biratu egiten dena.

7.4.2 Traslazioa

Koordenatu-sistemaren translazioa bat koordenatuen kenketa bezala uler daiteke:

$$EmaitzaX = DatuX - KonstanteX$$

$$EmaitzaY = DatuY - KonstanteY$$

`\TP70\07\KOORD_2A.PAS` fitxategian gorde den `KoordenatuAldaketak2A` izeneko programan lerro baten traslazioak egiten dira. `KoordenatuAldaketak2B` izeneko programan berriz, hiruki bat da trasladatu egiten dena.

7.4.3 Eskalatua

Periferikoen kordenatuak ez dira beti berdinak izaten x eta y ardatzetan, eskala aldaketa arazo hori konpontzera dator. Behar diren datuak pixel baten koordenatuak eta eskala-faktorearen balioak dira:

$$EmaitzaX = DatuX * EskalaX$$
$$EmaitzaY = DatuY * EskalaY$$

`\TP70\07\KOORD_3A.PAS` fitxategian gorde den `KoordenatuAldaketak3A` izeneko programan lerro baten eskalatua egiten dira. `KoordenatuAldaketak3B` izeneko programan berriz, hiruki bat da eskalatu egiten dena.

7.5 UNITATEEN ADIBIDEA: ANIMAZIOAK

Irudi estatikoak marrazteaz gain animazioak egin daitezke konputagailu baten bitartez, horren hastapeneko adibide bezala hona hemen hiru programen deskribapena:

`KoordenatuakFrogatzeko1` Programa honek `Crt` eta `Erpin1` unitateak darabiltza, karratu bi eta ardatzak marrazten ditu baina estatikoak dira biratzen ez dutelako. Programa hau `\TP70\07\GURPIL1.PAS` delako fitxategian gordeta dago.

`KoordenatuakFrogatzeko2` Izeneko programa honek `Crt` eta `Erpin2` unitateak darabiltza, hiru karratu eta ardatzak marraztu ondoren karratuek duten zentruaren inguruan biratzen dute. `\TP70\07\GURPIL2.PAS` fitxategian gordeta dago.

`KoordenatuakFrogatzeko3` Programan `Crt` eta `Erpin2` unitateak darabiltza, zentru bera duten hiru karratuekin batera beste laugarren bat marraztu eta birarazi egiten da. Programa `\TP70\07\GURPIL3.PAS` fitxategian gordeta dago.

7.6 UNITATEEN ARIKETA: TRIGONOMETRIA ERRAZTEN

Ezaguna da Turbo Pascal lengoaiaren funtzio trigonometrikoak erabiltzean datuak radianak jarri behar direla, baina tamalez, gehienontzat errazagoa zaigu angeluak gradu-minutu-segundo hirukotearen bitartez ematea. Horregatik, eraiki dezagun unitate bat sinua, cosinua eta tangentea kalkulatzeko gaitasuna eskainiko diguna, baina datua izango den angelua, nola ez, gradu-minutu-segundo bezala emango zaio. Hau litzateke unitatearen eta balizko bezero programa baten deskribapena:

`Trigonom` Programa hau unitate bat da eta bere barnean `cos()` eta `sin()` funtzio estandarrak darabiltza. Unitate honi esker graduak, minutuak eta segundoak onartzen dituzten `Kosinu()`, `Sinu()` eta `Tangente()` funtzioak definitu eta garatu egiten dira. Ikus `\TP70\07\TRIGONOM.PAS` fitxategia.

`TrigonometriaBezeroPrograma` Izeneko programa aurreko `Trigonom` unitatea darabil, gradu-minutu-segundo bezala definituriko angelu baten kosinua kalkulatzen du, eta bere kodea aztertu ahal izateko `\TP70\07\TRIG_BEZ.PAS` izena duen fitxategia ireki.

7.7 PROGRAMAK

Hona hemen 7.kapituluaren programak orrialdeen arabera sailkaturik:

Izena	*Programaren identifikadorea*	*ORRI.*	*Ikasgaia*
GRAF_0.PAS	GrafikoakLantzen0	7-06	Unitate grafikoa
GRAF_1.PAS	GrafikoakLantzen1	7-08	Unitate grafikoa
GRAF_2.PAS	GrafikoakLantzen2	7-08	Unitate grafikoa
GRAF_3.PAS	GrafikoakLantzen3	7-10	Unitate grafikoa
GRAF_4.PAS	GrafikoakLantzen4	7-11	Unitate grafikoa
GRAF_5.PAS	GrafikoakLantzen5	7-12	Unitate grafikoa
GRAF_6.PAS	GrafikoakLantzen6	7-13	Unitate grafikoa
GRAF_7.PAS	GrafikoakLantzen7	7-14	Unitate grafikoa
GRAF_8.PAS	GrafikoakLantzen8	714	Unitate grafikoa
GRAF_9.PAS	GrafikoakLantzen9	7-15	Unitate grafikoa
GRF_HIRU.PAS	HirukiaMarrazten	7-18	Unitate grafikoa
GRF_LAUK.PAS	LaukiaMarrazten	7-19	Unitate grafikoa
GRF_KARR.PAS	KarratuaMarrazten	7-20	Unitate grafikoa
GRF_LZUZ.PAS	LaukiZuzenaMarrazten	7-21	Unitate grafikoa
GRF_ZIR1.PAS	Zirk1Marrazten	7-23	Unitate grafikoa
GRF_ZIR2.PAS	Zirk2Marrazten	7-24	Unitate grafikoa
GRF_ELI1.PAS	Elip1Marrazten	7-25	Unitate grafikoa
GRF_ELIP.PAS	ElipseaMarrazten	7-25	Unitate grafikoa
GRF_ARK0.PAS	Arku0Marrazten	7-26	Unitate grafikoa
GRF_ARK1.PAS	Arku1Marrazten	7-26	Unitate grafikoa
GRF_ARK2.PAS	Arku2Marrazten	7-26	Unitate grafikoa
GRF_ARK3.PAS	Arku3Marrazten	7-26	Unitate grafikoa
GRF_ARK4.PAS	Arku4Marrazten	7-26	Unitate grafikoa
GRF_ARK5.PAS	Arku5Marrazten	7-27	Unitate grafikoa
GRF_ARKU.PAS	ArkuaMarrazten	7-27	Unitate grafikoa
GRF_SIN1.PAS	Sinu1Marrazten	7-30	Unitate grafikoa
GRF_SIN2.PAS	Sinu2Marrazten	7-30	Unitate grafikoa
ADIB1.PAS BEZERO1.PAS	Adib1 Bezero1Programa	7-35	Unitate baten sorrera
MAILA1.PAS MAILA2.PAS BEZERO2.PAS	Maila1 Maila2 Bezero2Programa	7-37	Unitateekin gertaerak
GRAFIKO.PAS HONDARTZ.PAS	Grafiko HondartzaMarrazten	7-42	Gure unitate grafikoa
KOORD_1A.PAS	KoordenatuAldaketak1A	7-43	Programa grafikoa
KOORD_1B.PAS	KoordenatuAldaketak1B	7-43	Programa grafikoa
KOORD_2A.PAS	KoordenatuAldaketak2A	7-43	Programa grafikoa
KOORD_2B.PAS	KoordenatuAldaketak2B	7-43	Programa grafikoa
KOORD_3A.PAS	KoordenatuAldaketak3A	7-44	Programa grafikoa
KOORD_3B.PAS	KoordenatuAldaketak3B	7-44	Programa grafikoa
GURPIL1.PAS ERPIN1.PAS	KoordenatuakFrogatzeko1 Erpin1	7-44	Animazioak
GURPIL2.PAS ERPIN2.PAS	KoordenatuakFrogatzeko2 Erpin2	7-44	Animazioak
GURPIL3.PAS ERPIN3.PAS	KoordenatuakFrogatzeko3 Erpin3	7-44	Animazioak
TRIGONOM.PAS TRIG_BEZ.PAS	Trigonometria TrigonometriaBezeroPrograma	7-44	Unitate baten adibidea

7.8 BIBLIOGRAFIA

- Decker R., Hirshfield S., *Pascal's Triangle*, PWS-Kent Publishing Company, Boston, 1992
- Berger M., *Graficación por Computador*, Addison-Wesley Iberoamericana, Wilmington, 1991
- Salmon W.I., *Introducción a la Computación con Turbo Pascal*, Addison-Wesley Iberoamericana, Wilmington, 1993

KONTZEPTUEN INDIZE ALFABETIKOA

—E—

—F—

—G—

—H—

—I—

—K—

—L—

—M—

—N—

—O—

—P—

—R—

—S—

—T—

—U—

—V—

—W—

—X—

—Z—