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.

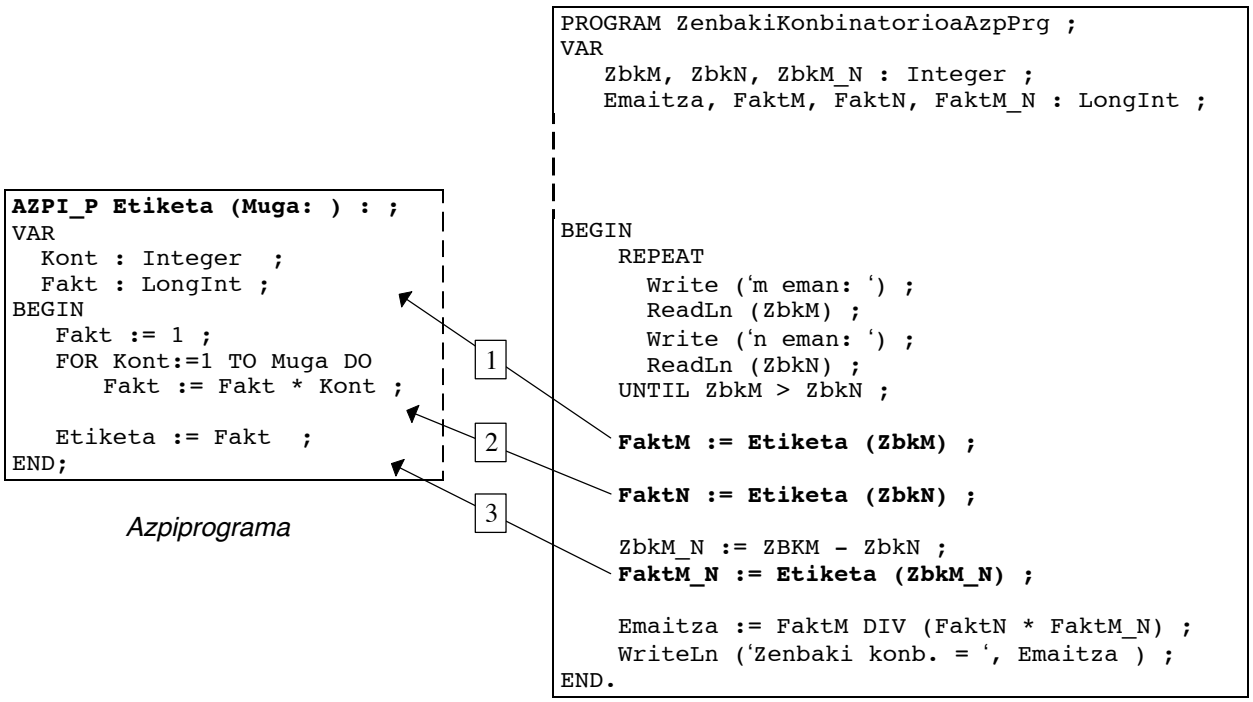
Azpiprogramaren kontzeptuak kodea ez errepikatzea dakar. Horretarako faktoriala ematen digun sententzien multzoa etiketatu eta toki bakar batean idatziko dugu, gero zerbitzu hori behar den unean eskatu (deitu) ahal izateko. Honelako bidea aukeratzean mailaketa[3] bat agertzen da azpiprogramaren kontzeptuak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



`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;
```

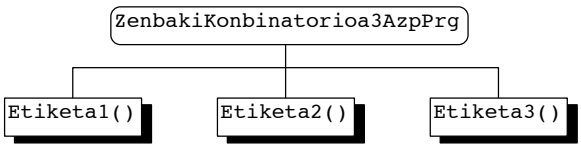
```
AZPI_P Etiketa3 (FktM, FktN, FktM_N: ) ;
VAR
  Ema : LongInt ;
BEGIN
 Ema := FktM DIV (FktN * FktM_N) ;
 WriteLn ('Zenb. konb. = ', Ema ) ;
END;
```

```
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:



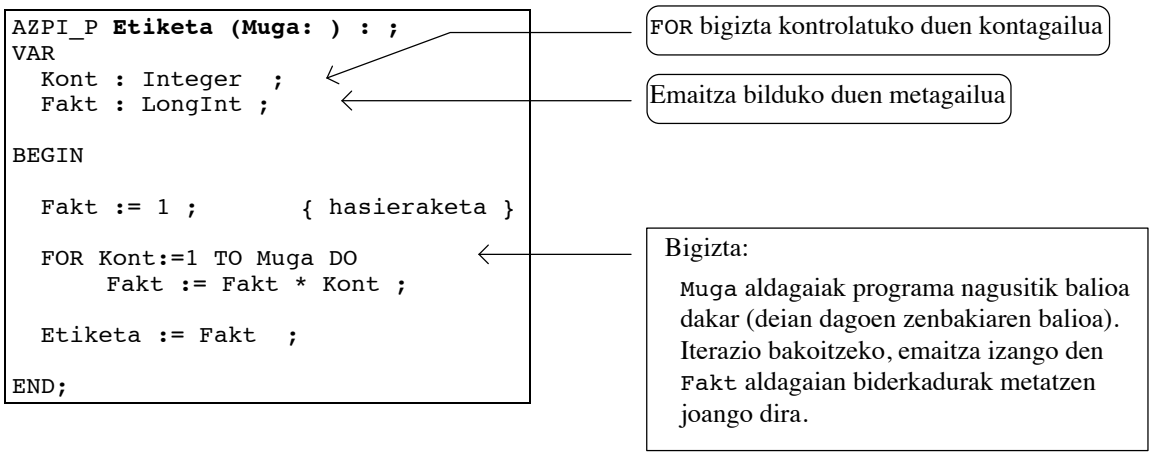
6.2.3 Kodearen independentzia

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.

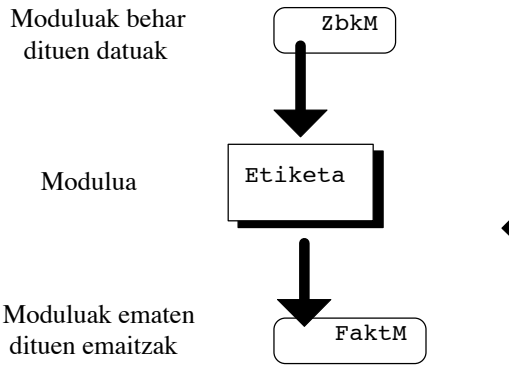
`Etiketa()` izeneko moduluari begiratzen badiogu honako hau esan dezakegu: azpiprograma batek, programa batek bezala, emaitzak lortzeko asmoz datuen aldaketa bat egiten du, baina datuak teklatutik irakurri beharrean kanpotik heltzen zaizkio, eta emaitzak pantailara igorri beharrean programa nagusira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

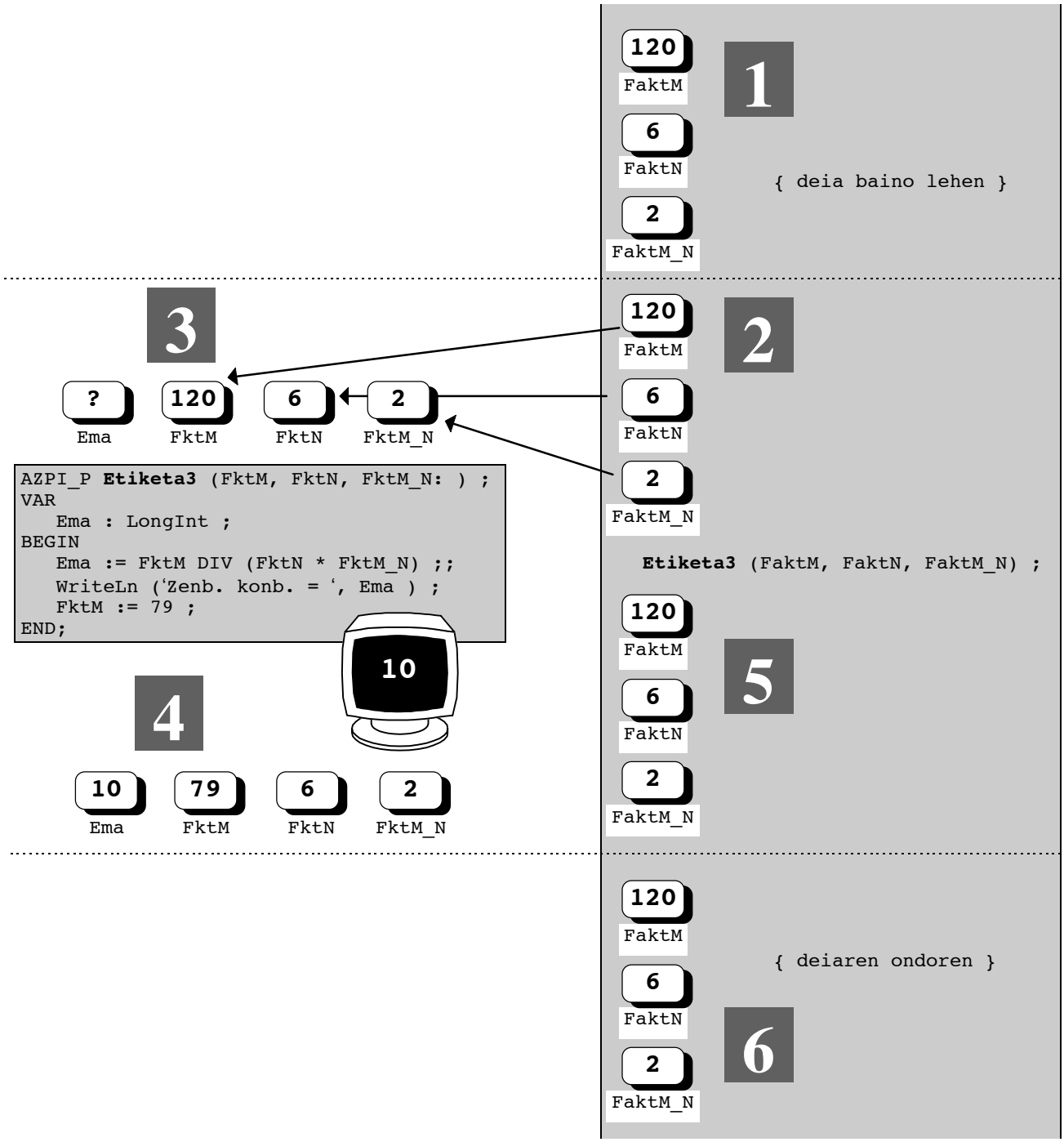
Programa nagusiaren zatia

`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 esangura-tsuagoak 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*).

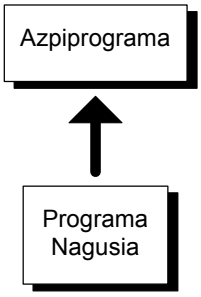
Parametroen jokaerak banan banan aztertuko ditugu hurrengo **6.2.3.1-6.2.3.2-6.2.3.3** ataletan.

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).

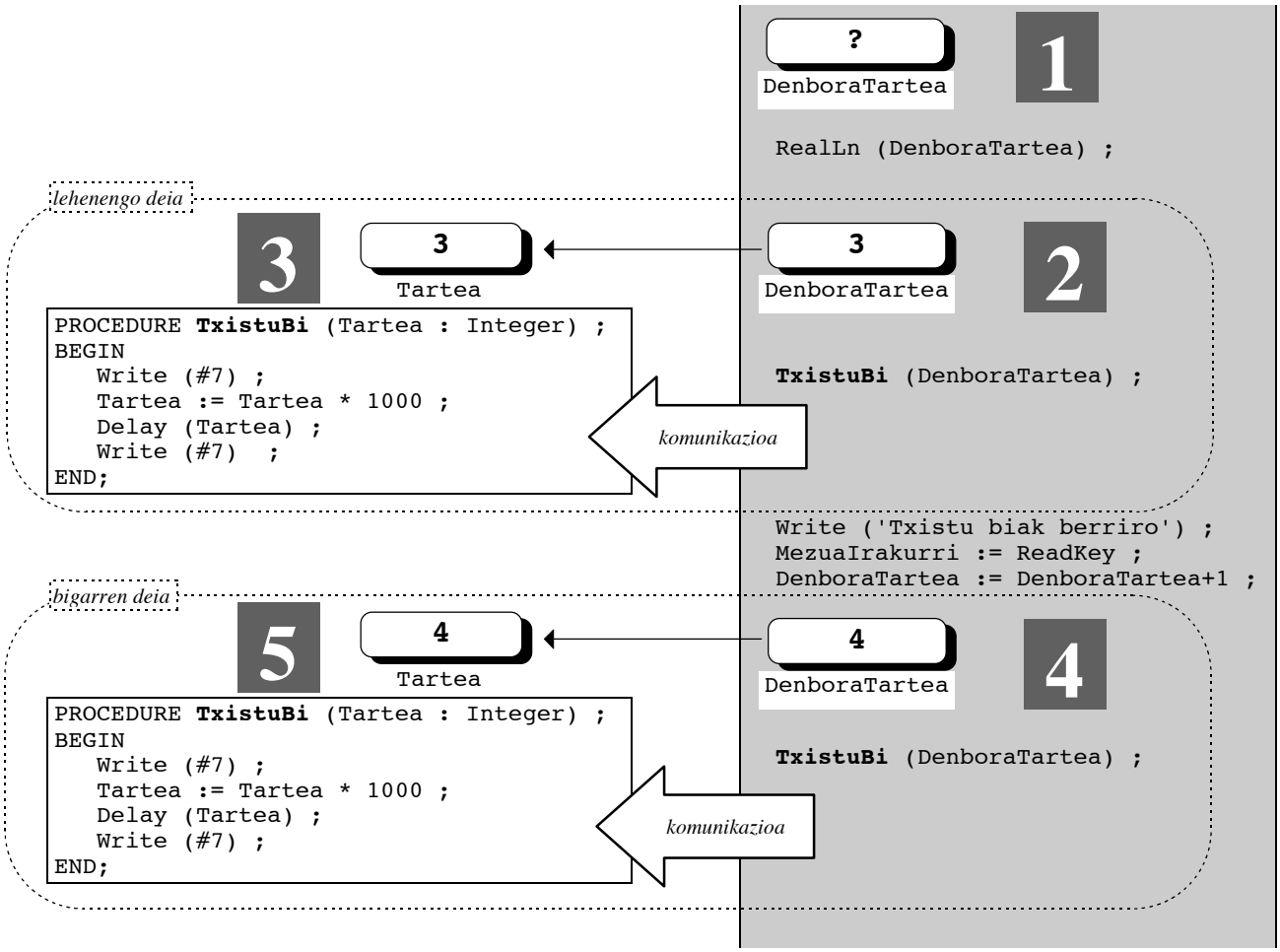
Hurrengo adibidean programa nagusitik datuak bidaliko dira azpiprogramari. Oraintsu erabilitako `Txistuak` prozedura aldatu egin da, txistu biren artean dagoen denbora tartea derrigorrez bi segundotakoa izan beharrean aukeragarria izan dadin. Programa nagusian ezagutzen den `DenboraTartea` aldagaiaren 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.

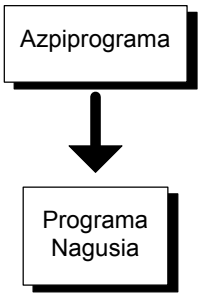
4 `TxistuBi()` prozeduraren bigarren deia. Azpirrutina deitu aurretik `DenboraTartea` aldagaiaren balioa inkrementatu egiten denez parametro erreal horrek `4`ko balioa du eta ...

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.

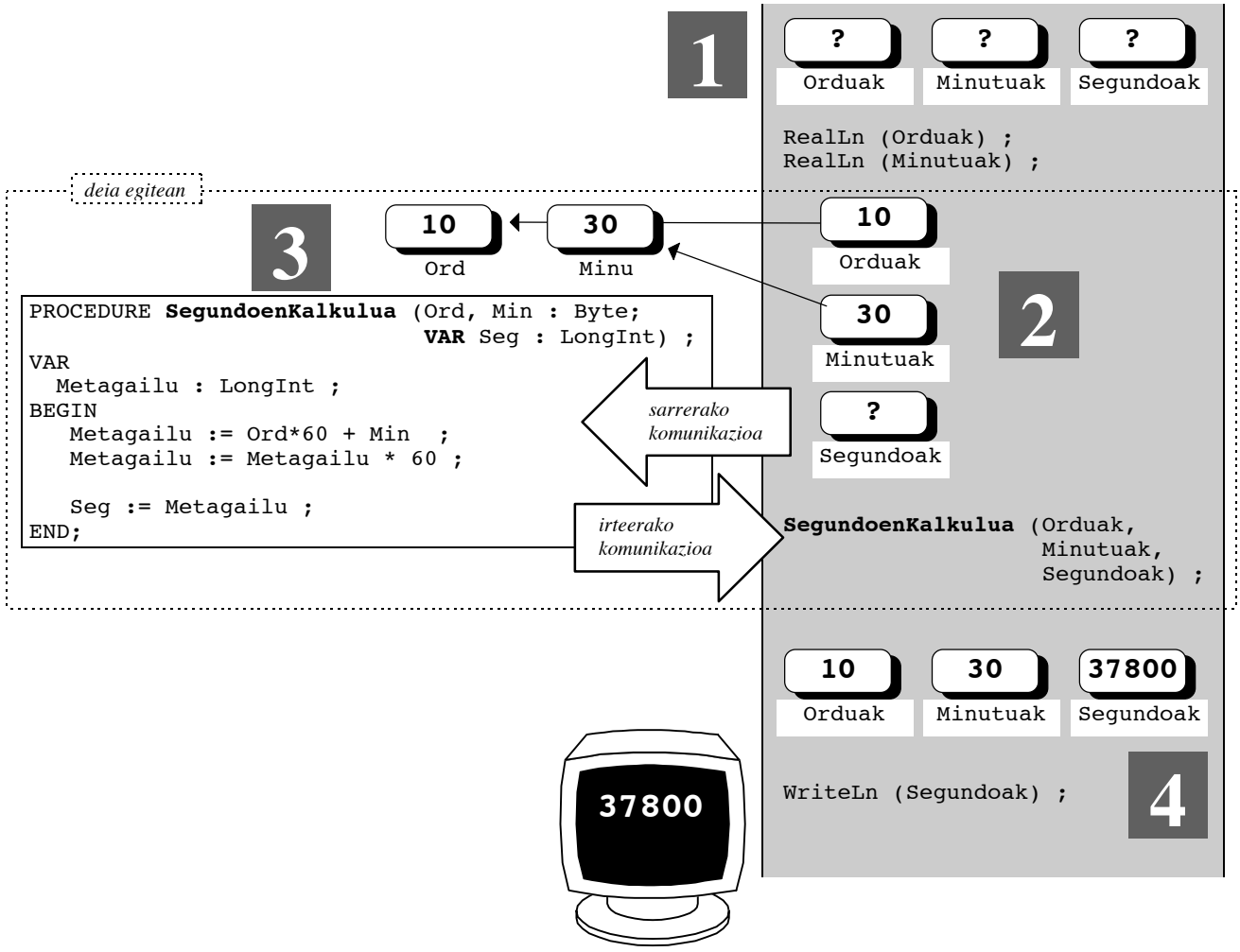
Ondoren `SegundoenKalkulua()` prozedura pizten da; zeinek `Orduak` eta `Minutuak` aldagaiak sarrerako parametro errealak erabiltzen dituen eta `Segundoak` irteerako parametro erreala den. Lehenengo biek `SegundoenKalkulua()` deia egiten denerako balio ezagunak izango dituzte, baina hirugarrenari balioa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

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.

`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		
	`MinutuIrakurketa()`		irteerakoa	
	`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).

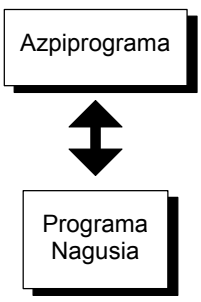
Non adierazten da parametro batek sarrerako ala irteerako portaera izango duen?. Zeharka bada ere **6.3.2.1.1** puntuan eta **6.3.2.2.1** puntu honetan ikusi dugu; *parametro batek duen jokaera azpiprogramaren goiburukoan, parametro formalaren bitartez zehazten da*. Adibide berdinaren bigarren bertsioarekin jarraituz parametro formalen taula bete dezagun:

		Parametro Formalak		
		`Ord`	`Min`	`Seg`
Goiburukoak	`OrduIrakurketa()`	`VAR`		
	`MinutuIrakurketa()`		`VAR`	
	`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.
```

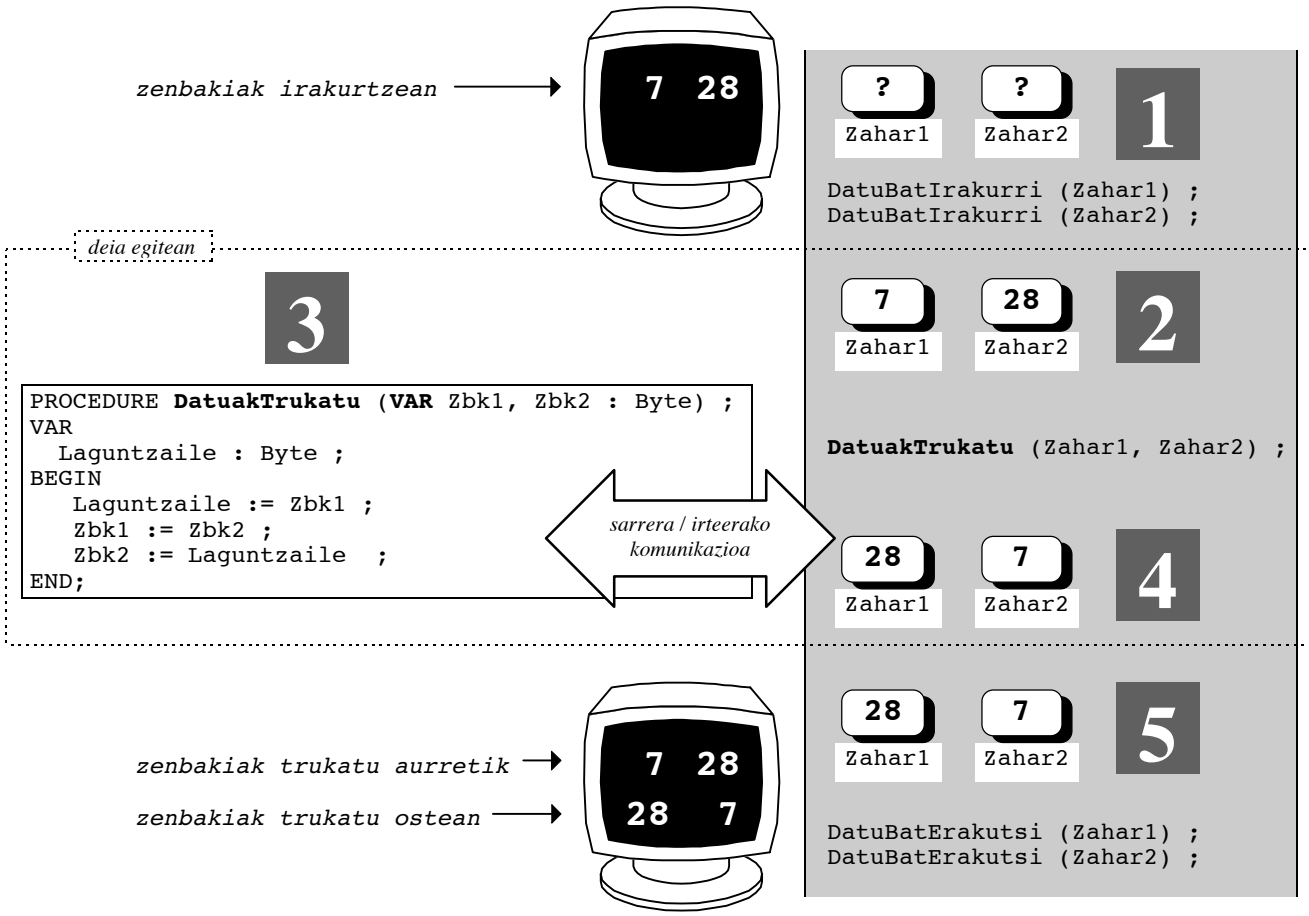
`SarreraIrteerakoParametroa1` izeneko programan dagoen desberdintasun bakarra `DatuakTrukatu()` prozeduran sarrera/irteera parametroak agertzen direla da. Horren deia egiten denean `Zahar1` eta `Zahar2` programa nagusiko aldagaiek derrigorrez balio ezagunak izango dituzte eta `Zbk1` zein `Zbk2` parametro 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`		
`DatuakTrukatu()`	sarrera irteerakoa	sarrera irteerakoa		`VAR`	`VAR`
`DatuBatErakutsi()`	sarrerakoa	sarrerakoa			

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):

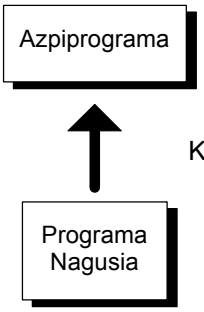
1. Baliozko parametroa
2. Aldagai-parametroa
3. Konstante-parametroa

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.

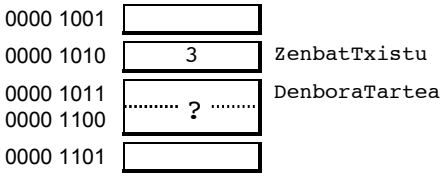
Baina baliozko parametroaren pasatze era adibide baten bitartez ikas dezagun. Demagun lehenago landutako `SarrerakoParametroa3` programaren beste bertsio bat egin nahi dugula, eta desberdintzeko `SarrerakoParametroa4` izendatuko dugula. Programa berri honek egingo duena honela definitzen da:

> *"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.
```

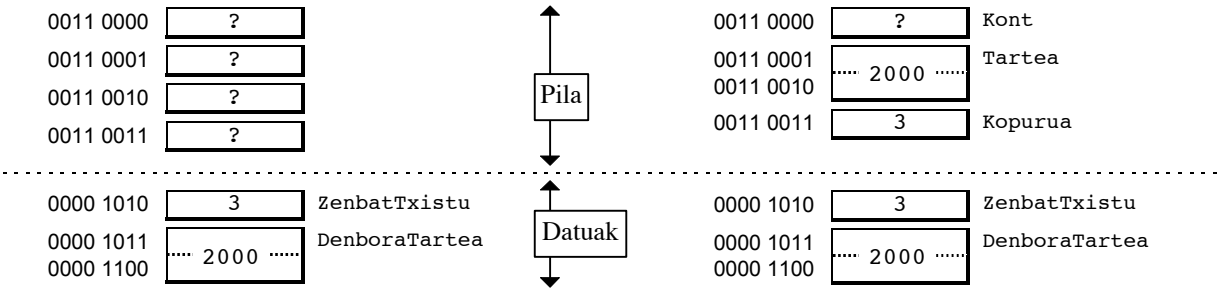
Ikus daitekeenez `SarrerakoParametroa4` programa nagusian `ZenbatTxistu` eta `DenboraTartea` aldagaiak deklaratu dira, lehenengoa `Byte` eta bigarrena `Integer` direlarik[11]. Demagun aldagai bakoitzaren memoria helbideak ondoko irudian emandakakoak direla, `SarrerakoParametroa4` programaren exekuzioa 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 ...

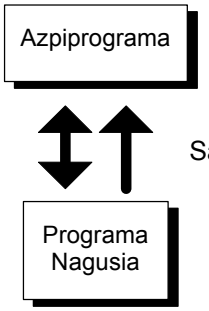
6 ... lehenengo urratsean aipatu dugun itzultzeko helbideaz baliatuko da. Beste modu batean azaldurik, azpiprogramaren deia egitean pilan grabaturik utzi den programa nagusiaren (modulu deitzailearen) hurrengo sententziaren helbidera jauzi egiten da, kontrola modulu deitzaileak berreskuratuz.

[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.

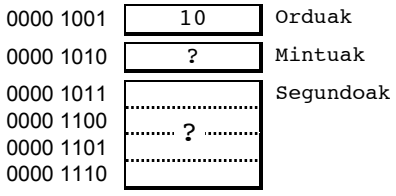
Sarrera/Irteerako Komunikazioa — Irteerako Komunikazioa — *Aldagai Parametroak*

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) ;
```

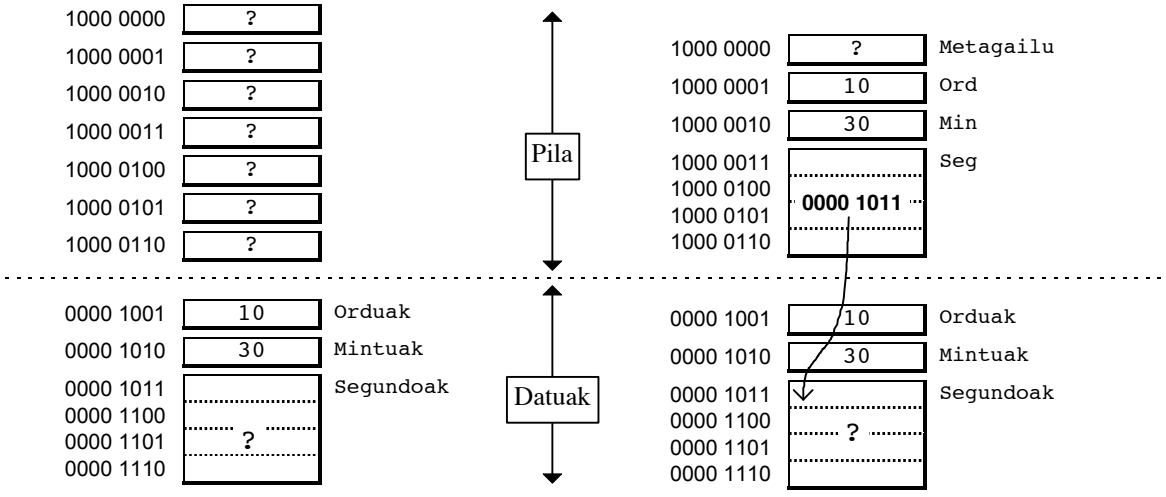
Bigarren irakurketa, `Minutuak` aldagaiarena, egin eta gero `0000 1010` helbidedun gelaska beteko litzateke. Baina `Segundoak` aldagaiak `SegundoenKalkulua()` prozeduraren barruan balioa hartzen duenez aldagai-parametro bezala pasatu beharko zaio, oraingoan ere parametro pasatze hau 6 urratsetan banatuko 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 }
```

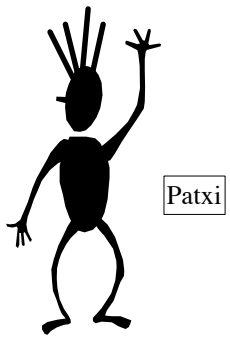
6.3.4 Azpiprogrameen arteko komunikazioa. Laburpena

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	~~X~~	`VAR`	`CONST`
Komunikazioa	Sarrera	Sarrera / Irteera Irteera	Sarrera
Param. formala	Balio bat	Helbide bat	Helbide bat
Param. erreala	Babesturik	Babestu gabe	Babesturik

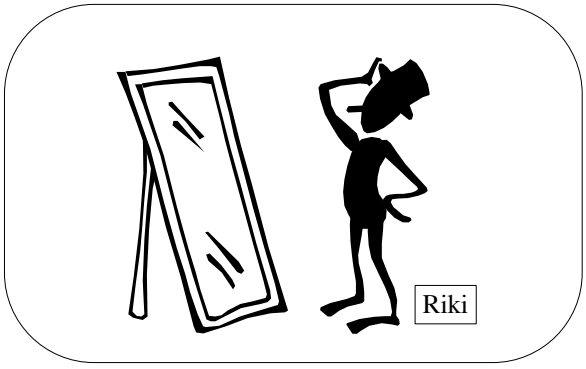
Taularen informazioa biltzen duen istorio bat kontatuko dugu orain, Patxi Punki-ren alegia. **Patxi Punki** bere herriko kale nagusitik dabilen laguna dugu, eta programa nagusiko aldagaitzat hartuko dugu. Bere herriko kale nagusian zenbait zerbitzu (zenbait azpirrutina) eskuragarri du Patxik, besteak 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:

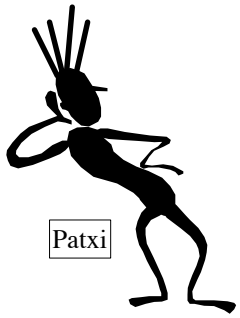


Patxi Punki kale nagusian

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:

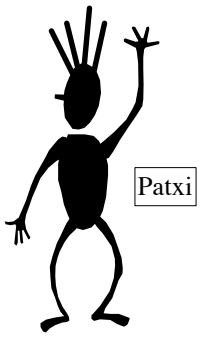


Riki Punki jostundegaiaren barruan



Patxi Punki kale nagusian itxaroten

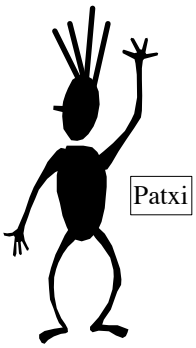
Jostundegiaren azpirrutina bukatu egiten denean Riki desagertzen da, eta kale nagusiko Patxik ez du arropa berririk jantziko.



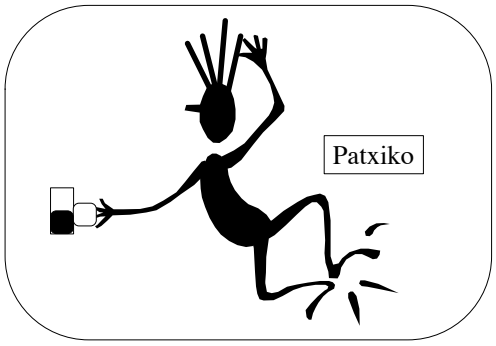
Patxi Punki berriro kale nagusian

B

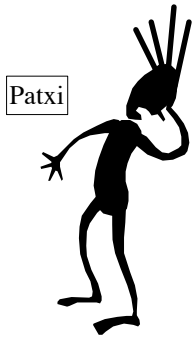
Gure Patxi egarri da eta oraingoan tabernara doa. Taberna barruan tabernariak, konfidantza duenez, Patxiko deitu egiten dio. Pertsona berbera izan arren kalean Patxi bezala ezaguna da eta tabernan Patxiko deritzo gure lagunari. Konturatzen gara Patxik edandakoa berak jasan beharko duela (Riki anairik ez baitago), Patxikok larregi edan ondoren horra hor kalera irtetean Patxik erakutsiko duen egoera.



Patxi kale nagusian tabernan sartu aurretik

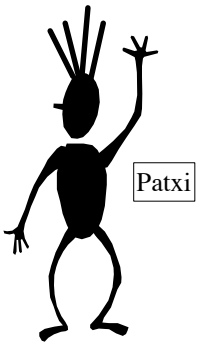


Patxiko (Patxi Punki) taberna barruan

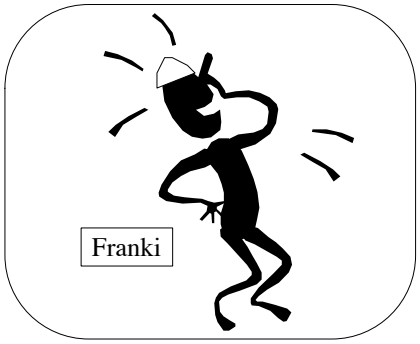


Patxi kale nagusian horrituta

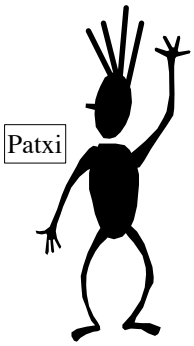
Kale nagusian dagoela ilea moztera erabaki du Patxik, horregatik Patxi Panki ileapaindegirantz abiatzen da; oraingoan ere Patxi bera doa Riki anaiaz baliatu gabe. Ileapaintzaileak Patxi Punki Franki bezala ezagutzen du. Tabernara ez bezala Patxi ileapaindegi barnera sartzean burua kasko batez babesten 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



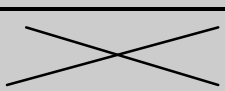
Franki (Patxi Punki) ileapaindegian gandorra babesturik



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.

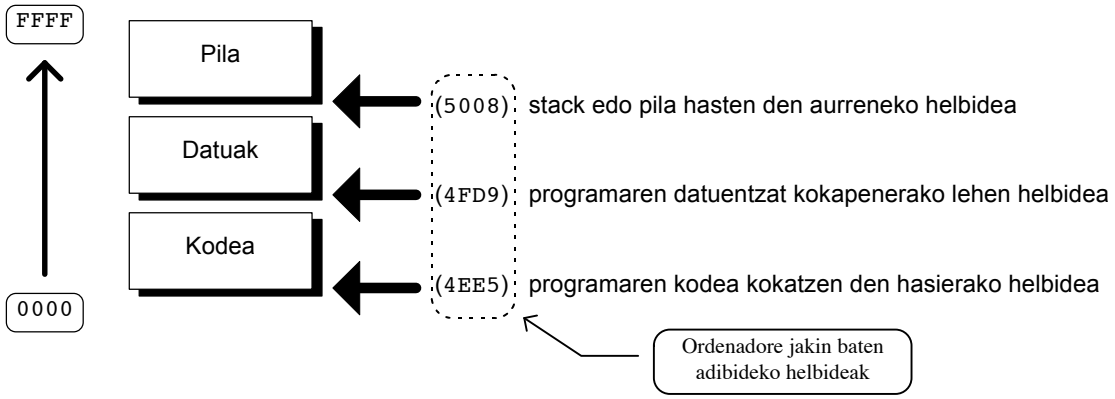
Patxi Punki-ren alegia ikusi ondoren hauxe da bere laburpena:

	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:

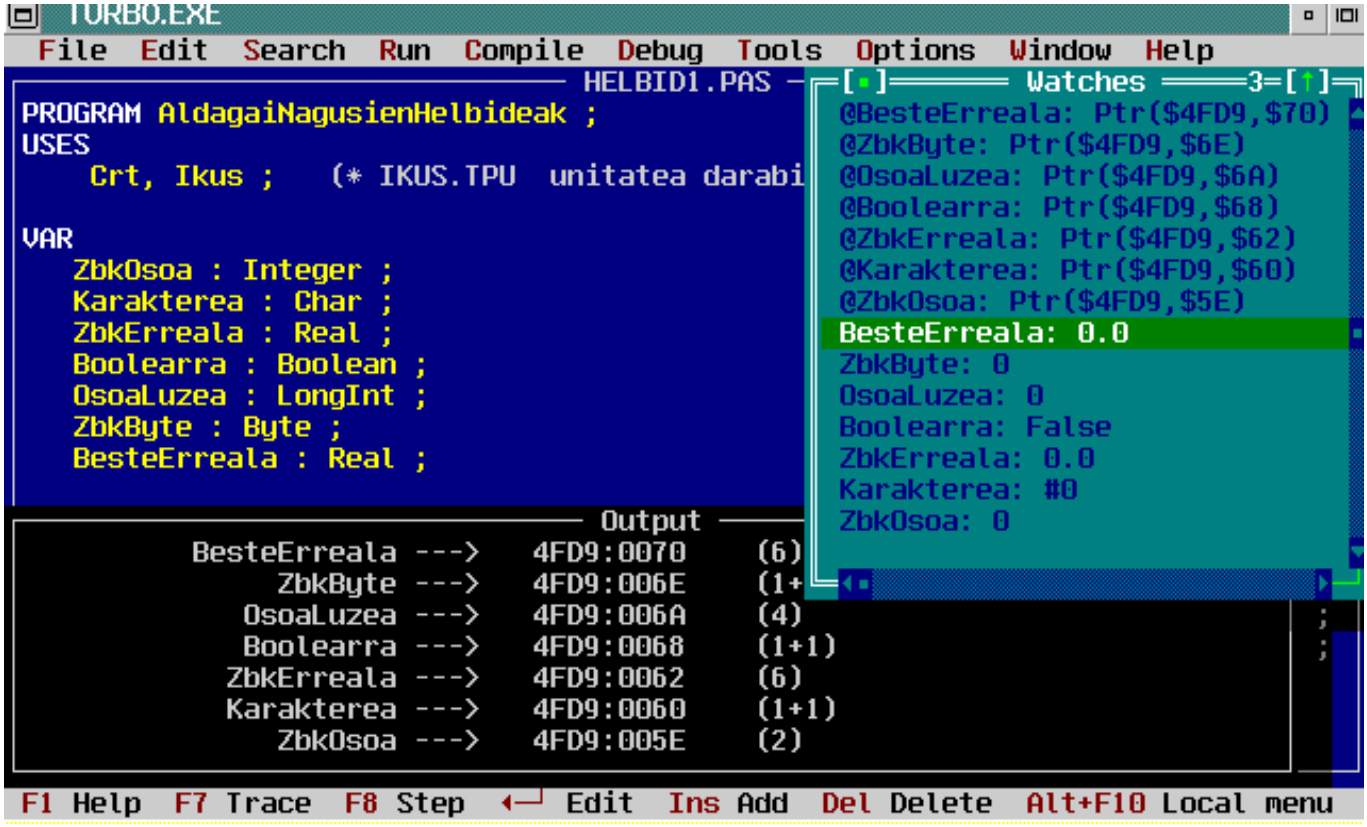
```
PROGRAMA NAGUSIAN EZAGUTZEN DIREN ALDAGAIEN MEMORI HELBIDEAK
============================================================

PILARI dagokion segmentuaren hasiera:  5008
DATUEI dagokion 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)
_
```

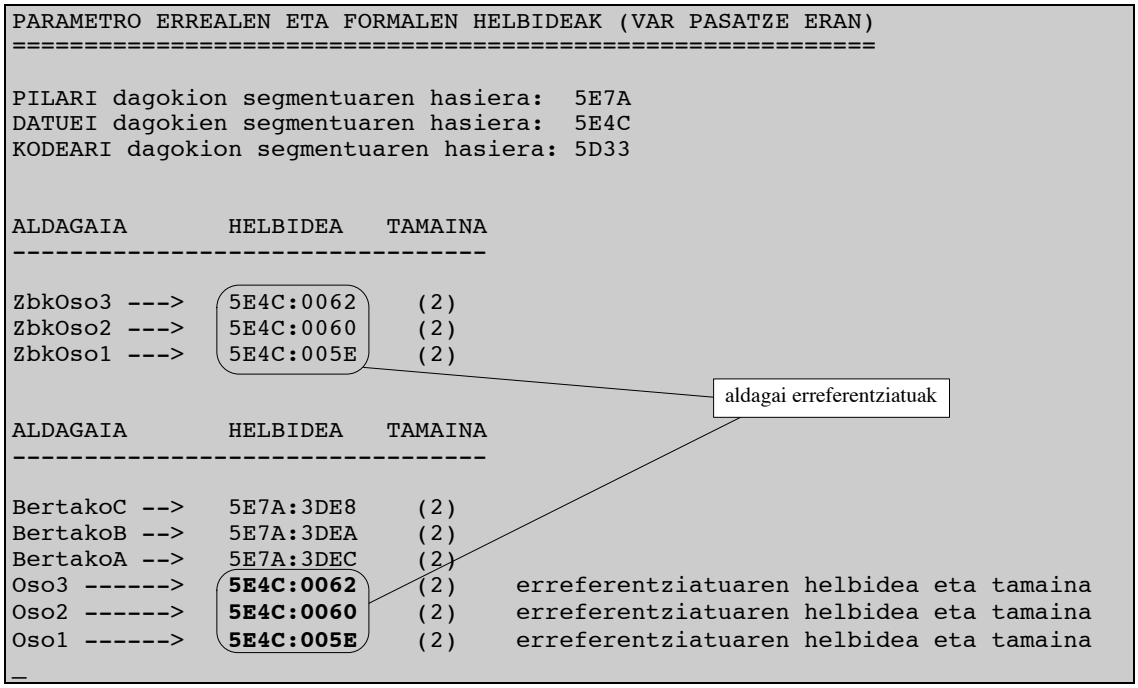
Programa nagusiko `ZbkOso1`, `ZbkOso2` eta `ZbkOso3` izeneko aldagaiak zenbaki osoak dira eta datuen segmentuan daude (adibide honetan `5E4A` helbideko segmentuan daude, eta bertan bi byteko desplazamendua dute). `AzpirrutinaBaliozko()` prozedurako `Oso1`, `Oso2` eta `Oso3` parametroak eta `BertakoA`, `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

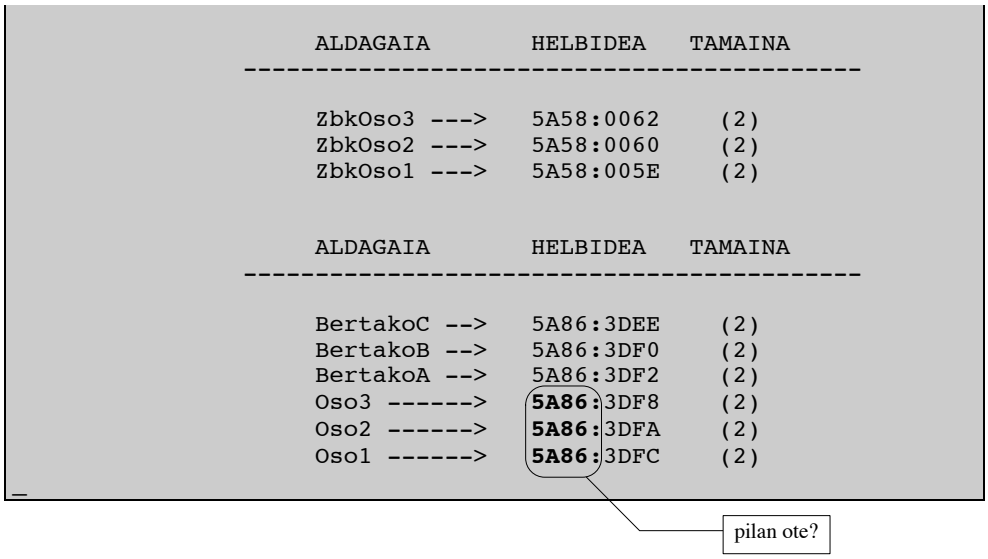
`ParametroFormalenHelbideakVAR_A` izeneko programan `VAR` pasatze era `CONST` pasatze moduagatik ordezkatuko bagenu, orain arte ikusitakoarekin erraza litzaiguke asmatzea programaren irteera zein izango litzatekeen. Konstante-parametro eta aldagai-parametro programen irteerak funtsean berdinak izan beharko 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

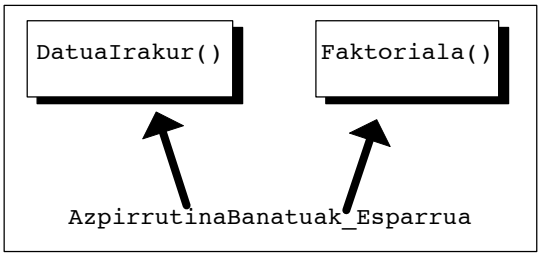
Kasu honetan `ZnbkNag` aldagaia ezezaguna da `Azpirrutina` izeneko prozeduran, ondorioz programaren irteera honelako zerbait izango litzateke:

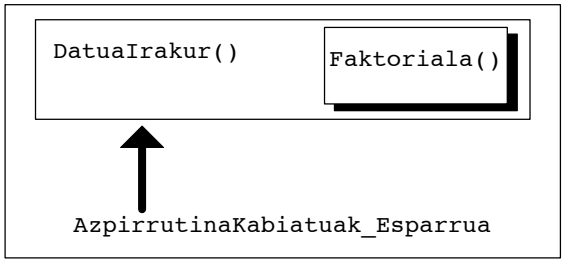
```
1 programa nagusian  ZnbkNag=0
Azpirrutina barruan  ZnbkAzp=22782
_
```

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.

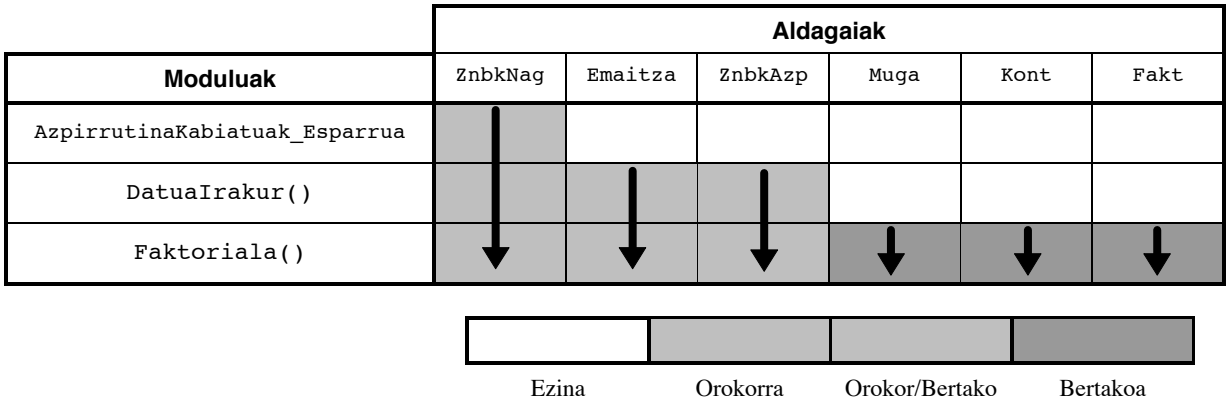
`AzpirrutinaBanatuak_Esparrua` programari dagokion esparru-taula:

	Aldagaiak					
Moduluak	`ZnbkNag`	`Emaitza`	`ZnbkAzp`	`Muga`	`Kont`	`Fakt`
`AzpirrutinaBanatuak_Esparrua`	↓	↓				
`DatuaIrakur()`			↓			
`Faktoriala()`	↓	↓		↓	↓	↓

Ezina Orokorra Bertakoa

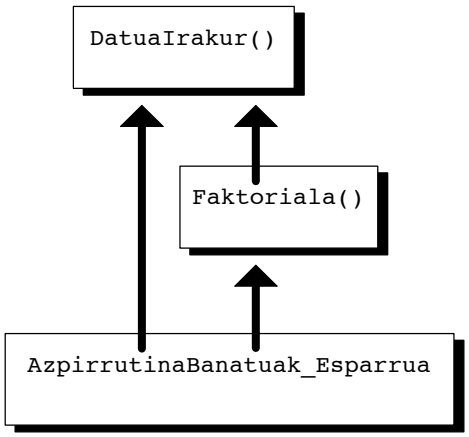
`AzpirrutinaBanatuak_Esparrua` programaren aldagaiak bi mailetan antolatzen dira, programaren hiru modulutan erabil daitezkeen `ZnbkNag` eta `Emaitza` aldagai orokorrak, eta, azpirrutina bakoitzaren bertako edo pribatuak diren bertako aldagaiak. Aldagairen bat erreferentzia modulu batean ezinezkoa denean 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`):

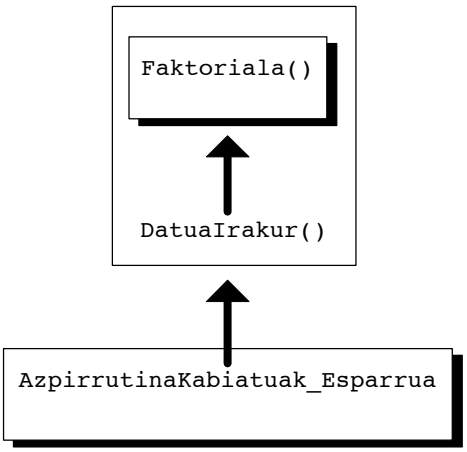


Ikus daitekeenez azpirrutinak kabiatzen direnean gerta daiteke aldagairen bat aldiberean pribatua eta orokorra izatea, `Emaitza` aldagaia esate baterako, kabiaturiko modulu edo moduluetako orokorra izango da eta kanpoko moduluek ezin izango dute atzitu pribatua delako eurentzat.

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.
```

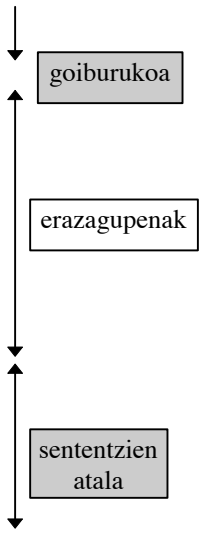
Hauxe da `AldagaiOrokorra`-ri dagokion irteeraren bat:

```
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.

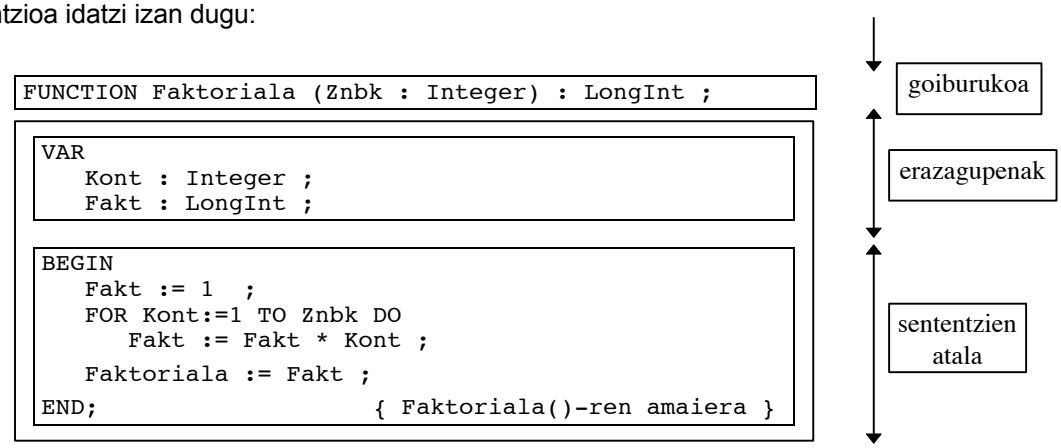
Prozedura bat, beraz, eginkizun jakin bat burutzeko sententzien multzoa litzateke (pantaila garbitzea, balioak bistaratzea, datuak aldagaietan gordetzea, ...). Prozedura batek behar izan dezake sarrerako parametrorik (baliozko parametroak edo eta konstante-parametroak), eta irteerarik kanporatu behar 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:



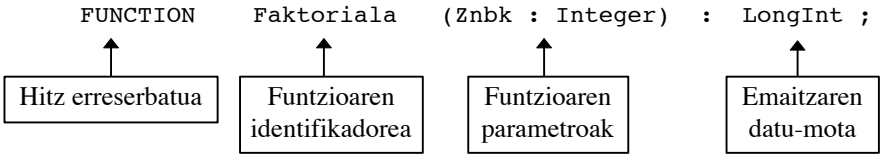
Funtzioaren egitura, ikus daitekeenez, Turbo Pascal programa baten egitura berbera da (laugarren kapituluko **4.4 PROGRAMA BATEN EGITURA** gogoratu), kasu honetan deklarazioak `Faktoriala()` funtzioaren sententzien atalan ezagunak izango dira (funtzioaren sententzien atalan eta izango lituzkeen beste 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).

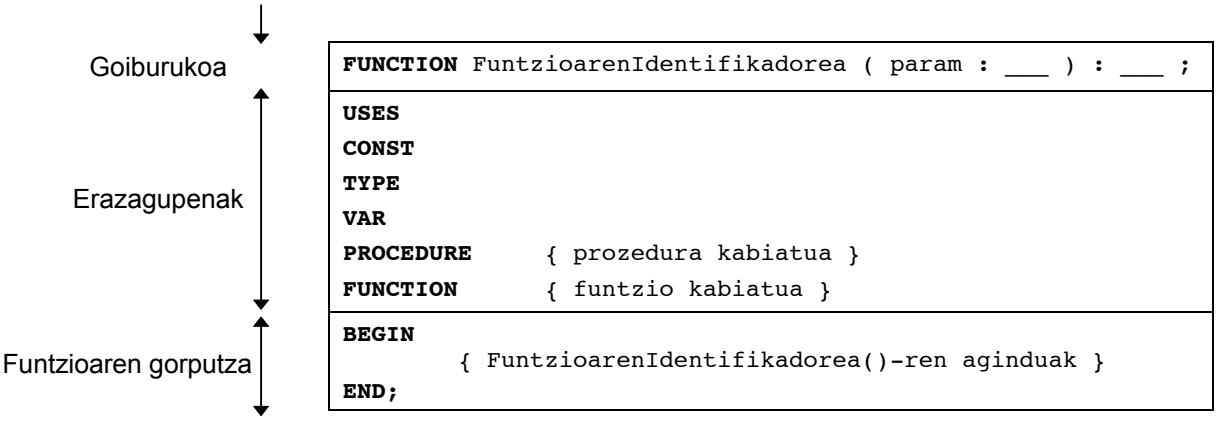
Funtzioaren identifikadoreak berebiziko garrantzia izango du, eta bi zereginetarako balio duela esan genezake. Batetik funtzioa bera deitu ahal izateko (dagoenekoz ez da guretzeat berria edozein azpiprograma pizteko bere etiketa behar dela); baina bestetik badu etiketa horrek funtzioentzat berezia den 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:

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

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:

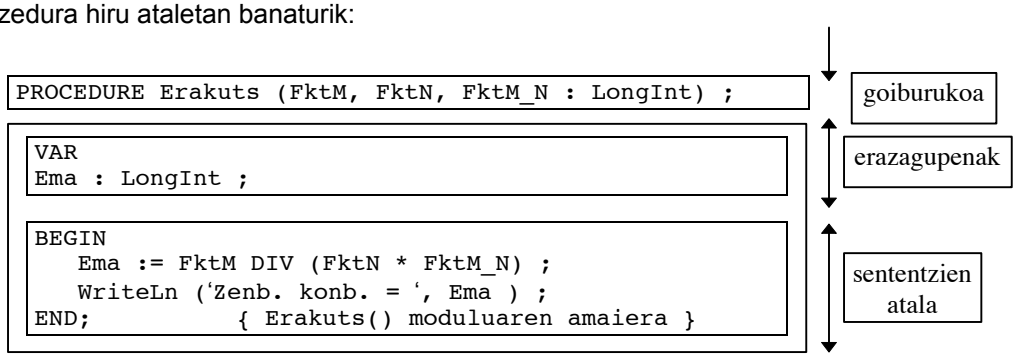
Modulua	Mota	Helburua	Itzulitakoa
`Faktoriala()`	Funtzioa	Parametroaren faktoriala kalkulua lortzea	Zenbaki oso bat
`DatuakHar()`	Prozedura	Bi balio oso teklatuz irakurri ahal izatea	Bi zenbaki oso
`Erakuts()`	Prozedura	Balio oso baten pantailaraketa	

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

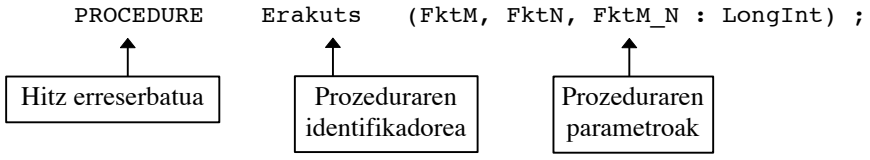
Edozein prozedurak, ikasi ditugun funtzio baten hiru zatiak izango ditu, jarraian datozen puntuetan zehaztasunez azaltzen direnak. Hau litzateke, esate baterako, `Erakuts()` prozedura hiru ataletan banaturik:



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).

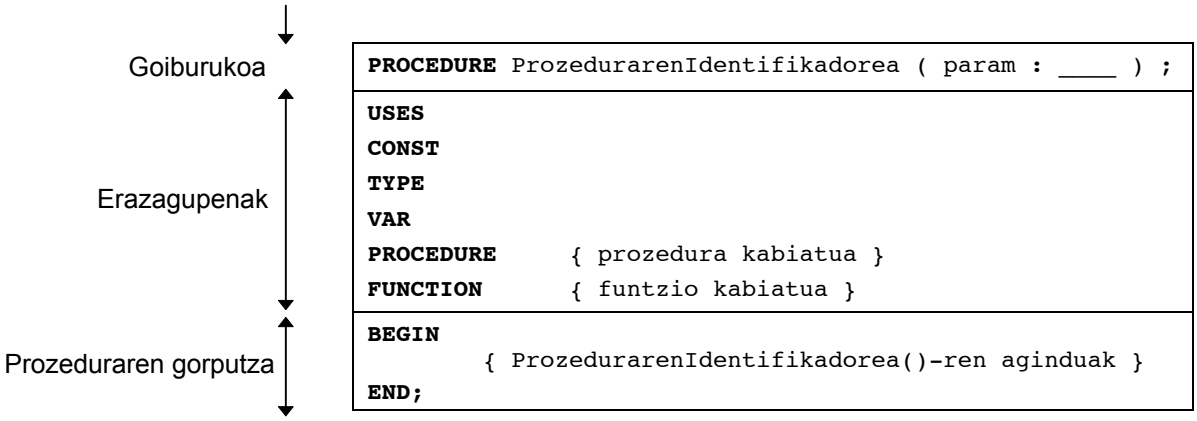
Modulu nagusiak prozedura bat aktibatzeko dei bat egin beharko dio eta horretarako prozeduraren identifikadorea erabiliko du, eta hori izango da prozedura baten identifikadoreak izango duen zeregin bakarra (funtzioetan azpirrutinaren identifikadoreak dituen ardurak bi direla ez ahaztu).

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:

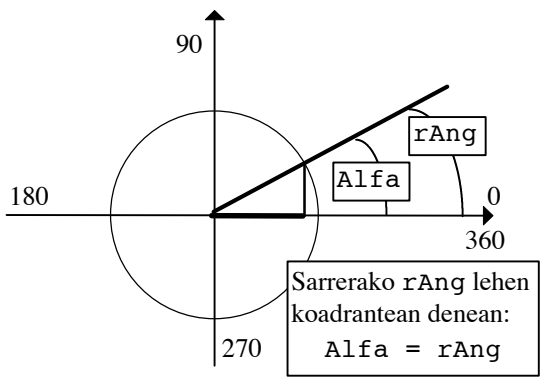
Programari eskatzen zaion doitasuna lasatzea, hau da `EPSILON` konstantea `0.0005` izan beharrean handiagoa izatea. Adibidez doitasuna honela definituz `EPSILON=0.0006` seigarren iterazioan batugaiaren balioa azpitik geratzen da eta `WHILE-DO` bigiztaren balditza beteko litzateke, `161`-ren kosinua zuzentasunez lortzen delarik.

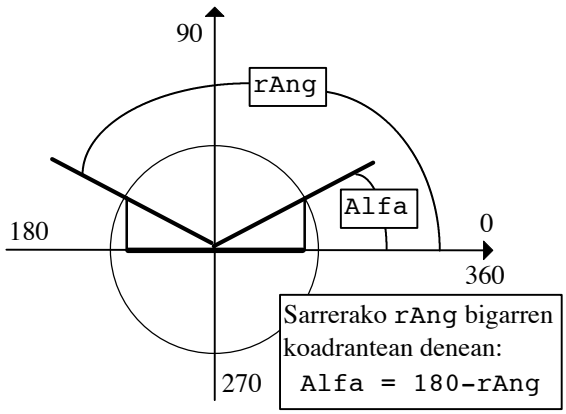
Tamalez, `EPSILON=0.0006` aldaketa horrekin ez da modu orokorrean arazoa konpontzen (froga daiteke orain gainezkada `164` gradutako angelu batentzat gertatzen dela).

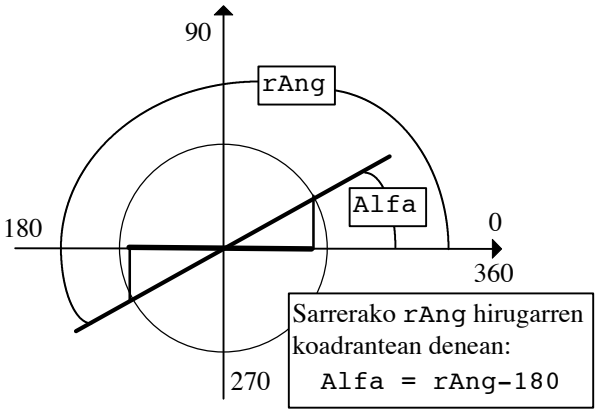
2

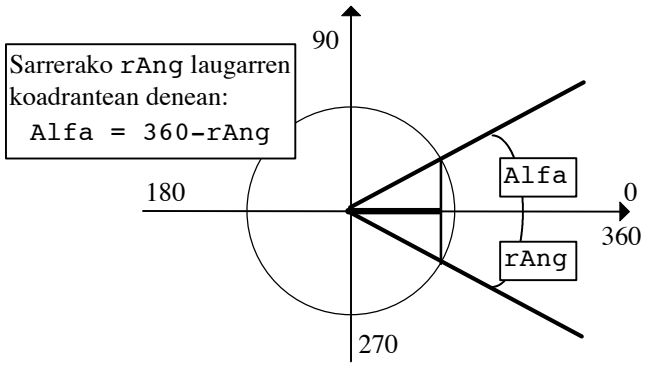
Programatik itxaroten den doitasuna finkoa eta aldaezina bada soluzioa sarrera angeluaren balioa aldatzetik etorriko da. Izan ere, doitasuna `0.0005` izanik gainezkadaren muga `160` graduetan dago, hots, kodrante oso bat eta bigarrenaren zati handi bat. Programaren azken helburua angelu baten kosinua 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"
_
```

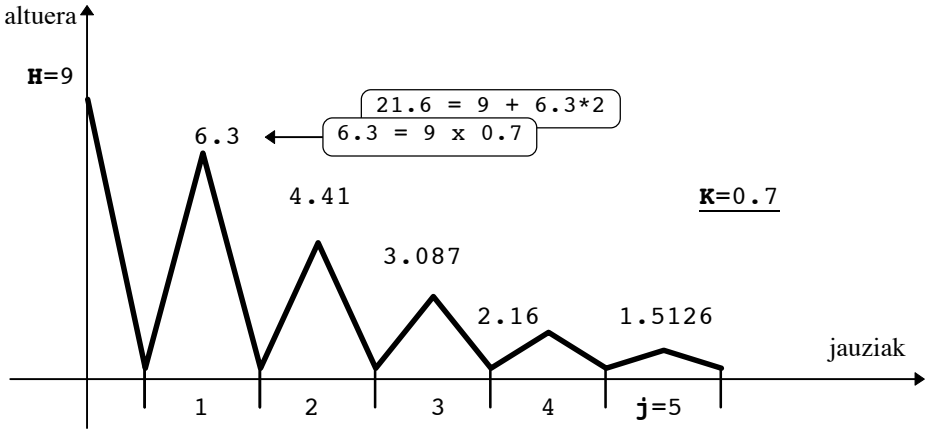
6.6.2.3.3 Pilota jauzika

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

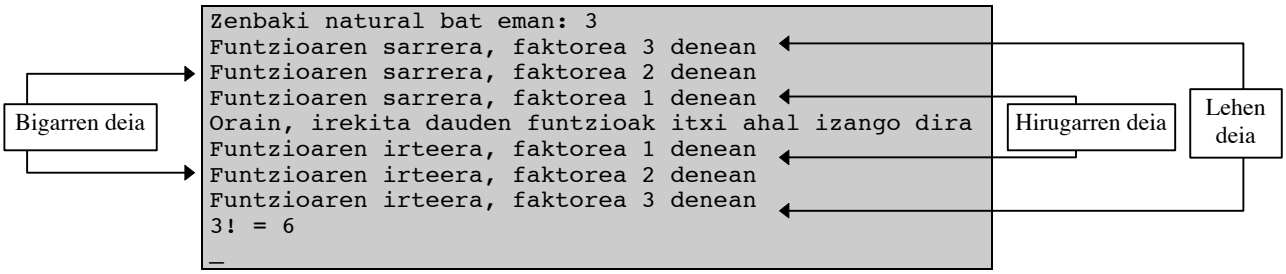
Zenbaki natural baten faktoriala kalkulatzen duen `FaktorialaErrekF()` funtzio errekurtsiboa ondoko programan idatzi da, zeinean zenbaki oso bat teklatuz irakurri ondoren bere faktoriala lortzeko `FOR-DO` bigizta iteratiborik ez dagoen:

```
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.
```

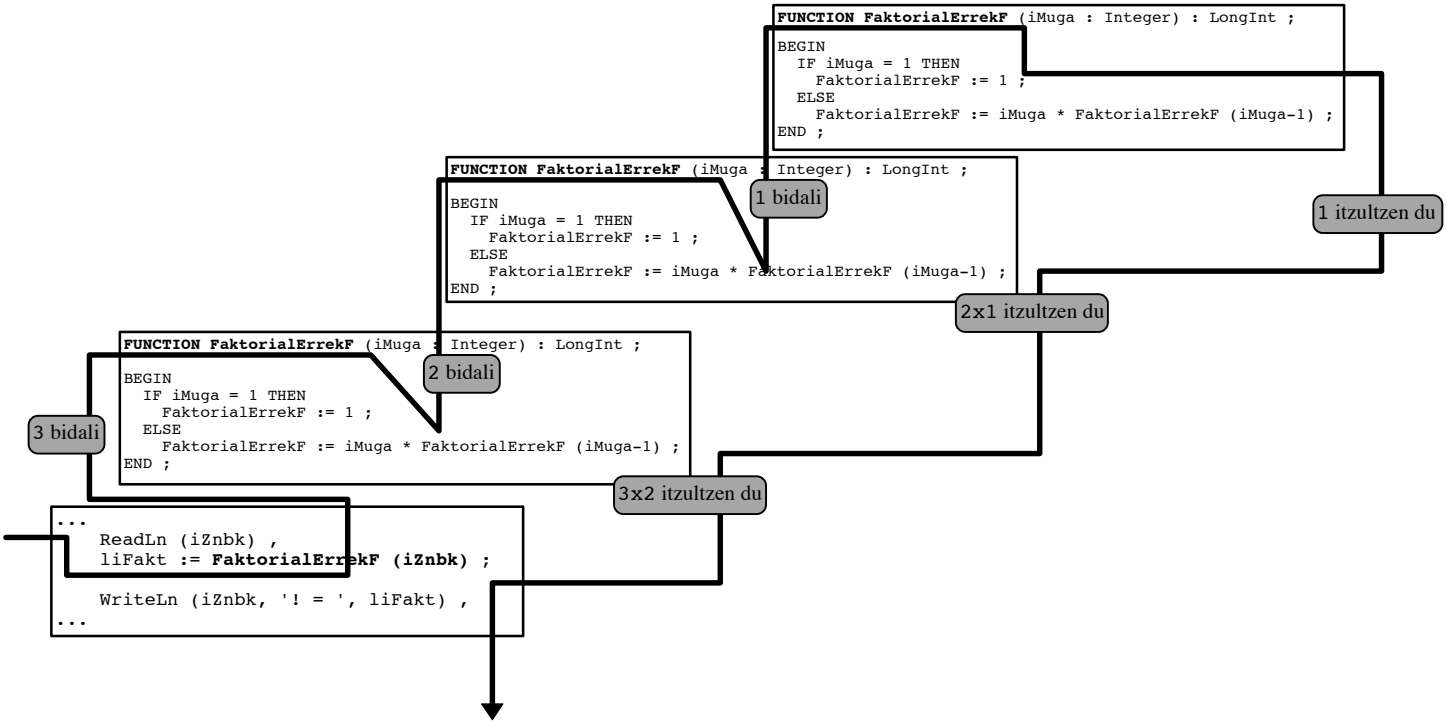
`FaktorialaErrekF()` funtzio errekurtsiboaren barruan dauden laguntza mezuak kenduz hau geratzen zaigu. Eta honela ulertzen da, `iMuga` parametroa `1` denean funtzioak `1` itzuliko du, bestela funtzioak bere buruari berriro deituko dio `iMuga-1` parametro errealarekin:

```
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:

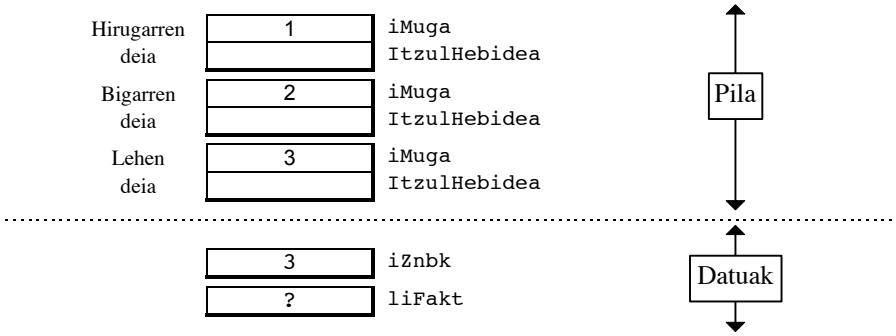


`FaktorialaErrekF()` funtziori hiru aldiz deituko zaio, behin programa nagusian eta ondoren errekurtsiboki beste bi aldiz. Hirugarren deia oraindik prozesatzen ari denean beste biak oraindik amaitu gabe egonen dira, hauxe da programaren fluxuarentzat marraz daitekeen eskema:



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