Konpiladorearen *Front-End*-a

Oier Álvarez eta Gorka Puente

E07

Aurkibidea

Sarrera

Hauek izan dira entrega honetan bete diren helburuak:

Oinarrizko helburuak

Legenda: 1. fasea, 2. fasea, 3. fasea , Garatutako gehigarria, Zuzenketak

- ~~Emandako enuntziatuaren ulermena~~
- ~~Analizatzaile lexikoaren espezifikazioa, diseinua, inplementazioa eta probak.~~
- ~~Analizatzaile sintaktikoaren inplementazioa eta probak.~~
- ~~SZIEaren garapena:~~
  - ~~Itzulpena definitzeko behar diren atributuak aztertu.~~
  - ~~Itzulpen-prozesua definituko duten ekintza semantikoak gehitu.~~
  - ~~Egindako SZIEaren zuzentasuna egiaztatu, beharrezko adibideen gainean probak eginaz.~~
- ~~Azken probak eta dokumentazioa.~~

Aukerazko helburuak

- ~~Itzultzailearen garapena eta proba.~~
- Errore mezu egokiak erabiltzea. Lehenengo errorean amaitzen ez den tratamendua.

Kontrol egitura

- Kontrol-egitura berriaren espezifikazio lexiko, sintaktiko eta semantikoa egin
- SZIE berriaren inplementazioa eta proba.

Adierazpen boolearrak

- Probarako adibide egokiak eskuz ere landu behar dira, zuhaitz dekoratuak eraikiz eta ebaluatuz, sortutako kodea emanez, eta egiaztatu automatikoki emaitza bera lortzen dela.
- SZIEaren hedapena adierazpen boolearren itzulpena tratatzeko.
- Itzultzailearen hedapena aurrekoak inplementatzeko.

Datu-mota boolearra

- Itzulpen numerikoa espezifikatu behar da, horrek dakarzkin aldaketa guztiekin: espezifikazio lexikoa, sintaktikoa, semantikoa eta, bereziki itzulpena (SZIE).
- Gramatikaren eta SZIEren hedapena datu-mota onartzeko.
- SZIE berriaren inplementazioa eta proba.

Azpiprogramen deiak

- Azpiprogramen deien egiaztapen semantiko osoa. Parametroen kopurua, klase egokia eta mota egiaztatzea ezinbestekoa da. Emaitza itzultzen denean ere mota egokia egiaztatu behar da, baita erabilera esparru egokiak kontuan izatea.
- SZIEren hedapena azpiprogramen deien erabilera egokia onartzeko.
- Beste aberasketa posible batzuk):
    - Funtzio deiak adierazpen aritmetikoetan onartzeko aldaketak sartu behar dira. Bakarrik onartu behar dira mota egokiko balioak itzultzen dituzten funtzioak.
    - Balio bat baino gehiago itzultzeko beharrezkoak diren aldaketak egin beharrezkoa den maila guztietan (lexiko, sintaktiko eta semantikoan);
    - Aurretiazko erazagupenak (forward declaration) egin ahal izateko aldaketak gehitzea, kontuan izanik erazagutzen diren unetik aurrera erabil daitezkeela. Erabilera esparru egokiaren egiaztapena egin behar da.

Dimentsio anitzeko taulak

- Dimentsio anitzeko taulen erazagupenaren eta erabileraren espezifikazioa.
- Gramatikaren eta SZIEaren hedapena dimentsio anitzeko taulak onartzeko.
- Taulen erabileraren inplementazioa eta proba.

Murriztapen semantikoak, identifikadoreen erabilera zuzena, etab.

- Murrizpen semantiko guztien definizioa eta dokumentazioa.
- SZIEaren hedapena zuzentasun semantikoa egiaztatzeko.
- Analisi semantikoaren inplementazioa eta proba.

Auto-ebaluazioa

Proiektuaren notari dagokionez[1], momenturarte egindakoarekin 5 bat izan beharko litzateke. Gainditura heltzeko eskatutako minimoak diseinatu, inplementatu eta frogatu dira eta arazorik gabe funtzionatzen dutelarik, nota maximoa merezi dugula erabaki dugu.

Hau da taldekide bakoitzak egindako ahaleginaren deskribapena:

Legenda: etxean, klasean, taldeka, banaka

	1. fasea	2. fasea	3. fasea	Gehigarria(k)
Oier Álvarez	5h + 4h 30’	2h 30’ + 3h	8h’ + 0h	6h’ + 0h
Gorka Puente	5h + 4h 30’	2h 30’ + 3h	-	-

Egindako gehigarria(k)

Praktika honetan adierazpen boolearrak gehitu dira. Horretarako gratikan 3 adierazpen berria sartu dira, boolear bakoitzeko bat (or, and eta not). Proba kasuak diseinatu dira eta proba baten zuhaitza eraiki da. Tamaina handikoa denez eta dokumentu honetan ez zenez sartuko, zuhaitza dokumentazio berean dagoen karpetan sartu da, irudi bezala, jpg formatuan.

[1] Bi taldekideok proiektuaren atal berdinak egin ditugu, beraz alderdi askotan dokumentazioa antzekoa izango da.

Analisi lexikoa

Espezifikazio lexikoa

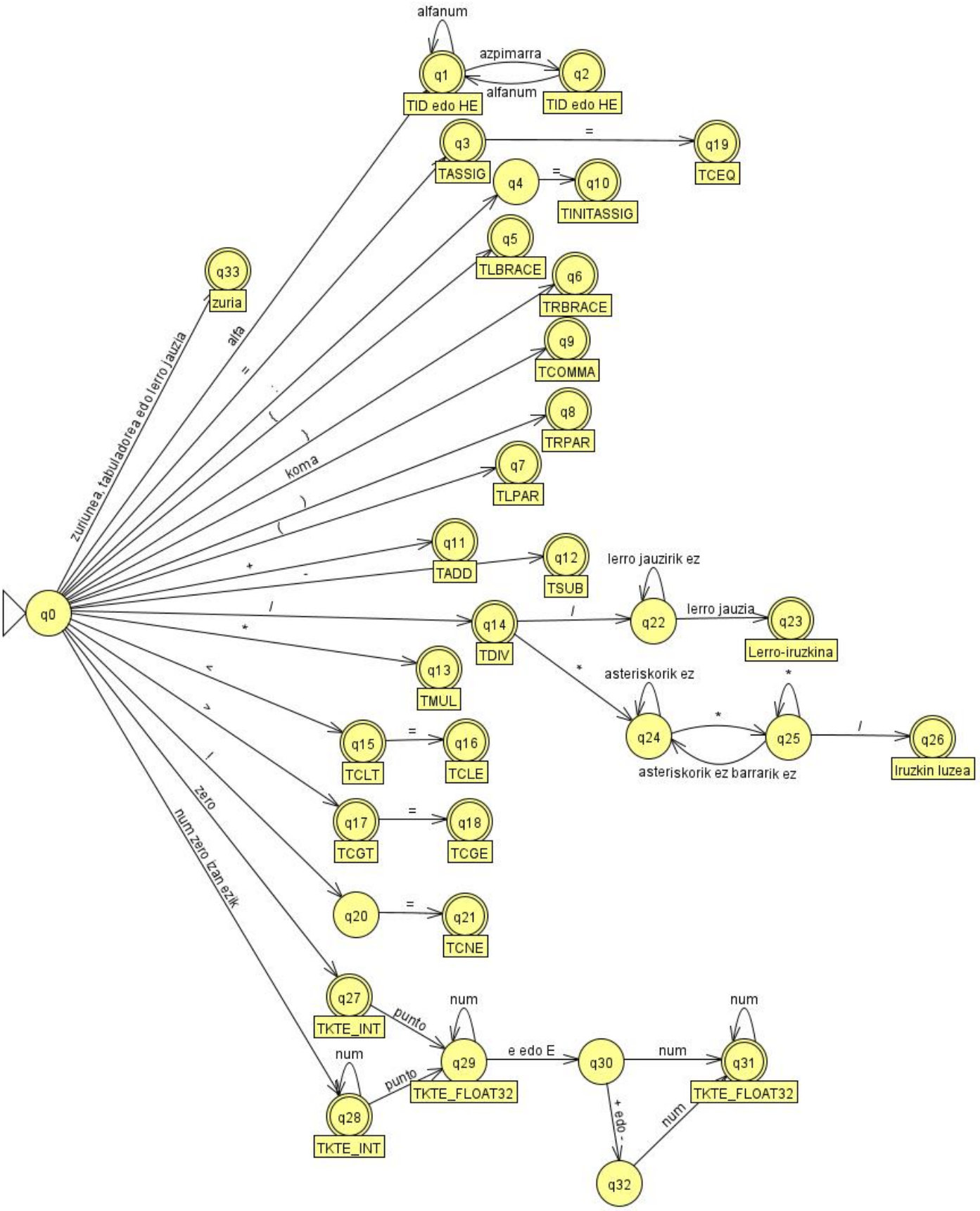
Token-mota	Espezifikazioa	Adibideak	Kontra adibideak
Hitz erreserbatua TCOMMA, TMUL* TADD\+ TSUB\- TDIV\/ TCLT (Token Comparison Less Than)\< TCGT (Greater)\> TCLE (Less Equal)\<= TCGE\>= TCNE (Not Equal)!=	(package |func |main |var |int |float32 |return |if |break |for |continue |read |println |and |or |not)		
TASSIG	=		
TINITASSIG	:=		
TLBRACE	\{		
TRBRACE	\}		
TLPAR	\(		
TRPAR	\)		
TCOMMA	,		
TMUL	*		
TADD	\+		
TSUB	\-		
TDIV	\/		
TCLT	\<		
TCGT	\>		
TCLE	\<=		
TCGE	\>=		

TCNE	!=		

Iruzkin lerroa	\/\/.*\n	//
		// iruzkina
		// hau iruzkina da
			/ /
			/a/
			/ hau ez da iruzkina

Iruzkin luzea	\/*([^*]|*+[^*\/])**+\/	/**/, /* hau iruzkina da*/, /***/	/* itxi gabe / * */
TID	[a-zA-Z]([a-zA-Z0-9]|_[a-zA-Z0-9])*_?	package, kaixo_mundua, kaixo2_mundua2, read, a_	08mendia, kaixo__mundua, pac.kage, _a, 6kaixo
TKTE_FLOAT32	(0|[1-9][0-9]*)\.[0-9]+([eE][-+]?[0-9]+)?	0.0, 10.10, 3.4e10, 5.6E+8, 98.765e-19	00.12, .38, 102., 0.67E, 45.987-3
TKTE_INT	(0|[1-9][0-9]*)	0, 1, 12, 123, 1234	012, 00, 0.85, 78a, 45_

Automata


alfanum
azpimarra
alfanum
q1
TID edo HE
q2
TID edo HE
q3
TASSIG
=
q19
TCEQ
q4
=
q10
TINITASSIG
q5
TLBRACE
q6
TRBRACE
q9
TCOMMA
q8
TRPAR
q7
TLPAR
q33
zuria
zuriunea, tabuladorea edo lerro jauzia
alfa
=
{
}
koma
)
(
+
-
q11
TADD
q12
TSUB
lerro jauzirik ez
q0
/
*
q14
TDIV
/
q22
lerro jauzia
q23
Lerro-iruzkina
*
q13
TMUL
asteriskorik ez
*
*
/
<
>
!
q15
=
q16
TCLT
TCLE
q24
q25
q26
asteriskorik ez barrarik ez
Iruzkin luzea
q17
=
q18
TCGT
TCGE
zero
num zero izan ezik
q20
=
q21
TCNE
q27
TKTE_INT
punto
num
num
q29
e edo E
q30
num
q31
TKTE_FLOAT32
TKTE_FLOAT32
num
punto
q28
TKTE_INT
+ edo -
num
q32

Itzulpen prozesuaren espezifikazioa

Gramatika

programa → **package main** bloke_nag

bloke_nag → bl_eraz azpiprogramak

azpiprogramak → azpiprogramak azpiprograma
| azpiprograma

azpiprograma → **func** izena argumentuak auk_mota bloke

izena → **main** | **id**

bloke → **{** bl_eraz anon_azpi sententzia_zerrenda **}**

anon_azpi → anon_azpi anon
| ξ

anon → **id := func** argumentuak auk_mota bloke

bl_eraz → **var (** eraz_l **)** bl_eraz
| **var** eraz bl_eraz
| ξ

eraz_l → eraz_l eraz
| eraz

eraz → id_zerrenda mota

id_zerrenda → **id** id_zerrendaren_bestea

id_zerrendaren_bestea → **,** **id** id_zerrendaren_bestea
| ξ

mota → **int** | **float32**

auk_mota → mota | ξ

argumentuak → **(** par_zerrenda **)**

par_zerrenda → id_zerrenda mota par_zerrendaren_bestea
| ξ

par_zerrendaren_bestea → **,** id_zerrenda mota par_zerrendaren_bestea
| ξ

sententzia_zerrenda → sententzia_zerrenda sententzia
| sententzia

sententzia → aldagaia **= id (**adi_zerrenda**)**
| aldagaia **=** adierazpena
| **id (**adi_zerrenda**)**
| **if** adierazpena bloke
| **for** bloke
| **for** adierazpena bloke
| **break** adierazpena
| **continue**
| **read (** aldagaia **)**
| **println (** adierazpena **)**
| **return** adierazpena

aldagaia → **id**

adi_zerrenda → adierazpena adi_zerrenda_bestea
| ξ

adi_zerrenda_bestea → **,** adierazpena adi_zerrenda_bestea
| ξ

adierazpena → adierazpena **+** adierazpena
| adierazpena **-** adierazpena
| adierazpena ***** adierazpena
| adierazpena **/** adierazpena
| adierazpena **==** adierazpena
| adierazpena **>** adierazpena
| adierazpena **<** adierazpena
| adierazpena **>=** adierazpena
| adierazpena **<=** adierazpena
| adierazpena **!=** adierazpena
| adierazpena or adierazpena
| adierazpena and adierazpena
| not adierazpena
| aldagaia
| **kte_int**
| **kte_float32**
| **(** adierazpena **)**

Atributuak

Ez-bukaerako / Bukaerako	Atributuen izenak	Informazio mota	Atributu-mota
kte_int	z_izena	karaktere katea.	lexikoa
kte_float32	z_izena	karaktere katea.	lexikoa
id	izena	karaktere katea	lexikoa
adierazpena	izena / true / false	karaktere katea / erreferentzia zerrenda / erreferentzia zerrenda	sintetizatua
izena	izena / main	karaktere katea / erreferentzia	sintetizatua
aldagaia	izena	karaktere katea	sintetizatua
mota	mota	karaketere katea	sintetizatua
id_zerrenda	izenak	karaktere kate zerrenda	sintetizatua
id_zerrendaren_ bestea	izenak	karaktere kate zerrenda	sintetizatua
azpiprogramak	main	erreferentzia	sintetizatua
azpiprograma	main	erreferentzia	sintetizatua

M	erref		erreferentzia	sintetizatua
adi_zerrenda	adi		karaketere kate zerrenda	sintetizatua
adi_zerrenda_bestea	adi		karaketere kate zerrenda	sintetizatua
bloke	break	continue	erreferentzia zerrenda	sintetizatua
sententzia	break	continue	erreferentzia zerrenda	sintetizatua
sententzia_zerrenda	break	continue	erreferentzia zerrenda	sintetizatua

Abstrakzio funtzionalak

Legenda: funtzionalitatea, adibideak, zertarako erabili den

id_berria: *kodea → izena*

aldagai osagarri berri bat itzuliko du (orain arte erabili gabea).

Adierazpen berri bat kalkulatzerakoan, id-aren izena ez errepikatzeko erabili da.

ag_gehitu: *kodea x agindua →*

kodea datu-egituran emandako agindua gehitzen du azken posizioan.

Itzulpena gerta dadin erabili da, emaitza lengoaia idazteko.

lortu_erref: *kodea → kode_erreferentzia*

kodea datu-egituran gehituko den hurrengo aginduaren kode erreferentzia itzuli.

Beranduago erabiliko den erreferentziak gordetzeko edo zuzenean kodearen erreferentziak lortzeko.

ag_osatu: *kodea x kode_erreferentzia_lista x kode_erreferentzia →*

kodea datu-egiturako emandako kode erreferentzien listako jauzi agindu osatugabeei emandako erreferentzia gehitu bukaeran aginduak osatzeko.

Aurretik bete gabe gelditu diren aginduak osatzeko erabili da, hala nola, eragile erlazionalak erabiltzean sortzen diren *goto*-ak betetzeko.

erazagupenak_gehitu: *kodea x mota x id_izenak → kodea*

sarrerako id izenentzako erazagupenak gehitzen ditu hurrengo moduan, parametroen motak (float edo int) eta identifikadoreak izanda,:

mota izena1

mota izena2

…

Erazagupenaren gramatika agertzen denean, funtzio honi deitzen zaio emaitza lengoaian erazagupenak gehi ditzan.

parametroak_gehitu: *kodea x mota x id_izenak → kodea*

sarrerako id izenentzako parametroak idazten ditu hurrengo moduan, parametroaren mota (float edo integer) eta identifikadoreen izenak izanda:

param_mota izena1

param_mota izena2

…

Azpiprograma bat definitzean honek erabiliko dituen parametroak definitzeko erabili da.

parametroekin_deitu: *kodea x parametro_izenak → kodea*

sarrerako parametroen izenentzako parametroen deia idazten du hurrengo moduan:

param izena1

param izena2

…

Azpiprograma baten deia egitean funtzio honi deitzen zaio zein adierazpen pasatuko zaizkion jakinarazteko.

lista_hutsa: *→ lista*

lista huts bat sortu eta itzultzen du.

Zerrenda bat dugunean, baina zerrenda horri baliorik gabe hasieratu behar dugunean

hasi_lista: *osagaia → lista*

lista bat sortu, emandako izena txertatu eta lista berria itzultzen du.

Lista bat erabili behar denean, hau balio batekin hasieratzeko erabili da.

hasieran_gehitu: *lista x osagaia → lista*

sarrerako listan izena hasieran txertatzen du eta lista berria itzultzen du.

Balio bat orden ezpezifiko batean sartu behar denean (kasu honetan balio berria zerrendaren hasieran) erabili da.

bildu: *lista x lista → lista*

lista bat sortu, emandako bi listetako osagaiak txertatu eta lista berria itzultzen du.

Bi listen balioak gora pasatu behar direnean; balioak beste zerrenda batean sartzeko erabili da.

sortuErlazionala: *osagaia x osagaia x osagaia → kodea*

hiru karaktere katee jasota (adierazpenen izenak eta eragilea), adierazpen berri bat sortzen du, haren true eta false listak uneko erreferentzia eta hurrengoarekin hasieratzen ditu. Ondoren kodea, if <adierazpena> goto eta goto idazten ditu.

Adierazpen erlazional bat sortu behar denenean erabiltzen da.

SZIE

programa → **package main** {ag_gehitu(proc main);} bloke_nag {ag_gehitu(halt);}

bloke_nag → bl_eraz M {ag_gehitu(goto);} azpiprogramak {ag_osatu(hasi_lista(M.erref), azpiprogramak.main);}

azpiprogramak → azpiprogramak azpiprograma
{azpiprogramak.main = azpiprogramak1.main + azpiprograma.main;}
| azpiprograma {azpiprogramak.main = azpiprograma.main;}

azpiprograma → **func** izena argumentuak auk_mota bloke
{azpiprograma.main = izena.main;
if (azpiprograma.main == 0) ag_gehitu(endproc || izena.izena);}

izena → **main** { izena.main = lortu_erref();}

| **id**
{izena.main = 0;
izena.izena = id.izena;
ag_gehitu(proc || id.izena); }

bloke → **{** bl_eraz anon_azpi sententzia_zerrenda **}**
{bloke.break = sententzia_zerrenda.break;
bloke.continue = sententzia_zerrenda.continue;}

anon_azpi → anon_azpi anon
| ξ

anon → **id := func** {ag_gehitu(proc || id.izena);} argumentuak auk_mota bloke
{ag_gehitu(endproc || id.izena);}

bl_eraz → **var (** eraz_l **)** bl_eraz
| **var** eraz bl_eraz
| ξ

eraz_l → eraz_l eraz
| eraz

eraz → id_zerrenda mota
{erazagupenak_gehitu(mota.mota, id_zerrenda.izenak);}

id_zerrenda → **id** id_zerrendaren_bestea
{id_zerrenda.izenak = hasieran_gehitu(id_zerrendaren_bestea.izenak, id.izena);}

id_zerrendaren_bestea → **,** **id** id_zerrendaren_bestea
{ id_zerrendaren_bestea.izenak = hasieran_gehitu(id_zerrendaren_bestea1.izenak, id.izena); }
| ξ {id_zerrendaren_bestea.izenak = lista_hutsa();}

mota → **int** {mota.mota = int; }
| **float32** {mota.mota = real; }

```
auk_mota  →     mota | ξ

argumentuak     →     ( par_zerrenda )


par_zerrenda     →     id_zerrenda mota {parametroak_gehitu(mota.mota,
id_zerrenda.izenak);} par_zerrendaren_bestea
                       | ξ

par_zerrendaren_bestea → , id_zerrenda mota {parametroak_gehitu(mota.mota,
id_zerrenda.izenak);} par_zerrendaren_bestea
                              | ξ

sententzia_zerrenda  → sententzia_zerrenda sententzia
            {sententzia_zerrenda.break = bildu(sententzia_zerrenda1.break,
sententententzia.break);
            sententzia_zerrenda.continue = bildu(sententzia_zerrenda1.continue,

            sententententzia.continue);}

                       | sententzia
            {sententzia_zerrenda.break = sententzia.break;
            sententzia_zerrenda.continue = sententzia.continue;}


sententzia →     aldagaia = id (adi_zerrenda)
                       {sententzia.break = lista_hutsa();
                       sententzia.continue = lista_hutsa();
                       parametroekin_deitu(adi_zerrenda.adi);
                       ag_gehitu(aldagaia.izena || := call || id.izena);}

                  | aldagaia = adierazpena
                       {sententzia.break = lista_hutsa();
                       sententzia.continue = lista_hutsa();
                       ag_gehitu(aldagaia.izena || := || adierazpena.izena);}

                  | id (adi_zerrenda)
                            {sententzia.break = lista_hutsa();
                            sententzia.continue = lista_hutsa();
                            parametroekin_deitu(adi_zerrenda.adi);
                            ag_gehitu(call || id.izena);}
                  | if adierazpena M bloke M
                            {sententzia.break = bloke.break;
                            sententzia.continue = bloke.continue;
                            ag_osatu(adierazpena.true, M1.erref);
                            ag_osatu(adierazpena.false, M2.erref);}

                  | for M bloke M
                       {sententzia.break = lista_hutsa();
                       sententzia.continue = lista_hutsa();
                       ag_osatu(bloke.break, M2.erref + 1 );
                       ag_osatu(bloke.continue, M1.erref);
                       ag_gehitu(goto || M1.erref);}

                  | for M adierazpena M bloke M
```

{sententzia.break = lista_hutsa();
sententzia.continue = lista_hutsa();
ag_osatu(bloke.break, M3.erref + 1);
ag_osatu(bloke.continue, M1.erref);
ag_osatu(adierazpena.true = M2.erref);
ag_osatu(adierazpena.false, M3.erref + 1);
ag_gehitu(goto || M1.erref);}

| **break** adierazpena
{sententzia.break = adierazpena.true;
sententzia.continue = lista_hutsa();
ag_osatu(adierazpena.false, lortu_erref());}

| **continue**
{sententzia.break = lista_hutsa();
sententzia.continue = hasi_lista(lortu_erref());
ag_gehitu(goto);}

| **read (** aldagaia **)**
{sententzia.break = lista_hutsa();
sententzia.continue = lista_hutsa();
ag_gehitu(read || aldagaia.izena);}

| **println (** adierazpena **)**
{sententzia.break = lista_hutsa();
sententzia.continue = lista_hutsa();
ag_gehitu(write || adierazpena.izena);
ag_gehitu(writeln);}

| **return** adierazpena
{sententzia.break = lista_hutsa();
sententzia.continue = lista_hutsa();
ag_gehitu(return || adierazpena.izena);}

aldagaia → **id** {aldagaia.izena = id.izena;}

adi_zerrenda → adierazpena adi_zerrenda_bestea
{adi_zerrenda.adi = hasieran_gehitu(adi_zerrenda.bestea.adi, adierazpena.izena); }
| ξ {adi_zerrenda.adi = lista_hutsa();}

adi_zerrenda_bestea → **,** adierazpena adi_zerrenda_bestea

{adi_zerrenda_bestea.adi = hasieran_gehitu(adi_zerrenda.bestea1.adi, adierazpena.izena);}
| ξ {adi_zerrenda_bestea.adi = lista_hutsa();}

adierazpena → adierazpena **+** adierazpena
{adierazpena.izena = id_berria();
adierazpena.true = lista_hutsa();
adierazpena.false = lista_hutsa();
ag_gehitu(adierazpena.izena || := || adierazpena1.izena || + ||
adierazpena2.izena);}
| adierazpena **-** adierazpena
{adierazpena.izena = id_berria();
adierazpena.true = lista_hutsa();
adierazpena.false = lista_hutsa();

```
        ag_gehitu(adierazpena.izena || := || adierazpena1.izena || - ||
                adierazpena2.izena);}
| adierazpena * adierazpena
        {adierazpena.izena = id_berria();
        adierazpena.true = lista_hutsa();
        adierazpena.false = lista_hutsa();
        ag_gehitu(adierazpena.izena || := || adierazpena1.izena || * ||
                adierazpena2.izena);}
| adierazpena / adierazpena
        {adierazpena.izena = id_berria();
        adierazpena.true = lista_hutsa();
        adierazpena.false = lista_hutsa();
        ag_gehitu(adierazpena.izena || := || adierazpena1.izena || / ||
                adierazpena2.izena);}
| adierazpena == adierazpena
        {sortuErlazionala(adierazpena1.izena, "==", adierazpena2.izena);}
| adierazpena > adierazpena
        {sortuErlazionala(adierazpena1.izena, ">", adierazpena2.izena);}
| adierazpena < adierazpena
        {sortuErlazionala(adierazpena1.izena, "<", adierazpena2.izena);}
| adierazpena >= adierazpena
        {sortuErlazionala(adierazpena1.izena, ">=", adierazpena2.izena);}
| adierazpena <= adierazpena
        {sortuErlazionala(adierazpena1.izena, "<=", adierazpena2.izena);}
| adierazpena != adierazpena
        {sortuErlazionala(adierazpena1.izena, "!=", adierazpena2.izena);}

| adierazpena or adierazpena
        {ag_osatu(adierazpena1.false, M.erref);
        adierazpena.izena := "";
        adierazpena.true := bildu(adierazpena1.true, adierazpena2.true);
        adierazpena.false := adierazpena2.false;  }
| adierazpena and adierazpena
        {ag_osatu(adierazpena1.true, M.erref);
        adierazpena.izena := "";
        adierazpena.false := bildu(adierazpena1.false, adierazpena2.false);
        adierazpena.true := adierazpena2.true; }
| not adierazpena
        {adierazpena.false := adierazpena1.true;
         adierazpena.true := E1 .false;
        adierazpena.izena := "";}

| aldagaia
        {adierazpena.izena = aldagaia.izena;
        adierazpena.true = lista_hutsa();
        adierazpena.false = lista_hutsa();}
| kte_int
        {adierazpena.izena = kte_int.z_izena;
        adierazpena.true = lista_hutsa();
        adierazpena.false = lista_hutsa();}
| kte_float32
        {adierazpena.izena = kte_float32.z_izena;
        adierazpena.true = lista_hutsa();
        adierazpena.false = lista_hutsa();}
| ( adierazpena )
```

{adierazpena.izena = adierazpena1.izena;
adierazpena.false = adierazpena1.false;
adierazpena.true = adierazpena1.true;}

M → ξ {M.erref = lortu_erref();}

Proba kasuak

Sartutako proba	Lortutako emaitza

package main
func main() {
  return 0
}
	1: proc main ;
	2: goto 3 ;
	3: return 0 ;
	4: halt ;
	ONDO bukatu da...

package main
func main() int {
  return 0
}
	1: proc main ;
	2: goto 0 ;
	3: proc funtziomota ;
	4: return 0 ;
	5: endproc funtziomota ;
	6: halt ;
	ONDO bukatu da...

package main
var a float32
func main() {
  continue
}
	1: proc main ;
	2: real a ;
	3: goto 4 ;
	4: goto ;
	5: halt ;
	ONDO bukatu da...

package main
var (a, b float32 c, d int)
 	
func main(){
  return 0
}
	1: proc main ;
	2: real a ;
	3: real b ;
	4: int c ;
	5: int d ;
	6: goto 7 ;
	7: return 0 ;
	8: halt ;
	ONDO bukatu da…

package main
func main(par int) {
  return 0
}
	1: proc main ;
	2: goto 0 ;
	3: proc funtzioa_parametroekin ;
	4: param_int par ;
	5: return 0 ;
	6: endproc funtzioa_parametroekin ;
	7: halt ;
	ONDO bukatu da...

package main
func main(par, par2 float32) {
  return 0
}
	1: proc main ;
	2: goto 0 ;
	3: proc funtzioa_parametroekin ;
	4: param_real par ;
	5: param_real par2 ;
	6: return 0 ;
	7: endproc funtzioa_parametroekin ;
	8: halt ;
	ONDO bukatu da…

package main
func main() int {
  return 0
}
	1: proc main ;
	2: goto 0 ;
	3: proc return_funtzioa ;
	4: return 0 ;
	5: endproc return_funtzioa ;
	6: halt ;
	ONDO bukatu da...

package main
func main() {
  funtzioa := func() {
  return 0
  }
  return 0
}
	1: proc main ;
	2: goto 3 ;
	3: proc funtzioa ;
	4: return 0 ;
	5: endproc funtzioa ;
	6: return 0 ;
	7: halt ;
	ONDO bukatu da...

package main
func main() {
  var a, b int
  a = 0
  b = 13
  for a / 3 <= 10 {
  a = a * 2 + (b - 1)
  break a == 10
  }
}
	1: proc main ;
	2: goto 3 ;
	3: int a ;
	4: int b ;
	5: a := 0 ;
	6: b := 13 ;
	7: __t1 := a/3 ;
	8: if __t1 <= 10 goto 10 ;
	9: goto 17 ;
	10: __t2 := a*2 ;
	11: __t3 := b-1 ;
	12: __t4 := __t2+__t3 ;
	13: a := __t4 ;
	14: if a == 10 goto 17 ;
	15: goto 16 ;
	16: goto 7 ;
	17: halt ;
	ONDO bukatu da...

package main
func main() {
 	
  var x int
 	
  x = 10
	1: proc main ;
	2: goto 3 ;
	3: int x ;
	4: x := 10 ;
	5: if x > 9 goto 7 ;
	6: goto 10 ;

```
  for {
    if  x > 9 {
      x = x + 1
      continue
    }

    break x <= 9
  }
}
```

```
7: __t1 := x+1 ;
8: x := __t1 ;
9: goto 5 ;
10: if x <= 9 goto 13 ;
11: goto 12 ;
12: goto 5 ;
13: halt ;
ONDO bukatu da...
```

```
package main
var ( a, b, c int
                d, e float32 )

func prozesatu(x int, y int) { // prozedura da,
ez du baliorik itzultzen
  var lag, bueltak, emaitza int
              /** funtzio anonimoa, habiaratua,
:= eragileak ezkerreko identifikadorea
               ** funtzio bezala erazagutzen
du. Kontuz, ez da esleipen arrunta! */
  gehibat := func(x int) int {
               return x + 1
              } // funtzio anonimoaren amaiera
  kenbat := func(x int) int {
               return x - 1
              } // funtzio anonimoaren amaiera
  lag = y emaitza = x
  if emaitza < 1000 {
               bueltak = 0
               for lag == 0 {
                     emaitza = gehibat(emaitza)
/* funtzio anonimoen deiak tratatzeko */
                     break emaitza > 100000
                     lag = kenbat(lag)          /*
funtzio anonimoen deiak tratatzeko */
                     bueltak = bueltak + 1
               } // for amaiera
               println(emaitza)
              } // if amaiera
              println(bueltak)
} // func prozesatu amaiera

func main () {
              /*** hau iruzkin ** / *
                 lerroanitza da ***/

  read(a) read(b)
              e = 0.400e-2
  d = 1/b
  prozesatu(a,b) /* prozedura deiak
tratatzeko */
```

```
1: proc main ;
2: int a ;
3: int b ;
4: int c ;
5: real d ;
6: real e ;
7: goto 45 ;
8: proc prozesatu ;
9: param_int x ;
10: param_int y ;
11: int lag ;
12: int bueltak ;
13: int emaitza ;
14: proc gehibat ;
15: param_int x ;
16: __t1 := x+1 ;
17: return __t1 ;
18: endproc gehibat ;
19: proc kenbat ;
20: param_int x ;
21: __t2 := x-1 ;
22: return __t2 ;
23: endproc kenbat ;
24: lag := y ;
25: emaitza := x ;
26: if emaitza < 1000 goto 28 ;
27: goto 42 ;
28: bueltak := 0 ;
29: if lag == 0 goto 31 ;
30: goto 40 ;
31: param emaitza ;
32: emaitza := call gehibat ;
33: if emaitza > 100000 goto 40 ;
34: goto 35 ;
35: param lag ;
36: lag := call kenbat ;
37: __t3 := bueltak+1 ;
38: bueltak := __t3 ;
39: goto 29 ;
40: write emaitza ;
41: writeln ;
42: write bueltak ;
```

c = c*(c*d)+e println(c) } // func main amaiera	43: writeln ; 44: endproc prozesatu ; 45: read a ; 46: read b ; 47: e := 0.400e-2 ; 48: __t4 := 1/b ; 49: d := __t4 ; 50: param a ; 51: param b ; 52: call prozesatu ; 53: __t5 := c*d ; 54: __t6 := c*__t5 ; 55: __t7 := __t6+e ; 56: c := __t7 ; 57: write c ; 58: writeln ; 59: halt ; ONDO bukatu da...
Sartutako proba kasu txarra	**Lortutako errorea edota emaitza**

package main
func main() {
var a, b int
a = 0
b = 13
for a / 3 <= 10 {
a = a * 2 + (b - 1)
break //Baldintza gabeko break
}
}
	line 9: syntax error, unexpected TRBRACE at '}'
	GAIZKI bukatu da.
	Errore kopurua: 1

package main
var (a, b float32 c, d) //Mota gabeko esleipena
 	
func main(){
return 0
}
	line 2: syntax error, unexpected TRPAR, expecting RINT or RFLOAT at ')'
	GAIZKI bukatu da.
	Errore kopurua: 1

package main
func main() {
funtzioa = func() { // ':=' egon beharrean '=' dago
return 0
}
return 0
}
	line 3: syntax error, unexpected RFUNC at 'func'
	GAIZKI bukatu da.
	Errore kopurua: 1

package main func main() int float32 { //Bi mota daude	line 2: syntax error, unexpected RFLOAT, expecting TLBRACE at 'float32'

return 0 }	GAIZKI bukatu da. Errore kopurua: 1

package main
func main() {
//sententzia falta da
}
	line 4: syntax error, unexpected TRBRACE at '}'
	GAIZKI bukatu da.
	Errore kopurua: 1

package main
func main() {
 	
var x int
 	
x = 10
 	
for {
if x > 9 {
x = x + 1
continue x == 3 //Baldintza duen continue
}
 	
break x <= 9
}
}
	line 11: syntax error, unexpected TCEQ, expecting TASSIG at '=='
	GAIZKI bukatu da.
	Errore kopurua: 1

package main
func main() {
 	
var x int
 	
x = 10
 	
for {
if { //Baldintza gabeko if
x = x + 1
continue
}
 	
break x <= 9
}
}
	line 9: syntax error, unexpected TLBRACE at '{'
	GAIZKI bukatu da.
	Errore kopurua: 1

package main func main() { var x int x = 10	line 8: syntax error, unexpected TLBRACE at '{' GAIZKI bukatu da. Errore kopurua: 1

{ //For gabeko continue
if x > 9 {
x = x + 1
continue
}
 	
break x <= 9
}
}	

package main
func main (int) { //Parametroa falta da
return 0
}
	line 2: syntax error, unexpected RINT, expecting TRPAR at 'int'
	GAIZKI bukatu da.
	Errore kopurua: 1

package main
func main(par, par2) { //Mota falta da
return 0
}
	line 2: syntax error, unexpected TRPAR, expecting RINT or RFLOAT at ')'
	GAIZKI bukatu da.
	Errore kopurua: 1

package main
func main() int {
return //Return soilik
}
	line 4: syntax error, unexpected TRBRACE at '}'
	GAIZKI bukatu da.
	Errore kopurua: 1

Sartutako proba (Boolearrak)	**Lortutako emaitza**

package main
func main() {
 	
var x int
 	
x = 10
 	
if x > 25 and not x-5 == 7 {
x = x + 1
}
 	
}
	1: proc main ;
	2: goto 3 ;
	3: int x ;
	4: x := 10 ;
	5: if x > 25 goto 7 ;
	6: goto 12 ;
	7: __t1 := x-5 ;
	8: if __t1 == 7 goto 12 ;
	9: goto 10 ;
	10: __t2 := x+1 ;
	11: x := __t2 ;
	12: halt ;
	ONDO bukatu da...

package main
func main() {
 	
var x int
 	
x = 10
 	
if x > 25 and x-5 == 7 {
x = x + 1
	1: proc main ;
	2: goto 3 ;
	3: int x ;
	4: x := 10 ;
	5: if x > 25 goto 7 ;
	6: goto 12 ;
	7: __t1 := x-5 ;
	8: if __t1 == 7 goto 10 ;
	9: goto 12 ;

} }	10: __t2 := x+1 ; 11: x := __t2 ; 12: halt ; ONDO bukatu da...

package main
func main() {
 	
var x int
 	
x = 10
y = 7
 	
if (x > 11 or x < 11) and (y < 10 or y > 9){
x = x + 1
}
}
	1: proc main ;
	2: goto 3 ;
	3: int x ;
	4: x := 10 ;
	5: y := 7 ;
	6: if x > 11 goto 10 ;
	7: goto 8 ;
	8: if x < 11 goto 10 ;
	9: goto 16 ;
	10: if y < 10 goto 14 ;
	11: goto 12 ;
	12: if y > 9 goto 14 ;
	13: goto 16 ;
	14: __t1 := x+1 ;
	15: x := __t1 ;
	16: halt ;
	ONDO bukatu da...

package main
func main() {
 	
var x int
 	
x = 10
y = 7
 	
if (x > 11 and x < 11) or (y < 10 and y > 9){
x = x + 1
}
}
	1: proc main ;
	2: goto 3 ;
	3: int x ;
	4: x := 10 ;
	5: y := 7 ;
	6: if x > 11 goto 8 ;
	7: goto 10 ;
	8: if x < 11 goto 14 ;
	9: goto 10 ;
	10: if y < 10 goto 12 ;
	11: goto 16 ;
	12: if y > 9 goto 14 ;
	13: goto 16 ;
	14: __t1 := x+1 ;
	15: x := __t1 ;
	16: halt ;
	ONDO bukatu da...

package main
func main() {
 	
var x int
 	
x = 10
y = 7
 	
if (x > 11 and x < 11) or (not y < 10 and y > 9){
x = x + 1
}
	1: proc main ;
	2: goto 3 ;
	3: int x ;
	4: x := 10 ;
	5: y := 7 ;
	6: if x > 11 goto 8 ;
	7: goto 10 ;
	8: if x < 11 goto 14 ;
	9: goto 10 ;
	10: if y < 10 goto 12 ;
	11: goto 14 ;

if not (x > 11 or x < 11) and (y < 10 or y > 9){
y = y + 1
}
}
	12: if y > 9 goto 16 ;
	13: goto 14 ;
	14: __t1 := x+1 ;
	15: x := __t1 ;
	16: if x > 11 goto 20 ;
	17: goto 18 ;
	18: if x < 11 goto 20 ;
	19: goto 24 ;
	20: if y < 10 goto 26 ;
	21: goto 22 ;
	22: if y > 9 goto 26 ;
	23: goto 24 ;
	24: __t2 := y+1 ;
	25: y := __t2 ;
	26: halt ;
	ONDO bukatu da...

package main
func main() {
 	
var x int
 	
x = 10
 	
for {
if not x < 11{
x = x + 1
continue
}
 	
break x <= 9
}
}
	
	1: proc main ;
	2: goto 3 ;
	3: int x ;
	4: x := 10 ;
	5: if x < 11 goto 10 ;
	6: goto 7 ;
	7: __t1 := x+1 ;
	8: x := __t1 ;
	9: goto 5 ;
	10: if x <= 9 goto 13 ;
	11: goto 12 ;
	12: goto 5 ;
	13: halt ;
	ONDO bukatu da...

package main
func main() {
 	
var x int
 	
x = 10
 	
for {
if x > 11 or x < 11{
x = x + 1
continue
}
 	
break x <= 9
}
}
	1: proc main ;
	2: goto 3 ;
	3: int x ;
	4: x := 10 ;
	5: if x > 11 goto 9 ;
	6: goto 7 ;
	7: if x < 11 goto 9 ;
	8: goto 12 ;
	9: __t1 := x+1 ;
	10: x := __t1 ;
	11: goto 5 ;
	12: if x <= 9 goto 15 ;
	13: goto 14 ;
	14: goto 5 ;
	15: halt ;
	ONDO bukatu da…

package main func main() { var x int	1: proc main ; 2: goto 3 ; 3: int x ; 4: x := 10 ;

x = 10
y = 7
 	
if (x > 11 and x < 11) or (y < 10 and y > 9){
x = x + 1
}
 	
if (x > 11 or not x < 11) and not (y < 10 or y > 9){
y = y + 1
}
}
	5: y := 7 ;
	6: if x > 11 goto 8 ;
	7: goto 10 ;
	8: if x < 11 goto 14 ;
	9: goto 10 ;
	10: if y < 10 goto 12 ;
	11: goto 16 ;
	12: if y > 9 goto 14 ;
	13: goto 16 ;
	14: __t1 := x+1 ;
	15: x := __t1 ;
	16: if x > 11 goto 20 ;
	17: goto 18 ;
	18: if x < 11 goto 26 ;
	19: goto 20 ;
	20: if y < 10 goto 26 ;
	21: goto 22 ;
	22: if y > 9 goto 26 ;
	23: goto 24 ;
	24: __t2 := y+1 ;
	25: y := __t2 ;
	26: halt ;
	ONDO bukatu da...

Sartutako proba kasu txarrak (Boolearrak)	**Lortutako errorea edota emaitza**

package main
func main() {
 	
var x int
 	
x = 10
 	
if x > 25 and or x+7 {
x = x + 1
}
 	
}
	line 8: syntax error, unexpected ROR at 'or'
	GAIZKI bukatu da.
	Errore kopurua: 1

package main
func main() {
 	
var x int
 	
x = 10
y = 7
 	
if and (x > 11 and x < 11) or (not y < 10 and y > 9){
x = x + 1
}
	line 9: syntax error, unexpected RAND at 'and'
	GAIZKI bukatu da.
	Errore kopurua: 1

if not (x > 11 or x < 11) and (y < 10 or not){
y = y + 1
}
}	

package main
func main() {
 	
var x int
 	
x = 10
 	
if x > 25 not and x-5 == 7 {
x = x + 1
}
 	
}
	line 8: syntax error, unexpected RNOT, expecting TLBRACE at 'not'
	GAIZKI bukatu da.
	Errore kopurua: 1

package main
func main() {
 	
var x int
 	
x = 10
 	
for {
if not x{
x = x + 1
continue
}
 	
break x <= 9
}
}
	1: proc main ;
	2: goto 3 ;
	3: int x ;
	4: x := 10 ;
	5: __t1 := x+1 ;
	6: x := __t1 ;
	7: goto 5 ;
	8: if x <= 9 goto 11 ;
	9: goto 10 ;
	10: goto 5 ;
	11: halt ;
	ONDO bukatu da...