KONPILAZIO PRAKTIKA

UPV/EHU

Javier Aguirre

14-05-2021

Sarrera

Proiektu honetan konpilatzaile baten inplementazioa landu dugu. Horretarako gramatika bat definituta izanda, haren automata eta espezifikazio lexikoa egin dugu. Ondoren, sententzia zerrenda itzulpen eskema eginda goranzko analisia teknika erabiliz konpilatzailearen inplementazioa egin da.

Auto-Ebaluazioa

Proiektua nire kabuz egin izan behar dudanez arazo asko izan ditut bidean zehar. Hasteko SZIEa ondo egitea lan handia izan da. Gero praktika osoa egoki funtzionatzea lan handiagoa ere. Egun osoak egon naiz “segmentation fault” erroreak erreparatzeko. Kasu batean SZIEaren atal bat gaizki zegoela ohartu nintzen, eta beste batean kodea gaizki zegoela. Bestalde boolearrak gehitzeko arazo txiki bat izan nuen “goto”aren helbidea agertzen ez zelako eta horrek ere denbora nahiko kendu zidan.

Klase kanpotik ordu asko sartu izan behar ditut eta egia esanda ordu gehiago pasa ditut erroreak erreparatzen kodetzen baino.

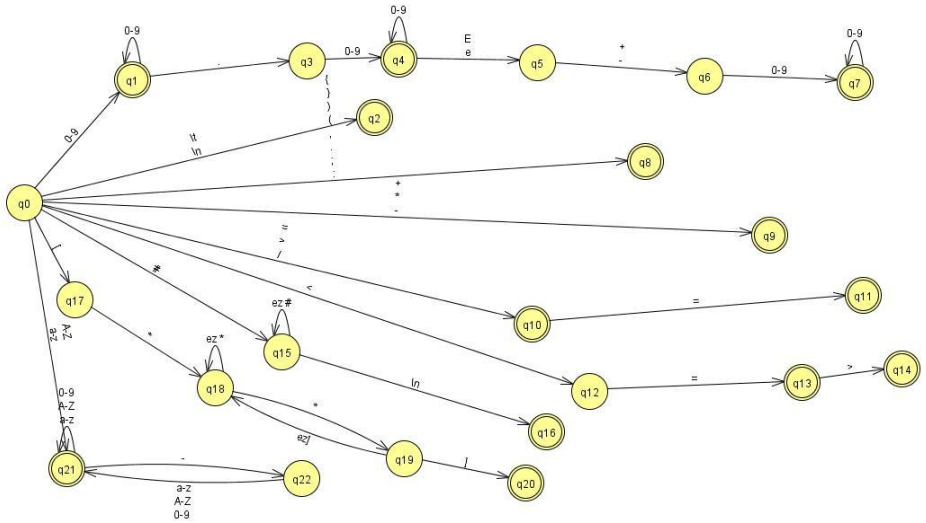
Lan guztia taldekiderik gabe egin behar izan dudanez ordu guztiak nik sartu ditut, eta egia esanda ez dakit zenbat ordu baina arratsalde oso asko eman ditut proiektua egiteko.

Espezifikazio Lexikoa

Token Mota	Sinboloak	Espezifikazio Lexikoak
Konstante Osoak	1, 2, 3, 10, 20, 100...	[0-9]+
Konstante Errealak	10.25, 11.11E+10...	[0-9]+\.[0-9]+([Ee][+|-][0-9]+)?
Hitz Erreserbatuak	prog, int, float, proc, do, until, skip, if, else, read, println, while, forever, exit	prog|int|float|proc|do|until|skip|if|else|read|println|while|forever|exit
Banatzaileak	( ) { } , . ; :	(|)|{|}|,|.|:|;
Identifikatzaileak	Hitz erreserbatuak ez diren hitz guztiak	[a-zA-Z](_?[a-zA-Z]0-9)*
Eragileak	+, -, /, *, =, ==, <, >, <=, >=, <=>, /=	+|-|/|*|=|==|<|>|<=|>=|<=>|/=
Iruzkinak	#edozer [*edozer*]	\#[^#\n]^\n \[*((*[^\]]+)|[^*])**+\]
Zuriune-tabuladore-lerro jauzi	hutsa	[\t\n]

Automata

AUTOMATA: Zoom egin ondo ikusteko:



Itzulpen Prozesuaren Espezifikazioa

Gramatika

programa → **program id**
erazagupenak
azpiprogramen_erazagupena
{ sententzia_zerreeranda **}**

erazagupenak→ mota id_zerrenda **;** erazagupenak
| ξ

id_zerrenda → **id** id_zerrendaren_bestea

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

mota → **int** | **float**

azpiprogramen_erazagupena→ azpiprogramaren_erazagupena azpiprogramen_erazagupena
| ξ

azpiprogramaren_erazagupena →
proc id argumentuak
erazagupenak
azpiprogramen_erazagupena
{ sententzia_zerrenda **}**

argumentuak →**(** par_zerrenda **)**
| ξ

par_zerrenda → mota par_mota id_zerrenda par_zerrendaren_bestea

par_mota → => | <= | <=>

par_zerrendaren_bestea → **;** mota par_mota id_zerrenda par_zerrendaren_bestea

| ξ

sententzia_zerrenda → sententzia sententzia_zerrenda

| ξ

sententzia → aldagaia = adierazpena **;**

| **if** adierazpena **{** sententzia_zerrenda **} ;**

| **while forever {** sententzia_zerrenda **} ;**

| **do {** sententzia_zerrenda **} until** adierazpena **else {** sententzia_zerrenda **} ;**

| **skip if** adierazpena **;**

| **exit ;**

| **read (** aldagaia **) ;**

| **println (** adierazpena **) ;**

aldagaia → **id**

adierazpena → adierazpena + adierazpena

| adierazpena **-** adierazpena

| adierazpena ***** adierazpena

| adierazpena / adierazpena

| adierazpena == adierazpena

| adierazpena **>** adierazpena

| adierazpena **<** adierazpena

| adierazpena >= adierazpena

| adierazpena <= adierazpena

| adierazpena **and** adierazpena

| adierazpena **or** adierazpena

| **not** adierazpena

| adierazpena /= adierazpena

| aldagaia

| **kte_osoko**

| **kte_erreal**

| **(** adierazpena **)**

Atributuak

Lexikoak:

id.izena, kte_osoko.izena, kte_erreal.izena: karaktere-kateak dira identifikadore bezala erabilita

Sintetizatuak:

aldagaia.izena: karaktere-katea non aldagaia gordeko den

id_zerrenda.izenak: identifikadoreen zerrenda

adierazpena.ture, adierazpena.false: osatugabeko jauzien erreferentzia zerrenda

mota.mota: aldagai baten mota erazagutzeko erabiltzen den atributua “integer edo float”.

par_mota.mota: funtzioaren parametroak zer joera duten adierazten du, hau da, sarrerako datuak, irteerako datuak ala sarrera irteerako datuak ote diren “<=”, “>=” edo “<=>”

N.next: dagoeneko aginduaren ondorengo erreferentzia

M.erref: memoriako posizio baten erreferentzia

sententzia.skip: “skip if” jauziaren helbidea gordetzen duen erreferentzia

sententzia.exit: “exit” jauziaren helbidea gordetzen duen erreferentzia

sententzia_zerrenda.skip: “skip if”jauzi guztien zerrenda godetzen duen erreferentzia

sententzia_zerrenda.exit: “exit” jauzi guztien zerrenda godetzen duen erreferentzia

Abstrakzio Funtzionalak

kodea_hasieratu: →kodea

ag_gehitu: kodea x agindua → kodea

emandako agindua kodearen azken posizioan gehitu

gehitu_elem: elementu x lista → lista

elementu bat lista baten bukaeran gehitzen du

erazagupenak_gehitu: kodea x mota x lista → kodea

identifikadore bakoitzerako agindu bat gehitzen du

parametroak_gehitu: mota x lista x par_mota→ kodea

identifikadore bakoitzerako agindu bat gehitzen du

ag_osatu: kodea x kode_erreferentzia_zerrenda x kode_erreferentzia → kodea

erreferentzia-zerrendan ematen diren aginduak (jauziak) emandako erreferentziarekin

osatzen dira

bildu: lista x lista → lista

bi lista eman eta haien “merge”-a bueltatzen du

lortu_erref: kodea → kode_erreferentzia

gehituko den hurrengo aginduaren posizioa itzultzen du

lista_hutsa:

lista hutsa bueltatzne du

hasi_lista: elementua → lista

elementu bakarreko lista bueltatzen du

kodea_idatzi: kodea x fitxategia → fitxategia

id_berria: kodea x fitxategia → izena

aldagai lagungarri berri bat (orain arte erabili gabekoa) itzuliko du

SZIE

programa → **program id**
{ag_gehitu(prog || id.izena);}
erazagupenak
azpiprogramen_erazagupena
{ sententzia_zerrenda **}**
{ag_gehitu(halt);}

erazagupenak→ mota id_zerrenda **;** erazagupenak
{erazagupenak_gehitu(mota.mota, id_zerrenda.izenak);}
| ξ

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

id_zerrendaren_bestea → **,** **id** id_zerrendaren_bestea
{id_zerrendaren_besteak.izenak=gehitu_elem(id.izena, id_zerrendaren_bestea.izenak)}
| ξ

mota → **int**
{mota.mota:=int;}

| **float**

{mota.mota:=float;}

azpiprogramen_erazagupena→ azpiprogramaren_erazagupena azpiprogramen_erazagupena

| ξ

azpiprogramaren_erazagupena →

proc id argumentuak

{ag_gehitu(proc||id.izena);}

erazagupenak

azpiprogramen_erazagupena

{ sententzia_zerrenda **}**

{ag_gehitu(endproc);}

argumentuak →**(** par_zerrenda **)**

| ξ

par_zerrenda → mota par_mota id_zerrenda

{ parametroak_gehitu(mota.mota, id_zerrenda.izenak, par_mota.mota);}

par_zerrendaren_bestea

par_mota → **=>** |

{par_mota.mota := =>;}

<= |

{par_mota.mota := <=;}

<=>

{par_mota.mota := <=>;}

par_zerrendaren_bestea → **;** mota par_mota id_zerrenda

{ parametroak_gehitu(mota.mota, id_zerrenda.izenak, par_mota.mota);}

par_zerrendaren_bestea

| ξ

sententzia_zerrenda → sententzia sententzia_zerrenda

{ sententzia_zerrenda.exit:=bildu(sententzia1.exit, sententzia_zerrenda1.exit);

sententzia_zerrenda.skip:=bildu(sententzia1.skip, sententzia_zerrenda1.skip);

}

| ξ

sententzia → aldagaia = adierazpena ;

{ ag_gehitu(aldagaia.izena || := || adierazpena1.izena);

sententzia.exit:=lista_hutsa();

sententzia.skip:=lista_hutsa();

}

| **if** adierazpena M **{** sententzia_zerrenda **}** **M ;** {

ag_osatu(adierazpena1.true, M_1.erref);

ag_osatu(adierazpena2.false, M_2.erref);

sententzia.skip = sententzia_zerrenda1.skip;

sententzia.exit = sententzia_zerrenda1.exit;

}

| **while forever M{** sententzia_zerrenda **}** N M**;**

{

ag_osatu (N.next, M_1.erref);

ag_osatu(sententzia_zerrenda1.exit, M_2.erref);

sententzia.skip := sententzia_zerrenda1.skip;

sententzia.exit:=lista_hutsa(); }

| **do M {** sententzia_zerrenda **}** **until** M adierazpena **else M {** sententzia_zerrenda **} ; M**

{ ag_osatu(adierazpena1.false, M_1.erref);

ag_osatu(adierazpena1.true, M_3.erref);

ag_osatu(sententia_zerrenda1.exit, M_4.erref);

ag_osatu(sententia_zerrenda2.exit, M_4.erref);

ag_osatu(sententia_zerrenda1.skip, M_2.erref);

sententzia.skip:=sententzia_zerrenda2.skip;}

| **skip if** adierazpena **; M**

{ ag_osatu(adierazpena1.false, M_1.erref);

sententzia.skip:=adierazpena.true;

}

| exit ;

{

sententzia.exit=hasi_lista(lortu_erref());

gehitu(goto);

}

| **read (** aldagaia **) ;**

{ag_osatu(read || aldagaia.izena);

sententzia.skip:=lista_hutsa();

sentenzia.exit:=lista_hutsa();}

| **println (** adierazpena **) ;**

{ag_gehitu(write || adierazpena.izena);

ag_gehitu(writeln);

sententzia.skip:=lista_hutsa();

sentenzia.exit:=lista_hutsa();}

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

adierazpena → adierazpena + adierazpena

{adierazpena.izena=id_berria();

ag_gehitu(adierazpena.izena=adierazpena1.izena||+||adierazpena2.izena);}

| adierazpena – adierazpena

{adierazpena.izena=id_berria();

ag_gehitu(adierazpena.izena=adierazpena1.izena||-||adierazpena2.izena);}

| adierazpena * adierazpena

{adierazpena.izena=id_berria();

ag_gehitu(adierazpena.izena=adierazpena1.izena||*||adierazpena2.izena);}

```
| adierazpena / adierazpena
{adierazpena.izena=id_berria();
 ag_gehitu(adierazpena.izena=adierazpena1.izena||/||adierazpena2.izena);}
| adierazpena == adierazpena
{adierazpena.true = hasi_lista(lortu_erref());
 adierazpena.false = hasi_lista(lortu_erref()+1);
ag_gehitu(if||adierazpena1.izena||=||adierazpena2.izena||goto);
ag_gehitu(goto);}
| adierazpena > adierazpena
{adierazpena.true = hasi_lista(lortu_erref());
 adierazpena.false = hasi_lista(lortu_erref()+1);
ag_gehitu(if||adierazpena1.izena||>||adierazpena2.izena||goto);
ag_gehitu(goto);}
| adierazpena < adierazpena
{adierazpena.true = hasi_lista(lortu_erref());
 adierazpena.false = hasi_lista(lortu_erref()+1);
ag_gehitu(if||adierazpena1.izena||<||adierazpena2.izena||goto);
ag_gehitu(goto);}
| adierazpena >= adierazpena
{adierazpena.true = hasi_lista(lortu_erref());
 adierazpena.false = hasi_lista(lortu_erref()+1);
ag_gehitu(if||adierazpena1.izena||>=||adierazpena2.izena||goto);
ag_gehitu(goto);}
| adierazpena <= adierazpena
{adierazpena.true = hasi_lista(lortu_erref());
 adierazpena.false = hasi_lista(lortu_erref()+1);
ag_gehitu(if||adierazpena1.izena||<=||adierazpena2.izena||goto);
ag_gehitu(goto);}
| adierazpena /= adierazpena
{adierazpena.true = hasi_lista(lortu_erref());
 adierazpena.false = hasi_lista(lortu_erref()+1);
ag_gehitu(if||adierazpena1.izena||/=||adierazpena2.izena||goto);
ag_gehitu(goto);}
```

| adierazpena **or** M adierazpena{

ag_osatu(adierazpena$_1$.false, M.erref);

adierazpena.false:=adierazpena$_2$.false;

adierazpena.true:=bildu(adierazpena$_1$.true, adierazpena$_2$.true);

adierazpena.izena=””

}

| adierazpena **and** M adierazpena{

ag_osatu(adierazpena$_1$.true, M.erref);

adierazpena.true:=adierazpena$_2$.true;

adierazpena.false:=bildu(adierazpena$_1$.false, adierazpena$_2$.false);

adierazpena.izena=””

}

| **not** adierazpena{

adierazpena.true:=adierazpena$_1$.false;;

adierazpena.false:= adierazpena$_1$.true;

adierazpena.izena=””

}

| aldagaia

{adierazpena.izena:=aldagaia.izena);}

| **kte_osoko**

{adierazpena.izena||=||kte_osoko.izena);}

| **kte_erreal**

{adierazpena.izena||=||kte_erreal.izena);}

| **(** adierazpena **)**

{

adierazpena.izena||=||adierazpena1.izena);}

Proba Kasuak

Proba1.in jatorrizko kodea (do-until-else, skip, erazagupenak, read, println)

```
[*skip, do-until-else, aldagai erazagupenak, skip if, read eta println probatzen dira*]
program proba1

  int a, b, c;
  float d, e;

  proc batu(int => x; int => y; int <=> emaitza)

     int lag, bueltak;

  {   lag = y; emaitza = x;

     if emaitza < 1000 {
          bueltak = 0;
          do {
                         emaitza = emaitza + 1;
          skip if emaitza > 100000;
                         lag = lag - 1;
                         bueltak = bueltak + 1;
                  } until lag == 0
    else {
          if emaitza < 0 { exit; };
          println(bueltak);
          }; # do amaiera
          }; # if amaiera
  } # proc amaiera
{ read(a); read(b);
  d = 1/b;
  [*batu(a,b,c);  prozeduren deiak tratatzen dituzten praktiketan soilik *]
  c = c*(c*d)+e;
  println(c);
}
```

Proba1.in ateratako kodea

```
1: prog proba1 ;
2: int c ;
3: int b ;
4: int a ;
5: real e ;
6: real d ;
7: proc batu ;
8: val_int x ;
9: val_int y ;
10: ref_int emaitza ;
11: int bueltak ;
12: int lag ;
13: lag := y ;
14: emaitza := x ;
15: if emaitza < 1000 goto 17 ;
16: goto 33 ;
17: bueltak := 0 ;
18: _t1 := emaitza + 1 ;
19: emaitza := _t1 ;
20: if emaitza > 100000 goto 26 ;
21: goto 22 ;
22: _t2 := lag - 1 ;
23: lag := _t2 ;
24: _t3 := bueltak + 1 ;
25: bueltak := _t3 ;
26: if lag == 0 goto 28 ;
27: goto 18 ;
28: if emaitza < 0 goto 30 ;
29: goto 31 ;
30: goto 33 ;
31: write bueltak ;
32: writeln ;
33: endproc ;
34: read a ;
35: read b ;
36: _t4 := 1 / b ;
37: d := _t4 ;
38: _t5 := c * d ;
39: _t6 := c * _t5 ;
40: _t7 := _t6 + e ;
41: c := _t7 ;
42: write c ;
43: writeln ;
44: halt ;
```

Proba2.in jatorrizko kodea (adierazpen boolearrak)

```
[*or, and eta not adierazpen boolearrak probatzen dira*]
program proba2

  int a, b, c;

  float d, e;

  proc batu(int => x; int => y; int <=> emaitza)

     int lag, bueltak;

  {   lag = y; emaitza = x;
     while forever{bueltak=10;};
     if emaitza < 1000 {
          bueltak = 0;
          do {
          if (bueltak < 0) or (emaitza < 3) {
                              emaitza = emaitza + 1;};
                              if (bueltak < 0) and (emaitza < 3) {
                              emaitza = emaitza + 2;};
                              if not (bueltak<0)  {
                              emaitza = emaitza + 3;};
          skip if emaitza > 100000;
                              lag = lag - 1;
                              bueltak = bueltak + 1;
                    } until lag == 0
    else {
          if emaitza < 0 { exit; };
          println(bueltak);
          }; # do amaiera
          }; # if amaiera
  } # proc amaiera
{ read(a); read(b);
  d = 1/b;
  [*batu(a,b,c);  prozeduren deiak tratatzen dituzten praktiketan soilik *]
  c = c*(c*d)+e;
  println(c);
}
```

Proba2.in ateratako kodea

```
1: prog proba2 ;
2: int c ;
3: int b ;
4: int a ;
5: real e ;
6: real d ;
7: proc batu ;
8: val_int x ;
9: val_int y ;
10: ref_int emaitza ;
11: int bueltak ;
12: int lag ;
13: lag := y ;
14: emaitza := x ;
15: bueltak := 10 ;
16: goto 15 ;
17: if emaitza < 1000 goto 19 ;
18: goto 49 ;
19: bueltak := 0 ;
20: if bueltak < 0 goto 24 ;
21: goto 22 ;
22: if emaitza < 3 goto 24 ;
23: goto 26 ;
24: _t1 := emaitza + 1 ;
25: emaitza := _t1 ;
26: if bueltak < 0 goto 28 ;
27: goto 32 ;
28: if emaitza < 3 goto 30 ;
29: goto 32 ;
30: _t2 := emaitza + 2 ;
31: emaitza := _t2 ;
32: if bueltak < 0 goto 36 ;
33: goto 34 ;
34: _t3 := emaitza + 3 ;
35: emaitza := _t3 ;
36: if emaitza > 100000 goto 42 ;
37: goto 38 ;
38: _t4 := lag - 1 ;
39: lag := _t4 ;
40: _t5 := bueltak + 1 ;
41: bueltak := _t5 ;
42: if lag == 0 goto 44 ;
43: goto 20 ;
44: if emaitza < 0 goto 46 ;
45: goto 47 ;
46: goto 49 ;
47: write bueltak ;
48: writeln ;
49: endproc ;
50: read a ;
51: read b ;
52: _t6 := 1 / b ;
53: d := _t6 ;
54: _t7 := c * d ;
55: _t8 := c * _t7 ;
56: _t9 := _t8 + e ;
57: c := _t9 ;
58: write c ;
59: writeln ;
60: halt ;
```

Proba3.in jatorrizko kodea (exit eta while forever)

```
program proba3

[*exit eta while forever probatzen dira*]

  int a, b, c;

  float d, e;

  proc batu(int => x; int => y; int <=> emaitza)

      int lag, bueltak;

  {lag = y; emaitza = x;} # proc amaiera
{ while forever{ exit; println(d);};
}
```

Proba3.in ateratako kodea

```
1: prog proba3 ;
2: int c ;
3: int b ;
4: int a ;
5: real e ;
6: real d ;
7: proc batu ;
8: val_int x ;
9: val_int y ;
10: ref_int emaitza ;
11: int bueltak ;
12: int lag ;
13: lag := y ;
14: emaitza := x ;
15: endproc ;
16: goto 20 ;
17: write d ;
18: writeln ;
19: goto 16 ;
20: halt ;
```