

Informe Compiladors

José Ruiz Bravo, 43150039S <joseruizbravo@gmail.com>,

Biel Moyà Alcover, 43142617E <bilibiel@gmail.com>,

Álvaro Medina i Ballester, 43176576X <alvaro@comiendolimones.com>

15 de gener de 2010

Resum

Compilador *compilemon* creat amb el llenguatge Ada. Està compost per un subconjunt bàsic d'instruccions en Ada.

1 Anàlisi Lèxica

1.1 Descripció del lèxic: *pk_ulexica.l*

```
1 -- Macros
2 lletra      [A-Za-z]
3 digit      [0-9]
4 separadors [" "\n\b\t\f]
5 car        ({lletra}|{digit})
6 caracter    \' {car} \'
7 car_string  [\040-\041\043-\0176]
8 string      \" ({car_string}|\" \")* \"
9
10
11 %%
12
13
14 -- Paraules clau
15 procedure   {mt_atom(tok_begin_line, tok_begin_col,
16                  yylval); return pc_procedure;}
17
18 begin       {mt_atom(tok_begin_line, tok_begin_col,
19                  yylval); return pc_begin;}
20
```

```
21 while      {mt_atom(tok_begin_line, tok_begin_col,
22              yylval); return pc_while;}
23
24 if          {mt_atom(tok_begin_line, tok_begin_col,
25              yylval); return pc_if;}
26
27 else       {mt_atom(tok_begin_line, tok_begin_col,
28              yylval); return pc_else;}
29
30 end        {mt_atom(tok_begin_line, tok_begin_col,
31              yylval); return pc_end;}
32
33 loop       {mt_atom(tok_begin_line, tok_begin_col,
34              yylval); return pc_loop;}
35
36 constant   {mt_atom(tok_begin_line, tok_begin_col,
37              yylval); return pc_constant;}
38
39 type       {mt_atom(tok_begin_line, tok_begin_col,
40              yylval); return pc_type;}
41
42 array      {mt_atom(tok_begin_line, tok_begin_col,
43              yylval); return pc_array;}
44
45 record     {mt_atom(tok_begin_line, tok_begin_col,
46              yylval); return pc_record;}
47
48 is         {mt_atom(tok_begin_line, tok_begin_col,
49              yylval); return pc_is;}
50
51 then       {mt_atom(tok_begin_line, tok_begin_col,
52              yylval); return pc_then;}
53
54 not        {mt_atom(tok_begin_line, tok_begin_col,
55              yylval); return pc_not;}
56
57 in         {mt_atom(tok_begin_line, tok_begin_col,
58              yylval); return pc_in;}
59
60 out        {mt_atom(tok_begin_line, tok_begin_col,
61              yylval); return pc_out;}
62
63 new        {mt_atom(tok_begin_line, tok_begin_col,
```

```
64         yylval); return pc_new;}
65
66 null      {mt_atom(tok_begin_line, tok_begin_col,
67                 yylval); return pc_null;}
68
69 of        {mt_atom(tok_begin_line, tok_begin_col,
70                 yylval); return pc_of;}
71
72 mod       {mt_atom(tok_begin_line, tok_begin_col,
73                 yylval); return pc_mod;}
74
75 range     {mt_atom(tok_begin_line, tok_begin_col,
76                 yylval); return pc_range;}
77
78 and       {mt_atom(tok_begin_line, tok_begin_col,
79                 yylval); return pc_or;}
80
81 or        {mt_atom(tok_begin_line, tok_begin_col,
82                 yylval); return pc_and;}
83
84 --Symbols
85 " := "    {mt_atom(tok_begin_line, tok_begin_col,
86                 yylval); return s_assignacio;}
87
88 " : "     {mt_atom(tok_begin_line, tok_begin_col,
89                 yylval); return s_dospunts;}
90
91 " ; "     {mt_atom(tok_begin_line, tok_begin_col,
92                 yylval); return s_final;}
93
94 " , "     {mt_atom(tok_begin_line, tok_begin_col,
95                 yylval); return s_coma;}
96
97 " ( "     {mt_atom(tok_begin_line, tok_begin_col,
98                 yylval); return s_parentesiobert;}
99
100 ") "     {mt_atom(tok_begin_line, tok_begin_col,
101                 yylval); return s_parentesitancat;}
102
103 " . . "   {mt_atom(tok_begin_line, tok_begin_col,
104                 yylval); return s_puntsrang;}
105
106 " . "     {mt_atom(tok_begin_line, tok_begin_col,
```

```
107         yylval); return s_puntrec;}
108
109 --Operadors
110 "<"      {mt_atom(tok_begin_line, tok_begin_col,
111                yylval); return op_menor;}
112
113 "<="     {mt_atom(tok_begin_line, tok_begin_col,
114                yylval); return op_menorigual;}
115
116 ">="     {mt_atom(tok_begin_line, tok_begin_col,
117                yylval); return op_majorigual;}
118
119 ">"      {mt_atom(tok_begin_line, tok_begin_col,
120                yylval); return op_major;}
121
122 "="      {mt_atom(tok_begin_line, tok_begin_col,
123                yylval); return op_igual;}
124
125 "/"      {mt_atom(tok_begin_line, tok_begin_col,
126                yylval); return op_distint;}
127
128 "+"      {mt_atom(tok_begin_line, tok_begin_col,
129                yylval); return op_suma;}
130
131 "-"      {mt_atom(tok_begin_line, tok_begin_col,
132                yylval); return op_resta;}
133
134 "*"      {mt_atom(tok_begin_line, tok_begin_col,
135                yylval); return op_multiplicacio;}
136
137 "/"      {mt_atom(tok_begin_line, tok_begin_col,
138                yylval); return op_divisio;}
139
140 --Digit
141 {digit}+ {mt_numero(tok_begin_line, tok_begin_col,
142                yytext, yylval); return const;}
143
144 --Lletra
145 {character} {mt_caracter(tok_begin_line, tok_begin_col,
146                yytext, yylval); return const;}
147
148 --String  mirar a les declaracions
149 {string}  {mt_string(tok_begin_line, tok_begin_col,
```

```
150         yytext, yylval); return const;}
151
152 --Identificador
153 {lletra}({car}|("_"{car}))*
154         {mt_identificador(tok_begin_line, tok_begin_col,
155         yytext, yylval); return id;}
156
157 --Comentaris
158 "--" [^\n]*\n      {null;}
159
160 --Separadors
161 {separadors}      {null;}
162
163 --Error
164 .      {return error;}
165
166
167 %%
168
169
170 with      decls.d_taula_de_noms ,
171           pk_usintactica_tokens ,
172           decls.d_atribut;
173
174 use      decls.d_taula_de_noms ,
175           pk_usintactica_tokens ,
176           decls.d_atribut;
177
178 package u_lexica is
179
180         function YYLex return token;
181
182 end u_lexica;
183
184
185
186 package body u_lexica is
187
188 ##
189
190
191 end u_lexica;
```

1.2 Taula de noms

1.2.1 Fitxer *decls-d_taula_de_noms.ads*

```
1  -- -----
2  --   Paquet de declaracions de la taula de noms
3  -- -----
4  --   Versio   :    0.2
5  --   Autors   :    Jose Ruiz Bravo
6  --               Biel Moya Alcover
7  --               Alvaro Medina Ballester
8  -- -----
9  --   Especificacio de l'estructura necessaria
10 -- per el maneig de la taula de noms i dels metodes
11 -- per tractar-la.
12 --
13 -- -----
14
15 WITH     decls.dgenerals;
16
17 USE       decls.dgenerals;
18
19
20 PACKAGE decls.d_taula_de_noms IS
21
22     --pragma pure;
23
24     -- Excepcions
25     E_Tids_Plana : EXCEPTION;
26     E_Tcar_Plana : EXCEPTION;
27
28     TYPE taula_de_noms IS LIMITED PRIVATE;
29
30     TYPE id_nom IS NEW integer
31         RANGE 0 .. max_id;
32
33     TYPE rang_dispersio IS NEW integer
34         RANGE 0 .. max_id;
35
36     TYPE rang_tcar IS NEW integer
37         RANGE 0 .. (long_num_ident*max_id);
38
39     -- Valor nul per al tipus id_nom
40     id_nul : CONSTANT id_nom := 0;
```

```
41
42     PROCEDURE tbuida
43         (tn : OUT taula_de_noms);
44
45     PROCEDURE posa_id
46         (tn : IN OUT taula_de_noms;
47          idn : OUT id_nom;
48          nom : IN string);
49
50     PROCEDURE posa_tc
51         (tn : IN OUT taula_de_noms;
52          nom : IN string);
53
54     PROCEDURE posa_str
55         (tn : IN OUT taula_de_noms;
56          ids : OUT rang_tcar;
57          s : IN string);
58
59     FUNCTION cons_nom
60         (tn : IN taula_de_noms;
61          idn : IN id_nom)
62         RETURN string;
63
64     FUNCTION cons_str
65         (tn : IN taula_de_noms;
66          ids : IN rang_tcar)
67         RETURN string;
68
69     FUNCTION fdisp_tn
70         (nom : IN string)
71         RETURN rang_dispersio;
72
73
74 PRIVATE
75
76     TYPE taula_dispersio IS ARRAY
77         (rang_dispersio) OF id_nom;
78
79     TYPE t_identificador IS RECORD
80         pos_tcar : rang_tcar;
81         seguent : id_nom;
82         long_paraula : Natural;
83 END RECORD;
```

```
84
85     TYPE taula_identificadors IS ARRAY
86       (1 .. id_nom'Last) OF t_identificador;
87
88     TYPE taula_characters IS ARRAY
89       (rang_tcar) OF character;
90
91     TYPE taula_de_noms IS RECORD
92       td : taula_dispersio;
93       tid : taula_identificadors;
94       tc : taula_characters;
95       nid : id_nom;
96       ncar : rang_tcar;
97     END RECORD;
98
99
100 END decls.d_taula_de_noms;
```


1.2.2 Fitxer *decls-d_taula_de_noms.adb*

```
1  -- -----
2  --   Paquet de declaracions de la taula de noms
3  -- -----
4  --   Versio   :    0.3
5  --   Autors   :    Jose Ruiz Bravo
6  --               Biel Moya Alcover
7  --               Alvaro Medina Ballester
8  -- -----
9  --   Implementacio dels procediments per al
10 -- tractament de la taula de noms:
11 --
12 --           - Buidat de la taula
13 --           - Insercio
14 --           - Insercio d'strings
15 --           - Consulta
16 --           - Funcio de hash
17 --
18 -- -----
19
20
21 PACKAGE BODY decls.d_taula_de_noms IS
22
23     -- Donam els valors per defecte de cada camp.
24     PROCEDURE tbuida
25         (tn : OUT taula_de_noms) IS
26
27     BEGIN
28         FOR i IN tn.td'RANGE LOOP
29             tn.td(i) := id_nul;
30         END LOOP;
31
32         tn.nid := 1;
33         tn.ncar := 1;
34         tn.tid(1).seguent := id_nul;
35     END tbuida;
36
37
38
39     PROCEDURE posa_id
40         (tn : IN OUT taula_de_noms;
41          idn : OUT id_nom;
```

```
42     nom : IN string) IS
43
44     -- Variable per el valor de la funcio de dispersio.
45     p_tid : rang_dispersio;
46
47     -- Index per recorre la taula d'identificadors.
48     idx : id_nom;
49     Trobat : boolean;
50
51     p : taula_identificadors RENAMES tn.tid;
52     d : taula_dispersio RENAMES tn.td;
53
54 BEGIN
55     p_tid := fdisp_tn(nom);
56     Idx := d(P_Tid);
57     Trobat := False;
58
59     WHILE NOT Trobat AND Idx/=Id_Nul LOOP
60         IF (Nom = Cons_Nom(Tn, Idx)) THEN
61             Trobat := True;
62         ELSE
63             Idx := p(Idx).Seguent;
64         END IF;
65     END LOOP;
66
67     IF NOT Trobat THEN
68         Idn := Tn.Nid;
69         p(idn).Pos_Tcar := Tn.Ncar;
70         p(idn).Seguent := d(P_Tid);
71         p(idn).Long_Paraula := Nom'Length;
72
73         d(P_Tid) := Tn.Nid;
74
75         posa_tc(tn, nom);
76     END IF;
77
78 END posa_id;
79
80
81
82 PROCEDURE posa_tc
83     (tn : IN OUT taula_de_noms;
84     nom : IN string) IS
```

```
85
86 BEGIN
87     tn.nid := tn.nid + 1;
88
89     FOR i IN 1 .. nom'Length LOOP
90         tn.tc(tn.ncar) := nom(i);
91         tn.ncar := tn.ncar + 1;
92     END LOOP;
93
94 END posa_tc;
95
96
97
98 PROCEDURE posa_str
99     (tn : IN OUT taula_de_noms;
100     ids : OUT rang_tcar;
101     s : IN string) IS
102
103     -- Index per recorre la taula de caracters.
104     jdx : rang_tcar;
105     long : rang_tcar RENAMES tn.ncar;
106
107 BEGIN
108     -- Excepcio per a controlar tc plena
109     IF (long + s'Length) > rang_tcar'Last THEN
110         RAISE E_Tcar_Plena;
111     END IF;
112
113     -- Omplim la taula de caracters, desde la primera
114     -- posicio lliure 'tn.ncar' renombrat a 'long'.
115     jdx := long;
116     ids := long;
117
118     FOR i IN 1..s'Length LOOP
119         tn.tc(jdx) := s(i);
120         jdx := jdx + 1;
121     END LOOP;
122
123     long := jdx + 1;
124     tn.tc(jdx) := Ascii.nul;
125
126 END posa_str;
127
```

```
128
129
130 FUNCTION cons_nom
131   (tn : IN taula_de_noms;
132    idn : IN id_nom)
133   RETURN string IS
134
135   It1, It2 : Rang_Tcar;
136
137 BEGIN
138   It1 := Tn.Tid(Idn).Pos_Tcar;
139   It2 := Rang_Tcar(Tn.Tid(Idn).Long_Paraula);
140   It2 := It2 + It1 - 1;
141
142   RETURN String(Tn.Tc(it1 .. it2));
143
144 END cons_nom;
145
146
147
148 FUNCTION cons_str
149   (tn : IN taula_de_noms;
150    ids : IN rang_tcar)
151   RETURN string IS
152
153   idx : rang_tcar;
154
155 BEGIN
156   idx := ids;
157   WHILE (tn.tc(idx) /= Ascii.nul) LOOP
158     idx := idx+1;
159   END LOOP;
160
161   RETURN string(tn.tc(ids..idx-1));
162
163 END cons_str;
164
165
166 FUNCTION fdisp_tn
167   (nom : IN string)
168   RETURN rang_dispersio IS
169
170   a : ARRAY (nom'RANGE) OF integer;
```

```
171     r : ARRAY (1..2*nom'Last) OF integer;
172
173     k, c, m, n : integer;
174
175     base : CONSTANT Integer :=
176         Character'Pos(Character'Last)+1;
177
178     BEGIN
179         n := nom'Last;
180         m := nom'Length;
181
182         FOR i IN 1..n LOOP
183             a(i) := character'Pos(nom(i));
184         END LOOP;
185
186         FOR i IN 1..2*n LOOP
187             r(i) := 0;
188         END LOOP;
189
190         FOR i IN 1..n LOOP
191             c := 0; k := i - 1;
192             FOR j IN 1..n LOOP
193                 c := c + r(k+j) + a(i) + a(j);
194                 r(k+j) := c MOD base;
195                 c := c/base;
196             END LOOP;
197             r(k+n+1) := r(k+n+1) + c;
198         END LOOP;
199
200         c := (r(n+1) * base + r(n)) MOD (max_id);
201
202         RETURN rang_dispersio(c);
203
204     END fdisp_tn;
205
206
207 END decls.d_taula_de_noms;
```

1.3 Atributs

1.3.1 Fitxer *decls-d_atribut.ads*

```
1  -- -----
2  --   Paquet de procediments dels atributs
3  -- -----
4  --   Versio   :    0.2
5  --   Autors   :    Jose Ruiz Bravo
6  --               Biel Moya Alcover
7  --               Alvaro Medina Ballester
8  -- -----
9  --       En aquest fitxer tenim implementats les
10 -- assignacions de cada tipus de token al tipus
11 -- atribut que li correspon. Cal destacar
12 -- l'utilització de la taula de noms en els
13 -- casos d'identificadors i strings.
14 --
15 -- -----
16
17 WITH     decls.Dgenerals,
18          decls.D_Taula_De_Noms;
19
20 USE      decls.Dgenerals,
21          decls.D_Taula_De_Noms;
22
23
24 PACKAGE decls.d_atribut IS
25
26     TYPE tipus_atribut IS
27         (atom,
28          a_ident,
29          a_lit);
30
31     TYPE valor IS NEW integer;
32
33     TYPE atribut (t : tipus_atribut := atom) IS RECORD
34         lin, col : natural;
35         CASE t IS
36             WHEN atom           => NULL;
37             WHEN a_ident        => idn : id_nom;
38             WHEN a_lit          => val : valor;
39         END CASE;
40     END RECORD;
```

```
41
42
43     PROCEDURE mt_atom
44         (l, c : IN natural;
45          a : OUT atribut);
46
47     PROCEDURE mt_identificador
48         (l, c : IN natural;
49          s : IN string;
50          a : OUT atribut);
51
52     PROCEDURE mt_string
53         (l, c : IN natural;
54          s : IN string;
55          a : OUT atribut);
56
57     PROCEDURE mt_caracter
58         (l, c : IN natural;
59          car : IN string;
60          a : OUT atribut);
61
62     PROCEDURE mt_numero
63         (l, c : IN natural;
64          s : IN string;
65          a : OUT atribut);
66
67     --Provisional
68     Tn : Taula_De_Noms;
69
70
71 END decls.d_atribut;
```

1.3.2 Fitxer *decls-d_atribut.adb*

```
1
2 -----
3 -- Paquet de procediments dels atributs
4 -----
5 -- Versio   :   0.1
6 -- Autors   :   Jose Ruiz Bravo
7 --           Biel Moya Alcover
8 --           Alvaro Medina Ballester
9 -----
10 --      En aquest fitxer tenim implementats les
11 --      assignacions de cada tipus de token al tipus
12 --      atribut que li correspon. Cal destacar
13 --      l'utilització de la taula de noms en els
14 --      casos d'identificadors i strings.
15 --
16 -----
17
18 WITH      U_Lexica;
19
20 USE      U_Lexica;
21
22
23 PACKAGE BODY decls.d_atribut IS
24
25     PROCEDURE mt_atom
26     (l, c : IN natural;
27      a : OUT atribut) IS
28     BEGIN
29         a := (atom, l, c);
30     END mt_atom;
31
32
33     PROCEDURE mt_identificador
34     (l, c : IN natural;
35      s : IN string;
36      a : OUT atribut) IS
37         id : id_nom;
38     BEGIN
39         id := id_nul;
40         posa_id(tn, id, s);
41         a := (a_ident, l, c, id);
```



```
42     END mt_identificador;  
43  
44  
45     PROCEDURE mt_string  
46         (l, c : IN natural;  
47          s : IN string;  
48          a : OUT atribut) IS  
49         id : rang_tcar;  
50     BEGIN  
51         posa_str(tn, id, s);  
52         a := (a_lit, l, c, valor(id));  
53     END mt_string;  
54  
55  
56     PROCEDURE mt_caracter  
57         (l, c : IN natural;  
58          car : IN string;  
59          a : OUT atribut) IS  
60     BEGIN  
61         a := (a_lit, l, c, valor(car'First+1));  
62     END mt_caracter;  
63  
64  
65     PROCEDURE mt_numero  
66         (l, c : IN natural;  
67          s : IN string;  
68          a : OUT atribut) IS  
69     BEGIN  
70         a := (a_lit, l, c, valor(Integer'value(s)));  
71     END mt_numero;  
72  
73  
74     END decls.d_atribut;
```


$dec_constant \rightarrow$ identificador : **PC_CONSTANT** identificador := *valor*;

valor \rightarrow *lit*
| $- lit$

– Manual d'usuari tipus

$dec_tipus \rightarrow dec_subrang$
| $dec_registre$
| $dec_coleccio$

$dec_subrang \rightarrow$ **PC_TYPE** identificador **PC_IS PC_NEW** identificador
PC_RANGE *valor* .. *valor*;

$dec_registre \rightarrow$ **PC_TYPE** identificador **PC_IS PC_RECORD**
ldc
PC_END PC_RECORD;

ldc $\rightarrow ldc\ dc$
| dc

dc \rightarrow identificador : identificador;

– Tipus colecció (*array*)

$dec_coleccio \rightarrow$ **PC_TYPE** identificador **PC_IS PC_ARRAY**
(*lid*) **PC_OF** identificador;

lid $\rightarrow lid$, identificador
| identificador

– Bloc d'instruccions

bloc $\rightarrow bloc\ sent$
| $sent$

sent $\rightarrow sassig$
| $scond$
| $srep$
| $crida_proc$

sassig $\rightarrow referencia := expressio$;

scond → **PC_IF** *expressio* **PC_THEN**
 bloc
 PC_END PC_IF;
 | **PC_IF** *expressio* **PC_THEN**
 bloc
 PC_ELSE
 bloc
 PC_END PC_IF;

srep → **PC_WHILE** *expressio* **PC_LOOP**
 bloc
 PC_END PC_LOOP;

crida_proc → *referencia*;

referencia → identificador
 | *referencia*.identificador
 | *referencia* (*prparam*)

prparam → *expressio*
 | *expressio*, *prparam*

expressio → *expressio* + *expressio*
 | *expressio* − *expressio*
 | *expressio* * *expressio*
 | *expressio* / *expressio*
 | *expressio* **PC_MOD** *expressio*
 | *expressio* > *expressio*
 | *expressio* < *expressio*
 | *expressio* ≥ *expressio*
 | *expressio* ≤ *expressio*
 | *expressio* ≠ *expressio*
 | *expressio* = *expressio*
 | − *expressio*
 | *expressio* && *expressio*
 | *expressio* || *expressio*
 | **PC_NOT** *expressio*
 | (*expressio*)
 | *referencia*
 | *lit*

2.2 Especificació *pk_usintactica.y*

```
1 -- Token
2 %token pc_procedure
3 %token pc_begin
4 %token pc_while
5 %token pc_if
6 %token pc_else
7 %token pc_end
8 %token pc_loop
9 %token pc_constant
10 %token pc_type
11 %token pc_array
12 %token pc_record
13 %token pc_is
14 %token pc_then
15 %token pc_not
16 %token pc_in
17 %token pc_out
18 %token pc_new
19 %token pc_null
20 %token pc_of
21 %token pc_mod
22 %token pc_range
23 %token pc_or
24 %token pc_and
25 %token s_assignacio
26 %token s_dospunts
27 %token s_final
28 %token s_coma
29 %token s_parentesiobert
30 %token s_parentesitancat
31 %token s_puntsrang
32 %token s_puntrec
33 %token op_menor
34 %token op_menorigual
35 %token op_majorigual
36 %token op_major
37 %token op_igual
38 %token op_distint
39 %token op_suma
40 %token op_resta
41 %token op_multiplicacio
```

```
42 %token op_divisio
43 %token id
44 %token const -- Per declarar constants (digit, lletra i string)
45               -- MIRAR d_token.ads: pc_do Quitar??? cadena???
46
47 --Precedencia
48 %left pc_or
49 %left pc_and
50 %nonassoc op_menor op_menorigual op_majorigual op_major op_igual op_disti
51 %left op_suma
52 %left op_resta
53 %left op_multiplicacio op_divisio pc_mod
54 %left pc_not
55 %left menys_unitari
56
57
58
59 --Definicio del tipus atribut
60 %WITH decls.d_atribut
61 {
62     SUBTYPE yystype IS decls.d_atribut.atribut;
63 }
64
65
66 %%
67
68
69 --Produccions de la gramatica del llenguatge
70 programa:
71     M1 dec_procediment
72     ;
73
74 M1:
75     ;
76
77 dec_procediment:
78     pc_procedure encap pc_is
79         declaracions
80     pc_begin
81         bloc
82     pc_end id s_final
83     ;
84
```

```
85
86 encap:
87     id
88     |
89     pencap s_parentesitancat
90     ;
91
92 pencap:
93     pencap s_final param
94     |
95     id s_parentesiobert param
96     ;
97
98 param:
99     id s_dospunts mode id
100    ;
101
102
103 mode:
104     pc_in
105     |
106     pc_out
107     |
108     pc_in pc_out
109     ;
110
111 declaracions:
112     declaracions declaracio
113     |
114     ;
115
116
117 -- DECLARACIONS
118 declaracio:
119     dec_var s_final
120     |
121     dec_constant s_final
122     |
123     dec_tipus s_final
124     |
125     dec_procediment --s_final
126     ;
127
```

```
128 dec_var:
129     id c_decl_var
130     ;
131
132 c_decl_var:
133     s_dospunts id c_decl_ass --s_final
134     |
135     s_coma id c_decl_var
136     ;
137
138 dec_constant:
139     id s_dospunts pc_constant id s_assignacio const --s_final -- ESTE
140     ;
141
142 c_decl_ass:
143     s_assignacio const
144     |
145     ;
146
147 -- TIPUS
148 dec_tipus:
149     decl_coleccio
150     |
151     decl_registre
152     |
153     decl_subrang
154     ;
155
156
157 -- TIPUS SUBRANG
158 decl_subrang:
159     pc_type id pc_is pc_new id pc_range limit s_puntsrang limit --s_f
160     ;
161
162 limit:
163     const
164     |
165     id
166     ;
167
168
169 -- TIPUS REGISTRE
170 decl_registre:
```



```
171         p_dregistre pc_end pc_record
172     ;
173
174 p_dregistre:
175     p_dregistre id s_dospunts id s_final
176     |
177     pc_type id pc_is pc_record id s_dospunts id s_final
178     ;
179
180
181 -- TIPUS COLECCIO
182 decl_coleccio:
183     p_dcoleccio s_parentesitancat pc_of id --s_final
184     ;
185
186 p_dcoleccio:
187     p_dcoleccio s_coma id
188     |
189     pc_type id pc_is pc_array s_parentesiobert id
190     ;
191
192
193 -- BLOC D'INSTRUCCIO
194 bloc:
195     bloc sentencia s_final
196     |
197     sentencia s_final
198     ;
199
200
201 -- SENTENCIES D'INSTRUCCIONS
202 sentencia:
203     sassig
204     |
205     scond
206     |
207     srep
208     |
209     crida_proc
210     ;
211
212 -- Sentencia assignacio
213 sassig:
```

```
214      referencia s_assignacio expressio
215      ;
216
217 -- Sentencia condicional
218 scond:
219     pc_if expressio pc_then
220         bloc
221     pc_end pc_if
222 |
223     pc_if expressio pc_then
224         bloc
225     pc_else
226         bloc
227     pc_end pc_if
228     ;
229
230 -- Sentencia bucle
231 srep:
232     pc_while expressio pc_loop
233         bloc
234     pc_end pc_loop
235     ;
236
237 -- Sentencia crida a procediment
238 crida_proc:
239     referencia --s_final
240     ;
241
242 referencia:
243     id
244 |
245     referencia s_puntrec id
246 |
247     pri s_parentesitancat
248     ;
249
250 pri:
251     referencia s_parentesiobert expressio
252 |
253     pri s_coma expressio
254     ;
255
256
```

```
257 -- Expressions
258 expressio:
259     expressio pc_or expressio
260 |
261     expressio pc_and expressio
262 |
263     pc_not expressio      %prec pc_not
264 |
265     expressio op_menor expressio
266 |
267     expressio op_menorigual expressio
268 |
269     expressio op_majorigual expressio
270 |
271     expressio op_major expressio
272 |
273     expressio op_igual expressio
274 |
275     expressio op_distint expressio
276 |
277     expressio op_suma expressio
278 |
279     expressio op_resta expressio
280 |
281     expressio op_multiplicacio expressio
282 |
283     expressio op_divisio expressio
284 |
285     expressio pc_mod expressio
286 |
287     op_resta expressio      %prec menys_unitari
288 |
289     s_parentesiobert expressio s_parentesitancat
290 |
291     referencia
292 |
293     const
294 ;
295
296
297 %%
298
299
```

```
300 PACKAGE pk_usintactica IS
301
302     PROCEDURE yyparse;
303
304
305 END pk_usintactica;
306
307
308
309 WITH     pk_usintactica_tokens ,
310          pk_usintactica_shift_reduce ,
311          pk_usintactica_goto ,
312          pk_ulexica_io ,
313          u_lexica ,
314          Ada.text_IO;
315
316
317 USE      pk_usintactica_tokens ,
318          pk_usintactica_shift_reduce ,
319          pk_usintactica_goto ,
320          pk_ulexica_io ,
321          u_lexica ,
322          Ada ,
323          ada.text_io;
324
325
326 PACKAGE BODY pk_usintactica IS
327
328     PROCEDURE YYError (e : IN string) IS
329
330     BEGIN
331
332         Put_Line(e);
333         RAISE Syntax_Error;
334
335     END YYError;
336
337 ##
338
339 END pk_usintactica;
```

3 Anàlisi Semàntica

3.1 Descripció

3.1.1 Fitxer *decls-dtdesc.ads*

```
1  -----
2  --   Paquet de declaracions del tipus descripcio
3  -----
4  --   Versio   :    0.1
5  --   Autors   :    Jose Ruiz Bravo
6  --               Biel Moya Alcover
7  --               Alvaro Medina Ballester
8  -----
9  --   Declaracions del tipus descripcio.
10 -----
11 -----
12
13
14 WITH      decls.dgenerals,
15           decls.d_taula_de_noms;
16
17 USE       decls.dgenerals,
18           decls.d_taula_de_noms;
19
20
21 PACKAGE decls.dtdesc IS
22
23     --pragma pure;
24
25     -- Representa tambit
26     max_nprof : CONSTANT integer := 10;
27     TYPE nprof IS NEW integer
28         RANGE 0 .. max_nprof;
29     nul_nprof : CONSTANT nprof := 0;
30
31     TYPE despl IS NEW natural;
32
33     TYPE valor IS NEW integer
34         RANGE 0 .. integer'Last;
35
36     max_var : CONSTANT integer := 1000;
37     TYPE num_var IS NEW natural
```

```

38     RANGE 0 .. max_var;
39
40     max_proc : CONSTANT integer := 100;
41     TYPE num_proc IS NEW natural
42         RANGE 0 .. max_proc;
43
44     -- Representa texpansio
45     TYPE rang_despl IS NEW integer
46         RANGE 0 .. (max_id * max_nprof);
47     nul_despl : CONSTANT rang_despl := 0;
48
49     TYPE tdescrip IS
50         (dnula,
51          dconst,
52          dvar,
53          dtipus,
54          dproc,
55          dcamp,
56          dargc);
57
58     TYPE tipusdetipus IS
59         (tsbool,
60          tscar,
61          tsent,
62          tsrec,
63          tsarr,
64          tsnul);
65
66     TYPE descriptipus (tt: tipusdetipus := tsnul) IS
67         RECORD
68             ocup : displ;
69             CASE tt IS
70                 WHEN tsbool | tscar | tsent =>
71                     linf, lsup : valor;
72                 WHEN tsarr => tcamp : id_nom;
73                 WHEN tsrec | tsnul => NULL;
74             END CASE;
75         END RECORD;
76
77     TYPE descrip (td : tdescrip := dnula) IS
78         RECORD
79             CASE td IS
80                 WHEN dnula => NULL;

```

```
81         WHEN dtipus => dt: descriptipus;
82         WHEN dvar    => tr: id_nom;
83                     nv: num_var;
84         WHEN dproc   => np: num_proc;
85         WHEN dconst  => tc: id_nom;
86                     vc: valor;
87         WHEN dargc   => nvarg: num_var;
88                     targ: id_nom;
89         WHEN dcamp   => tcamp: id_nom;
90                     dsp: rang_despl;
91     END CASE;
92 END RECORD;
93
94
95 END decls.dtdesc;
```

3.2 Taula de símbols

3.2.1 Fitxer *decls-dtsimbols.ads*

```
1  -- -----
2  --   Declaracions taula de símbols
3  -- -----
4  --   Versio   :    0.1
5  --   Autors   :    Jose Ruiz Bravo
6  --               Biel Moya Alcover
7  --               Alvaro Medina Ballester
8  -- -----
9  --   Declaracions dels procediments de la
10 --   taula de símbols.
11 --
12 -- -----
13
14
15 WITH   decls.dtdesc ,
16         decls.dgenerals ,
17         decls.d_taula_de_noms ,
18         Ada.Text_IO;
19
20 USE     decls.dtdesc ,
21         decls.dgenerals ,
22         decls.d_taula_de_noms ,
23         Ada.Text_IO;
24
25
26 PACKAGE decls.dtsimbols IS
27
28     --pragma pure;
29
30     TYPE tsimbols IS LIMITED PRIVATE;
31     TYPE cursor_idx IS NEW Rang_despl;
32     TYPE cursor_arg IS NEW Rang_despl;
33
34     -- Operacions
35     -- VERSIO 1: llenguatge simple sense estructura
36     -- de blocs estil Fortran.
37     PROCEDURE printts
38         (ts : IN tsimbols);
39
40     PROCEDURE tbuida
```



```
41     (ts : OUT tsimbols);
42
43     PROCEDURE posa
44     (ts : IN OUT tsimbols;
45      id : IN id_nom;
46      d : IN descrip;
47      e : OUT boolean);
48
49     FUNCTION cons
50     (ts : IN tsimbols;
51      id : IN id_nom) RETURN descrip;
52
53     -- VERSIO 2: Normal, llenguatge amb blocs
54     -- estil Pascal.
55     PROCEDURE entrabloc
56     (ts : IN OUT tsimbols);
57
58     PROCEDURE surtbloc
59     (ts : IN OUT tsimbols);
60
61     -- VERSIO 3: Blocs mes records.
62     PROCEDURE posacamp
63     (ts : IN OUT tsimbols;
64      idr : IN id_nom;
65      idc : IN id_nom;
66      d : IN descrip;
67      e : OUT boolean);
68
69     FUNCTION conscamp
70     (ts : IN tsimbols;
71      idr : IN id_nom;
72      idc : IN id_nom) RETURN descrip;
73
74     -- VERSIO 4: Arrays.
75     PROCEDURE posa_idx
76     (ts : IN OUT tsimbols;
77      ida : IN id_nom;
78      idi : IN id_nom;
79      e : OUT boolean);
80
81     FUNCTION primer_idx
82     (ts : IN tsimbols;
83      ida : IN id_nom) RETURN cursor_idx;
```

```
84
85     FUNCTION idx_valid
86         (ci : IN cursor_idx) RETURN boolean;
87
88     FUNCTION succ_idx
89         (ts : IN tsimbols;
90          ci : IN cursor_idx) RETURN cursor_idx;
91
92     FUNCTION cons_idx
93         (ts : IN tsimbols;
94          ci : IN cursor_idx) RETURN id_nom;
95
96     -- VERSIO 5: Procediments
97     PROCEDURE posa_arg
98         (ts : IN OUT tsimbols;
99          idp : IN id_nom;
100          ida : IN id_nom;
101          da : IN descrip;
102          e : OUT boolean);
103
104     FUNCTION primer_arg
105         (ts : IN tsimbols;
106          idp : IN id_nom) RETURN cursor_arg;
107
108     FUNCTION succ_arg
109         (ts : IN tsimbols;
110          ca : IN cursor_arg) RETURN cursor_arg;
111
112     FUNCTION arg_valid
113         (Ca : IN Cursor_arg) RETURN boolean;
114
115     PROCEDURE cons_arg
116         (ts : IN tsimbols;
117          ca : IN cursor_arg;
118          ida : OUT id_nom;
119          dn : OUT descrip);
120
121     PROCEDURE actualitza
122         (ts : IN OUT tsimbols;
123          id : IN id_nom;
124          d : IN descrip);
125
126 PRIVATE
```

```
127
128
129     TYPE tipus_descripcio IS RECORD
130         np : nprof;
131         d  : descripc;
132         s  : rang_despl;
133     END RECORD;
134
135     TYPE tipus_expansio IS RECORD
136         np : nprof;
137         d  : descripc;
138         id : id_nom;
139         s  : rang_despl; --Afegit per tal
140                           --d'implementar els records
141     END RECORD;
142
143     TYPE taula_ambits IS ARRAY
144         (1 .. nprof'Last) OF rang_despl;
145
146     TYPE taula_expansio IS ARRAY
147         (1 .. rang_despl'Last) OF tipus_expansio;
148
149     TYPE taula_desc IS ARRAY
150         (1 .. id_nom'Last) OF tipus_descripcio;
151
152     --TORNAR A DESCOMENTAR AIXO
153     --type cursor_idx is new rang_despl;
154     --type cursor_arg is new rang_despl;
155
156     TYPE tsimbols IS RECORD
157         tdesc : taula_desc;
158         texp  : taula_expansio;
159         tambit : taula_ambits;
160         prof  : nprof;
161     END RECORD;
162
163
164 END decls.dtsimbols;
```

3.2.2 Fitxer *decls-dtsimbols.adb*

```

1  -- -----
2  --   Procediments taula de simbols
3  -- -----
4  --   Versio   :    0.1
5  --   Autors   :    Jose Ruiz Bravo
6  --               Biel Moya Alcover
7  --               Alvaro Medina Ballester
8  -- -----
9  --   Procediments per tractar la taula de
10 --   simbols:
11 --       - Taula buida
12 --       - Posa
13 --       - Consulta
14 --       - Entra bloc
15 --       - Surt bloc
16 --       - Posa camp
17 --       - Consulta camp
18 --       - Posa Index
19 --       - Primer Index
20 --       - Successor Index
21 --       - Index valid?
22 --       - Consulta Index
23 --       - Posa argument
24 --       - Primer argument
25 --       - Successor argument
26 --       - Argument valid?
27 --       - Consulta argument
28 --       - Actualitza
29 --
30 -- -----
31
32 PACKAGE BODY decls.dtsimbols IS
33
34   PROCEDURE printts
35     (ts : IN tsimbols) IS
36   BEGIN
37     New_Line;
38     Put_Line("tambit -----");
39     FOR i IN 1 .. nprof'Last LOOP
40       Put_Line("tambit["&i'img&"] := "
41               &ts.tambit(i)'img);

```

```

42     END LOOP;
43
44     Put_Line("");
45     Put_Line("tdesc -----");
46     FOR i IN 1 .. (id_nom'Last-985) LOOP
47         Put("tdesc["&i'img&"] := (");
48         Put(ts.tdesc(i).np'img&", ");
49         CASE ts.tdesc(i).d.td IS
50             WHEN dnula => Put("dnula, ");
51             WHEN dtipus => Put("dtipus, ");
52             WHEN dvar => Put("dvar, ");
53             WHEN dproc => Put("dproc, ");
54             WHEN dconst => Put("dconst, ");
55             WHEN dargc => Put("dargc, ");
56             WHEN dcamp => Put("dcamp, ");
57         END CASE;
58         Put(ts.tdesc(i).s'img&")");
59         New_Line;
60     END LOOP;
61
62     New_Line;
63     Put_Line("texpansio -----");
64     FOR i IN 1 .. (rang_despl'Last-9985) LOOP
65         Put("texp["&i'img&"] := (");
66         Put(ts.texp(i).np'img&", ");
67         CASE ts.texp(i).d.td IS
68             WHEN dnula => Put("dnula, ");
69             WHEN dtipus => Put("dtipus, ");
70             WHEN dvar => Put("dvar, ");
71             WHEN dproc => Put("dproc, ");
72             WHEN dconst => Put("dconst, ");
73             WHEN dargc => Put("dargc, ");
74             WHEN dcamp => Put("dcamp, ");
75         END CASE;
76         Put(ts.texp(i).id'img&", ");
77         Put(ts.texp(i).s'img&")");
78         New_Line;
79     END LOOP;
80     Put_Line("PROFUNDITAT: "&ts.prof'img);
81 END printts;
82
83
84

```

```
85      -- VERSIO 1: llenguatge simple sense estructura
86      -- de blocs estil Fortran.
87      PROCEDURE tbuida
88          (ts : OUT tsimbols) IS
89
90          nul_desc : descripció(dnula);
91
92      BEGIN
93          ts.prof := 1;
94          ts.tambit(ts.prof) := nul_despl;
95
96          FOR i IN 1 .. id_nom'Last LOOP
97              ts.tdesc(i) := (nul_nprof, nul_desc,
98                             nul_despl);
99          END LOOP;
100
101      END tbuida;
102
103
104      PROCEDURE posa
105          (ts : IN OUT tsimbols;
106           id : IN id_nom;
107           d : IN descripció;
108           e : OUT boolean) IS
109
110          idespl : rang_despl;
111
112      BEGIN
113          e := (ts.tdesc(id).np = ts.prof);
114
115          IF NOT e THEN
116              ts.tambit(ts.prof) := ts.tambit(ts.prof) + 1;
117              idespl := ts.tambit(ts.prof);
118              ts.texp(idespl) := (ts.tdesc(id).np,
119                                 ts.tdesc(id).d, id, 0);
120              ts.tdesc(id) := (ts.prof, d, 0);
121          END IF;
122
123          printts(ts);
124
125      END posa;
126
127
```

```
128 FUNCTION cons
129     (ts : IN tsimbols;
130      id : IN id_nom)
131     RETURN descripció IS
132
133 BEGIN
134     RETURN ts.tdesc(id).d;
135 END cons;
136
137
138 -- VERSIO 2: Normal, llenguatge amb blocs estil
139 -- Pascal.
140 PROCEDURE entrabloc
141     (ts : IN OUT tsimbols) IS
142 BEGIN
143     ts.prof := ts.prof + 1;
144     ts.tambit(ts.prof) := ts.tambit(ts.prof - 1);
145 END entrabloc;
146
147
148 PROCEDURE surtbloc
149     (ts : IN OUT tsimbols) IS
150
151     idespl1 : rang_despl;
152     idespl2 : rang_despl;
153     id : id_nom;
154
155 BEGIN
156     idespl1 := ts.tambit(ts.prof);
157     ts.prof := ts.prof - 1;
158     idespl2 := ts.tambit(ts.prof) + 1;
159
160     FOR idespl IN REVERSE idespl1 .. idespl2 LOOP
161         IF ts.texp(idespl).np > 0 THEN
162             id := ts.texp(idespl).id;
163             ts.tdesc(id).d := ts.texp(idespl).d;
164             ts.tdesc(id).np := ts.texp(idespl).np;
165             ts.tdesc(id).s := ts.texp(idespl).s;
166         END IF;
167     END LOOP;
168
169 END surtbloc;
170
```

```

171
172 -- VERSIO 3: Blocs mes records.
173 PROCEDURE posacamp
174   (ts : IN OUT tsimbols;
175    idr : IN id_nom;
176    idc : IN id_nom;
177    d : IN descrip;
178    e : OUT boolean) IS
179
180     des : descrip;
181     td : descriptipus;
182     p : rang_despl;
183     itdespl : rang_despl;
184
185 BEGIN
186   des := ts.tdesc(idr).d;
187   IF des.td /= dtipus THEN e := TRUE; END IF;
188
189   td := des.dt;
190   IF td.tt /= tsrec THEN e := TRUE; END IF;
191
192   p := ts.tdesc(idr).s;
193   WHILE p /= 0 AND THEN ts.texp(p).id /= idc LOOP
194     p := ts.texp(p).s;
195   END LOOP;
196
197   e := (p /= 0);
198   IF NOT e THEN
199     ts.tambit(ts.prof) := ts.tambit(ts.prof) + 1;
200     itdespl := ts.tambit(ts.prof);
201     ts.texp(itdespl) := (nul_nprof, d, idc,
202                        ts.tdesc(idr).s);
203     ts.tdesc(idr).s := itdespl;
204   END IF;
205
206 END posacamp;
207
208
209 FUNCTION conscamp
210   (ts : IN tsimbols;
211    idr : IN id_nom;
212    idc : IN id_nom) RETURN descrip IS
213

```



```

214         d : descrip;
215         td : tdescrip;
216         p : rang_despl;
217         descnula : descrip(dnula);
218
219 BEGIN
220     d := ts.tdesc(idr).d;
221     td := d.td;
222
223     p := ts.tdesc(idr).s;
224     WHILE p /= 0 AND THEN ts.texp(p).id /= idc LOOP
225         p := ts.texp(p).s;
226     END LOOP;
227
228     IF p = 0 THEN
229         RETURN descnula;
230     ELSE
231         RETURN ts.texp(p).d;
232     END IF;
233
234 END conscamp;
235
236
237 -- VERSIO 4: Arrays.
238 PROCEDURE posa_idx
239     (ts : IN OUT tsimbols;
240      ida : IN id_nom;
241      idi : IN id_nom;
242      e : OUT boolean) IS
243
244         d : descrip;
245         dt : descriptipus;
246         p : rang_despl;
247         pp : rang_despl;
248         idespl : rang_despl;
249
250 BEGIN
251     d := ts.tdesc(ida).d;
252     IF d.td /= dtipus THEN e := TRUE; END IF;
253
254     dt := d.dt;
255     IF dt.tt /= tsarr THEN e := TRUE; END IF;
256

```

```

257     p := ts.tdesc(ida).s;
258     pp := 0;
259     WHILE p /= 0 LOOP -- Comprovar el 0
260         pp := p;
261         p := ts.texp(p).s;
262     END LOOP;
263
264     ts.tambit(ts.prof) := ts.tambit(ts.prof) + 1;
265     idespl := ts.tambit(ts.prof);
266     ts.texp(idespl) := (nul_nprof, (td => dnula),
267                        idi, 0);
268
269     IF pp /= 0 THEN
270         ts.texp(pp).s := idespl;
271     ELSE
272         ts.tdesc(ida).s := idespl;
273     END IF;
274
275
276 END posa_idx;
277
278
279 FUNCTION primer_idx
280     (ts : IN tsimbols;
281     ida : IN id_nom) RETURN cursor_idx IS
282 BEGIN
283     RETURN cursor_idx(ts.tdesc(ida).s);
284 END primer_idx;
285
286
287 FUNCTION idx_valid
288     (ci : IN cursor_idx) RETURN boolean IS
289 BEGIN
290     RETURN ci > 0;
291 END idx_valid;
292
293
294 FUNCTION succ_idx
295     (ts : IN tsimbols;
296     ci : IN cursor_idx) RETURN cursor_idx IS
297 BEGIN
298     IF idx_valid(ci) THEN
299         RETURN cursor_idx(ts.texp(rang_despl(ci)).s);

```

```
300         ELSE
301             RETURN 0; --Excepcio
302         END IF;
303     END succ_idx;
304
305
306     FUNCTION cons_idx
307     (ts : IN tsimbols;
308      ci : IN cursor_idx) RETURN id_nom IS
309     BEGIN
310         RETURN ts.texp(rang_despl(ci)).id;
311     END cons_idx;
312
313
314     PROCEDURE posa_arg
315     (ts : IN OUT tsimbols;
316      idp : IN id_nom;
317      ida : IN id_nom;
318      da : IN descrip;
319      e : OUT boolean) IS
320
321         d : descrip;
322         p : rang_despl;
323         pp : rang_despl;
324         idespl : rang_despl;
325
326     BEGIN
327         d := ts.tdesc(idp).d;
328         IF d.td /= dproc THEN e := TRUE; END IF;
329
330         p := ts.tdesc(idp).s;
331         pp := 0;
332         WHILE p /= 0 LOOP -- Comprovar el 0
333             pp := p;
334             p := ts.texp(p).s;
335         END LOOP;
336
337         ts.tambit(ts.prof) := ts.tambit(ts.prof) + 1;
338         idespl := ts.tambit(ts.prof);
339         ts.texp(idespl) := (nul_nprof, da, ida, 0);
340
341         IF pp /= 0 THEN
342             ts.texp(pp).s := idespl;
```

```
343         ELSE
344             ts.tdesc(idp).s := idespl;
345         END IF;
346
347     END Posa_Arg;
348
349
350     FUNCTION Primer_Arg
351     (Ts : IN Tsimbols;
352      Idp : IN Id_Nom) RETURN Cursor_Arg IS
353     BEGIN
354         RETURN cursor_arg(ts.tdesc(idp).s);
355     END Primer_Arg;
356
357
358     FUNCTION Succ_Arg
359     (ts : IN tsimbols;
360      ca : IN cursor_arg) RETURN cursor_arg IS
361     BEGIN
362         IF arg_valid(ca) THEN
363             RETURN cursor_arg(ts.texp(rang_despl(ca)).s);
364         ELSE
365             RETURN 0; --Excepcio
366         END IF;
367     END Succ_Arg;
368
369
370     FUNCTION Arg_Valid
371     (Ca : IN Cursor_Arg) RETURN Boolean IS
372     BEGIN
373         RETURN Ca > 0;
374     END Arg_Valid;
375
376
377     PROCEDURE cons_arg
378     (ts : IN tsimbols;
379      ca : IN cursor_arg;
380      ida : OUT id_nom;
381      Dn : OUT Descrip) IS
382     BEGIN
383         Ida := ts.texp(rang_despl(ca)).id;
384         Dn := Ts.Texp(Rang_Despl(Ca)).D;
385     END Cons_Arg;
```

```
386
387
388     PROCEDURE Actualitza
389         (Ts : IN OUT Tsimbols;
390          Id : IN Id_Nom;
391           D : IN Descrip) IS
392     BEGIN
393         Ts.Tdesc(id).D := D;
394     END Actualitza;
395
396
397 END decls.dtsimbols;
```

4 Declaracions i altres paquets

4.1 Fitxer *decls.ads*

```
1  -- -----
2  --   Paquet de Declaracions
3  -- -----
4  --   Versio   :    0.1
5  --   Autors   :    Jose Ruiz Bravo
6  --               Biel Moya Alcover
7  --               Alvaro Medina Ballester
8  -- -----
9  --   Paquet de declaracions pare.
10 --
11 -- -----
12
13 PACKAGE decls IS
14
15     --pragma pure;
16
17
18 END decls;
```

4.2 Fitxer *decls-dgenerals.ads*

```
1  -- -----
2  --   Paquet de declaracions generals
3  -- -----
4  --   Versio   :    0.3
5  --   Autors   :    Jose Ruiz Bravo
6  --               Biel Moya Alcover
7  --               Alvaro Medina Ballester
8  -- -----
9  --   Declaracions generals.
10 --
11 -- -----
12
13 PACKAGE decls.dgenerals IS
14
15     --pragma pure;
16
17     max_id : CONSTANT integer := 1000;
18     long_num_ident : CONSTANT integer := 40;
19
20
21 END decls.dgenerals;
```

5 Proves i programa principal

5.1 Fitxer *compilemon.adb*, programa principal

```
1  -- -----
2  -- Programa de prova
3  -- -----
4  -- Versio   :   0.2
5  -- Autors   :   Jose Ruiz Bravo
6  --           Biel Moya Alcover
7  --           Alvaro Medina Ballester
8  -- -----
9  -- Programa per comprovar les funcionalitats
10 -- del lexic i la taula de noms.
11 --
12 -- -----
13
14 WITH      Ada.Text_IO ,
15           Ada.Command_Line ,
16           decls.d_taula_de_noms ,
17           decls.dgenerals ,
18           decls.dtsimbols ,
19           decls.dtdesc ,
20           pk_usintactica_tokens ,
21           pk_ulexica_io ,
22           u_lexica ,
23           Pk_Usintactica ,
24           Decls.D_atribut;
25
26 USE      Ada.Text_IO ,
27           Ada.Command_Line ,
28           decls.d_taula_de_noms ,
29           decls.dgenerals ,
30           decls.dtsimbols ,
31           decls.dtdesc ,
32           pk_usintactica_tokens ,
33           pk_ulexica_io ,
34           u_lexica ,
35           Pk_Usintactica;
36
37 PROCEDURE compilemon IS
38 BEGIN
39
```



```
40  tbuida(Decls.D_Atribut.Tn);
41  Open_Input(Argument(1));
42  yyparse;
43  close_Input;
44
45  EXCEPTION
46      WHEN Syntax_Error =>
47          Put_Line("ERROR: Error a la linea "
48                  &yy_line_number'img&
49                  " i columna "&yy_begin_column'img);
50
51  END compilemon;
```

5.2 Proves

5.2.1 Programa de proves *compiproves.adb*

```
1  -- -----
2  --   Programa de prova
3  -- -----
4  --   Versio   :    0.1
5  --   Autors   :    Jose Ruiz Bravo
6  --               Biel Moya Alcover
7  --               Alvaro Medina Ballester
8  -- -----
9  --       Programa per comprovar les funcionalitats
10 --   de la taula de símbols.
11 --
12 -- -----
13
14 WITH   Ada.Text_IO,
15         Ada.Command_Line,
16         decls.d_taula_de_noms,
17         decls.tn,
18         decls.dgenerals,
19         d_token,
20         pk_ulexica_io,
21         u_lexica,
22         decls.dtsimbols,
23         decls.dtdesc;
24
25 USE     Ada.Text_IO,
26         Ada.Command_Line,
27         decls.d_taula_de_noms,
28         decls.tn,
29         decls.dgenerals,
30         d_token,
31         pk_ulexica_io,
32         u_lexica,
33         decls.dtsimbols,
34         decls.dtdesc;
35
36
37 PROCEDURE compiprueba IS
38     ts: tsimbols;
39     id: id_nom := 1;
40     d1: descrip(dproc);
```

```
41     d2: descrip(dproc);
42     d3: descrip(dvar);
43     d4: descrip(dtipus);
44     d5: descrip(dtipus);
45     D6: Descrip(Dvar);
46     e: boolean;
47     np1 : num_proc := 5;
48     np2 : num_proc := 7;
49     desctip : descriptipus(tsrec);
50     descarr : descriptipus(tsarr);
51 BEGIN
52
53     --Tbuida
54     tbuida(ts);
55
56     --Posa
57     d1.np := np1;
58     d2.np := np2;
59     id := 1;
60     posa(ts, id, d1, e);
61     id := 7;
62     posa(ts, id, d2, e);
63
64     --Cons
65     --New_Line;
66     --case cons(ts, 7).td is
67     --   when dnula => Put_Line("dnula");
68     --   when dtipus => Put_Line("dtipus");
69     --   when dvar => Put_Line("dvar");
70     --   when dproc => Put_Line("dproc");
71     --   when dconst => Put_Line("dconst");
72     --   when dargc => Put_Line("dargc");
73     --   when dcamp => Put_Line("dcamp");
74     --end case;
75     --Put_Line("np: "&cons(ts, 7).np'img);
76
77     --Entra bloc
78     entrabloc(ts);
79     D3.tr := 4;
80     d3.nv := 3;
81     posa(ts, id, d3, e);
82
83     --Surt bloc
```

```
84     surtbloc(ts);
85     printts(ts);
86
87     --Un altre entra bloc
88     --entrabloc(ts);
89     --posa(ts, id, d3, e);
90
91     --Comencam amb records
92     desctip.ocup := 8;
93     d4.dt := desctip;
94     id := 8;
95     posa(ts, id, d4, e);
96
97     --Ficam un posa camp
98     posacamp(ts, 8, 1, d3, e);
99     printts(ts);
100    posacamp(ts,8,7,d3,e);
101    printts(ts);
102
103    --Consulta camp
104    Put_Line("Cons camp: "&conscamp(ts,8,1).nv'img);
105    Put_Line("Cons camp: "&conscamp(ts,8,7).nv'img);
106
107    --Comencam amb els arrays
108    descarr.ocup := 8;
109    d5.dt := descarr;
110    posa(ts, 5, d5, e); --Ficam l'array
111    posa_idx(ts, 5, 31, e); --Afehim un camp a l'array
112    printts(ts);
113    posa_idx(ts, 5, 32, e); --Afehim un altre camp
114                                --a l'array
115    printts(ts);
116
117    --Primer_idx i idx_valid
118    Put_Line("PRIMER IDX: "
119            &primer_idx(ts, 5)'img);
120    Put_Line("IDX VALID: "
121            &idx_valid(primer_idx(ts, 5))'img);
122
123    --Provam el successor del camp 1
124    Put_Line("SUCCESSOR IDX: "
125            &succ_idx(ts, primer_idx(ts, 5))'img);
126    Put_Line("SUCCESSOR IDX: "
```

```

127         &succ_idx(ts, succ_idx(ts,
128             primer_idx(ts, 5)))'img);
129
130     --Consultam idx
131     Put_Line("CONS IDX: "
132         &cons_idx(ts, primer_idx(ts, 5))'img);
133     Put_Line("CONS IDX: "
134         &cons_idx(ts, succ_idx(ts,
135             primer_idx(ts, 5)))'img);
136
137     --Posa_arg
138     D6.Tr := 9;
139     d6.Nv := 5;
140     Posa(Ts, 9, D6, E); --Parametre variable 8 (argument)
141     Posa_Arg(Ts, 7, 9, D6, E); --Passam per primera vegada
142                               --l'argument
143     Posa_Arg(Ts, 7, 9, D6, E); --El passam per segona
144                               --vegada
145     Printts(Ts);
146
147     --Primer_arg
148
149     --Actualitza
150     Put_Line("Antes act: ");
151     CASE cons(ts, 7).td IS
152         WHEN dnula => Put_Line("dnula");
153         WHEN dtipus => Put_Line("dtipus");
154         WHEN dvar => Put_Line("dvar");
155         WHEN dproc => Put_Line("dproc");
156         WHEN dconst => Put_Line("dconst");
157         WHEN dargc => Put_Line("dargc");
158         WHEN dcamp => Put_Line("dcamp");
159     END CASE;
160
161     Actualitza(Ts, 7, D6);
162     Put_Line("Act dvar: ");
163     CASE cons(ts, 7).td IS
164         WHEN dnula => Put_Line("dnula");
165         WHEN dtipus => Put_Line("dtipus");
166         WHEN dvar => Put_Line("dvar");
167         WHEN dproc => Put_Line("dproc");
168         WHEN dconst => Put_Line("dconst");
169         WHEN dargc => Put_Line("dargc");

```

```
170     WHEN dcamp => Put_Line("dcamp");
171 END CASE;
172
173 IF e THEN
174     put_line("ERROR");
175 END IF;
176
177
178 END compiprueba;
```

5.2.2 Prova 1: fitxer *prova01.lem*

```
1  PROCEDURE compilemon IS
2      Tk:Token;
3      p:string;
4  BEGIN
5
6      --tbuida(tn);
7
8      Open_Input(Argument(1));
9      tk := Ylex;
10
11      WHILE tk /= end_of_input LOOP
12          Put(tok_begin_line);
13          Put_Line(Token(Tk));
14          tk := Ylex;
15      END LOOP;
16
17      a := 'a';
18      n := 9;
19      p := "Josep Lluís";
20
21  END compilemon;
```

5.2.3 Prova 2: fitxer *prova02.lem*

```
1  PROCEDURE joan IS
2
3      TYPE FCB IS RECORD
4          puyol : defensa;
5          xavi  : centre;
6          pique : defensa;
7          pep   : entrenador;
8      END RECORD;
9
10     bili : integer := 5;
11     bili : character;
12     alvs : CONSTANT integer := 69;
13     txavs : CONSTANT integer := 0;
14     pepe : character := 'p';
15
16     biel : string;
17
18     lluis : string := "Hola, tinc un iPhone";
19     miquel : string := "Hola "puc fer" webs";
20
21     PROCEDURE i IS
22     BEGIN
23         a:=5;
24     END i;
25
26     TYPE porcella IS NEW integer RANGE 1..345;
27     TYPE l IS ARRAY (porcella) OF integer;
28     TYPE l IS ARRAY (p) OF integer;
29
30 BEGIN
31
32     pepe := 4;
33
34     WHILE a = 1 LOOP
35         sa:=a+1;
36     END LOOP;
37
38     IF (p AND q) OR c THEN
39         alvaro := no;
40     ELSE
41         IF 1 THEN
```



```
42         e:= -3;
43         WHILE luis = pepe LOOP
44             k:=9;
45         END LOOP;
46     END IF;
47 END IF;
48
49     luis(d.t.r);
50     luis(k OR 2);
51     luis(k /= 2);
52
53     pp(-2);
54     pp(0-2);
55     lui(5*(3*3));
56
57     bili.s := 9;
58
59 END joan;
```

Índex

1	Anàlisi Lèxica	1
1.1	Descripció del lèxic: <i>pk_ulexica.l</i>	1
1.2	Taula de noms	6
1.2.1	Fitxer <i>decls-d_taula_de_noms.ads</i>	6
1.2.2	Fitxer <i>decls-d_taula_de_noms.adb</i>	9
1.3	Atributs	14
1.3.1	Fitxer <i>decls-d_atribut.ads</i>	14
1.3.2	Fitxer <i>decls-d_atribut.adb</i>	16
2	Anàlisi Sintàctica	18
2.1	Gramàtica del nostre llenguatge	18
2.2	Especificació <i>pk_usintactica.y</i>	21
3	Anàlisi Semàntica	29
3.1	Descripció	29
3.1.1	Fitxer <i>decls-dtdesc.ads</i>	29
3.2	Taula de símbols	32
3.2.1	Fitxer <i>decls-dtsimbols.ads</i>	32
3.2.2	Fitxer <i>decls-dtsimbols.adb</i>	36
4	Declaracions i altres paquets	46
4.1	Fitxer <i>decls.ads</i>	46
4.2	Fitxer <i>decls-dgenerals.ads</i>	47
5	Proves i programa principal	48
5.1	Fitxer <i>compilemon.adb</i> , programa principal	48
5.2	Proves	50
5.2.1	Programa de proves <i>compiproves.adb</i>	50
5.2.2	Prova 1: fitxer <i>prova01.lem</i>	55
5.2.3	Prova 2: fitxer <i>prova02.lem</i>	56