

Exercise P2. Parser for Simplified Modula

1 Aim of the Exercise

The aim of the exercise is to develop a simple parser for much simplified version of the programming language Modula. The parser should:

- recognize syntax of simplified Modula
- detect syntax errors

2 Preliminaries

After turning on the computer, one should select Linux, and in the lab log in as a user *student*. One should open a console window (e.g. press `F2` and type `xterm`), create one's own directory using a command `mkdir family name of the user`, and a subdirectory for the current exercise. Download files for Modula language from the Moodle web page of the course for the subject *Parsing*. The following files are to be found there:

- `Makefile` — needed for compilation with the command `make`
- `modula.y` — skeletal parser with comments and `FOUND` function already defined
- `test.mod` — test program correct under given grammar

Once the exercise has been completed, the directory should be removed.

3 Tasks

The (complete) lexical analyzer prepared in the previous exercise is a prerequisite for the current exercise. Any missing code should be added. The skeletal parser that is already available should be filled in with rules, and one has to show that the parser works correctly by testing it with test data made available in the exercise. The parser should print information about recognized syntactic constructions. To print such constructions, function `found()` has been made available. It has two parameters: the name of the construction (one should fill in the name of a grammar variable), and an argument that has a meaning (i.e. it is different from an empty string) for certain constructions, e.g. it can be the name of a function. One should strive to get the same output as in section 6.

The supplied code should be completed with the following items:

- import modules declaration (`IMPORT`)
- declaration of a constant (`CONST_DECL`)
- declaration of a variable (`VAR_DECL`)
- formal parameter section (`FP_SECTION`)
- procedure header (`PROC_HEAD`)
- procedure call (`PROCEDURE_CALL`)
- for loop (`FOR_STATEMENT`)
- procedure declaration (`PROC_DECL`)
- assignment (`ASSIGNMENT`)
- conditional (`IF_STATEMENT`)
- while loop (`WHILE_STATEMENT`)
- repeat until loop (`REPEAT_STATEMENT`)
- endless loop (`LOOP_STATEMENT`)
- case statement (`CASE_STATEMENT`)
- module definition (`PROGRAM_MODULE`)

The parser can be developed incrementally. Let us assume we have the following rule close to the beginning of the grammar:

```
1 A: B C D
2 ;
```

If we write it as above, we would have to rewrite all variables in the right-hand side of the rule. If A is the start symbol, we would have to write all the rules of the grammar. Not everyone manages to complete the whole parser in the lab. If the parser does not work, they get 0 points. However, it is possible to write the rules incrementally, item after item. In the rule for variable A, we initially comment out variables C and D:

```
1 A: B /* C D */
2 ;
```

Now, we have to rewrite variable B and all variables that show up in the derivation. The parser can be compiled and tested. Later, we can move the comment past variable C. Commenting out is a much better solution than skipping the rest of the rule, as it becomes immediately visible that the rule has further parts that have not been used yet.

Compiling the partial parser, one can encounter problems linked to %type directive that indicates variables for no rule has yet been written. The directive should be commented out until appropriate rules are added.

4 Grading

Each item from A to O deserves one point, thus 15 points in the lab. The points will be granted after a conversation with the teacher.

5 Test Data — File test.mod

```
1 (***** )
2 (* Program shows ASCII codes *)
3 (* Compilation: *)
4 (* m2c -all test.mod -o test *)
5 (* Running: *)
6 (* ./ test *)
7 (***** )
8 MODULE test;
9
10 FROM InOut IMPORT Write , WriteCard , WriteString , WriteLn;
11 CONST
12     FromAscii = 32;
13     ToAscii = 127;
14 VAR
15     i : CARDINAL;
16     fl : REAL;
17     t : ARRAY[1 .. 10] OF CARDINAL;
18     d : RECORD
19         year , month : CARDINAL;
20         day : CARDINAL;
21     END;
22
23 PROCEDURE ListAscii(StartCode , EndCode: CARDINAL; Precision: CARDINAL);
24 VAR
25     i: CARDINAL;
26     t1 : ARRAY[1 .. 10] OF CARDINAL;
27     d : RECORD
28         year: CARDINAL;
29         month , day : CARDINAL;
30     END;
31 BEGIN
32     WriteString(" ASCII codes ");
33     WriteLn;
34     FOR i := FromAscii TO ToAscii DO
35         WriteCard(i , 3);
36         Write(' ');
```

```

37         Write(CHR(i));
38         WriteLn
39     END;
40     t1[0] := t[0];
41     d.year := 2018
42 END ListAscii;
43
44 BEGIN
45     fl := 1.1 + 1.0E-2 + 1.0E+2 + 1.0E1; (* real numbers *)
46     IF (fl <= 11.11) AND (fl >= 1.111E1) THEN
47         WriteString("As expected!")
48     ELSE
49         WriteString("Gosh!")
50     END;
51     WriteLn;
52     i := 1;
53     WHILE i < 5 DO
54         WriteLn(i); i := i + 1
55     END;
56     REPEAT
57         WriteLn(i); i := i - 1
58     UNTIL i = 1;
59     LOOP
60         WriteLn("Spam")
61     END;
62     CASE CHR(FromAscii+16) OF
63         '0': WriteLn("Aha!")
64         | 'A','a': Writeln("Yes?")
65     ELSE
66         Writeln("O!")
67     END;
68     t[10] := 10;
69     FOR i := 9 DOWNTO 1 DO t[i] := t[i+1] * i * i END;
70     d.year := 2018; d.day := 1;
71     d.month := d.day * 10
72 END test.

```

6 Output of the Parser for test.mod

```

1 First and last name
2 yytext          Token type      Token value as string
3
4 MODULE          KW_MODULE
5 test            IDENT            test
6 ;
7 FROM            KW_FROM
8 InOut           IDENT            InOut
9 IMPORT          KW_IMPORT
10 Write           IDENT            Write
11 ,
12 WriteCard       IDENT            WriteCard
13 ,
14 WriteString     IDENT            WriteString
15 ,
16 WriteLn         IDENT            WriteLn
17 ;
18 ===== FOUND: IMPORT 'InOut' =====
19 CONST           KW_CONST
20 FromAscii       IDENT            FromAscii
21 =
22 32              INTEGER_CONST    32
23 ===== FOUND: CONST_DECL 'FromAscii' =====
24 ;
25 ToAscii         IDENT            ToAscii

```

```

26 =
27 127 INTEGER_CONST 127
28 ===== FOUND: CONST_DECL 'ToAscii' =====
29 ;
30 VAR KW_VAR
31 i IDENT i
32 :
33 CARDINAL IDENT CARDINAL
34 ;
35 ===== FOUND: VAR_DECL =====
36 fl IDENT fl
37 :
38 REAL IDENT REAL
39 ;
40 ===== FOUND: VAR_DECL =====
41 t IDENT t
42 :
43 ARRAY KW_ARRAY
44 [
45 1 INTEGER_CONST 1
46 .. RANGE
47 10 INTEGER_CONST 10
48 ]
49 OF KW_OF
50 CARDINAL IDENT CARDINAL
51 ;
52 ===== FOUND: VAR_DECL =====
53 d IDENT d
54 :
55 RECORD KW_RECORD
56 year IDENT year
57 ,
58 month IDENT month
59 :
60 CARDINAL IDENT CARDINAL
61 ;
62 day IDENT day
63 :
64 CARDINAL IDENT CARDINAL
65 ;
66 END KW_END
67 ;
68 ===== FOUND: VAR_DECL =====
69 PROCEDURE KW_PROCEDURE
70 ListAscii IDENT ListAscii
71 (
72 StartCode IDENT StartCode
73 ,
74 EndCode IDENT EndCode
75 :
76 CARDINAL IDENT CARDINAL
77 ;
78 ===== FOUND: FP_SECTION =====
79 Precision IDENT Precision
80 :
81 CARDINAL IDENT CARDINAL
82 )
83 ===== FOUND: FP_SECTION =====
84 ;
85 ===== FOUND: PROC_HEAD 'ListAscii' =====
86 VAR KW_VAR
87 i IDENT i
88 :
89 CARDINAL IDENT CARDINAL
90 ;
91 ===== FOUND: VAR_DECL =====

```

```

92 t1 IDENT t1
93 : :
94 ARRAY KW_ARRAY
95 [ [
96 1 INTEGER_CONST 1
97 .. RANGE
98 10 INTEGER_CONST 10
99 ] ]
100 OF KW_OF
101 CARDINAL IDENT CARDINAL
102 ; ;
103 ===== FOUND: VAR_DECL =====
104 d IDENT d
105 : :
106 RECORD KW_RECORD
107 year IDENT year
108 : :
109 CARDINAL IDENT CARDINAL
110 ; ;
111 month IDENT month
112 , ,
113 day IDENT day
114 : :
115 CARDINAL IDENT CARDINAL
116 ; ;
117 END KW_END
118 ; ;
119 ===== FOUND: VAR_DECL =====
120 BEGIN KW_BEGIN
121 WriteString IDENT WriteString
122 ( (
123 "ASCII codes" STRING_CONST "ASCII codes"
124 ) )
125 ===== FOUND: PROCEDURE_CALL 'WriteString' =====
126 ; ;
127 WriteLn IDENT WriteLn
128 ; ;
129 ===== FOUND: PROCEDURE_CALL 'WriteLn' =====
130 FOR KW_FOR
131 i IDENT i
132 := ASSIGN
133 FromAscii IDENT FromAscii
134 TO KW_TO
135 ToAscii IDENT ToAscii
136 DO KW_DO
137 WriteCard IDENT WriteCard
138 ( (
139 i IDENT i
140 , ,
141 3 INTEGER_CONST 3
142 ) )
143 ===== FOUND: PROCEDURE_CALL 'WriteCard' =====
144 ; ;
145 Write IDENT Write
146 ( (
147 ' ' CHAR_CONST ' '
148 ) )
149 ===== FOUND: PROCEDURE_CALL 'Write' =====
150 ; ;
151 Write IDENT Write
152 ( (
153 CHR IDENT CHR
154 ( (
155 i IDENT i
156 ) )
157 ) )

```

```

158 ===== FOUND: PROCEDURE_CALL 'Write'=====
159 ;
160 WriteLn IDENT WriteLn
161 END KW_END
162 ===== FOUND: PROCEDURE_CALL 'WriteLn'=====
163 ===== FOUND: FOR_STATEMENT 'i'=====
164 ;
165 t1 IDENT t1
166 [
167 0 INTEGER_CONST 0
168 ]
169 := ASSIGN
170 t IDENT t
171 [
172 0 INTEGER_CONST 0
173 ]
174 ;
175 ===== FOUND: ASSIGNMENT 't1'=====
176 d IDENT d
177 .
178 year IDENT year
179 := ASSIGN
180 2018 INTEGER_CONST 2018
181 END KW_END
182 ===== FOUND: ASSIGNMENT 'd'=====
183 ListAscii IDENT ListAscii
184 ===== FOUND: PROC_DECL 'ListAscii'=====
185 ;
186 BEGIN KW_BEGIN
187 f1 IDENT f1
188 := ASSIGN
189 1.1 FLOAT_CONST 1.1
190 +
191 1.0E-2 FLOAT_CONST 1.0E-2
192 +
193 1.0E+2 FLOAT_CONST 1.0E+2
194 +
195 1.0E1 FLOAT_CONST 1.0E1
196 ;
197 ===== FOUND: ASSIGNMENT 'f1'=====
198 IF KW_IF
199 (
200 f1 IDENT f1
201 <= LE
202 11.11 FLOAT_CONST 11.11
203 )
204 AND KW_AND
205 (
206 f1 IDENT f1
207 >= GE
208 1.111E1 FLOAT_CONST 1.111E1
209 )
210 THEN KW_THEN
211 WriteString IDENT WriteString
212 (
213 "As expected!" STRING_CONST "As expected!"
214 )
215 ===== FOUND: PROCEDURE_CALL 'WriteString'=====
216 ELSE KW_ELSE
217 WriteString IDENT WriteString
218 (
219 "Gosh!" STRING_CONST "Gosh!"
220 )
221 ===== FOUND: PROCEDURE_CALL 'WriteString'=====
222 END KW_END
223 ===== FOUND: IF_STATEMENT =====

```

```

224 ;
225 WriteLn IDENT WriteLn
226 ;
227 ===== FOUND: PROCEDURE_CALL 'WriteLn' =====
228 i IDENT i
229 := ASSIGN
230 1 INTEGER_CONST 1
231 ;
232 ===== FOUND: ASSIGNMENT 'i' =====
233 WHILE KW_WHILE
234 i IDENT i
235 < <
236 5 INTEGER_CONST 5
237 DO KW_DO
238 WriteLn IDENT WriteLn
239 ( (
240 i IDENT i
241 ) )
242 ===== FOUND: PROCEDURE_CALL 'WriteLn' =====
243 ;
244 i IDENT i
245 := ASSIGN
246 i IDENT i
247 + +
248 1 INTEGER_CONST 1
249 END KW_END
250 ===== FOUND: ASSIGNMENT 'i' =====
251 ===== FOUND: WHILE_STATEMENT =====
252 ;
253 REPEAT KW_REPEAT
254 WriteLn IDENT WriteLn
255 ( (
256 i IDENT i
257 ) )
258 ===== FOUND: PROCEDURE_CALL 'WriteLn' =====
259 ;
260 i IDENT i
261 := ASSIGN
262 i IDENT i
263 - -
264 1 INTEGER_CONST 1
265 UNTIL KW_UNTIL
266 ===== FOUND: ASSIGNMENT 'i' =====
267 i IDENT i
268 = =
269 1 INTEGER_CONST 1
270 ;
271 ===== FOUND: REPEAT_STATEMENT =====
272 LOOP KW_LOOP
273 WriteLn IDENT WriteLn
274 ( (
275 "Spam" STRING_CONST "Spam"
276 ) )
277 ===== FOUND: PROCEDURE_CALL 'WriteLn' =====
278 END KW_END
279 ===== FOUND: LOOP_STATEMENT =====
280 ;
281 CASE KW_CASE
282 CHR IDENT CHR
283 ( (
284 FromAscii IDENT FromAscii
285 + +
286 16 INTEGER_CONST 16
287 ) )
288 OF KW_OF
289 '0' CHAR_CONST '0'

```

```

290 :                               :
291 WriteLn                       IDENT           WriteLn
292 (                               (
293 "Aha!"                       STRING_CONST    "Aha!"
294 )                               )
295 ===== FOUND: PROCEDURE_CALL 'WriteLn' =====
296 |                               |
297 'A'                           CHAR_CONST     'A'
298 ,                               ,
299 'a'                           CHAR_CONST     'a'
300 :                               :
301 Writeln                       IDENT           Writeln
302 (                               (
303 "Yes?"                       STRING_CONST    "Yes?"
304 )                               )
305 ===== FOUND: PROCEDURE_CALL 'Writeln' =====
306 ELSE                           KW_ELSE
307 Writeln                       IDENT           Writeln
308 (                               (
309 "O!"                         STRING_CONST    "O!"
310 )                               )
311 ===== FOUND: PROCEDURE_CALL 'Writeln' =====
312 END                           KW_END
313 ===== FOUND: CASE_STATEMENT =====
314 ;                               ;
315 t                               IDENT           t
316 [                               [
317 10                             INTEGER_CONST  10
318 ]                               ]
319 :=                             ASSIGN
320 10                             INTEGER_CONST  10
321 ;                               ;
322 ===== FOUND: ASSIGNMENT 't' =====
323 FOR                           KW_FOR
324 i                               IDENT           i
325 :=                             ASSIGN
326 9                              INTEGER_CONST  9
327 DOWNTO                         KW_DOWNTO
328 1                              INTEGER_CONST  1
329 DO                             KW_DO
330 t                               IDENT           t
331 [                               [
332 i                               IDENT           i
333 ]                               ]
334 :=                             ASSIGN
335 t                               IDENT           t
336 [                               [
337 i                               IDENT           i
338 +                               +
339 1                              INTEGER_CONST  1
340 ]                               ]
341 *                               *
342 i                               IDENT           i
343 *                               *
344 i                               IDENT           i
345 END                           KW_END
346 ===== FOUND: ASSIGNMENT 't' =====
347 ===== FOUND: FOR_STATEMENT 'i' =====
348 ;                               ;
349 d                               IDENT           d
350 .                               .
351 year                           IDENT           year
352 :=                             ASSIGN
353 2018                           INTEGER_CONST  2018
354 ;                               ;
355 ===== FOUND: ASSIGNMENT 'd' =====

```



```

356 d IDENT d
357 .
358 day IDENT day
359 := ASSIGN
360 1 INTEGER_CONST 1
361 ;
362 ===== FOUND: ASSIGNMENT 'd' =====
363 d IDENT d
364 .
365 month IDENT month
366 := ASSIGN
367 d IDENT d
368 .
369 day IDENT day
370 *
371 10 INTEGER_CONST 10
372 END KW_END
373 ===== FOUND: ASSIGNMENT 'd' =====
374 test IDENT test
375 .
376 ===== FOUND: PROGRAM_MODULE 'test' =====

```