

# Exercise P2. Parser for Simplified Turbo Pascal

## 1 Aim of the Exercise

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

- recognize syntax of simplified Turbo Pascal
- 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 **[Alt]+[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 Turbo Pascal from the eNauzanie 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`
- `common.h` — header file defining the greatest length of strings
- `p.y` — skeletal parser with comments and `FOUND` function already defined
- `test.pas` — 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 available skeletal code should be supplemented with:

- A. program name (`PROGRAM_NAME`)
- B. declaration of a constant (`CONST`)
- C. section of declarations of constants (`CONST_SECT`)
- D. declarations of variables (`VAR`)
- E. variables declaration section (`VAR_SECT`)
- F. function/procedure declaration (`FUN_HEAD`)
- G. procedure definition (`PROCEDURE`)
- H. formal parameter (`FORM_PARAM`)
- I. block (`BLOCK`)
- J. function definition (`FUNCTION`)
- K. actual parameter (`ACT_PARAM`)
- L. function call (`FUNCT_CALL`)
- M. for loop (`FOR_INSTR`)
- N. assignment (`ASSIGN_INSTR`)

#### O. conditional statement (IF\_INSTR)

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 %ttype 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.pas

```
1 Program Testing;
2 (* Uses crt, dos; *)
3 Const (* range of displayed characters *)
4     minASCII = 30;
5     maxASCII = 255;
6     tekst = 'test string';
7 Var
8     c : Char;
9     r : real;
10    i, i1, _i, _00 : Integer;
11    t : array[1..10] of integer;
12    d : record
13        year, month : integer;
14        day          : integer
15    end;
16
17 Procedure Empty_Without_Parameters;
18 Begin
19 End;
20
21 Function Empty_With_Parameters( a : Integer, c : Char, r : Real ) : Integer;
22 Begin
23 End;
24
25 Procedure With_Declarations;
26 Const
27     r1 = 12.34;
28     r2 = 0.56;
29     r3 = 78.0;
30 Var
31     s : String;
32     t : array[1..10] of integer;
33     d : record
34         year, month : integer;
```

```

35         day          :   integer
36     end;
37 Begin
38 End;
39
40
41 Begin (* main block *)
42     Empty_Without_Parameters;
43     Empty_With_Parameters( 123, 'c', 12.34 );
44     ClrScr; (* intro opn clear screen *)
45     Writeln( 'Kody ASCII (30-255):' );
46     For i := minASCII To maxASCII Do (* display of given ASCII codes *)
47         Write( Chr( i ), ' ' );
48     ReadKey; (* wait for a key press *)
49     i := ( i1 + 3 ) * _00;
50     (* conditional instruction *)
51     if a > 10
52     then
53         b := a;
54     if ( a > 1 )
55     then
56         b := a
57     else
58         b := 1;
59     if ( a > b )
60     then
61         if ( a > c )
62         then
63             m := a
64         else
65             m := c
66         else
67             if ( b > c )
68             then
69                 m := b
70             else
71                 m := c;
72     t[10] := 1;
73     for i := 9 downto 1 do t[i] := t[i+1] * i * i;
74     d.year := 2018;
75     d.day := 1;
76     d.month := d.day * 10;
77 End.

```

## 6 Parser's Output for test.pas

```

1 Author: First and last name
2 yytext          Token type      Token value as string
3
4 Program          KW_PROGRAM
5 Testing          IDENT           Testing
6 ;
7 ===== FOUND: PROGRAM_NAME 'Testing'=====
8 Const           KW_CONST
9 minASCII         IDENT           minASCII
10 =
11 30               INTEGER_CONST   30
12 ===== FOUND: CONST 'minASCII'=====
13 ;
14 maxASCII         IDENT           maxASCII
15 =
16 255             INTEGER_CONST   255
17 ===== FOUND: CONST 'maxASCII'=====
18 ;

```

```

19 tekst          IDENT          tekst
20 =              =
21 'test string'  STRING_CONST   'test string'
22 ===== FOUND: CONST 'tekst'=====
23 ;
24 Var            KW_VAR
25 ===== FOUND: CONST_SECT =====
26 c              IDENT          c
27 :
28 Char           KW_CHAR
29 ===== FOUND: VAR =====
30 ;
31 r              IDENT          r
32 :
33 real           KW_REAL
34 ===== FOUND: VAR =====
35 ;
36 i              IDENT          i
37 ,
38 i1             IDENT          i1
39 ,
40 _i             IDENT          _i
41 ,
42 _00            IDENT          _00
43 :
44 Integer        KW_INTEGER
45 ===== FOUND: VAR =====
46 ;
47 t              IDENT          t
48 :
49 array          KW_ARRAY
50 [
51 1              INTEGER_CONST   1
52 ..            RANGE
53 10            INTEGER_CONST   10
54 ]
55 of             KW_OF
56 integer        KW_INTEGER
57 ===== FOUND: VAR =====
58 ;
59 d              IDENT          d
60 :
61 record         KW_RECORD
62 year           IDENT          year
63 ,
64 month          IDENT          month
65 :
66 integer        KW_INTEGER
67 ;
68 day            IDENT          day
69 :
70 integer        KW_INTEGER
71 end            KW_END
72 ===== FOUND: VAR =====
73 ;
74 Procedure      KW_PROCEDURE
75 ===== FOUND: VAR_SECT =====
76 Empty_Without_Param IDENT      Empty_Without_Parameters
77 ;
78 ===== FOUND: FUN_HEAD 'Empty_Without_Parameters'=====
79 Begin          KW_BEGIN
80 End            KW_END
81 ===== FOUND: BLOCK =====
82 ===== FOUND: PROCEDURE 'Empty_Without_Parameters'=====
83 ;
84 Function       KW_FUNCTION

```

```

85 Empty_With_Parameter IDENT      Empty_With_Parameters
86 (                      (
87 a                      IDENT      a
88 :                      :
89 Integer                KW_INTEGER
90 ===== FOUND: FORM_PARAM =====
91 ,                      ,
92 c                      IDENT      c
93 :                      :
94 Char                   KW_CHAR
95 ===== FOUND: FORM_PARAM =====
96 ,                      ,
97 r                      IDENT      r
98 :                      :
99 Real                   KW_REAL
100 ===== FOUND: FORM_PARAM =====
101 )                      )
102 ===== FOUND: FUN_HEAD 'Empty_With_Parameters' =====
103 :                      :
104 Integer                KW_INTEGER
105 ;                      ;
106 Begin                  KW_BEGIN
107 End                    KW_END
108 ===== FOUND: BLOCK =====
109 ===== FOUND: FUNCTION 'Empty_With_Parameters' =====
110 ;                      ;
111 Procedure              KW_PROCEDURE
112 With_Declarations      IDENT      With_Declarations
113 ;                      ;
114 ===== FOUND: FUN_HEAD 'With_Declarations' =====
115 Const                  KW_CONST
116 r1                      IDENT      r1
117 =                      =
118 12.34                   FLOAT_CONST 12.34
119 ===== FOUND: CONST 'r1' =====
120 ;                      ;
121 r2                      IDENT      r2
122 =                      =
123 0.56                    FLOAT_CONST 0.56
124 ===== FOUND: CONST 'r2' =====
125 ;                      ;
126 r3                      IDENT      r3
127 =                      =
128 78.0                    FLOAT_CONST 78.0
129 ===== FOUND: CONST 'r3' =====
130 ;                      ;
131 Var                    KW_VAR
132 ===== FOUND: CONST_SECT =====
133 s                      IDENT      s
134 :                      :
135 String                 KW_STRING
136 ===== FOUND: VAR =====
137 ;                      ;
138 t                      IDENT      t
139 :                      :
140 array                  KW_ARRAY
141 [                      [
142 1                      INTEGER_CONST 1
143 ..                     RANGE
144 10                     INTEGER_CONST 10
145 ]                      ]
146 of                     KW_OF
147 integer                KW_INTEGER
148 ===== FOUND: VAR =====
149 ;                      ;
150 d                      IDENT      d

```

```

151 :                               :
152 record                        KW_RECORD
153 year                          IDENT          year
154 ,                             ,
155 month                         IDENT          month
156 :                               :
157 integer                       KW_INTEGER
158 ;                             ;
159 day                           IDENT          day
160 :                               :
161 integer                       KW_INTEGER
162 end                            KW_END
163 ===== FOUND: VAR =====
164 ;                               ;
165 Begin                          KW_BEGIN
166 ===== FOUND: VAR_SECT =====
167 End                            KW_END
168 ===== FOUND: BLOCK =====
169 ===== FOUND: PROCEDURE 'With_Declarations'=====
170 ;                               ;
171 Begin                          KW_BEGIN
172 Empty_Without_ParamIDENT      Empty_Without_Parameters
173 ;                               ;
174 ===== FOUND: FUNCT_CALL 'Empty_Without_Parameters'=====
175 Empty_With_ParameterIDENT     Empty_With_Parameters
176 (                               (
177 123                             INTEGER_CONST 123
178 ===== FOUND: ACT_PARAM =====
179 ,                               ,
180 'c'                             STRING_CONST 'c'
181 ===== FOUND: ACT_PARAM =====
182 ,                               ,
183 12.34                           FLOAT_CONST 12.34
184 ===== FOUND: ACT_PARAM =====
185 )                               )
186 ===== FOUND: FUNCT_CALL 'Empty_With_Parameters'=====
187 ;                               ;
188 ClrScr                          IDENT          ClrScr
189 ;                               ;
190 ===== FOUND: FUNCT_CALL 'ClrScr'=====
191 Writeln                        IDENT          Writeln
192 (                               (
193 'Kody ASCII (30-255)STRING_CONST 'Kody ASCII (30-255):'
194 ===== FOUND: ACT_PARAM =====
195 )                               )
196 ===== FOUND: FUNCT_CALL 'Writeln'=====
197 ;                               ;
198 For                            KW_FOR
199 i                              IDENT          i
200 :=                             ASSIGN
201 minASCII                       IDENT          minASCII
202 To                             KW_TO
203 maxASCII                       IDENT          maxASCII
204 Do                             KW_DO
205 Write                          IDENT          Write
206 (                               (
207 Chr                            IDENT          Chr
208 (                               (
209 i                              IDENT          i
210 )                               )
211 ===== FOUND: FUNCT_CALL 'i'=====
212 ===== FOUND: ACT_PARAM =====
213 ===== FOUND: FUNCT_CALL 'Chr'=====
214 ===== FOUND: ACT_PARAM =====
215 ,                               ,
216 ' '                             STRING_CONST ' '

```

```

217 ===== FOUND: ACT_PARAM =====
218 )
219 ===== FOUND: FUNCT_CALL 'Write'=====
220 ===== FOUND: FOR_INSTR =====
221 ;
222 ReadKey IDENT ReadKey
223 ;
224 ===== FOUND: FUNCT_CALL 'ReadKey'=====
225 i IDENT i
226 := ASSIGN
227 (
228 i1 IDENT i1
229 +
230 3 INTEGER_CONST 3
231 )
232 *
233 _00 IDENT _00
234 ;
235 ===== FOUND: ASSIGN_INSTR 'i'=====
236 if KW_IF
237 a IDENT a
238 >
239 10 INTEGER_CONST 10
240 then KW_THEN
241 b IDENT b
242 := ASSIGN
243 a IDENT a
244 ;
245 ===== FOUND: ASSIGN_INSTR 'b'=====
246 ===== FOUND: IF_INSTR =====
247 if KW_IF
248 (
249 a IDENT a
250 >
251 1 INTEGER_CONST 1
252 )
253 then KW_THEN
254 b IDENT b
255 := ASSIGN
256 a IDENT a
257 else KW_ELSE
258 ===== FOUND: ASSIGN_INSTR 'b'=====
259 b IDENT b
260 := ASSIGN
261 1 INTEGER_CONST 1
262 ;
263 ===== FOUND: ASSIGN_INSTR 'b'=====
264 ===== FOUND: IF_INSTR =====
265 if KW_IF
266 (
267 a IDENT a
268 >
269 b IDENT b
270 )
271 then KW_THEN
272 if KW_IF
273 (
274 a IDENT a
275 >
276 c IDENT c
277 )
278 then KW_THEN
279 m IDENT m
280 := ASSIGN
281 a IDENT a
282 else KW_ELSE

```

```

283 ===== FOUND: ASSIGN_INSTR 'm'=====
284 m IDENT m
285 := ASSIGN
286 c IDENT c
287 else KW_ELSE
288 ===== FOUND: ASSIGN_INSTR 'm'=====
289 ===== FOUND: IF_INSTR =====
290 if KW_IF
291 ( (
292 b IDENT b
293 > >
294 c IDENT c
295 ) )
296 then KW_THEN
297 m IDENT m
298 := ASSIGN
299 b IDENT b
300 else KW_ELSE
301 ===== FOUND: ASSIGN_INSTR 'm'=====
302 m IDENT m
303 := ASSIGN
304 c IDENT c
305 ; ;
306 ===== FOUND: ASSIGN_INSTR 'm'=====
307 ===== FOUND: IF_INSTR =====
308 ===== FOUND: IF_INSTR =====
309 t IDENT t
310 [ [
311 10 INTEGER_CONST 10
312 ] ]
313 := ASSIGN
314 1 INTEGER_CONST 1
315 ; ;
316 ===== FOUND: ASSIGN_INSTR 't'=====
317 for KW_FOR
318 i IDENT i
319 := ASSIGN
320 9 INTEGER_CONST 9
321 downto KW_DOWNTO
322 1 INTEGER_CONST 1
323 do KW_DO
324 t IDENT t
325 [ [
326 i IDENT i
327 ] ]
328 := ASSIGN
329 t IDENT t
330 [ [
331 i IDENT i
332 + +
333 1 INTEGER_CONST 1
334 ] ]
335 * *
336 i IDENT i
337 * *
338 i IDENT i
339 ; ;
340 ===== FOUND: ASSIGN_INSTR 't'=====
341 ===== FOUND: FOR_INSTR =====
342 d IDENT d
343 . .
344 year IDENT year
345 := ASSIGN
346 2018 INTEGER_CONST 2018
347 ; ;
348 ===== FOUND: ASSIGN_INSTR 'd'=====

```



349	d	IDENT	d
350	.	.	
351	day	IDENT	day
352	:=	ASSIGN	
353	1	INTEGER_CONST	1
354	;	;	
355	===== FOUND: ASSIGN_INSTR 'd'=====		
356	d	IDENT	d
357	.	.	
358	month	IDENT	month
359	:=	ASSIGN	
360	d	IDENT	d
361	.	.	
362	day	IDENT	day
363	*	*	
364	10	INTEGER_CONST	10
365	;	;	
366	===== FOUND: ASSIGN_INSTR 'd'=====		
367	End	KW_END	
368	===== FOUND: BLOCK =====		
369	.	.	
370	===== FOUND: Complete program =====		