

Exercise P1. Lexical Scanner for Simplified Modula

1 Aim of the Exercise

The aim of the exercise is to build a simple scanner for a much simplified version of the programming language Modula. The task of the analyzer is:

- to recognize tokens of Modula and to determine their values
- to remove blanks and comments
- to recognize given directives
- to recognize lexical errors

2 Preliminaries

After turning on the computer, one should select Linux, and log in as *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 Modula from the Moodle web page of the course for the subject *Lexical Analysis*. The following files are to be found there:

- `p1m.pdf` — manual (just being read)
- `Makefile` — needed for compilation with the command `make`
- `modula.l` — skeletal lexical analyzer that needs to be completed; take a closer look at the definition of `process_token()`, which should be used in the rules
- `modula.y` — parser that is needed only for declaring tokens and for invoking the lexical analyzer
- `test.mod` — correct test program

After having completed the exercise, the directory should be removed.

3 Tasks

The supplied skeletal lexical analyzer should be extended so that it works correctly on supplied test programs. The analyzer should print information on recognized tokens in three columns:

1. matched text
2. recognized token
3. value of the token (only when it makes sense)

Function `process_token` is designed to print that information. The function returns a recognized token, so an action in a rule recognizing a token should contain `return process_token(. . .)` with appropriate parameters.

The supplied code needs to be completed with the following items:

- A. printing one's own name (in the `bison` program)
- B. detecting keywords specified in the source file for `bison`
- C. removing blanks
- D. recognition of multi-character operators (`<=`, `:=`, `...`) that appear in test programs
- E. recognition of identifiers
- F. recognition of integers
- G. recognition of floating point numbers
- H. recognition of strings in double quotes without start conditions
- I. recognition of character constants in apostrophes without start conditions

J. recognition of one-character tokens: operators and punctuation

K. recognition of strings in double quotes using start conditions

L. recognition of character constants in apostrophes using start conditions

M. removal of multi-line comments using start conditions

N. detection of comment end sequence without the beginning sequence using start conditions

O. detection of failure to close a comment with indications of the line where the comment begins

4 Grading

All items are graded as 1 point. If needed, items from K to O can be completed **at home for half a point each**. The file developed in the lab should be uploaded before the end of the class on Moodle. **The lexical analyzer will be needed for the next exercise.**

5 Start Conditions

- Start condition active at the start of the program: INITIAL
- Declaration: %x condition1, condition2, . . .
- Matching in a start condition:
 <con1> re1 action1;
 <con1,con2,INITIAL>re2 action2;
 <*>re3 działanie3
- changing start condition: BEGIN condition4
- current start condition: YY_START
- checking the current start condition after all input data has been read: in function yywrap, which must be defined, and which must return 1

6 Test Data — File test1.mod

```
1  (*****)  
2  (* Program pokazuje kody ASCII *)  
3  (* Kompilacja: *)  
4  (* m2c -all test.mod -o test *)  
5  (* Uruchomienie: *)  
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      rok, miesiac : CARDINAL;  
20      dzien : CARDINAL;  
21    END;  
22 BEGIN  
23    WriteString("Kody"); WriteString(" ASCII");  
24    WriteLn;  
25    FOR i := FromAscii TO ToAscii DO  
26      WriteCard(i, 3);
```

```

27   Write(' ');
28   Write(CHR(i));
29   WriteLn
30 END;
31 fl := 1.1 + 1.0E-2 + 1.0E+2 + 1.0E1; (* liczby rzeczywiste *)
32 IF (fl <= 11.11) AND (fl >= 1.111E1) THEN
33   WriteString("Zgodnie z oczekiwaniami")
34 ELSE
35   WriteString("Olaboga!")
36 END;
37 WriteLn;
38 i := 1;
39 WHILE i < 5 DO
40   WriteLn(i); i := i + 1
41 END;
42 REPEAT
43   WriteLn(i); i := i - 1
44 UNTIL i = 1;
45 LOOP
46   WriteLn("Spam")
47 END;
48 CASE CHR(FromAscii+16) OF
49   '0': WriteLn("Aha!")
50   | 'A', 'a': WriteLn("Tak?")
51 ELSE
52   WriteLn("O!")
53 END;
54 t[10] = 10;
55 FOR i := 9 DOWNTO 1 DO t[i] := t[i+1] * i * i END;
56 d.rok := 2018; d.dzien := 1;
57 d.miesiac := d.dzien * 10
58 END test.

```

7 Test Data — File test2.mod

```

1  (*****
2  (* Program pokazuje kody ASCII *)
3  (* Kompilacja: *)
4  (* m2c -all test.mod -o test *)
5  (* Uruchomienie: *)
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 BEGIN
18   WriteString("Kody"); WriteString(" ASCII");
19   WriteLn;
20   FOR i := FromAscii TO ToAscii DO
21     WriteCard(i, 3);
22     Write(' ');
23     Write(CHR(i));
24     WriteLn
25   END;
26   fl := 1.1 + 1.0E-2 + 1.0E+2 + 1.0E1; (* liczby rzeczywiste *)
27   IF (fl <= 11.11) AND (fl >= 1.111E1) THEN
28     WriteString("Zgodnie z oczekiwaniami")
29   ELSE

```

```

30   WriteString("Olaboga!")
31   END;
32   WriteLn;
33   i := 1;
34   WHILE i < 5 DO
35       WriteLn(i); i := i + 1
36   END;
37   REPEAT
38       WriteLn(i); i := i - 1
39   UNTIL i = 1;
40   LOOP *) (* zamkniecie komentarza bez otwarcia *)
41       WriteLn("Spam")
42   END;
43   CASE CHR(FromAscii+16) OF
44       '0': WriteLn("Aha!")
45       | 'A', 'a': Writeln("Tak?")
46   ELSE (* Ten komentarz nie ma zamknecia
47       Writeln("O!")
48   END
49 END test.

```

8 Output of the Lexical Analyzer for test1.mod

Imie i Nazwisko	Typ tokena	Wartosc tokena znakowo
yytext		
MODULE	KWMODULE	
test	IDENT	test
;	;	
FROM	KWFROM	
InOut	IDENT	InOut
IMPORT	KWIMPORT	
Write	IDENT	Write
,	,	
WriteCard	IDENT	WriteCard
,	,	
WriteString	IDENT	WriteString
,	,	
WriteLn	IDENT	WriteLn
;	;	
CONST	KW.CONST	
FromAscii	IDENT	FromAscii
=	=	
32	INTEGER.CONST	32
;	;	
ToAscii	IDENT	ToAscii
=	=	
127	INTEGER.CONST	127
;	;	
VAR	KW.VAR	
i	IDENT	i
:	:	
CARDINAL	IDENT	CARDINAL
;	;	
fl	IDENT	fl
:	:	
REAL	IDENT	REAL
;	;	
t	IDENT	t
:	:	
ARRAY	KW.ARRAY	
[[
1	INTEGER.CONST	1
..	RANGE	

42	10	INTEGER.CONST	10
43]]	
44	OF	KW.OF	
45	CARDINAL	IDENT	CARDINAL
46	;	;	
47	d	IDENT	d
48	:	:	
49	RECORD	KW.RECORD	
50	rok	IDENT	rok
51	,	,	
52	miesiac	IDENT	miesiac
53	:	:	
54	CARDINAL	IDENT	CARDINAL
55	;	;	
56	dzien	IDENT	dzien
57	:	:	
58	CARDINAL	IDENT	CARDINAL
59	;	;	
60	END	KW.END	
61	;	;	
62	BEGIN	KW.BEGIN	
63	WriteString	IDENT	WriteString
64	((
65	"Kody"	STRING.CONST	"Kody"
66))	
67	;	;	
68	WriteString	IDENT	WriteString
69	((
70	" ASCII"	STRING.CONST	" ASCII"
71))	
72	;	;	
73	WriteLn	IDENT	WriteLn
74	;	;	
75	FOR	KW.FOR	
76	i	IDENT	i
77	:=	ASSIGN	
78	FromAscii	IDENT	FromAscii
79	TO	KW.TO	
80	ToAscii	IDENT	ToAscii
81	DO	KW.DO	
82	WriteCard	IDENT	WriteCard
83	((
84	i	IDENT	i
85	,	,	
86	3	INTEGER.CONST	3
87))	
88	;	;	
89	Write	IDENT	Write
90	((
91	' '	CHAR.CONST	' '
92))	
93	;	;	
94	Write	IDENT	Write
95	((
96	CHR	IDENT	CHR
97	((
98	i	IDENT	i
99))	
100))	
101	;	;	
102	WriteLn	IDENT	WriteLn
103	END	KW.END	
104	;	;	
105	f1	IDENT	f1
106	:=	ASSIGN	
107	1.1	FLOAT.CONST	1.1

108	+	+	
109	1.0E-2	FLOAT.CONST	1.0E-2
110	+	+	
111	1.0E+2	FLOAT.CONST	1.0E+2
112	+	+	
113	1.0E1	FLOAT.CONST	1.0E1
114	;	;	
115	IF	KW_IF	
116	((
117	f1	IDENT	f1
118	<=	LE	
119	11.11	FLOAT.CONST	11.11
120))	
121	AND	KW_AND	
122	((
123	f1	IDENT	f1
124	>=	GE	
125	1.111E1	FLOAT.CONST	1.111E1
126))	
127	THEN	KW_THEN	
128	WriteString	IDENT	WriteString
129	((
130	"Zgodnie z oczekiwan	STRING.CONST	"Zgodnie z oczekiwaniami"
131))	
132	ELSE	KW_ELSE	
133	WriteString	IDENT	WriteString
134	((
135	"Olaboga!"	STRING.CONST	"Olaboga!"
136))	
137	END	KW_END	
138	;	;	
139	WriteLn	IDENT	WriteLn
140	;	;	
141	i	IDENT	i
142	:=	ASSIGN	
143	1	INTEGER.CONST	1
144	;	;	
145	WHILE	KW_WHILE	
146	i	IDENT	i
147	<	<	
148	5	INTEGER.CONST	5
149	DO	KW_DO	
150	WriteLn	IDENT	WriteLn
151	((
152	i	IDENT	i
153))	
154	;	;	
155	i	IDENT	i
156	:=	ASSIGN	
157	i	IDENT	i
158	+	+	
159	1	INTEGER.CONST	1
160	END	KW_END	
161	;	;	
162	REPEAT	KW_REPEAT	
163	WriteLn	IDENT	WriteLn
164	((
165	i	IDENT	i
166))	
167	;	;	
168	i	IDENT	i
169	:=	ASSIGN	
170	i	IDENT	i
171	-	-	
172	1	INTEGER.CONST	1
173	UNTIL	KW_UNTIL	

174	i	IDENT	i
175	=	=	
176	1	INTEGER_CONST	1
177	;	;	
178	LOOP	KW_LOOP	
179	WriteLn	IDENT	WriteLn
180	((
181	"Spam"	STRING_CONST	"Spam"
182))	
183	END	KW_END	
184	;	;	
185	CASE	KW_CASE	
186	CHR	IDENT	CHR
187	((
188	FromAscii	IDENT	FromAscii
189	+	+	
190	16	INTEGER_CONST	16
191))	
192	OF	KW_OF	
193	'0'	CHAR_CONST	'0'
194	:	:	
195	WriteLn	IDENT	WriteLn
196	((
197	"Aha!"	STRING_CONST	"Aha!"
198))	
199			
200	'A'	CHAR_CONST	'A'
201	,	,	
202	'a'	CHAR_CONST	'a'
203	:	:	
204	Writeln	IDENT	Writeln
205	((
206	"Tak?"	STRING_CONST	"Tak?"
207))	
208	ELSE	KW_ELSE	
209	Writeln	IDENT	Writeln
210	((
211	"O!"	STRING_CONST	"O!"
212))	
213	END	KW_END	
214	;	;	
215	t	IDENT	t
216	[[
217	10	INTEGER_CONST	10
218]]	
219	=	=	
220	10	INTEGER_CONST	10
221	;	;	
222	FOR	KW_FOR	
223	i	IDENT	i
224	:=	ASSIGN	
225	9	INTEGER_CONST	9
226	DOWNTO	KW_DOWNTO	
227	1	INTEGER_CONST	1
228	DO	KW_DO	
229	t	IDENT	t
230	[[
231	i	IDENT	i
232]]	
233	:=	ASSIGN	
234	t	IDENT	t
235	[[
236	i	IDENT	i
237	+	+	
238	1	INTEGER_CONST	1
239]]	

240	*	*	
241	i	IDENT	i
242	*	*	
243	i	IDENT	i
244	END	KW.END	
245	;	;	
246	d	IDENT	d
247	.	.	
248	rok	IDENT	rok
249	:=	ASSIGN	
250	2018	INTEGER_CONST	2018
251	;	;	
252	d	IDENT	d
253	.	.	
254	dzien	IDENT	dzien
255	:=	ASSIGN	
256	1	INTEGER_CONST	1
257	;	;	
258	d	IDENT	d
259	.	.	
260	miesiac	IDENT	miesiac
261	:=	ASSIGN	
262	d	IDENT	d
263	.	.	
264	dzien	IDENT	dzien
265	*	*	
266	10	INTEGER_CONST	10
267	END	KW.END	
268	test	IDENT	test
269	.	.	

9 Output of the Lexical Analyser for test2.mod

1	Imie i Nazwisko		
2	yytext	Typ tokena	Wartosc tokena znakowo
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	CONST	KW.CONST	
19	FromAscii	IDENT	FromAscii
20	=	=	
21	32	INTEGER_CONST	32
22	;	;	
23	ToAscii	IDENT	ToAscii
24	=	=	
25	127	INTEGER_CONST	127
26	;	;	
27	VAR	KW.VAR	
28	i	IDENT	i
29	:	:	
30	CARDINAL	IDENT	CARDINAL
31	;	;	

32	f1	IDENT	f1
33	:	:	
34	REAL	IDENT	REAL
35	;	;	
36	BEGIN	KW_BEGIN	
37	WriteString	IDENT	WriteString
38	((
39	"Kody"	STRING_CONST	"Kody"
40))	
41	;	;	
42	WriteString	IDENT	WriteString
43	((
44	" ASCII"	STRING_CONST	" ASCII"
45))	
46	;	;	
47	WriteLn	IDENT	WriteLn
48	;	;	
49	FOR	KW_FOR	
50	i	IDENT	i
51	:=	ASSIGN	
52	FromAscii	IDENT	FromAscii
53	TO	KW_TO	
54	ToAscii	IDENT	ToAscii
55	DO	KW_DO	
56	WriteCard	IDENT	WriteCard
57	((
58	i	IDENT	i
59	,	,	
60	3	INTEGER_CONST	3
61))	
62	;	;	
63	Write	IDENT	Write
64	((
65	' '	CHAR_CONST	' '
66))	
67	;	;	
68	Write	IDENT	Write
69	((
70	CHR	IDENT	CHR
71	((
72	i	IDENT	i
73))	
74))	
75	;	;	
76	WriteLn	IDENT	WriteLn
77	END	KW_END	
78	;	;	
79	f1	IDENT	f1
80	:=	ASSIGN	
81	1.1	FLOAT_CONST	1.1
82	+	+	
83	1.0E-2	FLOAT_CONST	1.0E-2
84	+	+	
85	1.0E+2	FLOAT_CONST	1.0E+2
86	+	+	
87	1.0E1	FLOAT_CONST	1.0E1
88	;	;	
89	IF	KW_IF	
90	((
91	f1	IDENT	f1
92	<=	LE	
93	11.11	FLOAT_CONST	11.11
94))	
95	AND	KW_AND	
96	((
97	f1	IDENT	f1

98	>=	GE	
99	1.111E1	FLOAT_CONST	1.111E1
100))	
101	THEN	KW.THEN	
102	WriteString	IDENT	WriteString
103	((
104	"Zgodnie z oczekiwan	STRING_CONST	"Zgodnie z oczekiwaniami"
105))	
106	ELSE	KW.ELSE	
107	WriteString	IDENT	WriteString
108	((
109	"Olaboga!"	STRING_CONST	"Olaboga!"
110))	
111	END	KW.END	
112	;	;	
113	WriteLn	IDENT	WriteLn
114	;	;	
115	i	IDENT	i
116	:=	ASSIGN	
117	1	INTEGER_CONST	1
118	;	;	
119	WHILE	KW.WHILE	
120	i	IDENT	i
121	<	<	
122	5	INTEGER_CONST	5
123	DO	KW.DO	
124	WriteLn	IDENT	WriteLn
125	((
126	i	IDENT	i
127))	
128	;	;	
129	i	IDENT	i
130	:=	ASSIGN	
131	i	IDENT	i
132	+	+	
133	1	INTEGER_CONST	1
134	END	KW.END	
135	;	;	
136	REPEAT	KW.REPEAT	
137	WriteLn	IDENT	WriteLn
138	((
139	i	IDENT	i
140))	
141	;	;	
142	i	IDENT	i
143	:=	ASSIGN	
144	i	IDENT	i
145	-	-	
146	1	INTEGER_CONST	1
147	UNTIL	KW.UNTIL	
148	i	IDENT	i
149	=	=	
150	1	INTEGER_CONST	1
151	;	;	
152	LOOP	KW.LOOP	
153	Comment closed in line 40 when none opened		
154	WriteLn	IDENT	WriteLn
155	((
156	"Spam"	STRING_CONST	"Spam"
157))	
158	END	KW.END	
159	;	;	
160	CASE	KW.CASE	
161	CHR	IDENT	CHR
162	((
163	FromAscii	IDENT	FromAscii

164	+	+	
165	16	INTEGER.CONST	16
166))	
167	OF	KW.OF	
168	'0'	CHAR.CONST	'0'
169	:	:	
170	WriteLn	IDENT	WriteLn
171	((
172	"Aha!"	STRING.CONST	"Aha!"
173))	
174			
175	'A'	CHAR.CONST	'A'
176	,	,	
177	'a'	CHAR.CONST	'a'
178	:	:	
179	Writeln	IDENT	Writeln
180	((
181	"Tak?"	STRING.CONST	"Tak?"
182))	
183	ELSE	KW.ELSE	
184	Comment opened in line 46 not closed		