Zajęcia P1. Analizator leksykalny dla uproszczonego języka Modula

1 Cel ćwiczeń

Celem ćwiczeń jest stworzenie prostego analizatora leksykalnego dla bardzo uproszczonej wersji języka programowania Modula. Zadaniem tworzonego analizatora jest:

- rozpoznawanie elementów języka Modula i określanie ich wartości
- usuwanie białych znaków i komentarzy
- wykrywanie wskazanych dyrektyw
- wykrywanie błędów leksykalnych

2 Czynności wstępne

Po włączeniu komputera należy wybrać system operacyjny Linux i zalogować się jako użytkownik `student`. Należy otworzyć okno konsoli (np. nacisnąć `Alt-F2` i napisać `xterm`), utworzyć własny podkatalog za pomocą polecenia `mkdir` *nazwisko użytkownika*, i podkatalog dla bieżącego ćwiczenia. Ze strony przedmiotu na platformie Moodle dla tematu *Analiza leksykalna* należy pobrać pliki. Znajdują się tam następujące pliki:

- `p1m.pdf` — instrukcja (właśnie czytany plik)
- `Makefile` — potrzebny do kompilacji za pomocą polecenia `make`
- `modula.l` — szkielet analizatora leksykalnego, który należy uzupełnić; należy zwrócić uwagę na definicję funkcji `process_token()`, którą należy wykorzystać
- `modula.y` — analizator składniowy, którego jedynymi zadaniami jest deklaracja rozpoznawanych elementów końcowych oraz wywołanie analizatora leksykalnego
- `test.mod` — program testowy poprawny w danej gramatyce

Po zakończeniu pracy wskazane jest usunięcie utworzonego katalogu wraz z zawartością.

3 Zadania do wykonania

Należy uzupełnić dostarczony szkielet analizatora leksykalnego i pokazać, że działa poprawnie testując do na dostarczonych programach testowych. Analizator powinien wypisywać informacje o rozpoznanych symbolach końcowych w trzech kolumnach:

1. dopasowany tekst
2. rozpoznany symbol
3. wartość symbolu (tylko w sytuacji, gdy symbol rzeczywiście ma wartość)

Do wypisywania tych informacji służy zawarta w dostarczonym szkielecie analizatora funkcja `process_token()`. Funkcja zwraca rozpoznany symbol, dlatego w działaniu dla reguły rozpoznającej symbol powinna znaleźć się instrukcja `return process_token(. . .)` z odpowiednimi parametrami funkcji.

Dostarczony kod należy uzupełnić o następujące elementy:

A. wypisanie własnego imienia i nazwiska (w pliku dla programu `bison`)

B. wykrywanie słów kluczowych zdefiniowanych w pliku źródłowym dla programu `bison` (występujących w większości w programach testowych)

C. usuwanie białych znaków

D. wykrywanie operatorów wieloznakowych (`<=`, `:=`,. . .) występujących w programach testowych

E. wykrywanie identyfikatorów

F. wykrywanie liczba całkowitych

G. wykrywanie liczb rzeczywistych

H. wykrywanie stałych tekstowych (napisów) w cudzysłowach bez użycia mechanizmu warunków początkowych

I. wykrywanie stałych znakowych w apostrofach bez użycia mechanizmu warunków początkowych

J. wykrywanie symboli końcowych jednoznakowych: operatorów, interpunkcji

K. wykrywanie napisów w cudzysłowach z użyciem warunków początkowych

L. wykrywanie stałych znakowych w apostrofach z użyciem warunków początkowych

M. usuwanie komentarzy wielowierszowych z użyciem warunków początkowych

N. znajdowanie znaków zamknięcia komentarza przy braku jego rozpoczęcia z użyciem warunków początkowych

O. wykrywanie niezamkniętego komentarza ze wskazaniem wiersza jego rozpoczęcia z użyciem warunków początkowych

4 Ocena

Za każdy element można dostać 1 punkt, czyli razem 15 punktów. Jeżeli ktoś nie zdąży zrobić wszystkich elementów na zajęciach, istnieje możliwość dokończenia analizatora w domu, ale **tylko elementów od K do O** i za każdy element **w domu przysługuje tylko połowa punktów**. Plik wykonany na zajęciach umieścić **na zajęciach** na platformie Moodle. UWAGA: **Analizator leksykalny potrzebny będzie na następnych zajęciach**.

5 Przypomnienie warunków początkowych

- Warunek początkowy aktywny na początku programu: `INITIAL`
- Deklaracja (w pierwszej części): `%x warunek1, warunek2,. . .`
- Dopasowanie w określonym stanie:
```
<war1> re1      działanie1;
<war1,war2,INITIAL>re2 działanie2;
<*>re3        działanie3
```
- zmiana warunku początkowego: `BEGIN warunek4`
- bieżący warunek początkowy: `YY_START`
- sprawdzenie bieżącego warunku po odczytaniu wszystkich danych wejściowych: w funkcji `yywrap`, którą należy zdefiniować i która musi zwrócić wartość 1

6 Dane testowe — plik test1.mod

```
(*******************************************************)
(* Program pokazuje kody ASCII                         *)
(* Kompilacja:                                         *)
(*    m2c -all test.mod -o test                        *)
(* Uruchomienie:                                       *)
(*    ./test                                           *)
(*******************************************************)
MODULE test;

FROM InOut IMPORT Write, WriteCard, WriteString, WriteLn;
CONST
   FromAscii = 32;
   ToAscii = 127;
```

```
VAR
   i : CARDINAL;
   fl : REAL;
   t : ARRAY[1 .. 10] OF CARDINAL;
   d : RECORD
         rok, miesiac : CARDINAL;
         dzien : CARDINAL;
   END;
BEGIN
   WriteString("Kody"); WriteString(" ASCII");
   WriteLn;
   FOR i := FromAscii TO ToAscii DO
      WriteCard(i, 3);
      Write(' ');
      Write(CHR(i));
      WriteLn
   END;
   fl := 1.1 + 1.0E-2 + 1.0E+2 + 1.0E1; (* liczby rzeczywiste *)
   IF (fl <= 11.11) AND (fl >= 1.111E1) THEN
      WriteString("Zgodnie z oczekiwaniami")
   ELSE
      WriteString("Olaboga!")
   END;
   WriteLn;
   i := 1;
   WHILE i < 5 DO
         WriteLn(i); i := i + 1
   END;
   REPEAT
         WriteLn(i); i := i - 1
   UNTIL i = 1;
   LOOP
         WriteLn("Spam")
   END;
   CASE CHR(FromAscii+16) OF
         '0': WriteLn("Aha!")
      | 'A','a': Writeln("Tak?")
   ELSE
         Writeln("O!")
   END;
   t[10] = 10;
   FOR i := 9 DOWNTO 1 DO t[i] := t[i+1] * i * i END;
   d.rok := 2018; d.dzien := 1;
   d.miesiac := d.dzien * 10
END test.
```

7 Dane testowe — plik test2.mod

```
(*******************************************************)
(* Program pokazuje kody ASCII                         *)
(* Kompilacja:                                         *)
(*    m2c -all test.mod -o test                        *)
(* Uruchomienie:                                       *)
(*    ./test                                           *)
(*******************************************************)
MODULE test;

FROM InOut IMPORT Write, WriteCard, WriteString, WriteLn;
CONST
   FromAscii = 32;
   ToAscii = 127;
VAR
   i : CARDINAL;
   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 zamkniecia
47       Writeln("O!")
48   END
49 END test.
```

8 Wyjście analizatora leksykalnego dla test1.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
```

```
:                   :
CARDINAL            IDENT           CARDINAL
;                   ;
fl                  IDENT           fl
:                   :
REAL                IDENT           REAL
;                   ;
t                   IDENT           t
:                   :
ARRAY               KW_ARRAY
[                   [
1                   INTEGER_CONST   1
..                  RANGE
10                  INTEGER_CONST   10
]                   ]
OF                  KW_OF
CARDINAL            IDENT           CARDINAL
;                   ;
d                   IDENT           d
:                   :
RECORD              KW_RECORD
rok                 IDENT           rok
,                   ,
miesiac             IDENT           miesiac
:                   :
CARDINAL            IDENT           CARDINAL
;                   ;
dzien               IDENT           dzien
:                   :
CARDINAL            IDENT           CARDINAL
;                   ;
END                 KW_END
;                   ;
BEGIN               KW_BEGIN
WriteString         IDENT           WriteString
(                   (
"Kody"              STRING_CONST    "Kody"
)                   )
;                   ;
WriteString         IDENT           WriteString
(                   (
" ASCII"            STRING_CONST    " ASCII"
)                   )
;                   ;
WriteLn             IDENT           WriteLn
;                   ;
FOR                 KW_FOR
i                   IDENT           i
:=                  ASSIGN
FromAscii           IDENT           FromAscii
TO                  KW_TO
ToAscii             IDENT           ToAscii
DO                  KW_DO
WriteCard           IDENT           WriteCard
(                   (
i                   IDENT           i
,                   ,
3                   INTEGER_CONST   3
)                   )
;                   ;
Write               IDENT           Write
(                   (
' '                 CHAR_CONST      ' '
)                   )
;                   ;
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 fl                      IDENT           fl
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 fl                      IDENT           fl
118 <=                      LE
119 11.11                   FLOAT_CONST     11.11
120 )                       )
121 AND                     KW_AND
122 (                       (
123 fl                      IDENT           fl
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
```

```
;                   ;
REPEAT              KW_REPEAT
 WriteLn            IDENT           WriteLn
 (                  (
 i                  IDENT           i
 )                  )
 ;                  ;
 i                  IDENT           i
 :=                 ASSIGN
 i                  IDENT           i
 -                  -
 1                  INTEGER_CONST   1
 UNTIL              KW_UNTIL
 i                  IDENT           i
 =                  =
 1                  INTEGER_CONST   1
 ;                  ;
LOOP                KW_LOOP
 WriteLn            IDENT           WriteLn
 (                  (
 "Spam"             STRING_CONST    "Spam"
 )                  )
END                 KW_END
 ;                  ;
CASE                KW_CASE
CHR                 IDENT           CHR
 (                  (
 FromAscii          IDENT           FromAscii
 +                  +
 16                 INTEGER_CONST   16
 )                  )
OF                  KW_OF
 '0'                CHAR_CONST      '0'
 :                  :
 WriteLn            IDENT           WriteLn
 (                  (
 "Aha!"             STRING_CONST    "Aha!"
 )                  )
 |                  |
 'A'                CHAR_CONST      'A'
 ,                  ,
 'a'                CHAR_CONST      'a'
 :                  :
 Writeln            IDENT           Writeln
 (                  (
 "Tak?"             STRING_CONST    "Tak?"
 )                  )
ELSE                KW_ELSE
 Writeln            IDENT           Writeln
 (                  (
 "O!"               STRING_CONST    "O!"
 )                  )
END                 KW_END
 ;                  ;
 t                  IDENT           t
 [                  [
 10                 INTEGER_CONST   10
 ]                  ]
 =                  =
 10                 INTEGER_CONST   10
 ;                  ;
FOR                 KW_FOR
 i                  IDENT           i
 :=                 ASSIGN
 9                  INTEGER_CONST   9
DOWNTO              KW_DOWNTO
```

```
1                       INTEGER_CONST   1
DO                      KW_DO
t                       IDENT           t
[                       [
i                       IDENT           i
]                       ]
:=                      ASSIGN
t                       IDENT           t
[                       [
i                       IDENT           i
+                       +
1                       INTEGER_CONST   1
]                       ]
*                       *
i                       IDENT           i
*                       *
i                       IDENT           i
END                     KW_END
;                       ;
d                       IDENT           d
.                       .
rok                     IDENT           rok
:=                      ASSIGN
2018                    INTEGER_CONST   2018
;                       ;
d                       IDENT           d
.                       .
dzien                   IDENT           dzien
:=                      ASSIGN
1                       INTEGER_CONST   1
;                       ;
d                       IDENT           d
.                       .
miesiac                 IDENT           miesiac
:=                      ASSIGN
d                       IDENT           d
.                       .
dzien                   IDENT           dzien
*                       *
10                      INTEGER_CONST   10
END                     KW_END
test                    IDENT           test
.                       .
```

9 Wyjście analizatora leksykalnego dla test2.mod

```
Imie i Nazwisko
yytext                  Typ tokena          Wartosc tokena znakowo

MODULE                  KW_MODULE
test                    IDENT           test
;                       ;
FROM                    KW_FROM
InOut                   IDENT           InOut
IMPORT                  KW_IMPORT
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
;                       ;
BEGIN                   KW_BEGIN
WriteString             IDENT               WriteString
(                       (
"Kody"                  STRING_CONST        "Kody"
)                       )
;                       ;
WriteString             IDENT               WriteString
(                       (
" ASCII"                STRING_CONST        " ASCII"
)                       )
;                       ;
WriteLn                 IDENT               WriteLn
;                       ;
FOR                     KW_FOR
i                       IDENT               i
:=                      ASSIGN
FromAscii               IDENT               FromAscii
TO                      KW_TO
ToAscii                 IDENT               ToAscii
DO                      KW_DO
WriteCard               IDENT               WriteCard
(                       (
i                       IDENT               i
,                       ,
3                       INTEGER_CONST       3
)                       )
;                       ;
Write                   IDENT               Write
(                       (
' '                     CHAR_CONST          ' '
)                       )
;                       ;
Write                   IDENT               Write
(                       (
CHR                     IDENT               CHR
(                       (
i                       IDENT               i
)                       )
)                       )
;                       ;
WriteLn                 IDENT               WriteLn
END                     KW_END
;                       ;
fl                      IDENT               fl
:=                      ASSIGN
1.1                     FLOAT_CONST         1.1
+                       +
1.0E-2                  FLOAT_CONST         1.0E-2
+                       +
```

```
85  1.0E+2                  FLOAT_CONST     1.0E+2
86  +                       +
87  1.0E1                   FLOAT_CONST     1.0E1
88  ;                       ;
89  IF                      KW_IF
90  (                       (
91  fl                      IDENT           fl
92  <=                      LE
93  11.11                   FLOAT_CONST     11.11
94  )                       )
95  AND                     KW_AND
96  (                       (
97  fl                      IDENT           fl
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
```