Zajęcia P2. Analizator składniowy dla uproszczonego języka Modula

1 Cel ćwiczeń

Celem ćwiczeń jest stworzenie prostego analizatora składniowego dla bardzo uproszczonej wersji języka programowania Modula2. Zadaniem tworzonego analizatora jest:

- rozpoznawanie konstrukcji składniowych języka Modula2
- wykrywanie błędów składniowych

2 Czynności wstępne

Po włączeniu komputera należy wybrać system operacyjny Linux i w laboratorium 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 eNauczanie należy pobrać następujące pliki:

- `Makefile` – potrzebny do kompilacji za pomocą polecenia `make`
- `modula.y` – szczątkowy analizator składniowy z komentarzami i zdefiniowaną funkcją FOUND
- `test.mod` – program testowy poprawny w danej gramatyce

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

3 Zadania do wykonania

Do wykonania zadania potrzebny jest analizator leksykalny przygotowany w poprzednim ćwiczeniu. W przypadku braków w analizatorze leksykalnym należy je uzupełnić. Należy uzupełnić dostarczony szkielet analizatora składniowego i pokazać, że działa poprawnie testując go na dostarczonych programach testowych. Analizator powinien wypisywać informacje o rozpoznanych konstrukcjach składniowych. Do wypisywania tych informacji służy zawarta w dostarczonym szkielecie analizatora funkcja `found()`. Funkcja ma dwa parametry: nazwa rozpoznanej konstrukcji (należy tu wpisać nazwę zmiennej gramatyki) oraz argument, który ma znaczenie (jest różny od pustego napisu) dla niektórych konstrukcji, np. może być nazwą funkcji. Należy dążyć do uzyskania takiego samego wyjścia programu jak w punkcie 6.

Dostarczony kod należy uzupełnić o następujące elementy:

A. deklaracja importu modułów (IMPORT)
B. deklaracja stałej (CONST_DECL)
C. deklaracja zmiennej (VAR_DECL)
D. sekcja parametrów formalnych (FP_SECTION)
E. nagłówek procedury (PROC_HEAD)
F. wywołanie procedury (PROCEDURE_CALL)
G. pętla for (FOR_STATEMENT)
H. deklaracja procedury (PROC_DECL)
I. przypisanie (ASSIGNMENT)
J. instrukcja warunkowa (IF_STATEMENT)
K. pętla while (WHILE_STATEMENT)
L. pętla repeat until (REPEAT_STATEMENT)
M. pętla nieskończona (LOOP_STATEMENT)
N. instrukcja wyboru (CASE_STATEMENT)

O. definicja modułu (PROGRAM_MODULE)

Analizator składniowy można uruchamiać przyrostowo. Przypuśćmy, że gdzieś na początku gramatyki mamy regułę:

```
A: B C D
;
```

Jeżeli ją tak zapiszemy, będziemy musieli rozwinąć także wszystkie zmienne po prawej stronie reguły. Jeżeli A jest symbolem początkowym gramatyki, musimy zapisać wszystkie reguły gramatyki. Nie wszystkim udaje się ukończyć cały analizator składniowy na zajęciach, a skoro analizator nie działa, dostają 0 punktów. Możliwe jest jednak pisanie reguł analizatora przyrostowo, punkt po punkcie. W regule rozwijającej zmienną A początkowo ujmujemy punkty C i D w komentarze.

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

Teraz musimy rozwinąć tylko zmienną B i zmienne pojawiające się w jej rozwinięciu. Analizator się skompiluje i można go przetestować. Potem można przesunąć komentarz za zmienną C. Ujmowanie w komentarze jest dużo lepszym rozwiązaniem niż pomijanie reszty reguły, gdyż widać, że reguła ma dalsze, nie używane jeszcze części.

Kompilując częściowo zrealizowany analizator składniowy możemy napotkać na problemy związane z dyrektywą `%type` wskazującą zmienne, które nie zostały jeszcze rozwinięte w żadnej regule. Należy wówczas ująć dyrektywę w komentarze do czasu dopisania odp. reguł.

4 Ocena

Za każdy można dostać 1 punkt, czyli 15 punktów na zajęciach. Punkty będą przyznawane dopiero po rozmowie z prowadzącym.

5 Dane testowe — plik test.mod

```
(*********************************************************)
(* Program shows ASCII codes                             *)
(* Compilation:                                          *)
(*    m2c -all test.mod -o test                          *)
(* Running:                                              *)
(*    ./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
        year, month : CARDINAL;
        day : CARDINAL;
  END;

  PROCEDURE ListAscii(StartCode, EndCode: CARDINAL; Precision: CARDINAL);
  VAR
        i: CARDINAL;
        t1 : ARRAY[1 .. 10] OF CARDINAL;
        d : RECORD
              year:             CARDINAL;
              month, day        : CARDINAL;
        END;
BEGIN
        WriteString("ASCII codes");
        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 Wyjście analizatora składniowego dla 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
```

```
===== FOUND: CONST_DECL 'FromAscii'=====
;                       ;
ToAscii                 IDENT           ToAscii
=                       =
127                     INTEGER_CONST   127
===== FOUND: CONST_DECL 'ToAscii'=====
;                       ;
VAR                     KW_VAR
i                       IDENT           i
:                       :
CARDINAL                IDENT           CARDINAL
;                       ;
===== FOUND: VAR_DECL =====
fl                      IDENT           fl
:                       :
REAL                    IDENT           REAL
;                       ;
===== FOUND: VAR_DECL =====
t                       IDENT           t
:                       :
ARRAY                   KW_ARRAY
[                       [
1                       INTEGER_CONST   1
..                      RANGE
10                      INTEGER_CONST   10
]                       ]
OF                      KW_OF
CARDINAL                IDENT           CARDINAL
;                       ;
===== FOUND: VAR_DECL =====
d                       IDENT           d
:                       :
RECORD                  KW_RECORD
year                    IDENT           year
,                       ,
month                   IDENT           month
:                       :
CARDINAL                IDENT           CARDINAL
;                       ;
day                     IDENT           day
:                       :
CARDINAL                IDENT           CARDINAL
;                       ;
END                     KW_END
;                       ;
===== FOUND: VAR_DECL =====
PROCEDURE               KW_PROCEDURE
ListAscii               IDENT           ListAscii
(                       (
StartCode               IDENT           StartCode
,                       ,
EndCode                 IDENT           EndCode
:                       :
CARDINAL                IDENT           CARDINAL
;                       ;
===== FOUND: FP_SECTION =====
Precision               IDENT           Precision
:                       :
CARDINAL                IDENT           CARDINAL
)                       )
===== FOUND: FP_SECTION =====
;                       ;
===== FOUND: PROC_HEAD 'ListAscii'=====
VAR                     KW_VAR
i                       IDENT           i
:                       :
```

```
CARDINAL                IDENT           CARDINAL
;                       ;
===== FOUND: VAR_DECL =====
t1                      IDENT           t1
:                       :
ARRAY                   KW_ARRAY
[                       [
1                       INTEGER_CONST   1
..                      RANGE
10                      INTEGER_CONST   10
]                       ]
OF                      KW_OF
CARDINAL                IDENT           CARDINAL
;                       ;
===== FOUND: VAR_DECL =====
d                       IDENT           d
:                       :
RECORD                  KW_RECORD
year                    IDENT           year
:                       :
CARDINAL                IDENT           CARDINAL
;                       ;
month                   IDENT           month
,                       ,
day                     IDENT           day
:                       :
CARDINAL                IDENT           CARDINAL
;                       ;
END                     KW_END
;                       ;
===== FOUND: VAR_DECL =====
BEGIN                   KW_BEGIN
WriteString             IDENT           WriteString
(                       (
"ASCII codes"           STRING_CONST    "ASCII codes"
)                       )
===== FOUND: PROCEDURE_CALL 'WriteString'=====
;                       ;
WriteLn                 IDENT           WriteLn
;                       ;
===== FOUND: PROCEDURE_CALL '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
)                       )
===== FOUND: PROCEDURE_CALL 'WriteCard'=====
;                       ;
Write                   IDENT           Write
(                       (
' '                     CHAR_CONST      ' '
)                       )
===== FOUND: PROCEDURE_CALL 'Write'=====
;                       ;
Write                   IDENT           Write
(                       (
CHR                     IDENT           CHR
(                       (
```

```
i                       IDENT           i
)                       )
)                       )
===== FOUND: PROCEDURE_CALL 'Write'=====
;                       ;
WriteLn                 IDENT           WriteLn
END                     KW_END
===== FOUND: PROCEDURE_CALL 'WriteLn'=====
===== FOUND: FOR_STATEMENT 'i'=====
;                       ;
t1                      IDENT           t1
[                       [
0                       INTEGER_CONST   0
]                       ]
:=                      ASSIGN
t                       IDENT           t
[                       [
0                       INTEGER_CONST   0
]                       ]
;                       ;
===== FOUND: ASSIGNMENT 't1'=====
d                       IDENT           d
.                       .
year                    IDENT           year
:=                      ASSIGN
2018                    INTEGER_CONST   2018
END                     KW_END
===== FOUND: ASSIGNMENT 'd'=====
ListAscii               IDENT           ListAscii
===== FOUND: PROC_DECL 'ListAscii'=====
;                       ;
BEGIN                   KW_BEGIN
fl                      IDENT           fl
:=                      ASSIGN
1.1                     FLOAT_CONST     1.1
+                       +
1.0E-2                  FLOAT_CONST     1.0E-2
+                       +
1.0E+2                  FLOAT_CONST     1.0E+2
+                       +
1.0E1                   FLOAT_CONST     1.0E1
;                       ;
===== FOUND: ASSIGNMENT 'fl'=====
IF                      KW_IF
(                       (
fl                      IDENT           fl
<=                      LE
11.11                   FLOAT_CONST     11.11
)                       )
AND                     KW_AND
(                       (
fl                      IDENT           fl
>=                      GE
1.111E1                 FLOAT_CONST     1.111E1
)                       )
THEN                    KW_THEN
WriteString             IDENT           WriteString
(                       (
"As expected!"          STRING_CONST    "As expected!"
)                       )
===== FOUND: PROCEDURE_CALL 'WriteString'=====
ELSE                    KW_ELSE
WriteString             IDENT           WriteString
(                       (
"Gosh!"                 STRING_CONST    "Gosh!"
)                       )
```

```
===== FOUND: PROCEDURE_CALL 'WriteString'=====
END                     KW_END
===== FOUND: IF_STATEMENT =====
;                       ;
WriteLn                 IDENT           WriteLn
;                       ;
===== FOUND: PROCEDURE_CALL 'WriteLn'=====
i                       IDENT           i
:=                      ASSIGN
1                       INTEGER_CONST   1
;                       ;
===== FOUND: ASSIGNMENT 'i'=====
WHILE                   KW_WHILE
i                       IDENT           i
<                       <
5                       INTEGER_CONST   5
DO                      KW_DO
WriteLn                 IDENT           WriteLn
(                       (
i                       IDENT           i
)                       )
===== FOUND: PROCEDURE_CALL 'WriteLn'=====
;                       ;
i                       IDENT           i
:=                      ASSIGN
i                       IDENT           i
+                       +
1                       INTEGER_CONST   1
END                     KW_END
===== FOUND: ASSIGNMENT 'i'=====
===== FOUND: WHILE_STATEMENT =====
;                       ;
REPEAT                  KW_REPEAT
WriteLn                 IDENT           WriteLn
(                       (
i                       IDENT           i
)                       )
===== FOUND: PROCEDURE_CALL 'WriteLn'=====
;                       ;
i                       IDENT           i
:=                      ASSIGN
i                       IDENT           i
-                       -
1                       INTEGER_CONST   1
UNTIL                   KW_UNTIL
===== FOUND: ASSIGNMENT 'i'=====
i                       IDENT           i
=                       =
1                       INTEGER_CONST   1
;                       ;
===== FOUND: REPEAT_StATEMENT =====
LOOP                    KW_LOOP
WriteLn                 IDENT           WriteLn
(                       (
"Spam"                  STRING_CONST    "Spam"
)                       )
===== FOUND: PROCEDURE_CALL 'WriteLn'=====
END                     KW_END
===== FOUND: LOOP_STATEMENT =====
;                       ;
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!"
)                       )
===== FOUND: PROCEDURE_CALL 'WriteLn'=====
|                       |
'A'                     CHAR_CONST      'A'
,                       ,
'a'                     CHAR_CONST      'a'
:                       :
Writeln                 IDENT           Writeln
(                       (
"Yes?"                  STRING_CONST    "Yes?"
)                       )
===== FOUND: PROCEDURE_CALL 'Writeln'=====
ELSE                    KW_ELSE
Writeln                 IDENT           Writeln
(                       (
"O!"                    STRING_CONST    "O!"
)                       )
===== FOUND: PROCEDURE_CALL 'Writeln'=====
END                     KW_END
===== FOUND: CASE_STATEMENT =====
;                       ;
t                       IDENT           t
[                       [
10                      INTEGER_CONST   10
]                       ]
:=                      ASSIGN
10                      INTEGER_CONST   10
;                       ;
===== FOUND: ASSIGNMENT 't'=====
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
===== FOUND: ASSIGNMENT 't'=====
===== FOUND: FOR_STATEMENT 'i'=====
;                       ;
d                       IDENT           d
.                       .
year                    IDENT           year
:=                      ASSIGN
```

```
2018                    INTEGER_CONST    2018
;                       ;
===== FOUND: ASSIGNMENT 'd'=====
d                       IDENT            d
.                       .
day                     IDENT            day
:=                      ASSIGN
1                       INTEGER_CONST    1
;                       ;
===== FOUND: ASSIGNMENT 'd'=====
d                       IDENT            d
.                       .
month                   IDENT            month
:=                      ASSIGN
d                       IDENT            d
.                       .
day                     IDENT            day
*                       *
10                      INTEGER_CONST    10
END                     KW_END
===== FOUND: ASSIGNMENT 'd'=====
test                    IDENT            test
.                       .
===== FOUND: PROGRAM_MODULE 'test'=====
```