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 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:

A. import modules declaration (IMPORT)

B. declaration of a constant (CONST_DECL)

C. declaration of a variable(VAR_DECL)

D. formal parameter section (FP_SECTION)

E. procedure header (PROC_HEAD)

F. procedure call (PROCEDURE_CALL)

G. for loop (FOR_STATEMENT)

H. procedure declaration (PROC_DECL)

I. assignment (ASSIGNMENT)

J. conditional (IF_STATEMENT)

K. while loop (WHILE_STATEMENT)

L. repeat until loop (REPEAT_STATEMENT)

M. endless loop (LOOP_STATEMENT)

N. case statement (CASE_STATEMENT)

O. 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:

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

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:

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

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

```
(********************************************************)
(* 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;
         FOR i := FromAscii TO ToAscii DO
               WriteCard(i, 3);
               Write(' ');
```

```
            Write(CHR(i));
            WriteLn
        END;
        t1[0] := t[0];
        d.year := 2018
    END ListAscii;

BEGIN
    fl := 1.1 + 1.0E-2 + 1.0E+2 + 1.0E1; (* real numbers *)
    IF (fl <= 11.11) AND (fl >= 1.111E1) THEN
      WriteString("As expected!")
    ELSE
      WriteString("Gosh!")
    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("Yes?")
    ELSE
          Writeln("O!")
    END;
    t[10] := 10;
    FOR i := 9 DOWNTO 1 DO t[i] := t[i+1] * i * i END;
    d.year := 2018; d.day := 1;
    d.month := d.day * 10
END test.
```

6 Output of the Parser for test.mod

```
First and last name
yytext              Token type      Token value as string

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
;                   ;
===== FOUND: IMPORT 'InOut'=====
CONST               KW_CONST
FromAscii           IDENT           FromAscii
=                   =
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'=====
```