Exercise P2. Parser for Simplified Turbo Pascal

1 Aim of the Excercise

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 eNauczanie 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 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:

```
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.pas

```
Program Testing;
(*Uses crt, dos;*)
Const (* range of displayed characters *)
        minASCII = 30;
        maxASCII = 255;
        tekst = 'test string';
Var
   c : Char;
   r : real;
   i, i1, _i, _00 : Integer;
   t : array[1..10] of integer;
   d :  record
           year, month : integer;
           day         :  integer
        end;

Procedure Empty_Without_Parameters;
Begin
End;

Function Empty_With_Parameters( a : Integer, c : Char, r : Real ) : Integer;
Begin
End;

Procedure With_Declarations;
Const
   r1 = 12.34;
   r2 = 0.56;
   r3 = 78.0;
Var
   s : String;
   t : array[1..10] of integer;
   d :  record
           year, month : integer;
```

```
            day         :   integer
         end;
Begin
End;


Begin (* main block *)
   Empty_Without_Parameters;
   Empty_With_Parameters( 123, 'c', 12.34 );
   ClrScr; (* intro opn clear screen *)
   Writeln( 'Kody ASCII (30-255):' );
   For i := minASCII To maxASCII Do (* display of given ASCII codes *)
      Write( Chr( i ), '   ' );
   ReadKey; (* wait for a key press *)
   i := ( i1 + 3 ) * _00;
   (* conditional instruction *)
   if  a > 10
      then
      b := a;
   if ( a > 1 )
      then
      b := a
   else
      b := 1;
   if ( a > b )
      then
      if ( a > c )
         then
         m := a
      else
         m := c
      else
         if ( b > c )
            then
            m := b
         else
            m := c;
   t[10] := 1;
   for i := 9 downto 1 do t[i] := t[i+1] * i * i;
   d.year := 2018;
   d.day := 1;
   d.month := d.day * 10;
End.
```

6 Parser's Output for test.pas

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

Program             KW_PROGRAM
Testing             IDENT           Testing
;                   ;
===== FOUND: PROGRAM_NAME 'Testing'=====
Const               KW_CONST
minASCII            IDENT           minASCII
=                   =
30                  INTEGER_CONST   30
===== FOUND: CONST 'minASCII'=====
;                   ;
maxASCII            IDENT           maxASCII
=                   =
255                 INTEGER_CONST   255
===== FOUND: CONST 'maxASCII'=====
;                   ;
```

```
tekst                IDENT              tekst
=                    =
'test string'        STRING_CONST       'test string'
===== FOUND: CONST 'tekst'=====
;                    ;
Var                  KW_VAR
===== FOUND: CONST_SECT =====
c                    IDENT              c
:                    :
Char                 KW_CHAR
===== FOUND: VAR =====
;                    ;
r                    IDENT              r
:                    :
real                 KW_REAL
===== FOUND: VAR =====
;                    ;
i                    IDENT              i
,                    ,
i1                   IDENT              i1
,                    ,
_i                   IDENT              _i
,                    ,
_00                  IDENT              _00
:                    :
Integer              KW_INTEGER
===== FOUND: VAR =====
;                    ;
t                    IDENT              t
:                    :
array                KW_ARRAY
[                    [
1                    INTEGER_CONST      1
..                   RANGE
10                   INTEGER_CONST      10
]                    ]
of                   KW_OF
integer              KW_INTEGER
===== FOUND: VAR =====
;                    ;
d                    IDENT              d
:                    :
record               KW_RECORD
year                 IDENT              year
,                    ,
month                IDENT              month
:                    :
integer              KW_INTEGER
;                    ;
day                  IDENT              day
:                    :
integer              KW_INTEGER
end                  KW_END
===== FOUND: VAR =====
;                    ;
Procedure            KW_PROCEDURE
===== FOUND: VAR_SECT =====
Empty_Without_ParameIDENT               Empty_Without_Parameters
;                    ;
===== FOUND: FUN_HEAD 'Empty_Without_Parameters'=====
Begin                KW_BEGIN
End                  KW_END
===== FOUND: BLOCK =====
===== FOUND: PROCEDURE 'Empty_Without_Parameters'=====
;                    ;
Function             KW_FUNCTION
```

```
Empty_With_ParameterIDENT           Empty_With_Parameters
(                   (
a                   IDENT           a
:                   :
Integer             KW_INTEGER
===== FOUND: FORM_PARAM =====
,                   ,
c                   IDENT           c
:                   :
Char                KW_CHAR
===== FOUND: FORM_PARAM =====
,                   ,
r                   IDENT           r
:                   :
Real                KW_REAL
===== FOUND: FORM_PARAM =====
)                   )
===== FOUND: FUN_HEAD 'Empty_With_Parameters'=====
:                   :
Integer             KW_INTEGER
;                   ;
Begin               KW_BEGIN
End                 KW_END
===== FOUND: BLOCK =====
===== FOUND: FUNCTION 'Empty_With_Parameters'=====
;                   ;
Procedure           KW_PROCEDURE
With_Declarations   IDENT           With_Declarations
;                   ;
===== FOUND: FUN_HEAD 'With_Declarations'=====
Const               KW_CONST
r1                  IDENT           r1
=                   =
12.34               FLOAT_CONST     12.34
===== FOUND: CONST 'r1'=====
;                   ;
r2                  IDENT           r2
=                   =
0.56                FLOAT_CONST     0.56
===== FOUND: CONST 'r2'=====
;                   ;
r3                  IDENT           r3
=                   =
78.0                FLOAT_CONST     78.0
===== FOUND: CONST 'r3'=====
;                   ;
Var                 KW_VAR
===== FOUND: CONST_SECT =====
s                   IDENT           s
:                   :
String              KW_STRING
===== FOUND: VAR =====
;                   ;
t                   IDENT           t
:                   :
array               KW_ARRAY
[                   [
1                   INTEGER_CONST   1
..                  RANGE
10                  INTEGER_CONST   10
]                   ]
of                  KW_OF
integer             KW_INTEGER
===== FOUND: VAR =====
;                   ;
d                   IDENT           d
```

```
:                        :
record                   KW_RECORD
year                     IDENT              year
,                        ,
month                    IDENT              month
:                        :
integer                  KW_INTEGER
;                        ;
day                      IDENT              day
:                        :
integer                  KW_INTEGER
end                      KW_END
===== FOUND: VAR =====
;                        ;
Begin                    KW_BEGIN
===== FOUND: VAR_SECT =====
End                      KW_END
===== FOUND: BLOCK =====
===== FOUND: PROCEDURE 'With_Declarations'=====
;                        ;
Begin                    KW_BEGIN
Empty_Without_ParameIDENT               Empty_Without_Parameters
;                        ;
===== FOUND: FUNCT_CALL 'Empty_Without_Parameters'=====
Empty_With_ParameterIDENT               Empty_With_Parameters
(                        (
123                      INTEGER_CONST      123
===== FOUND: ACT_PARAM =====
,                        ,
'c'                      STRING_CONST       'c'
===== FOUND: ACT_PARAM =====
,                        ,
12.34                    FLOAT_CONST        12.34
===== FOUND: ACT_PARAM =====
)                        )
===== FOUND: FUNCT_CALL 'Empty_With_Parameters'=====
;                        ;
ClrScr                   IDENT              ClrScr
;                        ;
===== FOUND: FUNCT_CALL 'ClrScr'=====
Writeln                  IDENT              Writeln
(                        (
'Kody ASCII (30-255)STRING_CONST       'Kody ASCII (30-255):'
===== FOUND: ACT_PARAM =====
)                        )
===== FOUND: FUNCT_CALL 'Writeln'=====
;                        ;
For                      KW_FOR
i                        IDENT              i
:=                       ASSIGN
minASCII                 IDENT              minASCII
To                       KW_TO
maxASCII                 IDENT              maxASCII
Do                       KW_DO
Write                    IDENT              Write
(                        (
Chr                      IDENT              Chr
(                        (
i                        IDENT              i
)                        )
===== FOUND: FUNCT_CALL 'i'=====
===== FOUND: ACT_PARAM =====
===== FOUND: FUNCT_CALL 'Chr'=====
===== FOUND: ACT_PARAM =====
,                        ,
'   '                    STRING_CONST       '   '
===== FOUND: ACT_PARAM =====
)                      )
===== FOUND: FUNCT_CALL 'Write'=====
===== FOUND: FOR_INSTR =====
;                      ;
ReadKey                IDENT              ReadKey
;                      ;
===== FOUND: FUNCT_CALL 'ReadKey'=====
i                      IDENT              i
:=                     ASSIGN
(                      (
i1                     IDENT              i1
+                      +
3                      INTEGER_CONST      3
)                      )
*                      *
_00                    IDENT              _00
;                      ;
===== FOUND: ASSIGN_INSTR 'i'=====
if                     KW_IF
a                      IDENT              a
>                      >
10                     INTEGER_CONST      10
then                   KW_THEN
b                      IDENT              b
:=                     ASSIGN
a                      IDENT              a
;                      ;
===== FOUND: ASSIGN_INSTR 'b'=====
===== FOUND: IF_INSTR =====
if                     KW_IF
(                      (
a                      IDENT              a
>                      >
1                      INTEGER_CONST      1
)                      )
then                   KW_THEN
b                      IDENT              b
:=                     ASSIGN
a                      IDENT              a
else                   KW_ELSE
===== FOUND: ASSIGN_INSTR 'b'=====
b                      IDENT              b
:=                     ASSIGN
1                      INTEGER_CONST      1
;                      ;
===== FOUND: ASSIGN_INSTR 'b'=====
===== FOUND: IF_INSTR =====
if                     KW_IF
(                      (
a                      IDENT              a
>                      >
b                      IDENT              b
)                      )
then                   KW_THEN
if                     KW_IF
(                      (
a                      IDENT              a
>                      >
c                      IDENT              c
)                      )
then                   KW_THEN
m                      IDENT              m
:=                     ASSIGN
a                      IDENT              a
else                   KW_ELSE
```

```
===== FOUND: ASSIGN_INSTR 'm'=====
m                       IDENT             m
:=                      ASSIGN
c                       IDENT             c
else                    KW_ELSE
===== FOUND: ASSIGN_INSTR 'm'=====
===== FOUND: IF_INSTR =====
if                      KW_IF
(                       (
b                       IDENT             b
>                       >
c                       IDENT             c
)                       )
then                    KW_THEN
m                       IDENT             m
:=                      ASSIGN
b                       IDENT             b
else                    KW_ELSE
===== FOUND: ASSIGN_INSTR 'm'=====
m                       IDENT             m
:=                      ASSIGN
c                       IDENT             c
;                       ;
===== FOUND: ASSIGN_INSTR 'm'=====
===== FOUND: IF_INSTR =====
===== FOUND: IF_INSTR =====
t                       IDENT             t
[                       [
10                      INTEGER_CONST     10
]                       ]
:=                      ASSIGN
1                       INTEGER_CONST     1
;                       ;
===== FOUND: ASSIGN_INSTR '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
;                       ;
===== FOUND: ASSIGN_INSTR 't'=====
===== FOUND: FOR_INSTR =====
d                       IDENT             d
.                       .
year                    IDENT             year
:=                      ASSIGN
2018                    INTEGER_CONST     2018
;                       ;
===== FOUND: ASSIGN_INSTR 'd'=====
d                   IDENT           d
.                   .
day                 IDENT           day
:=                  ASSIGN
1                   INTEGER_CONST   1
;                   ;
===== FOUND: ASSIGN_INSTR 'd'=====
d                   IDENT           d
.                   .
month               IDENT           month
:=                  ASSIGN
d                   IDENT           d
.                   .
day                 IDENT           day
*                   *
10                  INTEGER_CONST   10
;                   ;
===== FOUND: ASSIGN_INSTR 'd'=====
End                 KW_END
===== FOUND: BLOCK =====
.                   .
===== FOUND: Complete program =====
```