Exercise P2. Parser for Simplified C Programming Language

1 Aim of the Exercise

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 the simplified programming language C
- 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 the C 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`
- `common.h` – header file defining the greatest length of strings
- `c.y` – skeletal parser with comments, and with an already defined function `found`
- `test.c` – 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. variable declaraction (VAR)

B. function header (FUN_HEAD)

C. block (BLOCK)

D. function definition (S_FUNCTION)

E. formal parameter (FORM_PARAM)

F. declaration list (DECL_LIST)

G. function call (FUN_CALL)

H. actual parameter (ACT_PARAM)

I. assignment (ASSIGNMENT)

J. incrementation (INCR)

K. for loop (FOR_INSTR)

L. conditional instruction (IF_INSTR)

M. while loop (WHILE_INSTR)

N. do while loop (DO_WHILE)

O. conditional expression (COND_EXPR)

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.c

```
// Test program for C
int a;                          // declaration of one variable
float a1, _b, _00;              // declaration of 3 variables
double PI = 3.1415926;          // declaration of variable with initialization
unsigned char c;
int fromASCII = 128, toASCII = 255;
int t[10];
struct data {
  int year;
  int month, day;
} d;
void EmptyFunction( void )
{
}
int EmptyFunctionWithParameters( int a, double d )
{
}
float FunctionWithDeclarationOfVariables( double d )
{ // declaractio of variables
        int a;
        double half = .5;
        int t[7];
        struct data {
          int year, month;
          int day;
        } d1;
}
int x1 = fromASCII + 2 * ( 20 +  toASCII );
double realTest = 12.34 + .56 + 78.;
void main( void )
{
        int a = 1, b, c, m;
        int t[3];
        struct data {
```

```
          int day, month, year;
        } d;

        EmptyFunction();
        EmptyFunctionWithParameters( "x", 123, 12.34 );
        printf( "\n\n\nExtended ASCII codes\n\n" );
        // for loop
        for ( uc = fromASCII; uc <= toASCII; uc1++ )
        {
                int a;
                int t[2];
                t[0] = 1; t[1] = t[0];
                printf( "%3d:%2c", uc, uc );
                printf(",%d\n",t[1]);
                d.day = 1;
        }
        // conditional instruction
        if ( a > 10 )
                b = a;
        if ( a > 1 )
                b = a;
        else
                b = 1;
        if ( a > b )
                if ( a > c )
                        m = a;
                else
                        m = c;
        else
                if ( b > c )
                        m = b;
                else
                        m = c;
        while (a > 1)
                a = a - 2;
        d.year = 2010;
        do {
          a++; d.year++;
        } while (a < 1);
        m = a > b ? (a > c ? a : c) : (b > c ? b : c);
}
```

6 Parser's Output for test.c

```
Author: first nad last name
yytext              Token type      Token value as string

int                 KW_INT
a                   IDENT           a
;                   ;
======== FOUND: VAR 'a' ========
float               KW_FLOAT
a1                  IDENT           a1
,                   ,
======== FOUND: VAR 'a1' ========
_b                  IDENT           _b
,                   ,
======== FOUND: VAR '_b' ========
_00                 IDENT           _00
;                   ;
======== FOUND: VAR '_00' ========
double              KW_DOUBLE
PI                  IDENT           PI
=                   =
```

```
3.1415926               FLOAT_CONST     3.1415926
;                       ;
======== FOUND: VAR 'PI' ========
unsigned                KW_UNSIGNED
char                    KW_CHAR
c                       IDENT           c
;                       ;
======== FOUND: VAR 'c' ========
int                     KW_INT
fromASCII               IDENT           fromASCII
=                       =
128                     INTEGER_CONST   128
,                       ,
======== FOUND: VAR 'fromASCII' ========
toASCII                 IDENT           toASCII
=                       =
255                     INTEGER_CONST   255
;                       ;
======== FOUND: VAR 'toASCII' ========
int                     KW_INT
t                       IDENT           t
[                       [
10                      INTEGER_CONST   10
]                       ]
;                       ;
======== FOUND: VAR 't' ========
struct                  KW_STRUCT
data                    IDENT           data
{                       {
int                     KW_INT
year                    IDENT           year
;                       ;
======== FOUND: VAR 'year' ========
int                     KW_INT
month                   IDENT           month
,                       ,
======== FOUND: VAR 'month' ========
day                     IDENT           day
;                       ;
======== FOUND: VAR 'day' ========
}                       }
d                       IDENT           d
;                       ;
======== FOUND: VAR 'd' ========
void                    KW_VOID
EmptyFunction           IDENT           EmptyFunction
(                       (
void                    KW_VOID
)                       )
======== FOUND: FUN_HEAD 'EmptyFunction' ========
{                       {
}                       }
======== FOUND: BLOCK  ========
======== FOUND: S_FUNCTION 'EmptyFunction' ========
int                     KW_INT
EmptyFunctionWithParIDENT           EmptyFunctionWithParameters
(                       (
int                     KW_INT
a                       IDENT           a
======== FOUND: FORM_PARAM 'a' ========
,                       ,
double                  KW_DOUBLE
d                       IDENT           d
======== FOUND: FORM_PARAM 'd' ========
)                       )
======== FOUND: FUN_HEAD 'EmptyFunctionWithParameters' ========
```

```
{                       {
}                       }
======== FOUND: BLOCK  ========
======== FOUND: S_FUNCTION 'EmptyFunctionWithParameters' ========
float                   KW_FLOAT
FunctionWithDeclaratIDENT               FunctionWithDeclarationOfVariables
(                       (
double                  KW_DOUBLE
d                       IDENT           d
======== FOUND: FORM_PARAM 'd' ========
)                       )
======== FOUND: FUN_HEAD 'FunctionWithDeclarationOfVariables' ========
{                       {
int                     KW_INT
======== FOUND: DECL_LIST  ========
a                       IDENT           a
;                       ;
======== FOUND: VAR 'a' ========
double                  KW_DOUBLE
======== FOUND: DECL_LIST  ========
half                    IDENT           half
=                       =
.5                      FLOAT_CONST     .5
;                       ;
======== FOUND: VAR 'half' ========
int                     KW_INT
======== FOUND: DECL_LIST  ========
t                       IDENT           t
[                       [
7                       INTEGER_CONST   7
]                       ]
;                       ;
======== FOUND: VAR 't' ========
struct                  KW_STRUCT
======== FOUND: DECL_LIST  ========
data                    IDENT           data
{                       {
int                     KW_INT
year                    IDENT           year
,                       ,
======== FOUND: VAR 'year' ========
month                   IDENT           month
;                       ;
======== FOUND: VAR 'month' ========
int                     KW_INT
day                     IDENT           day
;                       ;
======== FOUND: VAR 'day' ========
}                       }
d1                      IDENT           d1
;                       ;
======== FOUND: VAR 'd1' ========
}                       }
======== FOUND: BLOCK  ========
======== FOUND: S_FUNCTION 'FunctionWithDeclarationOfVariables' ========
int                     KW_INT
x1                      IDENT           x1
=                       =
fromASCII               IDENT           fromASCII
+                       +
2                       INTEGER_CONST   2
*                       *
(                       (
20                      INTEGER_CONST   20
+                       +
toASCII                 IDENT           toASCII
```

```
)                       )
;                       ;
======== FOUND: VAR 'x1' ========
double                  KW_DOUBLE
realTest                IDENT           realTest
=                       =
12.34                   FLOAT_CONST     12.34
+                       +
.56                     FLOAT_CONST     .56
+                       +
78.                     FLOAT_CONST     78.
;                       ;
======== FOUND: VAR 'realTest' ========
void                    KW_VOID
main                    IDENT           main
(                       (
void                    KW_VOID
)                       )
======== FOUND: FUN_HEAD 'main' ========
{                       {
int                     KW_INT
======== FOUND: DECL_LIST  ========
a                       IDENT           a
=                       =
1                       INTEGER_CONST   1
,                       ,
======== FOUND: VAR 'a' ========
b                       IDENT           b
,                       ,
======== FOUND: VAR 'b' ========
c                       IDENT           c
,                       ,
======== FOUND: VAR 'c' ========
m                       IDENT           m
;                       ;
======== FOUND: VAR 'm' ========
int                     KW_INT
======== FOUND: DECL_LIST  ========
t                       IDENT           t
[                       [
3                       INTEGER_CONST   3
]                       ]
;                       ;
======== FOUND: VAR 't' ========
struct                  KW_STRUCT
======== FOUND: DECL_LIST  ========
data                    IDENT           data
{                       {
int                     KW_INT
day                     IDENT           day
,                       ,
======== FOUND: VAR 'day' ========
month                   IDENT           month
,                       ,
======== FOUND: VAR 'month' ========
year                    IDENT           year
;                       ;
======== FOUND: VAR 'year' ========
}                       }
d                       IDENT           d
;                       ;
======== FOUND: VAR 'd' ========
EmptyFunction           IDENT           EmptyFunction
(                       (
)                       )
;                       ;
```

```
======== FOUND: FUN_CALL 'EmptyFunction' ========
EmptyFunctionWithParIDENT            EmptyFunctionWithParameters
(                    (
"x"                  STRING_CONST    "x"
======== FOUND: ACT_PARAM ========
,                    ,
123                  INTEGER_CONST   123
,                    ,
======== FOUND: ACT_PARAM ========
12.34                FLOAT_CONST     12.34
)                    )
======== FOUND: ACT_PARAM ========
;                    ;
======== FOUND: FUN_CALL 'EmptyFunctionWithParameters' ========
printf               IDENT           printf
(                    (
"\n\n\nExtended ASCISTRING_CONST    "\n\n\nExtended ASCII codes\n\n"
======== FOUND: ACT_PARAM ========
)                    )
;                    ;
======== FOUND: FUN_CALL 'printf' ========
for                  KW_FOR
(                    (
uc                   IDENT           uc
=                    =
fromASCII            IDENT           fromASCII
;                    ;
======== FOUND: ASSIGNMENT 'uc' ========
uc                   IDENT           uc
<=                   LE
toASCII              IDENT           toASCII
;                    ;
uc1                  IDENT           uc1
++                   INC
======== FOUND: INCR 'uc1' ========
)                    )
{                    {
int                  KW_INT
======== FOUND: DECL_LIST ========
a                    IDENT           a
;                    ;
======== FOUND: VAR 'a' ========
int                  KW_INT
======== FOUND: DECL_LIST ========
t                    IDENT           t
[                    [
2                    INTEGER_CONST   2
]                    ]
;                    ;
======== FOUND: VAR 't' ========
t                    IDENT           t
[                    [
0                    INTEGER_CONST   0
]                    ]
=                    =
1                    INTEGER_CONST   1
;                    ;
======== FOUND: ASSIGNMENT 't' ========
t                    IDENT           t
[                    [
1                    INTEGER_CONST   1
]                    ]
=                    =
t                    IDENT           t
[                    [
0                    INTEGER_CONST   0
```

```
]                       ]
;                       ;
======== FOUND: ASSIGNMENT 't' ========
printf                  IDENT           printf
(                       (
"%3d:%2c"               STRING_CONST    "%3d:%2c"
======== FOUND: ACT_PARAM ========
,                       ,
uc                      IDENT           uc
,                       ,
======== FOUND: ACT_PARAM ========
uc                      IDENT           uc
)                       )
======== FOUND: ACT_PARAM ========
;                       ;
======== FOUND: FUN_CALL 'printf' ========
printf                  IDENT           printf
(                       (
",%d\n"                 STRING_CONST    ",%d\n"
======== FOUND: ACT_PARAM ========
,                       ,
t                       IDENT           t
[                       [
1                       INTEGER_CONST   1
]                       ]
)                       )
======== FOUND: ACT_PARAM ========
;                       ;
======== FOUND: FUN_CALL 'printf' ========
d                       IDENT           d
.                       .
day                     IDENT           day
=                       =
1                       INTEGER_CONST   1
;                       ;
======== FOUND: ASSIGNMENT 'd' ========
}                       }
======== FOUND: BLOCK ========
======== FOUND: FOR_INSTR ========
if                      KW_IF
(                       (
a                       IDENT           a
>                       >
10                      INTEGER_CONST   10
)                       )
b                       IDENT           b
=                       =
a                       IDENT           a
;                       ;
======== FOUND: ASSIGNMENT 'b' ========
if                      KW_IF
======== FOUND: IF_INSTR ========
(                       (
a                       IDENT           a
>                       >
1                       INTEGER_CONST   1
)                       )
b                       IDENT           b
=                       =
a                       IDENT           a
;                       ;
======== FOUND: ASSIGNMENT 'b' ========
else                    KW_ELSE
b                       IDENT           b
=                       =
1                       INTEGER_CONST   1
```

```
;                       ;
======== FOUND: ASSIGNMENT 'b' ========
======== FOUND: IF_INSTR ========
if                      KW_IF
(                       (
a                       IDENT           a
>                       >
b                       IDENT           b
)                       )
if                      KW_IF
(                       (
a                       IDENT           a
>                       >
c                       IDENT           c
)                       )
m                       IDENT           m
=                       =
a                       IDENT           a
;                       ;
======== FOUND: ASSIGNMENT 'm' ========
else                    KW_ELSE
m                       IDENT           m
=                       =
c                       IDENT           c
;                       ;
======== FOUND: ASSIGNMENT 'm' ========
======== FOUND: IF_INSTR ========
else                    KW_ELSE
if                      KW_IF
(                       (
b                       IDENT           b
>                       >
c                       IDENT           c
)                       )
m                       IDENT           m
=                       =
b                       IDENT           b
;                       ;
======== FOUND: ASSIGNMENT 'm' ========
else                    KW_ELSE
m                       IDENT           m
=                       =
c                       IDENT           c
;                       ;
======== FOUND: ASSIGNMENT 'm' ========
======== FOUND: IF_INSTR ========
======== FOUND: IF_INSTR ========
while                   KW_WHILE
(                       (
a                       IDENT           a
>                       >
1                       INTEGER_CONST   1
)                       )
a                       IDENT           a
=                       =
a                       IDENT           a
-                       -
2                       INTEGER_CONST   2
;                       ;
======== FOUND: ASSIGNMENT 'a' ========
======== FOUND: WHILE_INSTR ========
d                       IDENT           d
.                       .
year                    IDENT           year
=                       =
2010                    INTEGER_CONST   2010
```

```
;                       ;
======== FOUND: ASSIGNMENT 'd' ========
do                      KW_DO
{                       {
a                       IDENT           a
++                      INC
======== FOUND: INCR 'a' ========
;                       ;
d                       IDENT           d
.                       .
year                    IDENT           year
++                      INC
======== FOUND: INCR 'd' ========
;                       ;
}                       }
======== FOUND: BLOCK ========
while                   KW_WHILE
(                       (
a                       IDENT           a
<                       <
1                       INTEGER_CONST   1
)                       )
;                       ;
======== FOUND: DO_WHILE ========
m                       IDENT           m
=                       =
a                       IDENT           a
>                       >
b                       IDENT           b
?                       ?
(                       (
a                       IDENT           a
>                       >
c                       IDENT           c
?                       ?
a                       IDENT           a
:                       :
c                       IDENT           c
)                       )
======== FOUND: COND_EXPR ========
:                       :
(                       (
b                       IDENT           b
>                       >
c                       IDENT           c
?                       ?
b                       IDENT           b
:                       :
c                       IDENT           c
)                       )
======== FOUND: COND_EXPR ========
;                       ;
======== FOUND: COND_EXPR ========
======== FOUND: ASSIGNMENT 'm' ========
}                       }
======== FOUND: BLOCK ========
======== FOUND: S_FUNCTION 'main' ========
```