Zajęcia P2. Analizator składniowy dla uproszczonego języka C

1 Cel ćwiczeń

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

- rozpoznawanie konstrukcji składniowych języka C
- 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, z tematu *Analiza składniowa*, należy pobrać następujące pliki:

- `Makefile` – potrzebny do kompilacji za pomocą polecenia `make`
- `common.h` – plik nagłówkowy zawierający określenie największej długości napisów
- `c.y` – szczątkowy analizator składniowy z komentarzami i zdefiniowaną funkcją `found`
- `test.c` – 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 zmiennej (VAR)

B. nagłówek funkcji (FUN_HEAD)

C. blok (BLOCK)

D. definicja funkcji (S_FUNCTION)

E. parameter formalny (FORM_PARAM)

F. lista deklaracji (DECL_LIST)

G. wywołanie funkcji (FUN_CALL)

H. parametr aktualny (ACT_PARAM)

I. przypisanie (ASSIGNMENT)

J. instrukcja zwiększania (INCR)

K. pętla for (FOR_INSTR)

L. instrukcja warunkowa (IF_INSTR)

M. pętla while (WHILE_INSTR)

N. pętla do while (DO_WHILE)

O. wyrażenie warunkowe (COND_EXPR)

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.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 Wyjście analizatora składniowego dla 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 ASCIISTRING_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' ========
```