

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_uzytkownika`, i podkatalog dla bieżącego ćwiczenia. Ze strony przedmiotu na platformie eNauzanie, 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` – szcztakowy 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:

```
1 A: B C D
2 ;
```

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.

```
1 A: B /* C D */
2 ;
```

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 `%t` 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

```
1 // Test program for C
2 int a;                                // declaration of one variable
3 float a1, _b, _00;                    // declaration of 3 variables
4 double PI = 3.1415926;                // declaration of variable with initialization
5 unsigned char c;
6 int fromASCII = 128, toASCII = 255;
7 int t[10];
8 struct data {
9     int year;
10    int month, day;
11 } d;
12 void EmptyFunction( void )
13 {
14 }
15 int EmptyFunctionWithParameters( int a, double d )
16 {
17 }
18 float FunctionWithDeclarationOfVariables( double d )
19 { // declaratio of variables
20     int a;
21     double half = .5;
22     int t[7];
23     struct data {
24         int year, month;
25         int day;
26     } d1;
27 }
28 int x1 = fromASCII + 2 * ( 20 + toASCII );
29 double realTest = 12.34 + .56 + 78.;
30 void main( void )
```

```

31 {
32     int a = 1, b, c, m;
33     int t[3];
34     struct data {
35         int day, month, year;
36     } d;
37
38     EmptyFunction();
39     EmptyFunctionWithParameters( "x", 123, 12.34 );
40     printf( "\n\nExtended ASCII codes\n\n" );
41     // for loop
42     for ( uc = fromASCII; uc <= toASCII; uc1++ )
43     {
44         int a;
45         int t[2];
46         t[0] = 1; t[1] = t[0];
47         printf( "%3d:%2c", uc, uc );
48         printf(",%d\n",t[1]);
49         d.day = 1;
50     }
51     // conditional instruction
52     if ( a > 10 )
53         b = a;
54     if ( a > 1 )
55         b = a;
56     else
57         b = 1;
58     if ( a > b )
59         if ( a > c )
60             m = a;
61         else
62             m = c;
63     else
64         if ( b > c )
65             m = b;
66         else
67             m = c;
68     while ( a > 1 )
69         a = a - 2;
70     d.year = 2010;
71     do {
72         a++; d.year++;
73     } while ( a < 1);
74     m = a > b ? ( a > c ? a : c ) : ( b > c ? b : c );
75 }

```

6 Wyjście analizatora składniowego dla test.c

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
;	;	

```

17 ===== FOUND: VAR ' _00 ' =====
18 double KW_DOUBLE
19 PI IDENT PI
20 =
21 3.1415926 FLOAT_CONST 3.1415926
22 ;
23 ===== FOUND: VAR ' PI ' =====
24 unsigned KW_UNSIGNED
25 char KW_CHAR
26 c IDENT c
27 ;
28 ===== FOUND: VAR ' c ' =====
29 int KW_INT
30 fromASCII IDENT fromASCII
31 =
32 128 INTEGER_CONST 128
33 ,
34 ===== FOUND: VAR ' fromASCII ' =====
35 toASCII IDENT toASCII
36 =
37 255 INTEGER_CONST 255
38 ;
39 ===== FOUND: VAR ' toASCII ' =====
40 int KW_INT
41 t IDENT t
42 [
43 10 INTEGER_CONST 10
44 ]
45 ;
46 ===== FOUND: VAR ' t ' =====
47 struct KW_STRUCT
48 data IDENT data
49 {
50 int KW_INT
51 year IDENT year
52 ;
53 ===== FOUND: VAR ' year ' =====
54 int KW_INT
55 month IDENT month
56 ,
57 ===== FOUND: VAR ' month ' =====
58 day IDENT day
59 ;
60 ===== FOUND: VAR ' day ' =====
61 }
62 d IDENT d
63 ;
64 ===== FOUND: VAR ' d ' =====
65 void KW_VOID
66 EmptyFunction IDENT EmptyFunction
67 (
68 void KW_VOID
69 )
70 ===== FOUND: FUN_HEAD ' EmptyFunction ' =====
71 {
72 }
73 ===== FOUND: BLOCK =====
74 ===== FOUND: S_FUNCTION ' EmptyFunction ' =====
75 int KW_INT
76 EmptyFunctionWithPar IDENT EmptyFunctionWithParameters
77 (
78 int KW_INT
79 a IDENT a
80 ===== FOUND: FORM_PARAM ' a ' =====
81 ,
82 double KW_DOUBLE

```

```

83 d IDENT d
84 ===== FOUND: FORM_PARAM 'd' =====
85 ) )
86 ===== FOUND: FUN_HEAD 'EmptyFunctionWithParameters' =====
87 { {
88 } }
89 ===== FOUND: BLOCK =====
90 ===== FOUND: S_FUNCTION 'EmptyFunctionWithParameters' =====
91 float KW_FLOAT
92 FunctionWithDeclaratIDENT FunctionWithDeclarationOfVariables
93 ( (
94 double KW_DOUBLE
95 d IDENT d
96 ===== FOUND: FORM_PARAM 'd' =====
97 ) )
98 ===== FOUND: FUN_HEAD 'FunctionWithDeclarationOfVariables' =====
99 { {
100 int KW_INT
101 ===== FOUND: DECL_LIST =====
102 a IDENT a
103 ; ;
104 ===== FOUND: VAR 'a' =====
105 double KW_DOUBLE
106 ===== FOUND: DECL_LIST =====
107 half IDENT half
108 = =
109 .5 FLOAT_CONST .5
110 ; ;
111 ===== FOUND: VAR 'half' =====
112 int KW_INT
113 ===== FOUND: DECL_LIST =====
114 t IDENT t
115 [ [
116 7 INTEGER_CONST 7
117 ] ]
118 ; ;
119 ===== FOUND: VAR 't' =====
120 struct KW_STRUCT
121 ===== FOUND: DECL_LIST =====
122 data IDENT data
123 { {
124 int KW_INT
125 year IDENT year
126 , ,
127 ===== FOUND: VAR 'year' =====
128 month IDENT month
129 ; ;
130 ===== FOUND: VAR 'month' =====
131 int KW_INT
132 day IDENT day
133 ; ;
134 ===== FOUND: VAR 'day' =====
135 } }
136 d1 IDENT d1
137 ; ;
138 ===== FOUND: VAR 'd1' =====
139 } }
140 ===== FOUND: BLOCK =====
141 ===== FOUND: S_FUNCTION 'FunctionWithDeclarationOfVariables' =====
142 int KW_INT
143 x1 IDENT x1
144 = =
145 fromASCII IDENT fromASCII
146 + +
147 2 INTEGER_CONST 2
148 * *

```

```

149 (
150 20 INTEGER_CONST 20
151 +
152 toASCII IDENT toASCII
153 )
154 ;
155 ===== FOUND: VAR 'x1' =====
156 double KW_DOUBLE
157 realTest IDENT realTest
158 =
159 12.34 FLOAT_CONST 12.34
160 +
161 .56 FLOAT_CONST .56
162 +
163 78. FLOAT_CONST 78.
164 ;
165 ===== FOUND: VAR 'realTest' =====
166 void KW_VOID
167 main IDENT main
168 (
169 void KW_VOID
170 )
171 ===== FOUND: FUN_HEAD 'main' =====
172 {
173 int KW_INT
174 ===== FOUND: DECL_LIST =====
175 a IDENT a
176 =
177 1 INTEGER_CONST 1
178 ,
179 ===== FOUND: VAR 'a' =====
180 b IDENT b
181 ,
182 ===== FOUND: VAR 'b' =====
183 c IDENT c
184 ,
185 ===== FOUND: VAR 'c' =====
186 m IDENT m
187 ;
188 ===== FOUND: VAR 'm' =====
189 int KW_INT
190 ===== FOUND: DECL_LIST =====
191 t IDENT t
192 [
193 3 INTEGER_CONST 3
194 ]
195 ;
196 ===== FOUND: VAR 't' =====
197 struct KW_STRUCT
198 ===== FOUND: DECL_LIST =====
199 data IDENT data
200 {
201 int KW_INT
202 day IDENT day
203 ,
204 ===== FOUND: VAR 'day' =====
205 month IDENT month
206 ,
207 ===== FOUND: VAR 'month' =====
208 year IDENT year
209 ;
210 ===== FOUND: VAR 'year' =====
211 }
212 d IDENT d
213 ;
214 ===== FOUND: VAR 'd' =====

```

```

215 EmptyFunction      IDENT      EmptyFunction
216 (                  (
217 )                  )
218 ;                  ;
219 ===== FOUND: FUN_CALL 'EmptyFunction' =====
220 EmptyFunctionWithParIDENT      EmptyFunctionWithParameters
221 (                  (
222 "x"                STRING_CONST  "x"
223 ===== FOUND: ACT_PARAM =====
224 ,                  ,
225 123                 INTEGER_CONST 123
226 ,                  ,
227 ===== FOUND: ACT_PARAM =====
228 12.34               FLOAT_CONST  12.34
229 )                  )
230 ===== FOUND: ACT_PARAM =====
231 ;                  ;
232 ===== FOUND: FUN_CALL 'EmptyFunctionWithParameters' =====
233 printf              IDENT      printf
234 (                  (
235 "\n\n\nExtended ASCIISTRING_CONST  "\n\n\nExtended ASCII codes\n\n"
236 ===== FOUND: ACT_PARAM =====
237 )                  )
238 ;                  ;
239 ===== FOUND: FUN_CALL 'printf' =====
240 for                 KW_FOR
241 (                  (
242 uc                  IDENT      uc
243 =                  =
244 fromASCII           IDENT      fromASCII
245 ;                  ;
246 ===== FOUND: ASSIGNMENT 'uc' =====
247 uc                  IDENT      uc
248 <=                  LE
249 toASCII             IDENT      toASCII
250 ;                  ;
251 uc1                 IDENT      uc1
252 ++                  INC
253 ===== FOUND: INCR 'uc1' =====
254 )                  )
255 {                  {
256 int                 KW_INT
257 ===== FOUND: DECL_LIST =====
258 a                   IDENT      a
259 ;                  ;
260 ===== FOUND: VAR 'a' =====
261 int                 KW_INT
262 ===== FOUND: DECL_LIST =====
263 t                   IDENT      t
264 [                  [
265 2                   INTEGER_CONST 2
266 ]                  ]
267 ;                  ;
268 ===== FOUND: VAR 't' =====
269 t                   IDENT      t
270 [                  [
271 0                   INTEGER_CONST 0
272 ]                  ]
273 =                  =
274 1                   INTEGER_CONST 1
275 ;                  ;
276 ===== FOUND: ASSIGNMENT 't' =====
277 t                   IDENT      t
278 [                  [
279 1                   INTEGER_CONST 1
280 ]                  ]

```

```

281 =
282 t IDENT t
283 [ [
284 0 INTEGER_CONST 0
285 ] ]
286 ;
287 ===== FOUND: ASSIGNMENT 't' =====
288 printf IDENT printf
289 ( (
290 "%3d:%2c" STRING_CONST "%3d:%2c"
291 ===== FOUND: ACT_PARAM =====
292 ,
293 uc IDENT uc
294 ,
295 ===== FOUND: ACT_PARAM =====
296 uc IDENT uc
297 ) )
298 ===== FOUND: ACT_PARAM =====
299 ;
300 ===== FOUND: FUN_CALL 'printf' =====
301 printf IDENT printf
302 ( (
303 ",%d\n" STRING_CONST ",%d\n"
304 ===== FOUND: ACT_PARAM =====
305 ,
306 t IDENT t
307 [ [
308 1 INTEGER_CONST 1
309 ] ]
310 ) )
311 ===== FOUND: ACT_PARAM =====
312 ;
313 ===== FOUND: FUN_CALL 'printf' =====
314 d IDENT d
315 .
316 day IDENT day
317 =
318 1 INTEGER_CONST 1
319 ;
320 ===== FOUND: ASSIGNMENT 'd' =====
321 }
322 ===== FOUND: BLOCK =====
323 ===== FOUND: FOR_INSTR =====
324 if KW_IF
325 ( (
326 a IDENT a
327 >
328 10 INTEGER_CONST 10
329 ) )
330 b IDENT b
331 =
332 a IDENT a
333 ;
334 ===== FOUND: ASSIGNMENT 'b' =====
335 if KW_IF
336 ===== FOUND: IF_INSTR =====
337 ( (
338 a IDENT a
339 >
340 1 INTEGER_CONST 1
341 ) )
342 b IDENT b
343 =
344 a IDENT a
345 ;
346 ===== FOUND: ASSIGNMENT 'b' =====

```



```

347 else                KW_ELSE
348 b                    IDENT          b
349 =                    =
350 1                    INTEGER_CONST  1
351 ;                    ;
352 ===== FOUND: ASSIGNMENT 'b' =====
353 ===== FOUND: IF_INSTR =====
354 if                    KW_IF
355 (                    (
356 a                    IDENT          a
357 >                    >
358 b                    IDENT          b
359 )                    )
360 if                    KW_IF
361 (                    (
362 a                    IDENT          a
363 >                    >
364 c                    IDENT          c
365 )                    )
366 m                    IDENT          m
367 =                    =
368 a                    IDENT          a
369 ;                    ;
370 ===== FOUND: ASSIGNMENT 'm' =====
371 else                KW_ELSE
372 m                    IDENT          m
373 =                    =
374 c                    IDENT          c
375 ;                    ;
376 ===== FOUND: ASSIGNMENT 'm' =====
377 ===== FOUND: IF_INSTR =====
378 else                KW_ELSE
379 if                    KW_IF
380 (                    (
381 b                    IDENT          b
382 >                    >
383 c                    IDENT          c
384 )                    )
385 m                    IDENT          m
386 =                    =
387 b                    IDENT          b
388 ;                    ;
389 ===== FOUND: ASSIGNMENT 'm' =====
390 else                KW_ELSE
391 m                    IDENT          m
392 =                    =
393 c                    IDENT          c
394 ;                    ;
395 ===== FOUND: ASSIGNMENT 'm' =====
396 ===== FOUND: IF_INSTR =====
397 ===== FOUND: IF_INSTR =====
398 while                KW_WHILE
399 (                    (
400 a                    IDENT          a
401 >                    >
402 1                    INTEGER_CONST  1
403 )                    )
404 a                    IDENT          a
405 =                    =
406 a                    IDENT          a
407 -                    -
408 2                    INTEGER_CONST  2
409 ;                    ;
410 ===== FOUND: ASSIGNMENT 'a' =====
411 ===== FOUND: WHILE_INSTR =====
412 d                    IDENT          d

```

```

413 .
414 year IDENT year
415 =
416 2010 INTEGER_CONST 2010
417 ;
418 ===== FOUND: ASSIGNMENT 'd' =====
419 do KW_DO
420 {
421 a IDENT a
422 ++ INC
423 ===== FOUND: INCR 'a' =====
424 ;
425 d IDENT d
426 .
427 year IDENT year
428 ++ INC
429 ===== FOUND: INCR 'd' =====
430 ;
431 }
432 ===== FOUND: BLOCK =====
433 while KW_WHILE
434 (
435 a IDENT a
436 <
437 1 INTEGER_CONST 1
438 )
439 ;
440 ===== FOUND: DO_WHILE =====
441 m IDENT m
442 =
443 a IDENT a
444 >
445 b IDENT b
446 ?
447 (
448 a IDENT a
449 >
450 c IDENT c
451 ?
452 a IDENT a
453 :
454 c IDENT c
455 )
456 ===== FOUND: COND_EXPR =====
457 :
458 (
459 b IDENT b
460 >
461 c IDENT c
462 ?
463 b IDENT b
464 :
465 c IDENT c
466 )
467 ===== FOUND: COND_EXPR =====
468 ;
469 ===== FOUND: COND_EXPR =====
470 ===== FOUND: ASSIGNMENT 'm' =====
471 }
472 ===== FOUND: BLOCK =====
473 ===== FOUND: S_FUNCTION 'main' =====

```