

Санкт-Петербургский политехнический университет Петра Великого
Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе

Дисциплина: Транслирующие системы

Тема: Транслятор операторов *while* языка *C*

Выполнил студент гр. 43501/3
Преподаватель

Мальцев М.С.
Цыган В.Н.

Санкт-Петербург
26 марта 2019 г.

1 Задание

Транслятор операторов *while* языка *C*:

Тип данных: `int`.

Условное выражение: арифметическое бесскобочное выражение, т.е. операции выполняются слева на право.

В теле цикла:

1. операторы присваивания вида $id =$ бесскобочное арифметическое выражение
2. вложенный оператор *while*

Выходной продукт:

1. текст на языке ассемблера A86
2. тетрады матрицы синтаксического дерева

2 Ход работы

Задание выданное преподавателем было изучено и рассмотрено с точки зрения реализуемости.

Было решено:

- Использовать язык ассемблера для архитектуры x86. В связи с тем, что эта архитектура наиболее распространена и был найден удобный транслятор из языка C в язык ассемблера x86. (<https://gcc.godbolt.org/>)
- Использовать только одну переменную. Это решение было принято в связи с тем, что при использовании нескольких переменных, помимо выявления этих переменных необходимо было бы их различать, что повлекло бы за собой необходимость самописной структуры данных. К этой структуре появляются требования, такие как расширяемость. Что влечёт достаточно большой объем кода, который никак не связан с языками *lex* и *yacc*. Учитывая указанную проблему, было решено отказаться от использования нескольких переменных, но со стороны части программы, которая отвечает за синтаксический и лексический разбор, обеспечить необходимую функциональность для возможности дальнейшей работы с ними.
- Отказаться от использования операций типа:
`int a, b, c, d;`

$a = b + c + 1 + d;$

Подобное решение было связано с тем, что операции, состоящие из нескольких переменных, во первых были невозможны из-за предыдущего пункта этого списка, а во вторых из-за того что после трансляции они генерируют достаточно сложный текст.

```
1 main:
2  push rbp
3  mov rbp, rsp
4  mov edx, DWORD PTR [rbp-4]
5  mov eax, DWORD PTR [rbp-8]
6  add eax, edx
7  lea edx, [rax+1]
8  mov eax, DWORD PTR [rbp-12]
9  add eax, edx
10 mov DWORD PTR [rbp-16], eax
11 mov eax, 0
12 pop rbp
13 ret
```

- Также было решено отказаться от вложенных циклов, в связи с тем, что основная сложность подобных конструкций это генерация меток, что не является интересной задачей, и слабо относится к языкам синтаксического и лексического разбора.

Учитывая все перечисленные требования и нюансы была разработана программа:

```
1  %{
2  #include <stdlib.h>
3  #include "y.tab.h"
4  %}
5
6  VAR [a-zA-Z][0-9a-zA-Z_]*
7  D [0-9]+
8
9  %%
10 "+" |
11 "-" |
12 "*" |
13 "/" |
14 "=" |
15 ";" |
16 "{" |
17 "}" |
18 "(" |
19 ")" { return yytext[0]; }
20
21 "while" { return WHILE_KEY; }
22
23 "int " { return DEF_INT; }
24
25 {VAR} { yylval.text = strdup(yytext); return VARIABLE; }
26
27 {D} { yylval.ival = atoi(yytext); return NUM; }
28
29 (\n) { return EOL; }
```

```

30
31 [ \t]    { }
32 (.)      { }
33
34 %%
35
36 #ifndef yywrap
37 int yywrap () { return 1; }
38 #endif
39
40
41 // "int "[ ]+{VAR}([ ]*=[ ]*{D})?";" { printf("DEF-"); ECHO; printf("-DEF"); }
42 // "while "([a-zA-Z0-9 +-]+)" { printf("WHILE-"); ECHO; printf("-WHILE "); }
43 // "{" { printf("open-"); ECHO; }
44 // ([ ]+)?{VAR}[ ]*=[ ]*({VAR}|{D})[ ]*([+-])?+[ ]*";" { printf("OPER-"); ECHO
45 //      ; printf("-OPER"); }
46 // "}" { ECHO; printf("-close"); }
47 // (\n) { ECHO; }

```

```

1
2 %union {
3     int ival;
4     char* text;
5 };
6
7 %token <text> VARIABLE
8 %token <ival> NUM
9 %token EOL
10
11 %type <ival> number
12 %type <ival> condition
13
14 %token DEF_INT WHILE_KEY
15
16 %start commands
17
18
19 %%
20 commands:
21     | EOL commands
22     | definition ';' EOL commands
23     | while_cycle commands
24     | operation ';' EOL commands
25     ;
26
27 definition: DEF_INT VARIABLE                                { setVar(0); }
28     | DEF_INT VARIABLE '=' number                          { setVar($4); }
29     ;
30
31 operation: VARIABLE '=' number                              { setVar($3); }
32     | VARIABLE '+' '+'                                       { incVal(1); }
33     | VARIABLE '+' '=' number                               { incVal($4); }
34     | VARIABLE '=' VARIABLE '+' number                     { incVal($5); }
35     | VARIABLE '-' '-'                                       { decVal(1); }
36     | VARIABLE '-' '=' number                               { decVal($4); }
37     | VARIABLE '=' VARIABLE '-' number                     { decVal($5); }
38     ;
39
40 while_cycle: WHILE_KEY '(' condition ')' EOL                { startWhile($3); }
41     '{' EOL

```

```

42         commands
43         '}'
44         ;
45
46 condition: VARIABLE '+' NUM          { $$ = (-1) * $3; }
47           | VARIABLE '-' NUM        { $$ = $3; }
48           ;
49
50 number:   NUM                        { $$ = $1; }
51           | '-' NUM                  { $$ = (-1) * $2; }
52           | number '+' number        { $$ = $1 + $3; }
53           | number '-' number        { $$ = $1 - $3; }
54           | number '*' number        { $$ = $1 * $3; }
55           | number '/' number        { $$ = $1 / $3; }
56           ;
57 %%
58
59
60 int varValue = 0;
61
62 setVar(int value) {
63     varValue = value;
64     printf("mov DWORD PTR [rbp-4], %d\n", value);
65 }
66
67 int getVarVal() {
68     return varValue;
69 }
70
71 startWhile(int expr) {
72     printf(".L3:\n");
73     printf("cmp DWORD PTR [rbp-4], %d\n", expr);
74     printf("je .L2\n");
75 }
76
77 endWhile() {
78     printf("jmp .L3\n");
79     printf(".L2:\n");
80 }
81
82 incVal(int val) {
83     printf("add DWORD PTR [rbp-4], %d\n", val);
84 }
85
86 decVal(int val) {
87     printf("sub DWORD PTR [rbp-4], %d\n", val);
88 }

```

```

1  #include <stdio.h>
2
3  /* —— Define external objects —— */
4
5  int yydebug = 0;    /* To trace parser, set yydebug = 1      */
6                      /* ... and call yacc with options -vtd */
7                      /* To not trace, set yydebug = 0      */
8                      /* ... and call yacc with option -d      */
9
10 /* You can use "yyerror" for your own messages */
11 yyerror (char *s)
12 {

```

```

13     fprintf( stderr , "?-%s\n", s );
14 }
15
16 /* ————— Define starting point ————— */
17
18 main ()
19 {
20     printf("push rbp\n");
21     printf("mov rbp, rsp\n");
22     int ret = yyparse();
23     printf("pop rbp\n");
24     printf("ret\n");
25     return ret;
26 }

```

В качестве входных данных было подано следующее:

```

1 int sum = 1;
2 while (sum - 10)
3 {
4     sum ++ ;
5     sum +=2 ;
6     sum = sum +5+1 ;
7 }

```

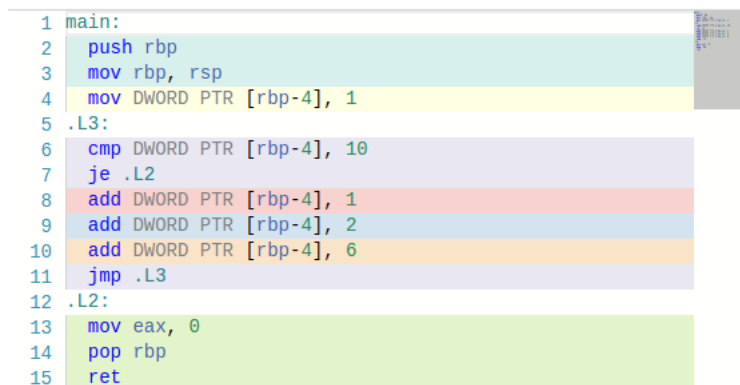
В результате работы программы получили следующее:

```

1 push rbp
2 mov rbp, rsp
3 mov DWORD PTR [rbp-4], 1
4 .L3:
5 cmp DWORD PTR [rbp-4], 10
6 je .L2
7 add DWORD PTR [rbp-4], 1
8 add DWORD PTR [rbp-4], 2
9 add DWORD PTR [rbp-4], 6
10 jmp .L3
11 .L2:
12 pop rbp
13 ret

```

Что соответствует результатам трансляции с помощью сторонней программы:



```

1 main:
2     push rbp
3     mov rbp, rsp
4     mov DWORD PTR [rbp-4], 1
5 .L3:
6     cmp DWORD PTR [rbp-4], 10
7     je .L2
8     add DWORD PTR [rbp-4], 1
9     add DWORD PTR [rbp-4], 2
10    add DWORD PTR [rbp-4], 6
11    jmp .L3
12 .L2:
13    mov eax, 0
14    pop rbp
15    ret

```

Следовательно, можно считать, что разработанная программа работает верно.

3 Возможные улучшения

Как в случае с использованием нескольких переменных, так и в случае вложенных циклов *while*, чтобы обеспечить их поддержку необходимо преодолеть ряд трудностей, которые связаны не с лексическим или синтаксическим разбором, а именно с генерацией кода на языке ассемблера.

В случае с вложенными циклами нужны уникальные значения для меток, по которым переходит программа во время итерации. В той реализации, которая предложена на данный момент метки называются одинаково. Что касается лексической и синтаксической обработки данных ситуаций, то с их стороны всё реализовано.

В случае с переменными необходимо определять, наличие или отсутствие объявления переменной указанной в выражении - это можно реализовать с помощью структуры данных, которая хранит в однозначном соответствии имя переменной и место в памяти, где расположено значение. Причём, структура должна быть расширяющейся, так как мы не можем заранее предсказать сколько переменных планирует использовать программист. В этом случае проблема отсутствия реализации в итоговой программе заключается в том, что подобная функциональность влечёт большой объем кода на языке C, слабо относящийся к языкам *lex* и *yacc*, изучение и работа с которыми ставятся в главные цели лабораторной работы.

4 Вывод

В результате выполнения лабораторной работы был написан простейший транслятор операторов *while* языка C. Для написания программы использовались генераторы синтаксического и лексического разбора *yacc* и *lex*. В ходе выполнения работы сначала было определено какую функциональность будет включать конечное приложение. Далее была разработана часть отвечающая за лексический разбор, в след за ней была написана часть, в задачи которой входит синтаксический разбор. Получившаяся программа была протестирована на различных входных данных. Полученные результаты соответствуют тому, что было описано в задании.

Проведенная работа позволила лучше понять принципы совместного использования генераторов для синтаксического и лексического разбора и принципы построения трансляторов для языков программирования. Также в ходе выполнения лабораторной были получены навыки разработки приложений на основе этих языков.

5 Используемая литература

- John Levine. Flex & Bison: Text Processing Tools. — O'Reilly Media, 2009

- Программирование лексического и синтаксического разбора на языках C, Lex и Yacc — А.В. Жуков, 2014