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 Мальцев М.С.
Преподаватель Цыган В.Н.

Санкт-Петербург
30 марта 2019 г.

1 Задание

Транслятор операторов *while* языка *C*:
Тип данных: int.
Условное выражение: арифметическое бесскобочнрое 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 ассембера A86
2. тетрады матрицы синтаксического дерева

2 Ход работы

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 ассембера x86. (https://gcc.godbolt.org/)
Учитывая перечисленные требования была разработана программа:

```
%{
#include <stdlib.h>
#include "y.tab.h"
%}

VAR [a-zA-Z][0-9a-zA-Z_]*
D [0-9]+

%%
"+" |
"-" |
"*" |
"/" |
"=" |
";" |
"{" |
"}" |
"(" |
")" { return yytext[0]; }

"while" { return WHILE_KEY; }

"int " { return DEF_INT; }

{VAR} { yylval.text = strdup(yytext); return VARIABLE; }

{D} { yylval.ival = atoi(yytext); return NUM; }
```

```
28
29 "main()" {yylval.text = strdup(yytext); return FUNC_MAIN; }
30
31 (\n)     { return EOL; }
32
33 [ \t]    { }
34 (.)      { }
35
36 %%
37
38 #ifndef yywrap
39 int yywrap () { return 1; }
40 #endif
```

```
1  %{
2  #include "./vector.h"
3  %}
4
5  %union {
6      int ival;
7      char* text;
8  };
9
10 %token <text> VARIABLE
11 %token <ival> NUM
12 %token EOL
13
14 %type <ival> number
15 %type <ival> condition
16
17 %token DEF_INT WHILE_KEY FUNC_MAIN
18
19 %start def_main
20
21
22 %%
23 def_main:
24               | EOL def_main
25               | DEF_INT FUNC_MAIN '{'          { initVector(); }
26                   commands
27               '}' def_main                     { freeVector(); }
28               ;
29
30 commands:
31               | EOL commands
32               | definition ';' EOL commands
33               | while_cycle commands
34               | operation ';' EOL commands
35               ;
36
37 definition: DEF_INT VARIABLE                   { initVar($2, 0); }
38               | DEF_INT VARIABLE '=' number    { initVar($2, $4); }
39               ;
40
41 operation:  VARIABLE '=' number                 { setVar($1, $3); }
42               | VARIABLE '=' VARIABLE          { varSetVar($1, $3); }
43               | VARIABLE '+' '+'               { incVal($1, 1); }
44               | VARIABLE '+' '=' number        { incVal($1, $4); }
45               | VARIABLE '-' '-'               { decVal($1, 1); }
46               | VARIABLE '-' '=' number        { decVal($1, $4); }
```

```
47              | VARIABLE '=' VARIABLE '+' number       { varAddNum($1, $3, $5); }
48              | VARIABLE '=' VARIABLE '-' number       { varAddNum($1, $3, (-1)*$5)
    ; }
49              | VARIABLE '=' VARIABLE '+' VARIABLE     { varAddVar($1, $3, $5); }
50              | VARIABLE '=' VARIABLE '-' VARIABLE     { varSubVar($1, $3, $5); }
51              ;
52
53  while_cycle: WHILE_KEY '(' VARIABLE condition ')' EOL  { startWhile($3, $4); }
54               '{' EOL
55               commands
56               '}'                                    { endWhile(); }
57               ;
58
59  condition:    '+' NUM                                { $$ = (-1) * $2; }
60              | '-' NUM                                { $$ = $2; }
61              ;
62
63  number:      NUM                                     { $$ = $1; }
64              | '-' NUM                                { $$ = (-1) * $2; }
65              | number '+' number                      { $$ = $1 + $3; }
66              | number '-' number                      { $$ = $1 - $3; }
67              | number '*' number                      { $$ = $1 * $3; }
68              | number '/' number                      { $$ = $1 / $3; }
69              ;
70  %%
71
72
73  int nestedLoop = 1;
74  int nextLoop = 0;
75  vector v;
76
77  initVector()
78  {
79      vector_init(&v);
80  }
81
82  freeVector()
83  {
84      vector_free(&v);
85  }
86
87  initVar(char* name, int value)
88  {
89      int index = vector_find_by_name(&v, name);
90      if (index != -1) {
91          printf("ERROR: redeclaration of '%s' \n", name);
92          exit(1);
93      }
94
95      vector_add(&v, name);
96
97      printf("mov DWORD PTR [rbp-%d], %d\n", vector_count(&v)*4, value);
98  }
99
100 setVar(char* name, int value)
101 {
102     int index = vector_find_by_name(&v, name);
103     if (index == -1) {
104         printf("ERROR: '%s' was not declared in this scope \n", name);
105         exit(1);
```

```
106     }
107
108     vector_set(&v, index, name);
109
110     printf("mov DWORD PTR [rbp-%d], %d\n", (index+1)*4, value);
111 }
112
113 startWhile(char* name, int expr)
114 {
115     int index = vector_find_by_name(&v, name);
116     if (index == -1) {
117         printf("ERROR: '%s' was not declared in this scope \n", name);
118         exit(1);
119     }
120
121     printf(".L%dD%d:\n", nestedLoop+1, nextLoop);
122     printf("cmp DWORD PRT [rbp-%d], %d\n", (index+1)*4, expr);
123     printf("je .L%dD%d\n", nestedLoop, nextLoop);
124
125     nestedLoop+=2;
126 }
127
128 endWhile()
129 {
130     nestedLoop-=2;
131
132     printf("jmp .L%dD%d\n", nestedLoop+1, nextLoop);
133     printf(".L%dD%d:\n", nestedLoop, nextLoop);
134
135     if (nestedLoop == 1){
136         nextLoop++;
137     }
138 }
139
140 incVal(char* name, int val)
141 {
142     int index = vector_find_by_name(&v, name);
143     if (index == -1) {
144         printf("ERROR: '%s' was not declared in this scope \n", name);
145         exit(1);
146     }
147     printf("add DWORD PTR [rbp-%d], %d\n", (index+1)*4, val);
148 }
149
150 decVal(char* name, int val)
151 {
152     int index = vector_find_by_name(&v, name);
153     if (index == -1) {
154         printf("ERROR: '%s' was not declared in this scope \n", name);
155         exit(1);
156     }
157     printf("sub DWORD PTR [rbp-%d], %d\n", (index+1)*4, val);
158 }
159
160 varAddNum(char* name1, char* name2, int num) {
161     int index1 = vector_find_by_name(&v, name1);
162     int index2 = vector_find_by_name(&v, name2);
163     if (index1 == -1) {
164         printf("ERROR: '%s' was not declared in this scope \n", name1);
165         exit(1);
```

```
166     }
167     if (index2 == -1) {
168         printf("ERROR: '%s' was not declared in this scope \n", name2);
169         exit(1);
170     }
171 
172     printf("mov eax, DWORD PTR [rbp-%d]\n", (index2+1)*4);
173     if (num >= 0) {
174         printf("add eax, %d\n", num);
175     } else {
176         printf("sub eax, %d\n", (-1)*num);
177     }
178     printf("DWORD PTR [rbp-%d], eax\n", (index1+1)*4);
179 }
180 
181 varSetVar(char* name1, char* name2) {
182     int index1 = vector_find_by_name(&v, name1);
183     int index2 = vector_find_by_name(&v, name2);
184     if (index1 == -1) {
185         printf("ERROR: '%s' was not declared in this scope \n", name1);
186         exit(1);
187     }
188     if (index2 == -1) {
189         printf("ERROR: '%s' was not declared in this scope \n", name2);
190         exit(1);
191     }
192 
193     printf("mov eax, DWORD PTR [rbp-%d]\n", (index2+1)*4);
194     printf("DWORD PTR [rbp-%d], eax\n", (index1+1)*4);
195 }
196 
197 varAddVar(char* name1, char* name2, char* name3) {
198     int index1 = vector_find_by_name(&v, name1);
199     int index2 = vector_find_by_name(&v, name2);
200     int index3 = vector_find_by_name(&v, name3);
201     if (index1 == -1) {
202         printf("ERROR: '%s' was not declared in this scope \n", name1);
203         exit(1);
204     }
205     if (index2 == -1) {
206         printf("ERROR: '%s' was not declared in this scope \n", name2);
207         exit(1);
208     }
209     if (index3 == -1) {
210         printf("ERROR: '%s' was not declared in this scope \n", name3);
211         exit(1);
212     }
213 
214     printf("mov edx, DWORD PTR [rbp-%d]\n", (index2+1)*4);
215     printf("mov eax, DWORD PTR [rbp-%d]\n", (index3+1)*4);
216     printf("add eax, edx\n");
217     printf("mov DWORD PTR [rbp-%d], eax\n", (index1+1)*4);
218 }
219 
220 varSubVar(char* name1, char* name2, char* name3) {
221     int index1 = vector_find_by_name(&v, name1);
222     int index2 = vector_find_by_name(&v, name2);
223     int index3 = vector_find_by_name(&v, name3);
224     if (index1 == -1) {
225         printf("ERROR: '%s' was not declared in this scope \n", name1);
```

```
226         exit(1);
227     }
228     if (index2 == -1) {
229         printf("ERROR: '%s' was not declared in this scope \n", name2);
230         exit(1);
231     }
232     if (index3 == -1) {
233         printf("ERROR: '%s' was not declared in this scope \n", name3);
234         exit(1);
235     }
236 
237     printf("mov edx, DWORD PTR [rbp-%d]\n", (index2+1)*4);
238     printf("mov eax, DWORD PTR [rbp-%d]\n", (index3+1)*4);
239     printf("sub eax, edx\n");
240     printf("mov DWORD PTR [rbp-%d], eax\n", (index1+1)*4);
241 }
```

```
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("mov eax, 0\n");
24     printf("pop rbp\n");
25     printf("ret\n");
26     return ret;
27 }
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  
4  #include "vector.h"
5  
6  void vector_init(vector *v)
7  {
8      v->names = NULL;
9      v->size = 0;
10     v->count = 0;
11 }
12 
13 int vector_count(vector *v)
14 {
15     return v->count;
```

```
16 }
17
18 void vector_add(vector *v, char *name)
19 {
20     if (v->size == 0) {
21         v->size = 10;
22         v->names = malloc(sizeof(char*) * v->size);
23         memset(v->names, '\0', sizeof(char) * v->size);
24     }
25
26     // condition to increase v->data:
27     // last slot exhausted
28     if (v->size == v->count) {
29         v->size *= 2;
30         v->names = realloc(v->names, sizeof(char*) * v->size);
31     }
32
33     v->names[v->count] = name;
34     v->count++;
35 }
36
37 void vector_set(vector *v, int index, char *name)
38 {
39     if (index >= v->count) {
40         return;
41     }
42
43     v->names[index] = name;
44 }
45
46 char* vector_get(vector *v, int index)
47 {
48     if (index >= v->count) {
49         return;
50     }
51
52     char* name = v->names[index];
53
54     return name;
55 }
56
57 int vector_find_by_name(vector *v, char *name)
58 {
59     int index = -1;
60     for (int i = 0; i < vector_count(v); i++) {
61         if (strcmp(vector_get(v, i), name) == 0) {
62             index = i;
63         }
64     }
65
66     return index;
67 }
68
69 void vector_delete(vector *v, int index)
70 {
71     if (index >= v->count) {
72         return;
73     }
74
75     v->names[index] = NULL;
```

```
76
77      int i, j;
78      char **newarrNames = (char**)malloc(sizeof(char*) * v->count * 2);
79      for (i = 0, j = 0; i < v->count; i++) {
80          if (v->names[i] != NULL) {
81              newarrNames[j] = v->names[i];
82              j++;
83          }
84      }
85      free(v->names);
86
87      v->names = newarrNames;
88      v->count--;
89  }
90
91  void vector_free(vector *v)
92  {
93      free(v->names);
94  }
95
96  int main1(void)
97  {
98      // vector v;
99      // vector_init(&v);
100
101     // vector_add(&v, "emil");
102     // vector_add(&v, "hannes");
103     // vector_add(&v, "lydia");
104     // vector_add(&v, "olle");
105     // vector_add(&v, "erik");
106
107     // int i;
108     // printf("first round:\n");
109     // for (i = 0; i < vector_count(&v); i++) {
110     //  printf("%s\n", vector_get(&v, i));
111     // }
112
113     // vector_delete(&v, 1);
114     // vector_delete(&v, 3);
115
116     // printf("second round:\n");
117     // for (i = 0; i < vector_count(&v); i++) {
118     //  printf("%s\n", vector_get(&v, i));
119     // }
120
121     // vector_free(&v);
122
123     return 0;
124 }
```

```
1  #ifndef VECTOR_H__
2  #define VECTOR_H__
3
4  typedef struct vector_ {
5      char** names;
6      int size;
7      int count;
8  } vector;
9
10
```

```
11 void vector_init(vector*);
12 int vector_count(vector*);
13 void vector_add(vector*, char*);
14 void vector_set(vector*, int, char*);
15 char* vector_get(vector*, int);
16 int vector_find_by_name(vector*, char*);
17 void vector_delete(vector*, int);
18 void vector_free(vector*);
19
20 #endif
```

В качестве входных данных было подано следующее:

```
1  int main() {
2      int sum = 0;
3      int sum1 = 25+1;
4      int sum2 = 1;
5      while (sum - 10)
6      {
7          sum2 = sum1 - sum;
8          sum = sum1;
9          while (sum1 - 100)
10         {
11             sum2 = sum1 - sum;
12             sum = sum1;
13         }
14     }
15
16     while (sum1 - 100)
17     {
18             sum2 = sum1 - sum;
19             sum = sum1;
20     }
21 }
```

В результате работы программы получили следующее:

```
1  push rbp
2  mov rbp, rsp
3  mov DWORD PTR [rbp-4], 0
4  mov DWORD PTR [rbp-8], 26
5  mov DWORD PTR [rbp-12], 1
6  .L2D0:
7  cmp DWORD PRT [rbp-4], 10
8  je .L1D0
9  mov edx, DWORD PTR [rbp-8]
10 mov eax, DWORD PTR [rbp-4]
11 sub eax, edx
12 mov DWORD PTR [rbp-12], eax
13 mov eax, DWORD PTR [rbp-8]
14 DWORD PTR [rbp-4], eax
15 .L4D0:
16 cmp DWORD PRT [rbp-8], 100
17 je .L3D0
18 mov edx, DWORD PTR [rbp-8]
19 mov eax, DWORD PTR [rbp-4]
20 sub eax, edx
21 mov DWORD PTR [rbp-12], eax
22 mov eax, DWORD PTR [rbp-8]
23 DWORD PTR [rbp-4], eax
24 jmp .L4D0
```

```
.L3D0:
jmp .L2D0
.L1D0:
.L2D1:
cmp DWORD PRT [rbp-8], 100
je .L1D1
mov edx, DWORD PTR [rbp-8]
mov eax, DWORD PTR [rbp-4]
sub eax, edx
mov DWORD PTR [rbp-12], eax
mov eax, DWORD PTR [rbp-8]
DWORD PTR [rbp-4], eax
jmp .L2D1
.L1D1:
mov eax, 0
pop rbp
ret
```

Что соответствует результатам трансляции с помощью сторонней программы:

```
main:
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], 0
        mov     DWORD PTR [rbp-8], 26
        mov     DWORD PTR [rbp-12], 1
.L5:
        cmp     DWORD PTR [rbp-4], 10
        je      .L2
        mov     eax, DWORD PTR [rbp-8]
        sub     eax, DWORD PTR [rbp-4]
        mov     DWORD PTR [rbp-12], eax
        mov     eax, DWORD PTR [rbp-8]
        mov     DWORD PTR [rbp-4], eax
.L4:
        cmp     DWORD PTR [rbp-8], 100
        je      .L5
        mov     eax, DWORD PTR [rbp-8]
        sub     eax, DWORD PTR [rbp-4]
        mov     DWORD PTR [rbp-12], eax
        mov     eax, DWORD PTR [rbp-8]
        mov     DWORD PTR [rbp-4], eax
        jmp     .L4
.L2:
        cmp     DWORD PTR [rbp-8], 100
        je      .L6
        mov     eax, DWORD PTR [rbp-8]
        sub     eax, DWORD PTR [rbp-4]
        mov     DWORD PTR [rbp-12], eax
        mov     eax, DWORD PTR [rbp-8]
        mov     DWORD PTR [rbp-4], eax
        jmp     .L2
.L6:
        mov     eax, 0
        pop     rbp
        ret
```

Следовательно, можно считать, что разработаная программа работает верно.

Было предусмотренно два диагностических сообщения:

1. Сообщение о повторном объявлении переменной.

   В качестве входных данных было подано следующее:

```
1 int main() {
2     int sum = 0;
3     int sum = 2;
4 }
```

   В результате работы программы получили следующее:

```
1 push rbp
2 mov rbp, rsp
3 mov DWORD PTR [rbp-4], 0
4 ERROR: redeclaration of 'sum'
```

2. Сообщение о исользовании не объявленной переменной.

   В качестве входных данных было подано следующее:

```
1 int main() {
2     int sum = 0;
3     sum++;
4     sum4++;
5 }
```

   В результате работы программы получили следующее:

```
1 push rbp
2 mov rbp, rsp
3 mov DWORD PTR [rbp-4], 0
4 add DWORD PTR [rbp-4], 1
5 ERROR: 'sum4' was not declared in this scope
```

3 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 был выбран язык ассембера, в который будут транслироваться операторы языка C. Далее был 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. Было создано два диагностических сообщения. Полученные результаты соответствуют тому, что было описано в задании.

Проведенная работа позволила лучше понять принципы совместного использования генерторов для синтаксического и лексического разбора и принципы посторения трансляторов для языков программирования. Также в ходе

выполнения лабораторной были получены навыки разработки приложений на основе этих языков.

4 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