Санкт-Петербургский государственный политехнический университет

Факультет технической кибернетики

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе №4**

«Индивидуальное задание (утилиты YACC и LEX)»

Работу выполнил студент группы № 4081/13 Бойцев Андрей Сергеевич

Работу принял преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Цыган Владимир Николаевич

Санкт-Петербург

2012

1. **Цель работы:**

- закрепление полученных навыков применения утилит yacc и lex для генерирования синтаксических анализаторов.

1. **Задание**

Построить транслятор оператора switch-case языка С в язык a86. На ветвях case расположены операторы присваивания. Условное выражение – арифметическое, бесскобочное. Все арифметические операции выполняются слева направо без учета ранга.

1. **Программа работы**

**Входная грамматика:**

op : ex | ex op;

ex : start\_switch cond body;

start\_switch : SWITCH;

cond : '('exp')';

iii : VARNAME

iii : VARVALUE

exp : iii ;

exp : exp '+' iii ;

exp : exp '-' iii ;

exp : exp '\*' iii ;

exp : exp '/' iii ;

body : '{' op2 '}';

op2 : ex2 | ex2 op2;

ex2 : start\_case cond2 assign1 start\_break ';';

start\_case : CASE;

cond2 : iii':';

assign1 : assign1 ex;

assign1 : ex;

assign1 : assign1 assign;

assign1 : assign;

assign : iii '=' iii ';';

start\_break : BREAK ;

Лексический анализатор:

%{

#include "y.tab.h"

%}

%%

"switch" {return SWITCH;}

"case" {return CASE;}

"break" {return BREAK;}

[a-zA-Z][0-9a-zA-Z]\* { yylval.cval = strdup(yytext); return VARNAME; }

[0-9]+ { yylval.cval = strdup(yytext); return VARVALUE;}

"(" |

")" |

"{" |

"}" |

";" |

":" |

"+" |

"-" |

"\*" |

"/" |

"=" {return yytext[0];}

[ \n\t] ;

%%

int yywrap() {return 1; }

Синтаксический анализатор:

%union {

char\* cval;

};

%{

#define YYDEBUG 1

extern int yydebug;

#include <stdlib.h>

#include <string.h>

int count\_case=1;

int count\_switch=0;

int cnt = 1;

%}

%token SWITCH CASE BREAK

%token <cval>VARNAME

%token <cval>VARVALUE

%start op

%%

op : ex | ex op;

ex : start\_switch cond body;

start\_switch : SWITCH {count\_switch++;};

cond : '('exp')';

iii : VARNAME

{$<cval>$ = $<cval>1;};

iii : VARVALUE

{$<cval>$ = $<cval>1;};

exp : iii { printf(" mov ax, %s\n", $<cval>1);};

exp : exp '+' iii { printf(" add ax, %s\n", $<cval>3);};

exp : exp '-' iii { printf(" sub ax, %s\n", $<cval>3);};

exp : exp '\*' iii { printf(" mul ax, %s\n", $<cval>3);};

exp : exp '/' iii { printf(" div ax, %s\n", $<cval>3);};

body : '{' op2 '}' {cnt=0; count\_switch--; count\_case--;

if(count\_switch==0) printf("end:");};

op2 : ex2 | ex2 op2;

ex2 : start\_case cond2 assign1 start\_break ';';

start\_case : CASE;

cond2 : iii':' {printf("L%d%d:\n",count\_switch,count\_case);

printf(" mov bx, %s\n cmp ax, bx\n", $<cval>1);

printf(" jne L%d%d\n", count\_switch, count\_case+1);

count\_case++;};

assign1 : assign1 ex;

assign1 : ex;

assign1 : assign1 assign;

assign1 : assign;

assign : iii '=' iii ';' {printf(" mov %s, %s\n", $<cval>1, $<cval>3);};

start\_break : BREAK { if(cnt==0) cnt=1;

else {

printf("L%d%d:\n",count\_switch,count\_case);

printf(" jmp end\n");

}

};

%%

Транслятор обрабатывает любое число последовательных и вложенных операторов switch-case. Переменная *count\_case* является счетчиком case, переменная count\_switch – счетчиком оператора switch. Эти переменные необходимы для формирования номера меток – **L №*текущее количество операторов switch* № *текущее количество операторов case***. Такое формирование необходимо для того, чтобы при наличии вложенных операторов не происходила путаница с метками.

В таблице 1 приведены эквивалентные записи операторов и действий на языке C и a86.

*Таблица 1*

|  |  |
| --- | --- |
| язык С | ассемблер a86 |
| switch(vvv\*3-4/2) | mov ax, vvv  mul ax, 3  sub ax, 4  div ax, 2 |
| case 1:  …  break; | L##:  mov bx, 1  cmp ax, bx  jne L##+1  …  jmp end |
| vvv = 3; | mov vvv, 3 |

Пример работы транслятора:

На входе:

switch(hey)

{

case 1:

id1 = id2;

switch(hello\*3-5)

{

case 1:

id2 = 4;

break;

}

break;

case 2:

counter = 3;

break;

}

Выход:

mov ax, hey

L11:

mov bx, 1

cmp ax, bx

jne L12

mov id1, id2

mov ax, hello

mul ax, 3

sub ax, 5

L22:

mov bx, 1

cmp ax, bx

jne L23

mov id2, 4

L23:

jmp end

L12:

mov bx, 2

cmp ax, bx

jne L13

mov counter, 3

L13:

jmp end

end:

1. **Выводы:**

В результате выполнения данной работы были закреплены полученные навыки применения утилит yacc и lex для генерирования синтаксических анализаторов. Был реализован простой транслятор, соответствующий установленным требованиям.

Стоит заметить, что на первый взгляд главной сложностью данной задачи является реализация обработки вложенных операторов, но она просто решается применением рекурсии.