Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе №2**

по дисциплине «Транслирующие системы»

«**Построение синтаксических анализаторов с помощью утилиты YACC**»

Работу выполнил студент группы № 43501/3 Коренёк Григорий Андреевич

Работу принял преподаватель Цыган Владимир Николаевич

Санкт-Петербург

2018

**Цель работы**

Цель работы - изучение и получение навыков применения утилиты YACC для генерирования синтаксических анализаторов.

**Программа работы**

1. Ознакомиться с работой программы YACC.

2. Протестировать примеры.

3. Выполнить индивидуальное задание.

**Выполнение работы**

1. Простейший синтаксический анализатор на языке yacc.

%token NUMBER MONTH

%start date

%%

date : MONTH NUMBER NUMBER

%%

%{

#include "y.tab.h"

%}

%%

[0-9]+ { return NUMBER; }

jan |

feb |

march |

apr |

may |

june |

july |

aug |

sep |

oct |

nov |

dec { return MONTH; }

[ \t\n] ;

. { return 0; }

%%

#ifndef yywrap

int yywrap () { return 1; }

#endif

Вход:

jan 12 89!

Выход:

В данном случае программа ничего не выводит и корректно завершается.

Добавим лишнее число:

Вход:

jan 12 89 12!

Выход:

?-syntax error

Включим режим трассировки:

Вход:

jan 12 89 12!

Выход:

yydebug: state 0, reading 258 (MONTH)

yydebug: state 0, shifting to state 1

yydebug: state 1, reading 257 (NUMBER)

yydebug: state 1, shifting to state 3

yydebug: state 3, reading 257 (NUMBER)

yydebug: state 3, shifting to state 4

yydebug: state 4, reducing by rule 1 (date : MONTH NUMBER NUMBER)

yydebug: after reduction, shifting from state 0 to state 2

yydebug: state 2, reading 257 (NUMBER)

?-syntax error

yydebug: error recovery discarding state 2

yydebug: error recovery discarding state 0

2. Литеральные лексемы

%token NUMBER MONTH

%start date

%%

date : MONTH NUMBER ',' NUMBER

%%

%{

#include "y.tab.h"

%}

%%

[0-9]+ { return NUMBER; }

jan |

feb |

march |

apr |

may |

june |

july |

aug |

sep |

oct |

nov |

dec { return MONTH; }

"," { return yytext[0]; }

[ \t\n] ;

. { return 0; }

%%

#ifndef yywrap

int yywrap () { return 1; }

#endif

Вход:

jan 01, 18

В результате программа корректно завершается.

3. Сопутствующие значения

%token NUMBER MONTH

%start date

%%

date : MONTH NUMBER ',' NUMBER

{ printf("m-d-y: %2u-%2u-%4u\n", $1+1, $2, $4); }

%%

%{

#include <stdlib.h>

#include "y.tab.h"

#define YYSTYPE int

extern YYSTYPE yylval;

%}

%%

[0-9]+ { yylval = atoi(yytext); return NUMBER; }

jan { yylval = 0; return MONTH; }

feb { yylval = 1; return MONTH; }

march { yylval = 2; return MONTH; }

apr { yylval = 3; return MONTH; }

may { yylval = 4; return MONTH; }

june { yylval = 5; return MONTH; }

july { yylval = 6; return MONTH; }

aug { yylval = 7; return MONTH; }

sep { yylval = 8; return MONTH; }

oct { yylval = 9; return MONTH; }

nov { yylval = 10; return MONTH; }

dec { yylval = 11; return MONTH; }

"," { return yytext[0]; }

[ \t\n] ;

. { return 0; }

%%

#ifndef yywrap

int yywrap () { return 1; }

#endif

Вход:

jan 12,2018

Выход:

m-d-y: 1-12-2018

4. Проверка даты и количества дней с 1970 года.

Модуль yacc:

%{

long abs\_date (int, int, int); /\* month (0-11), day, year \*/

%}

%token NUMBER MONTH

%start date

%%

date : MONTH NUMBER ',' NUMBER

{ printf("%ld\n", abs\_date($1, $2, $4)); printf("\nDEBUG: %i\n", $4); }

%%

abs\_date.c:

#include <time.h>

extern yyerror (char \*);

/\*

\* Check date, abort on error.

\* Returns no. of days since 1970-01-01

\*/

long abs\_date (int m, int d, int y)

{

struct tm t;

time\_t seconds;

t.tm\_sec = t.tm\_min = t.tm\_hour = 0;

t.tm\_mday = d; /\* day of the month - [1,31] \*/

t.tm\_mon = m; /\* months since January - [0,11] \*/

y -= 1900;

t.tm\_year = y; /\* years since 1900, for <mktime> \*/

if ((seconds = mktime(&t)) == (time\_t)-1) {

yyerror("Date is too far from 1970-01-01");

exit(1);

}

/\* mktime turns wrong date like 32-th April to 2-nd May \*/

/\* (POSIX tells better avoid feeding mktime with that) \*/

if (t.tm\_mday != d || t.tm\_mon != m || t.tm\_year != y) {

yyerror("Date is wrong (has been corrected)");

exit(1);

}

return seconds / (3600L \* 24L);

}

В ходе экспериментов функция mktime, при подаче на вход корректной даты, уменьшала значение часов на 1, что приводило к уменьшению дней на 1, что в свою очередь приводило к выводу вместо конечного результата сообщения «Date is wrong (has been corrected)». Т.к. условия, при которых происходит ошибка, не были выявлены, было решено убрать проверку:

…

//if (t.tm\_mday != d || t.tm\_mon != m || t.tm\_year != y) {

// yyerror("Date is wrong (has been corrected)");

// exit(1);

// }

…

Вход:

feb 12,2018

Выход:

17573

5. Вычисление разницы между датами

%{

long abs\_date (int, int, int); /\* month (0-11), day, year \*/

%}

%token NUMBER MONTH

%start between

%%

date : MONTH NUMBER ',' NUMBER

{ $$ = abs\_date($1, $2, $4); }

between : date '-' date

{ printf("%ld\n", $1 - $3); }

%%

Вход:

feb 29,2000 - dec 31,1999

Выход:

60

6. Сопутствующее значение нескольких типов.

%union

{

int ival;

char \* text;

};

%token NUMBER MONTH

%start date

%%

date : MONTH NUMBER ',' NUMBER

{ print($1, $2, $4); }

%%

int print (char \*m, int d, int y)

{

printf("%d-%s-%d\n", d, m, y);

}

Трансляция yacc-модуля не прошла, поскольку в нем не задана информация о типе $1, $2 и $4 — ведь теперь у сопутствующего значения не один тип, а два.

Тип можно указать при обращении к $-переменной:

%union

{

int ival;

char \* text;

};

%token NUMBER MONTH

%start date

%%

date : MONTH NUMBER ',' NUMBER

{ print($<text>1, $<ival>2, $<ival>4); }

%%

int print (char \*m, int d, int y)

{

printf("%d-%s-%d\n", d, m, y);

}

Тип может быть указан и при объявлении терминального символа, тогда при обращении к $-переменным уточнять его не придется:

%{

#include <stdlib.h>

%}

%union

{

int ival;

char \* text;

};

%token <ival> NUMBER

%token <text> MONTH

%start date

%%

date : MONTH NUMBER ',' NUMBER

{ print($1, $2, $4); }

%%

print (char \*m, int d, int y)

{

printf("%d-%s-%d\n", d, m, y);

free(m);

}

7. Вычисление количества дней между двумя датами с сопутствующими значениями 2 типов.

%{

long abs\_date (int m, int d, int y);

%}

%union

{

int ival;

long lval;

};

%token <ival> NUMBER MONTH

%type <lval> date

%start between

%%

date : MONTH NUMBER ',' NUMBER

{ $$ = abs\_date($1, $2, $4); }

between : date '-' date

{ printf("%ld\n", $1 - $3); }

%%

%{

#include <stdlib.h>

#include "y.tab.h"

%}

%%

[0-9]+ { yylval.ival = atoi(yytext); return NUMBER; }

jan { yylval.ival = 0; return MONTH; }

feb { yylval.ival = 1; return MONTH; }

march { yylval.ival = 2; return MONTH; }

apr { yylval.ival = 3; return MONTH; }

may { yylval.ival = 4; return MONTH; }

june { yylval.ival = 5; return MONTH; }

july { yylval.ival = 6; return MONTH; }

aug { yylval.ival = 7; return MONTH; }

sep { yylval.ival = 8; return MONTH; }

oct { yylval.ival = 9; return MONTH; }

nov { yylval.ival = 10; return MONTH; }

dec { yylval.ival = 11; return MONTH; }

[ \t\n] ;

. { return yytext[0]; }

%%

#ifndef yywrap

int yywrap () { return 1; }

#endif

Вход:

feb 29,2000 - dec 31,1999

Выход:

60

8. Разбор списка чисел

%token NUM

%start \_\_list

%%

\_\_list: \_list { printf("No. of items: %d\n", $1); }

\_list: /\* empty \*/ { $$ = 0; /\* size is 0 \*/ }

| list /\* not empty, $$ == $1 by default \*/

;

list: NUM { $$ = 1; } /\* size := 1 \*/

| NUM ',' list { $$ = $3 + 1; } /\* size := size of sublist + 1 \*/

;

%%

%{

#include <stdlib.h>

#include "y.tab.h"

#define YYSTYPE int

extern YYSTYPE yylval; /\* value of numeric token \*/

%}

%%

[0-9]+ { yylval = atoi(yytext); return NUM; }

\n ;

. return yytext[0];

%%

#ifndef yywrap

int yywrap () { return 1; }

#endif

Вход:

1,2,3\n

Выход:

?-syntax error

Выполним трассировку:

user@user-VirtualBox:~/tr/yacc/list/v0$ ./a.out

1,2,3

yydebug: state 0, reading 257 (NUM)

yydebug: state 0, shifting to state 1

yydebug: state 1, reading 44 (',')

yydebug: state 1, shifting to state 5

yydebug: state 5, reading 257 (NUM)

yydebug: state 5, shifting to state 1

yydebug: state 1, reading 44 (',')

yydebug: state 1, shifting to state 5

yydebug: state 5, reading 257 (NUM)

yydebug: state 5, shifting to state 1

yydebug: state 1, reading 10 ((null))

?-syntax error

yydebug: error recovery discarding state 1

yydebug: error recovery discarding state 5

yydebug: error recovery discarding state 1

yydebug: error recovery discarding state 5

yydebug: error recovery discarding state 1

yydebug: error recovery discarding state 0

В состоянии 0 получен код 257, что соответствует лексеме NUM; в результате перешли в состояние 1. Дальше, в состоянии 1 получен код 44, что соответствует ASCII-коду ',' и т. д. — до получения символа 10, недопустимого в состоянии 1. Код 10, по таблице ASCII, означает конец строки — литерал '\n'.

Исправить ситуацию можно двумя способами: удаление \n при лексическом разборе или включение при синтаксическом.

Удаление '\n' при лексическом разборе:

%{

#include <stdlib.h>

#include "y.tab.h"

#define YYSTYPE int

extern YYSTYPE yylval; /\* value of numeric token \*/

%}

%%

[0-9]+ { yylval = atoi(yytext); return NUM; }

\n ;

. return yytext[0];

%%

#ifndef yywrap

int yywrap () { return 1; }

#endif

Включение '\n' в синтаксический разбор:

%token NUM

%start \_\_list

%%

\_\_list: \_list '\n' { printf("No. of items: %d\n", $1); }

\_list: /\* empty \*/ { $$ = 0; }

| list

;

list: NUM { $$ = 1; }

| NUM ',' list { $$ = $3 + 1; }

;

%%

Теперь выясним, как программа реагирует на разделители.

Вход:

1 , 2, 5

Выход:

yydebug: state 0, reading 257 (NUM)

yydebug: state 0, shifting to state 1

yydebug: state 1, reading 32 ((null))

?-syntax error

yydebug: error recovery discarding state 1

yydebug: error recovery discarding state 0

Сбой происходит на литере с кодом 32 — то есть как раз на пробеле. Фильтрацию пробелов и табуляций имеет смысл выполнять в lex-модуле.

%{

#include <stdlib.h>

#include "y.tab.h"

#define YYSTYPE int

extern YYSTYPE yylval; /\* value of numeric token \*/

%}

%%

[0-9]+ { yylval = atoi(yytext); return NUM; }

[ \t\n]+ ;

. return yytext[0];

%%

#ifndef yywrap

int yywrap () { return 1; }

#endif

9. Вывод элементов списка (правая рекурсия)

%token NUM

%start \_\_list

%%

\_\_list: \_list '\n'

\_list: /\* empty \*/ { $$ = 0; }

| list

;

list: NUM { $$ = 1; print($$, $1, 1); }

| NUM

','

list { $$ = $3 + 1; print($$, $1, 2); }

;

%%

print (int len, int val, int rule)

{

printf("%d: %d (rule %d)\n", len, val, rule) ;

}

Вход:

1,2,3

Выход:

1: 3 (rule 1)

2: 2 (rule 2)

3: 1 (rule 2)

10. Вывод элементов списка (левая рекурсия)

%token NUM

%start \_\_list

%%

\_\_list: \_list '\n'

\_list: /\* empty \*/ { $$ = 0; }

| list

;

list: NUM { $$ = 1; print($$, $1, 1); }

| list

','

NUM { $$ = $1 + 1; print($$, $3, 2); }

;

%%

print (int len, int val, int rule)

{

printf("%d: %d (rule %d)\n", len, val, rule) ;

}

Вход:

1,2,3

Выход:

1: 1 (rule 1)

2: 2 (rule 2)

3: 3 (rule 2)

**Индивидуальное задание**

Лексемы:

<множитель> ::= идентификатор | число

<умножение> ::= ‘\*’ | ‘/’

<сложение> ::= ‘+’ | ‘-’

Грамматика:

<произведение> ::= <множитель> | <произведение> <умножение> <множитель>

<скобочная форма> ::= <произведение>

| <скобочная форма> <сложение> <скобочная форма>

| (<скобочная форма>) <умножение> <произведение>

| <произведение> <умножение> (<скобочная форма>)

| (<скобочная форма>) <умножение> (<скобочная форма>)

| (<скобочная форма>)

ind.l:

%{

#include <stdlib.h>

#include "y.tab.h"

#define YYSTYPE int

extern YYSTYPE yylval; /\* value of numeric token \*/

%}

%%

([a-zA-Z]+[0-9]\*)|([0-9]+) { yylval = strdup(yytext); return OPND; }

\+ { yylval = "add"; return SUM; }

\- { yylval = "sub"; return SUM; }

\\* { yylval = "mul"; return PROD; }

\/ { yylval = "div"; return PROD; }

(.|\n) return yytext[0];

%%

#ifndef yywrap

int yywrap () { return 1; }

#endif

ind.y:

%token OPND

%start result

%%

result: bf { printf("pop r1\n"); }

bf: prod

| bf SUM bf { printf("pop ax\npop r0\n%s ax r0\npush ax\n", $2); }

| '(' bf ')' PROD prod { printf("pop ax\npop r0\n%s r0\npush ax\n", $4); }

| prod PROD '(' bf ')' { printf("pop ax\npop r0\n%s r0\npush ax\n", $2); }

| '(' bf ')' PROD '(' bf ')' { printf("pop ax\npop r0\n%s ax r0\npush ax\n", $4); }

| '(' bf ')' { printf("push ax\n"); }

prod: OPND { printf("mov %s ax\npush ax\n", $1); }

| prod PROD OPND { printf("pop ax\nmov %s r0\n%s r0\npush ax\n", $3, $2); }

;

%%

Анализатор работает следующим образом: при обнаружении операции (+ - \* /) в соответствии с правилами грамматики производится генерация команд ассемблера, соответствующих выгрузке операндов из стека, выполнения арифметической операции и занесения обратно в стек. При обнаружении отдельного операнда – просто занесение в стек.

Приоритет операции умножения перед операцией сложения обеспечивается структурой грамматики. Все нетерминалы, соответствующие умножению, представляют собой только комбинации из произведения и выражения в скобках. Поэтому сложение может обнаружиться (и сгенерировать соответствующие команды) раньше умножения, только если сумма заключена в скобки.

Вход:

a+b\*c

Выход:

mov a ax

push ax

mov b ax

push ax

pop ax

mov c r0

mul r0

push ax

pop ax

pop r0

add ax r0

push ax

pop r1

Вход:

a\*b+c

Выход:

mov a ax

push ax

pop ax

mov b r0

mul r0

push ax

mov c ax

push ax

pop ax

pop r0

add ax r0

push ax

pop r1

Как видно из первых двух примеров, старшинство операций обрабатывается правильно. Рассмотрим более сложный пример – цепную дробь.

Вход:

1/(a+1/(b+1/(c+d)))

Выход:

mov 1 ax

push ax

mov a ax

push ax

mov 1 ax

push ax

mov b ax

push ax

mov 1 ax

push ax

mov c ax

push ax

mov d ax

push ax

pop ax

pop r0

add ax r0

push ax

pop ax

pop r0

div r0

push ax

pop ax

pop r0

add ax r0

push ax

pop ax

pop r0

div r0

push ax

pop ax

pop r0

add ax r0

push ax

pop ax

pop r0

div r0

push ax

pop r1

**Вывод**

В ходе работы были рассмотрены основные принципы работы с программой YACC. На примерах рассмотрена структура и синтаксис YACC -программы. Полученные знания были обобщены при работе над индивидуальным заданием. В качестве грамматики, описывающей скобочные выражения, была выбрана не самая простая, но обеспечивающая правильный приоритет операций.