Санкт-Петербургский государственный политехнический университет

Факультет технической кибернетики

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе №3**

«Построение синтаксических анализаторов   
(утилиты yacc и lex)»

Работу выполнил студент группы № 4081/12

Дорофеев Юрий Владимирович

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

г. Санкт-Петербург

2012

1. **Цель работы**

Изучение и получение навыков применения утилит yacc и lex для генерации синтаксических анализаторов.

1. **Программа работы**
2. **Включение трассировки правил**

Лексический анализ сводится к выявлению чисел и строк с названиями месяцев, что задано следующей lex-спецификацией:

%{

#include "y.tab.h"

%}

%%

[0-9]+ { return NUMBER; }

jan |

feb |

mar |

apr |

may |

jun |

jul |

aug |

sep |

nov |

dec { return MONTH; }

[ \t\n] ;

. { return 0; }

%%

int yywrap() { return 1; }

Синтаксический анализатор проверяет структуры ввода и выводит сообщения, поясняющих процесс синтаксического анализа:

%{

#define YYDEBUG 1

extern int yydebug;

%}

%token NUMBER MONTH

%start date

%%

date : MONTH NUMBER NUMBER

%%

Основная программа zz.c:

#include <stdio.h>

extern int yydebug;

main() {

yydebug = 1;

return yyparse();

}

yyerror(char \* s) {

fprintf( stderr, "%s\n", s);

}

В этой спецификации определены лексемы NUMBER и MONTH, и задан стартовый символ date. (Стартовый символ - это один из нетерминальных символов, обнаружение которого представляет главную цель синтаксического разбора). Переменная окружения YYDEBUG должна быть установлена равной 1 для вывода сообщений, поясняющих процесс синтаксического анализа.

Сборка и тестирование:

dorofeev@dorofeev-VirtualBox:~/lab3$ gcc -c -o zz.obj zz.c

dorofeev@dorofeev-VirtualBox:~/lab3$ yacc -vdt date.y

dorofeev@dorofeev-VirtualBox:~/lab3$ gcc -c -o y.obj y.tab.c

dorofeev@dorofeev-VirtualBox:~/lab3$ flex -lw date.l

dorofeev@dorofeev-VirtualBox:~/lab3$ cc -c -o scaner.obj lex.yy.c

dorofeev@dorofeev-VirtualBox:~/lab3$ cc -o syntax zz.obj y.obj scaner.obj

Протестируем исполняемую программу. На ее вход передаются файлы test\_ok (nov 02 2012) и test\_fail(dfe 21 2012), которые соответствуют правильным входным данным и некорректным.

Правильная работа:

dorofeev@dorofeev-VirtualBox:~/lab3$ ./syntax <test\_ok

Starting parse

Entering state 0

Reading a token: Next token is token MONTH ()

Shifting token MONTH ()

Entering state 1

Reading a token: Next token is token NUMBER ()

Shifting token NUMBER ()

Entering state 3

Reading a token: Next token is token NUMBER ()

Shifting token NUMBER ()

Entering state 5

Reducing stack by rule 1 (line 11):

$1 = token MONTH ()

$2 = token NUMBER ()

$3 = token NUMBER ()

-> $$ = nterm date ()

Stack now 0

Entering state 2

Reading a token: Now at end of input.

Shifting token $end ()

Entering state 4

Stack now 0 2 4

Cleanup: popping token $end ()

Cleanup: popping nterm date ()

Ошибка:

dorofeev@dorofeev-VirtualBox:~/lab3$ ./syntax <test\_fail

Starting parse

Entering state 0

Reading a token: Now at end of input.

syntax error

Cleanup: discarding lookahead token $end ()

Stack now 0

В сообщениях shift означает продолжение разбора правила, reduce - замену (сокращение) последовательности символов одним символом, в соответствии с некоторым правилом. В процессе разбора переключаются состояния (state) автомата, перечисленные в файле y.output.

Из заголовочного файла y\_tab.h видно, что коды терминальных символов, определенных при помощи ключевого слова %token, начинаются с 257. Код 0 зарезервирован для маркера конца ввода. Коды от 1 до 256 зарезервированы для литеральных лексем, или "литералов''.

1. **Литеральные лексемы**

Изменение в yacc-спецификации (date.y):

**date : MONTH NUMBER ',' NUMBER**

Запятая в одиночных кавычках задает литерал - терминальный символ, код которого равен ASCII-коду запятой.

В lex-спецификации добавлено правило:

**"," {return yytext[0];}**

Теперь числа должны разделяться запятой. Протестируем исполняемую программу. На ее вход передаются файлы test\_ok.txt (nov 02, 2012) и test\_fail.txt (nov 21 2012), которые соответствуют правильным входным данным и некорректным.

Правильная работа:

dorofeev@dorofeev-VirtualBox:~/lab3$ ./syntax <test\_ok

Starting parse

Entering state 0

Reading a token: Next token is token MONTH ()

Shifting token MONTH ()

Entering state 1

Reading a token: Next token is token NUMBER ()

Shifting token NUMBER ()

Entering state 3

Reading a token: Next token is token ',' ()

Shifting token ',' ()

Entering state 5

Reading a token: Next token is token NUMBER ()

Shifting token NUMBER ()

Entering state 6

Reducing stack by rule 1 (line 11):

$1 = token MONTH ()

$2 = token NUMBER ()

$3 = token ',' ()

$4 = token NUMBER ()

-> $$ = nterm date ()

Stack now 0

Entering state 2

Reading a token: Now at end of input.

Shifting token $end ()

Entering state 4

Stack now 0 2 4

Cleanup: popping token $end ()

Cleanup: popping nterm date ()

Добавилось одно состояние в файле y.output (по сравнению с предыдущим примером).

Ошибка:

dorofeev@dorofeev-VirtualBox:~/lab3$ ./syntax <test\_fail

Starting parse

Entering state 0

Reading a token: Next token is token MONTH ()

Shifting token MONTH ()

Entering state 1

Reading a token: Next token is token NUMBER ()

Shifting token NUMBER ()

Entering state 3

Reading a token: Next token is token NUMBER ()

syntax error

Error: popping token NUMBER ()

Stack now 0 1

Error: popping token MONTH ()

Stack now 0

Cleanup: discarding lookahead token NUMBER ()

Stack now 0

Видно, что в состоянии 3 считывается NUMBER, а должно ',' - поэтому выводится ошибка.

Скорректируем пример так, чтобы на месте запятой можно было бы использовать также точку с запятой.

Изменение в yacc-спецификации (date.y):

**date : MONTH NUMBER ',' NUMBER**

**| MONTH NUMBER ';' NUMBER**

**;**

В lex-спецификации добавлено правило:

**"," {return yytext[0];}**

**";" {return yytext[0];}**

Правильная работа ',':

dorofeev@dorofeev-VirtualBox:~/lab3$ ./syntax <test\_ok

Starting parse

Entering state 0

Reading a token: Next token is token MONTH ()

Shifting token MONTH ()

Entering state 1

Reading a token: Next token is token NUMBER ()

Shifting token NUMBER ()

Entering state 3

Reading a token: Next token is token ',' ()

Shifting token ',' ()

Entering state 5

Reading a token: Next token is token NUMBER ()

Shifting token NUMBER ()

Entering state 7

Reducing stack by rule 1 (line 11):

$1 = token MONTH ()

$2 = token NUMBER ()

$3 = token ',' ()

$4 = token NUMBER ()

-> $$ = nterm date ()

Stack now 0

Entering state 2

Reading a token: Now at end of input.

Shifting token $end ()

Entering state 4

Stack now 0 2 4

Cleanup: popping token $end ()

Cleanup: popping nterm date ()

Правильная работа ';':

dorofeev@dorofeev-VirtualBox:~/lab3$ ./syntax <test\_ok

Starting parse

Entering state 0

Reading a token: Next token is token MONTH ()

Shifting token MONTH ()

Entering state 1

Reading a token: Next token is token NUMBER ()

Shifting token NUMBER ()

Entering state 3

Reading a token: Next token is token ';' ()

Shifting token ';' ()

Entering state 6

Reading a token: Next token is token NUMBER ()

Shifting token NUMBER ()

Entering state 8

Reducing stack by rule 2 (line 12):

$1 = token MONTH ()

$2 = token NUMBER ()

$3 = token ';' ()

$4 = token NUMBER ()

-> $$ = nterm date ()

Stack now 0

Entering state 2

Reading a token: Now at end of input.

Shifting token $end ()

Entering state 4

Stack now 0 2 4

Cleanup: popping token $end ()

Cleanup: popping nterm date ()

В state 3 в зависимости от считанного литерала происходит переход либо в state 5 (','), либо в state 6 (';').

Ошибка:

dorofeev@dorofeev-VirtualBox:~/lab3$ ./syntax <test\_fail

Starting parse

Entering state 0

Reading a token: Next token is token MONTH ()

Shifting token MONTH ()

Entering state 1

Reading a token: Next token is token NUMBER ()

Shifting token NUMBER ()

Entering state 3

Reading a token: Next token is token ',' ()

Shifting token ',' ()

Entering state 5

Reading a token: Next token is token ';' ()

syntax error

Error: popping token ',' ()

Stack now 0 1 3

Error: popping token NUMBER ()

Stack now 0 1

Error: popping token MONTH ()

Stack now 0

Cleanup: discarding lookahead token ';' ()

Stack now 0

Видно, что в состоянии 3 считывается NUMBER, а должно ',' или ';' - поэтому выводится ошибка.

1. **Сопутствующие значения одного типа**

Лексический анализатор изменен следующим образом:

%{

#include <stdlib.h>

#include <time.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; }

mar { yylval = 2; return MONTH; }

apr { yylval = 3; return MONTH; }

may { yylval = 4; return MONTH; }

jun { yylval = 5; return MONTH; }

jul { 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; }

%%

int yywrap() { return 1; }

void abs\_date (const int month, const int day, const int year) {

struct tm time\_info;

time\_t time\_raw\_format;

time\_info.tm\_year = year - 1900;

time\_info.tm\_mon = month;

time\_info.tm\_mday = 1000;

time\_info.tm\_hour = 0;

time\_info.tm\_min = 0;

time\_info.tm\_sec = 0;

time\_info.tm\_isdst = 0;

time\_raw\_format = mktime(&time\_info);

printf("%ld", time\_raw\_format);

if(time\_raw\_format == -1){

printf("incorrect date\n");

}

}

Добавлено определение типа YYSTYPE сопутствующего значение и ссылка на внешнюю переменную yylval, принадлежащую yyparse(). Значение, сопутствующее лексеме NUMBER, - это значение числа. Лексеме MONTH сопутствует номер месяца в диапазоне [0..11].

Изменение в yacc-спецификации (date.y):

**date : MONTH NUMBER ',' NUMBER**

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

Правильная работа:

dorofeev@dorofeev-VirtualBox:~/lab3$ ./syntax < test\_ok.txt

Starting parse

Entering state 0

Reading a token: Next token is token MONTH ()

Shifting token MONTH ()

Entering state 1

Reading a token: Next token is token NUMBER ()

Shifting token NUMBER ()

Entering state 3

Reading a token: Next token is token ',' ()

Shifting token ',' ()

Entering state 5

Reading a token: Next token is token NUMBER ()

Shifting token NUMBER ()

Entering state 6

Reducing stack by rule 1 (line 11):

$1 = token MONTH ()

$2 = token NUMBER ()

$3 = token ',' ()

$4 = token NUMBER ()

-> $$ = nterm date ()

Stack now 0

Entering state 2

Reading a token: Now at end of input.

Shifting token $end ()

Entering state 4

Stack now 0 2 4

Cleanup: popping token $end ()

Cleanup: popping nterm date ()

1377460800

В состоянии 6 происходит замена последовательности. В состоянии 4 происходит запись сопутствующих значений в стэк.

1. **Сопутствующие значения различных типов**

В рассмотренном примере все сопутствующие значения имели один и тот же тип - int. Часто может требоваться возвращать сопутствующие значения разных типов, например, int и char\*, при том, что `канал передачи' значений от yylex() к yyparse() единственный - переменная yylval.

В такой ситуации следует воспользоваться объявлением типа сопутствующего значения в виде объединения. Поскольку теперь возможны два типа сопутствующих значений, требуется дополнительная информация для подстановки псевдопеременных $1, $2, $4(<ival>, <text>). В этом примере тип связан с терминальным символом, и уточнять тип каждого экземпляра псевдопеременных не требуется.

Изменение в yacc-спецификации (date.y):

%{

#define YYDEBUG 1

extern int yydebug;

%}

%union

{

int ival;

char \*text;

};

%token <text>MONTH

%token <ival> MONTH

%start date

%%

date : MONTH NUMBER ',' NUMBER

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

printf("%d.%s.%d\n", $2, $1, $4);}

%%

Лексический анализатор изменен следующим образом:

%{

#include <stdlib.h>

#include <time.h>

#include "y.tab.h"

%}

%%

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

jan |

feb |

mar |

apr |

may |

jun |

jul |

aug |

sep |

oct |

nov |

dec { yylval.text = strdup(yytext); return MONTH; }

"," { return yytext[0]; }

[ \t\n] ;

. { return 0; }

%%

int yywrap() { return 1; }

void abs\_date (const int month, const int day, const int year) {

struct tm time\_info;

time\_t time\_raw\_format;

time\_info.tm\_year = year - 1900;

time\_info.tm\_mon = month;

time\_info.tm\_mday = 1000;

time\_info.tm\_hour = 0;

time\_info.tm\_min = 0;

time\_info.tm\_sec = 0;

time\_info.tm\_isdst = 0;

time\_raw\_format = mktime(&time\_info);

printf("%ld", time\_raw\_format);

if(time\_raw\_format == -1){

printf("incorrect date\n");

}

}

Правильная работа:

dorofeev@dorofeev-VirtualBox:~/lab3$ ./syntax <test\_ok.txt

Starting parse

Entering state 0

Reading a token: Next token is token MONTH ()

Shifting token MONTH ()

Entering state 1

Reading a token: Next token is token NUMBER ()

Shifting token NUMBER ()

Entering state 3

Reading a token: Next token is token ',' ()

Shifting token ',' ()

Entering state 5

Reading a token: Next token is token NUMBER ()

Shifting token NUMBER ()

Entering state 6

Reducing stack by rule 1 (line 16):

$1 = token MONTH ()

$2 = token NUMBER ()

$3 = token ',' ()

$4 = token NUMBER ()

-1 incorrect date

**2.dec.2012**

-> $$ = nterm date ()

Stack now 0

Entering state 2

Reading a token: Now at end of input.

Shifting token $end ()

Entering state 4

Stack now 0 2 4

Cleanup: popping token $end ()

Cleanup: popping nterm date ()

Неправильная работа:

dorofeev@dorofeev-VirtualBox:~/lab3$ ./syntax < test\_fail.txt

Starting parse

Entering state 0

Reading a token: Next token is token MONTH ()

Shifting token MONTH ()

Entering state 1

Reading a token: Next token is token NUMBER ()

Shifting token NUMBER ()

Entering state 3

Reading a token: Next token is token NUMBER ()

syntax error

Error: popping token NUMBER ()

Stack now 0 1

Error: popping token MONTH ()

Stack now 0

Cleanup: discarding lookahead token NUMBER ()

Stack now 0

1. **Значения, сопутствующие нетерминальным символам**

В предыдущих примерах в стек значений записывалось содержимое yylval, устанавливаемое лексическим анализатором, и сопутствующее терминальным символам. Yacc позволяет также:

* изменять значения в стеке;
* формировать значения, сопутствующие нетерминальным символам.

Величина, сопутствующая нетерминальному символу, обозначается как $$. $$ и $1 фактически ссылаются на одну и ту же запись стека, откуда и следует "действие по умолчанию'': $$ = $1. Тип $$ при использовании %union может быть уточнен принудительно для каждого обращения к $$, в форме:

**$<type>$**

Удобнее связывать тип с нетерминальным символом в секции определений, в форме:

**%type sym**

Лексический анализатор изменен следующим образом:

%{

#include <stdlib.h>

#include <time.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; }

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

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

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

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

jul { 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; }

"," { return yytext[0]; }

"-" { return yytext[0]; }

[ \t\n] ;

. { return 0; }

%%

int yywrap() { return 1; }

long abs\_date (const int month, const int day, const int year) {

struct tm time\_info;

time\_t time\_raw\_format;

time\_info.tm\_year = year - 1900;

time\_info.tm\_mon = month;

time\_info.tm\_mday = day;

time\_info.tm\_hour = 0;

time\_info.tm\_min = 0;

time\_info.tm\_sec = 0;

time\_info.tm\_isdst = 0;

time\_raw\_format = mktime(&time\_info);

printf("%ld\n", time\_raw\_format);

if(time\_raw\_format == -1){

printf("incorrect date\n");

}

return time\_raw\_format;

}

Ниже определен синтаксический анализатор, вычисляющий количество дней между двумя датами:

%{

#define YYDEBUG 1

extern int yydebug;

%}

%union

{

int ival;

long lval;

};

%token <ival> MONTH NUMBER

%type <lval> date

%start between

%%

date : MONTH NUMBER ',' NUMBER

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

between : date '-' date

{ printf("%ld\n", ($1 - $3)/(3600L \* 24L) ); }

%%

Здесь сопутствующее значение может быть двух типов:

* int - для номеров дня, месяца и года;
* long - для значения, сопутствующего date.

Вычисление количества дней в году:

dorofeev@dorofeev-VirtualBox:~/lab3$ cat test\_ok.txt

jan 1, 2012 - jan 1, 2011

dorofeev@dorofeev-VirtualBox:~/lab3$ ./syntax < test\_ok.txt

...

Entering state 9

Reducing stack by rule 1 (line 11):

$1 = token MONTH ()

$2 = token NUMBER ()

$3 = token ',' ()

$4 = token NUMBER ()

1411675200

-> $$ = nterm date ()

Stack now 0

Entering state 2

Reading a token: Next token is token '-' ()

Shifting token '-' ()

...

Entering state 9

Reducing stack by rule 1 (line 11):

$1 = token MONTH ()

$2 = token NUMBER ()

$3 = token ',' ()

$4 = token NUMBER ()

1380139200

-> $$ = nterm date ()

Stack now 0 2 5

Entering state 8

Reducing stack by rule 2 (line 13):

$1 = nterm date ()

$2 = token '-' ()

$3 = nterm date ()

**DAYS: 365**

...

Вычисление периода в днях:

dorofeev@dorofeev-VirtualBox:~/lab3$ cat test\_ok.txt

jan 1, 2012 - dec 11, 2011

dorofeev@dorofeev-VirtualBox:~/lab3$ ./syntax < test\_ok.txt

...

Entering state 9

Reducing stack by rule 1 (line 11):

$1 = token MONTH ()

$2 = token NUMBER ()

$3 = token ',' ()

$4 = token NUMBER ()

1325361600

-> $$ = nterm date ()

Stack now 0

Entering state 2

Reading a token: Next token is token '-' ()

Shifting token '-' ()

...

Entering state 9

Reducing stack by rule 1 (line 11):

$1 = token MONTH ()

$2 = token NUMBER ()

$3 = token ',' ()

$4 = token NUMBER ()

1323547200

-> $$ = nterm date ()

Stack now 0 2 5

Entering state 8

Reducing stack by rule 2 (line 13):

$1 = nterm date ()

$2 = token '-' ()

$3 = nterm date ()

**DAYS: 21**

...

Ниже определен синтаксический анализатор, демонстрирующий изменение значений в стеке и применение действия внутри правила:

%union

{

int ival;

long lval;

};

%token <ival> MONTH NUMBER

%type <lval> date

%start between

%%

date : MONTH NUMBER ',' NUMBER

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

between : date

{ $<lval>$ = $1 / (3600L \* 24L); }

'-'

date

{ $4 = $4/(3600L \* 24L);

printf( "%ld\n", $<lval>2 - $4 ); }

%%

Действие в середине правила трактуется как отдельный нетерминальный псевдосимвол, с собственным сопутствующим значением (для которого резервируется запись в стеке). В таких действиях псевдопеременная $$ ссылается на значение, сопутствующее псевдосимволу. Поскольку тип значения, сопутствующего псевдосимволу($2), не очевиден, пришлось уточнять тип при каждом обращении к соответствующей записи стека. Для последующих сопутствующих значений номер записи в стеке сдвигается на 1 (для второй date не $3, а $4).

Вычисление количества дней в году:

dorofeev@dorofeev-VirtualBox:~/lab3$ $cat test\_ok.txt

jan 1, 2012 - jan 1, 2011

dorofeev@dorofeev-VirtualBox:~/lab3$ ./syntax < test\_ok.txt

...

Entering state 9

Reducing stack by rule 1 (line 11):

$1 = token MONTH ()

$2 = token NUMBER ()

$3 = token ',' ()

$4 = token NUMBER ()

1411675200

-> $$ = nterm date ()

Stack now 0

Entering state 2

Reducing stack by rule 2 (line 14):

-> $$ = nterm @1 ()

Stack now 0 2

Entering state 5

Reading a token: Next token is token '-' ()

Shifting token '-' ()

...

Entering state 9

Reducing stack by rule 1 (line 11):

$1 = token MONTH ()

$2 = token NUMBER ()

$3 = token ',' ()

$4 = token NUMBER ()

1380139200

-> $$ = nterm date ()

Stack now 0 2 5 8

Entering state 10

Reducing stack by rule 3 (line 13):

$1 = nterm date ()

$2 = nterm @1 ()

$3 = token '-' ()

$4 = nterm date ()

DAYS: **365**

...

Вычисление периода в днях:

dorofeev@dorofeev-VirtualBox:~/lab3$ $cat test\_ok.txt

jan 1, 2012 - dec 11, 2011

dorofeev@dorofeev-VirtualBox:~/lab3$ ./syntax < test\_ok.txt

...

Entering state 9

Reducing stack by rule 1 (line 11):

$1 = token MONTH ()

$2 = token NUMBER ()

$3 = token ',' ()

$4 = token NUMBER ()

1325361600

-> $$ = nterm date ()

Stack now 0

Entering state 2

Reducing stack by rule 2 (line 14):

-> $$ = nterm @1 ()

Stack now 0 2

Entering state 5

Reading a token: Next token is token '-' ()

Shifting token '-' ()

...

Entering state 9

Reducing stack by rule 1 (line 11):

$1 = token MONTH ()

$2 = token NUMBER ()

$3 = token ',' ()

$4 = token NUMBER ()

1323547200

-> $$ = nterm date ()

Stack now 0 2 5 8

Entering state 10

Reducing stack by rule 3 (line 13):

$1 = nterm date ()

$2 = nterm @1 ()

$3 = token '-' ()

$4 = nterm date ()

DAYS: **21**

...

1. **Выводы**

В результате выполнения данной работы были получены навыки применения утилит yacc и lex для генерации синтаксических анализаторов. Разбор данных примеров необходим для дальнейшей реализации индивидуального задания.