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

Отчёт по лабораторной работе

Дисциплина: Транслирующие системы

Тема: Программирование синтаксического разбора на языке *yacc*

Выполнил студент гр. 43501/3 Мальцев М.С.
Преподаватель Цыган В.Н.

Санкт-Петербург
24 марта 2019 г.

1 Цель работы

- Познакомиться с языком синтаксического разбора *yacc*
- Выполнить трансляцию предложенных программ на языке *yacc*. Протестировать их работу.

2 Ход работы

2.1 Проверка формата даты

Рассмотрим программу, на языках *lex* и *yacc*, которая проверяет корректность введенной даты:

```
%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
```

Сгенерированный файл:

```
#include <stdio.h>

/* ------ Define external objects ------ */

int yydebug = 0;     /* To trace parser, set yydebug = 1      */
                     /* ... and call yacc with options -vtd   */
                     /* 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     return yyparse();
21 }
```

Проверим корректность работы программы. Для этого подадим на вход следующие тестовые данные:

```
1 jan 12 89!
```

Программа отработала корректно. Так как ошибок обнаружено не было, то вывод программы тоже оказался пустым.

Входные данные были модифицированы, с целью получить сообщение об ошибке:

```
1 jan 12 89 84!
```

В итоге, в консоль было выведено сообщение $? - syntaxerror$. Что сведетельствует, о несоответствии входных данных разработаному шаблону формата даты. Программа обнаружила несоответствие формату в предложенных, заведомо некорректных, данных, следовательно, можно считать, что программа работает правильно.

2.2 Изменение структуры ввода даты

Модифицируем программу из предыдущего пункта. Изменим формат даты, так чтобы между числами должна была бы идти запятая:

```
1 %token   NUMBER MONTH
2 %start   date
3
4 %%
5 date :   MONTH NUMBER ',' NUMBER
6 %%
```

```
1 %{
2 #include "y.tab.h"
3 %}
4
5 %%
6 [0-9]+       { return NUMBER; }
7 jan          |
8 feb          |
9 march        |
10 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
```

Протестируем программу, в одном случае получив информацию, о корректном завершении, в другом об ошибке.

Входные данные, которые привели к успешному завершению:

```
dec 12,12
apr 1,1
nov 89,21
```

Входные данные, которые привели к ошибке:

```
dec 12,12,
apr 1.1
```

2.3 Доступ к семантическим значениям

Доступ к семантическим значениям осуществляется через псевдопеременные $n. Рассмотрим следующую программу:

```
%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
```

На вход программы было подано:

```
aug 1,1
```

В итоге получили:

```
m-d-y:  8- 1-   1
```

Следовательно можно сделать вывод, что программа отработала корректно.

2.4 Проверка даты и вывод количества дней от 1970 г.

Рассмотрим программу, которая выводит количество дней от 1970-01-01, а также проверяет корректность введенной даты:

```
%{
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)); }
%%
```

```
%{
#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
```

```
#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);
}
```

На вход программы было подано:

```
jan 1, 1975
```

В итоге получили:

```
1825
```

Следовательно можно сделать вывод, что программа отработала корректно.

2.5 Семантическое значение date и вычисление разницы между датами

В приведенной ниже программе вычисляется разница между двумя введенными датами, а также проверяется их соответствие с заданым форматом:

```
%{
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); }
%%
```

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


extern int 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; }
[ \t\n]     ;
.           { return yytext[0]; }   /* literal */
%%

#ifndef yywrap
int yywrap () { return 1; }
#endif
```

```
#include <time.h>
#include <stdlib.h>

```

```
4  extern yyerror (char *);
5
6  /*
7   *  Check date, abort on error.
8   *  Returns no. of days since 1970-01-01
9   */
10 long abs_date (int m, int d, int y)
11 {
12     struct tm t;
13     time_t seconds;
14
15
16     t.tm_sec = t.tm_min = t.tm_hour = 0;
17
18     t.tm_mday = d;     /* day of the month - [1,31] */
19     t.tm_mon  = m;     /* months since January - [0,11] */
20     y -= 1900;
21     t.tm_year = y;     /* years since 1900, for <mktime> */
22
23     if ((seconds = mktime(&t)) == (time_t)-1) {
24         yyerror("Date is too far from 1970-01-01");
25         exit(1);
26     }
27
28     /* mktime turns wrong date like 32-th April to 2-nd May */
29     /* (POSIX tells better avoid feeding mktime with that) */
30     if (t.tm_mday != d || t.tm_mon != m || t.tm_year != y) {
31         yyerror("Date is wrong (has been corrected)");
32         exit(1);
33     }
34     return seconds / (3600L * 24L);
35 }
```

На вход программы было подано:

```
1  feb 29,2000 - dec 31,1999
```

В итоге получили:

```
1  60
```

Следовательно можно сделать вывод, что программа отработала корректно.

2.6 Определение сопутствующего типа значения нескольких типов

Рассмотрим работу с двумя типами сопутствующих значений. Для этого модифицируем предыдущую программу:

```
1  %{
2  long abs_date (int m, int d, int y);
3  %}
4
5  %union
6  {
7      int   ival;
8      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
```

На вход программы было подано:

```
feb 29,2000 - dec 31,1999
```

В итоге получили:

```
70
```

Следовательно можно сделать вывод, что программа отработала корректно.

2.7 Рекурсивные правила

Рассмотрим программу разбирающую список чисел, разделенных запятыми:

```
%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
```

На вход программы было подано:

```
1,2,3,4,5,6,7,8,9,10,20,30,40,50,60,70,80,90,100
```

В итоге получили:

```
No. of items: 19
```

Следовательно можно сделать вывод, что программа отработала корректно.

2.8 Использование рекурсии при чтении списка

Рассмотрим программу для чтения списка с помощью рекурсии:

```
%token NUM
%start __list

%%
__list: _list

_list: /* empty */   { $$ = 0; }
     | list
     ;

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

```
14          list        { $$ = $3 + 1; print($$, $1, 2); }
15      ;
16 %%
17
18 print (int len, int val, int rule)
19 {
20      printf("%d: %d (rule %d)\n", len, val, rule) ;
21 }
```

```
1  %{
2  #include <stdlib.h>
3  #include "y.tab.h"
4
5  #define YYSTYPE int
6  extern YYSTYPE yylval;     /* value of numeric token */
7  %}
8
9  %%
10 [0-9]+   { yylval = atoi(yytext); return NUM; }
11 \n       return '\n';
12 .        return yytext[0];
13 %%
14
15 #ifndef yywrap
16 int yywrap () { return 1; }
17 #endif
```

На вход программы было подано:

```
1  1,2,3,4,5,6,7,8,9,10,20,30,40,50,60,70,80,90,100
```

В итоге получили:

```
1  1: 100 (rule 1)
2  2: 90 (rule 2)
3  3: 80 (rule 2)
4  4: 70 (rule 2)
5  5: 60 (rule 2)
6  6: 50 (rule 2)
7  7: 40 (rule 2)
8  8: 30 (rule 2)
9  9: 20 (rule 2)
10 10: 10 (rule 2)
11 11: 9 (rule 2)
12 12: 8 (rule 2)
13 13: 7 (rule 2)
14 14: 6 (rule 2)
15 15: 5 (rule 2)
16 16: 4 (rule 2)
17 17: 3 (rule 2)
18 18: 2 (rule 2)
19 19: 1 (rule 2)
```

Следовательно можно сделать вывод, что программа отработала корректно.

2.9 Дополнительное микрозадание

Микрозадание: В списке могут быть целые числа со знаком и без знака. Основной разделитель чисел - запятая, но после запятой допускается произвольное количество пробелов.

Подсчитать для каждой строки общее количество чисел и количество отрицательных чисел.

Текст разработанной программы:

```
%token NUM
%start __list


%%
__list: _list

_list: /* empty */  { $$ = 0; }
      | list
      ;

list: NUM               { inc($1); }
    | list ',' NUM      { inc($3); }
    | list '\n'         { print($$); $$ = $1 + 1; reset_numOfValues(); }
    | list NUM          { inc($2); }
    ;
%%

int numOfValues = 0;
int numOfNegativeValues = 0;

inc (int val) {
  numOfValues++;
  if (val < 0) {
    numOfNegativeValues++;
  }
}

print (int numOfStr) {
  printf("%d : all = %d , neg = %d \n", numOfStr, numOfValues,
    numOfNegativeValues);
}

reset_numOfValues () {
  numOfValues = 0;
  numOfNegativeValues = 0;
}
```

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

#define YYSTYPE int
extern YYSTYPE yylval;     /* value of numeric token */
%}

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

```
12 (.|\n)        return yytext[0];
13 %%
14
15 #ifndef yywrap
16 int yywrap () { return 1; }
17 #endif
```

Входные данные:

```
1 1,2,-3, -4,5,6,-7,        8,9,10,-20,30,40,50,60,70,80,90,100
2 101,  102,103, -104
```

Выходные данные:

```
1 1 : all = 19 , neg = 4
2 2 : all = 4 , neg = 1
```

Программа отработала корректно.

3 Вывод

В результате выполнения работы были получены навыки работы с генератором программ синтаксической обработки текстов *yacc* и закреплены навыки работы с генератором программ лексической обработки текстов *lex*. Было изучено и запущено восемь программ, тексты которых были распространены преподавателем. Также была разработана собственная программа, в соответствии с индивидуальным заданием. Программы были транслированы в текста на языке C, скомпилированы и запущены с различными входными данными. Эксперименты, которые наиболее полно отражают суть работы программы, приведены в отчёте. Проведенная работа позволила изучить проблемы, возникающие при разработке и шаблоны, которые могу служить решением для них.