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

ОТЧЕТ
по лабораторной работе №2

Построение синтаксических анализаторов с помощью утилиты YACC

(тема работы)

Транслирующие системы

(наименование дисциплины)

Работу выполнили:

Волкова М.Д.

подпись Ф.И.О.

Преподаватель:

Цыган В.Н.

подпись Ф.И.О.

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 уасс.

```
%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

В качестве индивидуального задания предлагается написать транслятор оператора цикла FOR из языка Си в код ассемблера a86.

Грамматика

<оператор> ::= выражение | выражение оператор

<выражение> ::= FOR условие тело

< iii > ::= NAME

< iii > ::= NUM

< ind > ::= NAME

< ind_t > ::= NAME

< mass > ::= NAME

<FOR> ::= FOR

< условие > ::= ‘(‘cond’)'

<cond> ::= exp1 ';' exp2 ';' exp3

<exp1> ::= <ind> ‘=’ <iii>

<exp2> ::= <iii> ‘>’ <iii> | <iii> ‘<’ <iii> | <iii> EQUAL <iii> | <iii> IEQUAL <iii> | <iii> LREQ <iii> | <iii> GREQ <iii>

<exp3> ::= <ind> INC | <ind> DEC

< тело > ::= ‘{’наполнение’}’

< наполнение > ::= наполнение выражение

< наполнение > ::= выражение

< наполнение > ::= наполнение наполнение_2

< наполнение > ::= наполнение_2

<наполнение 2> ::= <ind> ‘=’ <iii> ‘;’ | <ind> ‘=’ <mass> ‘[’ ind_t ‘]’ ‘;’

for.l:

```
%{
    #include "y.tab.h"
%}

%%
"for"    {return FOR;}
"++" {return INC;}
"--" {return DEC;}
"=="    {return EQUAL;}
"!="    {return IEQUAL;}
">="    {return LEEQ;}
"<="    {return GREQ;}

[a-zA-Z][0-9a-zA-Z]*   { yylval.cval = strdup(yytext); return NAME; }
[0-9]+                   { yylval.cval = strdup(yytext); return NUM;}
"("    |
")"    |
"{"    |
"}"    |
";"    |
">" |
"<" |
"[" |
"]" |
"="    {return yytext[0];}
[ \n\t]    ;

%%

int yywrap() {return 1; }
```

for.y:

```
%union {
    char* cval;
};

%{
    #include <stdlib.h>
    #include <string.h>
    #define    YYDEBUG    1
    extern int yydebug;
    int count = 0;
    int count2 = 1;
    %}

%token  FOR
%token  <cval>NAME
%token  <cval>NUM
%token  <cval>EQUAL
%token  <cval>IEQUAL
%token  <cval>LEEQ
%token  <cval>GREQ
%token  INC
%token  DEC
%start    op
```

```
%%
op: ex | ex op;

ex:    for conditional body;

for    :    FOR        {count++;};

conditional        :    '('cond')'

cond : exp1 ';'  exp2  ';' exp3

ind_t      :    NUM {$<cval>$ = $<cval>1;};
ind        :    NAME {$<cval>$ = $<cval>1;};
iii        :    NAME {$<cval>$ = $<cval>1;};
iii        :    NUM {$<cval>$ = $<cval>1;};

exp1        :  ind '=' iii {
    printf("    mov %s, %s\n", $<cval>1, $<cval>3);
    printf("F%d:\n", count);
};


exp2: iii '<' iii { printf("    mov ax, %s\n", $<cval>1);
    printf("    cmp ax, %s\n", $<cval>3);
    printf("    jge D%d\n", count2);};

exp2: iii '>' iii { printf("    mov ax, %s\n", $<cval>1);
    printf("    cmp ax, %s\n", $<cval>3);
    printf("    jle D%d\n", count2);};

exp2: iii EQUAL iii { printf("    mov ax, %s\n", $<cval>1);
    printf("    cmp ax, %s\n", $<cval>3);
    printf("    jne D%d\n", count2);
};

exp2: iii IEQUAL iii { printf("    mov ax, %s\n", $<cval>1);
    printf("    cmp ax, %s\n", $<cval>3);
    printf("    je D%d\n", count2);};

exp2: iii LEEQ iii { printf("    mov ax, %s\n", $<cval>1);
    printf("    cmp ax, %s\n", $<cval>3);
    printf("    jg D%d\n", count2);};

exp2: iii GREQ iii { printf("    mov ax, %s\n", $<cval>1);
    printf("    cmp ax, %s\n", $<cval>3);
    printf("    jl D%d\n", count2);};


exp3 : ind INC  {printf("    add %s, 1\n", $<cval>1);};
exp3 : iii DEC  {printf("    sub %s, 1\n", $<cval>1);};

body : '{'bexp'}'
bexp : bexp ex;
bexp : ex;
bexp : bexp const;
bexp : const;
const: ind '=' iii ';'{
    printf("D%d\n", count2);
    count2++;
    printf("    mov %s, %s\n", $<cval>1, $<cval>3);
```

```
    printf("    loop F%d\n", count);
};

const        :   ind '=' iii '[' ind_t ']' ';'{
    printf("D%d:\n", count2);
    count2++;
    printf("    lea %s, %s\n", $<cval>1,$<cval>3);
    printf("    mov cx, len\n");
    printf("    xor ax, ax\n");
    printf("cikl:\n     inc bx\n");
    printf("    loop cikl\n");
};

%%
```

Тестирование

Проверим программу на простейшем случае:
in.txt:

```
for (  i = 0; i < 10; i--) {
a = 10;
}
```

out.txt:

```
    mov i, 0
F1:
    mov ax, i
    cmp ax, 10
    jge D1
D1
    sub i, 1
    mov a, 10
    loop F1
```

Видно, что программа работает корректно.

Попробуем запустить нашу программу на обработку несколько последовательных функций.

in.txt:

```
for ( k = 0 ; k != 10; k++) {
a = mass[4];
}

for ( k = 0 ; k != 10; k++) {
a = mass[4];
}

for ( k = 0 ; k != 10; k++) {
a = mass[4];
}
```

out.txt:

```
    mov k, 0
F1:
    mov ax, k
    cmp ax, 10
    je D1
D1
    add k, 1
    lea a, mass
    mov cx, len
    xor ax, ax
cikl1:
     inc bx
    loop cikl1
    mov k, 0
F2:
    mov ax, k
```

```
    cmp ax, 10
    je D2
D2
    add k, 1
    lea a, mass
    mov cx, len
    xor ax, ax
cikl2:
     inc bx
    loop cikl2
    mov k, 0
F3:
    mov ax, k
    cmp ax, 10
    je D3
D3
    add k, 1
    lea a, mass
    mov cx, len
    xor ax, ax
cikl3:
     inc bx
    loop cikl3
```

Программа опять выдала правильный результат.

Теперь попробуем запустить с вложенными циклами.

in.txt:

```
for ( k = 0 ; k != 10; k++) {
for (  i = 0; i < 10; i--){
a = 5;
}}
```

out.txt:

```
    mov k, 0
F1:
    mov ax, k
    cmp ax, 10
    je D1
D1
    add k, 1
    mov i, 0
F2:
    mov ax, i
    cmp ax, 10
    jge D2
D2
    sub i, 1
    mov a, 5
    loop F2
```

Видим, что программа работает корректно во всех случаях

После встречи с преподавателем, были внесены некоторые изменения:

1. Исправлена ошибка, в результате которой имя переменной могло начинаться с цифр.

2. Исправлена часть кода с присвоением переменной элемента массива:

```
const        :   ind '=' mass '[' ind_t ']' ';'{
    printf("    lea bx, %s\n", $<cval>3); //
    printf("    add bx, [%s]\n",  $<cval>5);
    printf("    mov %s, [bx]\n",  $<cval>1);
};
```

После изменений протестируем программу еще раз, на все возможные примеры:

in.txt:

```
for ( i = 0 ; i != 10; i++) {
a = mass[5];
for ( k = 6 ; k <= 3; k--) {
b = 5;
c = b;
}
}

for ( z = 2 ; z == 1; k--) {
a = qwe[23];
}
```

out.txt:

```
    mov i, 0
F1:
    mov ax, i
    cmp ax, 10
    je D1
D1
    add i, 1
    lea bx, mass
    add bx, [5]
    mov a, [bx]
    mov k, 6
F2:
    mov ax, k
    cmp ax, 3
    jl D2
D2
    sub k, 1
    mov b, 5
    mov c, b
```

```
    loop F2
    loop F1
    mov z, 2
F3:
    mov ax, z
    cmp ax, 1
    jne D3
D3
    sub k, 1
    lea bx, qwe
    add bx, [23]
    mov a, [bx]
    loop F3
```

Видим, что программа работает корректно во всех случаях

Полный код программы после исправления представлен на страницах 27-28.

Вывод

В ходе работы были рассмотрены основные принципы работы с программой YACC. Программа YACC создает синтаксические анализаторы, которые определяют и контролируют структуру текстового ввода компьютерной программы. На примерах рассмотрена структура и синтаксис YACC - программы. Полученные знания были обобщены при работе над индивидуальным заданием.

В ходе работы над индивидуальным заданием, был успешно создан синтаксически-ориентированный транслятор, который преобразовывает код оператор цикла for на языке си, в код ассемблера a86. Самое сложное в индивидуальном задании было контролировать метки, вложенных циклов. Но в результате работы эта проблема была решена.

Код модуля for.l:

```
%{
    #include "y.tab.h"
%}

%%
"for"    {return FOR;}
"++" {return INC;}
"--" {return DEC;}
"=="    {return EQUAL;}
"!="    {return IEQUAL;}
">="    {return LEEQ;}
"<="    {return GREQ;}

[a-zA-Z][0-9a-zA-Z]*   { yylval.cval = strdup(yytext); return NAME; }
[0-9]+                   { yylval.cval = strdup(yytext); return NUM;}
"("    |
")"    |
"{"    |
"}"    |
";"    |
">" |
"<" |
"[" |
"]" |
"="    {return yytext[0];}
[ \n\t]    ;

%%

int yywrap() {return 1; }
```

Код модуля ffor.y:

```
%union {
    char* cval;
};

%{
    #include <stdlib.h>
    #include <string.h>
    #define    YYDEBUG    1
    extern int yydebug;
    int count = 0;
    int count2 = 1;
        int m = 0;
    %}

%token  FOR
%token  <cval>NAME
%token  <cval>NUM
%token  <cval>EQUAL
%token  <cval>IEQUAL
%token  <cval>LEEQ
%token  <cval>GREQ
%token  INC
%token  DEC
%start    op

%%
op: ex | ex op;

ex:    for conditional body;

for    :    FOR        {count++;};

conditional        :    '('cond')'

cond : exp1 ';'  exp2  ';' exp3

ind_t      :    NUM {$<cval>$ = $<cval>1;};
ind        :    NAME {$<cval>$ = $<cval>1;};
mass        :    NAME {$<cval>$ = $<cval>1;};

iii        :    NAME {$<cval>$ = $<cval>1;};
iii        :    NUM {$<cval>$ = $<cval>1;};

exp1        :  ind '=' iii {
    printf("    mov %s, %s\n", $<cval>1, $<cval>3);
    printf("F%d:\n", count);
};


exp2: iii '<' iii { printf("    mov ax, %s\n", $<cval>1);
    printf("    cmp ax, %s\n", $<cval>3);
    printf("    jge D%d\n", count2);};

exp2: iii '>' iii { printf("    mov ax, %s\n", $<cval>1);
    printf("    cmp ax, %s\n", $<cval>3);
    printf("    jle D%d\n", count2);};

exp2: iii EQUAL iii { printf("    mov ax, %s\n", $<cval>1);
```

```
    printf("    cmp ax, %s\n", $<cval>3);
    printf("    jne D%d\n", count2);
};

exp2: iii IEQUAL iii { printf("    mov ax, %s\n", $<cval>1);
    printf("    cmp ax, %s\n", $<cval>3);
    printf("    je D%d\n", count2);};

exp2: iii LEEQ iii { printf("    mov ax, %s\n", $<cval>1);
    printf("    cmp ax, %s\n", $<cval>3);
    printf("    jg D%d\n", count2);};

exp2: iii GREQ iii { printf("    mov ax, %s\n", $<cval>1);
    printf("    cmp ax, %s\n", $<cval>3);
    printf("    jl D%d\n", count2);};


exp3 : ind INC  {
    printf("D%d\n", count2);
    count2++;
    printf("    add %s, 1\n", $<cval>1);};
exp3 : iii DEC  {
    printf("D%d\n", count2);
    count2++;
    printf("    sub %s, 1\n", $<cval>1);};

body : '{'bexp'}'
bexp : bexp ex;
bexp : ex;
bexp : bexp const;
bexp : const;
const: ind '=' iii ';'{
    printf("    mov %s, %s\n", $<cval>1, $<cval>3);
    printf("    loop F%d\n", count);
    count--;
    if (count != 0){printf("    loop F%d\n", count);}
};

const        :   ind '=' mass '[' ind_t ']' ';'{
    printf("    lea bx, %s\n", $<cval>3); //
    printf("    add bx, [%s]\n",  $<cval>5);
    printf("    mov %s, [bx]\n",  $<cval>1);
    printf("    loop F%d\n", count);
    count--;
    if (count != 0){printf("    loop F%d\n", count);}
};

%%
```