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 Ерниязов Т.Е.

Работу принял преподаватель Цыган В.Н.

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 уасс:

```
%{
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

<оператор> ::= выражение | выражение оператор

< выражение > ::= DO тело WHILE условие

< DO > ::= DO

< WHILE > ::= WHILE

< условие > ::= ‘('тело условия')'

< тело условия > ::= <iii> ‘>’ <iii> | <iii> ‘<’ …

< iii > ::= NAME

< iii > ::= NUM

< тело > ::= ‘{’наполнение’}’

< наполнение > ::= наполнение выражение

< наполнение > ::= выражение

< наполнение > ::= наполнение наполнение2

< наполнение > ::= наполнение2

<наполнение 2> ::= <iii>

ind.l:

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

%%
"do"        {return DO;}
"while"                 {return WHILE;}
"=="    {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; }
```

ind.y:

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

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

%token  DO WHILE
%token    <cval>NAME
%token    <cval>NUM
%token  <cval>EQUAL
%token  <cval>IEQUAL
%token  <cval>LEEQ
%token  <cval>GREQ
%start    op

%%
op: ex | ex op;

ex:    start_do body start_while cond;

start_do : DO {count_do++;
    printf("    D%d:\n", count_do);};

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

body: '{'exp'}'
exp : exp ex;
exp : ex;
exp : exp exp2;
exp : exp2;
exp2 : ind '=' iii ';'{ printf("    mov %s, %s\n", $<cval>1,  $<cval>3);};

start_while :   WHILE

cond : '('cond_b')'

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

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

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

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

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

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

```
                count_do--;};

%%
```

Вход:

```
do{
do{
tim = 5;
 } while (boom != 10)
 } while (poom >= 10);
```

Выход:

```
D1:
D2:
   mov tim, 5
   mov ax, boom
   cmp ax, 10
   je D2
   mov ax, poom
   cmp ax, 3
   jg D1
```

Видно, что программа работает корректно, даже со вложенными циклами.

После встречи с преподавателем, в программу было внесено пару изменений:

1. Были добавлены простые операции (сложение, умножение, вычитание и деление) в теле цикла:

```
exp2    :    ind '=' iii  ';'{ printf("    mov %s, %s\n", $<cval>1, $<cval>3);};
exp2    :    ind '=' iii  { printf("    mov %s, %s\n", $<cval>1, $<cval>3);};
exp2    :   exp2 '+' iii ';'{ printf("    add %s, %s\n", $<cval>1, $<cval>3);};
exp2    :   exp2 '-' iii ';'{ printf("    sub %s, %s\n",$<cval>1, $<cval>3);};
exp2    :   exp2 '*' iii ';'{ printf("    mul %s, %s\n",$<cval>1, $<cval>3);};
exp2    :   exp2 '/' iii ';'{ printf("    div %s, %s\n",$<cval>1, $<cval>3);};
```

2. Была добавлена формулировка задания:

В качестве индивидуального задания предлагается написать транслятор оператора цикла do-while из языка Си в код ассемблера a86. Должны быть реализованы следующие условные выражения: равенство, не равенство, больше или равно, меньше или равно, больше меньше. В теле цикла возможны операции сложения, вычитания, умножения, деления и присваивания константе или переменной. Должны быть реализованы вложенные и последующие друг за другом циклы.

3. Были внесены соответствующие изменения в грамматику:

<оператор> ::= выражение | выражение оператор

< выражение > ::= DO тело WHILE условие

< DO > ::= DO

< WHILE > ::= WHILE

< условие > ::= ‘('тело условия')'

< тело условия > ::= <iii> ‘>’ <iii> | <iii> ‘<’ <iii> | <iii> ‘!=’ <iii> | <iii> ‘==‘ <iii> | <iii> ‘>=’ <iii> | <iii> ‘<=’ <iii>

< iii > ::= NAME

< iii > ::= NUM

< ind > ::= NAME

< тело > ::= ‘{’наполнение’}’

< наполнение > ::= наполнение выражение

< наполнение > ::= выражение

< наполнение > ::= наполнение наполнение2

< наполнение > ::= наполнение2

<наполнение 2> ::= <ind> ‘=’ <iii> ‘;’ | <ind> ‘=’ <iii> | <наполнение 2> ‘+’ <iii> | <наполнение 2> ‘-‘ <iii> | <наполнение 2> ‘*’ <iii> | <наполнение 2> ‘/’ <iii>

Полный код программы можно найти в листинге 1-2 в конце отчета.

После изменений было еще раз проведено тестирование программы:

Вход:

```
do{
a = b + 2;
c = d * a;
b = a-c;
z = 2/3;
d = b;
} while (t >= 3)

do{
a1 = a2;
} while (a1 < 2)

do{
do{
b = b + c;
} while (k != 12)
} while (b == 13)
```

Выход:

```
D1:
   mov a, b
   add a, 2
   mov c, d
   mul ax, a
   mov b, a
   sub ax, c
   mov z, 2
   div ax, 3
   mov d, b
   mov ax, t
   cmp ax, 3
   jg D1

D2:
```

```
    mov a1, a2
    mov ax, a1
    cmp ax, 2
    jge D2

D3:
D4:
    mov b, b
    add b, c
    mov ax, k
    cmp ax, 12
    je D3
    mov ax, b
    cmp ax, 13
    jne D4
```

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 ходе работы над индивидуальным заданием, был успешно реализован транслятор для оператора си do-while на язык ассемблера a86. Самым сложной частью работы было разработка грамматике для вложенных циклов.

Листинг.

1. Код программы lex;

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

%%
"do"    {return DO;}
"while"        {return WHILE;}
"=="    {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; }
```

2. Код программы yacc:

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

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

%token  DO WHILE
%token    <cval>NAME
%token    <cval>NUM
%token  <cval>EQUAL
%token  <cval>IEQUAL
%token  <cval>LEEQ
%token  <cval>GREQ
%start    op

%%
op: ex | ex op;

ex:    start_do body start_while cond;

start_do : DO {count_do++;
   printf("    D%d:\n", count_do);};

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

body: '{'exp'}'
exp : exp ex;
exp : ex;
exp : exp exp2;
exp : exp2;
exp2        :     ind '=' iii  ';'{ printf("    mov %s, %s\n", $<cval>1, $<cval>3);};
exp2        :     ind '=' iii  { printf("    mov %s, %s\n", $<cval>1, $<cval>3);};
exp2        :    exp2 '+' iii ';'{ printf("    add %s, %s\n", $<cval>1, $<cval>3);};
exp2        :    exp2 '-' iii ';'{ printf("    sub %s, %s\n",$<cval>1, $<cval>3);};
exp2        :    exp2 '*' iii ';'{ printf("    mul %s, %s\n",$<cval>1, $<cval>3);};
exp2        :    exp2 '/' iii ';'{ printf("    div %s, %s\n",$<cval>1, $<cval>3);};

start_while :   WHILE

cond : '('cond_b')'

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

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

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

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

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

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

%%
```