Санкт-Петербургский государственный политехнический университет

Факультет технической кибернетики

Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе №4

«Индивидуальное задание. Построение синтаксического анализатора (утилиты yacc и lex)»

Работу выполнил студент группы № 4081/12

Дорофеев Юрий Владимирович

Работу принял преподаватель _____

Цыган Владимир Николаевич

г. Санкт-Петербург

2012

1.Задание работы

Построить транслятор условного оператора if-else языка С в последовательность тетрад матрицы синтаксического дерева. В качестве условия – выражения условной операции сравнения (>,<,==,!=,<=,>=) Если результат не ноль (истина) выполняется тело цикла. Тело цикла состоит из операторов присваивания. А так же добавить возможность работы с массивами.

В трансляторе будем использовать переменные типа int и float.

2. Выполнение задания

Транслятор состоит из трех файлов: pars.y, pars.l и zz.c.

- Основная программа (zz.c):

```
dorofeev@dorofeev-VirtualBox:~/indyacc$ zz.c
#include <stdio.h>
extern int yydebug;
main() {
  yydebug = 1;
  return yyparse();
}
yyerror(char * s) {
  fprintf( stderr, "%s\n", s);
}
```

- Содержимое лексического анализатора (pars.y):

```
dorofeev@dorofeev-VirtualBox:~/indyacc$ pars.y
%{
#include "y.tab.h"
%}
%%
"if"       {return IF;}
"else"          {return ELSE;}
"=="       {return EQUAL;}
"!="       {return IEQUAL;}
"<="       {return LEEQ;}
">="       {return GREQ;}
([a-zA-Z][0-9a-zA-Z]*)("["[0-9]*"]")?   { yylval.cval =
strdup(yytext); return VARNAME; }
[0-9]+"."?[0-9]*{ yylval.cval = strdup(yytext);  return VARVALUE;}
"("   |
")"   |
"{"   |
"}"   |
";"   |
"="   |
">"   |
"<"    {return yytext[0];}
[ \n\t]    ;
%%
int yywrap() {return 1; }
```

- Содержимое синтаксического анализатора (pars.y):

```
dorofeev@dorofeev-VirtualBox:~/indyacc$ cat pars.y
%union {
char* cval;
};
%{

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
int count = 1;
int ifcount = 1;
int ifcount2 =1;
int stack[100];
int  sp = 0;
%}

%token  IF ELSE EQUAL GREQ IEQUAL LEEQ
%token      <cval>VARNAME
%token      <cval>VARVALUE
%start      op


%%
op          :     ex | ex op;
ex          :     start_if cond body | start_else cond2 body

start_if    :     IF
start_else  :     ELSE;

cond2       :     {printf("else\n"); push(ifcount); ifcount++;};

cond        :     '('cond_body')'                     {push(ifcount);
ifcount++;};

cond_body   :     iii '>' iii
                  { printf("T%d: > %s %s\n", count, $<cval>1, $<cval>3);
count++;
                  printf("T%d: JZ L%d\n", count, ifcount); count++;};
cond_body   :     iii '<' iii
                  { printf("T%d: < %s %s\n", count, $<cval>1, $<cval>3);
count++;
                  printf("T%d: JZ L%d\n", count, ifcount); count++;};
cond_body   :     iii EQUAL iii
                  { printf("T%d: == %s %s\n", count, $<cval>1, $<cval>3);
count++;
                  printf("T%d: JZ L%d\n", count, ifcount); count++;};
cond_body   :     iii IEQUAL iii
                  { printf("T%d: != %s %s\n", count, $<cval>1, $<cval>3);
count++;
                  printf("T%d: JZ L%d\n", count, ifcount); count++;};
cond_body   :     iii LEEQ iii
                  { printf("T%d: <= %s %s\n", count, $<cval>1, $<cval>3);
count++;
                  printf("T%d: JZ L%d\n", count, ifcount); count++;};
cond_body   :     iii GREQ iii
                  { printf("T%d: >= %s %s\n", count, $<cval>1, $<cval>3);
count++;
                  printf("T%d: JZ L%d\n", count, ifcount); count++;};
ind         :     VARNAME
                  {$<cval>$ = $<cval>1;};
iii         :     VARNAME
```

```
                    {$<cval>$ = $<cval>1;};
iii         :       VARVALUE
                    {$<cval>$ = $<cval>1;};
body        :       '{' assign1 '}' {printf("L%d:\n", pop());};
assign1             :       assign1 ex;
assign1             :       ex;
assign1             :       assign1 assign;
assign1             :       assign;
assign              :       ind '=' iii ';'
                    {printf("T%d: =  %s %s\n", count, $<cval>1, $<cval>3);
count++;};



%%

int push (int i)
{
if (sp>100)
{
printf ("ERROR: stack overflow");
return 1;
}
stack[sp++] = i;
return 0;
}
int pop (void)
{
if (sp==0)
{
printf ("ERROR: empty stack");
return 2;
}
return stack[--sp];
}
```

Транслятор обрабатывает любое число последовательных и вложенных условных операторов. Переменная count является счетчиком ячеек памяти, переменная ifcount – счетчиком меток переходов условного оператора. Push и pop – классическое помещение в стек/из стека

Для сохранения порядка меток типа L.

Формат записи

<метка>

<ячейка> <операция> <операнд 1> <операнд 2>

Знак равенства обозначает операцию присваивания, двойной – проверка на равенство. Присваивание числу переменной- невозможно.

В трансляторе используются следующие метки:

- L1, L2 – метки обозначающие конец условного оператора, место куда осуществляется переход при невыполнении условия;
- Else - метка начала выполнения цикла else;
- JZ – (JumpZero по аналогии с ассемблером) – переход при условии, что результат предыдущей операции – нуль, то есть ложь.

3. Тестирование работы транслятора

- Входной файл (test.in):

```
dorofeev@dorofeev-VirtualBox:~/indyacc$ cat test.in
if (x<=y)
{
c[5] = d;
a = 3.3;
      if (x>a)
      {
            if (x==a)
            {
                  d[234]=2.3;
            }


            if (a>v)
            {
                  s=ss;
            }
      }
f = d;
}

if (a>v)
{
s=ss;
}
else
{
b=5.2;
}
```

Для запуска транслятора выполним следующую последовательность действий:

```
dorofeev@dorofeev-VirtualBox:~/indyacc$ gcc -c -o zz.obj zz.c
dorofeev@dorofeev-VirtualBox:~/indyacc$ yacc -vdt pars.y
dorofeev@dorofeev-VirtualBox:~/indyacc$  gcc -c -o y.obj y.tab.c
dorofeev@dorofeev-VirtualBox:~/indyacc$ flex -lw pars.l
dorofeev@dorofeev-VirtualBox:~/indyacc$ cc -c -o scaner.obj lex.yy.c
dorofeev@dorofeev-VirtualBox:~/indyacc$ cc -o ifelse zz.obj y.obj scaner.obj
dorofeev@dorofeev-VirtualBox:~/indyacc$ ./ifelse <test.in> test.out
```

- Выходной файл (test.out):

```
dorofeev@dorofeev-VirtualBox:~/indyacc$ cat test.out
T1: <= x y
T2: JZ L1
T3: =  c[5] d
T4: =  a 3.3
T5: > x a
T6: JZ L2
T7: == x a
T8: JZ L3
T9: =  d[234] 2.3
L3:
T10: > a v
T11: JZ L4
T12: =  s ss
```

```
L4:
L2:
T13: =  f d
L1:
T14: > a v
T15: JZ L5
T16: =  s ss
L5:
else
T17: =  b 5.2
L6:
```

Проанализировав выходной файл, мы поняли, что транслятор работает верно.

4. Выводы

В ходе работы был создан транслятор для обработки кода условного оператора на языке C, и вывода его в последовательность тетрад матрицы синтаксического дерева. Были получены соответствующие навыки работы с утилитами Yacc и Lex.