Казанский (Приволжский) федеральный университет
Институт вычислительной математики и информационных технологий

Отчёт по дисциплине «Пакеты прикладных программ»

Работу выполнил:
Студент 09-811 группы
Царьков Максим Вячеславович

Работу проверил:
Доцент кафедры теоретической кибернетики
Гусенков Александр Михайлович

Казань 2021

Оглавление

Индивидуальное задание

Дана команда языка Visual FoxPro,

1) построить лексический анализатор с помощью инструмента LEX для заданной команды;
2) построить синтаксический анализатор с помощью инструмента YACC;
3) построить корректный загрузочный модуль совместного использования генераторов LEX и YACC;
4) реализовать оператор присваивания с динамической таблицей переменных и возможность динамического вычисления выражений;
5) реализовать обработку входного текста на предмет синтаксических и динамических ошибок.

Команда для реализации:

RECALL command

Убирает пометки со строк, помеченных для удаления.

RECALL [Scope] [FOR lExpression1] [WHILE lExpression2] [NOOPTIMIZE] [IN nWorkArea | cTableAlias]

Scope : ALL, NEXT nRecords, RECORD nRecordNumber, REST.

Параметры:

Scope

Задаёт диапазон записей для вызова. Область по умолчанию для отзыва – текущая (следующая 1). Вызываются только те записи, которые попадают в указанный диапазон. Возможные области действия: ALL, NEXT nRecord, RECORD nRecordNumber и REST.

For lExpression

Указывает, что вызываются только записи, для которых значение *lExpression* равно true. Эта опция позволяет отфильтровать нежелательные записи.

While lExpression

Указывает условие, при котором записи вызываются до тех пор, пока lExpression равно true.

NOOPTIMIZE

Предотвращает вызов оптимизации Rushmode.

IN nWorkArea | cTableAlias

Указывает рабочую область или псевдоним таблицы, на которые влияет команда отзыва. Используем, чтобы указать рабочую область или таблицу за пределами текущей рабочей области.

Commands

Задаёт набор команд Visual FoxPro, которые будут выполняться до тех пор, пока значение lExpression будет равно true (.T.)

LOOP

Возвращает управление программой непосредственно обратно, чтобы выполнить DO WHILE. LOOP может быть помещён в любом месте между DO WHILE и ENDDO.

EXIT

Передаёт управление программой из цикла DO WHILE в первую команду, следующую за ENDDO. EXIT может быть размещён в любом месте между DO WHILE и ENDDO.

Выполнение задания

Для выполнения задания я установит UNIX-подобная среда WSL Ubuntu 20.1 LTS и интерфейс командной строки Windows Terminal.

В качестве инструмента построения лексического анализа текста был установлен генератор лексических анализаторов Flex.

Для построения синтаксического анализатора была установлена программа GNU Bison, предназначенная для автоматического создания синтаксических анализаторов по данному описанию грамматики.

Так же добавим систему сборки Make при помощи Makefile. Настроим конфигурацию для удобной работы, распределения файлов по директориям и

быстрой сборки и запуска тестовых примеров. Так же для удобства будет использовать систему контроля версий Git.

Реализованные дополнительные конструкции лексического анализатора:

-Реализован оператор присваивания, с проверкой инициализации переменной.

-Динамические списки переменных, взаимодействие с этим списком, операции добавления переменной, взятие значения из переменной, поиск переменной.

-Работа с различными стандартными операциями над целыми числами, такими как умножение, сложение, деление, вычитание, возведение в степень (реализация с целыми числами).

-Реализованы логические операторы.

-Проверка на переполнение целочисленной переменной в функциях сложения, вычитания, умножения и положительно определённого возведения в степень.

-Реализована функция обработки ошибок с сообщением об ошибке, определением места ошибки и её типа.

-Реализован вывод всех динамических переменных и их количество. Вывод количества ошибок.

-Проверка длинны идентификатора.

Тестовый пример: input.txt

```
var_a = 0;
a=5;
pow2stepen3 = 2^3;
pow2stepen10 = 22^4;
pow10stepen5div2000 = 10^5 / 2000;
pow100stepenMinus2 = 10000^(-2);
powIntOverflow = 2^100;
bool_1_AND_0 = (var_a AND a);
bool_1_OR_0= (1 or 0);
bool_NOT_1= (not 1);
bool_NOT_0= not    0;
bool_NOT_1OR0 = not (1 or 0);
bool_NOT_1AND0 = not (1 and 0);
bool_NOT0_AND_0 = (NOT 0) AND 0;
bool_NOT0_AND_1 = (NOT 1) AND 0;
bool_NOT1_AND_0 = (NOT 0) AND 1;
bool_NOT1_AND_1 = (NOT 1) AND 1;
bool_NOT0_OR_0 = (NOT 0) OR 0;
bool_NOT0_OR_1 = (NOT 1) OR 0;
bool_NOT1_OR_0 = (NOT 0) OR 1;
bool_NOT1_OR_1 = (NOT 1) OR 1;
boolMultNumber8 = (not 0 and 1) * 8;
boolMultNumber7 = (not (1 and 0)) * 7;
boolMultNumber0 = (not 1 and 0) * 8;
b=76/((357/5)*0);
c=(34*5-78/3)+28;
d=(-(45*76-1223+(8*78)));
_0Tr = (1 + (a *d) * 78 - 136);
f = (4*6-a);
a1 = 1 + 2147483647;
```

```
tt = (56*3^^5);
tt = (56-^5);
a2 = 2;
a3 = (2 + 2)^2;
a4 = (2^2 + ((a2 + a2)^4 + a3)^6 + a3)^8;
a5 = 11323;
a6 = 1111111111;
b2 = 1231;
b3 = a6 * b2;
LongLongLongIdentifier = 1;
b5 = -1;
b6 = 1;
Y&-=6;
Y=5&&8;
b=1;
longexpr0 = (10 + b * a3 / 3 - (1 +100000 * 200));
longexpr1 =7631455564/((857/54)*3131);
longexpr2=(3456*53-7831334/3)+28132;
d=(-(45*76-1223+(8*78)));
____var____ = (1 + (a *d) * 78 - 136);
useNotDefExpr = (4*6-ch);
a1 = 1;
tt = (56*3^^5);
tt = (56-^5);
Y&-=6;
Y=5&&8;
recall foR(f == (4 * 0));
RecAll FoR((f == 78 - 3));
RECAll For((f > 2 + 30));
reCall FOr((f >= 14 + 78));
recALL fOR((f <= 26 + 1));
ReCaLl fOr((f != 54 - 82));
```

```
ReCaLl fOr(f AND (54 - 82));
REcaLL FOR((25) OR (19 * 9));
REcaLL FOR((25 OR (19 * 9));
recall IN 5;
rEcall iN 600 + +_2;
recalL In c;
Recall in (b+35*c);
Recall in (b+35+*c);
rEcall iN 678-90*0+a1;
rEcall iN a111;
reCall ALL;
recAll All;
recaLl Rest;
recalL REst in;
REcall Next 5;
REcAll nExt a;
REcaLL NExt(b + 35 * c);
ReCall NEXt 678 - 90 * 0 + a1;
recAlL RECORD 5;
recall rECOrd a;
recall reCOrD (b + 35 * c);
recall recoRd 678 - 90 * 0 + a1;
recall whilE(d == (25 + 30));
recall whILE((d == 25 + 30));
recall WHIle((d > 25 + 30));
recall WhIlE((d >= 25 + 30));
recall WhIlE((d =< 25 + 30));
recall whILE((d <= 25 + 30));
recall whILE((d => 25 + 30));
recall wHILe((d != 25 + 30));
Recall WHILe((d == 25 + 30));
REcAll Nooptimize;
```

```
RECALl nooPTIMIZE;
RECAlL All fOr((d != 25 + 30)) wHILe((d != 25 + 30)) iN 678 - 90 * 0 + a1;
ReCALL REst WhIlE((d >= 25 + 30)) nooPTIMIZE;
recall NEXt 678 - 90 * 0 + a1 iN 678 - 90 * 0 + a1;
ReCALL REst WhIlE((d >= 25 + 30)) nooPTIMIZE iN (678!=4);
d = 25^3 + 1;
ReCALL REst WhIlE((d > 25^3)) nooPTIMIZE iN 678 > 90 * 0;
d = 25^3 - 2 ;
ReCALL REst WhIlE((d > 25^3)) nooPTIMIZE iN 678 > 90 * 0;
ReCALL REst WhIlE((1 and not d)) nooPTIMIZE iN (678^0 == 90 * 0);
ReCALL REst WhIlE((d >= 25 + 30)) nooPTIMIZE iN 678 - 90 * 0 + a1;
ReCALL REst WhIlE(not 1 and not not not not not 0) nooPTIMIZE iN 678 - 90 * 0 + a1;
ReCALL REst WhIlE(not 1 0 not and 1) nooPTIMIZE iN b2;
d= 500^2;
x = 0;
recall wHILe((d != 500^2));
recall wHILe((x = (d ==500^2)));
recall wHILe((25!= 25 + 30 / (30 and x)));
x=1;
recall wHILe((x / 0));
recall wHILe((x!= x + x / 0));
recall wHILe((d != 25 + 30));
x = x+10;
ReCALL REst WhIlE(x^x) nooPTIMIZE iN 100/x-10;
ReCALL REst WhIlE(x^(-1)) nooPTIMIZE iN (-10*(-29)+200/(-3));
ReCALL WhIlE((d >= 25 + 30)) iN x;
longlonglonglonglonglonglonglonglonglongExpression = 1 and 1 or 0 not 0;
```

Вывод тестового примера: result.txt

```
reading file 'input.txt'
-------|.....................................................................
     1 |var_a = 0;
     2 |a=5;
     3 |pow2stepen3 = 2^3;
     4 |pow2stepen10 = 22^4;
     5 |pow10stepen5div2000 = 10^5 / 2000;
     6 |pow100stepenMinus2 = 10000^(-2);
     7 |powIntOverflow = 2^100;
..... !.....................^   position: 23
Error: int overflow! Line 7:c19 to 7:c21
     8 |bool_1_AND_0 = (var_a AND a);
     9 |bool_1_OR_0= (1 or 0);
    10 |bool_NOT_1= (not 1);
    11 |bool_NOT_0= not    0;
    12 |bool_NOT_1OR0 = not (1 or 0);
    13 |bool_NOT_1AND0 = not (1 and 0);
    14 |bool_NOT0_AND_0 = (NOT 0) AND 0;
    15 |bool_NOT0_AND_1 = (NOT 1) AND 0;
    16 |bool_NOT1_AND_0 = (NOT 0) AND 1;
    17 |bool_NOT1_AND_1 = (NOT 1) AND 1;
    18 |bool_NOT0_OR_0 = (NOT 0) OR 0;
    19 |bool_NOT0_OR_1 = (NOT 1) OR 0;
    20 |bool_NOT1_OR_0 = (NOT 0) OR 1;
    21 |bool_NOT1_OR_1 = (NOT 1) OR 1;
    22 |boolMultNumber8 = (not 0 and 1) * 8;
    23 |boolMultNumber7 = (not (1 and 0)) * 7;
    24 |boolMultNumber0 = (not 1 and 0) * 8;
```

```
   25 |b=76/((357/5)*0);
...... !..............^.   position: 16
Error: division by zero! Line 25:c5 to 25:c15
   26 |c=(34*5-78/3)+28;
   27 |d=(-(45*76-1223+(8*78)));
   28 |_0Tr = (1 + (a *d) * 78 - 136);
   29 |f = (4*6-a);
   30 |a1 = 1 + 2147483647;
...... !................^   position: 20
Error: int overflow! Line 30:c10 to 30:c19
   31 |tt = (56*3^^5);
...... !.........^....   position: 11
Error: syntax error
...... !.........^....   position: 11
Error: wrong arifmetic expression
   32 |tt = (56-^5);
...... !........^....   position: 9
Error: syntax error
...... !........^....   position: 9
Error: wrong arifmetic expression
   33 |a2 = 2;
   34 |a3 = (2 + 2)^2;
   35 |a4 = (2^2 + ((a2 + a2)^4 + a3)^6 + a3)^8;
...... !..............................^.......   position: 34
Error: int overflow! Line 35:c31 to 35:c31
...... !..............................^.......   position: 34
Error: int overflow! Line 35:c13 to 35:c31
...... !.....................................^...   position: 38
Error: int overflow! Line 35:c36 to 35:c37
...... !.....................................^   position: 41
Error: int overflow! Line 35:c39 to 35:c39
   36 |a5 = 11323;
```

```
   37 |a6 = 1111111111;
   38 |b2 = 1231;
   39 |b3 = a6 * b2;
...... !...........^   position: 13
Error: int overflow! Line 39:c11 to 39:c12
   40 |LongLongLongIdentifier = 1;
   41 |b5 = -1;
   42 |b6 = 1;
   43 |Y&-=6;
...... !.^....   position: 2
Error: syntax error
...... !.^....   position: 2
Error: wrong identifier
   44 |Y=5&&8;
...... !...^...   position: 4
Error: syntax error
...... !...^...   position: 4
Error: wrong number
   45 |b=1;
   46 |longexpr0 = (10 + b * a3 / 3 - (1 +100000 * 200));
   47 |longexpr1 =7631455564/((857/54)*3131);
   48 |longexpr2=(3456*53-7831334/3)+28132;
   49 |d=(-(45*76-1223+(8*78)));
   50 |____var____ = (1 + (a *d) * 78 - 136);
   51 |useNotDefExpr = (4*6-ch);
...... !.....................^.   position: 24
Error: reference to unknown variable 'ch'
   52 |a1 = 1;
   53 |tt = (56*3^^5);
...... !.........^....   position: 11
Error: syntax error
...... !.........^....   position: 11
```

```
Error: wrong arifmetic expression
   54 |tt = (56-^5);
...... !........^....   position: 9
Error: syntax error
...... !........^....   position: 9
Error: wrong arifmetic expression
   55 |Y&-=6;
...... !.^....   position: 2
Error: syntax error
...... !.^....   position: 2
Error: wrong identifier
   56 |Y=5&&8;
...... !...^...   position: 4
Error: syntax error
...... !...^...   position: 4
Error: wrong number
   57 |recall foR(f == (4 * 0));
   58 |RecAll FoR((f == 78 - 3));
   59 |RECAll For((f > 2 + 30));
   60 |reCall FOr((f >= 14 + 78));
   61 |recALL fOR((f <= 26 + 1));
   62 |ReCaLl fOr((f != 54 - 82));
   63 |ReCaLl fOr(f AND (54 - 82));
   64 |REcaLL FOR((25) OR (19 * 9));
   65 |REcaLL FOR((25 OR (19 * 9));
...... !.........................^.   position: 27
Error: syntax error
...... !.........................^.   position: 27
Error: wrong arifmetic expression
   66 |recall IN 5;
   67 |rEcall iN 600 + +_2;
...... !..................^   position: 20
```

```
Error: reference to unknown variable '_2'
    68 |recalL In c;
    69 |Recall in (b+35*c);
    70 |Recall in (b+35+*c);
...... !.............^....   position: 16
Error: syntax error
...... !.............^....   position: 16
Error: wrong arifmetic expression
    71 |rEcall iN 678-90*0+a1;
    72 |rEcall iN a111;
...... !.............^   position: 15
Error: reference to unknown variable 'a111'
    73 |reCall ALL;
    74 |recAll All;
    75 |recaLl Rest;
    76 |recalL REst in;
...... !.............^   position: 15
Error: syntax error
...... !.............^   position: 15
Error: wrong arifmetic expression
    77 |REcall Next 5;
    78 |REcAll nExt a;
    79 |REcaLL NExt(b + 35 * c);
    80 |ReCall NEXt 678 - 90 * 0 + a1;
    81 |recAlL RECORD 5;
    82 |recall rECOrd a;
    83 |recall reCOrD (b + 35 * c);
    84 |recall recoRd 678 - 90 * 0 + a1;
    85 |recall whilE(d == (25 + 30));
    86 |recall whILE((d == 25 + 30));
    87 |recall WHIle((d > 25 + 30));
    88 |recall WhIlE((d >= 25 + 30));
```

```
    89 |recall WhIlE((d =< 25 + 30));
...... !.............^.............  position: 17
Error: syntax error
...... !.............^.............  position: 17
Error: wrong identifier
    90 |recall whILE((d <= 25 + 30));
    91 |recall whILE((d => 25 + 30));
...... !.............^.............  position: 17
Error: syntax error
...... !.............^.............  position: 17
Error: wrong identifier
    92 |recall wHILe((d != 25 + 30));
    93 |Recall WHILe((d == 25 + 30));
    94 |REcAll Nooptimize;
    95 |RECALl nooPTIMIZE;
    96 |RECAlL All fOr((d != 25 + 30)) wHILe((d != 25 + 30)) iN 678 - 90 * 0 + a1;
    97 |ReCALL REst WhIlE((d >= 25 + 30)) nooPTIMIZE;
    98 |recall NEXt 678 - 90 * 0 + a1 iN 678 - 90 * 0 + a1;
    99 |ReCALL REst WhIlE((d >= 25 + 30)) nooPTIMIZE iN (678!=4);
   100 |d = 25^3 + 1;
   101 |ReCALL REst WhIlE((d > 25^3)) nooPTIMIZE iN 678 > 90 * 0;
   102 |d = 25^3 - 2 ;
   103 |ReCALL REst WhIlE((d > 25^3)) nooPTIMIZE iN 678 > 90 * 0;
   104 |ReCALL REst WhIlE((1 and not d)) nooPTIMIZE iN (678^0 == 90 * 0);
   105 |ReCALL REst WhIlE((d >= 25 + 30)) nooPTIMIZE iN 678 - 90 * 0 + a1;
   106 |ReCALL REst WhIlE(not 1 and not not not not not 0) nooPTIMIZE iN 678 - 90 * 0 + a1;
   107 |ReCALL REst WhIlE(not 1 0 not and 1) nooPTIMIZE iN b2;
...... !.....................^.....................  position: 25
Error: syntax error
...... !.....................^.....................  position: 25
Error: wrong number
   108 |d= 500^2;
```

```
   109 |x = 0;
   110 |recall wHILe((d != 500^2));
   111 |recall wHILe((x = (d ==500^2)));
...... !.............^.............   position: 17
Error: syntax error
...... !.............^.............   position: 17
Error: wrong identifier
...... !.........................^..   position: 30
Error: syntax error
   112 |recall wHILe((25!= 25 + 30 / (30 and x)));
...... !.....................................^...   position: 39
Error: division by zero! Line 112:c30 to 112:c39
...... !.....................................^...   position: 39
Error: int overflow! Line 112:c25 to 112:c39
   113 |x=1;
   114 |recall wHILe((x / 0));
...... !................^...   position: 19
Error: division by zero! Line 114:c19 to 114:c19
   115 |recall wHILe((x!= x + x / 0));
...... !......................^...   position: 27
Error: division by zero! Line 115:c27 to 115:c27
...... !......................^...   position: 27
Error: int overflow! Line 115:c23 to 115:c27
   116 |recall wHILe((d != 25 + 30));
   117 |x = x+10;
   118 |ReCALL REst WhIlE(x^x) nooPTIMIZE iN 100/x-10;
...... !..................^.....................   position: 22
Error: int overflow! Line 118:c20 to 118:c20
   119 |ReCALL REst WhIlE(x^(-1)) nooPTIMIZE iN (-10*(-29)+200/(-3));
   120 |ReCALL WhIlE((d >= 25 + 30)) iN x;
   121 |longlonglonglonglonglonglonglonglonglongExpression = 1 and 1 or 0 not 0;
...... !^.....................................................   position: 1
```

Error: the name of variable is too long, Max lenght = 32, your lenght = 54

..... !.....^..... position: 56

Error: syntax error

..... !.....^..... position: 56

Error: wrong syntax

final content of variables

	Name----------	Value----------
1	: var_a	: 0
2	: a	: 5
3	: pow2stepen3	: 8
4	: pow2stepen10	: 234256
5	: pow10stepen5div2000	: 50
6	: pow100stepenMinus2	: 0
7	: powIntOverflow	: NULL
8	: bool_1_AND_0	: 0
9	: bool_1_OR_0	: 1
10	: bool_NOT_1	: 0
11	: bool_NOT_0	: 1
12	: bool_NOT_1OR0	: 0
13	: bool_NOT_1AND0	: 1
14	: bool_NOT0_AND_0	: 0
15	: bool_NOT0_AND_1	: 0
16	: bool_NOT1_AND_0	: 1
17	: bool_NOT1_AND_1	: 0
18	: bool_NOT0_OR_0	: 1
19	: bool_NOT0_OR_1	: 0
20	: bool_NOT1_OR_0	: 1
21	: bool_NOT1_OR_1	: 1
22	: boolMultNumber8	: 8
23	: boolMultNumber7	: 7

24 : boolMultNumber0 : 0

25 : b : 1

26 : c : 172

27 : d : 250000

28 : _0Tr : -1100325

29 : f : 19

30 : a1 : 1

31 : tt : 57

32 : a2 : 2

33 : a3 : 16

34 : a4 : NULL

35 : a5 : 11323

36 : a6 : 1111111111

37 : b2 : 1231

38 : b3 : NULL

39 : LongLongLongIdentifier : 1

40 : b5 : NULL

41 : b6 : 1

42 : Y : NULL

43 : longexpr0 : -19999986

44 : longexpr1 : -20408

45 : longexpr2 : -2399144

46 : ____var____ : -1100325

47 : useNotDefExpr : 24

48 : ch : NULL

49 : _2 : NULL

50 : a111 : NULL

51 : x : 11

==== count of variables - 51 ====

==== count of used var - 43 ====

==== count of unused var - 8 ====

==== count of errors - 51 ====

Листинг программы

Исходный код находится в Git репозитории по ссылке
https://github.com/s1Sharp/lex-yacc

/include/includes.h

```h
#ifndef INCLUDES_H
#define INCLUDES_H

#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include <string.h>
#include <memory.h>
#include <stdlib.h>
#include <stdarg.h>
#include <malloc.h>
#include <limits.h>

#endif // INCLUDES_H
```

/include/racall.h

```h
#ifndef RECALL_H
#define RECALL_H

#include "includes.h"
#include "y.tab.h"

//file recal.c
//control input
extern int debug;
extern int overflow;

extern int yylex(void);
extern int yyparse(void);
extern void yyerror(char*);

extern void DumpRow(void);
extern int  GetNextChar(char *b, int maxBuffer);
extern void NewToken(char*);
extern void PrintError(char *s, ...);


//file math.c
//simple math func
extern int ReduceAdd(int, int, YYLTYPE*);
extern int ReduceSub(int, int, YYLTYPE*);
extern int ReduceMult(int, int, YYLTYPE*);
```

```
extern int ReduceDiv(int, int, YYLTYPE*);
extern int ReducePow(int, int, YYLTYPE*);


//file var.c
//using dump values
typedef struct Variable {
  char* name;
  int   value;
  int   init;
} Variable;

extern Variable *VarGet(char*, YYLTYPE*);
extern void VarSetValue(Variable*, int);
extern int VarGetValue(char*, YYLTYPE*);
extern void DumpAllVariables(int errorcount);


#endif
```

/lex-yacc/lex.l

```
%option noyywrap

%{
#include "recall.h"

#define IDENTIFIER_MAX_LEN 32

#define YY_INPUT(buf,result,max_size)  {\
    result = GetNextChar(buf, max_size); \
    if (  result <= 0  ) \
      result = YY_NULL; \
    }

%}


Identifier  [_a-zA-Z][_a-zA-Z0-9]*
Number      0|([1-9][0-9]*)
recall      [Rr][Ee][Cc][Aa][Ll][Ll]
next        [Nn][Ee][Xx][Tt]
rest        [Rr][Ee][Ss][Tt]
nooptimize  [Nn][Oo][Oo][Pp][Tt][Ii][Mm][Ii][Zz][Ee]
record      [Rr][Ee][Cc][Oo][Rr][Dd]
for         [Ff][Oo][Rr]
while       [Ww][Hh][Ii][Ll][Ee]
```

```
in          [Ii][Nn]
all         [Aa][Ll][Ll]
and         [Aa][Nn][Dd]
or          [Oo][Rr]
not         [Nn][Oo][Tt]


%%
\xD;
{recall}            { NewToken(yytext);  return(yRECALL);}
{next}              { NewToken(yytext);  return(yNEXT);}
{rest}              { NewToken(yytext);  return(yREST);}
{nooptimize}        { NewToken(yytext);  return(yNOOPTIMIZE);}
{for}               { NewToken(yytext);  return(yFOR);}
{record}            { NewToken(yytext);  return(yRECORD);}
{while}             { NewToken(yytext);  return(yWHILE);}
{in}                { NewToken(yytext);  return(yIN);}
{all}               { NewToken(yytext);  return(yALL);}
{and}               { NewToken(yytext);  return(yAND);}
{or}                { NewToken(yytext);  return(yOR);}
{not}               { NewToken(yytext);  return(yNOT);}
\/              { NewToken(yytext);  return(SIGNDIV);}
\+              { NewToken(yytext);  return(SIGNPLUS);}
\-              { NewToken(yytext);  return(SIGNMINUS);}
\*              { NewToken(yytext);  return(SIGNMULT);}
\^              { NewToken(yytext);  return(SIGNPOW);}
\<              { NewToken(yytext);  return(SIGNLESS);}
\>              { NewToken(yytext);  return(SIGNMORE);}
\=              { NewToken(yytext);  return(SIGNEQ);}
\(              { NewToken(yytext);  return(SYMLP);}
\)              { NewToken(yytext);  return(SYMRP);}
"<="            { NewToken(yytext);  return(SIGNLEQ);}
">="            { NewToken(yytext);  return(SIGNMEQ);}
"=="            { NewToken(yytext);  return(SIGNEQQ);}
"!="|"<>"       { NewToken(yytext);  return(SIGNNEQ);}

{Number}      {
                 NewToken(yytext);
                 yylval.value = atoi(yytext);
                 return(NUMBER);
              }
{Identifier}  {
                if(yyleng > IDENTIFIER_MAX_LEN)
                                    PrintError("the name of variable is too
long, Max lenght = %d, your lenght = %d",
                                IDENTIFIER_MAX_LEN, yyleng);
                else{
                    yylval.string = malloc(strlen(yytext)+1);
                    strcpy(yylval.string, yytext);NewToken(yytext);
                    return (IDENTIFIER);
                }
```

```
              }

[ \r\t\n]+  { NewToken(yytext);};
\;        { NewToken(yytext); return(ySEMICOLON);}
.         { NewToken(yytext); return(yytext[0]);};

%%
```

/lex-yacc/yacc.y

```
%{
#include "recall.h"

static Variable *var;
void HandleError(char*s);
int setvalnull = 0;
int count = 0;

%}

%union {
  int   value;
  char  *string;
}

%token <string>   IDENTIFIER
%token <value>    NUMBER
%type <value>     expr
%token yIN yFOR yALL yNEXT yREST yRECORD yWHILE yRECALL yNOOPTIMIZE SYMLP SYMRP
yAND yNOT yOR ySEMICOLON SIGNEQ
    /*  in  for  all  next  rest  record  while  recall  nooptimize   (     )
and  not  or       ;       =  */

%left yOR
%left yAND
%left yNOT

%left SIGNEQQ SIGNNEQ
    /*  ==     != <>                                    */
%left SIGNLESS SIGNMORE SIGNLEQ SIGNMEQ
    /*   <        >       <=      >=                    */
%left SIGNPLUS SIGNMINUS
    /*   +         -                                    */
%left SIGNMULT SIGNDIV
    /*   *       /                                      */
%left SIGNPOW
    /*   ^                                              */

%start program
```

```
%%
program : stat ySEMICOLON {count = 0; setvalnull=0; }
        | program stat ySEMICOLON {count = 0; setvalnull=0; };

stat    : error {HandleError("wrong syntax");}
        | yRECALL scope forexpr whileexpr nooptim inn
        | IDENTIFIER { var = VarGet($1, &@1);} SIGNEQ expr { if ($4 == -1)
setvalnull=1; VarSetValue(var, $4);}
        | IDENTIFIER error{HandleError("wrong identifier");}
        ;

scope   :
        | yALL
        | yREST
        | yNEXT expr
        | yRECORD expr
        ;

forexpr :
        |  yFOR expr
        ;

whileexpr   :
            | yWHILE expr
            ;

inn :
    | yIN expr
    ;

nooptim :
        | yNOOPTIMIZE
        ;


expr    : SYMLP expr SYMRP      {   $$=  $2;  }
        | SIGNMINUS expr        {   $$= -$2;  }
        | expr SIGNPLUS expr    {   $$=ReduceAdd($1, $3, &@3);  }
        | expr SIGNMINUS expr   {   $$=ReduceSub($1, $3, &@3);  }
        | expr SIGNMULT expr    {   $$=ReduceMult($1, $3, &@3); }
        | expr SIGNDIV expr     {   $$=ReduceDiv($1, $3, &@3);  }
        | expr SIGNPOW expr     {   $$=ReducePow($1, $3, &@3);  }
        | NUMBER                {   $$= $1;  }
        | NUMBER error          {   HandleError("wrong number"); $$=-1; }
        | error                 {   HandleError("wrong arifmetic expression");
$$=-1;  }
        | IDENTIFIER error      {   HandleError("wrong identifier"); $$=-1;
setvalnull=0; }
        | IDENTIFIER            {   $$ = VarGetValue($1, &@1); }
        | expr SIGNEQQ expr     {   $$=$1==$3; }
```

```
        | expr SIGNNEQ expr     {   $$=$1!=$3; }
        | expr SIGNLESS expr    {   $$=$1<$3 ; }
        | expr SIGNMORE expr    {   $$=$1>$3 ; }
        | expr SIGNLEQ expr     {   $$=$1<=$3; }
        | expr SIGNMEQ expr     {   $$=$1>=$3; }
        | expr yAND expr        {   $$=$1&&$3; }
        | expr yOR expr         {   $$=$1||$3; }
        | yNOT expr             {   $$=!$2   ; }
        ;
%%

void HandleError(char*s)
{
if(count==0)
{PrintError(s);}
count++;
}


extern
void yyerror(char *s)
{
    PrintError(s);
}
```

/src/math.c

```c
#include "recall.h"

extern int setvalnull;

//reduce add with check int overflow
extern
int ReduceAdd(int a, int b, YYLTYPE *bloc) {
    if(b < 0)
        return ReduceSub(a, -b, bloc);
    if(INT_MAX - b < a) {
        PrintError("int overflow! Line %d:c%d to %d:c%d",
                        bloc->first_line, bloc->first_column,
                        bloc->last_line, bloc->last_column);
        setvalnull = 1;
        return INT_MAX;
    }
    return a + b;
}

//reduce sub with check int overflow
extern
int ReduceSub(int a, int b, YYLTYPE *bloc) {
    if(b < 0)
```

```
        return ReduceAdd(a, -b, bloc);
    if(INT_MIN + b > a) {
        PrintError("int overflow! Line %d:c%d to %d:c%d",
                        bloc->first_line, bloc->first_column,
                        bloc->last_line, bloc->last_column);
        setvalnull = 1;
        return INT_MIN;
    }
    return a - b;
}

//reduce mult with check int overflow
extern
int ReduceMult(int a, int b, YYLTYPE *bloc) {
    int sign = 1;
    if(a == 0 || b == 0) return 0;
    if(a < 0) { a = -a; sign = -sign; }
    if(b < 0) { b = -b; sign = -sign; }
    if(INT_MAX / b < a) {
        PrintError("int overflow! Line %d:c%d to %d:c%d",
                        bloc->first_line, bloc->first_column,
                        bloc->last_line, bloc->last_column);
        setvalnull = 1;
        return (sign > 0) ? INT_MAX : INT_MIN;
    }
    return sign * a * b;
}

//reduce div with check int overflow
extern
int ReduceDiv(int a, int b, YYLTYPE *bloc) {
    if (  b == 0  ) {
        PrintError("division by zero! Line %d:c%d to %d:c%d",
                        bloc->first_line, bloc->first_column,
                        bloc->last_line, bloc->last_column);
    setvalnull = 1;
    return INT_MAX;
    }
  return a / b;
}

//reduce pow with check int overflow
extern
int ReducePow(int a, int b, YYLTYPE *bloc) {
    int sign = 1;
    int tmpa = a,tmpb = b;
    if( b == 0) return 1;
    if(tmpa < 0) { tmpa = -a; sign = -sign; }
    if(tmpb < 0) { tmpb = -b; sign = -sign; }
    if((int)pow(INT_MAX, (double)1/tmpb) < tmpa) {
        PrintError("int overflow! Line %d:c%d to %d:c%d",
```

```
                        bloc->first_line, bloc->first_column,
                        bloc->last_line, bloc->last_column);
        setvalnull = 1;
        return (sign > 0) ? INT_MAX : INT_MIN;
    }
    return (int)pow((int)a, (int)b);
}
```

/src/recall.c

```
#include "recall.h"

#define true 1
#define false 0

int debug=0;

//init static variables for control tokens and variables
static FILE *file;
static int eof = 0;
static int nRow = 0;
static int nBuffer = 0;
static int lBuffer = 0;
static int nTokenStart = 0;
static int nTokenLength = 0;
static int nTokenNextStart = 0;
static int lMaxBuffer = 1000;
static int errorcount = 0;
static char *buffer;

static
char dumpChar(char c) {
    if (  isprint(c)  )
        return c;
    return '@';
}

static
char *dumpString(char *s) {
    static char buf[101];
    int i;
    int n = strlen(s);

    if (  n > 100  )
        n = 100;

    for (i=0; i<n; i++)
        buf[i] = dumpChar(s[i]);
    buf[i] = 0;
```

```
    return buf;
}

extern
void DumpRow(void) {
    if (  nRow == 0  ) {
        int i;
        fprintf(stdout, "-------|");
        for (i=1; i<71; i++)
            fprintf(stdout, ".");
        fprintf(stdout, "\n");
    }
    else
        fprintf(stdout, "%6d |%.*s", nRow, lBuffer, buffer);
}

extern
void PrintError(char *errorstring, ...) {
    errorcount ++;
    static char errmsg[10000];
    va_list args;

    int start=nTokenStart;
    int end=start + nTokenLength - 1;
    int i;

    if (  eof  ) {
        fprintf(stdout, "...... !");
        for (i=0; i<lBuffer; i++)
            fprintf(stdout, ".");
        fprintf(stdout, "^-EOF\n");
    }
    else {
        fprintf(stdout, "...... !");
        for (i=1; i<start; i++)
            fprintf(stdout, ".");
        for (i=start; i<=end; i++)
            fprintf(stdout, "^");
        for (i=end+1; i<lBuffer; i++)
            fprintf(stdout, ".");
        fprintf(stdout, "   position: %d\n", end);
    }
    va_start(args, errorstring);
    vsprintf(errmsg, errorstring, args);
    va_end(args);

    fprintf(stdout, "Error: %s\n", errmsg);
}

static
int getNextLine(void) {
```

```
    int i;
    char *p;

    nBuffer = 0;
    nTokenStart = -1;
    nTokenNextStart = 1;
    eof = false;

    p = fgets(buffer, lMaxBuffer, file);
    if (  p == NULL  ) {
        if (  ferror(file)  )
            return -1;
        eof = true;
        return 1;
    }

    nRow += 1;
    lBuffer = strlen(buffer);
    DumpRow();
    return 0;
}

extern
int GetNextChar(char *b, int maxBuffer) {
    int frc;

    if (  eof  )
        return 0;

    while (  nBuffer >= lBuffer  ) {
        frc = getNextLine();
        if (  frc != 0  )
            return 0;
        }

    b[0] = buffer[nBuffer];
    nBuffer += 1;

    if (  debug  )
        printf("GetNextChar() => '%c'0x%02x at %d\n",
                                                dumpChar(b[0]), b[0], nBuffer);
    return b[0]==0?0:1;
}

extern
void NewToken(char *t) {
    nTokenStart = nTokenNextStart;
    nTokenLength = strlen(t);
    nTokenNextStart = nBuffer;

    yylloc.first_line = nRow;
```

```
    yylloc.first_column = nTokenStart;
    yylloc.last_line = nRow;
    yylloc.last_column = nTokenStart + nTokenLength - 1;

    if (  debug  ) {
        printf("Token '%s' at %d:%d next at %d\n", dumpString(t),
                                                   yylloc.first_column,
                                                   yylloc.last_column,
nTokenNextStart);
    }
}

extern
int main(int argc, char *argv[]) {
    int i;
    char *infile=NULL;

    debug = 0;
    printf("  \n");

    for (i=1; i<argc; i++) {
        if (  strcmp(argv[i], "-d") == 0  ) {
            printf("debugging activated\n");
            debug = 1;
        }
        else
            infile = argv[i];
    }

    if (  infile == NULL  )
        infile = "input.txt";

    printf("reading file '%s'\n", infile);
    file = fopen(infile, "r");
    if (  file == NULL  ) {
        printf("cannot open input\n");
        return 12;
        }

    buffer = malloc(lMaxBuffer);
    if (  buffer == NULL  ) {
        printf("cannot allocate %d bytes of memory\n", lMaxBuffer);
        fclose(file);
        return 12;
    }

    DumpRow();
    if (  getNextLine() == 0  )
        yyparse();

    free(buffer);
```

```
    fclose(file);
    printf("\n\n\t\tfinal content of variables\n");

    //watch all variables
    DumpAllVariables(errorcount);
    return 0;
}
```

/src/recall.c

```
#include "recall.h"

#define MAX_NAME_LEN 32
#define MAXVARS 11

//number current variable in block
static int nVars = 0;
//number of blocks
static int N = 0;
//block ptr for variables table
static Variable** vars=NULL;
extern int setvalnull;

static
Variable* findVar(char* varname) {
    int i, j;
    if (varname == NULL)
        return NULL;
    if (N == 1) {
        for (j = 0; j < nVars; j++) {
            if (strcmp(vars[0][j].name, varname) == 0)
                return (*(vars)+j);
        }
    }
    else if (N == 0)  return NULL;

    else
    {
        for (i = 0; i < N; i++)
            for (j = 0; j < MAXVARS; j++)
            {
                if (i * MAXVARS + j >= nVars + (N - 1) * MAXVARS)
                    return NULL;

                if (strcmp(vars[i][j].name, varname) == 0)
                    return (*(vars + i) + j);
            }
    }
    return NULL;
```

```
}

static
Variable* addVar(char* varname) {
    //empty name
    if (varname == NULL)
        return NULL;
    //end of block or start
    if ((nVars == MAXVARS)||(N == 0)) {
        nVars = 0;
        //realloc memory: add prt for next Variable block
        vars = (Variable**)realloc(vars, (N + 1) * sizeof(Variable*));
        //allocate MAXVARS Variables
        vars[N] = (Variable*)malloc(MAXVARS * sizeof(Variable));
        //reduce count of block
        N += 1;
    }

    //insert default values
    vars[N-1][nVars].value = 0;
    vars[N-1][nVars].init = 0;
    int len = strlen(varname) + 1;
    if (len > MAX_NAME_LEN)
        vars[N-1][nVars].name = malloc(strlen(varname) + 1);
    else
        vars[N-1][nVars].name = malloc(MAX_NAME_LEN);

    //check allocate for name
    if (vars[N-1][nVars].name == NULL) {
        PrintError("internal error creating variable '%s'", varname);
        return NULL;
    }

    //copy name of var
    strcpy(vars[N-1][nVars].name, varname);
    //upd count of var
    nVars += 1;
    //return ptr for Variable
    return (*(vars +  (N - 1)) + (nVars - 1));
}

extern
Variable* VarGet(char* varname, YYLTYPE* bloc) {

    Variable* var;

    //find Variable
    var = findVar(varname);
    //if hasn`t Varialbe by "varname" do create new variable
    if (var == NULL)
        var = addVar(varname);
```

```
    return var;
}

extern
void VarSetValue(Variable* var, int value) {
    if ( var == NULL )
        return;
    if ( setvalnull == 1)
    {
        var->value = 0;
        var->init = 0;
        setvalnull = 0;
        return;
    }
    var->value = value;
    var->init = 1;
    return;
}

extern
int VarGetValue(char* varname, YYLTYPE* bloc) {

    Variable* var = NULL;

    //find Variable
    var = findVar(varname);
    //if hasn`t Varialbe by "varname" do create new variable
    if (var == NULL) {
        PrintError("reference to unknown variable '%s'", varname);
        var = addVar(varname);
        return 0;
    }
    if (var->init == 0){
        PrintError("variable not init %s", varname);
        setvalnull = 1;
        return 0;
    }
    return var->value;
}


extern
void DumpAllVariables(int errorcount) {
    int i,j;
    int used = 0;
    char formatsymbols[MAX_NAME_LEN-3];
    for (int i =0;i<MAX_NAME_LEN-3;i++)
    {
        formatsymbols[i] = '-';
    }
    printf("\tName%.*s Value----------\n",MAX_NAME_LEN-3,formatsymbols);
```

```
    for (i = 0; i < N-1; i++) {
        for (j = 0; j < MAXVARS; j++) {
            if (vars[i][j].init == 1)
            {
            printf("%d\t: %-*.*s: %d\n", i*MAXVARS+j+1,MAX_NAME_LEN,MAX_NAME_LEN,
                vars[i][j].name, vars[i][j].value);
                used++;
            }
            else
            printf("%d\t: %-*.*s: %s\n", i*MAXVARS+j+1,MAX_NAME_LEN,MAX_NAME_LEN,
                vars[i][j].name, "NULL");
        }
    }
    for (i = N-1; i < N; i++) {
        for (j = 0; j < nVars; j++) {
            if (vars[i][j].init == 1)
            {
            printf("%d\t: %-*.*s: %d\n", i*MAXVARS+j+1,MAX_NAME_LEN,MAX_NAME_LEN,
                vars[i][j].name, vars[i][j].value);
                used++;
            }
            else
            printf("%d\t: %-*.*s: %s\n", i*MAXVARS+j+1,MAX_NAME_LEN,MAX_NAME_LEN,
                vars[i][j].name, "NULL");
        }
    }
    printf("==== count of variables - %d ====\n", (N-1)*MAXVARS+j );
    printf("==== count of used var - %d ====\n", used);
    printf("==== count of unused var - %d ====\n", (N-1)*MAXVARS+j - used);
    printf("==== count of errors - %d ====\n",errorcount);
    return;
}
```

/Makefile

```
ly = lex-yacc/
src = src/
hdr = include/
build = build/

CC = gcc
PROGRAM   =  app
CFLAGS = -I$(hdr) -I$(build) -lfl -o
SRC = $(build)*.c $(src)*.c -lm
DEBUG ?=
FILE = input.txt

.PHONY: all clean run result
```

```
all: $(PROGRAM) run
    cat result.txt

y.tab.c: $(ly)yacc.y
    bison -b $(build)y -vd $(ly)yacc.y | yacc -b $(build)y -vd $(ly)yacc.y

lex.yy.c:  $(ly)lex.l
ifeq ($(OS),Windows_NT)
    flex -o $(build)lex.yy.c $(ly)lex.l
else
    lex -o $(build)lex.yy.c $(ly)lex.l
endif

app: mkdir y.tab.c lex.yy.c
    $(CC) $(SRC) $(CFLAGS) app

clean:
    rm -rf $(build)
    rm result.txt
    rm app*

mkdir:
    mkdir -p $(build)

run:
    ./app $(DEBUG) $(FILE) > result.txt

result:
    cat result.txt
```

Список литературы

1) Гусенков А.М., Прокопьев Н.А.Специализированные языки обработки информации /А.М. Гусен-ков, Н.А. Прокопьев. –Казань: Казан. ун-т, 2018. –95с.
2) Хопкрофт, Джон, Э.,Мотвани, Раджив,Ульман, Джеффри, Д..Х78Введениевтеориюавтоматов, языковивычислений, 2-еизд.. : Пер. сангл. —М. : Издательскийдом “Вильямс”, 2008. — 528 с. : ил. — Парал. тит. англ.
3) LEX & YACC TUTORIAL by Tom Niemann epaperpress.com
4) lex & yacc by John R. Levine, Tony Mason and Doug Brown