Казанский (Приволжский) федеральный университет

Институт вычислительной математики и информационных технологий

Отчёт по дисциплине «Пакеты прикладных программ»

студента гр. 09 – 812

Работу выполнил:

Студент 09-812 группы

Садыков Ильдар Наилевич

Работу проверил:

Доцент кафедры теоретической кибернетики

Гусенков Александр Михайлович

Казань 2021

# **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 вычисление арифметических выражений с динамической таблицей переменных;
5. реализовать диагностику 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 команды**

*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 была использована виртуальная машина VirtualBox на которую поставлен Unix подобная операционная система Ubuntu.

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.

Сборка производится с помощью Makefile, чтобы автоматизировать рутинные действия по преобразованию файлов в нужные форматы и компиляции.

Для поддержания версионности программы используется система контроля версий Git.

**Листинг**

**recall.h:**

#ifndef RECALL\_H

#define RECALL\_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>

#include "y.tab.h"

extern int debug;

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, ...);

extern int ReduceAdd(int, int, YYLTYPE\*);

extern int ReduceSub(int, int, YYLTYPE\*);

extern int ReduceMult(int, int, YYLTYPE\*);

extern int ReduceDiv(int, int, YYLTYPE\*);

typedef struct Variable {

char\* name;

int value;

} Variable;

extern Variable\* VarGet(char\*, YYLTYPE\*);

extern void VarSetValue(Variable\*, int);

extern int VarGetValue(char\*, YYLTYPE\*);

extern void DumpAllVariables(int errorCount);

#endif

**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(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("var name should be no more than:%d, your length:%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]);};

%%

**yacc.y:**

%{

#include "recall.h"

static Variable \*var;

void HandleError(char\*s);

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

/\* \* / \*/

%start program

%%

program : stat ySEMICOLON {count = 0;}

| program stat ySEMICOLON {count = 0;};

stat : error {HandleError("wrong syntax");}

| yRECALL scope forexpr whileexpr nooptim inn

| IDENTIFIER { var = VarGet($1, &@1);} SIGNEQ expr {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); }

| NUMBER { $$= $1; }

| NUMBER error { HandleError("wrong number"); $$=INT\_MAX; }

| error { HandleError("wrong arifmetic expression"); $$=INT\_MAX; }

| IDENTIFIER error { HandleError("wrong identifier"); $$=INT\_MAX; }

| 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);

}

**math.c:**

#include "recall.h"

extern int ReduceAdd(int a, int b, YYLTYPE\* bloc)

{

return a + b;

}

extern int ReduceSub(int a, int b, YYLTYPE\* bloc)

{

return a - b;

}

extern int ReduceMult(int a, int b, YYLTYPE\* bloc)

{

return a \* b;

}

extern int ReduceDiv(int a, int b, YYLTYPE\* bloc)

{

if (b == 0) {

PrintError("division by zero exception line %d:c%d to %d:c%d", bloc->first\_line, bloc->first\_column,

bloc->last\_line, bloc->last\_column);

return INT\_MAX;

}

return a / b;

}

**recall.c:**

#include "recall.h"

#include <stdbool.h>

int debug = 0;

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 = 1024;

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) {

fprintf(stdout, "-------|");

for (int 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;

}

**var.c:**

#include "recall.h"

#define MAX\_NAME\_LEN 32

#define MAXVARS 11

static int nVars = 0;

static int N = 0;

static Variable\*\* vars = NULL;

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)

{

if (varname == NULL) {

return NULL;

}

if ((nVars == MAXVARS) || (N == 0)) {

nVars = 0;

vars = (Variable\*\*)realloc(vars, (N + 1) \* sizeof(Variable\*));

vars[N] = (Variable\*)malloc(MAXVARS \* sizeof(Variable));

N += 1;

}

vars[N - 1][nVars].value = 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);

}

if (vars[N - 1][nVars].name == NULL) {

PrintError("internal error creating variable '%s'", varname);

return NULL;

}

strcpy(vars[N - 1][nVars].name, varname);

nVars += 1;

return (\*(vars + (N - 1)) + (nVars - 1));

}

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

{

Variable\* var = findVar(varname);

if (var == NULL) {

var = addVar(varname);

}

return var;

}

extern void VarSetValue(Variable\* var, int value)

{

if (var == NULL) {

return;

}

var->value = value;

return;

}

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

{

Variable\* var = NULL;

var = findVar(varname);

if (var == NULL) {

PrintError("reference to unknown variable '%s'", varname);

var = addVar(varname);

return 0;

}

return var->value;

}

extern void DumpAllVariables(int errorcount)

{

int i, j;

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++) {

printf("%d\t: %-\*.\*s: %u\n", i \* MAXVARS + j + 1, MAX\_NAME\_LEN, MAX\_NAME\_LEN, vars[i][j].name,

vars[i][j].value);

}

}

for (i = N - 1; i < N; i++) {

for (j = 0; j < nVars; j++) {

printf("%d\t: %-\*.\*s: %u\n", i \* MAXVARS + j + 1, MAX\_NAME\_LEN, MAX\_NAME\_LEN, vars[i][j].name,

vars[i][j].value);

}

}

printf("==== count of variables - %d ====\n", (N - 1) \* MAXVARS + j);

printf("==== count of errors - %d ====\n", errorcount);

return;

}

**Результат выполнения на тестовых данных**

-------|......................................................................

1 |var1 = 0;

2 |var2 = ggg \* 6;

...... !...........^... position: 12

Error: reference to unknown variable 'ggg'

3 |var3 = 10;

4 |var4 = 100 / ((205 / 2) \* 0);

...... !..........................^.. position: 27

Error: division by zero exception line 4:c14 to 4:c27

5 |var5 = (45 \* var3 - 80 / 3) + var1;

6 |var6 = (-(45 \* 20 - 1105 + (8 \* 6)));

7 |var7 = (1 + (a \* d) \* 78 - 136);

...... !...............^................ position: 16

Error: reference to unknown variable 'a'

...... !..................^............. position: 19

Error: reference to unknown variable 'd'

8 |var8 (4 \* 6 - var1);

...... !......^.............. position: 7

Error: syntax error

...... !......^.............. position: 7

Error: wrong identifier

9 |

10 |bool1 = (var1 AND var2);

11 |bool2 = (var1 or var3);

12 |bool3 = (1 or 0);

13 |bool4 = (0 and 1);

14 |bool5 = (not 1);

15 |bool6 = (not 0);

16 |bool7 = not (1 or 0);

17 |bool8 = not (1 and 0);

18 |bool9 = (NOT 0) AND 0;

19 |bool10 = (NOT 1) AND 0;

20 |bool11 = (NOT 0) AND 1;

21 |bool12 = (NOT 1) AND 1;

22 |bool13 = (NOT 0) OR 0;

23 |bool14 = (NOT 1) OR 0;

24 |bool15 = (NOT 0) OR 1;

25 |bool16 = (NOT 1) OR 1;

26 |

27 |LongLongLongLong = 500;

28 |LongLongLongLongLongLongLongLongLongLongLongLongLongLongLongLong = 2 \* 2;

...... !^........................................................................ position: 1

Error: var name should be no more than:32, your length:64

...... !.................................................................^....... position: 66

Error: syntax error

...... !.................................................................^....... position: 66

Error: wrong syntax

29 |

30 |longVar0 = (534 + 1234 \* 500654 / 3 - (1 + 3456 \* 200));

31 |longVar1 = 8012301203 / ((1143 / 1234) \* 352435131);

...... !..................................................^. position: 51

Error: division by zero exception line 31:c25 to 31:c51

32 |longVar2 = (3456 \* 53 - 7831334 + 3) + 1234;

33 |

34 |a = 0;

35 |b = 10;

36 |c = 20;

37 |f = 10;

38 |x = 5;

39 |recall foR(f == (5 \* 0));

40 |RecAll FoR((f == 80 - 3));

41 |RECAll For((f > 6 + 50));

42 |reCall FOr((f >= 15 + 28));

43 |recALL fOR((f <= 26));

44 |ReCaLl fOr((f != 54 - a));

45 |ReCaLl fOr(f AND (54 - 10));

46 |REcaLL FOR((25) OR (20 \* 9));

47 |REcaLL FOR((25 OR (20 \* 9));

...... !..........................^. position: 27

Error: syntax error

...... !..........................^. position: 27

Error: wrong arifmetic expression

48 |

49 |recall IN 5;

50 |recalL In c;

51 |Recall in (b+35\*c);

52 |Recall in (b+35+\*c);

...... !...............^.... position: 16

Error: syntax error

...... !...............^.... position: 16

Error: wrong arifmetic expression

53 |rEcall iN 678-90\*0+a1;

...... !.....................^ position: 22

Error: reference to unknown variable 'a1'

54 |rEcall iN a111;

...... !..............^ position: 15

Error: reference to unknown variable 'a111'

55 |

56 |reCall ALL;

57 |recAll All;

58 |

59 |recaLl Rest;

60 |recalL REst in;

...... !..............^ position: 15

Error: syntax error

...... !..............^ position: 15

Error: wrong arifmetic expression

61 |

62 |REcall Next 5;

63 |REcAll nExt a;

64 |REcaLL NExt(b + 35 \* c);

65 |ReCall NEXt 678 - 90 \* 0 + a1;

66 |

67 |recAlL RECORD 5;

68 |recall rECOrd a;

69 |recall reCOrD (b + 35 \* c);

70 |recall recoRd 678 - 90 \* 0 + a1;

71 |

72 |recall whilE(d == (25 + 30));

73 |recall whILE((d == 25 + 30));

74 |recall WHIle((d > 25 + 30));

75 |recall WhIlE((d >= 25 + 30));

76 |recall WhIlE((d =< 25 + 30));

...... !................^............ position: 17

Error: syntax error

...... !................^............ position: 17

Error: wrong identifier

77 |recall whILE((d <= 25 + 30));

78 |recall whILE((d => 25 + 30));

...... !................^............ position: 17

Error: syntax error

...... !................^............ position: 17

Error: wrong identifier

79 |recall wHILe((d != 25 + 30));

80 |Recall WHILe((d == 25 + 30));

81 |

82 |REcAll Nooptimize;

83 |RECALl nooPTIMIZE;

84 |

85 |RECAlL All fOr((d != 25 + 30)) wHILe((d != 25 + 30)) iN 678 - 90 \* 0 + a1;

86 |ReCALL REst WhIlE((d >= 25 + 30)) nooPTIMIZE;

87 |recall NEXt 678 - 90 \* 0 + a1 iN 678 - 90 \* 0 + a1;

88 |ReCALL REst WhIlE((d >= 25 + 30)) nooPTIMIZE iN (678!=4);

89 |ReCALL REst WhIlE((d > 25)) nooPTIMIZE iN 678 > 90 \* 0;

90 |ReCALL REst WhIlE((1 and not d)) nooPTIMIZE iN (670 == 90 \* 0);

91 |ReCALL REst WhIlE((d >= 25 + 30)) nooPTIMIZE iN 678 - 90 \* 0 + a1;

92 |ReCALL REst WhIlE(not 1 and not not not not not 0) nooPTIMIZE iN 678 - 90 \* 0 + a1;

93 |ReCALL REst WhIlE(not 1 0 not and 1) nooPTIMIZE iN b2;

...... !........................^............................. position: 25

Error: syntax error

...... !........................^............................. position: 25

Error: wrong number

...... !.....................................................^ position: 54

Error: reference to unknown variable 'b2'

94 |

95 |recall wHILe((d != 500));

96 |recall wHILe((x = (d ==500)));

...... !................^............. position: 17

Error: syntax error

...... !................^............. position: 17

Error: wrong identifier

...... !...........................^.. position: 28

Error: syntax error

97 |recall wHILe((25!= 25 + 30 / (30 and x)));

98 |x = 1;

99 |recall wHILe((x / 0));

...... !..................^... position: 19

Error: division by zero exception line 99:c19 to 99:c19

100 |recall wHILe((x!= x + x / 0));

...... !..........................^... position: 27

Error: division by zero exception line 100:c27 to 100:c27

101 |recall wHILe((d != 25 + 30));

102 |x = x + 10;

103 |ReCALL REst WhIlE(x\*x) nooPTIMIZE iN 100/x-10;

104 |ReCALL REst WhIlE(x\*(-1)) nooPTIMIZE iN (-10\*(-29)+200/(-3));

105 |ReCALL WhIlE((d >= 25 + 30)) iN x;

final content of variables

Name----------------------------- Value----------

1 : var1 : 0

2 : var2 : 0

3 : ggg : 0

4 : var3 : 10

5 : var4 : 2147483647

6 : var5 : 424

7 : var6 : 157

8 : var7 : 4294967161

9 : a : 0

10 : d : 0

11 : bool1 : 0

12 : bool2 : 1

13 : bool3 : 1

14 : bool4 : 0

15 : bool5 : 0

16 : bool6 : 1

17 : bool7 : 0

18 : bool8 : 1

19 : bool9 : 0

20 : bool10 : 0

21 : bool11 : 1

22 : bool12 : 0

23 : bool13 : 1

24 : bool14 : 0

25 : bool15 : 1

26 : bool16 : 1

27 : LongLongLongLong : 500

28 : longVar0 : 205245011

29 : longVar1 : 2147483647

30 : longVar2 : 4287320367

31 : b : 10

32 : c : 20

33 : f : 10

34 : x : 11

35 : a1 : 0

36 : a111 : 0

37 : b2 : 0

==== count of variables - 37 ====

==== count of errors - 30 ====