/\* 01.07.2020 \*/

#include <iostream>

#include <string>

#include <cstdio>

#include <ctype.h>

#include <cstdlib>

#include <vector>

#include <stack>

#include <algorithm>

**using** **namespace** std**;**

enum type\_of\_lex **{**

LEX\_NULL**,** /\* 0\*/

LEX\_AND**,** LEX\_BEGIN**,** LEX\_BOOL**,** LEX\_DO**,** LEX\_ELSE**,** LEX\_END**,** LEX\_IF**,** LEX\_FALSE**,** LEX\_INT**,** /\* 9\*/

LEX\_NOT**,** LEX\_OR**,** LEX\_PROGRAM**,** LEX\_READ**,** LEX\_THEN**,** LEX\_TRUE**,** LEX\_VAR**,** LEX\_WHILE**,** LEX\_WRITE**,** /\*18\*/

LEX\_FIN**,** /\*19\*/

LEX\_SEMICOLON**,** LEX\_COMMA**,** LEX\_COLON**,** LEX\_ASSIGN**,** LEX\_LPAREN**,** LEX\_RPAREN**,** LEX\_EQ**,** LEX\_LSS**,** /\*27\*/

LEX\_GTR**,** LEX\_PLUS**,** LEX\_MINUS**,** LEX\_TIMES**,** LEX\_SLASH**,** LEX\_LEQ**,** LEX\_NEQ**,** LEX\_GEQ**,** /\*35\*/

LEX\_NUM**,** /\*36\*/

LEX\_ID**,** /\*37\*/

POLIZ\_LABEL**,** /\*38\*/

POLIZ\_ADDRESS**,** /\*39\*/

POLIZ\_GO**,** /\*40\*/

POLIZ\_FGO /\*41\*/

**};**

///////////////////////// Класс Lex //////////////////////////

class Lex **{**

type\_of\_lex t\_lex**;**

int v\_lex**;**

public**:**

Lex **(** type\_of\_lex t **=** LEX\_NULL**,** int v **=** 0 **):** t\_lex **(**t**),** v\_lex **(**v**)** **{** **}**

type\_of\_lex get\_type **()** const **{**

**return** t\_lex**;**

**}**

int get\_value **()** const **{**

**return** v\_lex**;**

**}**

friend ostream **&** **operator<<** **(** ostream **&**s**,** Lex l **);**

**};**

///////////////////// Класс Ident ////////////////////////////

class Ident **{**

string name**;**

bool declare**;**

type\_of\_lex type**;**

bool assign**;**

int value**;**

public**:**

Ident**()** **{**

declare **=** **false;**

assign **=** **false;**

**}**

bool **operator==** **(** const string**&** s **)** const **{**

**return** name **==** s**;**

**}**

Ident **(** const string n **)** **{**

name **=** n**;**

declare **=** **false;**

assign **=** **false;**

**}**

string get\_name **()** const **{**

**return** name**;**

**}**

bool get\_declare **()** const **{**

**return** declare**;**

**}**

void put\_declare **()** **{**

declare **=** **true;**

**}**

type\_of\_lex get\_type **()** const **{**

**return** type**;**

**}**

void put\_type **(** type\_of\_lex t **)** **{**

type **=** t**;**

**}**

bool get\_assign **()** const **{**

**return** assign**;**

**}**

void put\_assign **()** **{**

assign **=** **true;**

**}**

int get\_value **()** const **{**

**return** value**;**

**}**

void put\_value **(** int v **)** **{**

value **=** v**;**

**}**

**};**

////////////////////// TID ///////////////////////

vector**<**Ident**>** TID**;**

int put **(** const string **&** buf **)** **{**

vector**<**Ident**>::**iterator k**;**

**if** **(** **(** k **=** find **(** TID**.**begin **(),** TID**.**end **(),** buf **)** **)** **!=** TID**.**end **()** **)**

**return** k **-** TID**.**begin**();**

TID**.**push\_back **(** Ident**(**buf**)** **);**

**return** TID**.**size **()** **-** 1**;**

**}**

/////////////////////////////////////////////////////////////////

class Scanner **{**

FILE **\*** fp**;**

char c**;**

int look **(** const string buf**,** const char **\*\*** list **)** **{**

int i **=** 0**;**

**while** **(** list**[**i**]** **)** **{**

**if** **(** buf **==** list**[**i**]** **)**

**return** i**;**

**++**i**;**

**}**

**return** 0**;**

**}**

void gc **()** **{**

c **=** fgetc **(**fp**);**

**}**

public**:**

static const char **\*** TW **[],** **\*** TD **[];**

Scanner **(** const char **\*** program **)** **{**

fopen\_s**(&**fp**,** program**,** "r"**);**

**}**

Lex get\_lex **();**

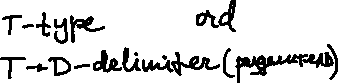
**};**

const char **\***



Scanner**::**TW **[]** **=** **{** ""**,** "and"**,** "begin"**,** "bool"**,** "do"**,** "else"**,** "end"**,** "if"**,** "false"**,** "int"**,** "not"**,** "or"**,** "program"**,**

"read"**,** "then"**,** "true"**,** "var"**,** "while"**,** "write"**,** **NULL** **};**



const char **\***

Scanner**::**TD **[]** **=** **{** "@"**,** ";"**,** ","**,** ":"**,** ":="**,** "("**,** ")"**,** "="**,** "<"**,** ">"**,** "+"**,** "-"**,** "\*"**,** "/"**,** "<="**,** "!="**,** ">="**,** **NULL** **};**

Lex Scanner**::**get\_lex **()** **{**

enum state **{** H**,** IDENT**,** NUMB**,** COM**,** ALE**,** NEQ **};**

int d**,** j**;**

string buf**;**

state CS **=** H**;**

**do** **{**



gc **();**



**switch** **(** CS **)** **{**



**case** H**:**

**if** **(** c**==**' ' **||** c **==** '\n' **||** c**==** '\r' **||** c **==** '\t' **);**

**else** **if** **(** isalpha **(**c**)** **)** **{**

buf**.**push\_back **(**c**);**

CS **=** IDENT**;**

**}**

**else** **if** **(** isdigit **(**c**)** **)** **{**

d **=** c **-** '0'**;**

CS **=** NUMB**;**

**}**

**else** **if** **(** c**==** '{' **)** **{**

CS **=** COM**;**

**}**

**else** **if** **(** c **==** ':' **||** c **==** '<' **||** c **==** '>' **)** **{**

buf**.**push\_back **(**c**);**

CS **=** ALE**;**

**}**

**else** **if** **(**c **==** '@'**)**

**return** Lex **(** LEX\_FIN **);**

**else** **if** **(**c **==** '!'**)** **{**

buf**.**push\_back **(**c**);**

CS **=** NEQ**;**

**}**

**else** **{**

buf**.**push\_back **(**c**);**

**if** **(** **(** j **=** look **(** buf**,** TD**)** **)** **){**

**return** Lex **(** **(**type\_of\_lex**)(** j **+** **(**int**)** LEX\_FIN **),** j **);**

**}**

**else**

**throw** c**;**

**}**

**break;**

**case** IDENT**:**

**if** **(** isalpha **(**c**)** **||** isdigit **(**c**)** **)** **{**

buf**.**push\_back **(**c**);**

**}**

**else** **{**

ungetc **(** c**,** fp **);**

**if** **(** **(**j **=** look **(** buf**,** TW**)** **)** **)** **{**

**return** Lex **(** **(**type\_of\_lex**)** j**,** j **);**

**}**

**else** **{**

j **=** put **(** buf **);**

**return** Lex **(** LEX\_ID**,** j **);**

**}**

**}**

**break;**

**case** NUMB**:**

**if** **(** isdigit **(**c**)** **)** **{**

d **=** d **\*** 10 **+** **(** c **-** '0' **);**

**}**

**else** **{**

ungetc **(** c**,** fp **);**

**return** Lex **(** LEX\_NUM**,** d **);**

**}**

**break;**

**case** COM**:**

**if** **(** c **==** '}' **)** **{**

CS **=** H**;**

**}**

**else** **if** **(** c **==** '@' **||** c **==** '{' **)**

**throw** c**;**

**break;**

**case** ALE**:**

**if** **(** c **==** '=' **)** **{**

buf**.**push\_back **(** c **);**

j **=** look **(** buf**,** TD **);**

**return** Lex **(** **(**type\_of\_lex**)** **(** j **+** **(**int**)** LEX\_FIN **),** j **);**

**}**

**else** **{**

ungetc **(** c**,** fp **);**

j **=** look **(** buf**,** TD **);**

**return** Lex **(** **(**type\_of\_lex**)** **(** j **+** **(**int**)** LEX\_FIN **),** j **);**

**}**

**break;**

**case** NEQ**:**

**if** **(** c **==** '=' **)** **{**

buf**.**push\_back**(**c**);**

j **=** look **(** buf**,** TD **);**

**return** Lex **(** LEX\_NEQ**,** j **);**

**}**

**else**

**throw** '!'**;**

**break;**

**}** //end switch

**}** **while** **(true);**

**}**

ostream **&** **operator<<** **(** ostream **&**s**,** Lex l **)** **{**

string t**;**

**if** **(** l**.**t\_lex **<=** LEX\_WRITE **)**

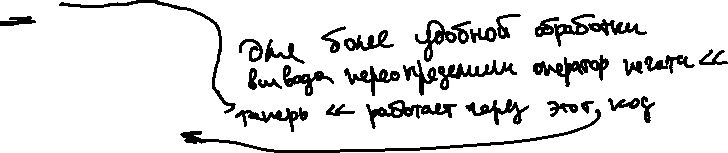
t **=** Scanner**::**TW**[**l**.**t\_lex**];**

**else** **if** **(** l**.**t\_lex **>=** LEX\_FIN **&&** l**.**t\_lex **<=** LEX\_GEQ **)**

t **=** Scanner**::**TD**[** l**.**t\_lex **-** LEX\_FIN **];**

**else** **if** **(** l**.**t\_lex **==** LEX\_NUM **)**

t **=** "NUMB"**;**



**else** **if** **(** l**.**t\_lex **==** LEX\_ID **)**

t **=** TID**[**l**.**v\_lex**].**get\_name **();**

**else** **if** **(** l**.**t\_lex **==** POLIZ\_LABEL **)**

t **=** "Label"**;**

**else** **if** **(** l**.**t\_lex **==** POLIZ\_ADDRESS **)**

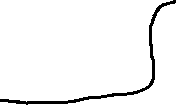
t **=** "Addr"**;**

**else** **if** **(** l**.**t\_lex **==** POLIZ\_GO **)**

t **=** "!"**;**

**else** **if** **(** l**.**t\_lex **==** POLIZ\_FGO **)**

t **=** "!F"**;**



**else**

**throw** l**;**

s **<<** '(' **<<** t **<<** ',' **<<** l**.**v\_lex **<<** ");" **<<** endl**;**

**return** s**;**

**}**

////////////////////////// Класс Parser /////////////////////////////////

template **<**class T**,** class T\_EL**>**

void from\_st **(** T **&** st**,** T\_EL **&** i **)** **{**

i **=** st**.**top**();** st**.**pop**();**

**}**

class Parser **{**

Lex curr\_lex**;**

type\_of\_lex c\_type**;**

int c\_val**;**

Scanner scan**;**

stack **<** int **>** st\_int**;**

stack **<** type\_of\_lex **>** st\_lex**;**

void P**();**

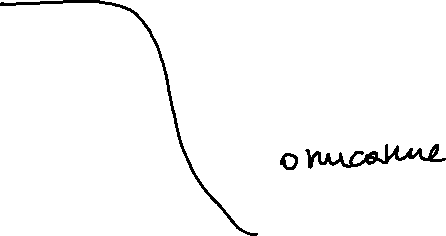
void D1**();**

void D**();**

void B**();**

void S**();**

void E**();**



void E1**();**

void T**();**

void F**();**



void dec **(** type\_of\_lex type**);**



void check\_id **();**

void check\_op **();**

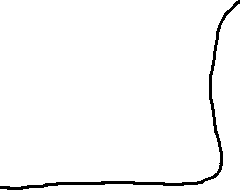
void check\_not **();**

void eq\_type **();**



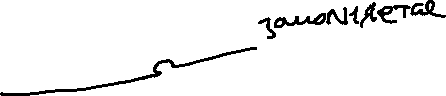
void eq\_bool **();**

void check\_id\_in\_read **();**



void gl **()** **{**

curr\_lex **=** scan**.**get\_lex **();**



c\_type **=** curr\_lex**.**get\_type **();**

c\_val **=** curr\_lex**.**get\_value **();**

**}**

public**:**

vector **<**Lex**>** poliz**;**

Parser **(** const char **\***program **)** **:** scan **(**program**)** **{** **}**

void analyze**();**

**};**



void Parser**::**analyze **()** **{**

gl **();**



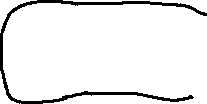
P **();**



**if** **(**c\_type **!=** LEX\_FIN**)**



**throw** curr\_lex**;**



//for\_each( poliz.begin(), poliz.end(), [](Lex l){ cout << l; });

**for** **(** Lex l **:** poliz **)**

cout **<<** l**;**



cout **<<** endl **<<** "Yes!!!" **<<** endl**;**

**}**



void Parser**::**P **()** **{**



**if** **(** c\_type **==** LEX\_PROGRAM **)** **{**



gl **();**

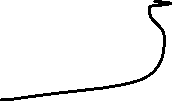
**}**

**else**

**throw** curr\_lex**;**

D1 **();**

**if** **(** c\_type **==** LEX\_SEMICOLON **)**



gl **();**



**else**



**throw** curr\_lex**;**



B **();**

**}**



void Parser**::**D1 **()** **{**

**if** **(** c\_type **==** LEX\_VAR **)** **{**



gl **();**



D **();**



**while** **(** c\_type **==** LEX\_COMMA **)** **{**

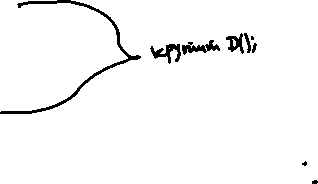
gl **();**

D **();**

**}**

**}**

**else**



**throw** curr\_lex**;**

**}**



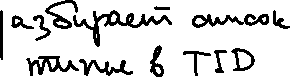
void Parser**::**D **()** **{**



**if** **(** c\_type **!=** LEX\_ID **)**



**throw** curr\_lex**;**



**else** **{**

st\_int**.**push **(** c\_val **);**

gl **();**

**while** **(** c\_type **==** LEX\_COMMA **)** **{**

gl **();**

**if** **(** c\_type **!=** LEX\_ID **)**

**throw** curr\_lex**;**

**else** **{**



st\_int**.**push **(** c\_val **);**

gl **();**

**}**

**}**

**if** **(** c\_type **!=** LEX\_COLON **)**

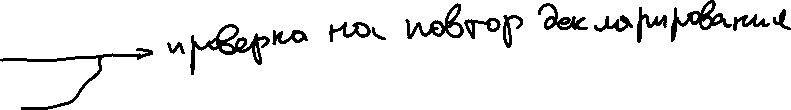
**throw** curr\_lex**;**

**else** **{**

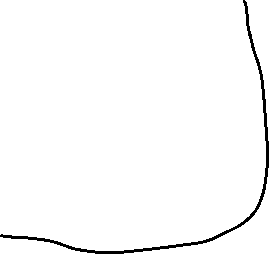
gl **();**



**if** **(** c\_type **==** LEX\_INT **)** **{**



dec **(** LEX\_INT **);**



gl **();**

**}**

**else**

**if** **(** c\_type **==** LEX\_BOOL **)** **{**

dec **(** LEX\_BOOL **);**

gl **();**

**}**

**else**

**throw** curr\_lex**;**

**}**

**}**

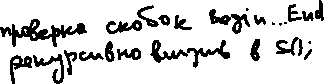
**}**

void Parser**::**B **()** **{**

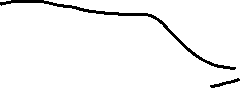
**if** **(** c\_type **==** LEX\_BEGIN **)** **{**

gl **();**

S **();**



**while** **(** c\_type **==** LEX\_SEMICOLON **)** **{**



gl **();**

S **();**

**}**

**if** **(** c\_type **==** LEX\_END **)** **{**

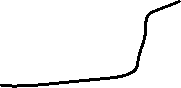
gl **();**

**}**

**else** **{**

**throw** curr\_lex**;**

**}**



**}**

**else**

**throw** curr\_lex**;**

**}**

void Parser**::**S **()** **{**

int pl0**,** pl1**,** pl2**,** pl3**;**

**if** **(** c\_type **==** LEX\_IF **)** **{**

gl **();**

E **();**

eq\_bool **();**



pl2 **=** poliz**.**size**();**



poliz**.**push\_back **(** Lex**()** **);**

poliz**.**push\_back **(** Lex**(**POLIZ\_FGO**)** **);**

**if** **(** c\_type **==** LEX\_THEN **)** **{**

gl **();**

S **();**

pl3 **=** poliz**.**size **();**

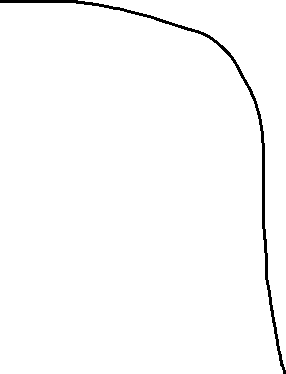
poliz**.**push\_back **(** Lex **()** **);**

poliz**.**push\_back **(** Lex **(** POLIZ\_GO **)** **);**

poliz**[**pl2**]** **=** Lex **(** POLIZ\_LABEL**,** poliz**.**size**()** **);**

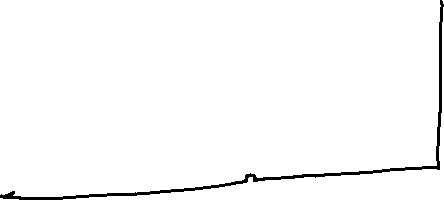
**if** **(** c\_type **==** LEX\_ELSE **)** **{**

gl **();**



S **();**

poliz**[**pl3**]** **=** Lex **(** POLIZ\_LABEL**,** poliz**.**size**()** **);**



**}**

**else**

**throw** curr\_lex**;**

**}**

**else**

**throw** curr\_lex**;**

**}**//end if

**else** **if** **(** c\_type **==** LEX\_WHILE **)** **{**

pl0 **=** poliz**.**size **();**

gl **();**

E **();**

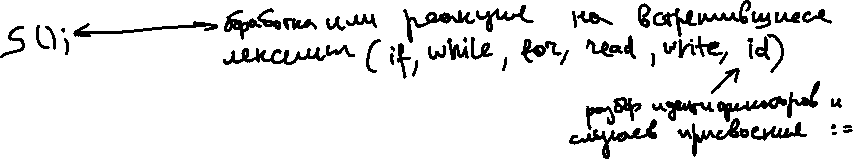
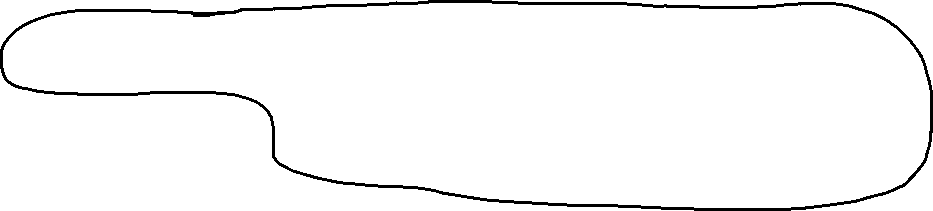
eq\_bool **();**

pl1 **=** poliz**.**size **();**

poliz**.**push\_back **(** Lex **()** **);**



poliz**.**push\_back **(** Lex **(**POLIZ\_FGO**)** **);**



**if** **(** c\_type **==** LEX\_DO **)** **{**

gl**();**

S**();**

poliz**.**push\_back **(** Lex **(** POLIZ\_LABEL**,** pl0 **)** **);**

poliz**.**push\_back **(** Lex **(** POLIZ\_GO**)** **);**

poliz**[**pl1**]** **=** Lex **(** POLIZ\_LABEL**,** poliz**.**size**()** **);**

**}**

**else**

**throw** curr\_lex**;**

**}**//end while



**else** **if** **(** c\_type **==** LEX\_READ **)** **{**

gl **();**

**if** **(** c\_type **==** LEX\_LPAREN **)** **{**

gl **();**

**if** **(** c\_type **==** LEX\_ID **)** **{**

check\_id\_in\_read **();**

poliz**.**push\_back **(** Lex**(** POLIZ\_ADDRESS**,** c\_val**)** **);**



gl**();**

**}**

**else**

**throw** curr\_lex**;**

**if** **(** c\_type **==** LEX\_RPAREN **)** **{**

gl **();**

poliz**.**push\_back **(** Lex **(**LEX\_READ**)** **);**

**}**

**else**

**throw** curr\_lex**;**

**}**

**else**

**throw** curr\_lex**;**

**}**//end read

**else** **if** **(** c\_type **==** LEX\_WRITE **)** **{**

gl **();**

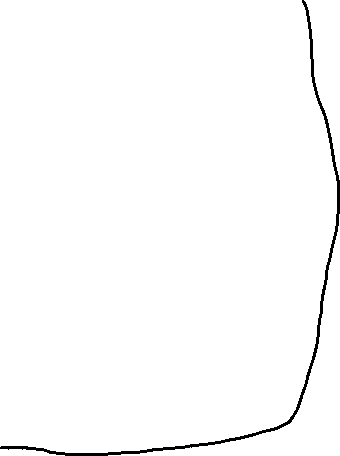
**if** **(** c\_type **==** LEX\_LPAREN **)** **{**

gl **();**

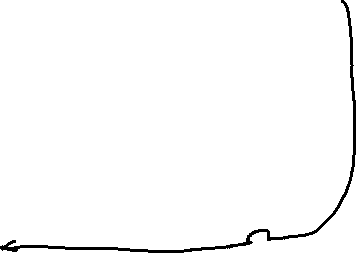
E **();**

**if** **(** c\_type **==** LEX\_RPAREN **)** **{**

gl **();**



poliz**.**push\_back **(** Lex **(** LEX\_WRITE **)** **);**



**}**

**else**

**throw** curr\_lex**;**

**}**

**else**

**throw** curr\_lex**;**

**}**//end write

**else** **if** **(** c\_type **==** LEX\_ID **)** **{**

check\_id **();**

poliz**.**push\_back **(**Lex **(** POLIZ\_ADDRESS**,** c\_val **)** **);**

gl**();**

**if** **(** c\_type **==** LEX\_ASSIGN **)** **{**

gl **();**

E **();**

eq\_type **();**

poliz**.**push\_back **(** Lex **(** LEX\_ASSIGN **)** **);**

**}**

**else**

**throw** curr\_lex**;**

**}**//assign-end

**else**

B**();**

**}**

void Parser**::**E **()** **{**



E1 **();**



**if** **(** c\_type **==** LEX\_EQ **||** c\_type **==** LEX\_LSS **||** c\_type **==** LEX\_GTR **||**

c\_type **==** LEX\_LEQ **||** c\_type **==** LEX\_GEQ **||** c\_type **==** LEX\_NEQ **)** **{**

st\_lex**.**push **(** c\_type **);**

gl **();**

E1 **();**



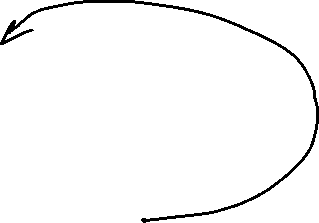
check\_op **();**



**}**



**}**



void Parser**::**E1 **()** **{**

T **();**



**while** **(** c\_type **==** LEX\_PLUS **||** c\_type **==** LEX\_MINUS **||** c\_type **==** LEX\_OR**)** **{**



st\_lex**.**push **(** c\_type **);**

gl **();**



T **();**



check\_op **();**

**}**

**}**



void Parser**::**T **()** **{**

F **();**

**while** **(** c\_type **==** LEX\_TIMES **||** c\_type **==** LEX\_SLASH **||** c\_type **==** LEX\_AND**)** **{**

st\_lex**.**push **(** c\_type **);**

gl **();**

F **();**

check\_op **();**

**}**

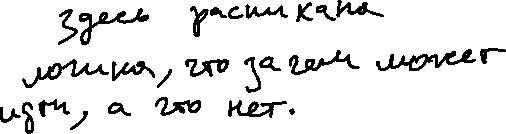
**}**

void Parser**::**F **()** **{**

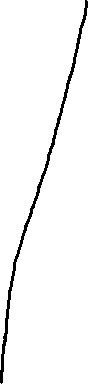
**if** **(** c\_type **==** LEX\_ID **)** **{**

check\_id **();**

poliz**.**push\_back **(** Lex **(** LEX\_ID**,** c\_val **)** **);**



gl **();**



**}**

**else** **if** **(** c\_type **==** LEX\_NUM **)** **{**

st\_lex**.**push **(** LEX\_INT **);**

poliz**.**push\_back **(** curr\_lex **);**

gl **();**

**}**

**else** **if** **(** c\_type **==** LEX\_TRUE **)** **{**

st\_lex**.**push **(** LEX\_BOOL **);**

poliz**.**push\_back **(** Lex **(**LEX\_TRUE**,** 1**)** **);**

gl **();**

**}**

**else** **if** **(** c\_type **==** LEX\_FALSE**)** **{**

st\_lex**.**push **(** LEX\_BOOL **);**

poliz**.**push\_back **(** Lex **(**LEX\_FALSE**,** 0**)** **);**

gl **();**

**}**

**else** **if** **(** c\_type **==** LEX\_NOT **)** **{**

gl **();**

F **();**

check\_not **();**

**}**

**else** **if** **(** c\_type **==** LEX\_LPAREN **)** **{**

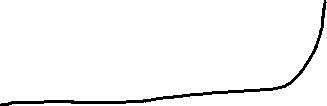
gl **();**

E **();**

**if** **(** c\_type **==** LEX\_RPAREN**)**

gl **();**

**else**



**throw** curr\_lex**;**

**}**

**else**

**throw** curr\_lex**;**

**}**

////////////////////////////////////////////////////////////////

void Parser**::**dec **(** type\_of\_lex type **)** **{**



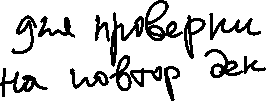
int i**;**

**while** **(** **!**st\_int**.**empty **()** **)** **{**

from\_st **(** st\_int**,** i **);**

**if** **(** TID**[**i**].**get\_declare **()** **)**

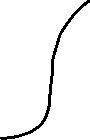
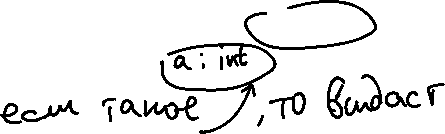
**throw** "twice"**;**



**else** **{**



TID**[**i**].**put\_declare **();**



TID**[**i**].**put\_type **(** type **);**

**}**

**}**

**}**

void Parser**::**check\_id **()** **{**

**if** **(** TID**[**c\_val**].**get\_declare**()** **)**

st\_lex**.**push **(** TID**[**c\_val**].**get\_type **()** **);**

**else**

**throw** "not declared"**;**

**}**

void Parser**::**check\_op **()** **{**

type\_of\_lex t1**,** t2**,** op**,** t **=** LEX\_INT**,** r **=** LEX\_BOOL**;**

from\_st **(** st\_lex**,** t2 **);**

from\_st **(** st\_lex**,** op **);**

from\_st **(** st\_lex**,** t1 **);**

**if** **(** op **==** LEX\_PLUS **||** op **==** LEX\_MINUS **||** op **==** LEX\_TIMES **||** op **==** LEX\_SLASH **)**

r **=** LEX\_INT**;**

**if** **(** op **==** LEX\_OR **||** op **==** LEX\_AND **)**

t **=** LEX\_BOOL**;**

**if** **(** t1 **==** t2 **&&** t1 **==** t **)**

st\_lex**.**push **(**r**);**

**else**

**throw** "wrong types are in operation"**;**

poliz**.**push\_back **(**Lex **(**op**)** **);**

**}**

void Parser**::**check\_not **()** **{**

**if** **(**st\_lex**.**top**()** **!=** LEX\_BOOL**)**

**throw** "wrong type is in not"**;**

**else**

poliz**.**push\_back **(** Lex **(**LEX\_NOT**)** **);**

**}**

void Parser**::**eq\_type **()** **{**

type\_of\_lex t**;**

from\_st **(** st\_lex**,** t **);**

**if** **(** t **!=** st\_lex**.**top **()** **)**

**throw** "wrong types are in :="**;**

st\_lex**.**pop**();**

**}**

void Parser**::**eq\_bool **()** **{**

**if** **(** st\_lex**.**top **()** **!=** LEX\_BOOL **)**

**throw** "expression is not boolean"**;**

st\_lex**.**pop **();**

**}**

void Parser**::**check\_id\_in\_read **()** **{**

**if** **(** **!**TID **[**c\_val**].**get\_declare**()** **)**

**throw** "not declared"**;**

**}**

////////////////////////////////////////////////////////////////

class Executer **{**

public**:**

void execute **(** vector**<**Lex**>** **&** poliz **);**

**};**

void Executer**::**execute **(** vector**<**Lex**>** **&** poliz **)** **{**

Lex pc\_el**;**

stack **<** int **>** args**;**

int i**,** j**,** index **=** 0**,** size **=** poliz**.**size**();**

**while** **(** index **<** size **)** **{**

pc\_el **=** poliz **[** index **];**

**switch** **(** pc\_el**.**get\_type **()** **)** **{**

**case** LEX\_TRUE**:** **case** LEX\_FALSE**:** **case** LEX\_NUM**:** **case** POLIZ\_ADDRESS**:** **case** POLIZ\_LABEL**:**

args**.**push **(** pc\_el**.**get\_value **()** **);**

**break;**

**case** LEX\_ID**:**

i **=** pc\_el**.**get\_value **();**

**if** **(** TID**[**i**].**get\_assign **()** **)** **{**

args**.**push **(** TID**[**i**].**get\_value **()** **);**

**break;**

**}**

**else**

**throw** "POLIZ: indefinite identifier"**;**

**case** LEX\_NOT**:**

from\_st **(** args**,** i **);**

args**.**push**(** **!**i **);**

**break;**

**case** LEX\_OR**:**

from\_st **(** args**,** i **);**

from\_st **(** args**,** j **);**

args**.**push **(** j **||** i **);**

**break;**

**case** LEX\_AND**:**

from\_st **(** args**,** i **);**

from\_st **(** args**,** j **);**

args**.**push **(** j **&&** i **);**

**break;**

**case** POLIZ\_GO**:**

from\_st **(** args**,** i **);**

index **=** i **-** 1**;**

**break;**

**case** POLIZ\_FGO**:**

from\_st **(** args**,** i **);**

from\_st **(** args**,** j **);**

**if** **(** **!**j **)** index **=** i **-** 1**;**

**break;**

**case** LEX\_WRITE**:**

from\_st **(** args**,** j **);**

cout **<<** j **<<** endl**;**

**break;**

**case** LEX\_READ**:**

int k**;**

from\_st **(** args**,** i **);**

**if** **(** TID**[**i**].**get\_type **()** **==** LEX\_INT **)** **{**

cout **<<** "Input int value for" **<<** TID**[**i**].**get\_name **()** **<<** endl**;**

cin **>>** k**;**

**}**

**else** **{**

string j**;**

**while** **(**1**)** **{**

cout **<<** "Input boolean value (true or false) for" **<<** TID**[**i**].**get\_name**()** **<<** endl**;**

cin **>>** j**;**

**if** **(** j **!=** "true" **&&** j **!=** "false" **)** **{**

cout **<<** "Error in input:true/false" **<<** endl**;**

**continue;**

**}**

k **=** **(** j **==** "true" **)** **?** 1 **:** 0**;**

**break;**

**}**

**}**

TID**[**i**].**put\_value **(**k**);**

TID**[**i**].**put\_assign **();**

**break;**

**case** LEX\_PLUS**:**

from\_st **(** args**,** i **);**

from\_st **(** args**,** j **);**

args**.**push **(** i **+** j **);**

**break;**

**case** LEX\_TIMES**:**

from\_st **(** args**,** i **);**

from\_st **(** args**,** j **);**

args**.**push **(** i **\*** j **);**

**break;**

**case** LEX\_MINUS**:**

from\_st **(** args**,** i **);**

from\_st **(** args**,** j **);**

args**.**push **(** j **-** i **);**

**break;**

**case** LEX\_SLASH**:**

from\_st **(** args**,** i **);**

from\_st **(** args**,** j **);**

**if** **(!**i**)** **{**

args**.**push **(** j **/** i **);**

**break;**

**}**

**else**

**throw** "POLIZ:divide by zero"**;**

**case** LEX\_EQ**:**

from\_st **(** args**,** i **);**

from\_st **(** args**,** j **);**

args**.**push **(** i **==** j **);**

**break;**

**case** LEX\_LSS**:**

from\_st **(** args**,** i **);**

from\_st **(** args**,** j **);**

args**.**push **(** j **<** i **);**

**break;**

**case** LEX\_GTR**:**

from\_st **(** args**,** i **);**

from\_st **(** args**,** j **);**

args**.**push **(** j **>** i **);**

**break;**

**case** LEX\_LEQ**:**

from\_st **(** args**,** i **);**

from\_st **(** args**,** j **);**

args**.**push **(** j **<=** i **);**

**break;**

**case** LEX\_GEQ**:**

from\_st **(** args**,** i **);**

from\_st **(** args**,** j **);**

args**.**push **(** j **>=** i **);**

**break;**

**case** LEX\_NEQ**:**

from\_st **(** args**,** i **);**

from\_st **(** args**,** j **);**

args**.**push **(** j **!=** i **);**

**break;**

**case** LEX\_ASSIGN**:**

from\_st **(** args**,** i **);**

from\_st **(** args**,** j **);**

TID**[**j**].**put\_value **(**i**);**

TID**[**j**].**put\_assign **();**

**break;**

**default:**

**throw** "POLIZ: unexpected elem"**;**

**}**//end of switch

**++**index**;**

**};**//end of while

cout **<<** "Finish of executing!!!" **<<** endl**;**

**}**

class Interpretator **{**

Parser pars**;**

Executer E**;**

public**:**

Interpretator **(** const char**\*** program **):** pars **(**program**)** **{}**

void interpretation **();**

**};**

void Interpretator**::**interpretation **()** **{**

pars**.**analyze **();**

E**.**execute **(** pars**.**poliz **);**

**}**

int main **()** **{**

**try** **{**

Interpretator I **(** "prog\_works.txt" **);**

I**.**interpretation **();**

**return** 0**;**

**}**

**catch** **(** char c **)** **{**

cout **<<** "unexpected symbol " **<<** c **<<** endl**;**

**return** 1**;**

**}**

**catch** **(** Lex l **)** **{**

cout **<<** "unexpected lexeme" **<<** l **<<** endl**;**

**return** 1**;**

**}**

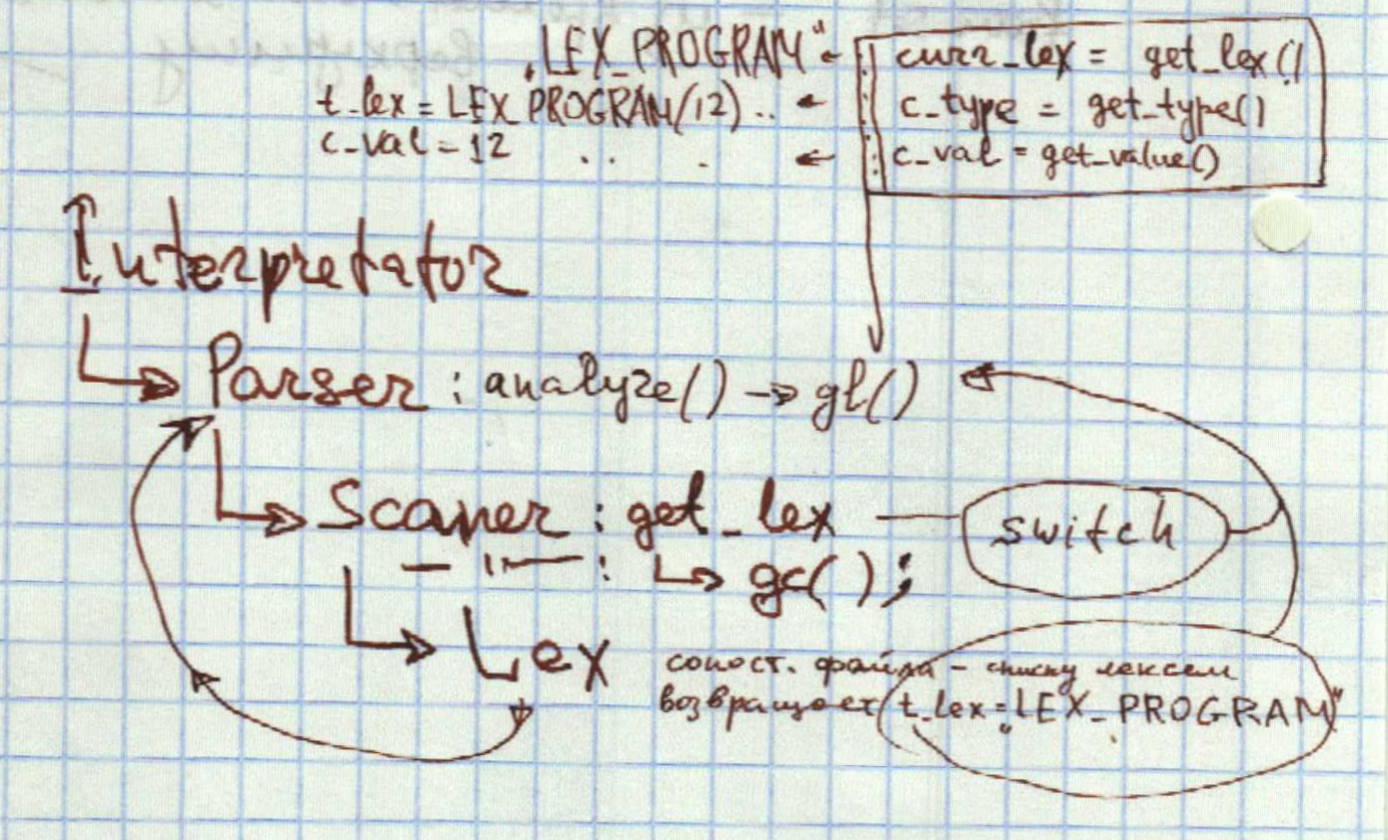
**catch** **(** const char **\***source **)** **{**

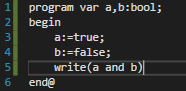
cout **<<** source **<<** endl**;**

**return** 1**;**

**}**

**}**



`