a∗b ⊥

Оператор условного перехода по лжи

$p_n \; B' \; !F$

!F ← оператор условного перехода по лжи

p_n ← номер перехода

B' ← полиз некоторого булевого выражения

B' - это полиз выражения B

a = count - size

b = a . count + d

Token

String name
String value

T1 (VAR, "a")
T2 (ASSIGN_OP, "=")
T3 (VAR, "count")
T4 (OP, "-")
T5 ...

```
void VAR() {
    match();
if(currentToken.getName().equals("VAR")){
    ... ....
}else {
throw new ParseException();
}
```

G(VT VN, P, S)

Нетерминалы
- assign → VAR ASSIGN_OP valueOPvalue
- value → VAR | DIGIT

Терминалы
- VAR → [a..z]+
- DIGIT → [0..9]+
- ASSIGN_OP → "="
- OP → "+" | "-" | "*" | "/"

Parser

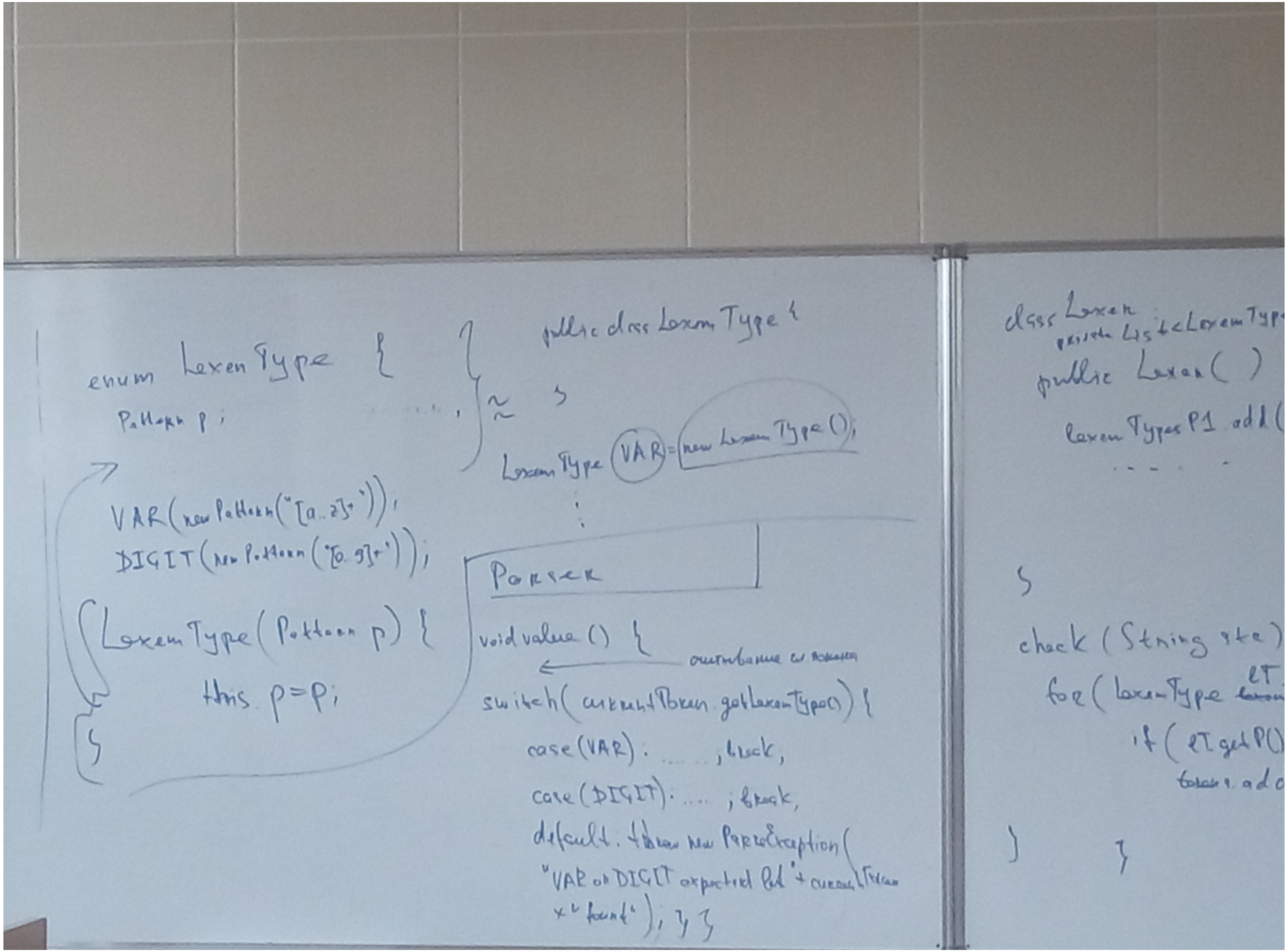
currentToken = ...

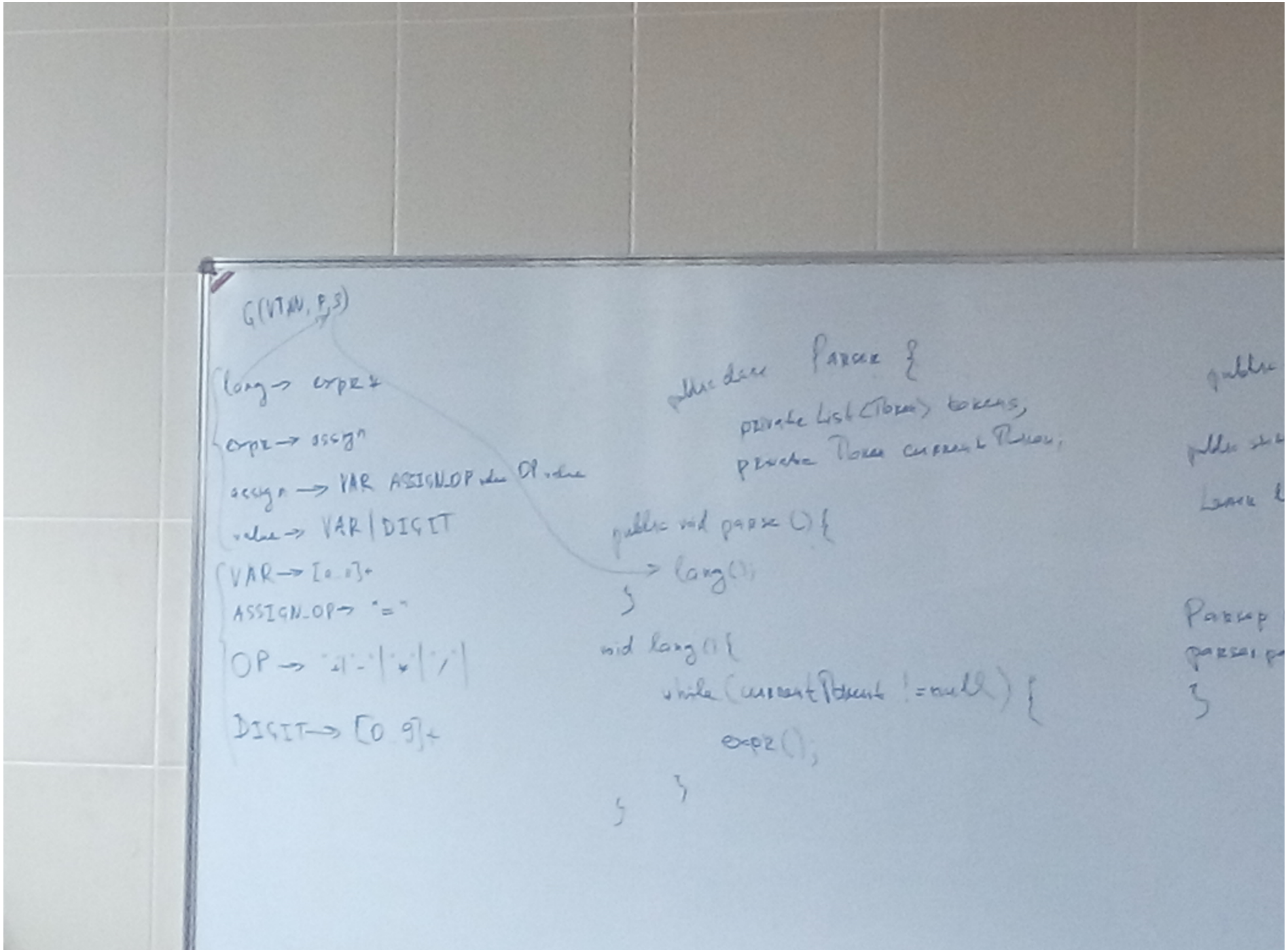
```
void assign() {
    VAR(); ASSIGN_OP(); value(); OP(); value();
}
```

```
class Lexer
    private List<LexemType> lexemTypesP1 = new ArrayList<LexemType>();
    private List<Token> tokens;
    public Lexer() {
        lexemTypesP1.add(LexemType.VAR);
        ....

    }

    check(String str) {
        for(LexemType lt : lexemTypesP1) {
            if(lt.getP().matcher(str).find()) {
                tokens.add(new Token(lt, str));
            }
        }
    }
```

```
class Token {
    private LexemType lexemType;
    private String value;
    public Token(LexemType lexemType, String value) {
        this.lexemType = lexemType;
        this.value = value;
    }
}
```

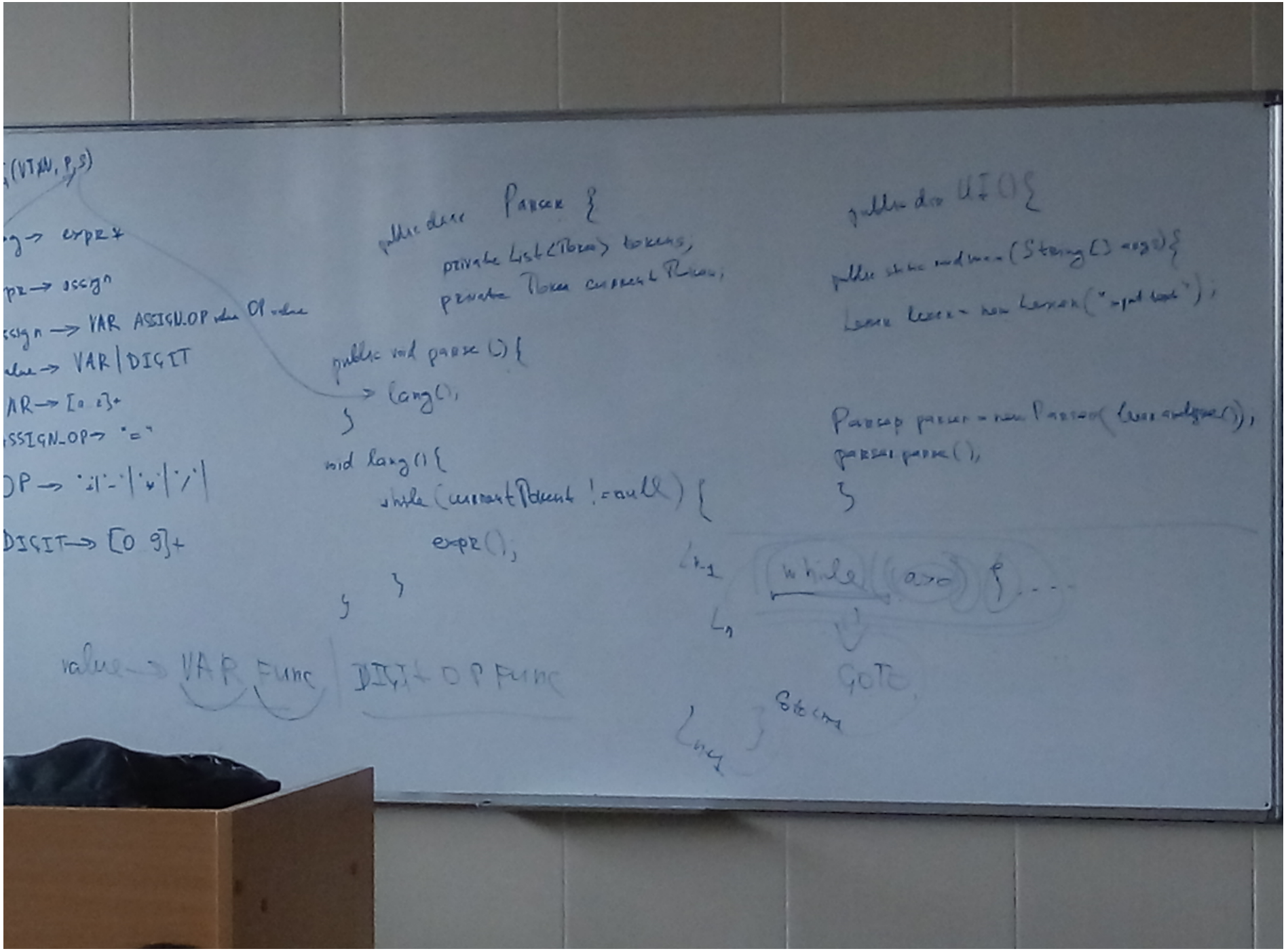
value → VAR | DIGIT


enum LexemType {
Pattern p;
public class LexemType {
LexemType (VAR) = new LexemType();
VAR(new Pattern("[a..z]+"));
DIGIT(new Pattern("[0..9]+"));
LexemType(Pattern p) {
this.p=p;
Parser
void value () {
switch(currentToken.getLexemType()) {
case(VAR): ....., break,
case(DIGIT): ..... ; break,
default. throw new ParseException(
"VAR or DIGIT expected but" + currentToken
+ "found");
class Lexer
public Lexer()
LexemType P1 add(
check(String ste)
for(LexemType
if(
token.ado


G(VTAN, P, S)
lang → expr+
expr → assgn
assgn → VAR ASSIGN_OP value OP value
value → VAR | DIGIT
VAR → [a..z]+
ASSIGN_OP → "="
OP → "+" | "-" | "*" | "/"
DIGIT → [0..9]+
public class Parse {
private List<Token> tokens;
private Token currentToken;
public void parse() {
lang();
}
void lang() {
while (currentToken != null) {
expr();
}
}

```
void expr() { assign(); }
void assign() {
    VAR();
    ASSIGN_OP();
    value();
    OP();
    value();
}
void VAR() {
    (match())
    if(currentToken.getLexemType().equals(LexemType.VAR)) {
        //ok
    } else { throw new Exception("VAR expected but " + currentToken +
        " found");
```

```
void value() {
    switch case
```


(VTN, P, S)
g → expr+
pr → assgn
ssgn → VAR ASSIGN_OP value OP value
lue → VAR | DIGIT
AR → [a z]+
SSIGN_OP → "="
OP → "+" | "-" | "*" | "/"
DIGIT → [0 9]+
public class Parser {
private List<Token> tokens;
private Token currentToken;
public void parse() {
lang();
}
void lang() {
while (currentToken != null) {
expr();
}
}
value → VAR Func | DIGIT OP Func
public class UI {
public static void main(String[] args) {
Lexer lexer = new Lexer("input text");
Parser parser = new Parser(lexer.analyze());
parser.parse();
}
while ((a>0)) {...
GOTO
Stack

void value() {
switch case

0,2,0,3 +

p!

B P !F → while_expr () {
заменить while_head

currentToken следующий токен из tokens

Type() equals(LexemType VAR) {

ption("VAR expected but " + currentToken +

Lexer → Parser
1) синтаксический анализ [illegible]
2) подготовить новый список токенов
[illegible] перед преобразованием в ПОЛИЗ
3) на основании нового списка токенов
получить ПОЛИЗ
4) вычислить ПОЛИЗ

```
void value() {
    switch case
```

```
while_expr() {
    заменить while_head
```

Lexer → Parser

1) синтаксический анализ [illegible]
2) подготовить новый список лексем [illegible] через преобразования в ПОЛИЗ
3) на основании нового списка лексем получить ПОЛИЗ
4) вычислить ПОЛИЗ