

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КІЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

**Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії**

**Лабораторна робота № 6
ГРАМАТИКА ЗАСОБАМИ ANTLR4**

(з дисципліни «Побудова компіляторів»)

Звіт студента I курсу, групи ІП-51мн
спеціальності F2 Інженерія програмного забезпечення

Панченко С. В.

(Прізвище, ім'я, по батькові)

_____ (Підпис)

Перевірив: доцент, к.т.н. Стативка Ю.І.

(Посада, науковий ступінь, прізвище та ініціали)

_____ (Підпис)

Київ – 2025

Зміст

1 Мета.....	3
2 Виконання.....	4
3 Висновок.....	36

1 META

Реалізувати лексичний та синтаксичний аналіз засобами ANTLR4.

2 ВИКОНАННЯ

Мова Lab є спрощеним варіантом Lisp, у якому вся структура програми визначається за допомогою S-виразів, записаних у круглих дужках. Програма складається з одного або кількох визначень функцій. Кожна функція описується списком, який починається ключовим словом fn, далі йде ім'я функції, тип поверненого значення, список параметрів, список локальних змінних і тіло функції, що містить послідовність операторів. Таким чином, базова структура програми виглядає як (fn <ім'я-функції> <тип> (<параметри>) (<змінні>) (<оператори>)).

У мові визначено три базові типи даних: int, float і bool. Цілі числа записуються як послідовність цифр, що може починатися зі знаку «+» або «-». Дійсні числа містять десяткову крапку. Логічні значення представлені словами true і false. Для позначення початку і кінця списків використовуються круглі дужки, які є основною синтаксичною конструкцією мови.

Ідентифікатори в MiniLisp можуть складатися з латинських літер, цифр та спеціальних символів !\$%&*:;<=>?^_~. Вони повинні починатися з літери або дозволеного символу, але не з цифри. окремими допустимими ідентифікаторами є також символи «+» і «-», які використовуються як імена для вбудованих функцій. Пропуски, табуляції та переходи рядків у мові ігноруються, окрім випадків, коли вони відділяють токени.

Синтаксис мови ґрунтуються на рекурсивній структурі списків. Кожен вираз або оператор є списком, першим елементом якого виступає ім'я операції або функції, а далі йдуть аргументи. Основними операторами є set, if, while і return. Оператор set виконує присвоєння значення змінній у формі (set <змінна> <вираз>). Конструкція (if <умова> (<гілка-істина>) (<гілка-хиба>)) реалізує розгалуження. Цикл записується як (while <умова> (<тіло>)), а оператор (return <вираз>) завершує виконання функції, повертаючи результат.

Вирази у мові можуть бути атомами (тобто змінними, константами або логічними значеннями) або викликами функцій. Виклик функції має вигляд (ім'я-функції аргумент1 аргумент2 ...). Аргументи при цьому можуть бути лише атомами, що відповідає обмеженню у побудованому синтаксичному аналізаторі.

Граматика, записана у форматі ANTLR4, описує як лексичні, так і синтаксичні правила мови. Вона визначає структуру програми, синтаксис функцій, виразів і операторів, а також описує токени для чисел, ідентифікаторів, типів, булевих значень і службових слів.

Повна граматика мови засобами ANTLR4.

```
// antlr4-parse lab_1/lab.g4 program -tree lab_1/example.txt
grammar lab;

program
: toplevel* EOF
;

toplevel
: LPAREN FN name=IDENT retType=type funcArgs varDecls stmtList
RPAREN
;

type
: INT_T
| FLOAT_T
| BOOL_T
;

funcArgs
: LPAREN (argPair+)? RPAREN
;

varDecls
: funcArgs
;

argPair
: LPAREN name=IDENT type RPAREN
;

stmtList
: LPAREN statement* RPAREN
```

```

;

statement
: LPAREN SET dest=IDENT expr RPAREN
| LPAREN IF cond=expr thenBranch=stmtList elseBranch=stmtList
RPAREN
| LPAREN WHILE cond=expr body=stmtList RPAREN
| LPAREN RETURN expr RPAREN
;

expr
: atom
| funcCall
;

atom
: IDENT
| INT
| FLOAT
| TRUE_T
| FALSE_T
;

funcCall
: LPAREN callee=IDENT (atom)* RPAREN
;

FN      : 'fn' ;
SET     : 'set' ;
IF      : 'if' ;
WHILE   : 'while' ;
RETURN  : 'return' ;
INT_T   : 'int' ;
FLOAT_T : 'float' ;
BOOL_T  : 'bool' ;
TRUE_T  : 'true' ;
FALSE_T : 'false' ;
LPAREN  : '(' ;
RPAREN  : ')' ;
FLOAT   : [+\\-]? DIGITS '.' DIGITS ;
INT     : [+\\-]? DIGITS ;
fragment DIGITS : [0-9]+ ;
IDENT   :
: [a-zA-Z!$%&*&/:<=>?^_~] [a-zA-Z!$%&*&/:<=>?^_~0-9]*
| [+\\-]
;
WS      : [ \t\r\n]+ -> skip ;

```

Проаналізуємо приклад мови та покажемо синтаксичне дерево мови.

```
(fn factorial int ((num int)) ((prod int) (i int) (acc int)) (
  (set prod 1)
  (set i 0)
  (set acc 1)
  (while (< i num) (
    (set prod (* prod acc))
    (set acc (+ acc 1)))
  ))
  (return prod)
))
(fn factorial float ((num float)) ((prod float) (i float) (acc float)) (
  (set prod 1.0)
  (set i 0.0)
  (set acc 1.0)
  (while (< i num) (
    (set prod (* prod acc))
    (set acc (+ acc 1.0)))
  ))
  (return prod)
))
(fn my_function int
  ((a int) (b int) (c float))
  ((x int) (i int) (y float) (z float))
(
  (set c 3333.0)
  (set a (* 2 b))
  (set b 12)
  (if (== 55 104) (
    (set x (factorial 5))
    (set y 33.23122222)
    (set a b)
  ) (
    (set z 3444.0)
  ))
  (set i 0)
  (while (< i 5) (
    (set i (+ i 1)))
  ))
  (return (* 1 2))
))
```

Згенеруємо парсер засобами ANTLR4.

```
antlr4 -Dlanguage=Python3 lab_6/lab.g4
```

Маємо згенерований парсер мовою Python.

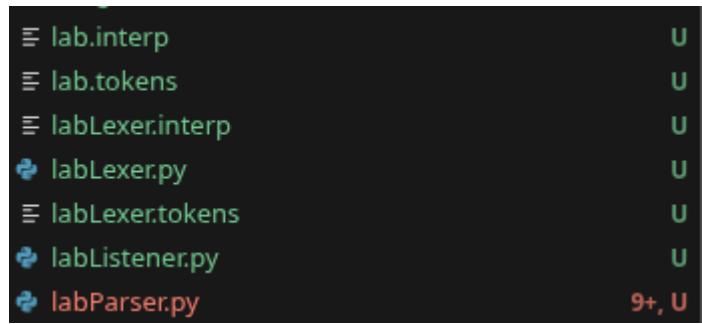


Рисунок 2.1

Код парсера.

```
# Generated from lab_6/lab.g4 by ANTLR 4.13.2
from antlr4 import *
from io import StringIO
import sys
if sys.version_info[1] > 5:
    from typing import TextIO
else:
    from typing.io import TextIO

def serializedATN():
    return [
        4, 0, 16, 122, 6, -
1, 2, 0, 7, 0, 2, 1, 7, 1, 2, 2, 7, 2, 2, 3, 7, 3, 2, 4, 7, 4, 2, 5, 7, 5,
2, 6, 7, 6, 2, 7, 7, 7, 2, 8, 7, 8, 2, 9, 7, 9, 2, 10, 7, 10, 2, 11, 7, 11, 2, 12, 7, 12, 2,
13, 7, 13, 2, 14, 7, 14, 2, 15, 7, 15, 2, 16, 7, 16, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
1, 1, 2, 1, 2, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1,
4, 1, 5, 1, 5, 1, 5, 1, 5, 1, 6, 1, 6, 1, 6, 1, 6, 1, 6, 1, 7, 1, 7, 1, 7, 1, 7, 1, 7, 1,
8, 1, 8, 1, 8, 1, 8, 1, 9, 1, 9, 1, 9, 1, 9, 1, 9, 1, 10, 1, 10, 1, 11, 1, 11, 1,
12, 3, 12, 90, 8, 12, 1, 12, 1, 12, 1, 12, 1, 13, 3, 13, 97, 8, 13, 1, 13, 1, 13,
1, 14, 4, 14, 102, 8, 14, 11, 14, 12, 14, 103, 1, 15, 1, 15, 5, 15, 108, 8, 15, 10, 15,
12, 15, 111, 9, 15, 1, 15, 3, 15, 114, 8, 15, 1, 16, 4, 16, 117, 8, 16, 11, 16, 12, 16,
118, 1, 16, 1, 16, 0, 0, 17, 1, 1, 3, 2, 5, 3, 7, 4, 9, 5, 11, 6, 13, 7, 15, 8, 17, 9, 19,
10, 21, 11, 23, 12, 25, 13, 27, 14, 29, 0, 31, 15, 33, 16, 1, 0, 5, 2, 0, 43, 43, 45, 45,
```

1, 0, 48, 57, 10, 0, 33, 33, 36, 38, 42, 42, 47, 47, 58, 58, 60, 63, 65, 90, 94, 95, 97,
122, 126, 126, 9, 0, 33, 33, 36, 38, 42, 42, 47, 58, 60, 63, 65, 90, 94, 95, 97, 122,
126, 126, 3, 0, 9, 10, 13, 13, 32, 32, 126, 0, 1, 1, 0, 0, 0, 0, 3, 1, 0, 0, 0, 0, 5, 1, 0,
0, 0, 0, 7, 1, 0, 0, 0, 0, 9, 1, 0, 0, 0, 0, 11, 1, 0, 0, 0, 0, 13, 1, 0, 0, 0, 0, 15, 1, 0, 0,
0, 0, 17, 1, 0, 0, 0, 0, 19, 1, 0, 0, 0, 0, 21, 1, 0, 0, 0, 0, 23, 1, 0, 0, 0, 0, 25, 1, 0, 0,
0, 0, 27, 1, 0, 0, 0, 0, 31, 1, 0, 0, 0, 0, 33, 1, 0, 0, 0, 1, 35, 1, 0, 0, 0, 3, 38, 1, 0, 0,
0, 5, 42, 1, 0, 0, 0, 7, 45, 1, 0, 0, 0, 9, 51, 1, 0, 0, 0, 11, 58, 1, 0, 0, 0, 13, 62, 1, 0,
0, 0, 15, 68, 1, 0, 0, 0, 17, 73, 1, 0, 0, 0, 19, 78, 1, 0, 0, 0, 21, 84, 1, 0, 0, 0, 23, 86,
1, 0, 0, 0, 25, 89, 1, 0, 0, 0, 27, 96, 1, 0, 0, 0, 29, 101, 1, 0, 0, 0, 31, 113, 1, 0, 0,
0, 33, 116, 1, 0, 0, 0, 35, 36, 5, 102, 0, 0, 36, 37, 5, 110, 0, 0, 37, 2, 1, 0, 0, 0, 38,
39, 5, 115, 0, 0, 39, 40, 5, 101, 0, 0, 40, 41, 5, 116, 0, 0, 41, 4, 1, 0, 0, 0, 42, 43,
5, 105, 0, 0, 43, 44, 5, 102, 0, 0, 44, 6, 1, 0, 0, 0, 45, 46, 5, 119, 0, 0, 46, 47, 5, 104,
0, 0, 47, 48, 5, 105, 0, 0, 48, 49, 5, 108, 0, 0, 49, 50, 5, 101, 0, 0, 50, 8, 1, 0, 0, 0,
51, 52, 5, 114, 0, 0, 52, 53, 5, 101, 0, 0, 53, 54, 5, 116, 0, 0, 54, 55, 5, 117, 0, 0,
55, 56, 5, 114, 0, 0, 56, 57, 5, 110, 0, 0, 57, 10, 1, 0, 0, 0, 58, 59, 5, 105, 0, 0, 59,
60, 5, 110, 0, 0, 60, 61, 5, 116, 0, 0, 61, 12, 1, 0, 0, 0, 62, 63, 5, 102, 0, 0, 63, 64,
5, 108, 0, 0, 64, 65, 5, 111, 0, 0, 65, 66, 5, 97, 0, 0, 66, 67, 5, 116, 0, 0, 67, 14, 1,
0, 0, 0, 68, 69, 5, 98, 0, 0, 69, 70, 5, 111, 0, 0, 70, 71, 5, 111, 0, 0, 71, 72, 5, 108,
0, 0, 72, 16, 1, 0, 0, 0, 73, 74, 5, 116, 0, 0, 74, 75, 5, 114, 0, 0, 75, 76, 5, 117, 0,
0, 76, 77, 5, 101, 0, 0, 77, 18, 1, 0, 0, 0, 78, 79, 5, 102, 0, 0, 79, 80, 5, 97, 0, 0, 80,
81, 5, 108, 0, 0, 81, 82, 5, 115, 0, 0, 82, 83, 5, 101, 0, 0, 83, 20, 1, 0, 0, 0, 84, 85,
5, 40, 0, 0, 85, 22, 1, 0, 0, 0, 86, 87, 5, 41, 0, 0, 87, 24, 1, 0, 0, 0, 88, 90, 7, 0, 0,
0, 89, 88, 1, 0, 0, 0, 89, 90, 1, 0, 0, 0, 90, 91, 1, 0, 0, 0, 91, 92, 3, 29, 14, 0, 92, 93,
5, 46, 0, 0, 93, 94, 3, 29, 14, 0, 94, 26, 1, 0, 0, 0, 95, 97, 7, 0, 0, 0, 96, 95, 1, 0, 0,

```
0,96,97,1,0,0,0,97,98,1,0,0,0,98,99,3,29,14,0,99,28,1,0,0,0,100,
102,7,1,0,0,101,100,1,0,0,0,102,103,1,0,0,0,103,101,1,0,0,0,103,
104,1,0,0,0,104,30,1,0,0,0,105,109,7,2,0,0,106,108,7,3,0,0,107,106,
1,0,0,0,108,111,1,0,0,0,109,107,1,0,0,0,109,110,1,0,0,0,110,114,
1,0,0,0,111,109,1,0,0,0,112,114,7,0,0,0,113,105,1,0,0,0,113,112,
1,0,0,0,114,32,1,0,0,0,115,117,7,4,0,0,116,115,1,0,0,0,117,118,1,
0,0,0,118,116,1,0,0,0,118,119,1,0,0,0,119,120,1,0,0,0,120,121,6,
    16,0,0,121,34,1,0,0,0,7,0,89,96,103,109,113,118,1,6,0,0
]

class labLexer(Lexer):

    atn = ATNDeserializer().deserialize(serializedATN())

    decisionsToDFA = [ DFA(ds, i) for i, ds in
enumerate(atn.decisionToState) ]

    FN = 1
    SET = 2
    IF = 3
    WHILE = 4
    RETURN = 5
    INT_T = 6
    FLOAT_T = 7
    BOOL_T = 8
    TRUE_T = 9
    FALSE_T = 10
    LPAREN = 11
    RPAREN = 12
    FLOAT = 13
    INT = 14
    IDENT = 15
    WS = 16

    channelNames = [ u"DEFAULT_TOKEN_CHANNEL", u"HIDDEN" ]

    modeNames = [ "DEFAULT_MODE" ]

    literalNames = [ "<INVALID>",
        "'fn'", "'set'", "'if'", "'while'", "'return'", "'int'",
        "'float'", "'bool'", "'true'", "'false'", "'('", "')'" ]
```

```
symbolicNames = [ "<INVALID>",
                  "FN", "SET", "IF", "WHILE", "RETURN", "INT_T",
"FLOAT_T", "BOOL_T",
                  "TRUE_T", "FALSE_T", "LPAREN", "RPAREN", "FLOAT", "INT",
"IDENT",
                  "WS" ]

ruleNames = [ "FN", "SET", "IF", "WHILE", "RETURN", "INT_T",
"FLOAT_T",
                  "BOOL_T", "TRUE_T", "FALSE_T", "LPAREN", "RPAREN",
"FLOAT",
                  "INT", "DIGITS", "IDENT", "WS" ]

grammarFileName = "lab.g4"

def __init__(self, input=None, output:TextIO = sys.stdout):
    super().__init__(input, output)
    self.checkVersion("4.13.2")
    self._interp = LexerATNSimulator(self, self.atn,
self.decisionsToDFA, PredictionContextCache())
    self._actions = None
    self._predicates = None

# Generated from lab_6/lab.g4 by ANTLR 4.13.2
from antlr4 import *
if "__name__" in __name__:
    from .labParser import labParser
else:
    from labParser import labParser

# This class defines a complete listener for a parse tree produced
# by labParser.
class labListener(ParseTreeListener):

    # Enter a parse tree produced by labParser#program.
    def enterProgram(self, ctx:labParser.ProgramContext):
        pass

    # Exit a parse tree produced by labParser#program.
    def exitProgram(self, ctx:labParser.ProgramContext):
        pass

    # Enter a parse tree produced by labParser#toplevel.
    def enterToplevel(self, ctx:labParser.ToplevelContext):
        pass

    # Exit a parse tree produced by labParser#toplevel.
```

```
def exitToplevel(self, ctx:labParser.ToplevelContext):
    pass

# Enter a parse tree produced by labParser#type.
def enterType(self, ctx:labParser.TypeContext):
    pass

# Exit a parse tree produced by labParser#type.
def exitType(self, ctx:labParser.TypeContext):
    pass

# Enter a parse tree produced by labParser#funcArgs.
def enterFuncArgs(self, ctx:labParser.FuncArgsContext):
    pass

# Exit a parse tree produced by labParser#funcArgs.
def exitFuncArgs(self, ctx:labParser.FuncArgsContext):
    pass

# Enter a parse tree produced by labParser#varDecls.
def enterVarDecls(self, ctx:labParser.VarDeclsContext):
    pass

# Exit a parse tree produced by labParser#varDecls.
def exitVarDecls(self, ctx:labParser.VarDeclsContext):
    pass

# Enter a parse tree produced by labParser#argPair.
def enterArgPair(self, ctx:labParser.ArgPairContext):
    pass

# Exit a parse tree produced by labParser#argPair.
def exitArgPair(self, ctx:labParser.ArgPairContext):
    pass

# Enter a parse tree produced by labParser#stmtList.
def enterStmtList(self, ctx:labParser.StmtListContext):
    pass

# Exit a parse tree produced by labParser#stmtList.
def exitStmtList(self, ctx:labParser.StmtListContext):
    pass
```

```
# Enter a parse tree produced by labParser#statement.
def enterStatement(self, ctx:labParser.StatementContext):
    pass

# Exit a parse tree produced by labParser#statement.
def exitStatement(self, ctx:labParser.StatementContext):
    pass

# Enter a parse tree produced by labParser#expr.
def enterExpr(self, ctx:labParser.ExprContext):
    pass

# Exit a parse tree produced by labParser#expr.
def exitExpr(self, ctx:labParser.ExprContext):
    pass

# Enter a parse tree produced by labParser#atom.
def enterAtom(self, ctx:labParser.AtomContext):
    pass

# Exit a parse tree produced by labParser#atom.
def exitAtom(self, ctx:labParser.AtomContext):
    pass

# Enter a parse tree produced by labParser#funcCall.
def enterFuncCall(self, ctx:labParser.FuncCallContext):
    pass

# Exit a parse tree produced by labParser#funcCall.
def exitFuncCall(self, ctx:labParser.FuncCallContext):
    pass

del labParser

# Generated from lab_6/lab.g4 by ANTLR 4.13.2
# encoding: utf-8
from antlr4 import *
from io import StringIO
import sys
if sys.version_info[1] > 5:
    from typing import TextIO
else:
    from typing.io import TextIO
```

```
12,0,0,72,92,1,0,0,0,73,74,5,11,0,0,74,75,5,3,0,0,75,76,3,16,8,0,
76,77,3,12,6,0,77,78,3,12,6,0,78,79,5,12,0,0,79,92,1,0,0,0,80,81,
5,11,0,0,81,82,5,4,0,0,82,83,3,16,8,0,83,84,3,12,6,0,84,85,5,12,
0,0,85,92,1,0,0,0,86,87,5,11,0,0,87,88,5,5,0,0,88,89,3,16,8,0,89,
90,5,12,0,0,90,92,1,0,0,0,91,67,1,0,0,0,91,73,1,0,0,0,91,80,1,0,
0,0,91,86,1,0,0,0,92,15,1,0,0,0,93,96,3,18,9,0,94,96,3,20,10,0,95,
93,1,0,0,0,95,94,1,0,0,0,96,17,1,0,0,0,97,98,7,1,0,0,98,19,1,0,0,
0,99,100,5,11,0,0,100,104,5,15,0,0,101,103,3,18,9,0,102,101,1,0,
0,0,103,106,1,0,0,0,104,102,1,0,0,0,104,105,1,0,0,0,105,107,1,0,
0,0,106,104,1,0,0,0,107,108,5,12,0,0,108,21,1,0,0,0,7,25,45,47,62,
         91,95,104
    ]
```

```
class labParser ( Parser ):

    grammarFileName = "lab.g4"

    atn = ATNDeserializer().deserialize(serializedATN())

    decisionsToDFA = [ DFA(ds, i) for i, ds in
enumerate(atn.decisionToState) ]

    sharedContextCache = PredictionContextCache()

    literalNames = [ "<INVALID>", "'fn'", "'set'", "'if'",
"'while'", "'return'", "'int'", "'float'", "'bool'", "'true'",
"'false'", "'('", "')'" ]
```

```
    symbolicNames = [ "<INVALID>", "FN", "SET", "IF", "WHILE",
"RETURN",
                     "INT_T", "FLOAT_T", "BOOL_T", "TRUE_T",
"FALSE_T",
                     "LPAREN", "RPAREN", "FLOAT", "INT", "IDENT",
"WS" ]
```

```
    RULE_program = 0
    RULE_toplevel = 1
```

```
RULE_type = 2
RULE_funcArgs = 3
RULE_varDecls = 4
RULE_argPair = 5
RULE_stmtList = 6
RULE_statement = 7
RULE_expr = 8
RULE_atom = 9
RULE_funcCall = 10

ruleNames = [ "program", "toplevel", "type", "funcArgs",
"varDecls",
"argPair", "stmtList", "statement", "expr",
"atom", "funcCall" ]

EOF = Token.EOF
FN=1
SET=2
IF=3
WHILE=4
RETURN=5
INT_T=6
FLOAT_T=7
BOOL_T=8
TRUE_T=9
FALSE_T=10
LPAREN=11
RPAREN=12
FLOAT=13
INT=14
IDENT=15
WS=16

def __init__(self, input:TokenStream, output:TextIO =
sys.stdout):
    super().__init__(input, output)
    self.checkVersion("4.13.2")
    self._interp = ParserATNSimulator(self, self.atn,
self.decisionsToDFA, self.sharedContextCache)
    self._predicates = None

class ProgramContext(ParserRuleContext):
    __slots__ = 'parser'

    def __init__(self, parser, parent:ParserRuleContext=None,
invokingState:int=-1):
```

```
super().__init__(parent, invokingState)
self.parser = parser

def EOF(self):
    return self.getToken(labParser.EOF, 0)

def toplevel(self, i:int=None):
    if i is None:
        return
self.getTypedRuleContexts(labParser.ToplevelContext)
else:
    return
self.getTypedRuleContext(labParser.ToplevelContext,i)

def getRuleIndex(self):
    return labParser.RULE_program

def enterRule(self, listener:ParseTreeListener):
    if hasattr( listener, "enterProgram" ):
        listener.enterProgram(self)

def exitRule(self, listener:ParseTreeListener):
    if hasattr( listener, "exitProgram" ):
        listener.exitProgram(self)

def program(self):
    localctx = labParser.ProgramContext(self, self._ctx,
self.state)
    self.enterRule(localctx, 0, self.RULE_program)
    self._la = 0 # Token type
    try:
        self.enterOuterAlt(localctx, 1)
        self.state = 25
        self._errHandler.sync(self)
        _la = self._input.LA(1)
        while _la==11:
            self.state = 22
            self.toplevel()
            self.state = 27
            self._errHandler.sync(self)
            _la = self._input.LA(1)

        self.state = 28
        self.match(labParser.EOF)
```

```
        except RecognitionException as re:
            localctx.exception = re
            self._errHandler.reportError(self, re)
            self._errHandler.recover(self, re)
        finally:
            self.exitRule()
        return localctx

class ToplevelContext(ParserRuleContext):
    __slots__ = 'parser'

    def __init__(self, parser, parent:ParserRuleContext=None,
invokingState:int=-1):
        super().__init__(parent, invokingState)
        self.parser = parser
        self.name = None # Token
        self.retType = None # TypeContext

    def LPAREN(self):
        return self.getToken(labParser.LPAREN, 0)

    def FN(self):
        return self.getToken(labParser.FN, 0)

    def funcArgs(self):
        return
    self.getTypedRuleContext(labParser.FuncArgsContext, 0)

    def varDecls(self):
        return
    self.getTypedRuleContext(labParser.VarDeclsContext, 0)

    def stmtList(self):
        return
    self.getTypedRuleContext(labParser.StmtListContext, 0)

    def RPAREN(self):
        return self.getToken(labParser.RPAREN, 0)

    def IDENT(self):
        return self.getToken(labParser.IDENT, 0)

    def type_(self):
        return self.getTypedRuleContext(labParser.TypeContext, 0)
```

```
def getRuleIndex(self):
    return labParser.RULE_toplevel

def enterRule(self, listener:ParseTreeListener):
    if hasattr( listener, "enterToplevel" ):
        listener.enterToplevel(self)

def exitRule(self, listener:ParseTreeListener):
    if hasattr( listener, "exitToplevel" ):
        listener.exitToplevel(self)

def toplevel(self):
    localctx = labParser.ToplevelContext(self, self._ctx,
self.state)
    self.enterRule(localctx, 2, self.RULE_toplevel)
    try:
        self.enterOuterAlt(localctx, 1)
        self.state = 30
        self.match(labParser.LPAREN)
        self.state = 31
        self.match(labParser.FN)
        self.state = 32
        localctx.name = self.match(labParser.IDENT)
        self.state = 33
        localctx.retType = self.type_()
        self.state = 34
        self.funcArgs()
        self.state = 35
        self.varDecls()
        self.state = 36
        self.stmtList()
        self.state = 37
        self.match(labParser.RPAREN)
    except RecognitionException as re:
        localctx.exception = re
        self._errHandler.reportError(self, re)
        self._errHandler.recover(self, re)
    finally:
        self.exitRule()
    return localctx

class TypeContext(ParserRuleContext):
    __slots__ = 'parser'
```

```
def __init__(self, parser, parent:ParserRuleContext=None,
invokingState:int=-1):
    super().__init__(parent, invokingState)
    self.parser = parser

def INT_T(self):
    return self.getToken(labParser.INT_T, 0)

def FLOAT_T(self):
    return self.getToken(labParser.FLOAT_T, 0)

def BOOL_T(self):
    return self.getToken(labParser.BOOL_T, 0)

def getRuleIndex(self):
    return labParser.RULE_type

def enterRule(self, listener:ParseTreeListener):
    if hasattr( listener, "enterType" ):
        listener.enterType(self)

def exitRule(self, listener:ParseTreeListener):
    if hasattr( listener, "exitType" ):
        listener.exitType(self)

def type_(self):
    localctx = labParser.TypeContext(self, self._ctx,
self.state)
    self.enterRule(localctx, 4, self.RULE_type)
    self._la = 0 # Token type
    try:
        self.enterOuterAlt(localctx, 1)
        self.state = 39
        _la = self._input.LA(1)
        if not((((_la) & ~0x3f) == 0 and ((1 << _la) & 448) != 0)):
            self._errHandler.recoverInline(self)
        else:
            self._errHandler.reportMatch(self)
            self.consume()
    except RecognitionException as re:
        localctx.exception = re
        self._errHandler.reportError(self, re)
        self._errHandler.recover(self, re)
```

```
        finally:
            self.exitRule()
        return localctx

class FuncArgsContext(ParserRuleContext):
    __slots__ = 'parser'

    def __init__(self, parser, parent:ParserRuleContext=None,
invokingState:int=-1):
        super().__init__(parent, invokingState)
        self.parser = parser

    def LPAREN(self):
        return self.getToken(labParser.LPAREN, 0)

    def RPAREN(self):
        return self.getToken(labParser.RPAREN, 0)

    def argPair(self, i:int=None):
        if i is None:
            return
        self.getTypedRuleContexts(labParser.ArgBaseContext)
        else:
            return
        self.getTypedRuleContext(labParser.ArgBaseContext,i)

    def getRuleIndex(self):
        return labParser.RULE_funcArgs

    def enterRule(self, listener:ParseTreeListener):
        if hasattr( listener, "enterFuncArgs" ):
            listener.enterFuncArgs(self)

    def exitRule(self, listener:ParseTreeListener):
        if hasattr( listener, "exitFuncArgs" ):
            listener.exitFuncArgs(self)

def funcArgs(self):

    localctx = labParser.FuncArgsContext(self, self._ctx,
self.state)
    self.enterRule(localctx, 6, self.RULE_funcArgs)
    self._la = 0 # Token type
    try:
```

```
        self.enterOuterAlt(localctx, 1)
        self.state = 41
        self.match(labParser.LPAREN)
        self.state = 47
        self._errHandler.sync(self)
        _la = self._input.LA(1)
        if _la==11:
            self.state = 43
            self._errHandler.sync(self)
            _la = self._input.LA(1)
            while True:
                self.state = 42
                self.argPair()
                self.state = 45
                self._errHandler.sync(self)
                _la = self._input.LA(1)
                if not (_la==11):
                    break

                self.state = 49
                self.match(labParser.RPAREN)
            except RecognitionException as re:
                localctx.exception = re
                self._errHandler.reportError(self, re)
                self._errHandler.recover(self, re)
            finally:
                self.exitRule()
        return localctx

class VarDeclsContext(ParserRuleContext):
    __slots__ = 'parser'

    def __init__(self, parser, parent:ParserRuleContext=None,
invokingState:int=-1):
        super().__init__(parent, invokingState)
        self.parser = parser

    def funcArgs(self):
        return
    self.getTypedRuleContext(labParser.FuncArgsContext, 0)

    def getRuleIndex(self):
        return labParser.RULE_varDecls

    def enterRule(self, listener:ParseTreeListener):
```

```
        if hasattr( listener, "enterVarDecls" ):
            listener.enterVarDecls(self)

    def exitRule(self, listener:ParseTreeListener):
        if hasattr( listener, "exitVarDecls" ):
            listener.exitVarDecls(self)

def varDecls(self):

    localctx = labParser.VarDeclsContext(self, self._ctx,
self.state)
    self.enterRule(localctx, 8, self.RULE_varDecls)
    try:
        self.enterOuterAlt(localctx, 1)
        self.state = 51
        self.funcArgs()
    except RecognitionException as re:
        localctx.exception = re
        self._errHandler.reportError(self, re)
        self._errHandler.recover(self, re)
    finally:
        self.exitRule()
    return localctx

class ArgPairContext(ParserRuleContext):
    __slots__ = 'parser'

    def __init__(self, parser, parent:ParserRuleContext=None,
invokingState:int=-1):
        super().__init__(parent, invokingState)
        self.parser = parser
        self.name = None # Token

    def LPAREN(self):
        return self.getToken(labParser.LPAREN, 0)

    def type_(self):
        return self.getTypedRuleContext(labParser.TypeContext, 0)

    def RPAREN(self):
        return self.getToken(labParser.RPAREN, 0)

    def IDENT(self):
        return self.getToken(labParser.IDENT, 0)
```

```
def getRuleIndex(self):
    return labParser.RULE_argPair

def enterRule(self, listener:ParseTreeListener):
    if hasattr( listener, "enterArgPair" ):
        listener.enterArgPair(self)

def exitRule(self, listener:ParseTreeListener):
    if hasattr( listener, "exitArgPair" ):
        listener.exitArgPair(self)

def argPair(self):

    localctx = labParser.ArgBaseContext(self, self._ctx,
self.state)
    self.enterRule(localctx, 10, self.RULE_argPair)
    try:
        self.enterOuterAlt(localctx, 1)
        self.state = 53
        self.match(labParser.LPAREN)
        self.state = 54
        localctx.name = self.match(labParser.IDENT)
        self.state = 55
        self.type_()
        self.state = 56
        self.match(labParser.RPAREN)
    except RecognitionException as re:
        localctx.exception = re
        self._errHandler.reportError(self, re)
        self._errHandler.recover(self, re)
    finally:
        self.exitRule()
    return localctx

class StmtListContext(ParserRuleContext):
    __slots__ = 'parser'

    def __init__(self, parser, parent:ParserRuleContext=None,
invokingState:int=-1):
        super().__init__(parent, invokingState)
        self.parser = parser

    def LPAREN(self):
        return self.getToken(labParser.LPAREN, 0)
```

```
def RPAREN(self):
    return self.getToken(labParser.RPAREN, 0)

def statement(self, i:int=None):
    if i is None:
        return
self.getTypedRuleContexts(labParser.StatementContext)
    else:
        return
self.getTypedRuleContext(labParser.StatementContext,i)

def getRuleIndex(self):
    return labParser.RULE_stmtList

def enterRule(self, listener:ParseTreeListener):
    if hasattr( listener, "enterStmtList" ):
        listener.enterStmtList(self)

def exitRule(self, listener:ParseTreeListener):
    if hasattr( listener, "exitStmtList" ):
        listener.exitStmtList(self)

def stmtList(self):

    localctx = labParser.StmtListContext(self, self._ctx,
self.state)
    self.enterRule(localctx, 12, self.RULE_stmtList)
    self._la = 0 # Token type
    try:
        self.enterOuterAlt(localctx, 1)
        self.state = 58
        self.match(labParser.LPAREN)
        self.state = 62
        self._errHandler.sync(self)
        _la = self._input.LA(1)
        while _la==11:
            self.state = 59
            self.statement()
            self.state = 64
            self._errHandler.sync(self)
            _la = self._input.LA(1)

        self.state = 65
        self.match(labParser.RPAREN)
```

```
        except RecognitionException as re:
            localctx.exception = re
            self._errHandler.reportError(self, re)
            self._errHandler.recover(self, re)
        finally:
            self.exitRule()
        return localctx

class StatementContext(ParserRuleContext):
    __slots__ = 'parser'

    def __init__(self, parser, parent:ParserRuleContext=None,
invokingState:int=-1):
        super().__init__(parent, invokingState)
        self.parser = parser
        self.dest = None # Token
        self.cond = None # ExprContext
        self.thenBranch = None # StmtListContext
        self.elseBranch = None # StmtListContext
        self.body = None # StmtListContext

    def LPAREN(self):
        return self.getToken(labParser.LPAREN, 0)

    def SET(self):
        return self.getToken(labParser.SET, 0)

    def expr(self):
        return self.getTypedRuleContext(labParser.ExprContext, 0)

    def RPAREN(self):
        return self.getToken(labParser.RPAREN, 0)

    def IDENT(self):
        return self.getToken(labParser.IDENT, 0)

    def IF(self):
        return self.getToken(labParser.IF, 0)

    def stmtList(self, i:int=None):
        if i is None:
            return
        self.getTypedRuleContexts(labParser.StmtListContext)
        else:
            return
        self.getTypedRuleContext(labParser.StmtListContext, i)
```

```
def WHILE(self):
    return self.getToken(labParser.WHILE, 0)

def RETURN(self):
    return self.getToken(labParser.RETURN, 0)

def getRuleIndex(self):
    return labParser.RULE_statement

def enterRule(self, listener:ParseTreeListener):
    if hasattr( listener, "enterStatement" ):
        listener.enterStatement(self)

def exitRule(self, listener:ParseTreeListener):
    if hasattr( listener, "exitStatement" ):
        listener.exitStatement(self)

def statement(self):

    localctx = labParser.StatementContext(self, self._ctx,
self.state)
    self.enterRule(localctx, 14, self.RULE_statement)
    try:
        self.state = 91
        self._errHandler.sync(self)
        la_ =
self._interp.adaptivePredict(self._input,4,self._ctx)
        if la_ == 1:
            self.enterOuterAlt(localctx, 1)
            self.state = 67
            self.match(labParser.LPAREN)
            self.state = 68
            self.match(labParser.SET)
            self.state = 69
            localctx.dest = self.match(labParser.IDENT)
            self.state = 70
            self.expr()
            self.state = 71
            self.match(labParser.RPAREN)
            pass

        elif la_ == 2:
            self.enterOuterAlt(localctx, 2)
            self.state = 73
            self.match(labParser.LPAREN)
```

```
        self.state = 74
        self.match(labParser.IF)
        self.state = 75
        localctx.cond = self.expr()
        self.state = 76
        localctx.thenBranch = self.stmtList()
        self.state = 77
        localctx.elseBranch = self.stmtList()
        self.state = 78
        self.match(labParser.RPAREN)
        pass

    elif la_ == 3:
        self.enterOuterAlt(localctx, 3)
        self.state = 80
        self.match(labParser.LPAREN)
        self.state = 81
        self.match(labParser.WHILE)
        self.state = 82
        localctx.cond = self.expr()
        self.state = 83
        localctx.body = self.stmtList()
        self.state = 84
        self.match(labParser.RPAREN)
        pass

    elif la_ == 4:
        self.enterOuterAlt(localctx, 4)
        self.state = 86
        self.match(labParser.LPAREN)
        self.state = 87
        self.match(labParser.RETURN)
        self.state = 88
        self.expr()
        self.state = 89
        self.match(labParser.RPAREN)
        pass

except RecognitionException as re:
    localctx.exception = re
    self._errHandler.reportError(self, re)
    self._errHandler.recover(self, re)
finally:
    self.exitRule()
return localctx
```

```
class ExprContext(ParserRuleContext):
```

```
__slots__ = 'parser'

    def __init__(self, parser, parent:ParserRuleContext=None,
invokingState:int=-1):
        super().__init__(parent, invokingState)
        self.parser = parser

    def atom(self):
        return self.getTypedRuleContext(labParser.AtomContext, 0)

    def funcCall(self):
        return
self.getTypedRuleContext(labParser.FuncCallContext, 0)

    def getRuleIndex(self):
        return labParser.RULE_expr

    def enterRule(self, listener:ParseTreeListener):
        if hasattr( listener, "enterExpr" ):
            listener.enterExpr(self)

    def exitRule(self, listener:ParseTreeListener):
        if hasattr( listener, "exitExpr" ):
            listener.exitExpr(self)

def expr(self):

    localctx = labParser.ExprContext(self, self._ctx,
self.state)
    self.enterRule(localctx, 16, self.RULE_expr)
    try:
        self.state = 95
        self._errHandler.sync(self)
        token = self._input.LA(1)
        if token in [9, 10, 13, 14, 15]:
            self.enterOuterAlt(localctx, 1)
            self.state = 93
            self.atom()
            pass
        elif token in [11]:
            self.enterOuterAlt(localctx, 2)
            self.state = 94
            self.funcCall()
            pass
```

```
        else:
            raise NoViableAltException(self)

    except RecognitionException as re:
        localctx.exception = re
        self._errHandler.reportError(self, re)
        self._errHandler.recover(self, re)
    finally:
        self.exitRule()
    return localctx


class AtomContext(ParserRuleContext):
    __slots__ = 'parser'

    def __init__(self, parser, parent:ParserRuleContext=None,
invokingState:int=-1):
        super().__init__(parent, invokingState)
        self.parser = parser

    def IDENT(self):
        return self.getToken(labParser.IDENT, 0)

    def INT(self):
        return self.getToken(labParser.INT, 0)

    def FLOAT(self):
        return self.getToken(labParser.FLOAT, 0)

    def TRUE_T(self):
        return self.getToken(labParser.TRUE_T, 0)

    def FALSE_T(self):
        return self.getToken(labParser.FALSE_T, 0)

    def getRuleIndex(self):
        return labParser.RULE_atom

    def enterRule(self, listener:ParseTreeListener):
        if hasattr( listener, "enterAtom" ):
            listener.enterAtom(self)

    def exitRule(self, listener:ParseTreeListener):
        if hasattr( listener, "exitAtom" ):
            listener.exitAtom(self)
```

```
def atom(self):

    localctx = labParser.AtomContext(self, self._ctx,
self.state)
    self.enterRule(localctx, 18, self.RULE_atom)
    self._la = 0 # Token type
    try:
        self.enterOuterAlt(localctx, 1)
        self.state = 97
        _la = self._input.LA(1)
        if not((((_la) & ~0x3f) == 0 and ((1 << _la) & 58880) != 0)):
            self._errHandler.recoverInline(self)
        else:
            self._errHandler.reportMatch(self)
            self.consume()
    except RecognitionException as re:
        localctx.exception = re
        self._errHandler.reportError(self, re)
        self._errHandler.recover(self, re)
    finally:
        self.exitRule()
    return localctx

class FuncCallContext(ParserRuleContext):
    __slots__ = 'parser'

    def __init__(self, parser, parent:ParserRuleContext=None,
invokingState:int=-1):
        super().__init__(parent, invokingState)
        self.parser = parser
        self.callee = None # Token

    def LPAREN(self):
        return self.getToken(labParser.LPAREN, 0)

    def RPAREN(self):
        return self.getToken(labParser.RPAREN, 0)

    def IDENT(self):
        return self.getToken(labParser.IDENT, 0)

    def atom(self, i:int=None):
        if i is None:
            return
        self.getTypedRuleContexts(labParser.AtomContext)
    else:
```

```
        return
self.getTypedRuleContext(labParser.AtomContext,i)

    def getRuleIndex(self):
        return labParser.RULE_funcCall

    def enterRule(self, listener:ParseTreeListener):
        if hasattr( listener, "enterFuncCall" ):
            listener.enterFuncCall(self)

    def exitRule(self, listener:ParseTreeListener):
        if hasattr( listener, "exitFuncCall" ):
            listener.exitFuncCall(self)

def funcCall(self):

    localctx = labParser.FuncCallContext(self, self._ctx,
self.state)
    self.enterRule(localctx, 20, self.RULE_funcCall)
    self._la = 0 # Token type
    try:
        self.enterOuterAlt(localctx, 1)
        self.state = 99
        self.match(labParser.LPAREN)
        self.state = 100
        localctx.callee = self.match(labParser.IDENT)
        self.state = 104
        self._errHandler.sync(self)
        _la = self._input.LA(1)
        while (((_la) & ~0x3f) == 0 and ((1 << _la) & 58880) != 0):
            self.state = 101
            self.atom()
            self.state = 106
            self._errHandler.sync(self)
            _la = self._input.LA(1)

        self.state = 107
        self.match(labParser.RPAREN)
    except RecognitionException as re:
        localctx.exception = re
        self._errHandler.reportError(self, re)
        self._errHandler.recover(self, re)
    finally:
        self.exitRule()
```

```
    return localctx
```

Тепер напишемо виведення дерева згенерованим парсером.

```
import sys
from antlr4 import *

# Assuming your grammar file is named 'lab.g4', ANTLR generates
these
from labLexer import labLexer
from labParser import labParser

def parse_file(file_path, start_rule_name):
    """
        Reads an input file and uses the ANTLR-generated parser to
create a parse tree.
    """
    try:
        # Read the file content
        input_stream = FileStream(file_path, encoding='utf-8')
    except Exception as e:
        print(f"Error reading file: {e}")
        return

    # 1. Create a Lexer
    lexer = labLexer(input_stream)
    stream = CommonTokenStream(lexer)

    # 2. Create a Parser
    parser = labParser(stream)

    # Enable error reporting (optional, but useful)
    # parser.addErrorListener(ConsoleErrorListener.INSTANCE)

    # 3. Invoke the starting rule dynamically
    # The 'start_rule_name' is the first rule the parser should
match (e.g., 'program')
    try:
        start_rule = getattr(parser, start_rule_name)
        tree = start_rule()

        # 4. Print the parse tree in LISP format
        print("--- Parse Tree Output (LISP format) ---")
        print(tree.toStringTree(recog=parser))
        print("-----")

        # Optional: Walk the tree (for actual compiler logic)
        # walker = ParseTreeWalker()
```

```
# walker.walk(MyListener(), tree) # Use a custom Listener or
Visitor here

except AttributeError:
    print(f"Error: Start rule '{start_rule_name}' not found in
the generated parser.")
except Exception as e:
    print(f"Parsing error: {e}")

if __name__ == '__main__':
    # You are in the 'lab_6' directory, so the paths are local
    grammar_file = 'lab.g4' # For reference
    input_file = 'example.txt'

    # Use the same starting rule you used with antlr4-parse
    start_rule = 'program'

    print(f"Parsing '{input_file}' using rule '{start_rule}'...")
    parse_file(input_file, start_rule)
```

Згенероване синтаксичне дерево засобами ANTLR4.

```
Parsing 'example.txt' using rule 'program'...
--- Parse Tree Output (LISP format) ---
(program (toplevel ( fn factorial (type int) (funcArgs ( (argPair
( num (type int) )))) (varDecls (funcArgs ( (argPair ( prod (type
int) )) (argPair ( i (type int) )) (argPair ( acc (type int) )) ))))
(stmtList ( (statement ( set prod (expr (atom 1)))) (statement
( set i (expr (atom 0)))) (statement ( set acc (expr (atom 1)))) )
(statement ( while (expr (funcCall ( < (atom i) (atom num) )))))
(stmtList ( (statement ( set prod (expr (funcCall ( * (atom prod)
(atom acc) )))) (statement ( set acc (expr (funcCall ( + (atom
acc) (atom 1) ))))) (statement ( set prod (expr (return (expr (atom prod)))) ))
(toplevel ( fn factorial (type float) (funcArgs ( (argPair
( num (type float) )))) (varDecls (funcArgs ( (argPair ( prod (type
float) )) (argPair ( i (type float) )) (argPair ( acc (type
float) ))))) (stmtList ( (statement ( set prod (expr (atom 1.0)))) )
(statement ( set i (expr (atom 0.0)))) (statement ( set acc (expr
(atom 1.0)))) (statement ( while (expr (funcCall ( < (atom i) (atom
num) )))) (stmtList ( (statement ( set prod (expr (funcCall ( * (atom
prod) (atom acc) )))) (statement ( set acc (expr (funcCall ( +
(atom acc) (atom 1.0) ))))) (statement ( return (expr (atom prod)))) )) (statement ( set prod (expr (atom 1.0)))) (toplevel ( fn my_function (type int) (funcArgs
( (argPair ( a (type int) )) (argPair ( b (type int) )) (argPair ( c
(type float) )))) (varDecls (funcArgs ( (argPair ( x (type int) ))
(argPair ( i (type int) )) (argPair ( y (type float) )) (argPair ( z
(type float) ))))) (stmtList ( (statement ( set c (expr (atom
3333.0)))) (statement ( set a (expr (funcCall ( * (atom 2) (atom b
```

```
))) )) (statement ( set b (expr (atom 12)) )) (statement ( if (expr
(funcCall ( == (atom 55) (atom 104) ))) (stmtList ( (statement ( set
x (expr (funcCall ( factorial (atom 5) )))) )) (statement ( set y
(expr (atom 33.23122222)) )) (statement ( set a (expr (atom
b)) )) )) (stmtList ( (statement ( set z (expr (atom
3444.0)) )) )) )) (statement ( set i (expr (atom 0)) )) (statement (
while (expr (funcCall ( < (atom i) (atom 5) ))) (stmtList
( (statement ( set i (expr (funcCall ( + (atom i) (atom 1) )))) )) ))
)) (statement ( return (expr (funcCall ( * (atom 1) (atom
2) )))) )) )) <EOF>)
```

3 ВИСНОВОК

У ході виконання лабораторної роботи було розроблено граматику мови Lab та реалізовано лексичний і синтаксичний аналіз за допомогою інструменту ANTLR4.