**Програмна реалізація лексичного аналізатора заданої мови з використанням діаграми станів**

**Завдання варіанту № 15**

Арифметика: цiлі та дiйсні числа, основні чотири арифметичні операції (додавання, віднімання, ділення та множення), піднесення до степення (правоасоціативна операція), дужки

Особливості:

1. експоненційна форма дійсного числа
2. інструкція повторення: for (<ід>=<вираз1>; <відношення>; <вираз2>)<блок операторів>
3. інструкція розгалуження: if <відношення> then goto <мітка>

**Граматика мови**

Program = program ProgName DeclSection DoSection

ProgName = Ident

Ident = Letter {Letter | Digit }

DeclSection = var DeclarList

DeclarList = Declaration {’,’ Declaration }

Declaration = identtList ’:’ Type

identtList = ident {’,’ ident}

Type = int | real | boolean

DoSection = begin StatementList end

StatementList = Statement {’;’ Statement }

Statement = [Mark ‘:’] Assign | in | Out | ForStatement | Conditional

Assign = ident ’=’ Expression

Expression = ArithmExpression | BoolExpr

BoolExpr = [‘!’] (Expression RelOp Expression) | BoolConst | Ident

ArithmExpression = [Sign] Term | ArithmExpression ’+’ Term |

ArithmExpression ’-’ Term

Term = Factor | Term ’\*’ Factor | Term ’/’ Factor | Term ’^’ Factor

Factor = ident | Const | ’(’ ArithmExpression ’)’

in = read ’(’ identtList ’)’

Out = write ’(’ identtList ’)’

ForStatement = for indExpr DoBlock

indExpr = ‘(‘ Assign ‘;’ BoolExpr ’;’ Expression ‘)’

DoBlock = Statement ‘;’ | ’begin’ StatementList ’end’

Const = intNumb | RealNumb | BoolConst

Conditional = if BoolExpr then goto Mark

Mark = ident

intNumb = [Sign] Unsignedint

RealNumb = [Sign] UnsignedReal

Sign = ’+’ | ’-’

Unsignedint = Digit {Digit}

UnsignedReal = ’.’ Unsignedint | Unsignedint ’.’ | Unsignedint ’.’

Unsignedint | ExponentForm

ExponentForm = UnsignedReal ‘E’ [Sign] intNumb

Letter = ’a’ | ’b’ | ’c’ | ’d’ | ’e’ | ’f’ | ’g’ | ’h’ | ’i’ | ’j’ | ’k’ | ’l’ | ’m’ | ’n’ | ’o’ | ’p’ | ’q’ | ’r’ | ’s’ | ’t’ | ’u’ | ’v’ | ’w’ | ’x’ | ’y’ | ’z’ | ’A’ | ’B’ | ’C’ | ’D’ | ’E’ | ’F’ | ’G’ | ’H’ | ’i’ | ’J’ | ’K’ | ’L’ | ’M’ | ’N’ | ’O’ | ’P’ | ’Q’ | ’R’ | ’S’ | ’T’ | ’U’ | ’V’ | ’W’ | ’X’ | ’Y’ | ’Z’

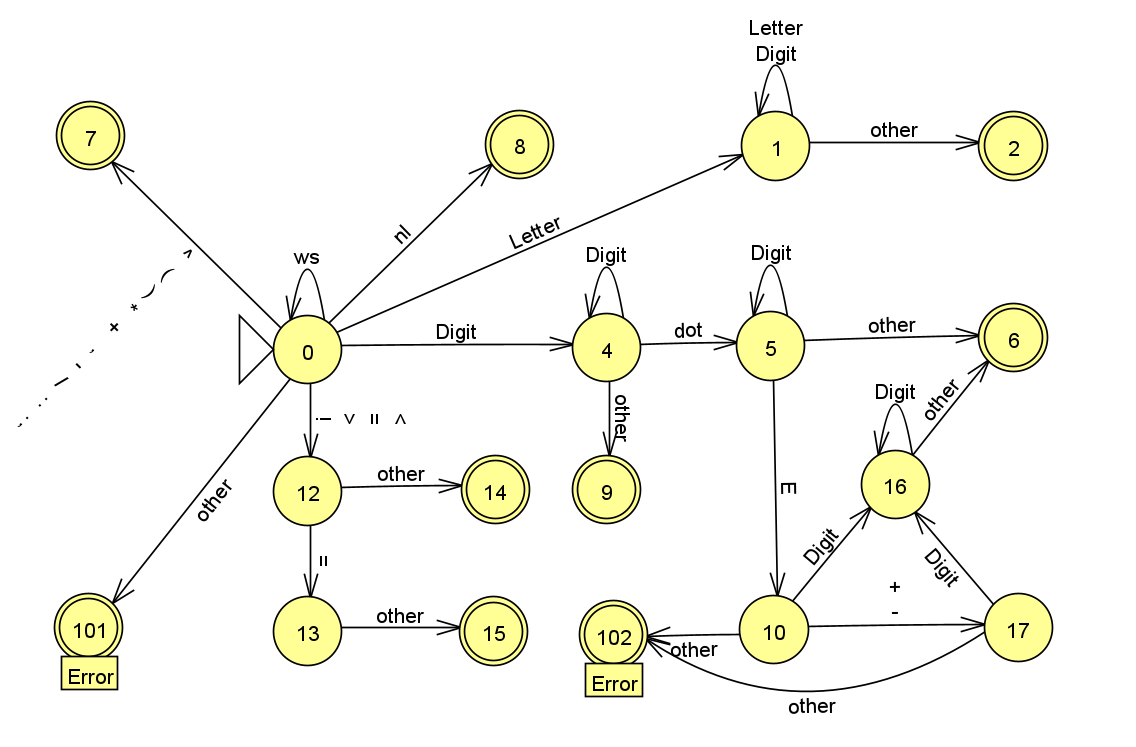
Digit = ’0’ | ’1’ | ’2’ | ’3’ | ’4’ | ’5’ | ’6’ | ’7’ | ’8’ | ’9’

BoolConst = true | false

RelOp = ’==’ | ’<= ’ | ’<’ | ’>’ | ’>=’ | ‘!=’

**Таблиця лексем**

|  |  |  |  |
| --- | --- | --- | --- |
| Код | Приклади лексем | Токен | Неформальний опис |
| 1 | Ab , g , Gg3, h1hj4l | id | ідентифікатор |
| 2 | 123, 0, 4632 | int | ціле без знака |
| 3 | 1.2, .44, 222. , 23.3-E5, 2.E2 | real | дійсне без знака |
| 4 | true, false | bool | логічне значення |
| 5 | program | keyword | символ program |
| 6 | var | keyword | символ var |
| 7 | begin | keyword | символ begin |
| 8 | end | keyword | символ end |
| 9 | end. | keyword | символ end. |
| 10 | int | keyword | символ int |
| 11 | boolean | keyword | символ boolean |
| 12 | read | keyword | символ read |
| 13 | write | keyword | символ write |
| 14 | for | keyword | символ for |
| 15 | if | keyword | символ if |
| 16 | then | keyword | символ then |
| 17 | goto | keyword | символ goto |
| 18 | = | assign\_op | символ = |
| 19 | + | add\_op | символ + |
| 20 | - | add\_op | символ - |
| 21 | \* | mult\_op | символ \* |
| 22 | / | mult\_op | символ / |
| 23 | ^ | pow\_op | символ ^ |
| 24 | < | rel\_op | символ < |
| 25 | <= | rel\_op | символ <= |
| 26 | == | rel\_op | символ == |
| 27 | > | rel\_op | символ > |
| 28 | >= | rel\_op | символ >= |
| 29 | != | rel\_op | символ != |
| 30 | ! | rel\_op | символ ! |
| 31 | ( | brackets\_op | символ ( |
| 32 | ) | brackets\_op | символ ) |
| 33 | . | dot | символ . |
| 34 | , | coma | символ , |
| 35 | : | colon | символ : |
| 36 | ; | end\_colon | символ ; |
| 37 | \n | eol | кінець рядка |
| 38 | tb | keyword | символ tb |
| 39 | nl | keyword | символ nl |

**Діаграма станів**

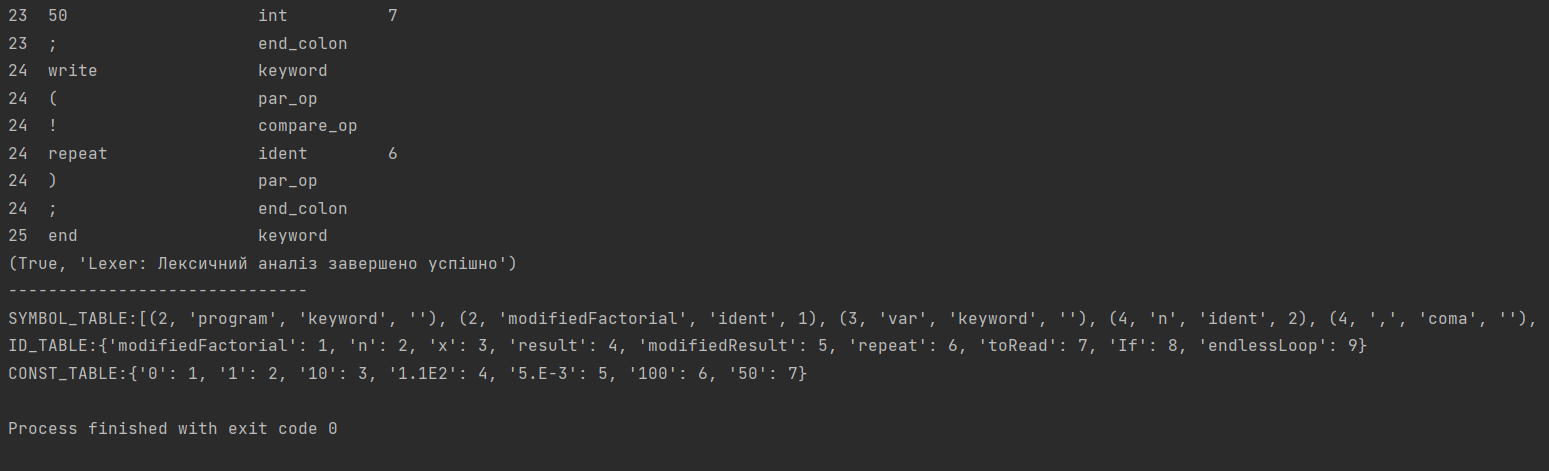
**Опис**

Лексичний аналізатор зчитує по одному символу. За допомогою нього та значенню попереднього стану визначається клас символу, наприклад з символ належить до цифр (Digit). З класу та попереднього стану дізнаємося в новий стан. Якщо цей стан не кінцевий(див. діаграму), то символи складаються в лексему, інакше ці лексеми визначаються як ключові слова, константи, зміні та операції. На цьому етапі можливо отримати помилку та може відбуватися перехід на новий рядок. Усі лексеми записуються у відповідні таблиці: констант, змінних та лексем.

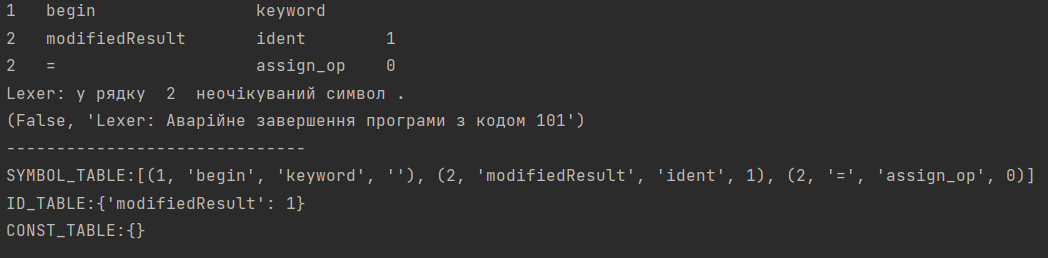
Якщо лексичний аналізатор добирається до фінального стану, то він анулює поточний стан та зчитує непередбачуваний символ (other) ще раз. Аналізатор працює до останнього символу файлу або ж до першої лексичної помилки : при використані символів, що немає використовуються в програмі – помилка 101 – або помилка 102 – невдало описана експоненціальна форма нецілого числа.

Важливо зазначити, що лексичний аналізатор не аналізує синтаксис та семантику, тож відповідно ці помилки ігноруються.

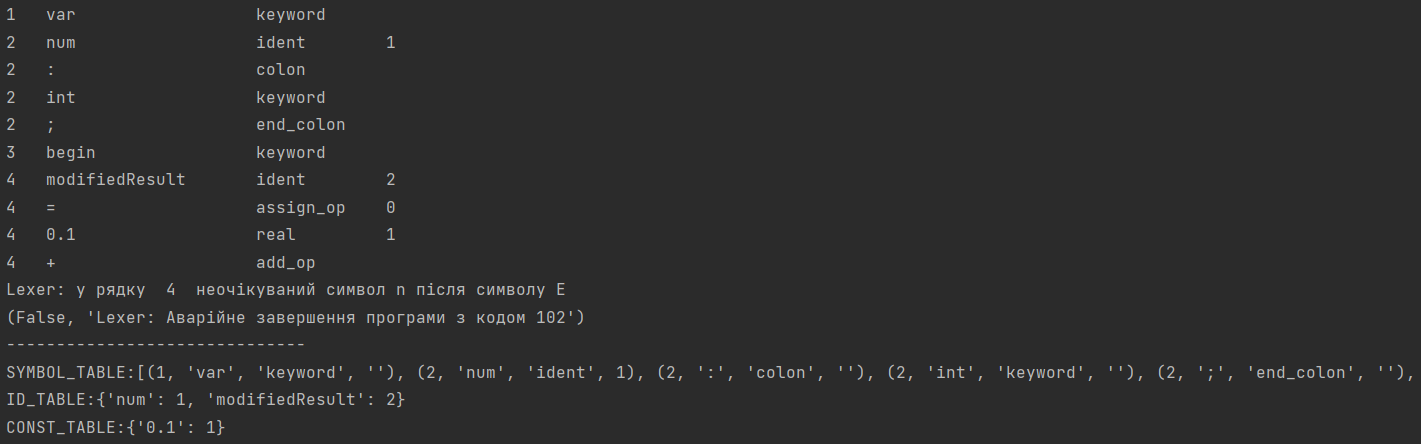
**Тестування базового прикладу**

program modifiedFactorial  
var  
n , x , result :int;  
modifiedResult :real;  
repeat :boolean  
begin  
toRead: read(n);  
if n <= 0 then goto toRead;  
result = 1;  
for (x = 1; x < n ; x + 1)  
begin  
result = result \* (x+1);  
end;  
write(result, nl);  
modifiedResult = ( result / 10^(n+1) ) - 1.1E2 + 5.E-3;  
write(modifiedResult, tb);  
If 100 >= modifiedResult then goto toRead;  
read(repeat);  
if repeat == true then goto toRead;  
endlessLoop :repeat = false;  
If repeat != false then goto endlessLoop;  
repeat = modifiedResult >= 50;  
write(!repeat);  
end

**Тестові приклади з помилками**

begin  
modifiedResult = .1  
end

Виводиться помилка 101, оскільки лексема не може починатися з точки.

var  
num : int;  
begin  
modifiedResult = 0.1 + 5.Enum;  
end

Виводиться помилка 102, бо експоненціальна форма числа з плаваючою точкою введена невірно. Експонента меже містити тільки цілі додатні чи від’ємні числа.

**Висновки**

В ході виконання цієї лабораторної роботи було створено лексичний аналізатор, що перевіряє коректність слів, символьних сполучень та робить висновки щодо кожної лексеми, що зустрічається у визначеному файлі, та записує їх у відповідні таблиці. Надалі цей аналізатор буде використано для створення мови програмування.