**Трансляцiя у ПОЛIЗ та виконання програм з інструкціями**

**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 | bool

DoSection = begin StatementList end

StatementList = Statement { ‘;’ Statement }

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

Assign = Ident ’=’ Expression

Expression = [‘!’] ArthmExpression | Expression RelOp ArthmExpression |

Expression LogicOp ArthmExpression

ArthmExpression = [Sign] Term | ArthmExpression ‘+’ Term |

ArthmExpression ‘-’ Term

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

Factor = Element | Element ‘^’ Factor

Element = Ident | Const | ’(‘Expression’)’ | BoolConst

in = read ’(’ identtList ’)’

Out = write ’(’ identtList ’)’

ForStatement = for indExpr DoSection

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

Const = intNumb | RealNumb

Conditional = if Expression ‘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’

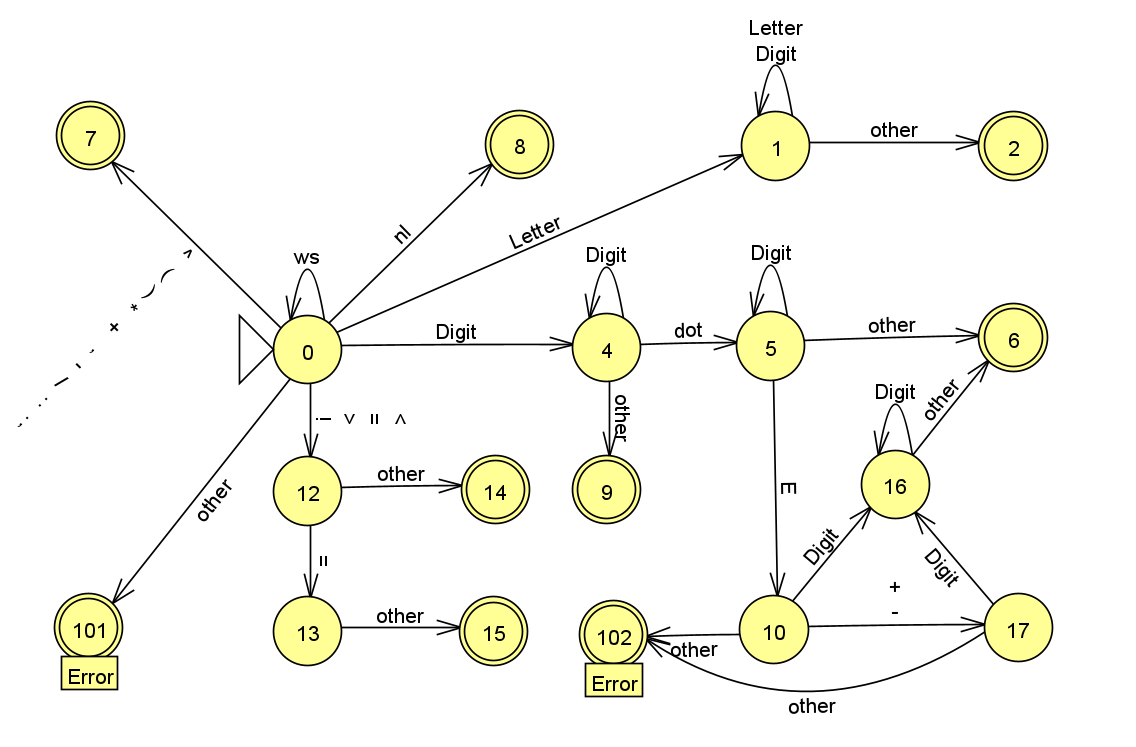
BoolConst = ‘true’ | ‘false’

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

LogicOp = ‘or’ | ‘and’

**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 | int | keyword | символ int |
| 10 | boolean | keyword | символ boolean |
| 11 | read | keyword | символ read |
| 12 | write | keyword | символ write |
| 13 | for | keyword | символ for |
| 14 | if | keyword | символ if |
| 15 | then | keyword | символ then |
| 16 | goto | keyword | символ goto |
| 17 | = | assign\_op | символ = |
| 18 | + | add\_op | символ + |
| 19 | - | add\_op | символ - |
| 20 | \* | mult\_op | символ \* |
| 21 | / | mult\_op | символ / |
| 22 | ^ | pow\_op | символ ^ |
| 23 | < | rel\_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 | ( | par\_op | символ ( |
| 31 | ) | par\_op | символ ) |
| 32 | . | dot | символ . |
| 33 | , | coma | символ , |
| 34 | : | colon | символ : |
| 35 | ; | end\_colon | символ ; |
| 36 | or | logic\_op | символ or |
| 37 | and | logic\_op | символ and |

**Діаграма станів**

**Опис програмної реалізації лексичного аналізатора**

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.

**Опис програмної реалізації синтаксичне аналізатора**

Синтаксичний аналізатор складається з 4 рівнів:

1. Базовий рівень з функціями аналізу токену, отримання символу, обробки помилки, обробки виразів, констант та змінних
2. Рівень аналізу булевих виразів
3. Рівень декларації секції з листів змінних
4. Рівень високорівневих функції, що користуються вище наведеними. Серед них функція аналізу циклу, умовного твердження, присвоєння, вводу/виведення.

На вхід синтаксичний аналізатор бере таблицю символів, що створює лексичний аналізатор. Далі аналізатор розпочинає проходити по кожному символу в таблиці символів і звіряє їх згідно граматики мови. Розглянемо, те як розпочинається програма.

Program = program ProgName DeclSection DoSection

Аналіз мови розпочинається з перевірки наявності ключового слова program та після нього нетерміналу ProgName, що містить лише зміну. Далі перевіряється наявність нетерміналів DeclSection та DoSection, у яких всередені інші нетермінали та термінали згідно граматики мови.

При відсутності необхідного символу виводиться помилка та дані, такі як рядок в якому вона виникла, символ чи токен помилки та очікуваний символ чи токен.

Граматика мови трішки змінився порівняно з попередньою лабораторною роботою задля коректної роботи з транслятором й інтерпретатором постфіксного коду.

Добавлена таблиця з мітками, що можна розмістити перед Statement (див. граматику мови). Ці мітки створені для роботи з умовними розгалуженнями. При цьому мітки видаляються з таблиці змінних. Створено відповідну помилку, що виникає при повторному використані мітки.

Декларативний блок зі змінними присвоює тип зміним згідно позначеного та присвоює значення зміним. Це 0, 0.0 та False.

**Опис транслятору постфіксного коду**

Транслятор постфіксного коду інтегрований в синтаксичний аналізатор. Оскільки синтаксичний аналізатор зчитує лексеми рекурсивно, то при зчитувані лексеми додаються до листа постфіксного коду. У результаті маємо постфіксний код готовий для інтерпретації

Для завершення роботи умовного блоку в операторі розгалуження та в інших подібних конструкціях було добавлено символ ‘end\_expression’ для транслятору, який пізніше реалізує інтерпретатор, просто вивільняючи зайву лексему. Для унарного мінусу застосовується символ ‘&’.

**Опис інтерпретатору постфiксного коду**

Спершу інтерпретатор переміщує лексеми з листа постфіксного коду в стек. Якщо стек йдуть лексеми, які є змінними та константами, то вони просто там зберігаються. Якщо зустрічаються унарні та бінарні операції, то вони реалізуються з першим верхнім елементом стеку та з першими двома верхніми елементами стеку відповідно.

Бінарні операції – це додавання, віднімання, множення, ділення, піднесення до ступеня та логічні порівняння. Унарні операції – це логічне ‘ні’ та унарний мінус.

Змінні при присвоєні автоматично парсять значення відповідно типу. Тобто, якщо int зміні присвоїти значення, наприклад 0.5, то ця змінна міститиме значення 0.

Розроблені помилки при ділені на 0 та використані незадекларованих зміних.

**Специфікація усіх конструкцій, на які розширюється транслятор**

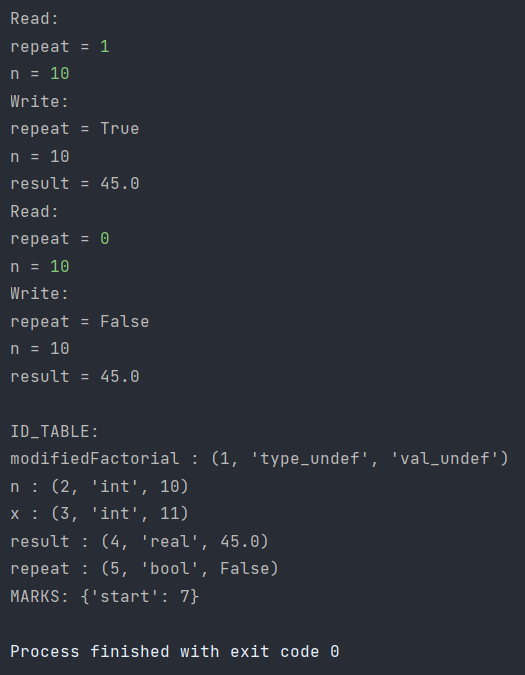
Транслятор розширюється на конструкції умовного розгалуження, циклу та вводу/виводу. Розглянемо детальніше кожен з них.

Умовне розгалуження реалізоване через скачок на відповідну відмітку, якщо умовний блок відповідає істинності інакше умовне розгалуження пропускається. Відмітка може знаходитися до та після умовного оператора. Вона має існувати та не бути зміною.

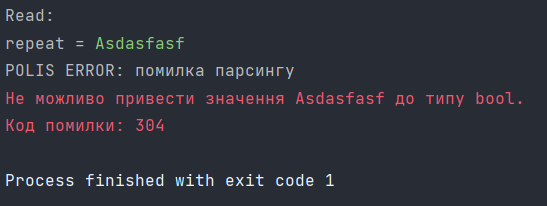
Цикл спершу присвоює певній зміні Х початкове значення. [мітка] Далі відбувається перевірка чи якесь твердження правильне. Якщо твердження є неістинним, то тоді цикл завершується. Інакше розпочинається тіло циклу. Після завершення тіла циклу відбувається невидиме присвоювання змінній Х значення виразу нового кроку. Вертаємося до мітки [мітка].

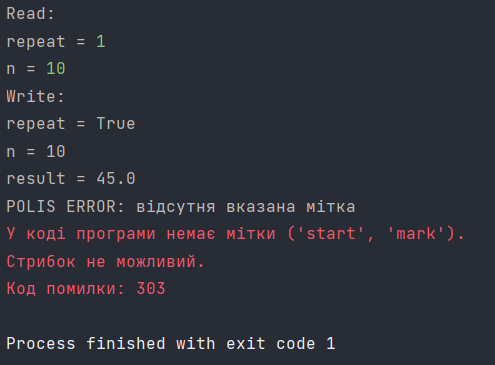
Конструкції вводу/виводу використовують лист зі змінними для відповідної операції, інакше виникне помилка. Відповідним зміним присвоюються/виводяться відповідні значення.

**Тестування базового прикладу**

program modifiedFactorial  
var  
 n,x : int;  
 result : real;  
 repeat: bool  
begin  
 start: read(n, repeat);  
 result = 0;  
 for (x = 1; x < n ; x + 1)  
 begin  
 for (x = 1; x < n ; x + 1)  
 begin  
 result = result + x  
 end  
 end;  
 write(result, n, repeat);  
 if (result > 100) or repeat then goto start  
end

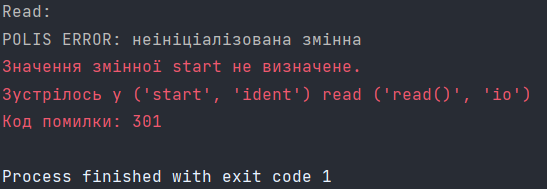
**Тестові приклади з помилками**

Помилка 304 парсингу рядку до певного типу.

program modifiedFactorial  
var  
 n,x : int;  
 result : real;  
 repeat: bool  
begin  
 asdasdwfasfasfa: read(n, repeat);  
 result = 0;  
 for (x = 1; x < n ; x + 1)  
 begin  
 for (x = 1; x < n ; x + 1)  
 begin  
 result = result + x  
 end  
 end;  
 write(result, n, repeat);  
 if (result > 100) or repeat then goto start  
end

Оскільки немає в коді відмітки start, то немає куди скакати і виводиться помилка 303

program modifiedFactorial  
var  
 n,x : int;  
 result : real;  
 repeat: bool  
begin  
 start: read(n, start);  
 result = 0;  
 for (x = 1; x < n ; x + 1)  
 begin  
 for (x = 1; x < n ; x + 1)  
 begin  
 result = result + x  
 end  
 end;  
 write(result, n, repeat);  
 if (result > 100) or repeat then goto start  
end

Помилка 301. Не можливо присвоїти/вивести мітці значення, тож вибиває помилка, що там має бути зміна.

**Висновки**

В ході виконання цієї лабораторної роботи було дороблено транслятор та інтерпретатор постфіксного коду так, що тепер він працює з програмними інструкціями типу if, for, write та read. Тобто в програмній мові реалізований ввід/вивід інформації, оператор розгалуження та цикл. Можна вважати, що мова програмування створена.