Лексичний аналiз методом дiаграми станiв

Варіант № 12

Арифметика: цілі та дійсні числа, основні чотири арифметичні операції (додавання, віднімання, ділення та множення), піднесення до степеня(правоасоціативна операція), дужки

Особливості: унарний мінус

Інструкція повторення: for <ід>=<вираз> by <вираз> while<відношення> do <оператор>

Інструкція розгалуження: if <відношення> {<список операторів>}

Повна граматика мови KK

Program = start ProgName ‘;’ StatementList

ProgName = Ident

*Алфавіт*

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’

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

SpecSsign = ’.’ | ’,’ | ’:’ | ’;’ | ’(’ | ’)’| ’=’ | ’+’ | ’-’ | ’\*’ | ’/’ | ‘^’ | ’<’ | ’>’| WhiteSpace | EndOfLine

WhiteSpace = ’ ’ | ’\t’

EndOfLine = ’\n’ | ’\r’ | ’\r\n’ | ’\n\r’

*Символи*

SpecSymbols = ArithOp | RelOp | BracketsOp | AssignOp | Punct

ArithOp = AddOp | MultOp | NeltOp

AddOp = ’-’

MultOp = ’\*’ | ’/’

NeltOp = ‘^’

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

BracketsOp = ’(’ | ’)’

AssignOp = ’=’

Punct = ’.’ | ’,’ | ’:’ | ’;’

*Індифікатор*

Ident = Letter {Letter | Digit }

*Конастанти*

Const = IntNumb | RealNumb | BoolConst

IntNumb = [Sign] UnsignedInt

RealNumb = [Sign] UnsignedReal

Sign = ’-’

UnsignedInt = Digit {Digit}

UnsignedReal = ’.’ UnsignedInt| UnsignedInt ’.’ | UnsignedInt ’.’ UnsignedInt

BoolConst = true | false

*Ключові слова*

KeyWords = start | read | print | for | by | while | do | if | end

*Синтаксис*

Expression = ArithmExpression | BoolExpr

BoolExpr = Expression RelOp Expression | true | false

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

Term = Factor | Term ’\*’ Factor | Term ’/’ Factor | ‘(‘ Tern ‘^’ Factor ‘)’

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

*Арифметичні операції*

AddOp = ’+’| ’-’

MultOp = ’\*’ | ’/’

NeltOp = ‘^’

*Оголошення*

IdenttList = Ident {’,’ Ident}

*Інструкції*

DoSection = StatementList

StatementList = Statement {’;’ Statement }

Statement = Assign | Inp | Out | ForStatement | IfStatment

*Присвоєення*

Assign = Ident ’=’ Expression

*Введення*

Inp = read ’(’ IdenttList ’)’.

*Виведення*

Out = print ’(’ IdenttList ’)’.

*Інструкція повторення*

ForStatement = for ‘(‘ IndExpr1 ‘)’ by ‘(‘ IndExpr2 ‘)’ while ’(‘ BoolExpr ‘)’ do ‘{‘ DoBlock ‘}’

IndExpr1 = Ident ’=’ ArithmExpression1

IndExpr2 = ArithmExpression2

DoBlock = ’{’ Statement ’}’ | ’{’ StatementList ’}’

*Умовний оператор*

IfStatement = if ‘(’ Relation ‘)’ ‘{‘ DoBlock ‘}’

Relation = BoolExpr

Таблиця лексем

|  |  |  |  |
| --- | --- | --- | --- |
| Код | Приклади лексем | Токен | Неформальний опис |
| 1 | а, xyz1, data3 | іdent | ідентифікатор |
| 2 | 123, 0, 521 | іntnum | ціле без знака |
| 3 | 34.76 | realnum | дійсне без знака |
| 4 | -6.7 | realnum | дійсне зі знаком |
| 5 | true | boolval | логічне значення |
| 6 | start | keyword | символ start |
| 7 | read | keyword | символ read |
| 8 | print | keyword | символ print |
| 9 | for | keyword | символ for |
| 10 | by | keyword | символ by |
| 14 | do | keyword | символ do |
| 16 | while | keyword | символ while |
| 17 | if | keyword | символ if |
| 18 | end | keyword | символ end |
| 19 | false | boolval | символ false |
| 20 | = | assіgn\_op | символ = |
| 21 | + | add\_op | символ + |
| 22 | - | add\_op | символ - |
| 23 | \* | mult\_op | символ \* |
| 24 | / | mult\_op | символ / |
| 25 | ^ | nelt\_op | символ ^ |
| 26 | < | rel\_op | символ < |
| 27 | <= | rel\_op | символ <= |
| 28 | == | rel\_op | символ === |
| 29 | > | rel\_op | символ > |
| 30 | >= | rel\_op | символ >= |
| 31 | != | rel\_op | символ !== |
| 32 | ( | par\_op | символ ( |
| 33 | ) | par\_op | символ ) |
| 34 | { | par\_op | символ { |
| 35 | } | par\_op | символ } |
| 36 | . | punct | символ . |
| 37 | , | punct | символ , |
| 38 | : | punct | символ : |
| 39 | ; | punct | символ ; |
| 40 | \32,\t | ws | пробільні символи |
| 41 | \n, \r, \r\n, \n\r | ls | кінець рядка |

1. Дiаграма станiв для розпiзнавання лексем мови

1) Класи символiв

Letter та Digit - для символiв, позначених однойменними нетермiналами в граматицi

Brackets – для символів ‘(‘, ‘)’, ‘{‘, ‘}’

MultOp – для символів ‘\*’, ‘/’

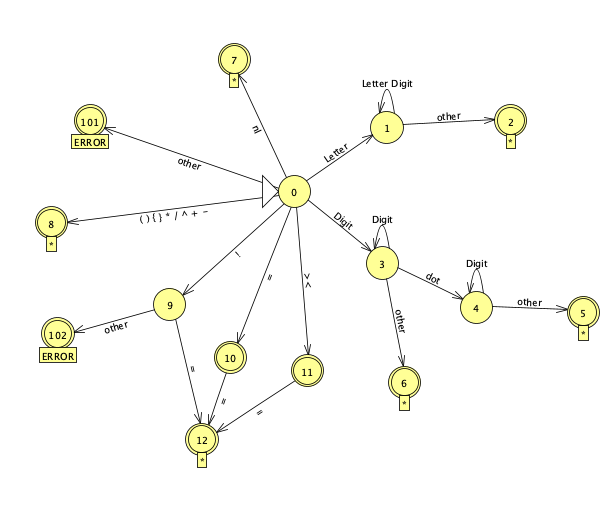
NeltOp – для символів ‘\*’, ‘/’

AddOp – для символів ‘+’, ‘-‘

ws - для пробiльних символiв

nl - для символу нового рядка

other – для символiв, що не належать до поточної лексеми.

2) Графiчно в JFLAP

3) Запис у символьнiй формi

Дiаграма станiв як детермiнований скiнченний автомат:

M = (Q, Σ, δ, q0, F)

Множина станiв:

Q = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 101, 102}

Алфавiт:

Σ = {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, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., ,, ;, !, =, ∗, +, −, /, ^, (, ), {, }, <, >, \n, \t}

Функцiя переходiв:

δ (0, 'ws') = 0,

δ (0, 'nl') = 7,

δ (0, 'AddOp') = 8, δ (0, 'MultOp') = 8, δ (0, '^') = 8, δ (0, ',') = 8, δ (0, ';') = 8,

δ (0, 'Brackets') = 8,

δ(0, 'Letter') = 1, δ (1, 'Letter') = 1, δ(1, 'Digit') = 1, δ (1, 'other') = 2,

δ (0, 'Digit') = 3, δ (3, 'Digit') = 3, δ (3, 'other') = 6, δ (3, 'dot') = 4,

δ (4, 'Digit') = 4, δ (4, 'other') = 5,

δ (0, '=') = 10, δ (0, '!') = 9, δ (0, '<') = 11, δ (0, '>') = 11, δ (9, '=') = 12,

δ (11, '=') = 12, δ (10, '=') = 12, δ (0, 'other') = 101, δ (9, 'other') = 102

Стартовий стан: q0 = 0

Множина заключних станiв: F = {2, 5, 6, 7, 8, 10, 11, 12, 101, 102}

Заключнi стани, що потребують додаткової обробки:

Fstar = {2, 5, 6, 7, 8, 12}

ERROR = {101, 102}

2. Cемантичнi процедури для кожного заключного стану дiаграми станiв

|  |  |  |
| --- | --- | --- |
| Стан | Токен | Семантичні процедури |
| 2 | Id, keyword | · Визначити: id чи keyword?  · Якщо id – обробити таблицю iдентифiкаторiв  · Занести лексему в таблицю розбору  · Повернути необроблений символ у вхiдний потiк  · Перейти у стартовий стан |
| 7 | real | · Обробити таблицю констант  · Занести лексему в таблицю розбору  · Повернути необроблений символ у вхiдний потiк  · Перейти у стартовий стан |
| 8 | int | · Обробити таблицю констант  · Занести лексему в таблицю розбору  · Повернути необроблений символ у вхiдний потiк  · Перейти у стартовий стан |
| 13 | assign\_op | · Занести лексему в таблицю розбору  · Перейти у стартовий стан |
| 9 | nl | · Збiльшити лiчильник рядкiв на 1  · Перейти у стартовий стан |
| 10 | Add\_op, mult\_op,  nelt\_op, brackets | · Занести лексему в таблицю розбору  · Перейти у стартовий стан |
| 14 | Rel\_op | · Занести лексему в таблицю розбору  · Перейти у стартовий стан |
| 15 | Rel\_op | · Занести лексему в таблицю розбору  · Перейти у стартовий стан |

3. Надати лексичному аналiзаторовi iнформацiю

1) Функцiя (метод) для вiднесення символiв до одного з класiв:

def classOfChar(char):  
 if char in "!=<>^,;E":  
 res = char  
 elif char in '.':  
 res = "dot"  
 elif char in "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ":  
 res = "Letter"  
 elif char in "0123456789":  
 res = "Digit"  
 elif char in " \t":  
 res = "ws"  
 elif char in "\n":  
 res = "nl"  
 elif char in "(){}":  
 res = "Brackets"  
 elif char in "+-":  
 res = "AddOp"  
 elif char in "\*/":  
 res = "MultOp"  
 elif char in "^":  
 res = "NeltOp"  
 else:  
 res = 'символ не належить алфавiту'  
 return res

2) Структури даних для зберiгання iнформацiї про дiаграму станiв

Функцiя переходiв δ представлена у формi словника:

stf = {

(0, 'ws'): 0,

(0, 'nl'): 7,

(0, 'other'): 101,

(0, 'AddOp'): 8, (0, 'MultOp'): 8, (0, '^'): 8, (0, ','): 8, (0, ';'): 8, (0, 'Brackets'): 8,

(0, 'Letter'): 1, (1, 'Letter'): 1, (1, 'Digit'): 1, (1, 'other'): 2,

(0, 'Digit'): 3, (3, 'Digit'): 3, (3, 'other'): 6, (3, 'dot'): 4, (4, 'Digit'): 4, (4, 'other'): 5,

(0, '='): 10,

(0, '!'): 9, (0, '<'): 11, (0, '>'): 11, (9, '='): 12, (11, '='): 12, (10, '='): 12,

(9, 'other'): 102

}3

3) Структуру даних для зберiгання iнформацiї про таблицю символiв мови

Таблиця символiв мови реалiзована у формi двох словникiв:

tableOfLanguageTokens = {

'start': 'keyword',

'end': 'keyword',

'read': 'keyword',

'print': 'keyword',

'for': 'keyword',

'by': 'keyword',

'while': 'keyword',

'do': 'keyword',

'if': 'keyword',

'=': 'assign\_op',

'.': 'dot',

'-': 'add\_op',

'+': 'add\_op',

'\*': 'mult\_op',

'/': 'mult\_op',

'^': 'nelt\_op',

'<': 'rel\_op',

'<=': 'rel\_op',

'>=': 'rel\_op',

'>': 'rel\_op',

'==': 'rel\_op',

'!=': 'rel\_op',

'(': 'brackets\_op',

')': 'brackets\_op',

'{': 'brackets\_op',

'}': 'brackets\_op',

',': 'punct',

';': 'punct',

' ': 'ws', '\t': 'ws', '\n': 'nl'

}

tableIdentFloatInt = {2: 'ident', 5: 'real', 6: 'integer'}

4. Вихiд лексичного аналiзатора (результат роботи).

1) Повiдомлення про успiшнiсть/неуспiшнiсть лексичного аналiзу вхiдної програми, дiагностичне повiдомлення:

print('\nLexer: Лексичний аналiз завершено успішно!')

def fail():  
 if state == 101:  
 print('\nLexer ERROR:\n\t[{0}]: Неочікуваний символ \'{1}\'.'  
 .format(numLine, char))  
 exit(101)  
 if state == 102:  
 print('\nLexer ERROR:\n\t[{0}]: Неочікуваний символ \'{1}\'.'  
 '\n\tОчікувався - \'=\'.'.format(numLine, char))  
 exit(102)  
 if state == 103:  
 print('\nLexer ERROR:\n\t[{0}]: Неправильний формат числа.'  
 '\n\tОчікувалася цифра після символа \'.\'.'.format(numLine, char))  
 exit(103)  
 if state == 104:  
 print('\nLexer ERROR:\n\t[{0}]: Неправильний формат числа.'  
 '\n\tОчікувалася цифра після символа \'e\'.'.format(numLine, char))  
 exit(104)  
 if state == 105:  
 print('\nLexer ERROR:\n\t[{0}]: Неправильний формат числа.'  
 '\n\tОчікувалася цифра після знаку.'.format(numLine, char))  
 exit(105)

2) Таблиця розбору (таблиця символiв програми)

Таблиця розбору реалiзована як словник tableOfSymb у форматi:

**{ n\_rec : (num\_line, lexeme, token, idxIdConst) }** де:

n\_rec – номер запису в таблицi символiв програми;

num\_line – номер рядка вхiдної програми;

lexeme – лексема;

token – токен лексеми;

idxIdConst – iндекс iдентифiкатора або константи у таблицi iдентифiкаторiв та констант вiдповiдно; для iнших лексем – порожнiй рядок.

3) Таблицi iдентифiкаторiв, констант, мiток (якщо потрiбнi):

**{ Id : idxId) }** де:

Id – iдентифiкатор (лексема);

idxId – iндекс iдентифiкатора у таблицi iдентифiкаторiв.

**{ Const : idxConst}** де:

Const –константа (лексема);

idxConst – iндекс константи у таблицi констант.

5. Програмна реалiзацiя сканера (лексичного аналiзатора)

1) Функцiї/методи для читання вхiдного потоку та повернення у вхiдний

потiк:

def nextChar():  
 global numChar  
 numChar += 1  
 return sourceCode[numChar]

2) Функцiї/методи для переходу у наступний стан:

def nextState(state, classCh):  
 try:  
 return stf[(state, classCh)]  
 except KeyError:  
 return stf[(state, 'other')]

3) Функцiї/методи для реалiзацiї семантичних процедур:

def processing():

global state, lexeme, numLine, numChar

if state in (2, 5, 6):

token = getToken(state, lexeme)

if token != 'keyword':

index = indexIdConst(state, lexeme, token)

print('{0:<3d} {1:<10s} {2:<10s} {3:<2d} '.format(numLine, lexeme, token, index))

tableOfSymb[len(tableOfSymb) + 1] = (numLine, lexeme, token, index)

else: # якщо keyword

print('{0:<3d} {1:<10s} {2:<10s} '.format(numLine, lexeme, token))

tableOfSymb[len(tableOfSymb) + 1] = (numLine, lexeme, token, '')

lexeme = ''

numChar = putCharBack(numChar)

state = initState

if state == 7:

print('-----------------------------------------------')

numLine += 1

state = initState

if state == 11:

lexeme += char

if nextChar() == '=':

classCh = classOfChar('=')

state = nextState(state, classCh)

else:

numChar -= 1

token = getToken(state, lexeme)

print('{0:<3d} {1:<10s} {2:<10s} '.format(numLine, lexeme, token))

tableOfSymb[len(tableOfSymb) + 1] = (numLine, lexeme, token, '')

lexeme = ''

state = initState

if state == 10:

lexeme += char

if nextChar() == '=':

classCh = classOfChar('=')

state = nextState(state, classCh)

else:

numChar -= 1

token = getToken(state, lexeme)

print('{0:<3d} {1:<10s} {2:<10s} '.format(numLine, lexeme, token))

tableOfSymb[len(tableOfSymb) + 1] = (numLine, lexeme, token, '')

lexeme = ''

state = initState

if state == 8:

lexeme += char

token = getToken(state, lexeme)

print('{0:<3d} {1:<10s} {2:<10s} '.format(numLine, lexeme, token))

tableOfSymb[len(tableOfSymb) + 1] = (numLine, lexeme, token, '')

lexeme = ''

state = initState

if state == 12:

lexeme += '='

token = getToken(state, lexeme)

print('{0:<3d} {1:<10s} {2:<10s} '.format(numLine, lexeme, token))

tableOfSymb[len(tableOfSymb) + 1] = (numLine, lexeme, token, '')

lexeme = ''

state = initState

if state in Ferror:

fail()

**6. Тестування**

1. Частина базового прикладу програми без помилок:

start ProgramName

k = 54

d = 3.0 ^ (5.0 / 10.0);

for ( I = 1 ) by ( I = I + 1) while (I < 10) do {

print(i);

}

I = 1;

If ( I < 10 ) {

print (i);

}

end

Результат:

Lexer: Лексичний аналiз завершено успішно!

Lexer: Таблиця символів: {1: (1, 'start', 'keyword', ''), 2: (1, 'ProgramName', 'ident', 1), 3: (2, 'k', 'ident', 2), 4: (2, '=', 'assign\_op', ''), 5: (2, '54', 'integer', 1), 6: (3, 'd', 'ident', 3), 7: (3, '=', 'assign\_op', ''), 8: (3, '3.0', 'real', 2), 9: (3, '^', 'nelt\_op', ''), 10: (3, '(', 'brackets\_op', ''), 11: (3, '5.0', 'real', 3), 12: (3, '/', 'mult\_op', ''), 13: (3, '10.0', 'real', 4), 14: (3, ')', 'brackets\_op', ''), 15: (3, ';', 'punct', ''), 16: (5, 'for', 'keyword', ''), 17: (5, '(', 'brackets\_op', ''), 18: (5, 'I', 'ident', 4), 19: (5, '=', 'assign\_op', ''), 20: (5, '1', 'integer', 5), 21: (5, ')', 'brackets\_op', ''), 22: (5, 'by', 'keyword', ''), 23: (5, '(', 'brackets\_op', ''), 24: (5, 'I', 'ident', 4), 25: (5, '=', 'assign\_op', ''), 26: (5, 'I', 'ident', 4), 27: (5, '+', 'add\_op', ''), 28: (5, '1', 'integer', 6), 29: (5, ')', 'brackets\_op', ''), 30: (5, 'while', 'keyword', ''), 31: (5, '(', 'brackets\_op', ''), 32: (5, 'I', 'ident', 4), 33: (5, '<', 'rel\_op', ''), 34: (5, '10', 'integer', 6), 35: (5, ')', 'brackets\_op', ''), 36: (5, 'do', 'keyword', ''), 37: (5, '{', 'brackets\_op', ''), 38: (6, 'print', 'keyword', ''), 39: (6, '(', 'brackets\_op', ''), 40: (6, 'i', 'ident', 5), 41: (6, ')', 'brackets\_op', ''), 42: (6, ';', 'punct', ''), 43: (7, '}', 'brackets\_op', ''), 44: (9, 'I', 'ident', 4), 45: (9, '=', 'assign\_op', ''), 46: (9, '1', 'integer', 7), 47: (9, ';', 'punct', ''), 48: (11, 'If', 'ident', 6), 49: (11, '(', 'brackets\_op', ''), 50: (11, 'I', 'ident', 4), 51: (11, '<', 'rel\_op', ''), 52: (11, '10', 'integer', 7), 53: (11, ')', 'brackets\_op', ''), 54: (11, '{', 'brackets\_op', ''), 55: (12, 'print', 'keyword', ''), 56: (12, '(', 'brackets\_op', ''), 57: (12, 'i', 'ident', 5), 58: (12, ')', 'brackets\_op', ''), 59: (12, ';', 'punct', ''), 60: (13, '}', 'brackets\_op', ''), 61: (14, 'end', 'keyword', '')}

Lexer: Таблиця змінних: {'ProgramName': (1, 'type\_undef', 'val\_undef'), 'k': (2, 'type\_undef', 'val\_undef'), 'd': (3, 'type\_undef', 'val\_undef'), 'I': (4, 'type\_undef', 'val\_undef'), 'i': (5, 'type\_undef', 'val\_undef'), 'If': (6, 'type\_undef', 'val\_undef')}

Lexer: Таблиця констант: {'54': (1, 'integer', 54), '3.0': (2, 'real', 3.0), '5.0': (3, 'real', 5.0), '10.0': (4, 'real', 10.0), '1': (7, 'integer', 1), '10': (7, 'integer', 10)}

Лексичний аналiзатор правильно розпiзнав усi лексеми та визначив їх токени, зокрема ProgramName розпiзнано як iдентифiкатор. Синтаксичну правильнiсть вiн не перевiряє, тому з точки зору лексики наведена програма коректна i лексер повiдомив про успiшнiсть лексичного розбору.

1. Error 101

start ProgramName

k = $54

d = 3.0 ^ (5.0 / 10.0);

end

1 start keyword

1 ProgramName ident 1

-----------------------------------------------

2 k ident 2

2 = assign\_op

Lexer ERROR:

[2]: Неочікуваний символ '$'.

Lexer: Аварійне завершення програми з кодом 101

1. Error 102

start ProgramName

for ( I = 1 ) by ( I = I + 1) while (I < 10) do {

if ( I !: 4) {

print(i);

}

}

end

1 start keyword

1 ProgramName ident 1

-----------------------------------------------

2 for keyword

2 ( brackets\_op

2 I ident 2

2 = assign\_op

2 1 integer 1

2 ) brackets\_op

2 by keyword

2 ( brackets\_op

2 I ident 2

2 = assign\_op

2 I ident 2

2 + add\_op

2 1 integer 2

2 ) brackets\_op

2 while keyword

2 ( brackets\_op

2 I ident 2

2 < rel\_op

2 10 integer 2

2 ) brackets\_op

2 do keyword

2 { brackets\_op

-----------------------------------------------

3 if keyword

3 ( brackets\_op

3 I ident 2

Lexer ERROR:

[3]: Неочікуваний символ ':'.

Очікувався - '='.

Lexer: Аварійне завершення програми з кодом 102

У цих випадках аналізатор сам інформує про помилку і вказує на причину.

**Висновок**: під час виконання цієї лабораторної работи було освоєно основи розробки лексичного аналізатору, набуто навички з його розробки та використано їх на практиці. В результаті виконання роботи було розроблено лексичний аналізатор для мови програмування KK та протестовано його.