**Лексичний аналiз методом дiаграми станiв**

Варіант № 7

Арифметика: цілі та дійсні числа, основні чотири арифметичні операції (додавання,

віднімання, ділення та множення), піднесення до степеня (правоасоціативна операція),

дужки

Особливості: експоненційна форма дійсного числа

Інструкція повторення: for (<ід>=<вираз>; <відношення>; <ід>=<вираз>){<cписок

операторів>}

Інструкція розгалуження: if <відношення> then goto <мітка>

**Короткий опис мови**

Мова програмування FuriousPeach.

Програма складається iз списку iнструкцiй присвоювання. Мова пiдтримує роботу з дiйсними та цiлими виразами. Дiйснi константи представленi у форматi з плаваючою крапкою. Вирази можуть мiстити чотири основнi (лiвоасоцiативнi) арифметичнi операцiї та круглi дужки.

**Повна граматика мови FuriousPeach**

Program = program ProgName DeclSection DoSection

ProgName = Ident

Ident = Letter {Letter | Digit }

DeclSection = DeclarList

DeclarList = Declaration { Declaration }

Declaration = Type ’ ’ IdenttList

IdenttList = Ident {’,’ Ident}

Type = int | real | bool

DoSection = StatementList

StatementList = Statement {’;’ Statement }

Statement = Assign | Input | Print | ForStatement

Assign = Ident ’=’ (Expression | BoolExpr)

Expression = ArithmExpression | BoolExpr

BoolExpr = ArithmExpression Rel ArithmExpression

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

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

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

Input = input ’(’ IdenttList ’)’

Print = print ’(’ IdenttList ’)’

ForStatement = for ’(’ IndExpr ’)’ ’{’DoBlock ’}’

IndExpr = Ident ’=’ ArithmExpression1 ’;’

BoolExpr ’;’ ArithmExpression2

ArithmExpression1 = ArithmExpression

ArithmExpression2 = Ident Add Step

DoBlock = Statement | StatementList

IfStatement = if BoolExpr ’then ’ ’goto’ Ident1 | ’else’ ’goto’ Ident2

Ident1 = Letter {Letter | Digit }

Ident2 = Letter {Letter | Digit }

Const = IntNumb | RealNumb | BoolConst

IntNumb = [Sign] UnsignedInt

RealNumb = [Sign] UnsignedReal

Sign = ’+’ | ’-’

UnsignedInt = Digit {Digit}

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

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’

BoolConst = true | false

Rel = ’==’ | ’<= ’ | ’<’ | ’>’ | ’>=’ | ’<>’

**Таблиця лексем мови FuriousPeach**

|  |  |  |  |
| --- | --- | --- | --- |
| Код | Приклади лексем | Токен | Неформальний опис |
| 1 | a, x1, z12f | id | iдентифiкатор |
| 2 | 123, 0, 521 | intnum | цiле без знаку |
| 3 | .012, 34.76, 876. | realnum | дiйсне без знаку |
| 4 | true, false | boolval | логiчне значення |
| 5 | program | keyword | символ program |
| 6 | endl | keyword | символ endl |
| 7 | int | keyword | символ int |
| 8 | real | keyword | символ real |
| 9 | bool | keyword | символ bool |
| 10 | input | keyword | символ input |
| 11 | print | keyword | символ print |
| 12 | for | keyword | символ for |
| 13 | if | keyword | символ if |
| 14 | goto | keyword | символ goto |
| 15 | then | keyword | символ then |
| 16 | else | keyword | символ else |
| 17 | = | assign\_op | символ = |
| 18 | + | add\_op | символ + |
| 19 | - | add\_op | символ - |
| 20 | \* | mult\_op | символ \* |
| 21 | / | mult\_op | символ / |

|  |  |  |  |
| --- | --- | --- | --- |
| 22 | div | mult\_op | символ div |
| 23 | < | rel\_op | символ < |
| 24 | <= | rel\_op | символ <= |
| 25 | == | rel\_op | символ == |
| 26 | >= | rel\_op | символ >= |
| 27 | <> | rel\_op | символ <> |
| 28 | ^ | pow\_op | символ ^ |
| 29 | ( | brackets\_op | символ <> |
| 30 | ) | brackets\_op | символ <> |
| 31 | . | punct | символ . |
| 32 | , | punct | символ , |
| 33 | : | punct | символ : |
| 34 | ; | punct | символ ; |
| 35 | { | punct | символ { |
| 36 | } | punct | символ } |
| 37 | \32,\t | ws | пробiльнi символи |
| 38 | \n, \r\n | eol | кiнець рядка |
| 39 | \0 | eof | кiнець файлу |

**Розробка лексичного аналiзатора**

**Програмна реалiзацiя лексичного аналiзатора**

**Дiаграма станiв i таблиця лексем у кодi**

Лексичний аналiзатор реалiзується мовою python. Усi необхiднi таблицi ре-

алiзуються як словники (dictionary) мови python.

**Визначення класу вхiдного символа**

Функцiя classOfChar() повертає клас, до якого належить символ char:

1 def classOfChar(char):

2 if char in '.' :

3 res="dot"

4 elif char in 'abcdefghijklmnopqrstuvwxyz' :

5 res="Letter"

6 elif char in "0123456789" :

7 res="Digit"

8 elif char in " \t" :

9 res="ws"

**Дiаграма станiв i таблиця лексем у кодi**

|  |  |  |
| --- | --- | --- |
| Стан | Токен | Семантичнi процедури |
| 2 | id або 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 |
| 6 | real | Обробити таблицю констант  Занести лексему в таблицю розбору  Повернути необроблений символ у вхiдний потiк Перейти у стартовий стан |
| 9 | int | Обробити таблицю констант  Занести лексему в таблицю розбору  Повернути необроблений символ у вхiдний потiк Перейти у стартовий стан |
| 12 | assign\_op | Занести лексему в таблицю розбору  Перейти у стартовий стан |
| 13 | nl | Збiльшити лiчильник рядкiв на 1  Перейти у стартовий стан |
| 14 | add\_op, mult\_op, par\_op | Занести лексему в таблицю розбору  Перейти у стартовий стан |

Табл. 2. Семантичнi процедури

10 elif char in "\n" :

11 res="nl"

12 elif char in "+-=\*/()" :

13 res=char

14 else: res='символ не належить алфавіту'

15 return res

**Дiаграма станiв у кодi**

Функцiя переходiв *δ* представлена у формi словника (dictionary) stf (state transition function) мови python:

1 # δ - state-transition\_function

2 stf = {(0, 'Letter'): 1, (1, 'Letter'): 1, (1, 'Digit'): 1,

3 (1,'other'): 2, \

4 (0, 'Digit'): 4, (4, 'Digit'): 4, (4, 'dot'): 5, (4, 'other'): 9,

5 (5, 'Digit'): 5, (5, 'other'): 6, \

6 (0, '='): 12, \

7 (11, 'other'): 102, \

8 (0, 'ws'): 0, \

9 (0, 'nl'): 13, \

10 (0, '+'): 14, (0, '-'): 14, (0, '\*'): 14, (0, '/'): 14,

11 (0, '('): 14, (0, ')'): 14, (0, '{'): 14, (0, '}'): 14,

12 (0, '<'): 14, (0, '>'): 14, (0, '<'): 14, (0, '>'): 14,

13 (0, ';'): 14, (0, '^'): 14, \

14 (0, 'other'): 101

15 }

16 initState = 0 # q0 - стартовий стан

17 F={2,6,9,12,13,14,101,102}

18 Fstar={2,6,9} # зірочка

19 Ferror={101,102}# обробка помилок

**Таблиця символiв мови у кодi**

Таблиця символiв мови реалiзована у формi двох словникiв – tableOfLanguageTokens та tableIdentFloatInt. Перший мiстить iнформацiю про токени, що однозначно представленi перелiком лексем, другий – про iдентифiкатори та константи.

1 # Таблиця лексем мови

2 tableOfLanguageTokens = {'program': 'keyword', 'bye': 3 'keyword', 'print': 'keyword', 'input': 'keyword',

4 'if': 'keyword', 'goto': 'keyword', 'for': 'keyword', 5 'then': 'keyword', 'int': 'keyword', 'bool':

6 'keyword', 'real': 'keyword', '=': 'assign\_op', '.': 7 'dot', ' ': 'ws', '\t': 'ws', '\n': 'nl', '-':

8 'add\_op', '+': 'add\_op', '\*': 'mult\_op', '/':

9 'mult\_op', '(': 'brackets\_op', ')': 'brackets\_op',

10 '{': 'punct', '}': 'punct', ';': 'punct', '<':

11 'rel\_op', '>': 'rel\_op', '>=': 'rel\_op', '<=':

12 'rel\_op', '==': 'rel\_op', '^': 'pow\_op' }

13 # Решту токенів визначаємо не за лексемою, а за заключним станом

14 tableIdentFloatInt = {2:'ident', 6:'real', 9:'int'}

**Вихiд лексичного аналiзатора**

**Резюме про успiшнiсть лексичного розбору**

Повiдомлення формуються наступними командами:

1. про успiшнiсть (функцiя lex())

– print(’Lexer: Лексичний аналiз завершено успiшно’)

2. про неуспiшнiсть (функцiя lex())

– print(’Lexer: Аварiйне завершення програми з кодом 0’.format(e))

3. дiагностичнi повiдомлення у станах 101 та 102 вiдповiдно (функцiя fail())

– print(’Lexer: у рядку ’,numLine,’ неочiкуваний символ ’+char)

– print(’Lexer: у рядку ’,numLine,’ очiкувався символ =, а не ’+char)

**Таблиця символiв програми (таблиця розбору)**

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.

**Програмна реалiзацiя основних функцiй**

**Таблиця iдентифiкаторiв**

Таблиця iдентифiкаторiв tableOfId: { Id : idxId) }

де: Id – iдентифiкатор (лексема);

idxId – iндекс iдентифiкатора у таблицi iдентифiкаторiв.

**Таблиця констант**

Таблиця констант tableOfConst: { Const : idxConst}

де: Const –константа (лексема); idxConst – iндекс константи у таблицi констант.

**Таблиця мiток**

Таблиця мiток tableOfLabel: { Label : idxLbl }

де: Label –мiтка (лексема); idxLbl – iндекс мiтки у таблицi мiток.

**Програмна реалiзацiя основних функцiй**

**Функцiя верхнього рiвня**

Функцiя lex(), у циклi, див. рядки 11-21 коду, намагається проаналiзува ти всi символи програми вхiдною мовою (while numChar<lenCode, рядок 11). Зчитування символу, вiднесення його до одного з прийнятих класiв та перехiд у наступний стан здiйснюється у рядках 12 - 14 коду. Коли перехiд здiйснює ться у заключний, стартовий чи iнший стан (див. рядки 15, 17 та 19), здiйсню ється, вiдповiдно, виконання семантичних процедур (функцiя processing()), пiдготовка до пошуку нової лексеми (рядок 18), або приєднання прочитаного символу до поточної лексеми (рядок 20).

Якщо при розборi буде досягнуто заключного стану iз множини *FERROR*, то фукнцiєю fail() буде згенеровано виняток, який буде перехоплено та оброблено у блоцi except, рядки 15 - 18.

1 def lex():

2 global state,numLine,char,lexeme,numChar,FSuccess

3 try:

4 while numChar<lenCode:

5 char=nextChar() # прочитати наступний символ

6 classCh=classOfChar(char) # до якого класу належить

7 state=nextState(state,classCh) # обчислити наступний стан

8 if (is\_final(state)): # якщо стан заключний

9 processing() # виконати семантичні процедури

10 # if state in Ferror: # якщо це стан обробки помилки

11 # break # то припинити подальшу обробку

12 elif state==initState:lexeme='' # якщо стан НЕ заключний, а стартовий - нова лексема

13 else: lexeme+=char # якщо стан НЕ закл. і не стартовий - додати символ до лексеми

14 print('Lexer: Лексичний аналіз завершено успішно')

15 except SystemExit as e:

16 # Встановити ознаку неуспішності

17 FSuccess = (False,'Lexer')

18 # Повідомити про факт виявлення помилки

19 print('Lexer: Аварійне завершення програми з кодом {0}'.format(e))

**Функцiї читання символу та його повернення у вхiдний потiк (зірочка)**

Вхiдна програма зберiгається у рядку (String) sourceCode. Тому читання на ступного символа реалiзується функцiєю nextChar() шляхом зчитування сим вола з попередньо iнкрементованим його номером numChar.

def nextChar():

global numChar

numChar+=1

return sourceCode[numChar]

Повернення символу у вхiдний потiк реалiзується функцiєю putCharBack(numChar) шляхом декрементацiї номера символу numChar.

def putCharBack(numChar):

return numChar-1

**Переходи у дiаграмi станiв**

Функцiя nextState(state,classCh) намагається знайти значення у словнику stf значення за ключем (state,classCh). Якщо таке значення знайдено, то воно i є iм’ям наступного стану, якщо ж запису з таким ключем немає – то перхоплюється згенерований системою python виняток KeyError, а перехiд зi стану state здiйснюється по класу символiв other.

def nextState(state,classCh):

try:

return stf[(state,classCh)]

except KeyError:

return stf[(state,’other’)]

**Реалiзацiя семантичних процедур**

1 def processing():

2 global state,lexeme,char,numLine,numChar, tableOfSymb

3 if state==13: # \n

4 numLine+=1

5 state=initState

6 if state in (2,6,9): # keyword, ident, float, int

7 token=getToken(state,lexeme)

8 if token!='keyword': # не keyword

9 index=indexIdConst(state,lexeme)

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

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

12 else: # якщо keyword

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

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

15 lexeme=''

16 numChar=putCharBack(numChar) # зірочка

17 state=initState

18 if state in (12,14): #12: # assign\_op # in (12,14):

19 lexeme+=char

20 token=getToken(state,lexeme)

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

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

23 lexeme=''

24 state=initState

25 if state in Ferror: #(101,102): # ERROR

26 fail()

**Тестування**

**Тест №1.** Програма, всi лексеми якої допустимi у мовi FuriousPeach, проте їх використання суперечить синтаксису мови. У цьому випадку лексичний аналiзатор помилок НЕ ЗНАХОДИТЬ.

Синтаксично помилковi фрагменти такi:

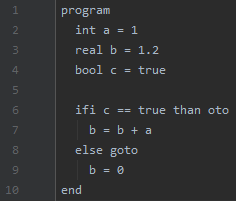
1. Замість if у 6 рядку написано ifi;

2. Замість then у 6 рядку написано than;

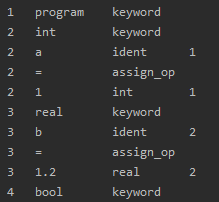
3. Замість goto у 6 рядку написано oto;

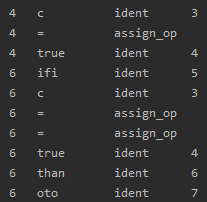
|  |  |  |  |
| --- | --- | --- | --- |
| № | Тип випробування | Очiкуванний результат | Кiлькiсть випробувань |
| 1 | базовий приклад | успiшне виконання, таблицi iдентифiкаторiв,констант, мiток | 1 |
| 2 | вкладенi конструкцiї | успiшне виконання таблицi iдентифiкаторiв, констант, мiток | 1 |
| 3 | наявнiсть недопустимих лексем | повiдомлення про помилку та дiагностика, таблицi iдентифiкаторiв,  констант, мiток | 2 |

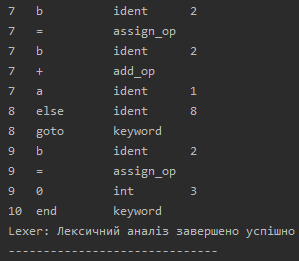
Табл. 3. План тестування



Пiсля розбору лексичний аналiзатор повiдомляє:





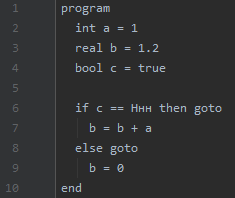


Лексичний аналiзатор правильно розпiзнав усi лексеми та визначив їх токени, зокрема ifi, than, oto розпiзнано як iдентифiкатори. Синтаксичну правильнiсть вiн не перевiряє, тому з точки зору лексики наведена програма коректна i лексер повiдомив про успiшнiсть лексичного розбору.

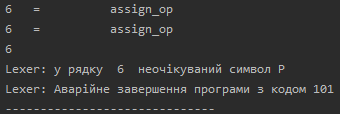
**Тест №2.** У програмі містяться недопустимі лексеми, тоді аналізатор повідомляє про помилку.

Синтаксично помилковий фрагмент такий:

У рядку 6 відбувається перевірка змінної c на значення Ннн, яке не визначене в мові FuriousPeach.



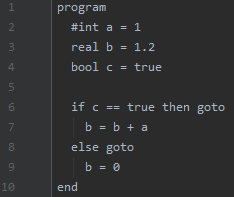
Пiсля розбору лексичний аналiзатор повiдомляє:



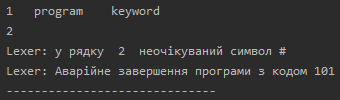
Лексичний аналiзатор розпiзнав невідомий ідентифікатор Ннн як P, оскільки в англійській розкладці символу Н відповідає P, до того ж він не визначений в мові FuriousPeach, аналізатор видав помилку та аварійно завершив програму з кодом 101.

**Тест №3.** У програмі містяться недопустимі символи, тоді аналізатор повідомляє про помилку.

У 2 рядку біля об’явлення змінної a є символ #, який не визначено в мові FuriousPeach.



Пiсля розбору лексичний аналiзатор повiдомляє:



Лексичний аналiзатор розпiзнав невідомий символ #, видав помилку та аварійно завершив програму з кодом 101.

**Висновок***:* Був створений лексичний аналізатор для мови FuriousPeach, розглануті принципи його побудови з відповідними фрагментами коду та проведені тестування, при яких перевірялася робота аналізатора, якщо у коді є невизначені лексеми, символи, або невірно написані ключові слова. Із результатів тестувань можна зробити висновок, що аналізатор працює так, як потрібно.