Міністерство освіти та науки України
Київський національний університет імені Тараса Шевченка

Звіт до лабораторної роботи №1

«Розробка програми-парсера алгебраїчного виразу»

Виконали студенти
1-го курсу магістратури
спеціальності «інформатика»
Онищенко Ігор Орестович
Мітічкін Дмитро Сергійович

Київ – 2019

Опис завдання

Розробити програму, яка:

1. здійснює розбір математичного виразу *f(x),* заданого в лінійній формі, наприклад: $2\sin\left(\frac{1}{e^{3x}+1}\right) - tg(x+\frac{\pi}{2})$;
2. вказує на основні типи помилок, якщо вони є: невідповідність дужок, неправильний запис назв функцій тощо;
3. здійснює табуляцію функції $y = f(x)$ на заданому проміжку $[x_1; x_2]$;
4. креслить графік функції $y = f(x)$ на проміжку $[x_1; x_2]$.

Теоретичні відомості

Регулярні вирази

Регулярний вираз, або шаблон — це рядок, що описує або збігається з множиною рядків, відповідно до набору спеціальних синтаксичних правил. Ці послідовності використовують для точного описання множини без перелічення всіх її елементів. Наприклад, множина, що складається із слів «грати» та «ґрати» може бути описана регулярним виразом «(г|ґ)рати».

Наведемо деякі приклади синтаксису регулярних виразів:

- АБО:
  Позначається вертикальною рискою. Наприклад, виразом «грати|ґрати» описується як «грати», так і «ґрати».
- Групування:
  Дужки використовуються для вказання області дії та порядку операторів. Так, шаблони «грати|ґрати» і «(г|ґ)рати» — еквівалентні.
- Квантифікація:

Квантифікатор після елементу вказує, скільки разів елемент може повторюватись.
? — знак питання позначає 0 або 1 повторення. «аб?в» позначає «ав» і «абв», але не «аббв»

***** — зірочка позначає 0 або більше повторень. «аб*в» позначає «ав», «абв», «аббв», «абббв» і так далі.

+ — плюс позначає 1 або більше повторень. «аб+в» позначає «абв», «аббв», «абббв» і так далі, але не «ав».

{n} — рівно n повторень.

{min,} — min або більше повторень.

{min, max} — не менше min і не більше max повторень.

- Байдужий символ:
  Байдужий символ «.» позначає будь-який символ. «а.б» позначає рядок, який складається з трьох символів, починається з «а» і закінчується на «б». А «а.*б» позначає довільний рядок, який починається з «а» і закінчується на «б».

Синтаксис регулярних виразів дещо варіюється в залежності від інструментів або контексту, але в будь-якому випадку можна побудувати скіль завгодно складні регулярні вирази для точного описання тих чи інших множин. Регулярні вирази використовуються в пошукових системах, текстових редакторах, і для лексичного аналізу.

Польский інвертний запис

Польский інвертний запис (далі ПОЛІЗ, англ. Reverse Polish Notation/RPN) — це така форма запису математичних та логічних операцій, в якій операнди стоять перед знаками операцій (постфіксний запис).

В ПОЛІЗі операнди розділяються пробілами, вираз читається зліва направо. Коли зустрічається знак операції, ця операція виконується над необхідною кількістю останніх перед нею в записі операндів, у порядку запису. Після цього знак операції і її операнди замінюються на її результат, і продовжується читання по тій самій схемі. Результат обчислення — результат виконання останньої операції.

Наприклад, вираз «$42 - 23 \cdot 16$» запишеться як «$42\ 23\ 16\ \cdot\ -$».

ПОЛІЗ буде коротшим за звичайний інфіксний запис, він не потребує вказування пріорітетів операцій і використання дужок, однак дві операції з різною кількістю операндів не можна позначати одним і тим самим символом, інакше вираз неможливо буде обчислити.

Алгоритм обчислення ПОЛІЗу використовує стек. Вираз обчислюється наступним чином:

1. Якщо на вхід подається операнд, він вкладається в стек.
   Якщо подається знак операції, то вона виконується над необхідною кількістю операндів, які беруться зі стека в порядку їх вкладання до нього. Результат вкладається в стек.
2. Якщо вхідний набір символів не оброблений повністю, перейти до кроку 1.
3. Коли всі символи буде оброблено, результат знаходитиметься на вершині стека.

Алгоритм сортувальної станції

Алгоритм сортувальної станції виконує синтаксичний розбір виразів в інфіксній нотації і може бути застосований для отримання ПОЛІЗу даного виразу (для цього треба буде почати з даного рядка у зворотньому напрямку, і розвернути чергу на виході). Алгоритм використовує стек і працює над вхідним та вихідним рядками:

- Допоки на вході є токени:
    - Прочитати токен.
    - Якщо токен — число, додати до черги на виході.
    - Якщо токен — функція, тоді заштовхнути його у стек.
    - Якщо токен — це відокремлювач аргументів функції (наприклад, кома):
        - Доки токен на верхівці стеку не ліва дужка, виштовхувати зі стеку оператори до черги на виході. Якщо ліва дужка так і не зустрілась, тоді або відокремлювач не на своєму місці, або пропущені дужки.
    - Якщо токен — це оператор, o_1, тоді:
        - доки на верхівці стеку оператор, o_2, і

        або o_1 ліво-асоціативний і його першість менша ніж чи дорівнює першості o_2,

        або o_1 право-асоціативний і першість менша ніж у o_2,

        виштовхнути o_2 зі стеку до черги на виході;
        - заштовхнути o_1 на стек.
    - Якщо токен — це ліва дужка, заштовхнути на стек.
    - Якщо права дужка:

    - Допоки не зустрінеться ліва дужка, виштовхувати оператори зі стека до черги на виході.
    - Виштовхнути ліву дужку зі стеку, але не до черги на виході.
    - Якщо токен на верхівці стеку — це функція, виштовхнути його до черги на виході.
    - Якщо стек завершився, а ліва дужка не зустрілась, тоді вона була пропущена.
- Коли вже немає токенів на вході:
    - Доки ще є оператори в стеку:
        - Якщо оператор на верхівці стеку — це дужка, значить була пропущена дужка.
        - Виштовхнути оператори до черги на виході.
        - Вихід.

Розбір виразу і обчислення його значення

Далі наведено частину коду програми, яка відповідає за розбір і обчислення значення виразу.

calculator.py:

```
import lexscan
from math import pi, sin, cos, tan, log, sqrt, exp
import numpy as np
import matplotlib.pyplot as plt
from pprint import pprint


class Calculator:
    prec = {'^': 4, '*': 3, '/': 3, '%': 3, '+': 2, '-': 2}
    assoc = {'^': 'right', '*': 'left', '/': 'left', '%': 'left', '+': 'left', '-': 'left'}
    constants = {'PI': pi}
    operators = {'+': lambda x, y: x+y, '-': lambda x, y: x - y, '*': lambda x, y: x*y, '/': lambda x, y: x/y,
    functions = {'sin': lambda x: sin(x), 'cos': lambda x: cos(x),  '%': lambda x, y: x % y, '^': lambda x, y: x**y}
                 'tan': lambda x: tan(x), 'log': lambda x: log(x),
                 'sqrt': lambda x: sqrt(x), 'exp': lambda x: exp(x),
                 'unary_minus': lambda x: -x}

    def __init__(self):
        self.expr = None
        self.tokens = None
        self.RPN = None
        self.a = None
        self.b = None

    @staticmethod
    def _getTokens(expr):
        spaceexp = lexscan.ScanExp(r'\s+', significant=False)
        wordexp = lexscan.ScanExp(r'[a-z]+', name="function")
        numexp = lexscan.ScanExp(r'[0-9]*\.?[0-9]+', name="number")
        leftPar = lexscan.ScanExp(r'\(', name="leftPar")
        rightPar = lexscan.ScanExp(r'\)', name="rightPar")
        operator = lexscan.ScanExp(r'[\+\-\*\/\^\%]', name="operator")
        badentry = lexscan.ScanExp(r'[\$\,\\\@\#\!\&\~\=]', name="badentry")
        return lexscan.tokenize(expr, (spaceexp, wordexp, numexp, leftPar, rightPar, operator, badentry))
```

```python
@staticmethod
def _getRPN(tokens):
    res = []
    stack = []
    for (i, tok) in enumerate(tokens):
        # some utils
        if tok.type == 'badentry':
            raise Exception('невідомий символ.')
        if tok.text in Calculator.constants.keys():
            tok.text = Calculator.constants[tok.text]
            tok.type = 'number'
        elif tok.text == 'x':
            tok.type = 'number'

        if tok.text == '-' and (i == 0 or tokens[i-1].type == 'operator' or tokens[i-1].type == 'leftPar'):
            tok.type = 'function'
            tok.text = 'unary_minus'

        if tok.type == 'number':
            res.append(tok)
        elif tok.type == 'function':
            stack.append(tok)
        elif tok.type == 'operator':
            while len(stack) > 0 and (stack[-1].type == 'function' or
                res.append(stack.pop())
            stack.append(tok)
        elif tok.type == 'leftPar':
            stack.append(tok)
        elif tok.type == 'rightPar':
            if len(stack) == 0:
                raise Exception("перевірте дужки.")
            while stack[-1].type != 'leftPar':
                res.append(stack.pop())
                if len(stack) == 0:
                    raise Exception("перевірте дужки.")
            if stack[-1].type == 'leftPar':
                stack.pop()
        else:
            raise Exception("неіснуюча операція.")

    while len(stack) > 0:
        top = stack.pop()
        if top.type == 'leftPar' or top.type == 'rightPar':
            raise Exception("перевірте дужки.")
        res.append(top)

    return res
```

(stack[-1].type == 'operator' and (Calculator.prec[stack[-1].text] > Calculator.prec[tok.text] or
(Calculator.prec[stack[-1].text] == Calculator.prec[tok.text] and Calculator.assoc[stack[-1].text] == 'left'))):

```python
@staticmethod
def _getResult(rpn_tokens, x):
    stack = []
    for tok in rpn_tokens:
        if tok.type == 'function':
            try:
                stack.append(float(Calculator.functions[tok.text](stack.pop())))
            except ValueError:
                raise Exception('некоректний аргумент функції.')
            except:
                raise Exception("неімплементована функція.")
        elif tok.type == 'operator':
            if len(stack) == 0:
                raise Exception("недостатня кількість операндів.")
            op2 = stack.pop()
            if len(stack) == 0:
                raise Exception("недостатня кількість операндів.")
            op1 = stack.pop()
            try:
                stack.append(float(Calculator.operators[tok.text](op1, op2)))
            except ZeroDivisionError:
                raise Exception('ділення на нуль')
            except:
                raise Exception("неімплементований оператор.")
        else:

            if tok.text == 'x':
                stack.append(x)
            else:
                stack.append(float(tok.text))

    res = stack.pop()
    if len(stack) > 0:
        print(stack)
        raise Exception('вираз містить забагато аргументів.')
    return res
```

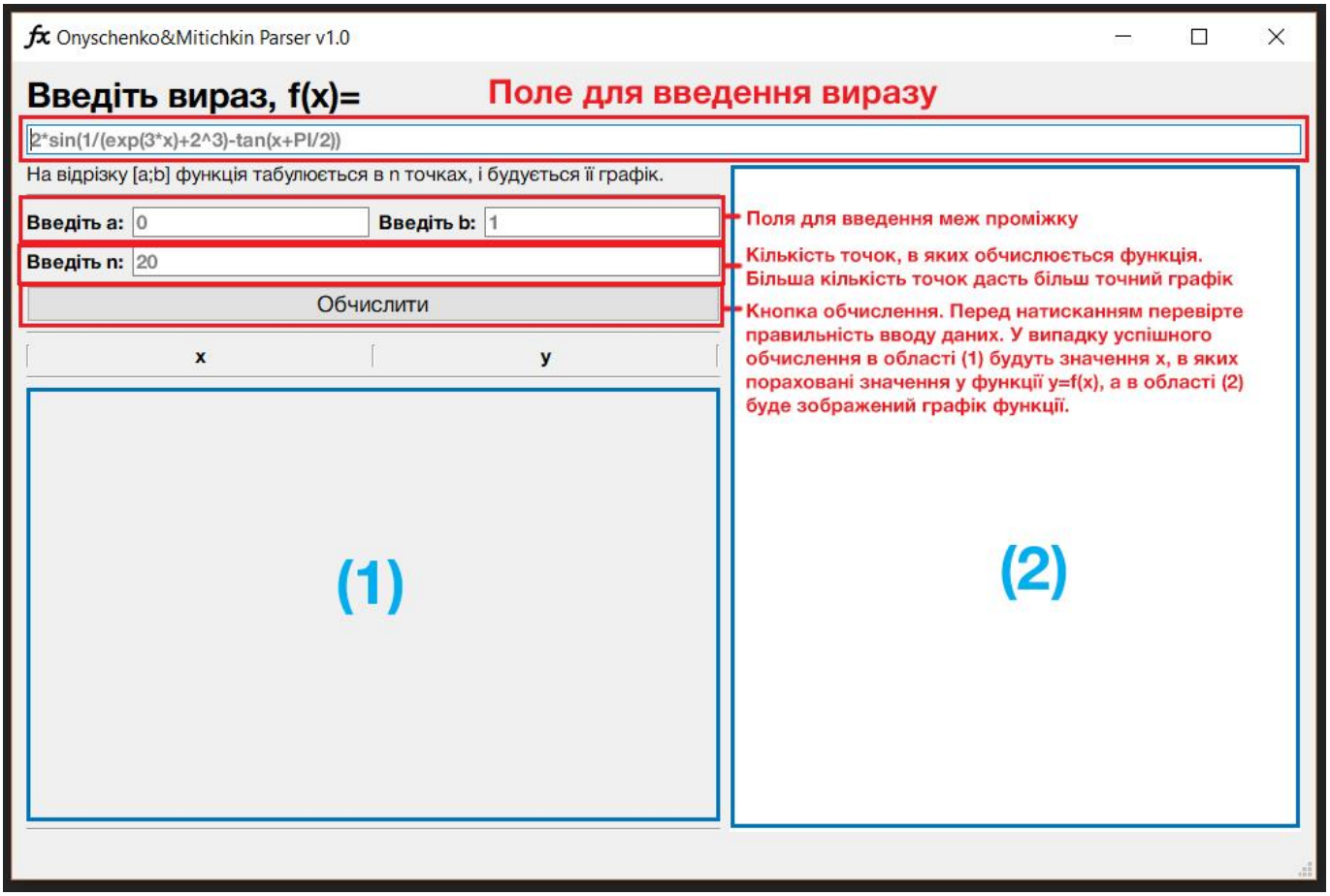
lexscan.py:

```
import regex

__version__ = '1.0.0'


def tokenize(strinput, expressions, source=None, newline='\n'):
    for exp in expressions:
        exp.setsearchstring(strinput)
    tokens = []
    strlen = len(strinput)
    strpos = 0
    strline = 1
    while strpos < strlen:

        bestexp = None
        bestmatch = None
        bestlen = 0
        for exp in expressions:
            match = exp.search(strpos)
            if match and (match.start() == strpos):
                matchlen = (match.end() - strpos) if match else 0
                if matchlen > bestlen:
                    bestexp = exp
                    bestmatch = match
                    bestlen = matchlen

        if bestexp:
            matchstr = bestmatch.group()
            if bestexp.significant:
                tokens.append(ScanToken(matchstr, bestexp, bestmatch, strpos, strline, source))
            strline += matchstr.count(newline)
            strpos += bestlen
        else:
            tokens.append(ScanToken(strinput[strpos], None, None, strpos, strline, source))
            strline += (strinput[strpos] == newline)
            strpos += 1
    return tokens
```

```
class ScanExp(object):


    def __init__(self, expression, settings=regex.IGNORECASE, significant=True, name='', precompile=True):

        self.expression = expression
        self.settings = settings
        self.significant = significant
        self.name = name

        self.regex = self.compile() if precompile else None

        self.cachematch = None
        self.cachepos = -1
        self.cachestr = ''

    def setsearchstring(self, string):
        self.clearcache()
        self.cachestr = string

    def search(self, start=0):
        if not self.regex:
            self.compile()
        if (self.cachepos >= start):
            return self.cachematch
        else:
            match = self.regex.search(self.cachestr, start)
            self.cachepos = match.start() if match else -1
            self.cachematch = match
            return match

    def clearcache(self):
        self.cachematch = None
        self.cachepos = -1
        self.cachestr = ''

    def compile(self):
        return regex.compile(self.expression, self.settings)



    def __str__(self):
        return self.name if self.name else self.expression

    def __repr__(self):
        return (self.name + ": " if self.name else "") + self.expression + " (" + (
        "sig" if self.significant else "non") + ")"

class ScanToken(object):


    def __init__(self, text='', expression=None, match=None, strpos=0, linepos=0, source=None):
        self.text = text
        self.match = match
        self.expression = expression
        self.strpos = strpos
        self.linepos = linepos
        self.source = source
        self.type = expression.name



    def __str__(self):
        return self.text

    def __repr__(self):
        return (str(self.source) + ":" if self.source else "") + str(self.linepos) + ":" + str(
            self.strpos) + ": '" + str(self.text) + "' " + self.type
```

Інструкція до використання програми

Програма написана мовою Python з використанням regex, numpy, matplotlib та PyQt5.



Onyschenko&Mitichkin Parser v1.0

Введіть вираз, f(x)= **Успішно виконане обчислення:**

4*sin(1/x)

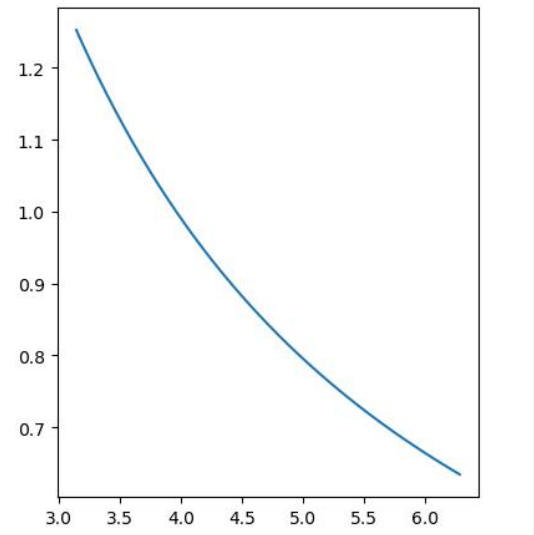
На відрізку [a;b] функція табулюється в n точках, і будується її графік.

Введіть a: 3.14 **Введіть b:** 6.28

Введіть n: 250

Обчислити

x	y
3.14	1.2524605325146396
3.1526104417670684	1.2476202045942102
3.1652208835341367	1.2428164445197791
3.177831325301205	1.2380488482617327
3.190441767068273	1.2333170175694776
3.2030522088353415	1.228620559871811
3.2156626506024097	1.2239590881792528
3.228273092369478	1.219332220988301
3.2408835341365463	1.2147395821875664
3.2534939759036146	1.2101808009657482
3.266104417670683	1.205655511721411
3.278714859437751	1.201163353974524
3.2913253012048194	1.1967039722797241
3.3039357429718876	1.192277016141269



Onyschenko&Mitichkin Parser v1.0

Введіть вираз, f(x)=

4*sin(1/x))

На відрізку [a;b] функція табулюється в n точках, і будується її графік.

Введіть a: 3.14 **Введіть b:** 6.28

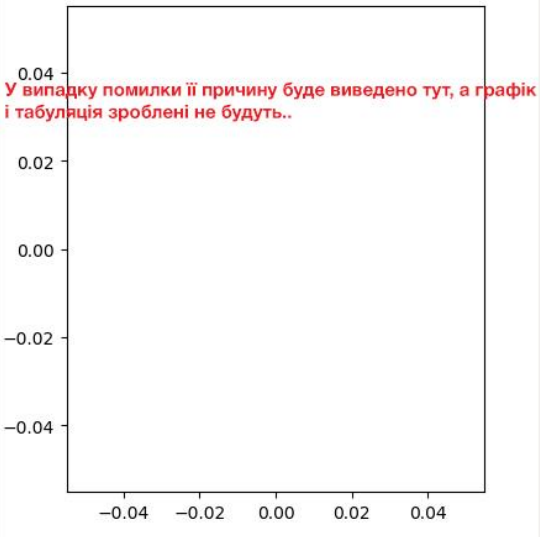
Введіть n: 250

Обчислити

Помилка: перевірте дужки.

У випадку помилки її причину буде виведено тут, а графік і табуляція зроблені не будуть..

x	y



Список використаних джерел

1. https://en.wikipedia.org/wiki/Regular_expression
2. https://en.wikipedia.org/wiki/Shunting-yard_algorithm
3. https://en.wikipedia.org/wiki/Reverse_Polish_notation
4. https://pypi.org/project/PyQt5/