**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Факультет комп’ютерних наук та кібернетики

Кафедра математичної інформатики

**Звіт**

**до лабораторної роботи на тему:**

**“Розробка програми-парсера алгебраїчного виразу.”**

студента першого курсу магістратури Інформатики

за спеціальністю 122 Комп’ютерні науки

**Баса Максима Олександровича**

# 

**Завдання**

Створити програму, яка:

* Здійснює розбір математичного виразу f(x), заданого в лінійній формі, наприклад: 2\*sin(1/(exp(3\*x)+1)-tg(x+PI/2)
* Вказує на основні типи помилок, якщо вони є (невідповідність дужок, неправильний запис назв функцій і т.д.)
* Здійснює табуляцію функції y=f(x) на заданому проміжку [x1..x2].
* Креслить графік функції y=f(x) на проміжку [x1..x2]

Математичні функції, які мають оброблятися:

* sin
* cos
* tg
* exp
* abs
* pow
* sqrt

Допустимі константи:

* E
* PI

**Теоретичні відомості**

Кожний метод парсингу математичного виразу можна поділити на декілька частин або підзадач:

* оцінка коректності виразу
* розбір виразу на певні частини, власне парсинг
* обчислення виразу

Одним із популярних алгоритмів є **shunting-yard algorithm(Алгоритм сортувальної станції) -** метод синтаксичного розбору математичних виразів наданих в **інфіксній нотації**. Його можна використовувати для отримання інверсного польського запису (ПОЛІЗ) або абстрактного синтаксичного дерева. Алгоритм було винайдено Дейкстрою і названо алгоритм «сортувальної станції», бо ця операція нагадує дію залізничної сортувальної станції. Даний алгоритм базується на стеку.

**Інфіксні форму** виразів використовує більшість людей, наприклад 3+4 або 3+4\*(2−1). Інфіксну форму можна перевести в **постфіксну**, після чого отримаємо 3 4 + для прикладу 3 + 4.

Етапи роботи алгоритму сортувальної станції:

1. Для вхідної послідовності символів виразу генерується вектор **токенів.** Токен - певний об’єкт, що має свої певні характеристики. Для алгебраїчного виразу існує певний набір токенів, а саме: **число, ім’я змінної, ім’я функції або оператор.** У кожної функції і оператора мають бути певні властивості, а саме: асоціативність, пріоритет та кількість операндів.
2. Перетворення вектору токенів в інфіксній формі до вектору токенів в постфіксній формі. На цьому етапі алгоритм потребує 2 вектори та додатковий стек для функцій та операторів.
3. Оцінка постфіксної послідовності.

Розберемо детальніше процес генерування постфіксної послідовності токенів на прикладі:

Початкові дані: 3+4\*2-1

1. Додаємо 3 до постфіксної послідовності
2. Оператор **+** додається до стеку операторів
3. Додаємо 4 до постфіксної послідовності
4. Оператор **\*** додається до стеку операторів, після перевірки того факту, що в стеку не знаходиться оператор з більшим пріоритетом
5. Додаємо 2 до постфіксної послідовності
6. Оператор **-** додається до стеку операторів, але перед цим додаємо оператор \* до постфіксної послідовності, оскільки він має більший пріоритет ніж оператор -
7. Додаємо 1 до постфіксної послідовності
8. Оскільки вхідна інфіксна послідовність порожня, додаємо оператори зі стеку до постфіксної послідовності
9. Вихід 3 4 2 \* 1 - +

Оцінка постфіксної послідовності проводиться наступним чином з використанням стеку результатів:

Початкові дані: 3 4 2 \* 1 - +, стек = []

1. Додаємо 3 до стеку результатів; стек = [3]
2. Додаємо 4 до стеку результатів; стек = [3, 4]
3. Додаємо 2 до стеку результатів; стек = [3, 4, 2]
4. Наступний токен \* це оператор, який застосовується до двох операндів, тому дістаємо зі стеку два значення, проводимо над ними операцію \* і додаємо до стеку нове значення; стек = [3, 8]
5. Додаємо 1 до стеку результатів; стек = [3, 8, 1]
6. Наступний токен - це оператор, який застосовується до двох операндів, тому дістаємо зі стеку два значення, проводимо над ними операцію - і додаємо до стеку нове значення; стек = [3, 7]
7. Наступний токен + це оператор, який застосовується до двох операндів, тому дістаємо зі стеку два значення, проводимо над ними операцію + і додаємо до стеку нове значення; стек = [10]
8. Вихід 10

У результаті повинно залишитись одне значення, якщо немає таких операторів, які можуть видавати результатами пару або більше значень.

Більш простим методом є рекурсивна оцінка виразу. Даний підхід проходиться по кожному символу вхідного рядка виразу і певним чином додає до результату. Даний метод дуже схожий на те, як людина оцінює вираз. Програмно даний підхід завжди “спускається” від менш пріоритетних операції, до більш пріоритетних, а кінцевими пунктами є символ числа або імені функції. Розглянемо детальніше на прикладі:

Вхідні дані: 3+4\*2-1, pos = 0 - позиція символу в рядку, symb = 3 - поточний символ

1. Починаємо з додавання/віднімання, тобто дивимося на вираз як на суму спускаємось нижче по рівню пріоритету; pos = 0, symb = 3
2. Розгядаємо множення/ділення; pos = 0, symb = 3
3. Розглядаємо дужки; pos = 0, symb = 3
4. Розглядаємо значення, оскільки symb = 3, символ числа переходимо до парсингу числа; pos = 0, symb = 3
5. Читаємо символи поки вони є частиною числа, результат 3; pos=1, symb = +
6. Оскілки symb=+, в результат записуємо 3 та продовжуємо парс рядка з наступного символу; pos = 2, symb = 4
7. Знову число, результат 4; pos = 3, symb = \*
8. Оскільки символ \*, записуємо до результату на цьому рівні 4 та продовжуємо з наступного символу; pos = 4, symb = 2
9. Знову число, результат 2; pos = 5, symb = -
10. Закінчуємо з оператором \*, результат 8 піднімаємось вище; pos = 5, symb = -
11. Закінчуємо з оператором +, результат 11, продовжуємо парсинг; pos = 5, symb = -
12. Оскільки символ -, записуємо в результат 11, продовжуємо з наступного символу; pos = 6, symb = 1
13. Число, результат 1; pos=7, symb = None
14. Закінчуємо з оператором -, результат 10 піднімаємось вище
15. Оскільки рядок закінчився припиняємо парсинг і вертаємо значення
16. Вихід 10

**Реалізація**

Було реалізовано shunting-yard algorithm на мові Python. Алгоритм має три головні функції, а саме:

* **tokenize -** функція, що перетворює вхідний рядок виразу, в інфіксну послідовність токенів

**def tokenize(self):**

**self.tokens = []**

**token\_start = 0**

**expression\_curr = 0**

**while expression\_curr < len(self.expression):**

**token\_start\_char = self.expression[token\_start]**

**token = ''**

**if token\_start\_char in NUMBERS:**

**condition = lambda ch : ch in NUMBERS or ch == '.'**

**token = self.collect\_while(token\_start, condition)**

**self.tokens.append(string\_to\_float(token))**

**expression\_curr += len(token)**

**elif token\_start\_char in NAMECHARS:**

**condition = lambda ch : ch in NAMECHARS**

**token = self.collect\_while(token\_start, condition)**

**if token in FUNCTIONS or token in CONSTANTS or token in VARNAMES:**

**self.tokens.append(token)**

**else:**

**raise Exception("undefind name " + token)**

**expression\_curr += len(token)**

**else:**

**self.tokens.append(token\_start\_char)**

**expression\_curr += 1**

**token\_start = expression\_curr**

* **infix -** функція, що трансформує інфіксну форму у постфіксну

**def infix(self):**

**operator\_stack = []**

**self.postfix\_tokens = []**

**for token in self.tokens:**

**if is\_number(token):**

**self.postfix\_tokens.append(token)**

**elif token in CONSTANTS:**

**self.postfix\_tokens.append(token)**

**elif token in VARNAMES:**

**self.postfix\_tokens.append(token)**

**elif token in FUNCTIONS:**

**operator = FUNCTIONS[token]**

**while operator\_stack:**

**if not operator\_stack[-1] in FUNCTIONS:**

**break**

**curr\_stack\_operator = FUNCTIONS[operator\_stack[-1]]**

**if curr\_stack\_operator['priority'] <= operator['priority'] and operator['assoc'] == 'left':**

**self.postfix\_tokens.append(operator\_stack.pop())**

**else:**

**break**

**operator\_stack.append(token)**

**elif token == '(':**

**operator\_stack.append(token)**

**elif token == ')' or token == ',':**

**if not '(' in operator\_stack:**

**raise Exception("Missing opening bracket")**

**while operator\_stack and operator\_stack[-1] != '(':**

**self.postfix\_tokens.append(operator\_stack.pop())**

**if operator\_stack[-1] == '(' and token == ')':**

**operator\_stack.pop()**

**else:**

**raise Exception("Invalid or unsupported function " + token)**

**if '(' in operator\_stack:**

**raise Exception("Missing closing brakcet")**

**while operator\_stack:**

**self.postfix\_tokens.append(operator\_stack.pop())**

* **postfix -** функція, що оцінює постфіксну послідовність

**def postfix(self, x):**

**result\_stack = []**

**tokens = self.postfix\_tokens[:]**

**while tokens:**

**token = tokens.pop(0)**

**if token in FUNCTIONS:**

**operator = FUNCTIONS[token]**

**args = []**

**for i in range(operator['arity']):**

**if len(result\_stack) == 0:**

**raise Exception("Too few operands for operation " + token)**

**args.append(result\_stack.pop())**

**args.reverse()**

**result\_stack.append(operator['func'](\*args))**

**elif token in CONSTANTS:**

**result\_stack.append(CONSTANTS[token])**

**elif is\_number(token):**

**result\_stack.append(token)**

**elif token in VARNAMES:**

**result\_stack.append(x)**

**return result\_stack.pop()**

**Інструкція використання**

Для використання парсингу виразу достатньо створити екземпляр класу з відповідним виразом наступним чином:

**>>> from shuting\_yard import \***

**>>> expression = ShuntingYard("2\*sin(1/(exp(3\*x)+1)-tg(x+PI/2))")**

після чого існує три види взаємодії з даним екземпляром класу, а саме:

* можна отримати значення функції для певного аргументу, наприклад

**>>> expression(0)**

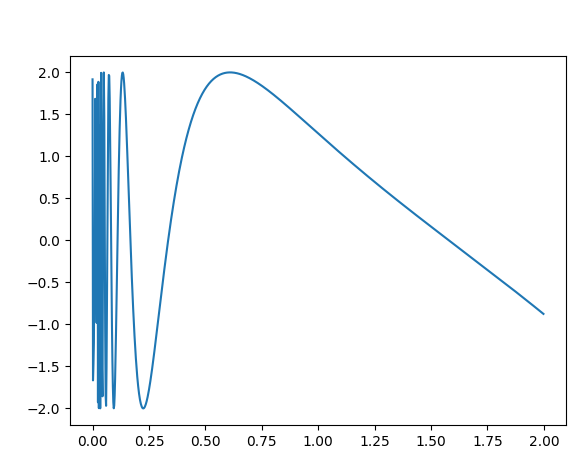
**1.9173935258570953**

**>>> expression(12)**

**-1.9999964765731992**

* можна отримати графік функції вказавши інтервал, після чого відкриється додаткове вікно з графіком функції на даному інтервалі:

**>>> expression.plot(0, 2)**



* можна отримати табуляцію функції вказавши інтервал:

**>>> expression.tabulate(0, 2)**

**x y**

**0.00 1.92**

**0.10 -1.65**

**0.20 -1.68**

**0.30 -0.74**

**0.40 1.04**

**0.50 1.81**

**0.60 2.00**

**0.70 1.93**

**0.80 1.74**

**0.90 1.51**

**1.00 1.27**

**1.10 1.04**

**1.20 0.81**

**1.30 0.59**

**1.40 0.37**

**1.50 0.16**

**1.60 -0.04**

**1.70 -0.25**

**1.80 -0.45**

**1.90 -0.66**

**2.00 -0.88**

**Джерела**

<https://en.wikipedia.org/wiki/Shunting-yard_algorithm>

Результат: [https://github.com/Basicula/Labs/tree/master/5thCourseS1/ТОтаКТ/FuncParser](https://github.com/Basicula/Labs/tree/master/5thCourseS1/%D0%A2%D0%9E%D1%82%D0%B0%D0%9A%D0%A2/FuncParser)