Київський національний університет
імені Т.Шевченка

Звіт

до лабораторної роботи №5
з дисципліни:
«Системне Програмування»

Студента третього курсу
Групи МІ-32
Факультету комп'ютерних наук
та кібернетики
Федорича Андрія

Київ-2023

Мета

Метою лабораторної роботи є на базі Lex/YACC (або аналога) розробити синтаксичний аналізатор (з генерацією коду або обчисленнями - опціонально) для C або іншої сучасної імперативної (ООП, процедурної, тощо) мови програмування, або для арифметичних виразів (спрощений варіант).

Основні принципи виконання роботи

Основні виконані завдання:

2 бали - коректний розв'язок (працюючий проєкт)
2 бали - відповіді на питання по коду
+2 бали - за відповіді на питання по теорії: LALR-аналізатор, принципи роботи yacc, як розв'язуються конфлікти (shift-reduce, reduce-reduce) в ході розбору...

Завдання виконані на додаткові бали:

+3 бали - за додавання семантики до AST (реалізацію обчислень)
+3 бали - за візуалізацію AST (d3.js, або будь-яким зручним для користувача та наочним засобом)

Мова виконання: Python

Джерело:

Код програми, тестові та результуючі файли можна знайти за посиланням:
https://github.com/StopFuture/KNU_Bachelor_Assignments/tree/main/SystemProgramming/Lab_5

Реалізація:

- Як основу виконання лабораторної використано PLY (Python Lex-Yacc). PLY - це pure-Python реалізація популярних інструментів побудови компіляторів lex і yacc.

  Основна мета PLY - залишатися досить вірним тому, як працюють традиційні інструменти lex/yacc. Це включає в себе підтримку аналізу LALR (1), а також забезпечення великої перевірки вхідних даних, звітів про помилки та діагностики.

  Таким чином, якщо користувач використовував yacc в іншій мові програмування, йому повинно бути надзвичайно просто використати PLY.

- Структкура поекту:

```
.
├── AST
│   │
│   ├── ast_drawer.py
│   ├── ast_nodes.py
│
├── Parser
│   │
│   ├── lex.py
│   ├── parser.out
│   ├── parsetab.py
│   └── yacc_grammar.py
│
├── main.py
```

├── requirementx.txt

Граматика та її властивості

1. Бінарні функції:
- Включають операції такі як плюс (+), мінус (-), множення (*) і ділення (/).
- Їх пріоритет заданий через правило 'прецедентності', де множення і ділення мають перевагу над додаванням і відніманням.

2. Унарна операція мінус:
- У програмі втілено унарний мінус (UMINUS), який відрізняється високим пріоритетом і некомутативністю.

3. Використання дужок:
- Граматика дозволяє застосування круглих дужок для зміни порядку виконання операцій.

4. Розширені функції:
- Програма обладнана оператором піднесення до степеня (^).

5. Впровадження змінних:
- Граматика розширена для включення змінних, включаючи присвоєння значень (=) та їх використання в рівняннях.

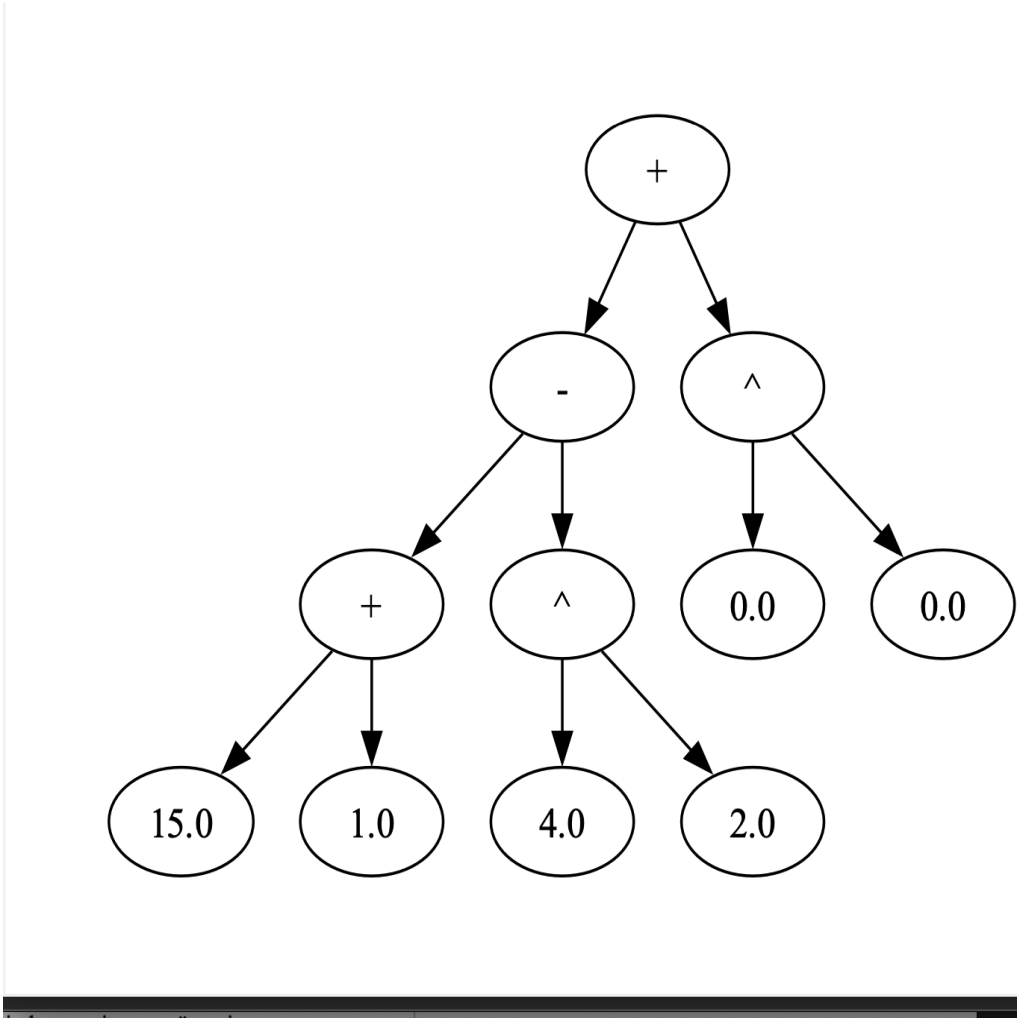
6. Граматичні правила:
- Основні правила включають визначення виразів через рекурсивний метод, де вираз може включати числа, бінарні операції, унарний мінус, дужки та змінні.

Реалізована граматика є контекстно-вільною граматикою, оскільки кожне правило граматики замінює один символ (наприклад, `expression`) на послідовність інших символів без урахування контексту цих символів.

Демонстрація роботи

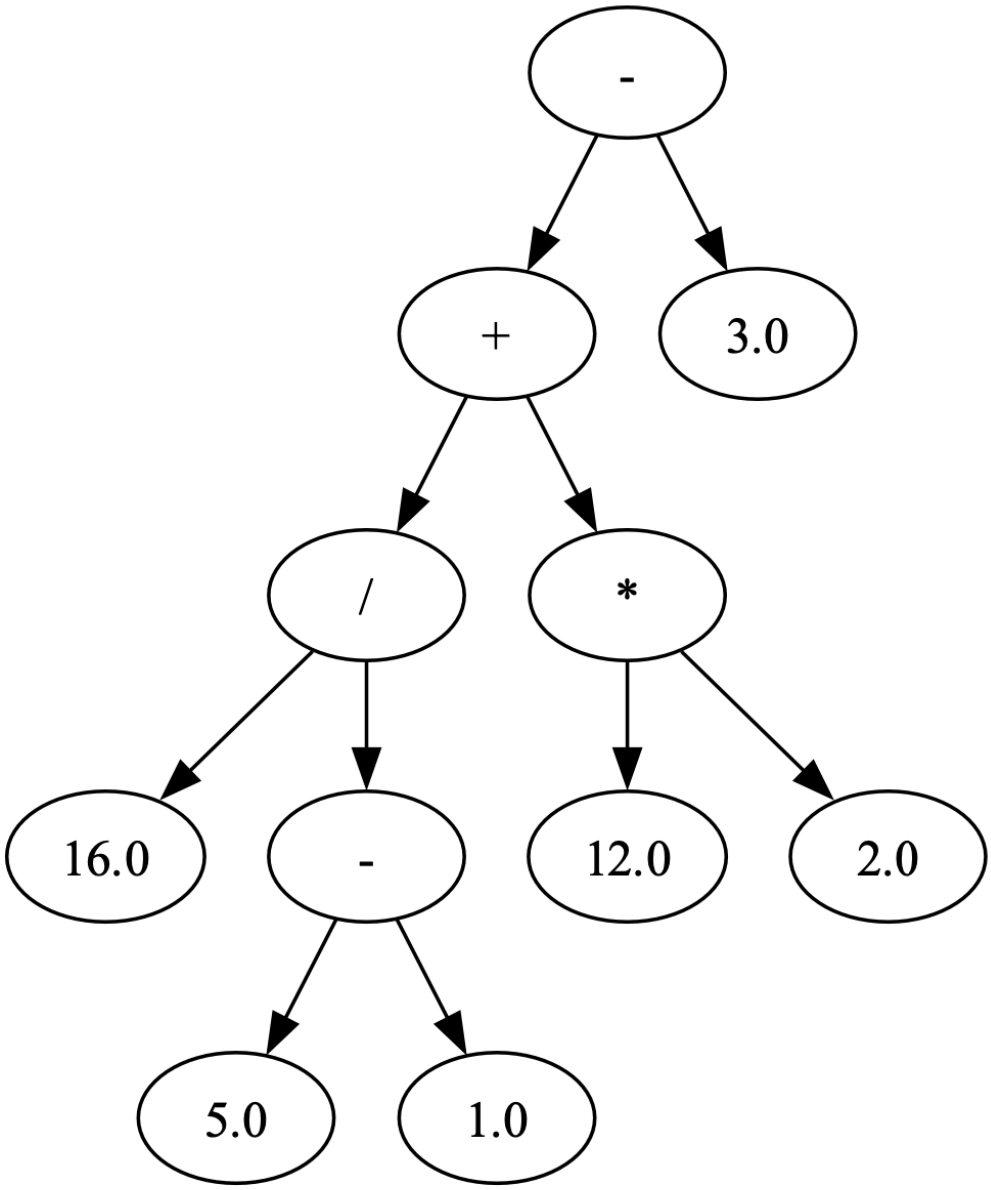
Приклад 1

```
/Users/stopfuture/PycharmProjects/Lab_5_Lex_Yacc/venv/bin/python
Enter a statement or expression: (15 + 1) - 4 ^ 2 + 0^0
1.0
```



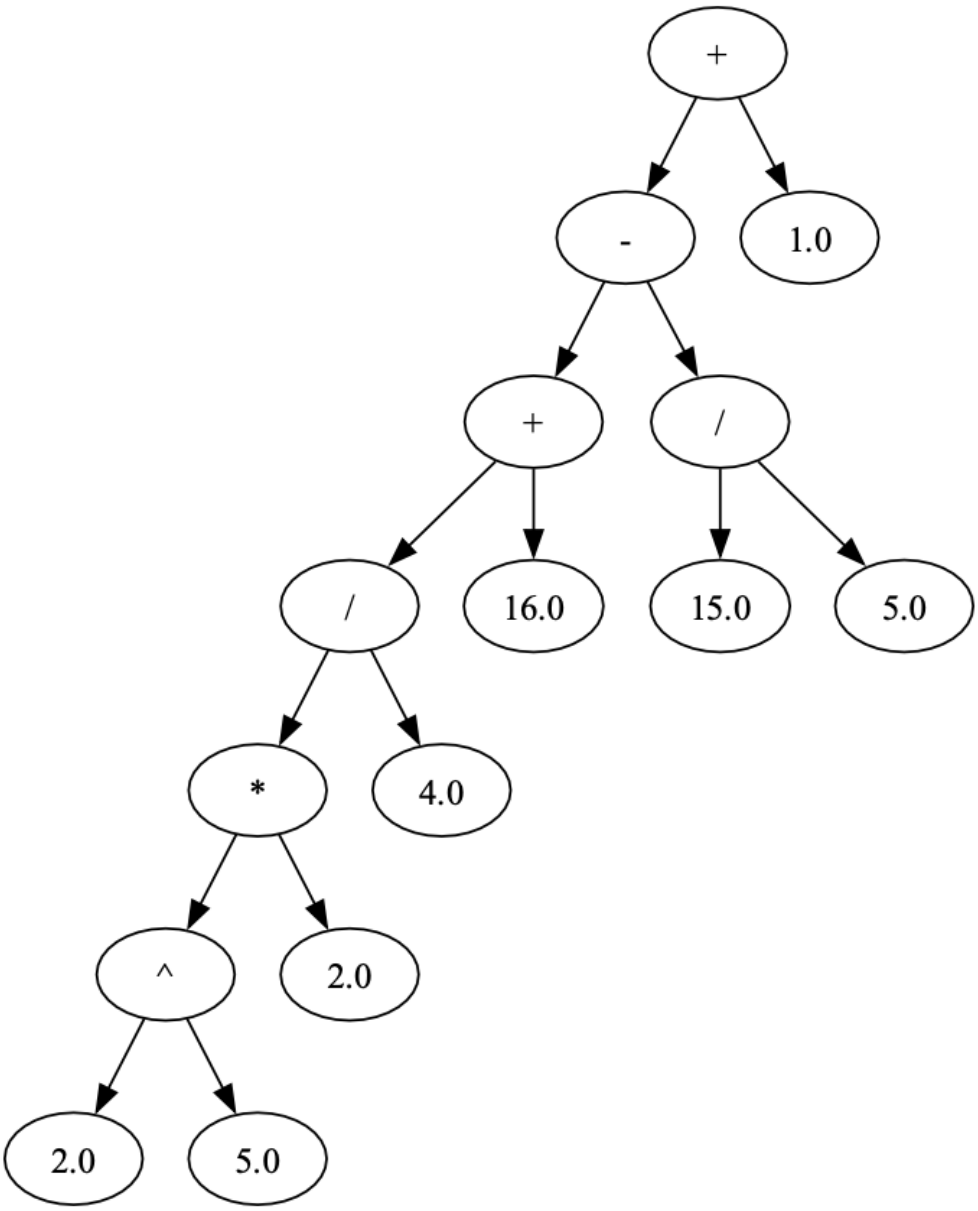
Приклад 2

```
Enter a statement or expression: 16 / (5 - 1) + 12 * 2 - 3
25.0
```



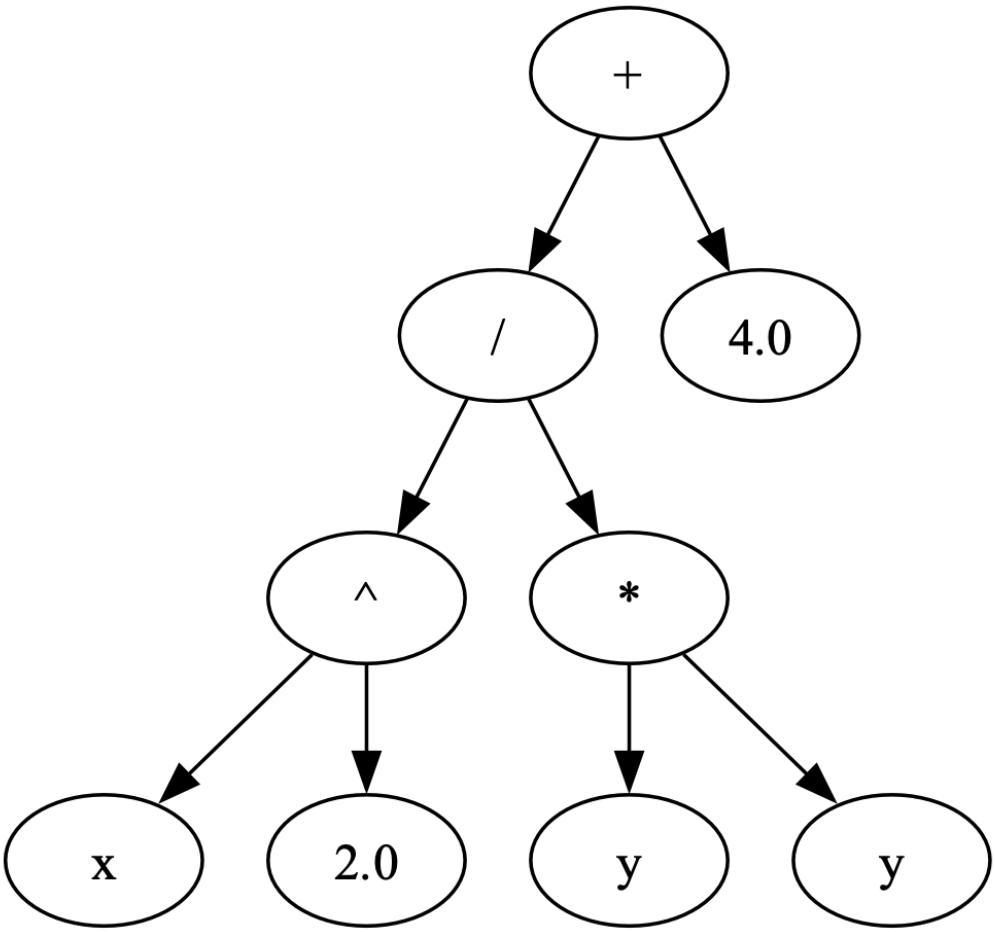
Приклад 3

```
Enter a statement or expression: 2^5 * 2 / 4 + 16 - 15 / 5 + 1
30.0
```



Приклад 4 (Змінні)

```
Enter a statement or expression: x = 12 - 5 + 3^2
16.0
Enter a statement or expression: y = x + 7
23.0
Enter a statement or expression: z = x^2 / (y * y) + 4
4.4839319470699435
```



Висновок

У цій лабораторній роботі було розроблено синтаксичний аналізатор на базі Lex/YACC для арифметичних виразів, що представляє собою ключовий елемент у вивченні синтаксичного аналізу в мовах програмування. В ході роботи було успішно реалізовано базову граматику, що демонструє здатність розпізнавати та обробляти бінарні та унарні операції, скобки, змінні, а також підтримує піднесення до степеня.

Крім того, було досягнуто значного прогресу в плані візуалізації AST, використовуючи сучасні інструменти, що надають зручний та наочний спосіб представлення абстрактного синтаксичного дерева. Це не лише полегшує розуміння внутрішньої структури аналізатора, але й забезпечує відмінну основу для подальших досліджень і вдосконалення в цій області.

Загалом, ця лабораторна робота успішно демонструє глибоке розуміння принципів синтаксичного аналізу та їх практичне застосування, що відображається у коректному розв'язку задачі, здатності відповідати на запитання, пов'язані з кодом та теорією, а також у реалізації додаткових функцій, які значно підвищують якість та ефективність проекту.