ОС і СП
Лабораторна робота 12: інтерпретація за формулами

Вивчити модель інтерпретатора формул за алгебраїчними правилами. Доповнити інтерпретатор іншими бінарними і унарними операціями.

Матеріали до завдання

- текст лекції "L11 Обчислення формул інтерпретацією.pdf";
- для поглибленого вивчення питань цієї лабораторної роботи можна звернутись до лекцій "L12 Сканер для формул електронної таблиці.pdf" і "L13 Перетворення виразів ЕТ в постфіксну форму.pdf", в яких подано реалізацію обчислень в форматі електронних таблиць, реалізованих мовою C++;
- посилання на електронні джерела попередньої лабораторної роботи:
  https://uk.wikipedia.org/wiki/Інтерпретатор
  http://kytok.org.ua/?p=732
  https://uk.wikipedia.org/wiki/Компілятор
- файл `arithexpr.py` прикладу інтерпретатора.

За основу роботи прийняти доданий файл `arithexpr.py` і уважно вивчити алгоритм і особливості реалізації сканування (перший перегляд) і особливості нисхідного граматичного розбору відповідно до граматичних правил визначення формули (другий перегляд). Деякі важливі особливості коротко описані тут далі в самому завданні, а всі інші - як коментарі в програмному коді файла `arithexpr.py`.

Граматичні (синтаксичні) правила для формул записані не в класичній формі зображення синтаксису, а різновидом нотації Бекуса-Наура з використанням регулярних виразів:

https://uk.wikipedia.org/wiki/Регулярний_вираз

За темою граматичного розбору можна переглянути посилання:

https://uk.wikibooks.org/wiki/Реалізація_алгоритму_рекурсивного_спуску_з_прикладами_на_C%2B%2B

Завдання частина 1: вивчення, означення і експерименти

1. В текстовому файлі чи в текстовому рядку записана формула, яка складається з цілих чисел, знаків операцій +, -, *, /, круглих дужок. Наприклад:

49 – 108 / (6 + 11) * (100 – 94)

Необхідно виконати обчислення формули за правилами алгебри:

```
arith_expr ::=  term  ( ( "+" | "-" )  term ) *
term ::=  factor  ( ( "*" | "/" )  factor ) *
factor ::=   number  |  "(" arith_expr ")"
number  ::=  cipher  cipher *
cipher ::= "0"| "1"| "2"| "3"| "4"| "5"| "6"| "7"| "8"| "9"
```

Аксіомою цієї граматики є `arith_expr`. Реалізацію граматики виразів і розпізнавання виразу реалізуємо методом нисхідного граматичного розбору за алгоритмом *рекурсивного спуску*. За методом рекурсивного спуску до кожного нетермінального символа будують окрему функцію, яка повинна розпізнати праву частину граматичного правила. При цьому функція може викликати в потрібні моменти інші функції для розпізнавання частини конструкції.

Вхідними даними для програми обчислення виразу є масив лексем, побудований сканером. Арифметичний вираз може мати довільний вигляд в межах записаних вище синтаксичних правил. Тому доцільно підготувати список лексем в формі, зручній для синтаксичного розбору. Кожну лексему можна подати у вигляді пари `(код,значення)`, де код – числове позначення лексеми кожного виду.

2. Програмний код мовою Python інтерпретатора алгебраїчних формул є в доданому файлі `arithexpr.py`. Вивчити програмний код. Можна прийняти його за основу або реалізувати іншою алгоритмічною мовою.

3. Зробимо важливі зауваження щодо алгоритму.

3.1. Як було сказано вище, кожна лексема списку є парою `(код,значення)`, що дозволяє спростити перегляд лексем лише за кодами, а значення – за потреби.

3.2. В кінець списку лексем додаємо в методі `scanner(self)` обмежувач списку лексем `(self.empty,'#')`. Це дозволяє уникнути частих перевірок вичерпання списку лексем в кожному методі. У випадку досягнення обмежувача і спроби продовжити аналіз відбудеться генерування помилки і автоматичне припинення процедури розбору і обчислення.

3.3. Метод `GetNextToken(self)` виконує контрольований перехід до наступної лексеми і генерування помилки у випадку необхідності.

3.4. Методи розбору і обчислення `arithexpr()`, `term()`, `factor()` і `GetNextToken()` мають бути дуже точно синхронізовані щодо пересування списком лексем. Перед викликом кожного з методів `arithexpr()`, `term()`, `factor()` черговою лексемою для аналізу має бути готовою найперша лексема відповідної частини арифметичного виразу.

4. Виконати обчислювальні експерименти з інтерпретатором формул. По-перше, можна розкоментувати в методі `calc()` оператори `print()`, щоб побачити кроки роботи. Наприклад, для показаної вище формули

```
49 - 108 / (6 + 11) * (100 - 94)
```

отримаємо такі результати:

```
49-108/(6+11)*(100-94)
[(1,49),(6,'-'),(1,108),(8,'/'),(3,'('),(1,6),(5,'+'),
 (1,11),(4,')'),(7,'*'),(3,'('),(1,100),(6,'-'),(1,94),
 (4,')'),(0,'#')]
10.882352941176471
```

По-друге, варто записати формулу з синтаксичними помилками і переглянути повідомлення інтерпретатора для випадків помилок.

По-третє, можна додати для експериментів інші оператори `print()` до тексту методів, і отримати спостереження за ходом виконання інтерпретатора.

Завдання частина 2: розширення граматики і програмування

5. Розширення змісту формул для підготовки задачі. Записані вище правила будови формул доповнити такими можливостями (рекомендуємо в порядку перелічення).

5.1. Додати бінарні операції ділення на ціло // і остачу від ділення %.

5.2. Додати операцію піднесення до степеня **; звернемо увагу, що піднесення до степеня має вищий ранг, ніж операції множення і ділення.

5.3. Додати функцію sin(x) і ще одну іншу функцію.

5.4. Додати *унарні* операції + і -; їх можна записати перед будь-яким співмножником чи доданком; мають вищий ранг від бінарних операцій *, /, +, -, але нижчий від операції піднесення до степеня **; наприклад -2+-6, 4*+8; отже, якщо записані два знаки операцій підряд, тоді перший знак означає бінарну операцію, а другий – унарну.

5.5. Крім цілих чисел дозволити дійсні числа форматів фіксованої крапки і в експоненціальній формі, наприклад 45.02, 1.0e-5, 28e4; зафіксуємо вимогу, що запис будь-якого числа завжди починається з цифри.

```
arith_expr ::=  term  ( ( "+" | "-" )  term ) *
  # term ::=  factor  ( ( "*" | "/" )  factor ) *    # було
term ::=  factor  ( ( "*" | "/" | "//" | "%" )  factor ) *
  # factor ::=  number  |  "(" arith_expr ")"       # було
factor ::= [( + | - )] ( number | "(" arith_expr ")" | function )
             [( "**" factor )*]
number ::=  cipher  cipher * [ . cipher cipher * ]
               [ (e|E) [(+|-)] cipher cipher * ]
function ::=  "sin(" arith_expr ")"
cipher ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" |  "9"
```

Отже, правила змінили так, щоб зберегти раніше визначену частину правил, додати нові операції і врахувати приорітети нових операцій.

Порівняти новий варіант правил з попереднім. Зробити пропозиції до подальшого розширення формул.

6. Доопрацювати інтерпретатор `arithexpr.py` для реалізації перерахованих розширень.

Перелік можливих змін є такий:

1) доповнити сканер новими лексемами;

2) додати до методу `term()` нові бінарні операції;

3) змінивши правило для `factor`, тепер треба відповідно переписати метод `factor()`. Для цього можна повторювати в циклі (або рекурсивно) основну частину записаної раніше функції і в цій же функції обчислювати піднесення до степеня;

4) треба інакше визначити метод `onenumber()`, щоб він дозволяв форму числа цілого і дійсного;

5) варто визначити окремий метод `funcname()` для сканера, щоб виділити ім'я функції `sin` чи іншої;

6) до методу `factor()` додати можливість унарної операції + або -.

Звіт за роботу

В результаті виконання роботи надіслати:

1) програмний код файла python доопрацьованого інтерпретатора або архів проєкта, якщо реалізація виконана іншою мовою; це має бути програмна реалізація п.5 завдання - розширення змісту формул, можливий варіант якої описаний в п.6; звертаємо увагу, що програмний код треба обов'язково коментувати за змістом доопрацювання;

2) тестові приклади формул, на яких виконали перевірку інтерпретатора; перелік тестів має достатнім для повної демонстрації всіх елементів завдання;

3) роздрук отриманих результатів разом з коментарями – протокол виконання на своєму комп'ютері.