Лабораторная работа №1: Синтаксический анализатор на основе контекстно-свободной грамматики

1. Цель работы:
Разработать программу (синтаксический анализатор или "парсер"), которая проверяет соответствие входной строки заданной контекстно-свободной грамматике.

2. Теоретическая часть:
Определение КС-грамматики: G = (T, N, P, S).
Основные понятия: терминалы, нетерминалы, правила вывода, аксиома.

3. Постановка задачи:
Необходимо написать программу-анализатор для языка простых арифметических выражений со скобками и базовыми операциями. (макс. 6 баллов)

Вариант грамматики (для проверки корректности выражений):

S -> TE
E -> + TE | - T E | ε
T -> FT | *FT | / FT | ε
F -> (S) | number | id
number — последовательность цифр (например, 42, 123).
id — последовательность букв и цифр, начинающаяся с буквы (например, x, count, a1).

Входные данные:
Строка, введенная пользователем или прочитанная из файла. Например: (a + 5) * 3 - value / 2

Выходные данные:

Сообщение о том, является ли строка корректным выражением согласно заданной грамматике.

(+ 2 балла) В случае успешного разбора вывести синтаксическое дерево в виде текста (с отступами или в скобочной нотации).

4. Примеры работы программы:

Ввод: 2 + 3 * 4
Вывод: Выражение корректно.
Ввод: a * (b - 10)
Вывод: Выражение корректно.
Ввод: 5 + + 3
Вывод: Ошибка! Ожидалось: number, id или '('
Ввод: (7 * 2
Вывод: Ошибка! Ожидалось: ')'

5. Этапы выполнения:

Этап 1: Подготовка и проектирование.

Внимательно изучить грамматику. Определить множества терминалов (+, -, *, /, (,), number, id) и нетерминалов (E, E', T, T', F). Продумать, как будет реализовано чтение токенов. Можно

реализовать простую функцию getNextToken(), которая будет возвращать очередной токен из строки. Токен — это пара (тип, значение). Спроектировать функции для каждого нетерминала: parseS(), parseEPrime(), parseT(), parseTPrime(), parseF().
Этап 2: Реализация лексического анализатора.
Написать функцию, которая разбивает входную строку на токены. Использовать регулярные выражения — идеальный способ для этого.

Этап 3: Реализация синтаксического анализатора.
Реализовать функции парсера по правилам грамматики.

Функция для нетерминала F будет проверять: если текущий токен (, то вызвать match('('), затем parseS(), затем match(')'). Иначе, если токен number или id, просто "съесть" его (match()). Иначе — сообщить об ошибке.
Функция match(expected_token) сравнивает текущий токен с ожидаемым. Если совпадает — получает следующий токен. Если нет — генерирует ошибку.
Аксиомой грамматики является S. Поэтому главная функция будет вызывать parseS(), а затем проверять, что достигнут конец входной строки (получен специальный токен EOF).

Этап 4: Дополнительное задание.
Модифицировать функции парсера так, чтобы они не только проверяли синтаксис, но и возвращали фрагмент дерева разбора (в виде tuple, списка или экземпляра класса Node). Затем написать функцию для красивого вывода этого дерева.

6. Рекомендации по реализации:

Язык программирования: Python, Java, C++ или любой другой ООП язык. Python рекомендуется для быстроты реализации.
Обработка ошибок: Ошибки должны быть понятными, с указанием позиции в строке и информации о том, какой токен ожидался.
Тестирование: Обязательно создать набор тестовых строк (корректных и некорректных) для проверки работы программы.