Тестирование приложений

Замечание

Написать само приложение - это только половина задачи Одинаковый акцент нужен на двух сторонах: работа самого приложения и тесты. Все тесты должны быть автоматизированы. Должна быть возможность прогнать все тесты одной командой.

Разработка приложения и тестов должна отслеживаться системой контроля версий (git). Выполнению каждого этапа задачи должен соответствовать коммит (коммитов может быть больше чем этапов) и тэг, с тем чтобы можно было легко переключиться на соответствующую стадию и проверить её.

Например, `git checkout stage2`.

Задание

Написать небольшое приложение-калькулятор

На вход ему подаётся арифметическое выражение: числа, соединенные знаками арифметических операций. Пробелы между лексемами игнорируются: `1 + 1`, `1+ 1`, `1 +1`, `1+1` - одно и то же выражение. Но `11 + 1` и `1 1 + 1` - разные выражения (последнее некорректно). Далее на последующих этапах требование к выражению будет изменяться.

На выходе должен быть результат или сообщение об ошибке, если выражение нельзя вычислить. При ошибке, код возврата приложения должен быть отличен от 0.

В приложении должно быть как минимум 2 функции/класса: * Парсер. Принимает строку (например `2+3*5`), возвращает разобранное выражение (например, дерево `Plus(2, Mult(3, 5))`). * Вычислитель выражения. По заданному внутреннему представлению вычислить значение (наприемер `Plus(2, Mult(3, 5))` -> `17`).

Этап 1

Должна быть поддержка операций `+`, `-`, `/`, `*`, а также целых и нецелых чисел.

Вычисления должны производиться хотя бы с точностью, обеспечиваемой типом float64. Следовательно, потеря точности вычислений не является ошибкой. `"1 + 0.00000000000000000001" -> 1` - корректный результат. `"1 +`

0.000000000000000000001" -> 1.000000000000000000001 - тоже корректный результат, но требует более высокой точности вычислений.

Результат должен отображаться в виде десятичной дроби 12.3456 или в виде экспоненциальной записи 1.35e-23.

Модульные тесты

Должны быть написаны модульные тесты: * Для модуля парсинга. - Число - это выражение - Все арифметические операции и их комбинации - 1, 2 и 3-значные числа в выражениях - Некорректные/неподдерживаемые выражения: 2 ^ 4, 2 /, 1 + 4j и др. приводят к ошибке. * Для вычислителя - Все деревья выражений с двумя операциями. - Выражение должно вычисляться в ожидаемое значение: Plus(2, 2) -> 4 - Невозможные арифметические операции приводят к ошибке (Div(2, 0) -> Ошибка). - Арифметическое переполнение - должно регистрироваться как ошибка. (Div(1e300, 1e-300)).

В ходе тестов -- в этом и последующих этапах -- должны быть пройдены все ветки исполнения тестируемых функций.

Неподдерживаемые выражения не должны вычисляться.

В ходе разбора и вычисления выражения не должен исполняться произвольный код.

Например, если реализовать калькулятор через exec на Python, то существует возможность выполнить произвольный код:

```
python3 lab3.py "0; import shutil; shutil.rmtree("very/important/directory")
```

Команда попытается удалить папку very/important/directory.

Этап 2

Парсинг чисел в "научной" нотации: 1.25e+09 -> 1250000000.

Поддержка операции возведения в степень ^: 3^4 -> 81.

Поддержка скобочных выражений: 1 + 2 / (3 + 4) -> 1.2857142857142856.

Добавить соответствующие модульные тесты. Удалить/выключить неактуальные тесты.

Интеграционные тесты

evaluate(parse("1+1"))

Добавить тесты для работы двух модулей в связке: 3.375e+09^(1/3) -> [Парсер + Вычислитель] -> 1500

Ошибки парсера и вычислителя должны корректно возвращаться.

`1 /` -> [Парсер + Вычислитель] -> `Ошибка парсера: неверное выражение` `1 / 0` -> [Парсер + Вычислитель] -> `Ошибка при вычислении: деление на 0`.

Этап 3

Поддержка функций `sqrt`, `sin`, `cos`, `tg`, `ctg`, `ln`, `exp`. Поддержка констант `pi`, `e`. `sqrt(ln(e))` -> [Парсер + Вычислитель] -> `1`.

Поддержка аргументов тригонометрических функций в виде градусов и в виде радиан посредством флага (CLI). Радианы - по умолчанию.

Добавить соответствующие тесты. Удалить/выключить неактуальные тесты.

Функциональные тесты

Добавление тестов приложения в целом.

- `calc --angle-unit=degree "sin(90)"` -> `1`
- `calc --angle-unit=radian "sin(pi/2)"` -> `1`
- `calc "sin(pi/2)"` -> `1`
- `calc "sqrt(2^2 * 5 + 1)"` -> `9`
- `calc "exp(ln(2))"` -> `2`
- `calc "ln(exp(2))"` -> `2`
- `calc "ln(e^2)"` -> `2`

Нагрузочные тесты

Проверка, что программа способна выполнять запросы за определенное время и что программа не вылетает при вводе больших выражений и чисел. Время выполнения программы не должно превышать порога: например 200мс, для строк длиной до 1000 символов (независимо от корректности).

Примеры (это не полный список, дописать самостоятельно еще 10-15 вариантов):

```
time calc "1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +
1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +
```

1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1" -> "250" [0.08 s]

time calc "1 + " -> "Ошибка ..." [0.07 s]

time calc "1000+1000+1000+1000" -> "4e+249" [0.06 s]

time calc "1 ^ 36893488147419103232" -> "1" [0.03 s] time calc "1.000000000000001 ^ 36893488147419103232" -> "Ошибка: численное переполнение" [0.06 s]

time calc
"1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1
+1+1" -> "1026" [0.221 s] (Разрешено превышение, так как выражение длинее 1000)