Лабораторная работа 5: наследование

Цель работы: Реализовать интерпретатор S-выражений.

Структура проекта:

```
task06-calc
|-- .clang-format [опционально]
|-- .gitignore
|-- .gitlab-ci.yml
|-- Makefile
`-- calc.cpp
```

После выполнения команды `make` должен создаваться исполняемый файл `calc`.

Руководство

Ниже предложен один из вариантов реализации. Вы можете предложить свой способ,
но обязательно нужно соблюсти следующие требования:

1. Типы узлов абстрактного синтаксического дерева представлены классами, объединенными в иерархию.
2. Работа интерпретатора состоит из двух этапов: парсинг выражения и интерпретация.

Ваша задача — реализовать интерпретатор S-выражений. Структура такого выражения:

```
выражение = '(' op список_операндов ')'
```

Где каждый операнд — число с плавающей точкой или вложенное выражение.
Примеры таких выражений:

```
(+ 40 2)         ; 42.0000
(+ 40 1.0 1.0)   ; 42.0000

(* 3.4641 (- (+ 1
                (/ 1 (* 5 3 3))
                (/ 1 (* 9 3 3 3 3))
                (/ 1 (* 15 3 3 3 3 3 3)))
             (/ 1 (* 3 3))
             (/ 1 (* 7 3 3 3))
             (/ 1 (* 11 3 3 3 3 3))))  ; ≈3.1416
```

Приложение построчно читает выражения со стандартного потока ввода и выводит результат выражения с четырьмя знаками после точки в стандартный поток вывода.
Если выражение некорректное, вывести nan. Для простоты парсинга примем, что одна строка может содержать только одно выражение.
Такой синтаксис не очень удобен для записи формул, но значительно упрощает алгоритм парсинга, избавляя разработчика от необходимости поддерживать приоритеты операторов.

Иерархия классов

Для представления узлов абстрактного синтаксического дерева реализуйте иерархию классов:

(родительский класс)
class Expression {
    <<abstract>>
    # operands: List~Expression~
    + add_operand(expression)
    + eval(): double
}
(дочерние классы))
  Expression <|--Number
  Expression <|-- Add
  Expression <|-- Sub
  Expression <|-- Mul
  Expression <|-- Div
  Expression <|-- Invalid

Замечания о нотации

Условные обозначения:

- + — public
- # — protected
- - — private

UML-диаграммы могут составляться независимо от конкретного языка программирования. В приведенной диаграмме не отражена специфика C++:

- не используются ссылки, указатели и константы;
- вместо конкретного контейнера используется абстрактный `List`;
- не отмечены конструкторы и деструкторы.

Решения по всем этим пунктам вам следует принять самостоятельно.