Отчет

Тема: Корректность программ. Эталоны и методы проверки корректности

Теоретическая часть: Корректность программ

1. Корректность программы

Корректность программы означает, что программа делает именно то, для чего была создана — она правильно реализует заданные требования и выдает правильный результат при допустимых входных данных.

- Корректность — это свойство программы давать правильный результат при корректных входных данных.

- Надежность — способность программы работать устойчиво даже при некорректных или неожиданных входных данных или сбоях.

Примеры:

- Надежная, но некорректная программа: Программа обработки платежей, которая никогда не падает (надежная), но иногда списывает неверную сумму (некорректная).

- Корректная, но ненадежная программа: Калькулятор, который всегда считает верно (корректный), но падает при делении на ноль (ненадежный).

2. Спецификации корректности

Спецификация программы — это формальное описание того, \*что\* должна делать программа. Без спецификации невозможно точно определить, является ли программа корректной.

Виды спецификаций:

- Функциональные требования — перечень действий, которые должна выполнять программа.

- Предусловия — условия, которые должны выполняться до начала работы функции.

- Постусловия — гарантии о состоянии системы после выполнения функции.

- Инварианты — условия, которые остаются истинными на протяжении всего выполнения программы или цикла.

Для функции `calculate\_positive\_average(numbers)`:

- Предусловие:

`numbers` — это список чисел (или другая итерируемая структура, содержащая числа). Может быть пустой.

- Постусловие:

Функция возвращает среднее арифметическое всех положительных элементов списка. Если таких элементов нет, возвращается 0.

3. Методы проверки корректности

Тестирование

Суть метода: Проверка работы программы на различных наборах входных данных.

- Сильные стороны: Простота применения, возможность автоматизации, хороша для регрессионного тестирования.

- Слабые стороны: Не гарантирует отсутствия ошибок во всех возможных случаях.

- Тест-кейс — конкретный набор входных данных и ожидаемый результат.

Статический анализ

Суть метода: Анализ кода без его запуска с целью выявления потенциальных ошибок.

- Может находить: обращение к неопределенным переменным, выход за границы массива, типовые ошибки.

- Пример инструмента: `pylint`, `flake8`, `SonarQube`.

Формальные методы (доказательство корректности)

Идея: Использование математических методов для доказательства того, что программа соответствует своей спецификации.

- Инвариант цикла — условие, которое остается истинным перед началом и после каждого шага цикла. Нужен для доказательства корректности циклических алгоритмов.

Сравнение методов

|  |  |  |
| --- | --- | --- |
| Метод | Подходит для маленькой функции? | Подходит для большой системы? |
| Тестирование | ✅ Да | ✅ Да |
| Статический анализ | ✅ Да | ✅ Очень полезен |
| Формальные методы | ❌ Нет | 🔁 Иногда (в критических системах) |

Вывод:

- Для небольшой функции достаточно тестирования и статического анализа.

- Для больших систем необходимы статический анализ, покрытие тестами и, возможно, частичное применение формальных методов в критически важных модулях.

Практическая часть:

Отчет по анализу функции `calculate\_positive\_average`

1. Спецификация

Предусловия:

- `numbers` — это \*\*итерируемый объект\*\* (например, список, кортеж), содержащий числовые значения (`int` или `float`).

- Элементы списка могут быть положительными, отрицательными или равными нулю.

- Список может быть пустым.

> Функция \*\*не проверяет тип входных данных\*\*, поэтому ожидается, что вызывающий код соблюдает контракт.

Постусловие:

- Если в списке есть хотя бы одно положительное число, функция возвращает среднее арифметическое всех положительных чисел.

- Если в списке нет положительных чисел (все элементы ≤ 0 или список пустой), функция возвращает 0.

Инвариант цикла:

Да, инвариант есть:

На каждой итерации цикла:

- `total` равно сумме всех положительных чисел из `numbers`, обработанных на предыдущих шагах.

- `count` равно количеству этих положительных чисел.

Этот инвариант поддерживается условием `if num > 0`.

2. Ручное тестирование

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные (`numbers`) | Ожидаемый результат | Обоснование |
| 1 | `[2, -3, 4, 0, 5]` | `(2 + 4 + 5) / 3 = 3.666...` → `3.67` (округлено) | Обычный случай: есть положительные числа |
| 2 | `[-1, -2, 0]` | `0` | Граничный случай: нет положительных чисел |
| 3 | `"not a list"` | Ошибка (например, исключение) | Нарушение предусловия |

3. Анализ корректности и ошибок

### Строка A: `average = total / count`

- \*\*Гарантированно не произойдет деления на ноль\*\*, потому что перед этой строкой стоит проверка:

```python

if count > 0:

```

- Это условие \*\*обеспечивает безопасность\*\* операции деления.

### Обработка пустого списка

- При `numbers = []`:

- Цикл не выполняется.

- `count` остается равным `0`.

- Условие `count > 0` ложно → возвращается `0`.

Соответствует спецификации: да, функция возвращает `0`, если нет положительных чисел.

Возможная ошибка при нарушении предусловия

Если вместо списка передать, например, строку:

```python

calculate\_positive\_average("abc")

```

- Произойдет ошибка: `TypeError` — нельзя итерироваться по строке в данном контексте.

Как улучшить функцию?

1. \*\*Добавить проверку типа:\*\*

```python

if not isinstance(numbers, (list, tuple)):

raise TypeError("Ожидается список или кортеж чисел.")

```

2. \*\*Проверять тип каждого элемента:\*\*

```python

for num in numbers:

if not isinstance(num, (int, float)):

raise ValueError("Все элементы должны быть числами.")

```

4. Выводы

Корректна ли функция относительно спецификации?

Да, функция корректна\*\* относительно своей спецификации:

- Она вычисляет среднее только положительных чисел.

- Возвращает `0`, если таких чисел нет.

- Все тестовые случаи проходят успешно.

- Деление на ноль исключено.

- Пустой список обрабатывается корректно.

Какой метод наиболее эффективен для обнаружения ошибки, если бы не было проверки `count > 0`?

Тестирование — самый эффективный метод в данном случае.

- Позволяет быстро обнаружить ошибку деления на ноль через тестовый сценарий, где `count == 0`.

- Легко воспроизводится и автоматизируется.

- В отличие от статического анализа, позволяет проверить поведение программы во время выполнения.

- Формальные методы в данном случае избыточны для такой простой функции.