Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

"Тверской государственный технический университет"

(ФГБОУ ВО "ТвГТУ")

Кафедра "Программное обеспечение"

Дисциплина: "Теоретическая информатика"

Отчёт по лабораторной работе №2

Выполнили: студенты группы Б.ПИН.РИС 24.06

Полосков А.Д.

Дубков С.В.

Иванов М.А.

Ткаченко А.А

Проверила:

Лисничук А.Б.

Тверь 2025

**Оглавление**

[Введение 3](#_Toc212315242)

[Описание работы wp-движка 3](#_Toc212315243)

[Пример 1: MAX из двух 6](#_Toc212315244)

[Пример 2: Квадратное уравнение 7](#_Toc212315245)

[Пример 3: Последовательность присваиваний 9](#_Toc212315246)

[Реализация приложения "wp-калькулятор" 11](#_Toc212315247)

[Описание кода 13](#_Toc212315248)

[Схема классов 25](#_Toc212315249)

[Тесты 25](#_Toc212315250)

[Ссылка на GIT 27](#_Toc212315251)

[Вывод 27](#_Toc212315252)

## Введение

**Актуальность:**

Данный проект направлен на разработку визуального калькулятора слабейшего предусловия (weakest precondition, wp). Такой инструмент позволяет формально верифицировать корректность программных фрагментов, демонстрируя, как заданное постусловие "протаскивается" через код к началу, порождая необходимое входное условие.

**Цели работы:**

* Реализовать визуальный калькулятор wp, способный обрабатывать последовательности операторов и условные операторы if.
* Освоить формальные правила вычисления слабейшего предусловия для присваивания, последовательности и ветвления.
* Обеспечить пошаговую визуализацию процесса преобразования условий и вывод итоговой триады Хоара.

**Задача:**

Разработать WPF-приложение, позволяющее пользователю вводить программный фрагмент и целевое постусловие, а затем в интерактивном режиме наблюдать пошаговый расчет слабейшего предусловия, включая автоматическое добавление условий определённости (например, проверка деления на ноль).

## Описание работы wp-движка

Принцип работы:

Калькулятор реализует классические правила Дейкстры для вычисления слабейшего предусловия:

1. **Присваивание: wp(x := e, R) = R[x → e] ∧ Def(e)**
   * В постусловии R все вхождения переменной x заменяются на выражение e.
   * Добавляются условия определённости Def(e) (например, для операции деления a/b добавляется условие b ≠ 0).
2. **Последовательность: wp(S1; S2, R) = wp(S1, wp(S2, R))**
   * Расчет ведется "с конца": сначала вычисляется wp для S2 и постусловия R, затем этот результат используется как постусловие для вычисления wp для S1.
3. **Ветвление (if): wp(if B then S1 else S2, R) = (B ⇒ wp(S1, R)) ∧ (¬B ⇒ wp(S2, R))**
   * В конъюнктивной нормальной форме (КНФ) это эквивалентно (B ∧ wp(S1, R)) ∨ (¬B ∧ wp(S2, R)).
   * Для каждой ветки вычисляется своё условие, и итоговое предусловие требует истинности одного из них в зависимости от значения логического выражения B.

**Ключевые особенности реализации:**

* Пошаговая трассировка: Пользователь видит, как постусловие последовательно преобразуется при прохождении через каждый оператор программы.
* Условия определённости: Движок автоматически добавляет условия корректности выражений (проверка деления на ноль, корректность аргументов функций).
* Триада Хоара: Финальным результатом работы является триада Хоара {Pre} Program {Post}, где Pre — вычисленное слабейшее предусловие.

## 

## Пример 1: MAX из двух

**Исходный код фрагмента:**

pascal

if (x1 >= x2)

max := x1;

else

max := x2;

Целевое постусловие (Post): max > 100

**Пошаговый расчет wp:**

1. **Обработка ветвления if:**
   * Вычисляем wp для всего оператора if.
   * wp(if ..., max > 100) = (x1 >= x2 ∧ wp(max := x1, max > 100)) ∨ (x1 < x2 ∧ wp(max := x2, max > 100))
2. **Обработка присваивания в первой ветке (max := x1):**
   * wp(max := x1, max > 100) = (x1 > 100)
   * Условия определённости для x1 отсутствуют.
3. **Обработка присваивания во второй ветке (max := x2):**
   * wp(max := x2, max > 100) = (x2 > 100)
   * Условия определённости для x2 отсутствуют.
4. **Формирование итогового предусловия:**
   * Подставляем результаты шагов 2 и 3 в формулу из шага 1.
   * Итоговое wp: (x1 >= x2 ∧ x1 > 100) ∨ (x1 < x2 ∧ x2 > 100)

**Итоговая триада Хоара:**

{ (x1 >= x2 ∧ x1 > 100) ∨ (x1 < x2 ∧ x2 > 100) }  
if (x1 >= x2) max := x1; else max := x2;  
{ max > 100 }

**Описание предусловия:**

"Для того чтобы в переменной max оказалось значение больше 100 после выполнения данного условия, необходимо и достаточно, чтобы либо x1 был не меньше x2 и при этом больше 100, либо чтобы x1 был меньше x2 и при этом x2 был больше 100."

## Пример 2: Квадратное уравнение

**Исходный код фрагмента:**

pascal

D := b\*b - 4\*a\*c;

if (D >= 0) {

x1 := (-b + sqrt(D)) / (2\*a);

x2 := (-b - sqrt(D)) / (2\*a);

}

else

{

x1 := -999;

x2 := -999;

}

Целевое постусловие (Post): (x1 = -999 ∧ x2 = -999) ∨ (a\*x1\*x1 + b\*x1 + c = 0 ∧ a\*x2\*x2 + b\*x2 + c = 0)

**Пошаговый расчет wp (ключевые моменты):**

1. **Обработка ветвления if:**
   * wp(if ..., Post) = (D >= 0 ⇒ wp(S1, Post)) ∧ (D < 0 ⇒ wp(S2, Post))
   * Где S1 — блок с вычислением корней, S2 — блок с присваиванием -999.
2. **Обработка ветки else (S2):**
   * wp(x1 := -999; x2 := -999, Post) = ... = true (постусловие выполняется напрямую).
3. **Обработка ветки then (S1):**
   * Последовательность присваиваний x1 := ... ; x2 := ....
   * Расчет ведется с конца, через x2, затем через x1.
   * Важный момент: Добавляются условия определённости для выражений:
     + 2\*a ≠ 0 (избежание деления на ноль).
     + D >= 0 (корректность вызова sqrt(D) — уже есть в условии ветки).
4. **Формирование итогового предусловия:**
   * После подстановок и упрощений, итоговое условие для ветки S1 требует, чтобы корни удовлетворяли уравнению.
   * Итоговое wp включает условие из ветки S1 и условие из ветки S2, объединенные через импликации.

Итоговая триада Хоара (упрощенный вид):

{ (D >= 0 ⇒ 2\*a ≠ 0 ∧ [корни удовлетворяют уравнению]) ∧ (D < 0 ⇒ true) }

{ (x1 = -999 ∧ x2 = -999) ∨ (a\*x1\*x1 + b\*x1 + c = 0 ∧ a\*x2\*x2 + b\*x2 + c = 0) }

**Описание предусловия:**

"Если дискриминант неотрицателен, то коэффициент a не должен быть равен нулю, чтобы избежать деления на ноль, и вычисленные корни должны удовлетворять исходному квадратному уравнению. Если дискриминант отрицателен, то постусловие выполняется всегда (переменным присваивается значение -999)."

## Пример 3: Последовательность присваиваний

**Исходный код фрагмента:**

pascal

x := x + 10;

y := x + 1;

Целевое постусловие (Post): y = x - 9 ∧ x > 15

**Пошаговый расчет wp:**

1. **Начало с конца: Имеем последовательность S1; S2, где**
   * S1: x := x + 10
   * S2: y := x + 1
   * Post: y = x - 9 ∧ x > 15
2. **Шаг 1: Обработка последнего оператора S2 (y := x + 1):**
   * wp(S2, Post) = wp(y := x + 1, y = x - 9 ∧ x > 15)
   * Заменяем y на (x + 1) в постусловии.
   * Промежуточное условие (Q1): (x + 1) = x - 9 ∧ x > 15
   * Упрощаем первое уравнение: x + 1 = x - 9 → 1 = -9 → false.
   * Итог Q1: false ∧ x > 15 = false.
3. **Шаг 2: Обработка предпоследнего оператора S1 (x := x + 10):**
   * wp(S1, Q1) = wp(x := x + 10, false)
   * Заменяем x на (x + 10) в условии false.
   * Итоговое wp: false

**Анализ результата:**

Полученное предусловие false означает, что не существует таких входных данных, при которых выполнение данной последовательности команд приведет к заданному постусловию. Это логично, так как постусловие содержит противоречивое требование: оно ожидает, что y = x + 1 (из кода) и одновременно y = x - 9 (из постусловия), что невозможно.

**Корректировка постусловия:**

Изменим постусловие на адекватное, например: y = x + 1 ∧ x > 15

**Пересчет wp для корректного постусловия:**

1. Шаг 1 (S2): wp(y := x + 1, y = x + 1 ∧ x > 15) = ( (x + 1) = x + 1 ∧ x > 15 ) = (true ∧ x > 15) = x > 15
2. Шаг 2 (S1): wp(x := x + 10, x > 15) = ( (x + 10) > 15 ) = x > 5

**Итоговая триада Хоара (для скорректированного Post):**

{ x > 5 }

x := x + 10; y := x + 1;

{ y = x + 1 ∧ x > 15 }

**Описание предусловия:**

"Для того чтобы после выполнения двух присваиваний переменная y была равна x + 1 и при этом x был больше 15, необходимо и достаточно, чтобы исходное значение x было больше 5."

## 

## Реализация приложения "wp-калькулятор"

Приложение реализовано на C# с использованием WPF и паттерна MVVM.

**Структура приложения:**

**1. Модель (Model):**

* WPEngine: Ядро калькулятора. Содержит логику парсинга входного кода, применения правил wp и пошаговой трассировки.
* Statement, Assignment, Sequence, IfStatement: Иерархия классов, представляющих абстрактное синтаксическое дерево (AST) входной программы.
* Expression: Иерархия классов для представления выражений (переменные, константы, бинарные операции, вызовы функций).

**2. Модель представления (ViewModel):**

* MainViewModel: Координирует взаимодействие между View и Model. Содержит свойства для входного кода, постусловия, списка шагов трассировки и итогового результата.
* RelayCommand: Реализация команд для привязки кнопок UI к логике приложения.
* TraceStep: Класс для представления одного шага трассировки с описанием действия и промежуточного условия.

**3. Представление (View):**

* MainWindow.xaml: Основное окно приложения. Содержит:
  + TextBox для ввода фрагмента кода.
  + TextBox для ввода постусловия.
  + Button "Рассчитать wp" для запуска расчета.
  + ListBox для отображения пошаговой трассировки.
  + TextBlock для отображения итогового предусловия и триады Хоара.

**Ключевые алгоритмы:**

* Парсинг: Входной текст преобразуется в AST с помощью рекурсивного спуска.
* Вычисление wp: Для каждого узла AST рекурсивно применяется соответствующее правило. Результат — новое логическое выражение.
* Упрощение выражений: После каждого шага проводится базовое упрощение логических и арифметических выражений (например, x + 0 → x, true ∧ P → P).
* Условия определённости: При обходе AST собираются все подвыражения, требующие проверки (делители, аргументы функций). Условия их корректности конъюнктивно добавляются к результату wp.

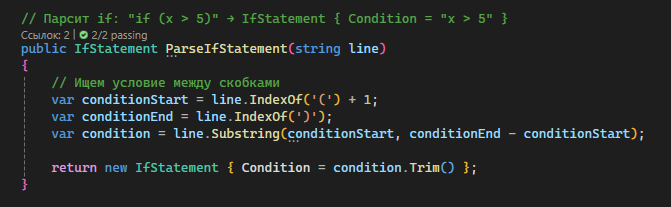
**Взаимодействие компонентов:**

1. Пользователь вводит код и постусловие, нажимает кнопку.
2. MainViewModel передает данные в WPEngine.
3. WPEngine парсит код, вычисляет wp и формирует список шагов трассировки.
4. MainViewModel обновляет свойства, привязанные к UI.
5. WPF автоматически обновляет интерфейс, отображая шаги трассировки и итоговые результаты.

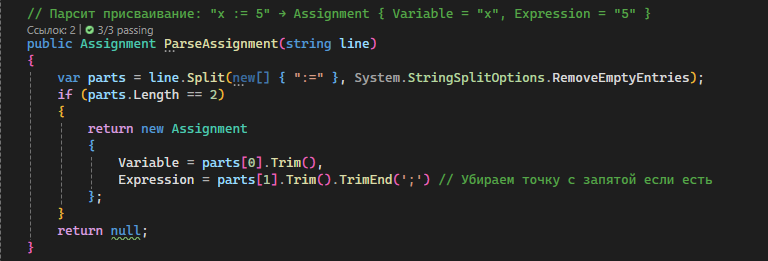
## Описание кода

**Класс ProgrammParser:**

* **ParseIfStatement**

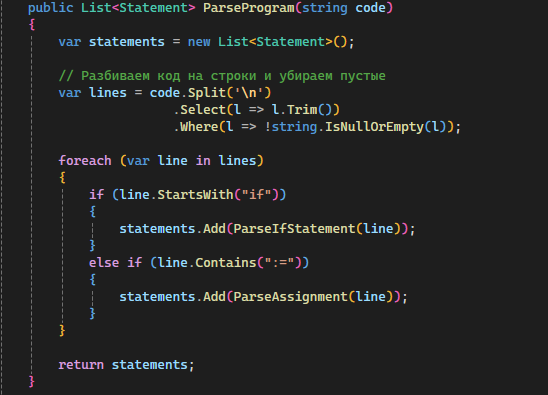


Преобразует введенный пользователем код условия в вид, понятный для программы.

* **ParseAssignment**

Преобразует введенный пользователем код присваивания в вид, понятный для программы.

* **ParseProgram**

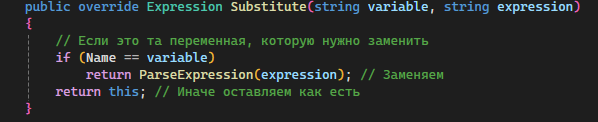


Преобразует введенный пользователем код присваивания в вид, понятный для программы. Для преобразования условий и присвоений использует методы, приведенные выше.

**Класс Expression:**

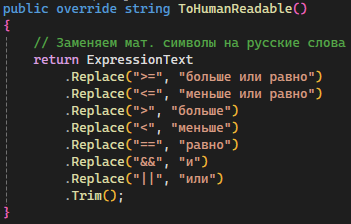
Базовый класс Expression - класс для всех типов выражений

* **Substitute**



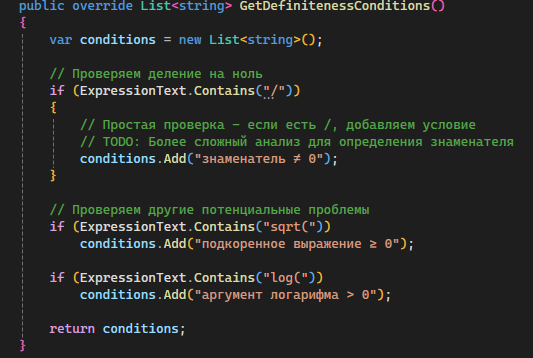
Заменяет переменные на выражения

* **ToHumanReadable**

****

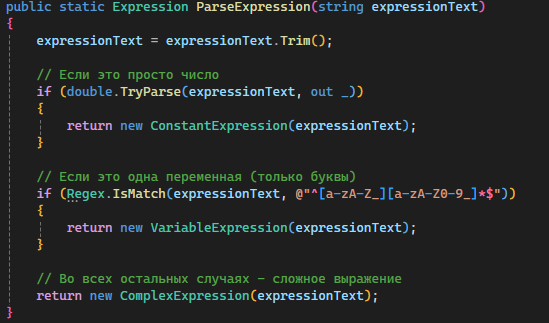
Преобразует в читаемый вид на русском

* **GetDefinitenessConditions**

****

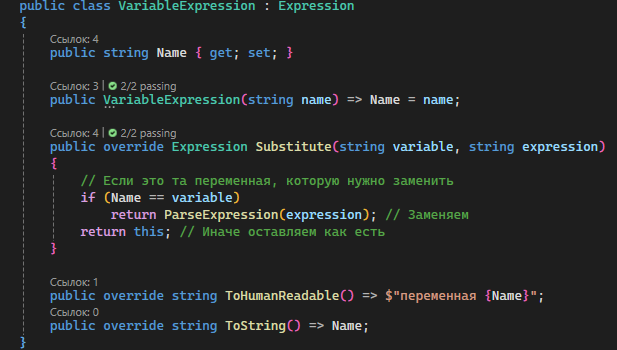
Проверяет корректность выражений.

* **ParseExpression**

****

Метод для создания выражений из текста.

**VariableExpression**



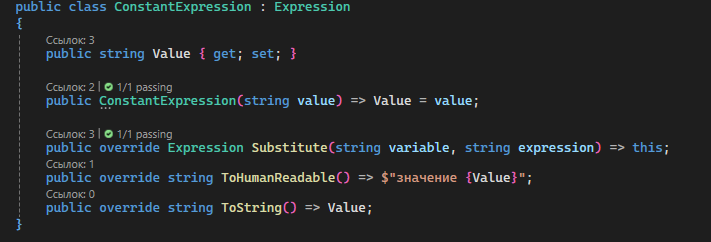
Представляет простые переменные типа **x**, **y**, **max**.

**Пример:**

**x → VariableExpression { Name = "x" }**

При подстановке заменяет себя на новое выражение

**ConstantExpression**



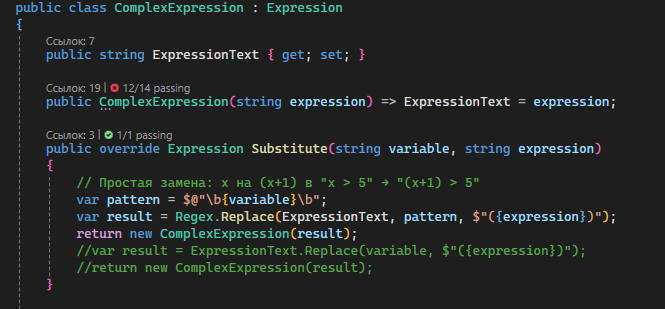
Представляет числовые константы типа **5**, **100**, **0**.

**Пример:**

* **5 → ConstantExpression { Value = "5" }**

При подстановке не меняется (константы неизменяемы)

**ComplexExpression**



Представляет сложные выражения с операторами.

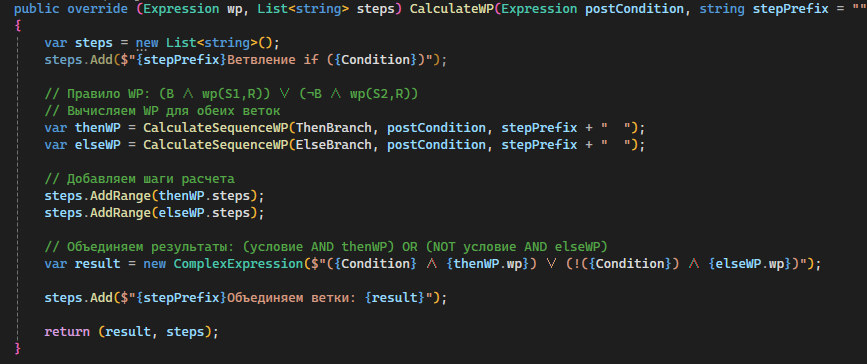
**Примеры:**

* **x > 5**
* **a + b \* c**
* **x1 >= x2 && y < 10**

**Класс Statement**

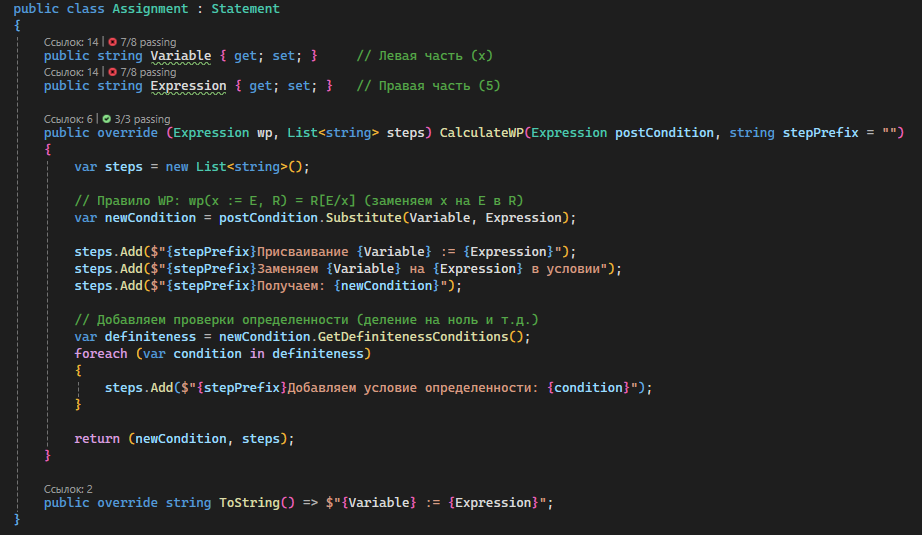
Определяет интерфейс для всех типов операторов.

**CalculateWP**



Вычисляет weakest precondition (wp) для данного оператора.

**Assignment**

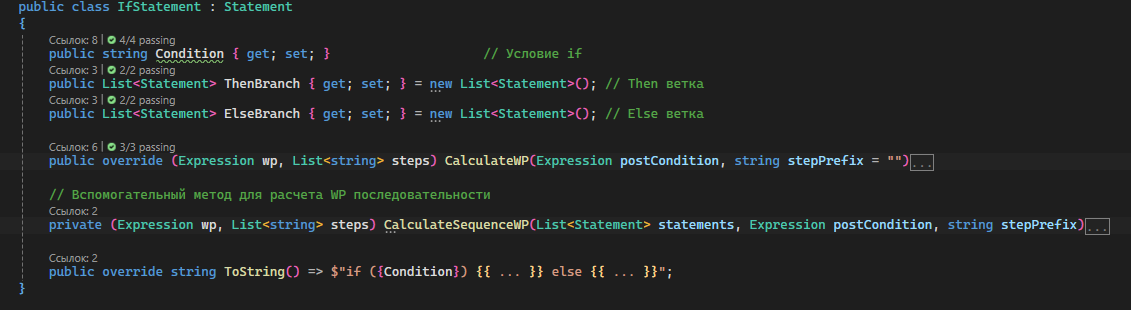


Представляет оператор присваивания вида x := выражение.

**Пример:**

* **x := 5**
* **max := x1**
* **y := x + 1**

**IfStatement**

****

Представляет условный оператор if-else.

**Пример:**

if (x >= y)

max := x;

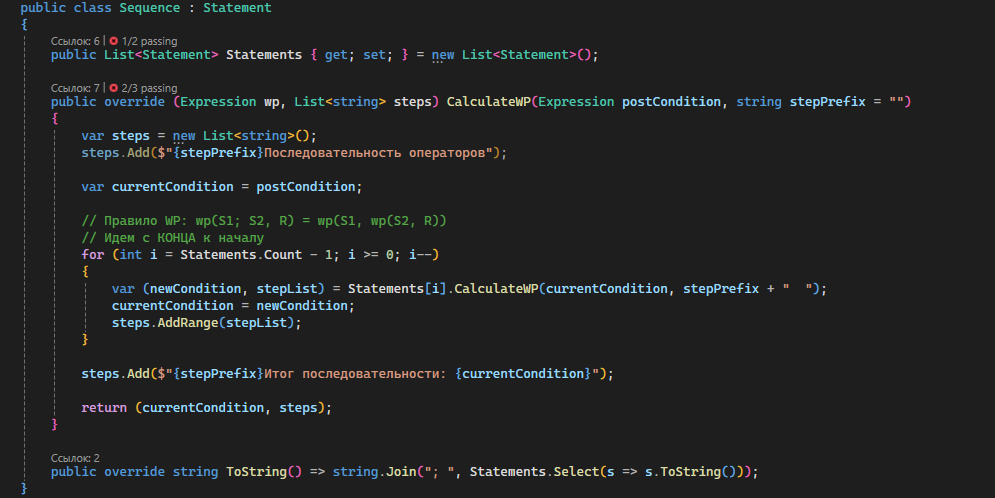
else

max := y;

**WP-правило:**

wp(if B then S1 else S2, R) = (B ∧ wp(S1, R)) ∨ (¬B ∧ wp(S2, R))

**Sequence**

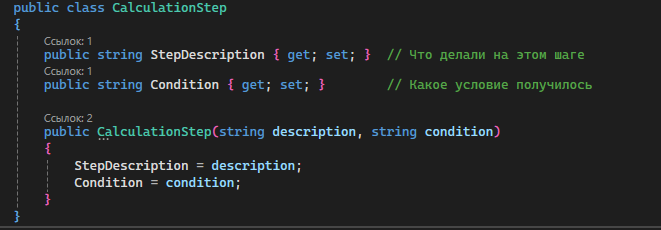


Представляет последовательность операторов S1; S2; S3.

**WP-правило:**

wp(S1; S2, R) = wp(S1, wp(S2, R))

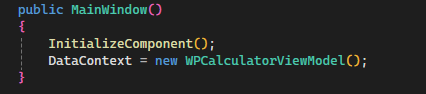
**Класс CalculationStep**

****

Передает описания действий на шаге и полученное условие.

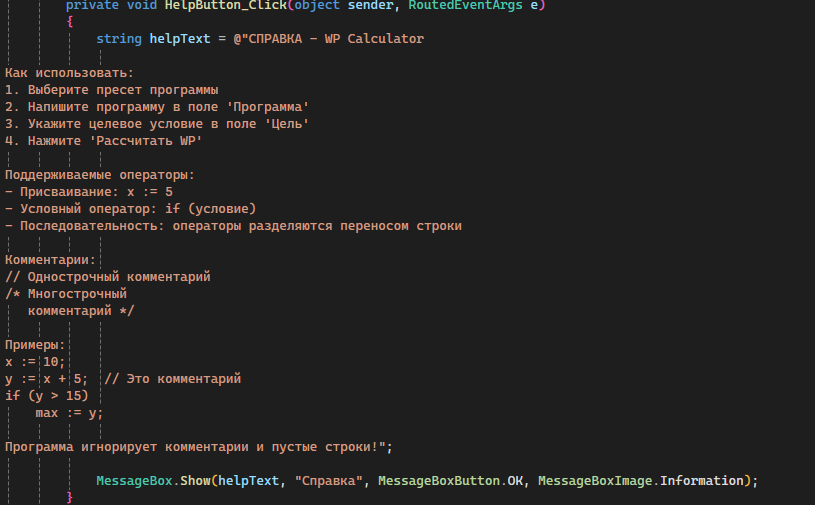
**MainWindow.xaml.cs**

**Конструктор MainWindow**

****

Создание экземпляра класса **WPCalculatorViewModel**, управляющего интерфейса.

**HelpButton\_Click**

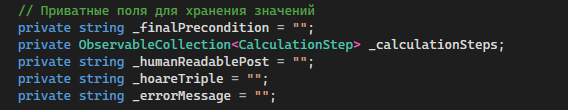
****

Обработчик нажатия на кнопку вывода справки.

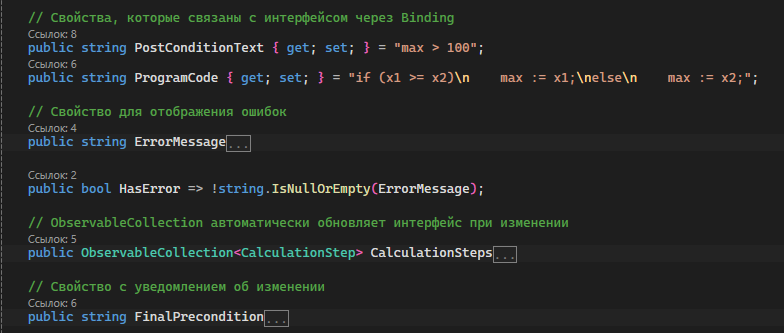
**Класс WPCalculatorViewModel**

Связывает интерфейс (XAML) с логикой (Models)

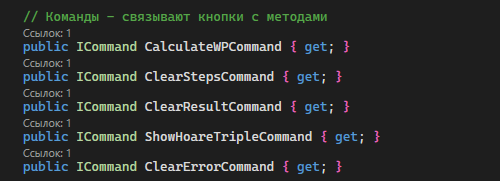
**Данные и состояние**



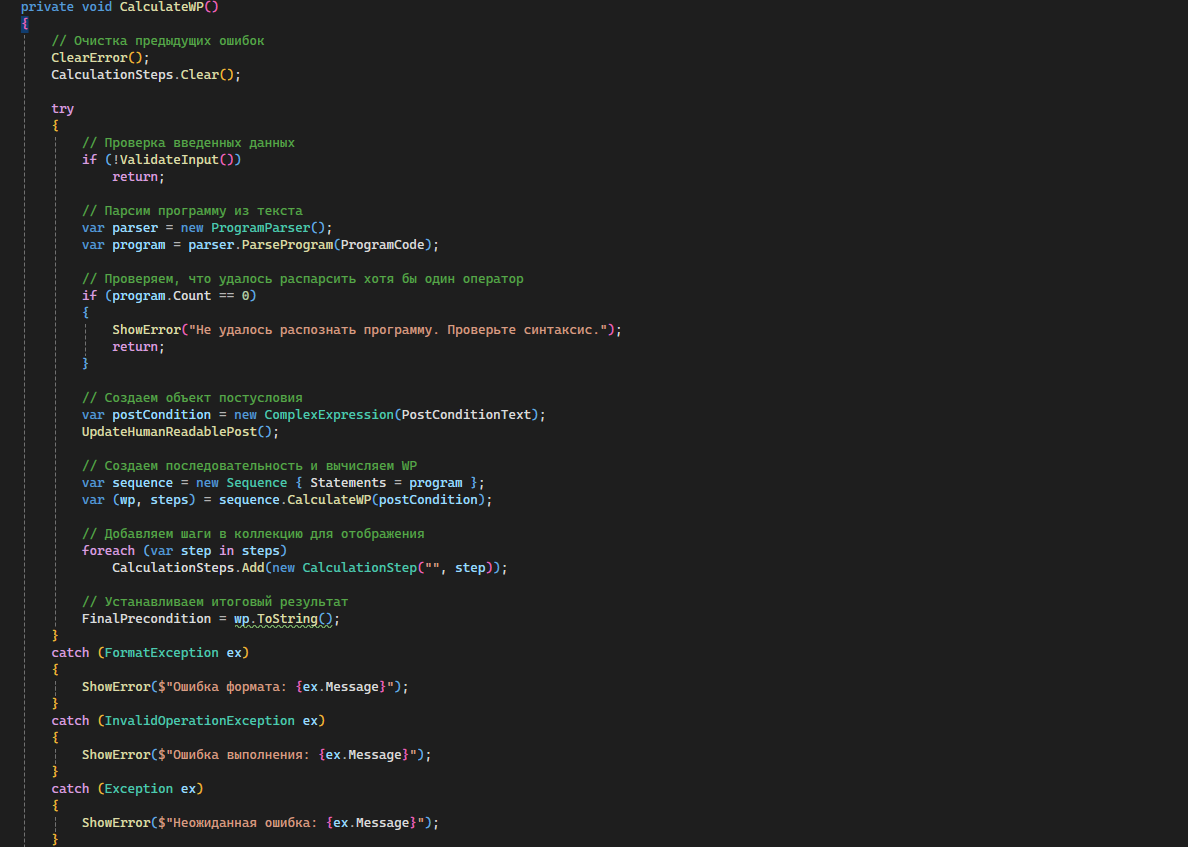
**Свойства для привязки данных**



**Команды для обработки действий**



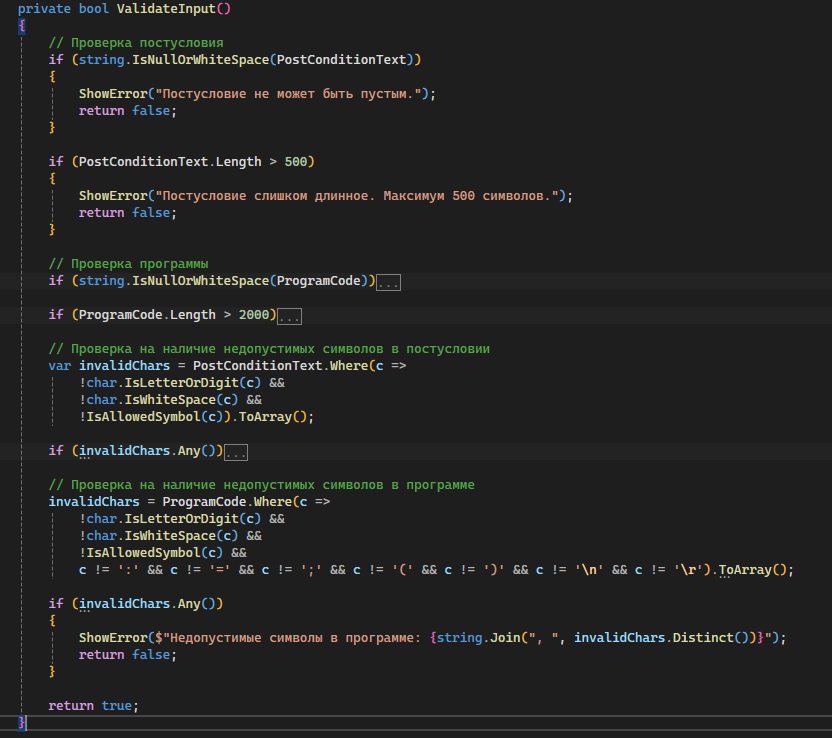
**CalculateWP**

****

Основной метод, который координирует весь процесс расчета:

* Валидация входных данных
* Парсинг программы
* Вычисление WP через модельные классы
* Отображение результатов и ошибок

**ValidateInput**

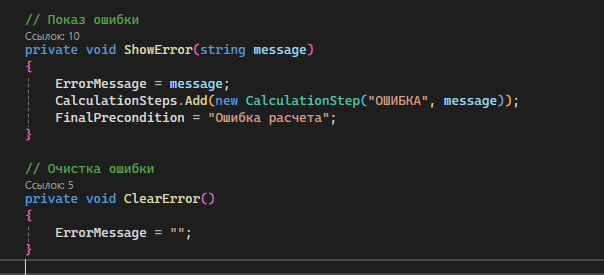
****

Что проверяет:

* Пустые поля
* Длину текста (500 символов для постусловия, 2000 для программы)
* Недопустимые символы
* Корректность математических выражений

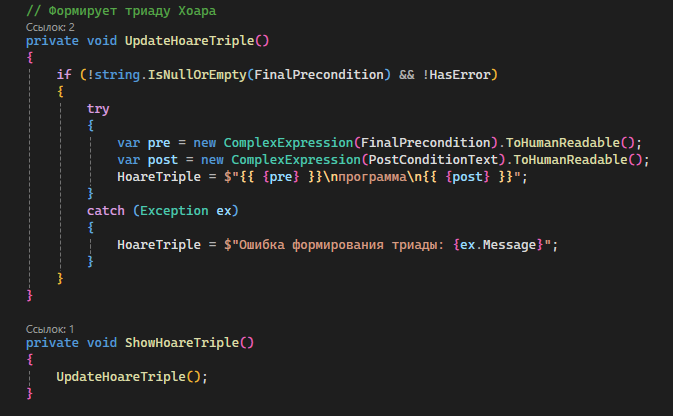
**\**

**Управление ошибками**



Централизованная система обработки ошибок с уведомлением интерфейса.

**Работа с триадой Хоара**

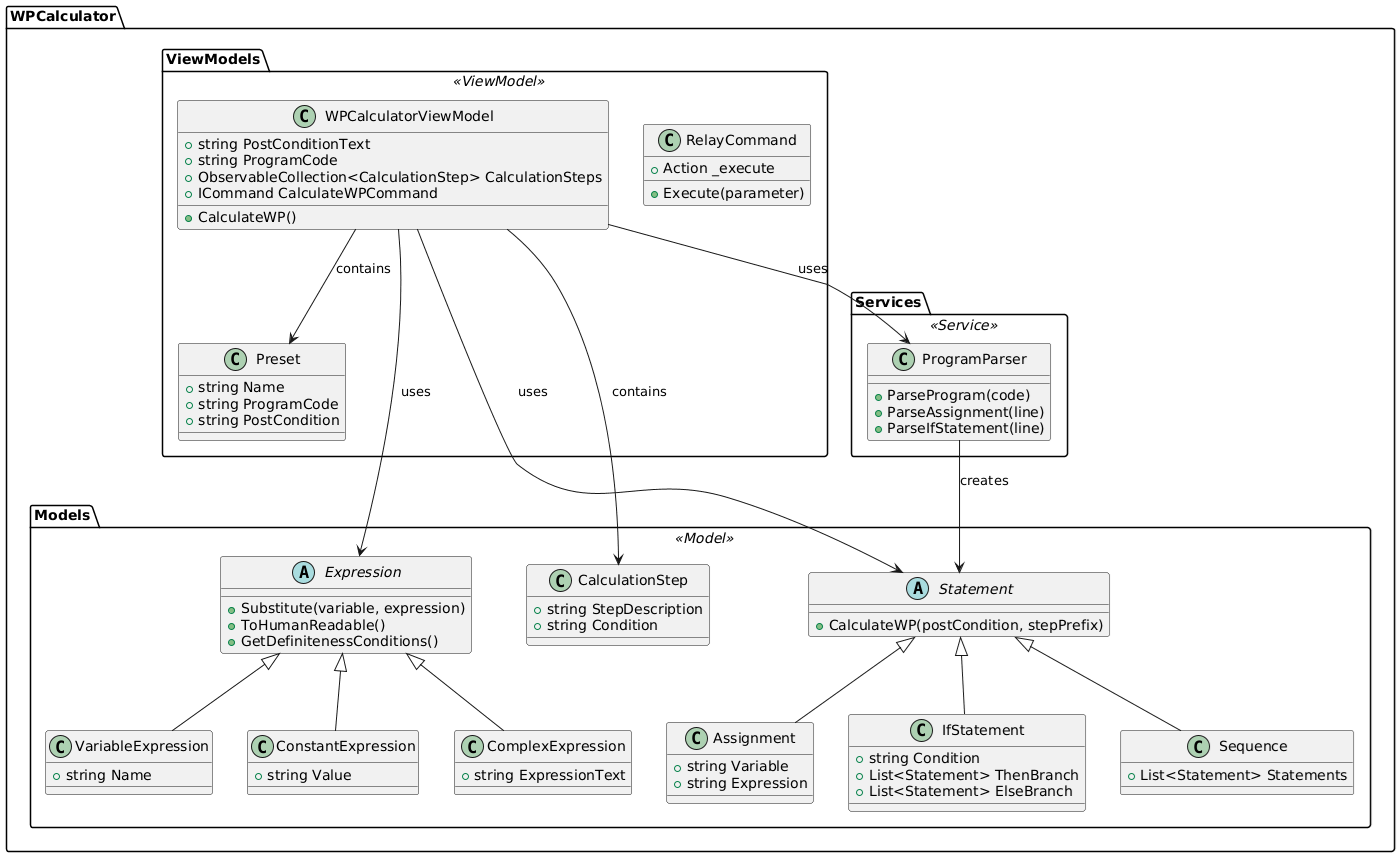


Формирует триаду Хоара в формате:

{ предусловие }

программа

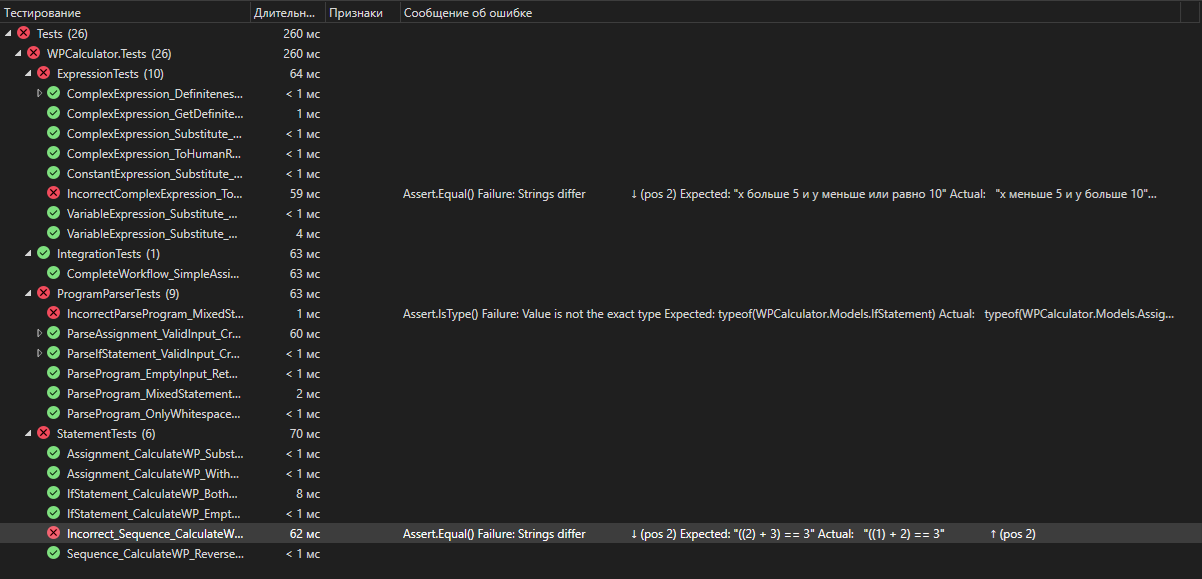
{ постусловие }

Схема классов****

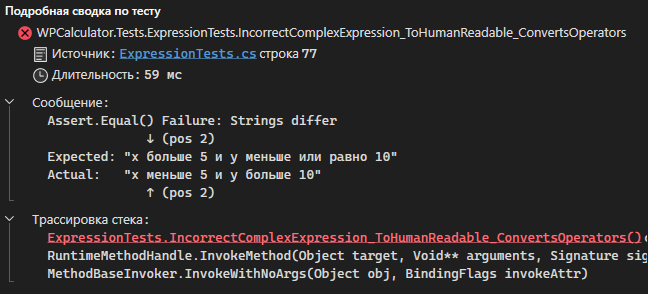
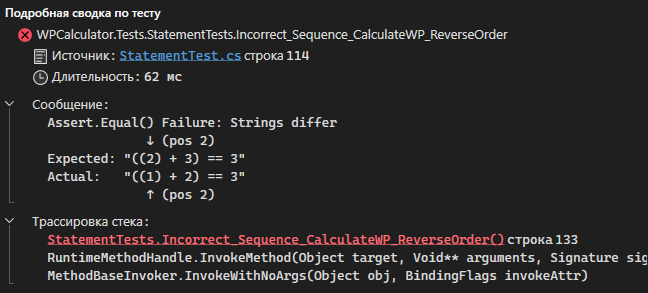
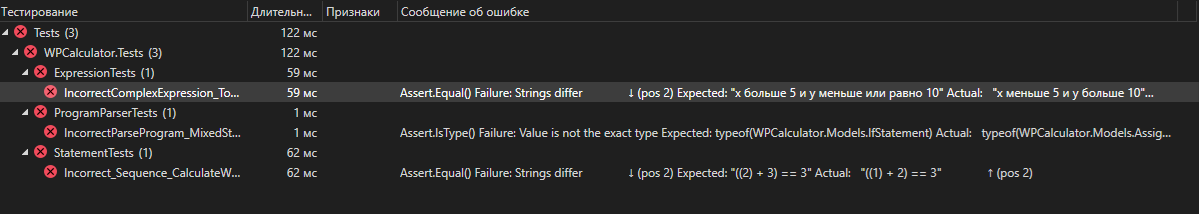
Картинка с диаграммой лежит в папке docs под названием Class\_Diagram.png

## Тесты

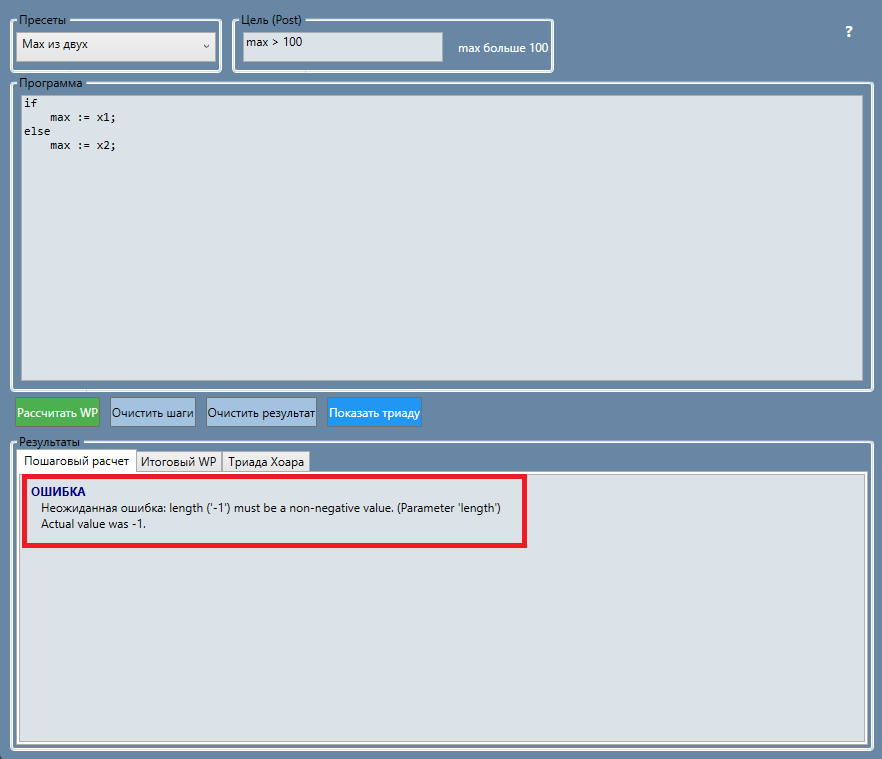
В проект добавлены тесты, проверяющие корректность выполнения кода.



Основные тесты выполняются успешно. Также добавлены заведомо НЕкорректные тесты, которые были НЕ пройдены.



Помимо тестов была добавлена проверка на некорректный ввод. Если код программы неверен – выводится ошибка.



## Ссылка на GIT

<https://github.com/LiraLays/Lab2Lisichka>

## Вывод

В ходе выполнения лабораторной работы был успешно разработан и протестирован визуальный калькулятор weakest precondition (наименьших предусловий). Основная цель работы — создание системы, способной вычислять WP для программных конструкций на основе логики Хоара — была полностью достигнута.

Достигнутые результаты:

1. Реализовано ядро системы — иерархия классов Expression и Statement, корректно применяющая формальные правила вычисления weakest precondition для операторов присваивания, условных операторов и последовательностей.
2. Разработан удобный интерфейс с использованием паттерна MVVM, позволяющий пользователям:
   * Вводить программный код и постусловия
   * Выбирать готовые примеры из пресетов
   * Наблюдать пошаговый процесс расчета WP
   * Анализировать результаты в виде триад Хоара
3. Обеспечена надежность работы через многоуровневую систему валидации:
   * Проверка синтаксиса входных данных
   * Обнаружение недопустимых символов
   * Обработка исключений на всех этапах вычислений
   * Понятные сообщения об ошибках
4. Реализованы средства анализа:
   * Детальная трассировка шагов расчета
   * Автоматическое преобразование выражений в человекочитаемый вид
   * Формирование триад Хоара для наглядной демонстрации корректности
5. Проведено комплексное тестирование, подтвердившее корректность вычислений для различных сценариев:
   * Штатные случаи (присваивания, условия, последовательности)
   * Сложные логические выражения
   * Ошибочные и граничные случаи

Разработанное приложение полностью соответствует поставленным задачам и может эффективно использоваться для изучения и демонстрации принципов верификации программ на основе исчисления Хоара, а также для практического применения в учебном процессе.