МИНОБОРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Ижевский государственный технический университет
имени М.Т. Калашникова»
Институт «Информатики и вычислительной техники»
Кафедра «Программное обеспечение»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

По дисциплине «Тестирование ПО»

Вариант 22

Выполнил
студент гр. Б22-191-1зу Н.А. Бобров

Руководитель
Старший преподаватель кафедры
«Программное обеспечение» Е.В. Старыгина

Ижевск 2025

Оглавление

2. Задание

Дан массив действительных чисел $a_0,...,a_{n-1}$. Вычислить сумму отрицательных и произведение нечетных элементов данного массива.

Требования:

- Язык программирования: C#
- Тестирование: xUnit, Moq
- Автоматизация тестов: GitHub Actions
- Размер массива: до 1024 элементов
- Тестирование на основе потокового и информационного графов, областей эквивалентности и условий.

3. Блок-схема и описание алгоритма

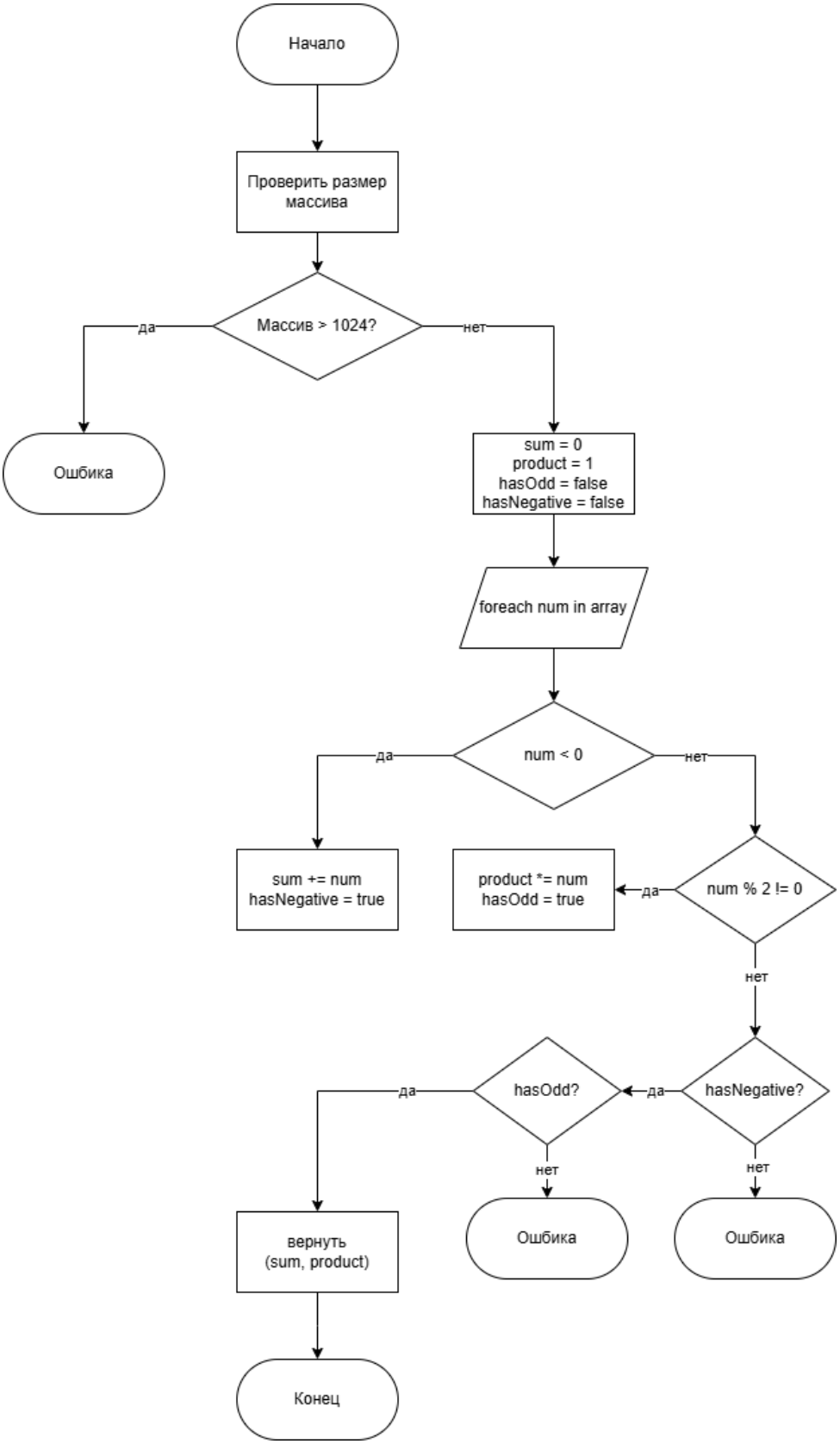

Рисунок 1

Описание алгоритма:

1. Инициализируются переменные:
    a. sum — для хранения суммы отрицательных элементов (начально 0),
    b. product — для хранения произведения нечётных элементов (начально 1),
    c. флаги hasNegative и hasOdd — для отслеживания наличия хотя бы одного отрицательного и нечётного элемента соответственно.
2. Проверяется размер массива:
    a. Если размер больше 1024 — выбрасывается исключение.
3. Проходим по каждому элементу массива:
    a. Если элемент отрицательный, добавляем его к sum и устанавливаем hasNegative = true.
    b. Если элемент нечётный (остаток от деления на 2 не равен 0), умножаем его на product и устанавливаем hasOdd = true.
4. После обработки всех элементов:
    a. Если не найдено ни одного отрицательного элемента, выбрасывается исключение.
    b. Если не найдено ни одного нечётного элемента, также выбрасывается исключение.
5. Возвращается кортеж из sum и product.

4. Тестирование базового пути

4.1. Потоковый граф

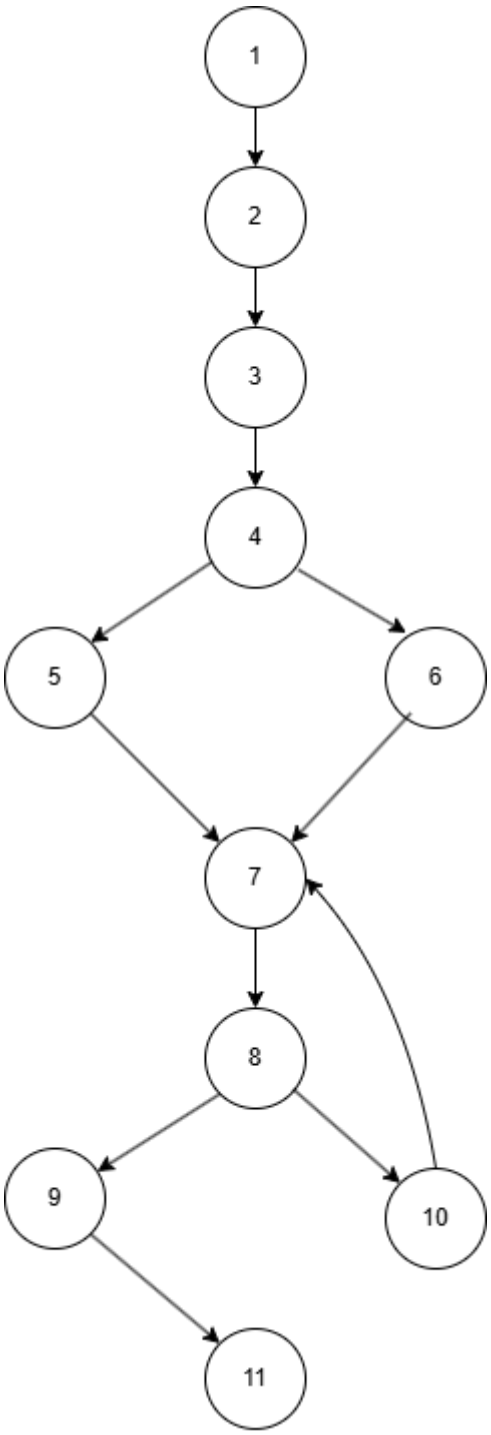

Рисунок 2

4.2. Цикломатическая сложность

Расчёт цикломатической сложности, несколько методов

1) $V(G) = E - N + 2$, где:

- E — количество ребер.
- N — количество узлов.

Для данного графа $V(G) = 12 - 11 + 2 = 3$.

2) потоковый граф имеет 3 региона

3) 2 предикатных узла + 1

4.3. Базовое множество независимых линейных путей

1. Путь 1: Массив пустой.
2. Путь 2: Массив содержит только положительные элементы.
3. Путь 3: Массив содержит только отрицательные элементы.
4. Путь 4: Массив содержит только четные элементы.
5. Путь 5: Массив содержит только нечетные элементы.
6. Путь 6: Массив содержит смешанные элементы.

4.4. Тестовые варианты

Таблица 1

Исходные данные (ИД)	**Ожидаемые результаты (ОЖ.РЕЗ.)**
[-1, 2, -3, 4, 5]	(-4, 15)
[-2, 2, 4]	Исключение
[1025 элементов]	Исключение
[]	Исключение
[-1, 2, 0, 3, -4, 5]	(-5, -15)

5. Тестирование потоков данных

5.1. Информационный граф

Информационный граф показывает зависимости между данными.

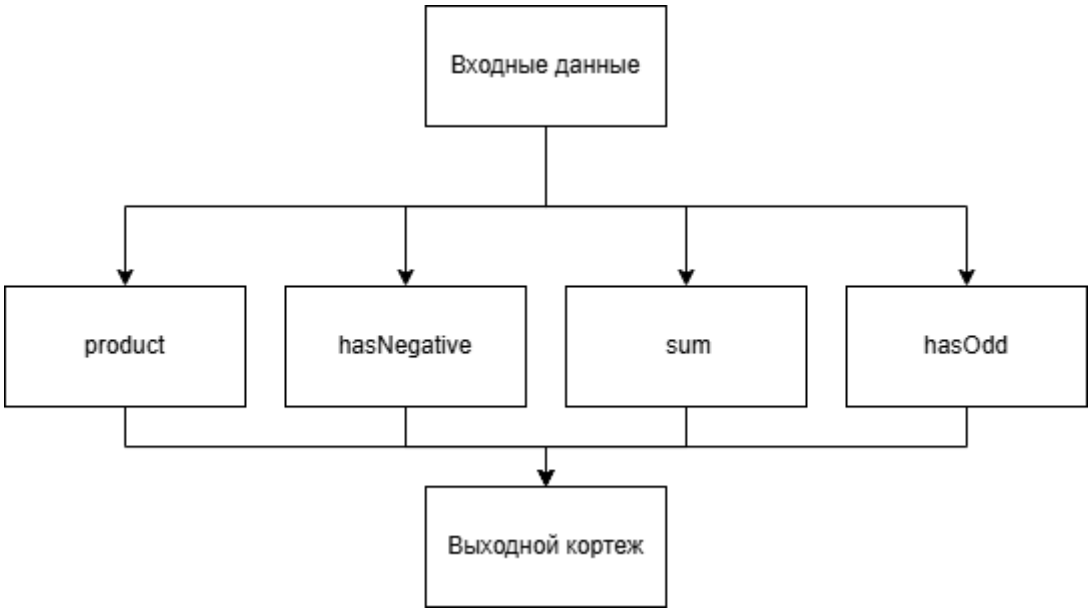

Рисунок 3

5.2. Формирование полного набора DU-цепочек.

1. Определение sum → Использование sum.
2. Определение product → Использование product.
3. Определение hasNegative → Использование hasNegative.
4. Определение hasOdd → Использование hasOdd.

5.3. Формирование полного набора отрезков путей в управляющем графе

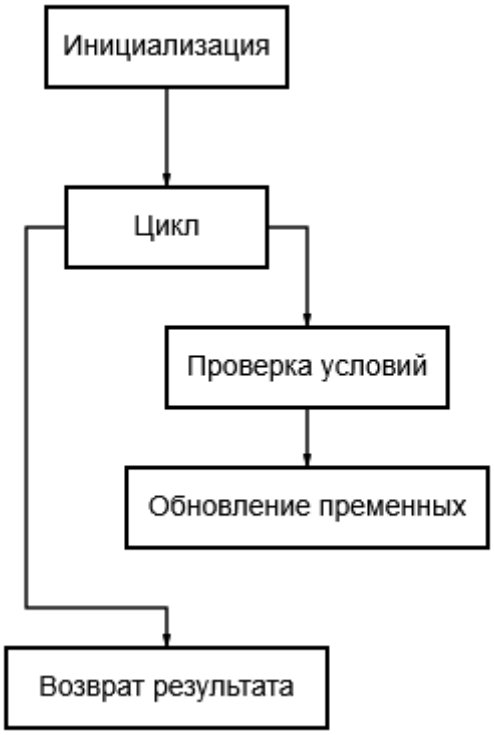

Рисунок 4 – управляющий граф

5.4. Построение маршрутов

Маршруты соответствуют базовым путям, описанным в разделе 4.3.

6. Области эквивалентности

1. Минимальная граница: Пустой массив.
2. Средняя область: Массив с положительными и отрицательными элементами.
3. Максимальная граница: Массив из 1024 элементов.

7. Контрольный пример

```
PS D:\projects\arrayTesting\ArrayProcessingApp> dotnet run
Input: 1,2,3,4,5 -> Error: No negative elements in the array.
Input: -1,-2,0,3,4,5 -> Sum of negative: -3, Product of odd: -15
Input: 1,3,5,7 -> Error: No negative elements in the array.
Input: -2,-4,-6 -> Error: No odd elements in the array.
Input: 0,0,0  -> Error: No negative elements in the array.
```

Рисунок 5 – вывод выполнения программы

```
PS D:\projects\arrayTesting\ArrayProcessingApp> dotnet test
Восстановление завершено (0,2 с)
  ArrayProcessingApp успешно выполнено (0,2 с) → bin\Debug\net9.0\ArrayProcessingApp.dll
[xUnit.net 00:00:00.00] xUnit.net VSTest Adapter v2.5.1.1+5fe7eebc65 (64-bit .NET 9.0.4)
[xUnit.net 00:00:00.22]   Discovering: ArrayProcessingApp
[xUnit.net 00:00:00.24]   Discovered:  ArrayProcessingApp
[xUnit.net 00:00:00.24]   Starting:    ArrayProcessingApp
[xUnit.net 00:00:00.39]   Finished:    ArrayProcessingApp
  ArrayProcessingApp (тест) успешно выполнено (1,8 с)

Сводка теста: всего: 6; сбой: 0; успешно: 6; пропущено: 0; длительность: 1,8 с
Сборка успешно выполнено через 2,5 с
```

8. Протокол тестирования

Таблица 2

Код теста	**Исходные данные (ИД)**	**Ожидаемые результаты (ОЖ.РЕЗ.)**	**Фактические результаты**	**Вывод**
ValidInput	[-1, 2, -3, 4, 5]	(-4, 15)	(-4, 15)	Пройден
NoOddElements	[-2, 2, 4]	No odd elements in the array	No odd elements in the array	Пройден
ArraySizeExceedsLimit	[1025 элементов]	Array size must be less than or equal to 1024.	Array size must be less than or equal to 1024.	Пройден
EmptyArray	[]	No negative elements in the array.	No negative elements in the array.	Пройден
WithPositiveAndZeroNumbers	[-1, 2, 0, 3, -4, 5]	(-5, -15)	(-5, -15)	Пройден

9. Текст программы

Файл Program.cs

```
using System;
using System.IO;
using System.Linq;
using System.Globalization;
using System.Collections.Generic;

namespace ArrayProcessingApp
{
    class Program
    {
        static void Main(string[] args)
        {
            // Создаем экземпляр класса ArrayProcessor
            var processor = new ArrayProcessor();

            // Подгрузка файла из корневой директории проекта
            string filePath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "..", "..", "..",
"input.txt");
            //string filePath = "input.txt";

            // Считать тестовые наборы
            processor.ProcessFile(filePath);
        }
    }
}
```

Файл ArrayProcessor.cs

```
using System;
using System.IO;
using System.Linq;

namespace ArrayProcessingApp
{
    public class ArrayProcessor
    {
        // Метод для вычисления суммы отрицательных и произведения нечетных элементов массива
        public (double sum, double product) ProcessArray(double[] array)
        {
            // Проверка на размер массива
            if (array.Length > 1024)
                throw new ArgumentException("Array size must be less than or equal to 1024.");

            double sum = 0; // Сумма отрицательных элементов
            double product = 1; // Произведение нечетных элементов
            bool hasOdd = false; // Флаг наличия нечетных элементов
            bool hasNegative = false; // Флаг наличия отрицательных элементов

            // Проход по всем элементам массива
            foreach (var num in array)
            {
                // Если элемент отрицательный, добавляем его к сумме
                if (num < 0)
                {
                    sum += num;
                    hasNegative = true; // Устанавливаем флаг, если найден отрицательный элемент
                }

                // Если элемент нечетный, умножаем его на произведение
                if (num % 2 != 0)
```

```
        {
            product *= num;
            hasOdd = true; // Устанавливаем флаг, если найден нечетный элемент
        }
    }

    // Если отрицательных элементов нет, выбрасываем исключение
    if (!hasNegative)
        throw new InvalidOperationException("No negative elements in the array.");

    // Если нечетных элементов нет, выбрасываем исключение
    if (!hasOdd)
        throw new InvalidOperationException("No odd elements in the array.");

    // Возвращаем кортеж с результатами
    return (sum, product);
}

// Метод для чтения входных данных из файла и обработки их
public void ProcessFile(string inputFile)
{
    try
    {
        string[] lines = File.ReadAllLines(inputFile);

        foreach (var line in lines)
        {
            double[] numbers = line.Split(',')
                                   .Select(s => double.TryParse(s, out var n) ? n : double.NaN)
                                   .Where(n => !double.IsNaN(n))
                                   .ToArray();

            if (numbers.Length == 0)
```

```
                {
                    Console.WriteLine($"Input: {line} -> Error: Invalid data");
                    continue;
                }

                try
                {
                    var result = ProcessArray(numbers);
                    Console.WriteLine($"Input: {line} -> Sum of negative: {result.sum}, Product of odd: {result.product}");
                }
                catch (Exception ex)
                {
                    Console.WriteLine($"Input: {line} -> Error: {ex.Message}");
                }
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error processing file: {ex.Message}");
        }
    }
  }
}
```

Файл ArrayProcessorTests.cs

```
using System;
using Xunit;


namespace ArrayProcessingApp.Tests
{
   public class ArrayProcessorTests
   {
      // Тест для проверки корректного вычисления суммы отрицательных и произведения нечетных элементов
      [Fact]
      public void ValidInput()
      {
         var processor = new ArrayProcessor();
         double[] array = { -1, 2, -3, 4, 5 };


         var result = processor.ProcessArray(array);


         Assert.Equal(-4, result.sum);  // -1+(-3) = -4 (отрицательные элементы)
         Assert.Equal(15, result.product);  // -1*-3*5 = 15 (нечетные элементы)
      }


      // Тест для проверки исключения при отсутствии нечетных элементов
      [Fact]
      public void NoOddElements()
      {
         var processor = new ArrayProcessor();
         double[] array = { -2, 2, 4 };


         var exception = Assert.Throws<InvalidOperationException>(() => processor.ProcessArray(array));
         Assert.Equal("No odd elements in the array.", exception.Message);
      }
```

```
    // Тест для проверки исключения при отсутствии отрицательных элементов
    [Fact]
    public void NoNegativeElements()
    {
        var processor = new ArrayProcessor();
        double[] array = { 1, 3, 5, 2 };

        var exception = Assert.Throws<InvalidOperationException>(() =>
processor.ProcessArray(array));
        Assert.Equal("No negative elements in the array.", exception.Message);
    }

    // Тест для проверки исключения при превышении размера массива
    [Fact]
    public void ArraySizeExceedsLimit()
    {
        var processor = new ArrayProcessor();
        double[] array = new double[1025]; // Массив из 1025 элементов

        var exception = Assert.Throws<ArgumentException>(() => processor.ProcessArray(array));
        Assert.Equal("Array size must be less than or equal to 1024.", exception.Message);
    }

    // Тест для проверки работы с пустым массивом
    [Fact]
    public void EmptyArray()
    {
        var processor = new ArrayProcessor();
        double[] array = { };

        var exception = Assert.Throws<InvalidOperationException>(() =>
processor.ProcessArray(array));
        // Either message could be thrown since empty array has neither negative nor odd elements
```

```
        Assert.Contains(exception.Message, new[] { "No negative elements in the array.", "No odd elements in the array." });
    }

    // Тест для проверки работы с массивом, содержащим положительные числа и нули
    [Fact]
    public void WithPositiveAndZeroNumbers()
    {
        var processor = new ArrayProcessor();
        double[] array = { -1, 2, 0, 3, -4, 5 };

        var result = processor.ProcessArray(array);

        Assert.Equal(-5, result.sum);  // -1+(-4) = -5 (отрицательные элементы)
        Assert.Equal(-15, result.product);  // -1*3*5 = -15 (нечетные элементы)
    }
  }
}
```

10. Входные и выходные данные

Входные данные: Файл input.txt с тестовыми наборами. **Выходные данные**: Протокол тестирования (см. раздел 8).

Файл input.txt

1,2,3,4,5

-1,-2,0,3,4,5

1,3,5,7

-2,-4,-6

0,0,0

11. Заключение

Лабораторная работа выполнена в соответствии с требованиями. Все тесты пройдены успешно. Код размещен в репозитории на GitHub с настроенными GitHub Actions для автоматического тестирования.