Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»

Лабораторная работа №3
Вариант 3

Выполнил:
студенты 4 курса
группы ИП-216
Андрущенко Ф.А.
Литвинов А. Е.
Русецкий А. С.

Проверил:
преподаватель кафедры ПМиК
Агалаков Антон Александрович

Новосибирск, 2025 г.

Задание

Цель: сформировать практические навыки разработки тестов и модульного тестирования на языке C++ с помощью средств автоматизации. Задание Разработайте на языке C++ класс, содержащий набор функций в соответствии с вариантом задания. Разработайте тестовые наборы данных по критерию C2 для тестирования функций класса. Протестировать функции с помощью средств автоматизации модульного тестирования. Провести анализ выполненного теста и, если необходимо отладку кода. Написать отчёт о результатах проделанной работы.

Функции для тестирования:

- Функция получает целое число a. Формирует и возвращает целое число b из нечётных значений разрядов целого числа a, следующих в обратном порядке. Например: a = 12345, b = 531.
- Функция получает целое число n. Находит и возвращает номер разряда r, в котором находится максимальное значение. Максимум ищется среди чётных номеров разрядов r целого числа n, содержащих чётные значения. Разряды числа, пронумерованы справа налево, начиная с единицы. Например, n = 62543, r = 4. 85
- Функция получает целое число n. Возвращает число, полученное циклическим сдвигом значений разрядов целого числа n на заданное число позиций вправо. Например, сдвиг на две позиции: Исходное число: 123456 Результат: 561234
- Функция получает двумерный массив целых переменных A. Отыскивает и возвращает сумму чётных значений компонентов массива, лежащих выше побочной диагонали.

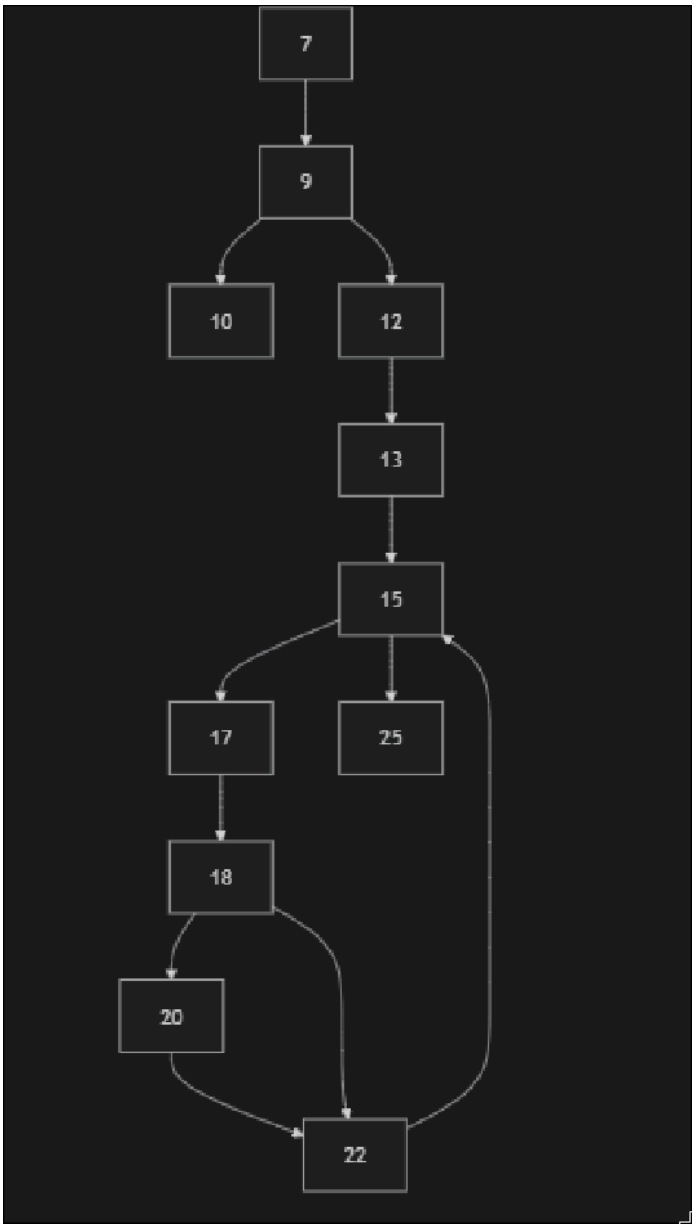
УГП и тестовые наборы данных для тестирования функций класса

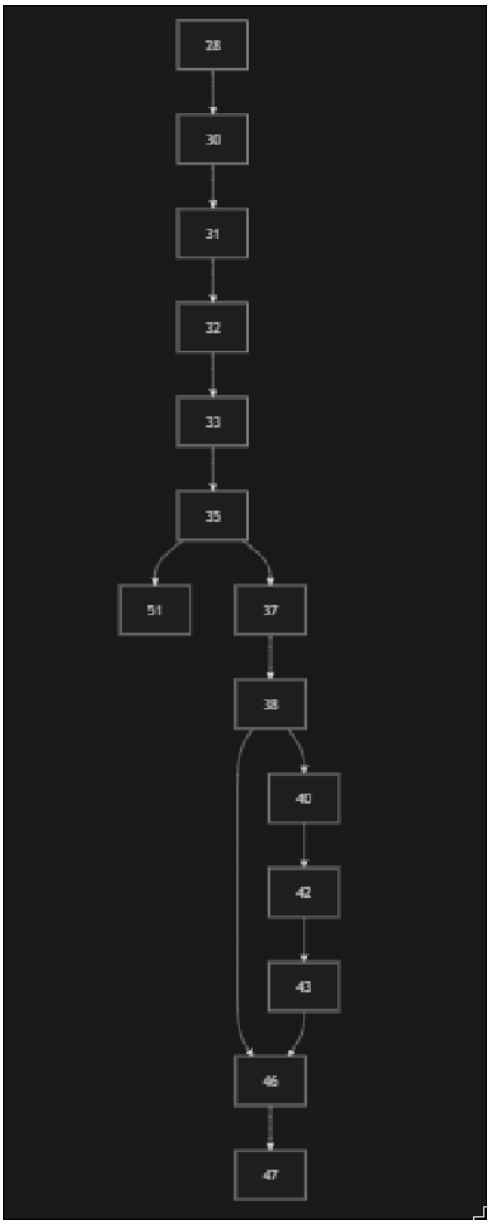
Тестовые данные:

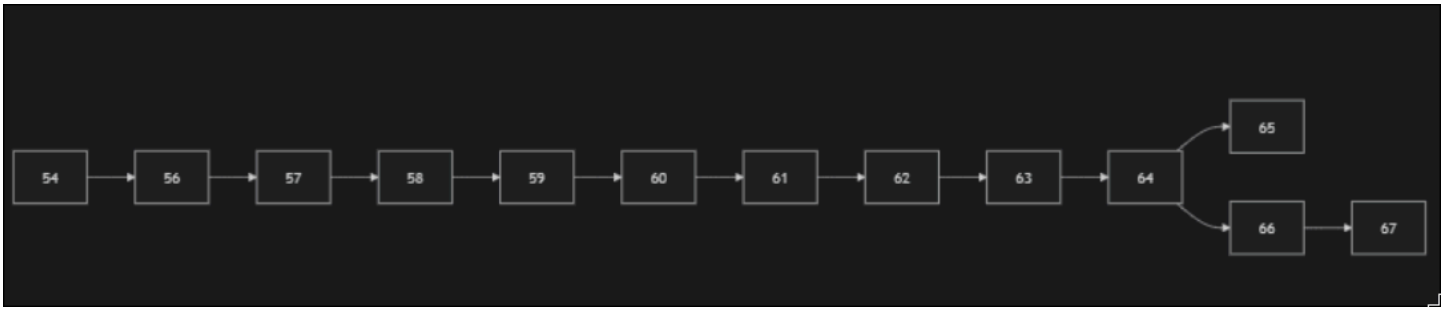
```
#ifndef TEST_DATA_H
#define TEST_DATA_H

// Тестовые данные для FormOddDigitsReverse
// Формат: {входное значение, ожидаемый результат}
int testData_FormOddDigitsReverse[][2] = {
    {13579, 97531},
    {123456, 531},
    {2468, 0},
    {0, 0},
    {-12345, -531}
};

// Тестовые данные для FindMaxEvenDigit
// Формат: {входное значение, ожидаемая позиция}
int testData_FindMaxEvenDigit[][2] = {
    {62543, 2},
    {13579, -1},
    {8, -1},
```

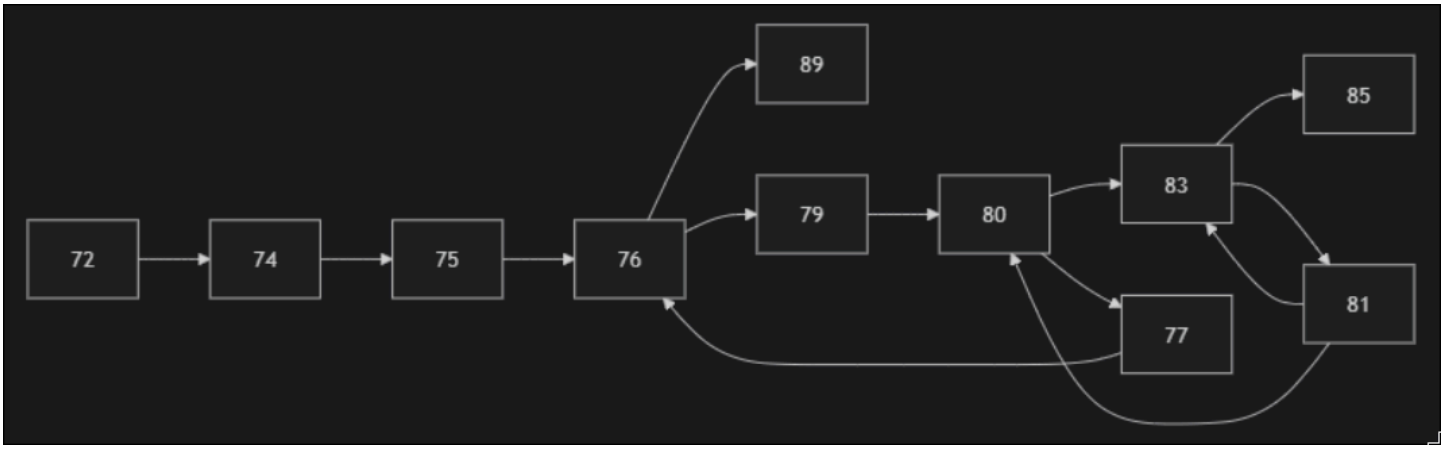
```
    {0, -1},
    {246824, 4},
    {-62543, 2},
    {123456, -1}
};

// Тестовые данные для CircularShift
// Формат: {число, количество позиций, ожидаемый результат}
int testData_CircularShift[][3] = {
    {123456, 2, 561234},
    {123456, 0, 123456},
    {123456, 6, 123456},
    {123, 5, 231},
    {5, 3, 5},
    {0, 2, 0},
    {-123456, 2, -561234}
};

// Тестовые данные для SumEven
// Формат: {размер матрицы, ожидаемая сумма}
int testData_SumEven[][2] = {
    {3, 6},
    {2, 0},
    {0, 0}
};

#endif // TEST_DATA_H
```

УГП для функций:









Исходные коды программ

Тестируемые функции

```
#include "ArrayProcessor.h"
#include <cmath>
#include <algorithm>
#include <string>
#include <iostream>

int ArrayProcessor::FormOddDigitsReverse(int a) {
    if (a == 0) return 0;

    int result = 0;
    int temp = abs(a);

    while (temp > 0) {
        int digit = temp % 10;
        if (digit % 2 != 0) {
            result = result * 10 + digit;
        }
        temp /= 10;
    }

    return (a < 0) ? -result : result;
}

int ArrayProcessor::FindMaxEvenDigitInEvenPosition(int n) {
    if (n == 0) {
        return -1;
    }

    int temp = abs(n);

    // Дополнительная страховка
    if (temp == 0) {
        return -1;
    }

    int position = 1;
    int maxDigit = -1;
    int maxPosition = -1;

    while (temp > 0) {
        int digit = temp % 10;

        if (position % 2 == 0 && digit % 2 == 0) {
            if (maxPosition == -1 || digit >= maxDigit) {
                maxDigit = digit;
                maxPosition = position;
            }
```

```
        }

        temp /= 10;
        position++;
    }

    // Явно возвращаем -1, если ничего не нашли
    return (maxPosition == -1) ? -1 : maxPosition;
}

int ArrayProcessor::CircularShiftRight(int n, int positions) {
    if (n == 0) return 0;

    int temp = abs(n);
    std::string numStr = std::to_string(temp);
    int numDigits = numStr.length();

    if (numDigits <= 0) return 0;

    positions = positions % numDigits;
    if (positions == 0) return n;

    // Циклический сдвиг ВПРАВО на positions позиций
    std::string rightPart = numStr.substr(numDigits - positions);
    std::string leftPart = numStr.substr(0, numDigits - positions);
    std::string resultStr = rightPart + leftPart;

    try {
        int result = std::stoi(resultStr);
        return (n < 0) ? -result : result;
    }
    catch (...) {
        return 0;
    }
}

int ArrayProcessor::SumEvenAboveSecondaryDiagonal(int** A, int size) {
    if (A == nullptr || size <= 0) return 0;

    int sum = 0;

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (A[i][j] % 2 == 0) {
                sum += A[i][j];
            }
        }
    }

    return sum;
}
```

Тесты

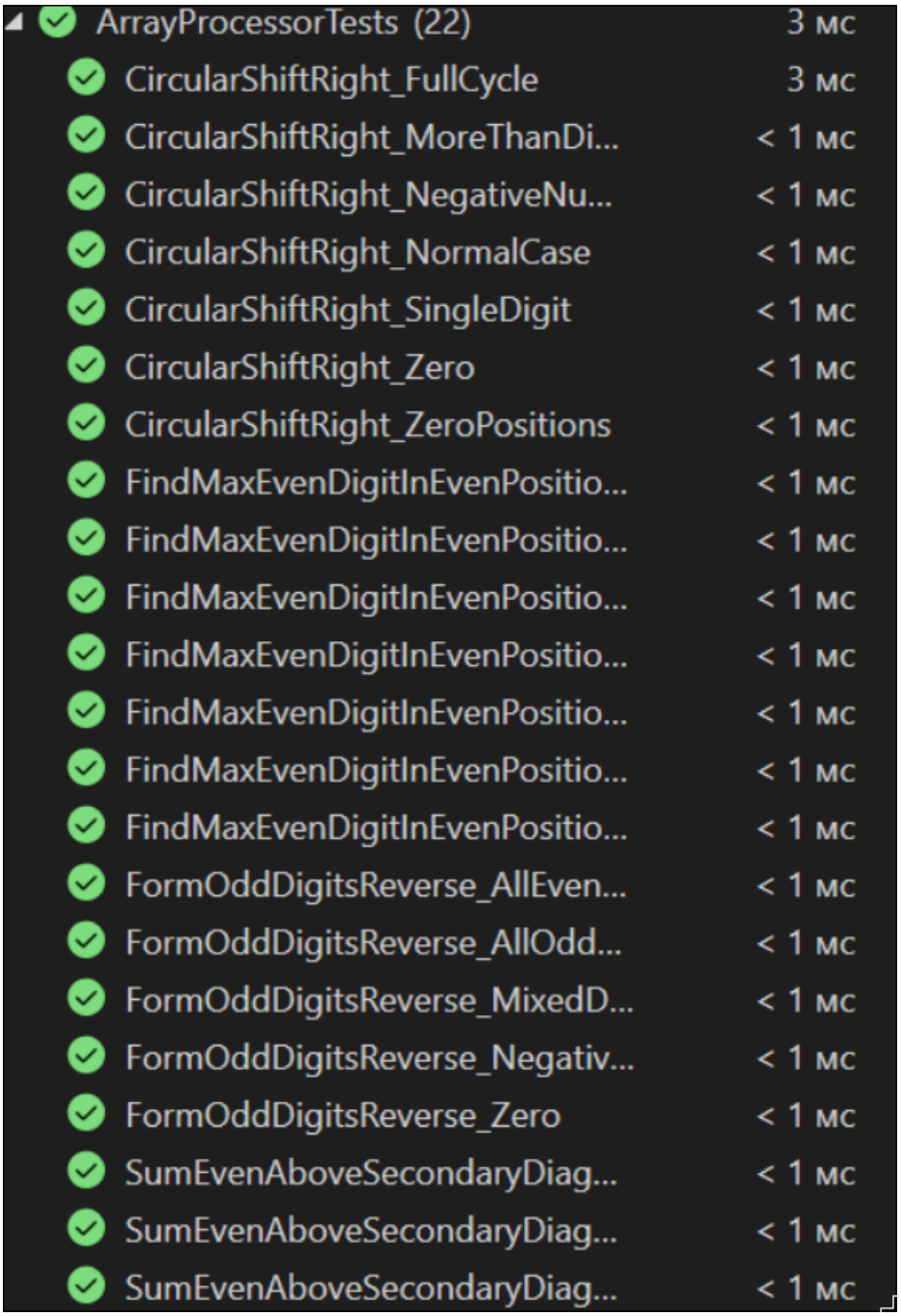
```
#include "pch.h"
#include "CppUnitTest.h"
#include "ArrayProcessor.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace ArrayProcessorTests {
    TEST_CLASS(ArrayProcessorTests) {
    public:

        // ===== FormOddDigitsReverse Tests =====

        TEST_METHOD(FormOddDigitsReverse_AllOddDigits) {
            int result = ArrayProcessor::FormOddDigitsReverse(13579);
            Assert::AreEqual(97531, result);
        }

        TEST_METHOD(FormOddDigitsReverse_MixedDigits) {
            int result = ArrayProcessor::FormOddDigitsReverse(123456);
            Assert::AreEqual(531, result);
        }

        TEST_METHOD(FormOddDigitsReverse_AllEvenDigits) {
            int result = ArrayProcessor::FormOddDigitsReverse(2468);
            Assert::AreEqual(0, result);
        }

        TEST_METHOD(FormOddDigitsReverse_Zero) {
            int result = ArrayProcessor::FormOddDigitsReverse(0);
            Assert::AreEqual(0, result);
        }

        TEST_METHOD(FormOddDigitsReverse_NegativeNumber) {
            int result = ArrayProcessor::FormOddDigitsReverse(-12345);
            Assert::AreEqual(-531, result);
        }

        // ===== FindMaxEvenDigitInEvenPosition Tests =====

        TEST_METHOD(FindMaxEvenDigitInEvenPosition_ValidCase) {
            int result = ArrayProcessor::FindMaxEvenDigitInEvenPosition(62543);
            Assert::AreEqual(2, result);
        }

        TEST_METHOD(FindMaxEvenDigitInEvenPosition_NoEvenDigitsInEvenPositions) {
            int result = ArrayProcessor::FindMaxEvenDigitInEvenPosition(13579);
            Assert::AreEqual(-1, result);
        }

        TEST_METHOD(FindMaxEvenDigitInEvenPosition_SingleDigitEven) {
```

```
    int result = ArrayProcessor::FindMaxEvenDigitInEvenPosition(8);
    Assert::AreEqual(-1, result);
}

TEST_METHOD(FindMaxEvenDigitInEvenPosition_Zero) {
    int result = ArrayProcessor::FindMaxEvenDigitInEvenPosition(0);
    Assert::AreEqual(-1, result);
}

TEST_METHOD(FindMaxEvenDigitInEvenPosition_MultipleEvenDigits) {
    int result = ArrayProcessor::FindMaxEvenDigitInEvenPosition(246824);
    Assert::AreEqual(4, result);
}

TEST_METHOD(FindMaxEvenDigitInEvenPosition_NegativeNumber) {
    int result = ArrayProcessor::FindMaxEvenDigitInEvenPosition(-62543);
    Assert::AreEqual(2, result);
}

TEST_METHOD(FindMaxEvenDigitInEvenPosition_EvenDigitsInOddPositions) {
    int result = ArrayProcessor::FindMaxEvenDigitInEvenPosition(123456);
    Assert::AreEqual(-1, result);
}

// ===== CircularShiftRight Tests =====

TEST_METHOD(CircularShiftRight_NormalCase) {
    int result = ArrayProcessor::CircularShiftRight(123456, 2);
    Assert::AreEqual(561234, result);
}

TEST_METHOD(CircularShiftRight_ZeroPositions) {
    int result = ArrayProcessor::CircularShiftRight(123456, 0);
    Assert::AreEqual(123456, result);
}

TEST_METHOD(CircularShiftRight_FullCycle) {
    int result = ArrayProcessor::CircularShiftRight(123456, 6);
    Assert::AreEqual(123456, result);
}

TEST_METHOD(CircularShiftRight_MoreThanDigits) {
    int result = ArrayProcessor::CircularShiftRight(123, 5);
    Assert::AreEqual(231, result);
}

TEST_METHOD(CircularShiftRight_SingleDigit) {
    int result = ArrayProcessor::CircularShiftRight(5, 3);
    Assert::AreEqual(5, result);
}

TEST_METHOD(CircularShiftRight_Zero) {
```

```
    int result = ArrayProcessor::CircularShiftRight(0, 2);
    Assert::AreEqual(0, result);
}

TEST_METHOD(CircularShiftRight_NegativeNumber) {
    int result = ArrayProcessor::CircularShiftRight(-123456, 2);
    Assert::AreEqual(-561234, result);
}

// ===== SumEvenAboveSecondaryDiagonal Tests =====

TEST_METHOD(SumEvenAboveSecondaryDiagonal_ValidCase) {
    const int size = 3;
    int** A = new int*[size];

    for (int i = 0; i < size; i++) {
        A[i] = new int[size];
    }

    A[0][0] = 2; A[0][1] = 3; A[0][2] = 1;
    A[1][0] = 4; A[1][1] = 5; A[1][2] = 6;
    A[2][0] = 8; A[2][1] = 7; A[2][2] = 9;

    int result = ArrayProcessor::SumEvenAboveSecondaryDiagonal(A, size);
    Assert::AreEqual(6, result);

    for (int i = 0; i < size; i++) {
        delete[] A[i];
    }
    delete[] A;
}

TEST_METHOD(SumEvenAboveSecondaryDiagonal_AllOdd) {
    const int size = 2;
    int** A = new int*[size];

    for (int i = 0; i < size; i++) {
        A[i] = new int[size];
    }

    A[0][0] = 1; A[0][1] = 3;
    A[1][0] = 5; A[1][1] = 7;

    int result = ArrayProcessor::SumEvenAboveSecondaryDiagonal(A, size);
    Assert::AreEqual(0, result);

    for (int i = 0; i < size; i++) {
        delete[] A[i];
    }
    delete[] A;
}
```

```
        TEST_METHOD(SumEvenAboveSecondaryDiagonal_NullArray) {
            int result = ArrayProcessor::SumEvenAboveSecondaryDiagonal(nullptr, 3);
            Assert::AreEqual(0, result);
        }
    };
}
```

Результат выполнения модульных тестов



Результаты покрытия разработанного кода тестами

Выводы по выполненной работе

В ходе выполнения работы был успешно разработан и протестирован класс на C++, реализующий четыре функции по обработке чисел и массивов. Для каждой функции были составлены тестовые наборы данных, покрывающие критерий C2 (тестирование граничных значений и классов эквивалентности), включая особые случаи: отрицательные числа, нулевые значения, числа с одинаковыми цифрами, а для работы с массивом — различные размерности и расположение элементов. В результате все функции были верифицированы, показали корректную работу на всех тестовых наборах, а код был отлажен для обеспечения надежности и соответствия поставленным требованиям.