Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Лабораторная работа №3

«Модульное тестирование программ на языке C++ в среде Visual Studio»

Вариант №4

Выполнил: студент 4 курса
ИВТ, гр. ИП-113
Шпилев Д. И.
Проверил: старший преподаватель кафедры ПМиК
Агалаков А.А.

Новосибирск, 2024 г.

Цель

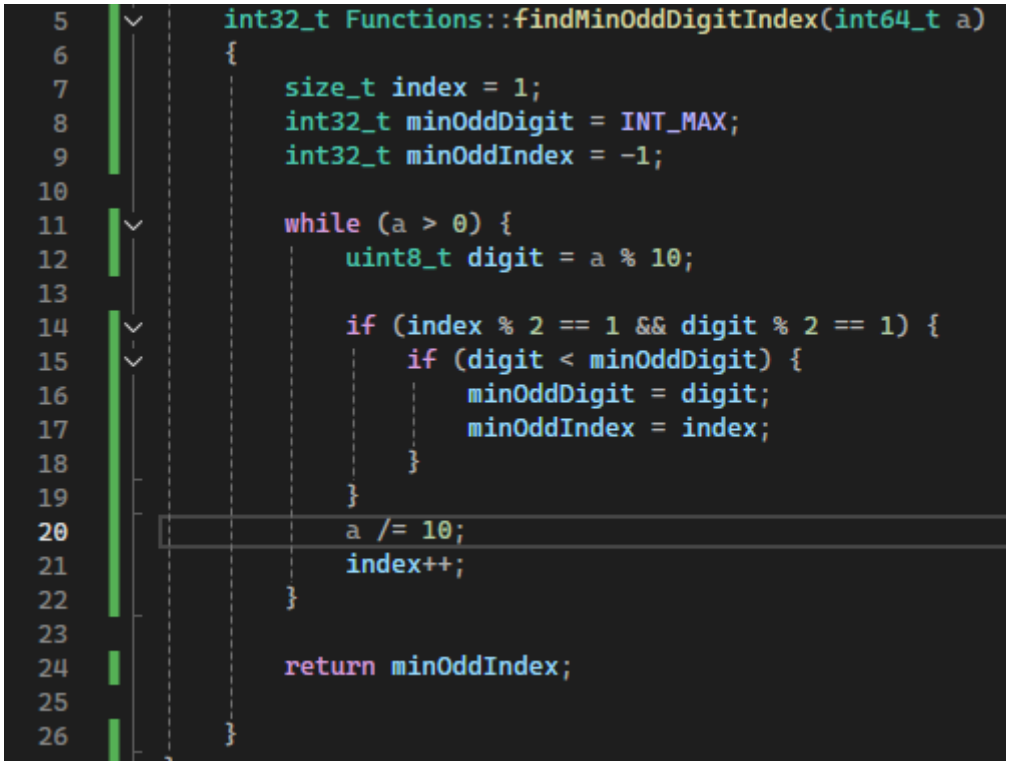
Сформировать практические навыки разработки тестов и модульного тестирования на языке C++ с помощью средств автоматизации Visual Studio

Задание

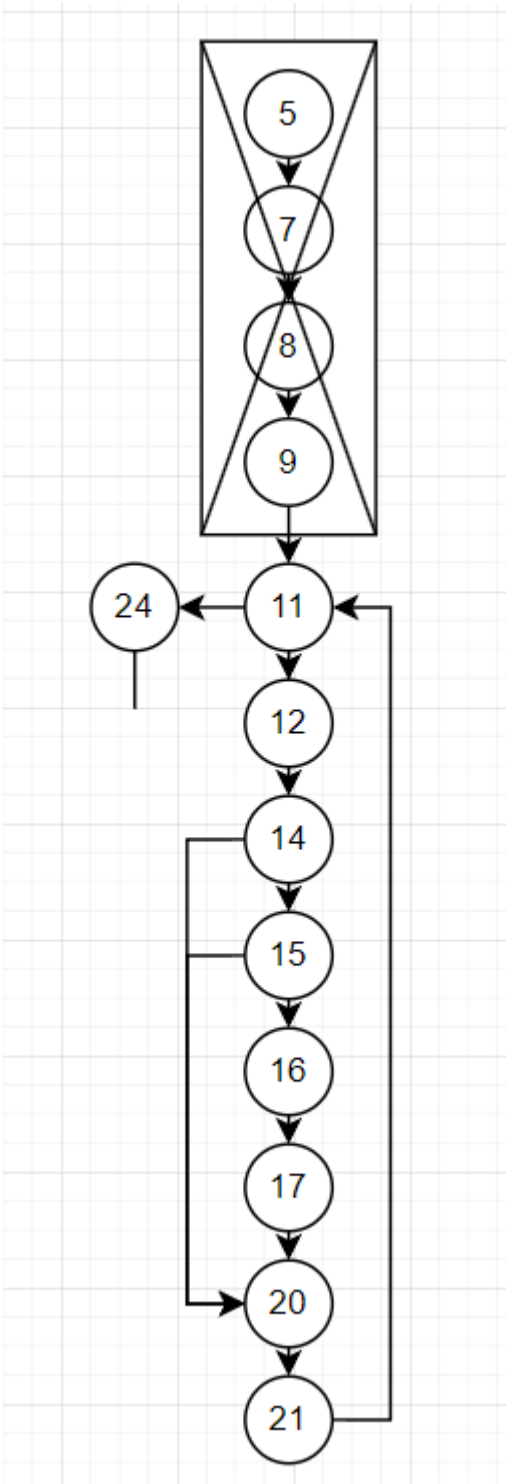
Разработайте на языке C++ класс, содержащий набор функций в соответствии с вариантом задания. Разработайте тестовые наборы данных по критерию C2 для тестирования функций класса. Протестировать функции с помощью средств автоматизации модульного тестирования Visual Studio. Провести анализ выполненного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

1. Функция получает целое числа a. Находит и возвращает номер разряда, в котором находится минимальное значение r среди нечётных разрядов целого числа a с нечётным. Разряды числа, пронумерованы справа налево, начиная с единицы. Например, a = 12543, r = 3.
2. Функция получает целое числа a. Возвращает число, полученное циклическим сдвигом значений разрядов целого числа a на заданное число позиций влево. Например, сдвиг на две позиции: Исходное число: 123456 Результат: 345612
3. Функция получает целые числа a, b и n. Возвращает число, полученное путём вставки разрядов числа b в целое число a после разряда, заданного числом n. Разряды нумеруются слева направо, начиная с 1. Например, вставить после 2 разряда значение 6: Исходное число: 123457 вставить 6 после 2 разряда Результат: 1263457
4. Функция получает двумерный массив вещественных переменных A. Отыскивает и возвращает сумму чётных значений компонентов массива, лежащих ниже побочной диагонали

УГП и тестовые наборы данных для тестирования функций класса

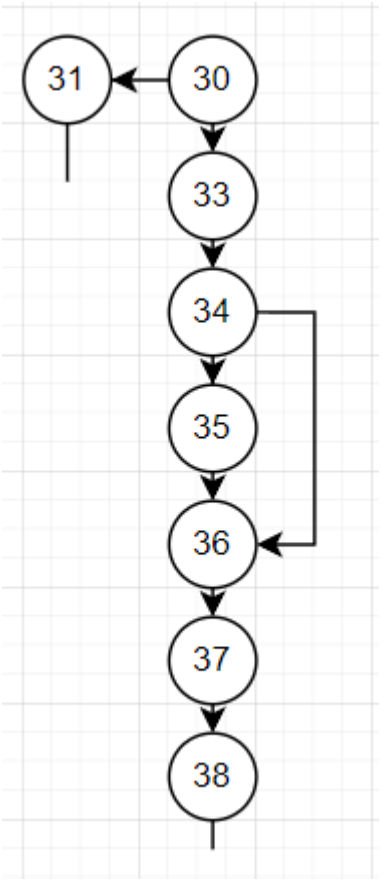


```
5   int32_t Functions::findMinOddDigitIndex(int64_t a)
6   {
7       size_t index = 1;
8       int32_t minOddDigit = INT_MAX;
9       int32_t minOddIndex = -1;
10
11      while (a > 0) {
12          uint8_t digit = a % 10;
13
14          if (index % 2 == 1 && digit % 2 == 1) {
15              if (digit < minOddDigit) {
16                  minOddDigit = digit;
17                  minOddIndex = index;
18              }
19          }
20          a /= 10;
21          index++;
22      }
23
24      return minOddIndex;
25
26  }
```



Пути:
(5-7-8-9-11-24) a = 0
(5-7-8-9-11-12-14-20-21-24) a = 2
(5-7-8-9-11-12-14-15-20-21-24) a = 321
(5-7-8-9-11-12-14-15-16-17-20-21-24) a = 12543

```cpp
int64_t Functions::cyclicLeftShift(const int64_t& a, int8_t shift)
{
    if (a < 0) {
        throw std::invalid_argument("Number need be positiv!");
    }
    std::string num = std::to_string(a);
    if (shift < 0)
        shift += num.length();
    shift %= num.length();
    std::string shiftedStr = num.substr(shift) + num.substr(0, shift);
    return _atoi64(shiftedStr.c_str());
}
```



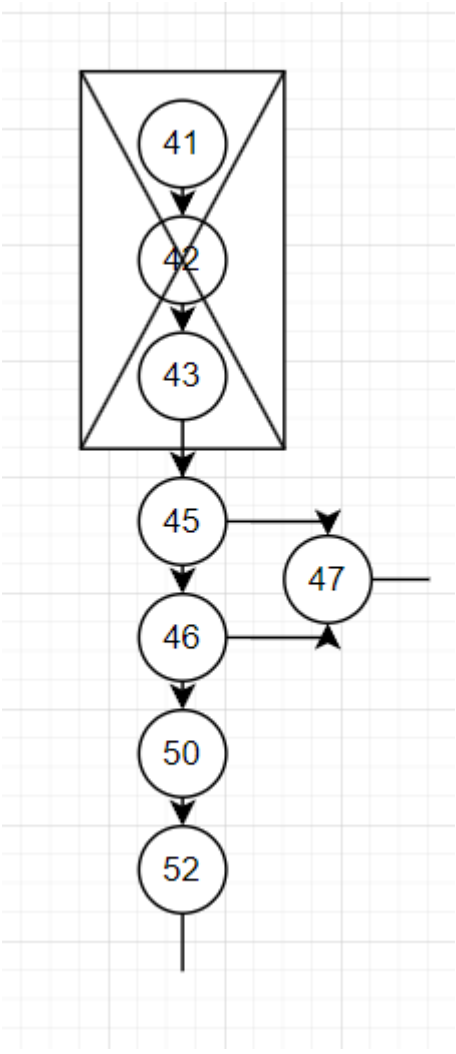
Пути:

(30-31) a = -1

(30-33-34-36-37-38) a = 12345 shift = 3

(30-33-34-35-36-37-38) a = 12345 shift = -2

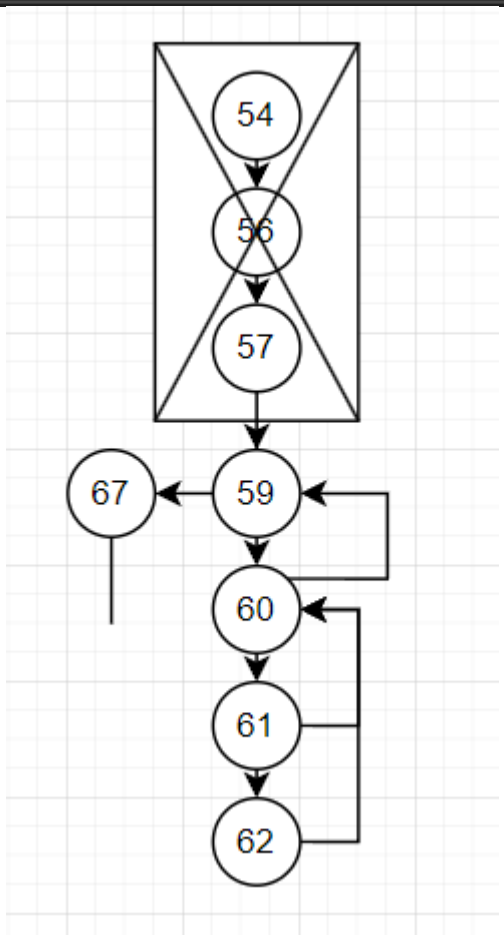
```cpp
int64_t Functions::insertDigits(int64_t a, int64_t b, size_t n) {
    std::string strA = std::to_string(a);
    std::string strB = std::to_string(b);

    if (n < 1
        || n > strA.length()) {
        throw std::out_of_range("Position n is out of range.");
    }

    std::string result = strA.substr(0, n) + strB + strA.substr(n);

    return std::stoll(result);
}
```



Пути:

(41-42-43-45-47) n = -1
(41-42-43-45-46-47) a = 12345 n = 7
(41-42-43-45-46-50-52) a = 123457 b = 6 n = 5

```
54  double Functions::sumEvenBelowAntiDiagonal(const std::vector<std::vector<double>>& A)
55  {
56      int N = A.size();
57      double sum = 0.0;
58
59      for (int i = 0; i < N; ++i) {
60          for (int j = 0; j < N; ++j) {
61              if (i + j >= N && static_cast<int>(A[i][j]) % 2 == 0) {
62                  sum += A[i][j];
63              }
64          }
65      }
66
67      return sum;
68  }
```



Пути:
(59-67) A = {{}}
(59-60-61-67) A = {{1.0, 1.0}, {1.0, 1.0}}
(59-60-61-62-67) A = {{1.0, 2.0, 3.0},{4.0, 6.0, 8.0},{10.0, 12.0, 14.0}}

Листинг программы:

Functions.h:

```
#pragma once
#include <string>
#include <stdexcept>
#include <vector>

namespace Lab3 {

	class Functions {
		Functions() = delete;
		Functions(const Functions& other) = delete;
		Functions(Functions&& other) = delete;

	public:
		static int32_t findMinOddDigitIndex(int64_t a);
		static int64_t cyclicLeftShift(const int64_t& a, int16_t shift);
		static int64_t insertDigits(int64_t a, int64_t b, int16_t n);
		static double sumEvenBelowAntiDiagonal(const
std::vector<std::vector<double>>& A);
	};
}
```

Functions.cpp:

```
#include "Functions.h"

namespace Lab3 {

	int32_t Functions::findMinOddDigitIndex(int64_t a)
	{
		int32_t index = 1;
		int32_t minOddDigit = INT_MAX;
		int32_t minOddIndex = -1;

		while (a > 0) {
			uint16_t digit = a % 10;

			if (index % 2 == 1 && digit % 2 == 1) {
				if (digit < minOddDigit) {
					minOddDigit = digit;
					minOddIndex = index;
				}
			}
			a /= 10;
			index++;
		}

		return minOddIndex;

	}

	int64_t Functions::cyclicLeftShift(const int64_t& a, int16_t shift)
	{
		if (a < 0) {
			throw std::invalid_argument("Number need be positiv!");
		}
		std::string num = std::to_string(a);
		if (shift < 0)
			shift += num.length();
		shift %= num.length();
```

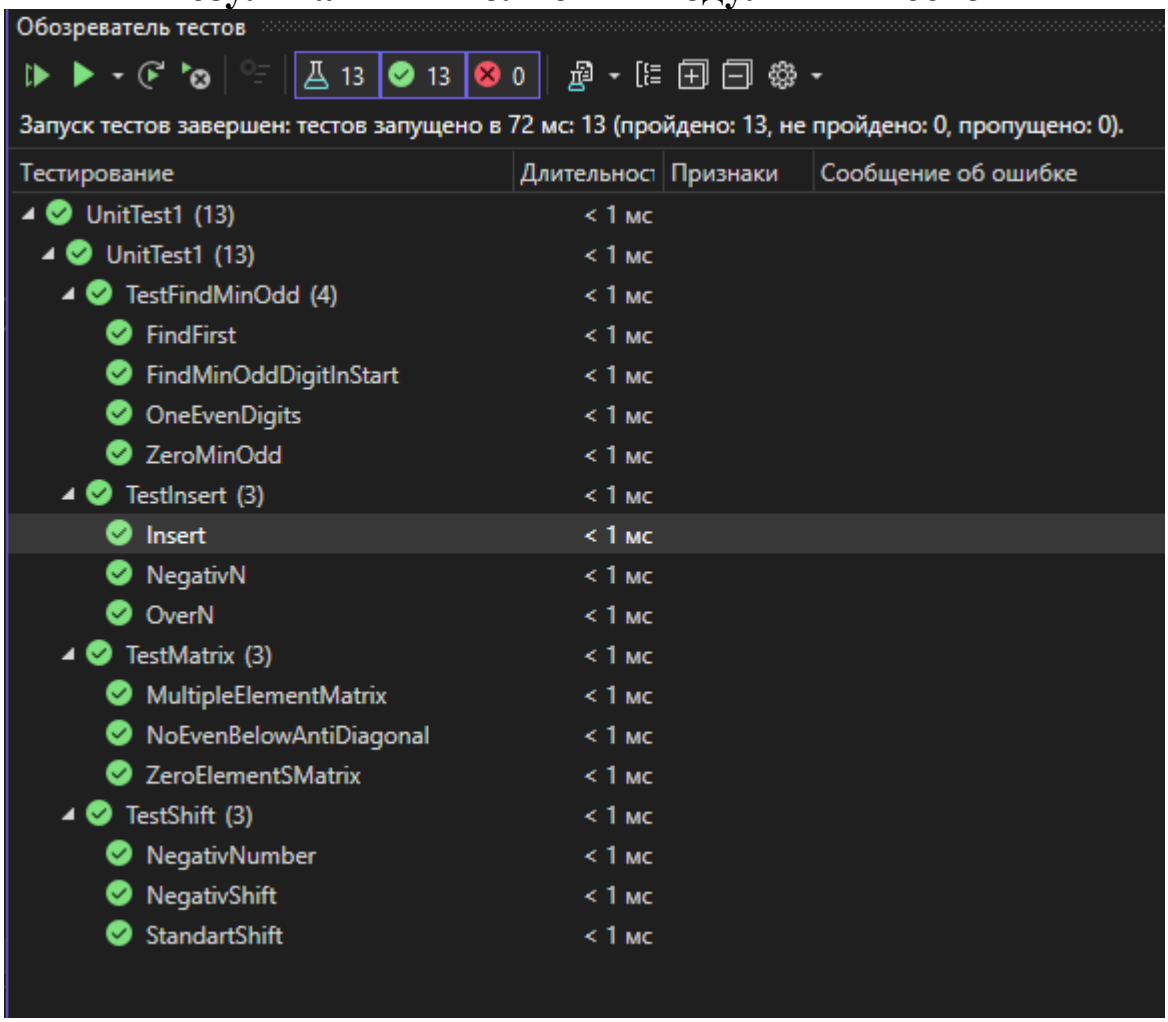
```
            std::string shiftedStr = num.substr(shift) + num.substr(0, shift);
            return std::stoll(shiftedStr);
      }

      int64_t Functions::insertDigits(int64_t a, int64_t b, int16_t n) {
            std::string strA = std::to_string(a);
            std::string strB = std::to_string(b);

            if (n < 1 || n > strA.length()) {
                  throw std::out_of_range("Position n is out of range");
            }

            std::string result = strA.substr(0, n) + strB + strA.substr(n);

            return std::stoll(result);
      }

      double Functions::sumEvenBelowAntiDiagonal(const
std::vector<std::vector<double>>& A)
      {
            int N = A.size();
            double sum = 0.0;

            for (int i = 0; i < N; ++i) {
                  for (int j = 0; j < N; ++j) {
                        if (i + j >= N && static_cast<int>(A[i][j]) % 2 == 0) {
                              sum += A[i][j];
                        }
                  }
            }

            return sum;
      }
}
```

UnitTest1.cpp:

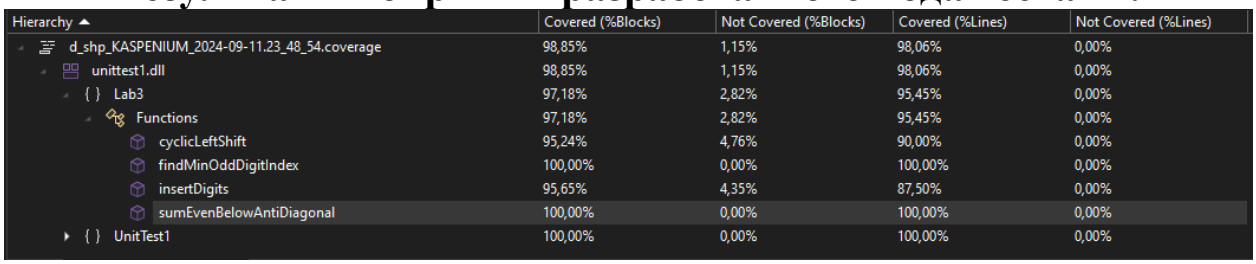
```
#include "CppUnitTest.h"
#include "../Lab3/Functions.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1
{
	TEST_CLASS(TestFindMinOdd)
	{
	public:

		TEST_METHOD(FindMinOddDigitInStart)
		{
			int32_t result = Lab3::Functions::findMinOddDigitIndex(12543);
			Assert::AreEqual(5, result);
		}

		TEST_METHOD(ZeroMinOdd)
		{
			int32_t result = Lab3::Functions::findMinOddDigitIndex(0);
			Assert::AreEqual(-1, result);
		}

		TEST_METHOD(OneEvenDigits)
		{
			int32_t result = Lab3::Functions::findMinOddDigitIndex(2);
			Assert::AreEqual(-1, result);
		}

		TEST_METHOD(FindFirst)
		{
			int32_t result = Lab3::Functions::findMinOddDigitIndex(321);
			Assert::AreEqual(1, result);
		}
	};

	TEST_CLASS(TestShift)
	{
	public:

		TEST_METHOD(NegativNumber)
		{
			Assert::ExpectException<std::invalid_argument>([]
{Lab3::Functions::cyclicLeftShift(-1236, 1); }, L"Number need be positiv!");
		}

		TEST_METHOD(StandartShift)
		{
			int64_t a = 12345;
			int8_t shift = 3;
			int64_t result = Lab3::Functions::cyclicLeftShift(a, shift);
			Assert::AreEqual(45123LL, result);
		}

		TEST_METHOD(NegativShift)
		{
			int64_t a = 12345;
			int8_t shift = -2;
			int64_t result = Lab3::Functions::cyclicLeftShift(a, shift);
			Assert::AreEqual(45123LL, result);
		}
	};
```

```
	TEST_CLASS(TestInsert)
	{
	public:

		TEST_METHOD(NegativN)
		{
			Assert::ExpectException<std::out_of_range>([]
{Lab3::Functions::insertDigits(12345, 1, -1); }, L"Position n is out of range");
		}

		TEST_METHOD(OverN)
		{
			Assert::ExpectException<std::out_of_range>([]
{Lab3::Functions::insertDigits(12345, 1, 6); }, L"Position n is out of range");
		}

		TEST_METHOD(Insert)
		{
			int64_t a = 123457;
			int64_t b = 6;
			int8_t n = 6;
			int64_t result = Lab3::Functions::insertDigits(123457, 6, 5);
			Assert::AreEqual(1234567LL, result);
		}
	};

	TEST_CLASS(TestMatrix)
	{
	public:

		TEST_METHOD(ZeroElementSMatrix)
		{
			std::vector<std::vector<double>> A = { {} };
			double result = Lab3::Functions::sumEvenBelowAntiDiagonal(A);
			Assert::AreEqual(0.0, result);
		}

		TEST_METHOD(MultipleElementMatrix)
		{
			std::vector<std::vector<double>> A = {
				{1.0, 2.0, 3.0},
				{4.0, 6.0, 8.0},
				{10.0, 12.0, 14.0}
			};
			double result = Lab3::Functions::sumEvenBelowAntiDiagonal(A);
			Assert::AreEqual(34.0, result);
		}

		TEST_METHOD(NoEvenBelowAntiDiagonal)
		{
			std::vector<std::vector<double>> A = {
				{1.1, 3.3},
				{5.5, 7.7}
			};
			double result = Lab3::Functions::sumEvenBelowAntiDiagonal(A);
			Assert::AreEqual(0.0, result);
		}
	};
}
```

Результаты выполнения модульных тестов



Результаты покрытия разработанного кода тестами.



Hierarchy	Covered (%Blocks)	Not Covered (%Blocks)	Covered (%Lines)	Not Covered (%Lines)
d_shp_KASPENIUM_2024-09-11.23_48_54.coverage	98,85%	1,15%	98,06%	0,00%
unittest1.dll	98,85%	1,15%	98,06%	0,00%
{ } Lab3	97,18%	2,82%	95,45%	0,00%
Functions	97,18%	2,82%	95,45%	0,00%
cyclicLeftShift	95,24%	4,76%	90,00%	0,00%
findMinOddDigitIndex	100,00%	0,00%	100,00%	0,00%
insertDigits	95,65%	4,35%	87,50%	0,00%
sumEvenBelowAntiDiagonal	100,00%	0,00%	100,00%	0,00%
{ } UnitTest1	100,00%	0,00%	100,00%	0,00%

Вывод

Были сформированы практические навыки разработки и выполнения модульного тестирования с помощью средств автоматизации Visual Studio, разработан класс на языке C++, содержащий функции в соответствии с вариантом задания, разработаны тестовые наборы данных для тестирования функций класса, по критерию C2.