

# **Математические основы защиты информации и информационной безопасности.**

**Лабораторная работа №5.**

Подмогильный Иван Александрович.

# Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	13

# List of Figures

3.1	Функция теста Ферма . . . . .	7
3.2	Функция теста Соловья-Штрассена . . . . .	8
3.3	тест Миллера-Рабина . . . . .	8
3.4	Вспомогательная функция вычисления случайного числа . . . . .	9
3.5	Вспомогательная функция вычисления символа Якоби . . . . .	10
3.6	Вспомогательная функция подсчета делителей . . . . .	11
3.7	Вспомогательная функция проверки числа . . . . .	11
3.8	CMake файл . . . . .	11
3.9	main.cpp файл . . . . .	12
3.10	Результаты тестов . . . . .	12

## List of Tables

# 1 Цель работы

Освоить на практике вероятностные алгоритмы проверки чисел на простоту

## 2 Задание

1. Реализовать алгоритм теста Ферма
2. Реализовать вычисления символа Якоби
3. Реализовать алгоритм теста Соловья-Штрассена
4. Реализовать алгоритм теста Миллера-Рабина

### 3 Выполнение лабораторной работы

Написал код для вычисления теста Ферма. Все функции реализованы шаблонами:

```
#ifndef LAB05_PRIMENUMBERSHELPER_H
#define LAB05_PRIMENUMBERSHELPER_H

#include <random>
#include <stdexcept>

class PrimeNumbersHelper {
public:
    template <typename T>
    // true - probably prime number
    // false - composite number
    static bool FermatTest(const T& n){
        if (!check_number(n, 5)){
            throw std::invalid_argument( "number must be >= 5 and odd number" );
        }
        T a = randN(2, n - 2);
        T r = (T)pow(a, n - 1) % n;
        if (r == 1){
            return true;
        }
        else{
            return false;
        }
    }
};
```

Figure 3.1: Функция теста Ферма

Написал код для вычисления теста Соловья-Штрассена.

```

template<typename T>
// true - probably prime number
// false - composite number
static bool SolovayStrassenTest(const T& n){
    if (!check_number(n, 5)){
        throw std::invalid_argument( "number must be >= 5 and odd number" );
    }
    T a = randN(2, n - 2);
    T r = (T)pow(a, (n-1)/2) % n;
    if (r != 1 && r != (n-1)){
        return false;
    }
    T s = symbolJacobi(n, a);
    if (r % n == s){
        return false;
    }
    else{
        return true;
    }
}

```

Figure 3.2: Функция теста Соловья-Штрассена

Реализовал вычисление теста Миллера-Рабина

```

template<typename T>
// true - probably prime number
// false - composite number
static bool millerRabinTest(const T& n){
    if (!check_number(n, 5)){
        throw std::invalid_argument( "number must be >= 5 and odd number" );
    }
    T s = dividersCount(n-1, 2);
    T r = (n-1) / (T)pow(2, s);
    T a = randN(2, n-2);
    T y = (T)pow(a, r) % n;
    if (y != 1 && y != n-1){
        T j = 1;
        if (j < s - 1 && y != n - 1){
            y = y*y % n;
            if (y == 1){
                return false;
            }
            j += 1;
        }
        if (y != n-1){
            return false;
        }
    }
    return true;
}

```

Figure 3.3: тест Миллера-Рабина

Реализовал вспомогательную функцию вычисления случайного числа



```
template <typename T>
static decltype(auto) randN(const T& low, const T& high){
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> distr(low, high);
    return distr( & gen);
}
```

Figure 3.4: Вспомогательная функция вычисления случайного числа

Написал вспомогательную функцию вычисления символа Якоби

```

template<typename T>
static decltype(auto) symbolJacobi(const T& n, const T& a){
    if (n < 3){
        throw std::invalid_argument( "number must be >= 3" );
    }
    T g = 1;
    T res, s;
    T nCopy = n;
    T aCopy = a;

    while(true){
        if (aCopy == 0){
            res = 0;
            return res;
        }
        else if (aCopy == 1){
            res = g;
            return res;
        }
        T k = dividersCount(aCopy, 2);
        T a1 = aCopy / (T)pow(2, k);
        if (k % 2 == 0){
            s = 1;
        }
        else{
            if (nCopy % 8 == 1 | nCopy % 8 == -1){
                s = 1;
            }
            else if (nCopy % 8 == 3 | nCopy % 8 == -3){
                s = -1;
            }
        }
        if (a1 == 1){
            res = g * s;
            return res;
        }
        if (nCopy % 4 == 3 && a1 % 4 == 3){
            s = -s;
        }
        aCopy = nCopy % a1;
        nCopy = a1;
        g = g * s;
    }
}

```

Figure 3.5: Вспомогательная функция вычисления символа Якоби

Написал вспомогательную функцию подсчета делителей divider в числе n

```

template<typename T>
static decltype(auto) dividersCount(const T& n, const T& divider){
    int i = 0;
    T nCopy = n;
    while (nCopy % divider == 0){
        i += 1;
        nCopy /= divider;
    }
    return i;
}

```

Figure 3.6: Вспомогательная функция подсчета делителей

Написал вспомогательную функцию проверки числа (она используется в начале каждой функции)

```

template<typename T>
static bool check_number(const T& n, const T& threshold){
    if (n < threshold){
        return false;
    }
    if (n % 2 == 0){
        return false;
    }
    return true;
};

```

Figure 3.7: Вспомогательная функция проверки числа

Написал CMake файл

```

1 cmake_minimum_required(VERSION 3.20)
2 project(Lab05)
3
4 set(CMAKE_CXX_STANDARD 14)
5
6 add_library(Lab05 src/PrimeNumbersHelper.cpp include/PrimeNumbersHelper.h)
7
8 add_executable(main main.cpp)
9 target_link_libraries(main Lab05)

```

Figure 3.8: CMake файл

Написал main.cpp файл, в котором есть тесты реализованных функций.

```

1 #include <iostream>
2 #include "include/PrimeNumbersHelper.h"
3
4 int main() {
5     int a = 127;
6
7     std::cout << std::endl << "true - probably prime number, false - composite number" << std::endl << std::string(55, '@') << std::endl;
8     std::cout << PrimeNumbersHelper::FermatTest(a) << std::endl;
9     std::cout << PrimeNumbersHelper::SolovayStrassenTest(a) << std::endl;
10    std::cout << PrimeNumbersHelper::MillerRabinTest(a) << std::endl;
11
12    return 0;
13 }

```

Figure 3.9: main.cpp файл

Результаты тестов.

```

/home/pi/education/pfur_masters/mat0snovyInfBez/labs/lab05/cmake-build-debug/main

true - probably prime number, false - composite number
-----
0
0
0

Process finished with exit code 0
|

```

Figure 3.10: Результаты тестов

## 4 Выводы

Освоил на практике вероятностные алгоритмы проверки чисел на простоту