Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Теоретической и прикладной математики

Лабораторная работа № 6

по дисциплине «ОСНОВЫ ТЕОРИИ ИНФОРМАЦИИИ КРИПТОГРАФИИ»

Тестирование чисел на простоту и построение больших простых чисел

Факультет:	ПМИ
Группа:	ПМИ-02
Вариант:	7
Студент:	Сидоров Даниил,
	Дюков Богдан
Преподаватель:	Авдеенко Татьяна Владимировна,
	Сивак Мария Алексеевна.

Новосибирск

2026

1. Цель работы

Освоить основные программные методы тестирования чисел на простоту.

2. Задача

I. Реализовать приложение, удовлетворяющее следующим требованиям:

1. Во входном файле хранятся входные данные, необходимые для работы программы (например, разрядность простого числа, вероятность признать составное число простым).

2. Программа генерирует простое число с помощью заданного в варианте теста простоты.

3. Программа выдаёт сгенерированное число, время и количество итераций основного цикла, потребовавшихся для генерации.

II. С помощью реализованного приложения выполнить следующие задания:

1. Протестировать правильность работы приложения на числах разной длины.

2. Сделать выводы о проделанной работе.

Вариант	Тест
7	Тест Соловея-Штрассена

3. Метод решения задачи

Любое нечетное *n* тогда и только тогда является простым, если для всех *a* выполняется следующее сравнение:

$$a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) (mod\ n) \quad (2)$$

где $\left(\frac{a}{n}\right)$ символ Лежандра.

Р. Соловей и В. Штрассен предложили следующий вероятностный тест для проверки простоты числа:

1. Выбираем случайным образом число *a* из интервала [1; n–1].

2. Проверяем с помощью алгоритма Евклида условие НОД (*a*, n) = 1. Если оно не выполняется, то *n* – составное.

3. Проверяем выполнимость сравнения (2). Если оно не выполняется, то *n* – составное. Если сравнение выполнено, то ответ неизвестен (и тест можно повторить еще раз).

Данный тест полностью аналогичен тесту на основе малой теоремы Ферма, однако, он обладает решающим преимуществом – при его использовании возникает только две ситуации:

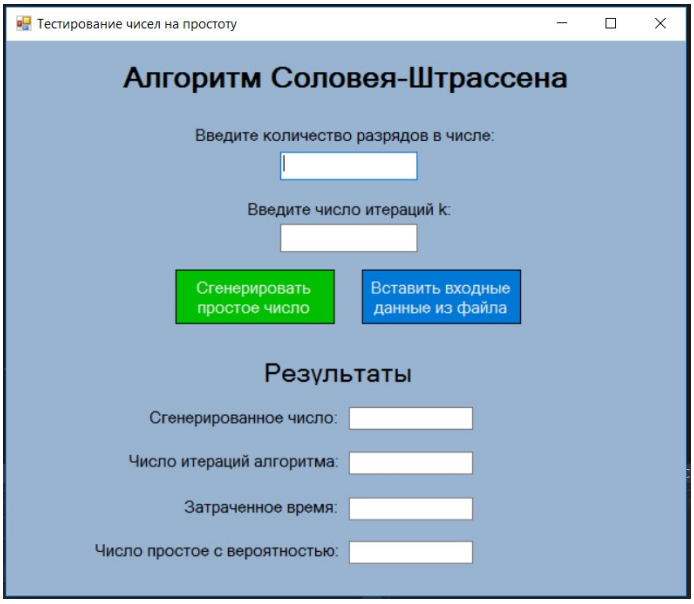
- число *n* простое и тест всегда говорит “не известно”;

• число *n* составное и тест с вероятностью успеха не меньше 0.5 дает ответ “*n* – составное”.

После повторения теста *k* раз вероятность неотбраковки составного числа не превосходит 0.5^k.

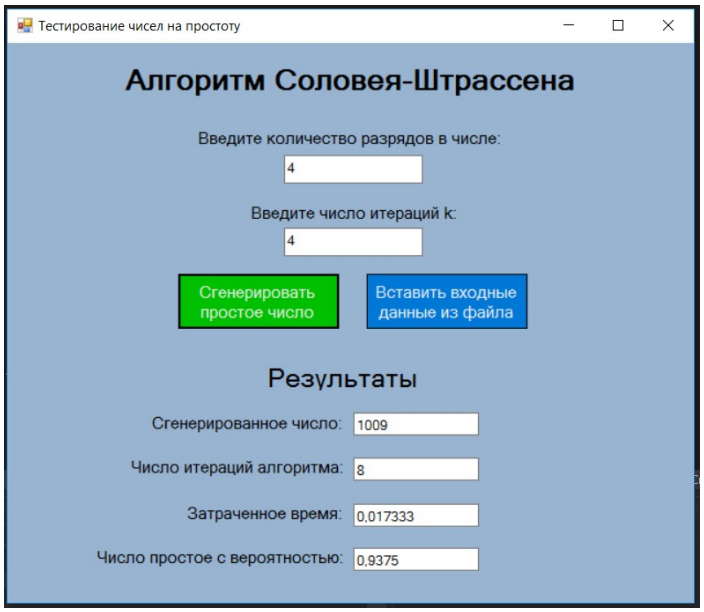
4. Разработанное программное средство

Разработанное программное средство представляет собой приложение Windows Forms. Вся лабораторная работа сделана в одном окне:



У пользователя есть возможность ввода количества разрядов в простом числе, которое будет получено в результате, а также числа итераций, сколько раз для каждого входного числа будет выполняться тест Соловея-Штрассена. Эти данные можно получить из одного файла, в котором через пробел написаны количество разрядов и число итераций.

Протестируем приложение с простейшими входными данными:



Простое 4-х значное число было найдено за 0.017 секунд и 8 итераций основного цикла. Вдобавок ко всему, на форму была выведена вероятность того, что найденное число – простое. Данная вероятность была найдена по формуле: $1 - 2^{-k}$.

5. Код программы

Файл SoloveyStrassen.h

```
#pragma once
#ifndef _SoloveyStrassen_H
#define _SoloveyStrassen_H

#include <cstring>
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <vector>
#include <chrono>
#include <cassert>
#include <string>
#include <msclr\marshal_cppstd.h>
#include <iomanip>

#define ll long long

// Алгоритм Евклида для вычисления НОД
ll Nod(ll a, ll b);

// Модульное возведение в степень
ll ModularExponentiation(ll a, ll exponent, ll n);

// Проверка числа на четность
bool Odd(int i);

// Рекурсивное вычисление символа Якоби
int GetJacobiSymbol(int a, int b);

// Тест Соловея-Штрассена
bool CheckingSoloveyStrassen(ll n, int iteration, int& AllIter);

// Получить информацию из файла
std::pair<int, int> GetInfoFromFile(std::string fileName);

// Записать информацию в файл
void WriteInfoToFile(std::string fileName, int result, int AllIter,
std::chrono::duration<double, std::milli> duration, std::string ver);

#endif
```

Файл SoloveyStrassen.cpp

```
#include "SoloveyStrassen.h"

// Алгоритм Евклида для вычисления НОД
ll Nod(ll a, ll b)
{

    if (b == 0)
    {
        return a;
    }
```

```
    return Nod(b, a % b);
}

// Модульное возведение в степень
ll ModularExponentiation(ll a, ll exponent, ll n)
{
    ll x = 1;
    ll y = a;

    while (exponent > 0)
    {

        if (exponent % 2 == 1)
        {
            x = (x * y) % n;
        }

        y = (y * y) % n;
        exponent = exponent / 2;
    }

    return x % n;
}

// Проверка числа на четность
bool Odd(int i)
{

    if (i % 2 == 1)
    {
        return true;
    }

    return false;
}

// Рекурсивное вычисление символа Якоби
int GetJacobiSymbol(int a, int b)
{
    int g;

    if (b != 2)
        assert(Odd(b));

    if (a >= b)
        a %= b;

    if (a == 0)
        return 0;

    if (a == 1)
        return 1;

    if (a < 0)
        if ((b - 1) / 2 % 2 == 0)
            return GetJacobiSymbol(-a, b);
        else
            return -GetJacobiSymbol(-a, b);

    if (a % 2 == 0)
        if (((b * b - 1) / 8) % 2 == 0)
            return GetJacobiSymbol(a / 2, b);
        else
            return -GetJacobiSymbol(a / 2, b);

    g = Nod(a, b);
    assert(Odd(a));

    if (g == a)
```

```
        return 0;
    else
        if (g != 1)
            return GetJacobiSymbol(g, b) * GetJacobiSymbol(a / g, b);
        else
            if (((a - 1) * (b - 1) / 4) % 2 == 0)
                return GetJacobiSymbol(b, a);
            else
                return -GetJacobiSymbol(b, a);

}

// Тест Соловея-Штрассена
bool CheckingSoloveyStrassen(ll n, int iteration, int& allIter)
{

    if (n < 2 || (n != 2 && n % 2 == 0))
    {
        return false;
    }

    for (int i = 0; i < iteration; i++)
    {
        allIter++;

        ll a = rand() % (n - 1) + 1;

        if (Nod(a, n) != 1)
        {
            return false;
        }

        ll jacobian = (n + GetJacobiSymbol(a, n)) % n;
        ll mod = ModularExponentiation(a, (n - 1) / 2, n);

        if (!jacobian || mod != jacobian)
        {
            return false;
        }

    }

    return true;
}

// Получить информацию из файла
std::pair<int, int> GetInfoFromFile(std::string fileName)
{
    std::ifstream file;
    file.open(fileName);

    // Если файл не открыт, генерируем исключение
    if (!file.is_open())
    {
        throw std::invalid_argument("The file is not open");
    }

    std::pair<int, int> Input;
    file >> Input.first;
    file >> Input.second;
    file.close();

    return Input;
}

// Записать информацию в файл
void WriteInfoToFile(std::string fileName, int result, int AllIter,
std::chrono::duration<double, std::milli> duration, std::string ver)
{
```

```
    std::ofstream file;
    file.open(fileName);

    // Если файл не открыт, генерируем исключение
    if (!file.is_open())
    {
        throw std::invalid_argument("The file is not open");
    }

    file << result << " " << AllIter << " " << duration.count() << " " << ver << std::endl;
    file.close();
}
```

6. Тесты

Разрядность	Число итераций k	Полученное число	Число итераций алгоритма	Время работы	Вероятность простоты числа	Комментарий
3	50	101	51	0.045	0.99	Результаты корректны
7	50	1000003	53	0.088	0.99	Результаты корректны
10	50	1000000007	57	0.112	0.99	Результаты корректны
10	1	1000000007	8	0.012	0.5	Меньше итераций – быстрее программа
10	0	1000000001	1	0.0001	0	Ошибка, так как 0 повторений, а значит 100% шанс ошибки
1	1	2	1	0.001	0.5	Результаты корректны
11	10	-	-	-	-	Предел допустимого числа $<2^{31}$
0	10	-	-	-	-	Некорректный ввод

7. Вывод

В ходе проведения лабораторной работы мы освоили различные способы получения простых чисел и тесты для определения простых чисел.