Учреждение образования

«Белорусский государственный технологический университет»

**Отчет по лабораторной работе №1**

по дисциплине: «Защита информации и надежность информационных систем»

Исполнитель:

Воликов Д. А., ФИТ 4-7

Руководитель:

Асс. Савельева М. Г.

Минск 2024

# Лабораторная работа №1. Основы теории чисел и их использование в криптографии

**Цель**: приобретение практических навыков выполнения операций с числами для решения задач в области криптографии и разработка приложений для автоматизации этих операций.

**Задачи**:

1. Закрепить теоретические знания по высшей арифметике.
2. Научиться практически решать задачи с использованием простых и взаимно простых чисел, вычислений по правилам модулярной арифметики и нахождению обратных чисел по модулю.
3. Ознакомиться с особенностями реализации готового программного средства L\_PROST и особенностями выполнения с его помощью операций над простыми числами.
4. Разработать приложение для реализации указанных преподавателем операций с числами.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения эксперимента с использованием приложения и результатов эксперимента.

## Теоретические сведения

Множество всех целых чисел (обозначим буквой *Z*) есть набор всех действительных чисел без дробной части: {..., –3, –2, –1, 0, 1, 2, 3, ...}.

Натуральные числа являются подмножеством целых чисел и образуют множество *N*: {1, 2, 3, ...}.

Делимость – одно из основных понятий теории чисел. Если для некоторого целого числа a и натурального числа *b* существует целое число *q*, при котором *bq* = *a*, то говорят, что число *a* делится на *b*. В этом случае *b* называется делителем числа *a*, а *a* называется кратным числу *b*.

Делитель a называется собственным делителем числа *b*, если 1 < |*a*| < |*b*|, и несобственным – в противном случае.

Всякое целое число *а* можно представить с помощью положительного целого числа *b* равенством вида *а* = *bq* + *r*, 0 ≤ *r* ≤ *b*. Число *q* называется неполным частным, а число *r* – остатком отделения *а* на *b*.

Натуральное число *n* называется простым, если *n* > 1 и не имеет положительных делителей, отличных от 1 и *n*.

Любое составное число представляется уникальным образом в виде произведения простых чисел; иначе еще говорят, что разложение числа на простые множители однозначно.

Всякое натуральное число *n*, кроме 1, можно представить как произведение простых множителей: *n* = *p*1*p*2*p*3...*p*z, *z* > 1.

Простых чисел бесконечно много, причем существует примерно *n* / ln(*n*) простых чисел, меньших числа *n*.

Наименьший простой делитель составного числа *n* не превышает *√n*, поэтому для проверки простоты числа достаточно проверить его делимость на 2 и на все нечетные (а еще лучше простые) числа, не превосходящие *√n*.

Любое четное число, большее 2, представимо в виде суммы двух простых чисел, а любое нечетное, большее 5, представимо в виде суммы трех простых чисел.

Натуральное число *n* называется составным, если *n* > 1 и имеет по крайней мере один положительный делитель, отличный от 1 и *n*.

Первый алгоритм нахождения простых чисел, не превышающих n, был придуман Эратосфеном во II в. до н. э. и известен сейчас как «решето Эратосфена». Его суть в последовательном исключении из списка целых чисел от 1 до *n* (или из сокращенного диапазона, например от *m* до *n*, 1 < *m* ≤ *n*) чисел, кратных 2, 3, 5 и другим простым числам, уже найденным «решетом».

Наибольшее целое число, которое делит без остатка числа *a* и *b*, называется наибольшим общим делителем этих чисел – НОД (*a*, *b*). Простым и эффективным средством вычисления НОД (*a*, *b*) является алгоритм Евклида.

Взаимно простыми являются целые числа, наибольший общий делитель которых равен 1.

Целые числа *a* и *b* взаимно просты тогда и только тогда, когда существуют такие целые числа *u* и *v*, что выполняется равенство:

*au* + *bv* = 1

Если НОД (*a, b*) = *d*, то справедливо следующее соотношение (соотношение Безу):

*au + bv = d*

Количество натуральных чисел, не превосходящих *n* и взаимно простых с *n*, называется функцией Эйлера и обозначается *φ*(*n*).

Если *p* – простое число, то *φ*(*p*) = *p* – 1, если числа *p* и *q* являются простыми и *p ≠ q*, то *φ*(*p*) = (*p* – 1)(*q* – 1).

Модулярная арифметика так же коммутативна, ассоциативна и дистрибутивна, как и обычная арифметика. В силу этих свойств сравнения можно почленно складывать, вычитать, умножать, возводить в степень.

Малая теорема Ферма. Если *n* – простое число, а число *а* не кратно *n*, то справедливо:

*an ≡* 1 *mod n*

В соответствии с обобщением Эйлера приведенной теоремы, если НОД (*а, n*) = 1, то справедливо:

*aφ*(*n*) mod *n* *≡* 1

Последнее выражение можно переписать в следующем виде:

*aφ*(*n*) - 1 mod *n* *≡ a-1* mod *n*

## 1.2. Практическое задание

**Вариант 2**. По условию *m* = 521, *n* = 553.

**Задание №1**. Используя *L\_PROST*, найти все простые числа в интервале [2, *n*]. Значение *n* соответствует варианту. Подсчитать количество простых чисел в указанном интервале. Сравнить это число с *n/*ln(*n*).

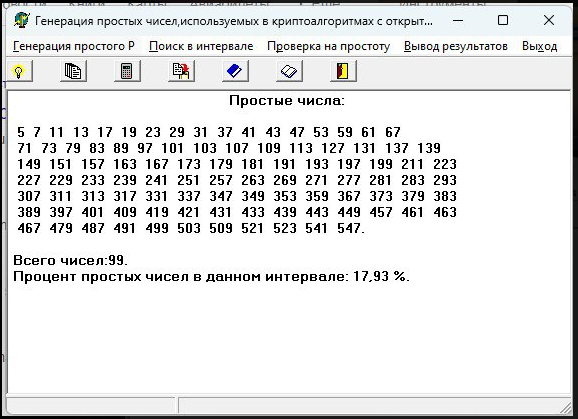


Рисунок 1.1 – Результат программы *L\_PROST* для интервала [2, *n*]

По подсчётам программы простых чисел в интервале [2, *n*] – 101. По подсчётам формулы – 87.56. Результат программы близок к значению формулы, но всё же разница довольно большая.

**Задание №2**. Повторить п. 1 для интервала [*m, n*]. Сравнить полученные результаты с «ручными» вычислениями, используя «решето Эратосфена».

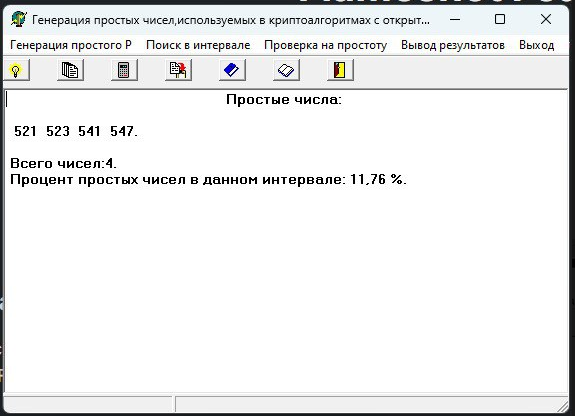


Рисунок 1.2 – Результат программы *L\_PROST* для интервала [*m, n*]

По результатам программы количество простых чисел в интервале [*m, n*] равно 4-ём. Теперь просчитаем всё это способом решета Эратосфена.

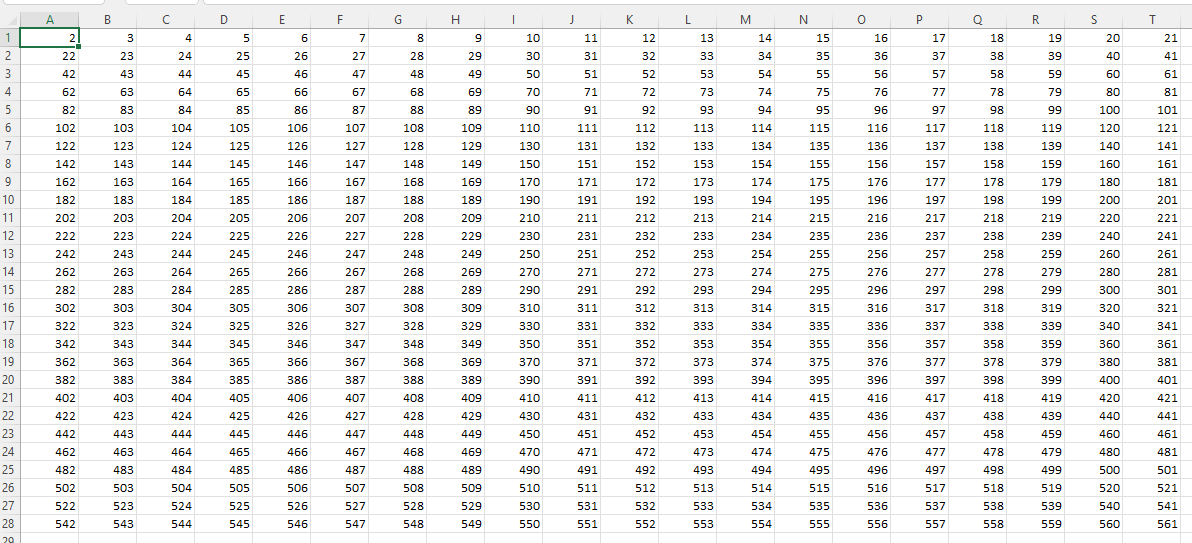


Рисунок 1.3 – Решето Эратосфена до отсекания чисел

Отсечём из решета все числа, которые делятся на 2, кроме самой двойки.

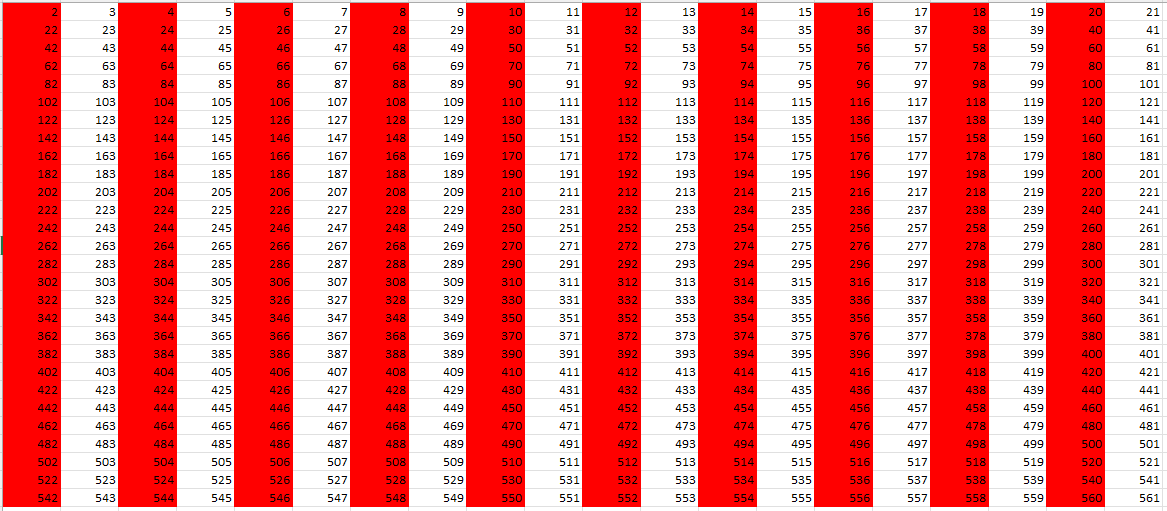


Рисунок 1.4 – Решето Эратосфена после отсечения чисел кратных 2

Далее отсечём все числа, которые делятся на 3, кроме самой тройки.

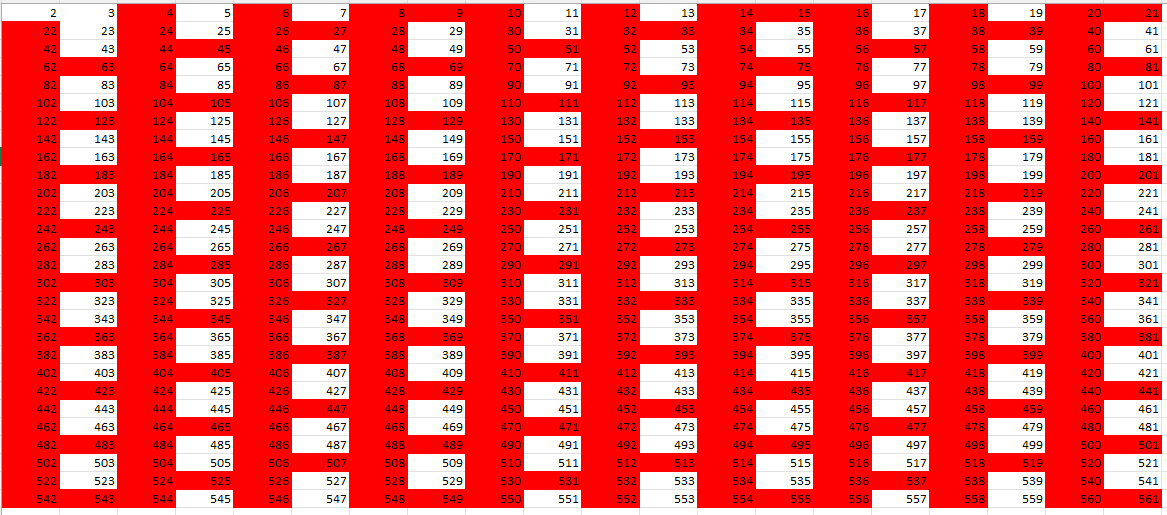


Рисунок 1.5 – Решето Эратосфена после отсечения чисел кратных 3

Далее отсечём все числа, которые делятся на 5, кроме самой пятёрки.

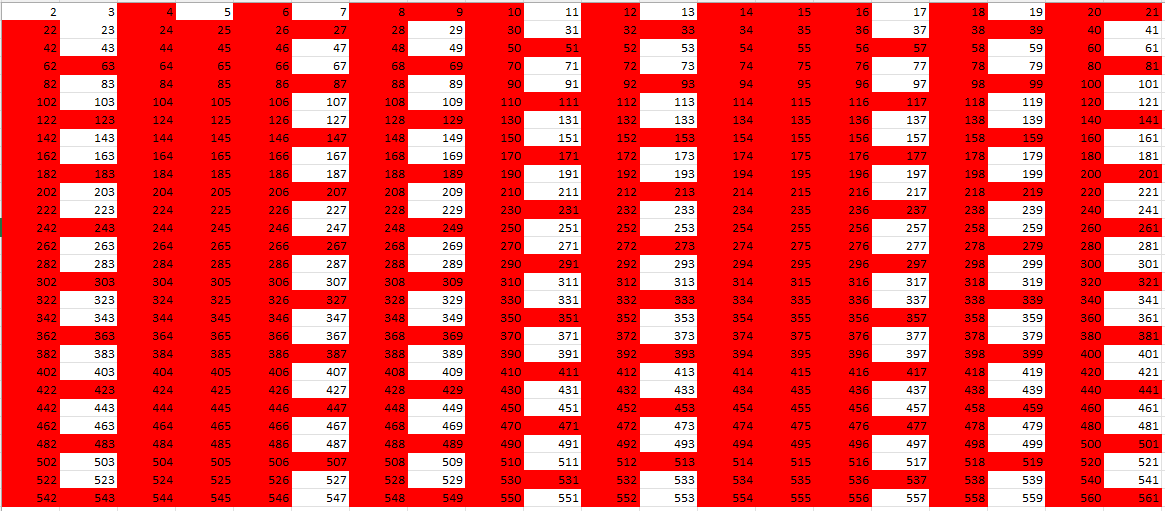


Рисунок 1.6 – Решето Эратосфена после отсечения чисел кратных 5

Далее отсечём числа, которые делятся на 7, кроме самой семёрки.

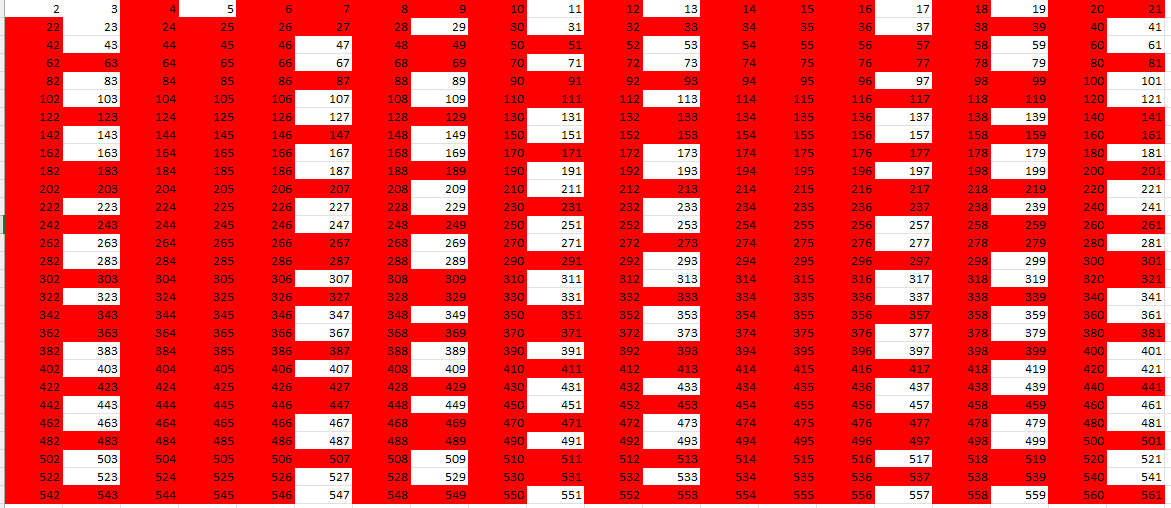


Рисунок 1.7 – Решето Эратосфена после отсечения чисел кратных 7

Теперь отсечём числа, кратные 11, 13, 17, 19 и 23. Этих чисел должно хватить по свойству.

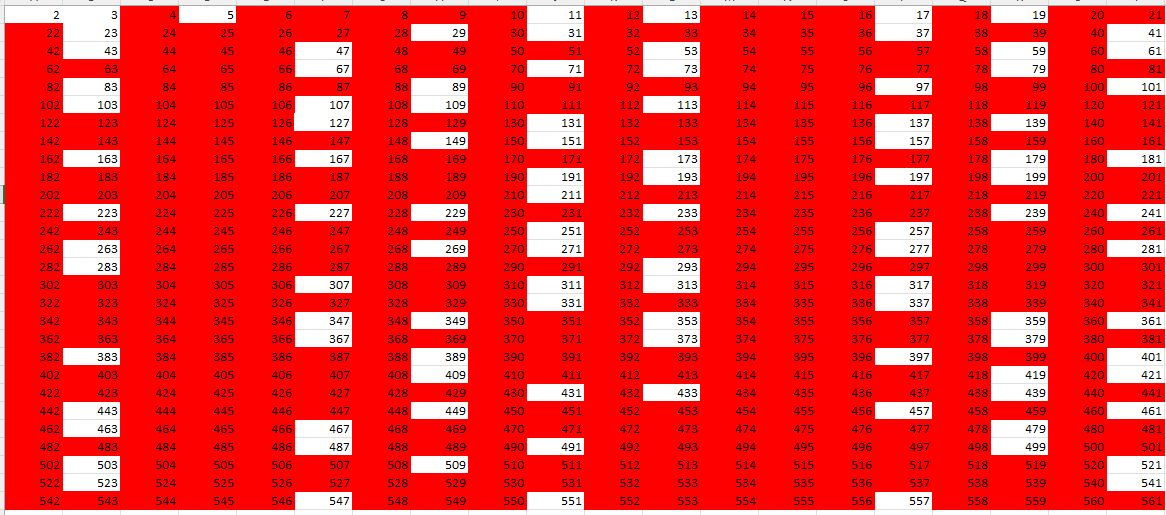


Рисунок 1.8 – Решето Эратосфена после отсечения оставшихся чисел

По расчётам способом решета Эратосфена, в интервале [2, *n*] – 101 простых чисел, из них только 4, которые входят в интервал [*m, n*].

**Задание №3**. Записать числа *m* и *n* в виде произведения простых множителей (форма записи – каноническая).

Число 521 в каноническом виде: 521 = 1 \* 521 (число простое).

Число 553 в каноническом виде: 553 = 1 \* 7 \* 79.

**Задание №4**. Проверить, является ли число, состоящее из конкатенации цифр *m* ǀǀ *n* (табл. 1.2), простым.

Исходное число 521553, которое получилось путём конкатенации *n* и *m*, является непростым, ибо сумма цифр делится на 3 (5 + 2 + 1 + 5 + 5 + 3 = 21 ⋮ 3).

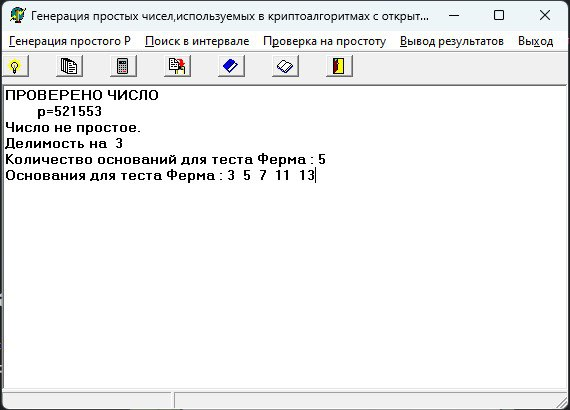


Рисунок 1.9 – Результат программы *L\_PROST* проверки на простоту

**Задание №5**. Найти НОД (*m, n*).

Обратимся к заданию 3 и проанализируем канонические представления каждого из чисел. Так, как у них нет общих множителей, кроме единицы, то и НОД (*m, n*) равен единице.

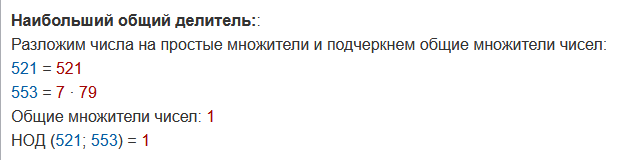


Рисунок 1.10 – Результат нахождения НОД (*m, n*)

**Задание №6 - 7**. Разработать авторское приложение в соответствии с целью лабораторной работы. Приложение должно реализовывать следующие операции:

* вычислять НОД двух либо трех чисел;
* выполнять поиск простых чисел.

С помощью созданного приложения выполнить задания по условиям п.1 и п.2.

Сперва проверим работу программы для нахождения НОД у двух заданных чисел.

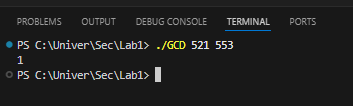


Рисунок 1.11 – Результат авторской программы нахождения НОД 2-ух чисел

Как можем заметить, результат совпал с результатом в задании №5. Теперь посчитаем НОД для трёх чисел.

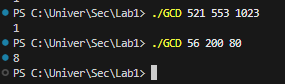


Рисунок 1.12 – Результат авторской программы нахождения НОД 3-ёх чисел

Как можем заметить, результат действительно верный. Значит алгоритм написана правильно, и программа выдаёт истинные значения.

Теперь проверим поиск простых чисел в заданных диапазонах, аналогичных заданиям 2 и 3.

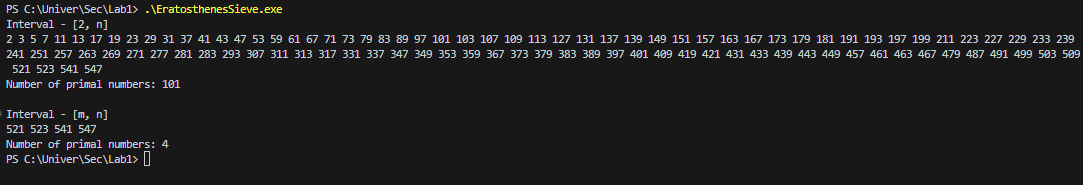


Рисунок 1.13 – Результат авторской программы нахождения простых чисел

Действительно, программа выдаёт такой же результат, как и ручной расчёт методом решета Эратосфена. Из этого следует, что программа написана правильно и выдаёт истинные значения.

**Вывод**: в ходе этой лабораторной работы я повторил основные свойства теории чисел и их использование в криптографии. После я написал программу, которая считает НОД двух (трёх) чисел, а также ищет простые числа в определённом диапазоне.

# Листинг «Лабораторная работа №1»

#include <stdio.h>

#include <stdlib.h>

int GCD(int first, int second)

{

    if (first <= 0 || second <= 0)

    {

        return -1;

    }

    int remainder = 0;

    while (first % second != 0)

    {

        remainder = first % second;

        first = second;

        second = remainder;

    }

    return remainder;

}

int main(int argc, char \*argv[])

{

    if (argc < 3)

    {

        printf("No GCD arguments: 2 int requiered");

        return -1;

    }

    if (argc == 3)

    {

        printf("%d", GCD(atoi(argv[1]), atoi(argv[2])));

    }

    else

    {

        printf("%d", GCD(GCD(atoi(argv[1]), atoi(argv[2])), atoi(argv[3])));

    }

    return 0;

}

Листинг 1.1 – Программа нахождения НОД двух (трёх) чисел

#include <stdio.h>

#include <stdlib.h>

#include <memory.h>

int \*Sieve(int n)

{

    int \*array = (int \*)malloc(n \* sizeof(int));

    if (array == NULL)

    {

        printf("Memory allocating error");

        return NULL;

    }

    memset(array, 0, n \* sizeof(int));

    array[0] = 1;

    array[1] = 1;

    for (int i = 2; i \* i < n; i++)

    {

        if (array[i] == 0)

        {

            for (int j = i \* i; j <= n; j += i)

            {

                array[j] = 1;

            }

        }

    }

    return array;

}

int main()

{

    int m = 521;

    int n = 553;

    int counter = 0;

    int \*array = Sieve(n + 1);

    printf("Interval - [2, n]\n");

    for (int i = 0; i < n + 1; i++)

    {

        if (array[i] == 0)

        {

            printf("%d ", i);

            counter++;

        }

    }

    printf("\nNumber of primal numbers: %d\n\n", counter);

    counter = 0;

    printf("Interval - [m, n]\n");

    for (int i = m; i < n + 1; i++)

    {

        if (array[i] == 0)

        {

            printf("%d ", i);

            counter++;

        }

    }

    printf("\nNumber of primal numbers: %d\n", counter);

    free(array);

    return 0;

}

Листинг 1.2 – Программа для нахождения простых чисел