Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Основы теории чисел и их использование в криптографии**

Студент: Буданова К. А.

ФИТ 3 курс 5 группа

Преподаватель:

Савельева Маргарита Геннадьевна

1. **Цель работы**

Создать приложение на C#, которое позволяет:

* находить все простые числа в интервале [2; *n*];
* сравнивать это число с *n*/ln(*n*);
* выполнять поиск простых чисел на промежутке [367; 401];
* сравнить полученные результаты c вычислениями, используя «решето Эратосфена»;
* записывать числа *m* и *n* в виде произведения простых множителей (форма записи – каноническая);
* проверять, является ли число, состоящее из конкатенации цифр *m*ǀǀ*n*, простым;
* вычислять НОД двух и трех чисел.

1. **Ход работы**

Для нахождения всех простых чисел в интервале [2; *n*] была разработана функция FindPrimes(). Её реализация и результат выполнения представлены на рисунках 2.1, 2.2 и 2.3.

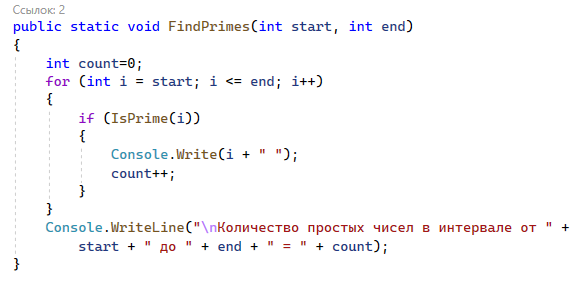


Рис. 2.1 – Функция для нахождения простых чисел в интервале

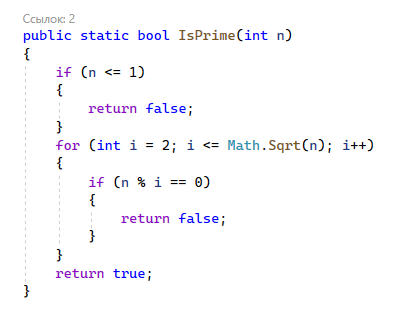


Рис. 2.2 – Функция, которая определяет, является ли число простым

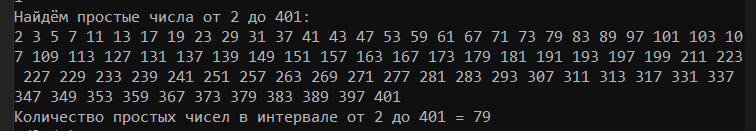


Рис. 2.3 – Результат выполнения функции FindPrimes()

Также, нужно было сравнить полученный результат с *n*/ln(*n*). Он представлен на рисунке 2.4. Такой результат обуславливается тем, что *n*/ln(*n*) даёт погрешность на больших промежутках.



Рис. 2.4 – Получение количества простых чисел с использованием *n*/ln(*n*)

Далее, нужно было выполнить поиск простых чисел на промежутке [*n*; *m*] и сравнить результат, полученный при использовании «решета Эратосфена».

Функция, реализующая «решето Эратосфена» представлена на рисунке 2.5, а результат сравнения этой функции с функцией FindPrimes() представлен на рисунке 2.6.

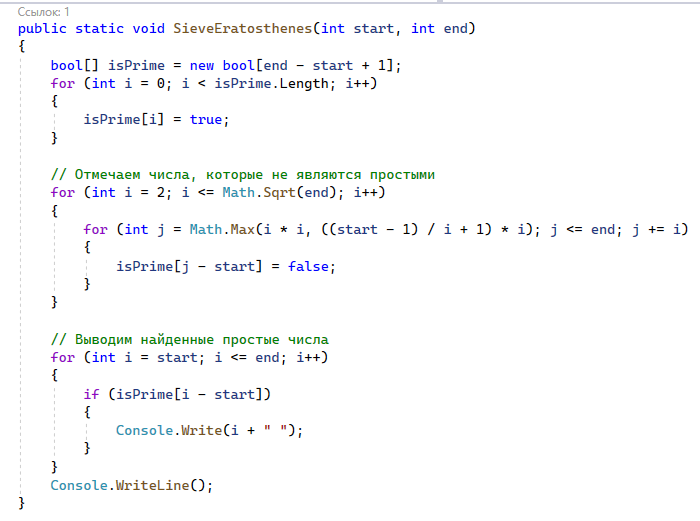


Рис. 2.5 – Решето Эратосфена

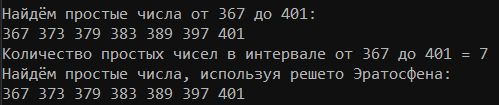


Рис. 2.6 – Сравнение функций FindPrimes() и SieveEratosthenes()

Следующим шагом нужно было сравнить полученные результаты с «ручными» вычислениями, используя «решето Эратосфена» на промежутке [367; 401]. Для этого выполним следующие шаги:

1. Выпишем числа от 367 до 401:

367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401.

1. Воспользуемся свойством 3 простых чисел и вычислим , т. е. меньше 20.
2. Удалим последовательно все числа, делящиеся на простые числа от 2 до 20. Такими простыми числами являются: 2, 3, 5, 7, 11, 13, 17, 19.

Числа, которые делятся на 2: 368, 370, 372, 374, 376, 378, 380, 382, 384, 386, 388, 390, 392, 394, 396, 398, 400.

Оставшиеся числа: 367, 369, 371, 373, 375, 377, 379, 381, 383, 385, 387, 389, 391, 393, 395, 397, 399, 401.

Числа, которые делятся на 3: 369, 375, 381, 387, 393, 399.

Оставшиеся числа: 367, 371, 373, 377, 379, 383, 385, 389, 391, 395, 397, 401.

Числа, которые делятся на 5: 385, 395.

Оставшиеся числа: 367, 371, 373, 377, 379, 383, 389, 391, 397, 401.

Числа, которые делятся на 7: 371.

Оставшиеся числа: 367, 373, 377, 379, 383, 389, 391, 397, 401.

Числа, которые делятся на 13: 377.

Оставшиеся числа: 367, 373, 379, 383, 389, 391, 397, 401.

Числа, которые делятся на 17: 391.

Оставшиеся числа: 367, 373, 379, 383, 389, 397, 401.

Числа, которые делятся на 19: 389.

Оставшиеся числа: 367, 373, 379, 383, 397, 401.

В итоге, после выполнения всех операций в «решете» останутся числа: 367, 373, 379, 383, 397, 401. Эти числа совпадают с найденными ранее простыми числами на промежутке [367, 401].

Далее, нужно было записать числа *m* и *n* в виде произведения простых множителей (форма записи – каноническая). Для этого была разработана функция WriteFactors(), код которой представлен на рисунке 2.7, а результат – на рисунке 2.8.

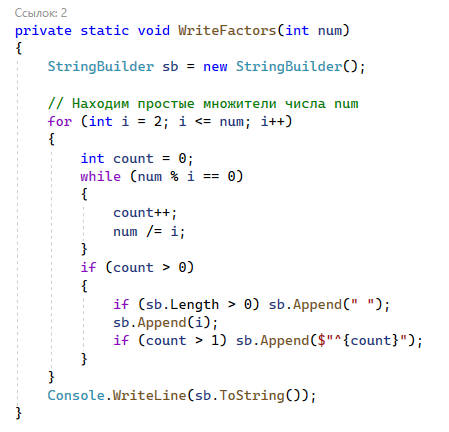


Рис. 2.7 – Функция для записи числа в виде произведения простых множителей в каноническом виде

Поскольку 367 и 401 – простые числа, для наглядности используется число 18:



Рис. 2.8 – Результат выполнения функции WriteFactors()

Следующим шагом проверим, является ли число, состоящее из конкатенации цифр *m*ǀǀ*n*, простым. Для этого была разработана функция IsConcatenationPrime(), представленная на рисунке 2.9.

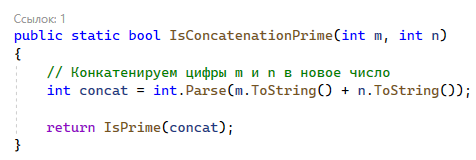


Рис. 2.9 – Функция IsConcatenationPrime()

Чтобы найти НОД двух или трёх чисел была разработана функция FindGCD(), реализующая алгоритм Евклида. Описание данной функции представлено на рисунке 2.10.

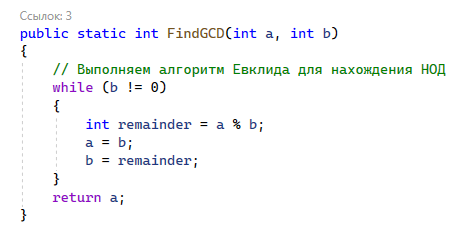


Рис. 2.10 – Функция для нахождения НОД

Для нахождения НОД трёх чисел, сначала мы находим НОД для двух чисел, затем находим НОД для третьего числа и первых двух чисел. Пример представлен на рисунке 2.11.



Рис. 2.11 – Функция для нахождения НОД

Результат выполнения функции представлен на рисунке 2.12.



Рис. 2.12 – Нахождение НОД для двух и трёх чисел

**Вывод**

В ходе лабораторной работы были приобретены навыки выполнения операций с числами для решения задач в области криптографии, а также разработано приложение для автоматизации этих операций.