1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации
5. **К У Р С О В А Я Р А Б О Т А**

**«Криптосистема с открытым ключом»**

по дисциплине «Структуры данных»

1. Выполнил
2. студент гр. 4831001/10003 Улановский Г.А.

*<подпись>*

1. Преподаватель
2. асс. преподавателя Панков И.Д.

<*подпись*>

1. Санкт-Петербург
2. 2023

**СОДЕРЖАНИЕ**

[**Введение 3**](#_Toc107145394)

[1.1. Теоретические сведения 3](#_Toc107145395)

[1.2. Цель и задачи работы 6](#_Toc107145396)

[**2. Ход работы 7**](#_Toc107145397)

[2.1. Оптимальная структура данных 7](#_Toc107145398)

[2.2. Используемые функции и алгоритмы 8](#_Toc107145399)

[2.3. Реализация ввода данных 18](#_Toc107145400)

[2.4. Логирование 20](#_Toc107145401)

[**3. Результат работы 21**](#_Toc107145402)

[**4. Тест на быстродействие 22**](#_Toc107145403)

[**5. Заключение 26**](#_Toc107145404)

[**Приложение А 27**](#_Toc107145405)

[**Приложение Б 31**](#_Toc107145406)

[**Приложение В 49**](#_Toc107145407)

[**Приложение Г 50**](#_Toc107145408)

[**Приложение Д 54**](#_Toc107145409)

[**Приложение Е 55**](#_Toc107145410)

[**Приложение Ё 56**](#_Toc107145411)

[**Приложение Ж 58**](#_Toc107145412)

[**Приложение З 66**](#_Toc107145413)

[**Приложение И 67**](#_Toc107145414)

[**Приложение Й 69**](#_Toc107145415)

# Введение

## Теоретические сведения

RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) — [криптографический алгоритм с открытым ключом](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC), основывающийся на [вычислительной сложности](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) [задачи факторизации](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D1%86%D0%B5%D0%BB%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB) больших [целых чисел](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE).

Криптосистема RSA стала первой системой, пригодной и для [шифрования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), и для [цифровой подписи](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%8C). Алгоритм используется в большом числе криптографических приложений, включая [PGP](https://ru.wikipedia.org/wiki/PGP), [S/MIME](https://ru.wikipedia.org/wiki/S/MIME), [TLS](https://ru.wikipedia.org/wiki/TLS)/[SSL](https://ru.wikipedia.org/wiki/SSL), [IPSEC](https://ru.wikipedia.org/wiki/IPSEC)/[IKE](https://ru.wikipedia.org/wiki/IKE).

Криптографические системы с открытым ключом используют так называемые [односторонние функции](https://ru.wikipedia.org/wiki/%D0%9E%D0%B4%D0%BD%D0%BE%D1%81%D1%82%D0%BE%D1%80%D0%BE%D0%BD%D0%BD%D1%8F%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F), которые обладают следующим свойством:

если известно значение x, то значение f(x) вычислить относительно просто;

если известно значение f(x), то для вычисления значения x нет простого (эффективного) пути.

Под односторонностью понимается не математически доказанная однонаправленность, а практическая невозможность вычислить обратное значение, используя современные вычислительные средства, за обозримый интервал времени.

В основу криптографической системы с открытым ключом RSA положена сложность [задачи факторизации](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D1%86%D0%B5%D0%BB%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB) произведения двух больших простых чисел. Для шифрования используется операция [возведения в степень по модулю](https://ru.wikipedia.org/wiki/%D0%92%D0%BE%D0%B7%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B2_%D1%81%D1%82%D0%B5%D0%BF%D0%B5%D0%BD%D1%8C_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E) большого числа. Для дешифрования (обратной операции) за разумное время необходимо уметь вычислять [функцию Эйлера](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%AD%D0%B9%D0%BB%D0%B5%D1%80%D0%B0) от данного большого числа, для чего необходимо знать разложение числа на простые множители.

В криптографической системе с открытым ключом каждый участник располагает как открытым ключом ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) public key), так и закрытым ключом ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) private key). В криптографической системе RSA каждый ключ состоит из пары целых чисел. Каждый участник создаёт свой открытый и закрытый ключ самостоятельно. Закрытый ключ каждый из них держит в секрете, а открытые ключи можно сообщать кому угодно или даже публиковать их. Открытый и закрытый ключи каждого участника обмена сообщениями в криптосистеме RSA образуют «согласованную пару» в том смысле, что они являются взаимно обратными, то есть:

Для любых допустимых пар открытого и закрытого ключей p и s существуют функции шифрования E (x, p) и расшифрования D (x, s) такие, что для любого сообщения m верно:

m = D (E (m, p), s) = E (D (m, s), p)

RSA-ключи генерируются следующим образом:

1. выбираются два различных [случайных простых числа](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) p и q заданного размера;
2. вычисляется их произведение n = p \* q, которое называется модулем;
3. вычисляется значение [функции Эйлера](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%AD%D0%B9%D0%BB%D0%B5%D1%80%D0%B0) от числа n: phi(n)= (p-1)\*(q-1);
4. выбирается натуральное число e, меньшее phi(n) и [взаимно простое](https://ru.wikipedia.org/wiki/%D0%92%D0%B7%D0%B0%D0%B8%D0%BC%D0%BD%D0%BE_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%8B%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%B0) со значением функции phi(n), которое называется открытой экспонентой.

Обычно в качестве e берут простые числа, содержащие небольшое количество единичных бит в [двоичной записи](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F), например,  
простые из [чисел Ферма](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%B0_%D0%A4%D0%B5%D1%80%D0%BC%D0%B0): 17, 257 или 65537, так как в этом случае время, необходимое для шифрования с использованием  
[быстрого возведения в степень](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%B1%D1%8B%D1%81%D1%82%D1%80%D0%BE%D0%B3%D0%BE_%D0%B2%D0%BE%D0%B7%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2_%D1%81%D1%82%D0%B5%D0%BF%D0%B5%D0%BD%D1%8C), будет меньше;

1. вычисляется число d (секретная экспонента), [мультипликативно обратное](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_%D1%87%D0%B8%D1%81%D0%BB%D0%BE) к числу e по модулю phi(n), то есть число, удовлетворяющее [сравнению](https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E):
2. пара чисел (e, n) публикуется в качестве открытого ключа RSA;
3. (d, n) играет роль закрытого ключа RSA и держится в секрете.

Операция шифрования сообщения m с помощью ключей e и n и получения шифротекста c выглядит так:

Операция дешифрования шифротекста c с помощью ключей d и n и получения сообщения m выглядит так:

Операция создания цифровой подписи сообщения m с помощью ключей d и n и подписи s выглядит так:

Операция проверки цифровой подписи сообщения m с помощью ключей e и n и подписи s выглядит так:

Сообщение является подлинным, если m и m’ совпадают.

Из-за увеличения вычислительной мощности современных компьютеров для RSA-шифрования на данный момент не используют ключи размером меньше 2048 бит. Кроме того, RSA обычно входит в состав гибридных криптосистем для шифрования сеансового ключа, так как скорость шифрования и дешифрования RSA чрезвычайно низкая.

Алгоритм быстрого умножения методом Карацубы - метод быстрого умножения, который позволяет перемножать два [n-значных](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B7%D0%B8%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) числа с битовой [вычислительной сложностью](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C):

Два n-битовых числа можно представить в виде ax+b, cx+d, где x=2^(n/2); a,b,c,d < 2^(n/2).

Умножение на 2^(n/2) ([битовый сдвиг](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%81%D0%B4%D0%B2%D0%B8%D0%B3)) и сложение делаются за пренебрежимо малое время O(1), поэтому задача умножения сводится к вычислению коэффициентов [многочлена](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D1%87%D0%BB%D0%B5%D0%BD) (ax+b)(cx+d).

Этот многочлен можно представить в виде

(ax+b) (cx+d)= a\*c\*x^2 + ((a+b)\*(c+d)-a\*c-b\*d)\*x+b\*d

В последнем выражении участвуют три произведения n/2-значных чисел. Если вычислять каждое из них по той же схеме, получится алгоритм с [рекуррентной оценкой времени работы](https://ru.wikipedia.org/wiki/%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D0%B0%D1%8F_%D1%82%D0%B5%D0%BE%D1%80%D0%B5%D0%BC%D0%B0_%D0%BE_%D1%80%D0%B5%D0%BA%D1%83%D1%80%D1%80%D0%B5%D0%BD%D1%82%D0%BD%D1%8B%D1%85_%D1%81%D0%BE%D0%BE%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D1%8F%D1%85)

## Цель и задачи работы

Цель работы – создание консольного приложения, которое позволяет генерировать ключи, подписывать файл, проверять подпись файла, шифровать файл и расшифровывать файл. Программа должна поддерживать запуск из консоли с передачей аргументов и вести log-файл своей деятельности.

Для достижения данной цели были поставлены следующие задачи:

1. Поиск оптимальной структуры данных для представления больших чисел и создание соответствующей библиотеки для работы с ними;
2. Изучение оптимальных алгоритмов выполнения арифметических операций для ускорения вычислений.

# Ход работы

## Оптимальная структура данных

Оптимальной структурой данных для хранения больших чисел была выбрана struct. В реализации в данной структуре содержится размер динамического массива, текущее количество элементов в массиве и динамический массив uint8\_t значений.

Хранение чисел производится путем сохранения их бинарного вида в динамический массив по правилам хранения чисел в стандартных типах, за исключением отсутствия максимального размера и, как следствие, возможности переполнения. Число хранится как последовательность uint8\_t чисел в диапазоне 0-1 в обратном порядке.

Реализованная структура поддерживает хранение целых чисел с возможностью управления знаком через дополнительный код аналогично встроенным целочисленным типам.

Для перечисления exit-кодов для отладки был использован перечисляемый тип Enum.

Хранение «больших» чисел в бинарном виде было выбрано как средство для ускорения вычислений. В данной программе не требуется сохранять и отображать числа в десятичном виде, а бинарная арифметика позволяет ускорить расчеты:

1. Операции умножения и деления на 2 производятся за O(n), так как реализованы с помощью операций сдвига, а не полноценных функций деления и умножения;
2. Вычитание реализуется через сложение двух чисел, операция взятия дополнительного кода быстрее чем операция отрицания для числа с большей степенью счисления;
3. Двоичные числа просто хранить в виде символов или других встроенных типов;
4. Нет необходимости преобразования числа к двоичному виду во время чтения сохраненного числа и в алгоритме быстрого возведения в степень.

## Используемые функции и алгоритмы

В программе реализованы базовые функции для взаимодействия со структурой «больших» чисел:

1. Перевод числа в прямой или дополнительный код (рисунок 1);
2. Инициализация (создание) пустого числа с начальными параметрами (рисунок 2);
3. Копирование чисел (присваивание) (рисунок 3);
4. Бинарные сдвиговые операции (рисунок 4);
5. Операции сравнения (==0 и A==B) (рисунок 5);
6. Операция сложения (рисунок 6);
7. Операция вычитания (рисунок 7);
8. Операция умножения (рисунок 8);
9. Операция деления (рисунок 9);
10. Операция возведения в степень по модулю (рисунок 10);
11. Обычный и модифицированный алгоритм Евклида (рисунок 11).

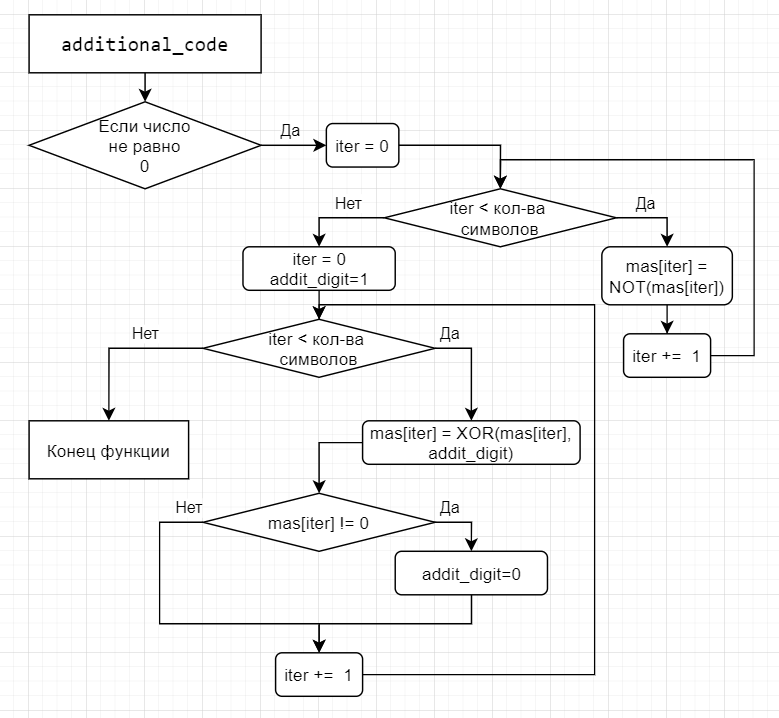


Рисунок 1 – Блок-схема функции «additional\_code»

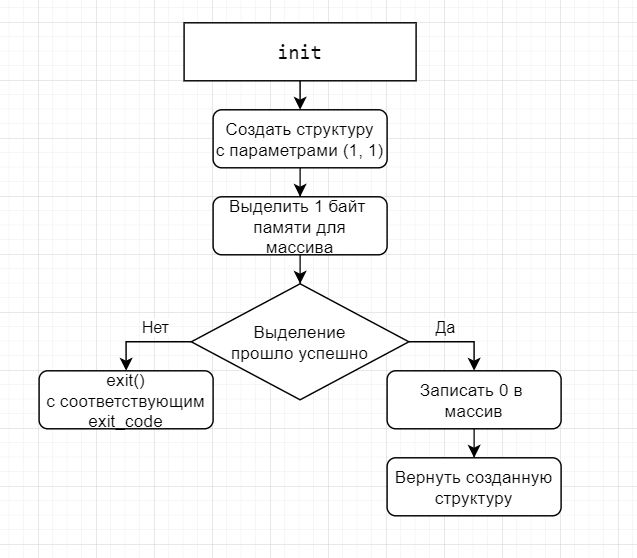


Рисунок 2 – Блок-схема функции «init»

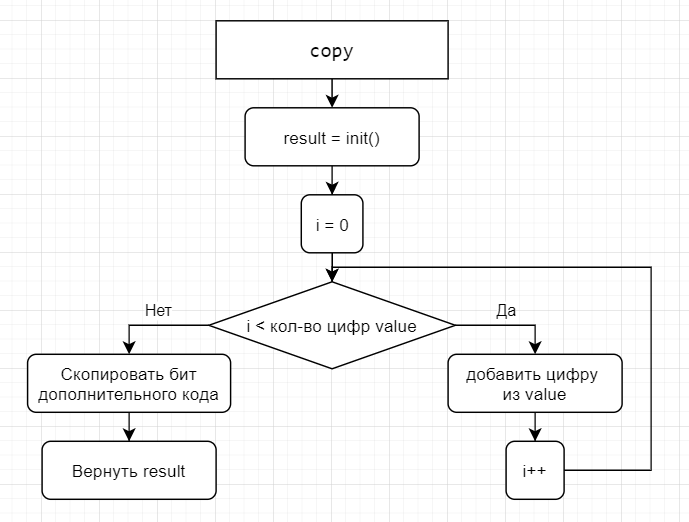


Рисунок 3 – Блок-схема функции «copy»

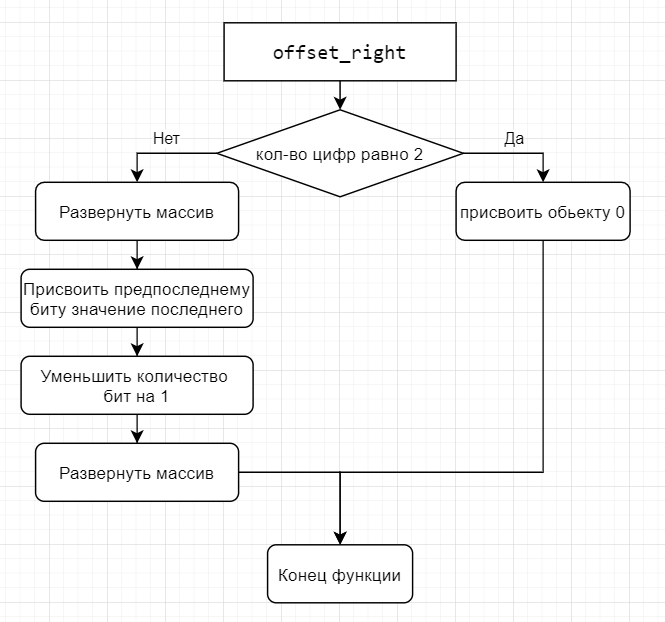


Рисунок 4 – Блок-схема функции «offset\_right»

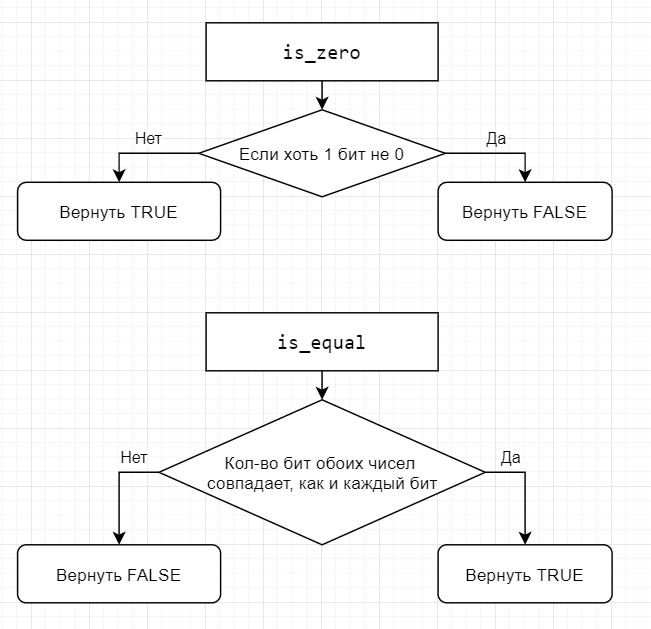


Рисунок 5 – Блок-схема функций «is\_zero» и «is\_equal»

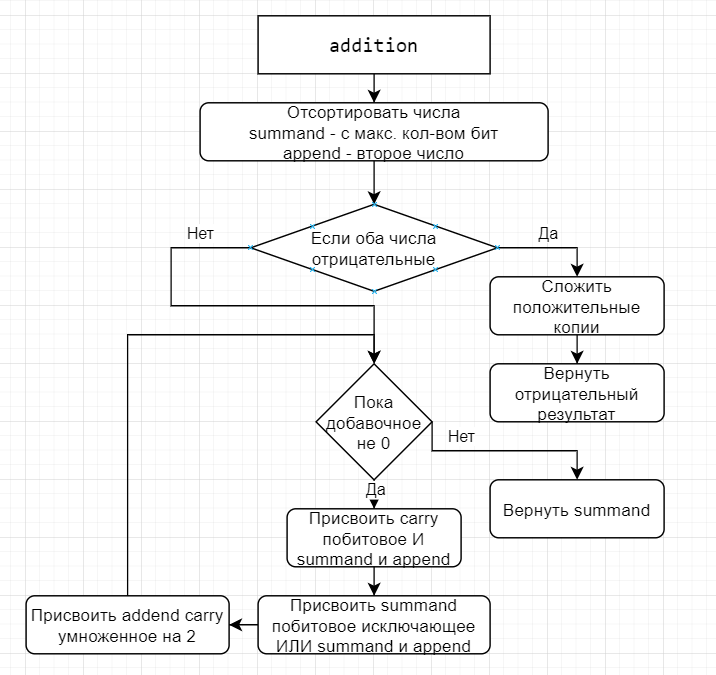


Рисунок 6 – Блок-схема функции «addition»

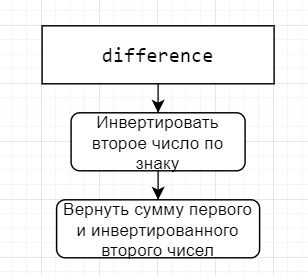


Рисунок 7 – Блок-схема функции «difference»

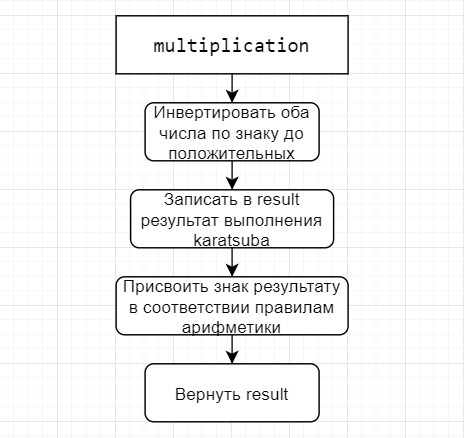


Рисунок 8 – Блок-схема функции «multiplication»

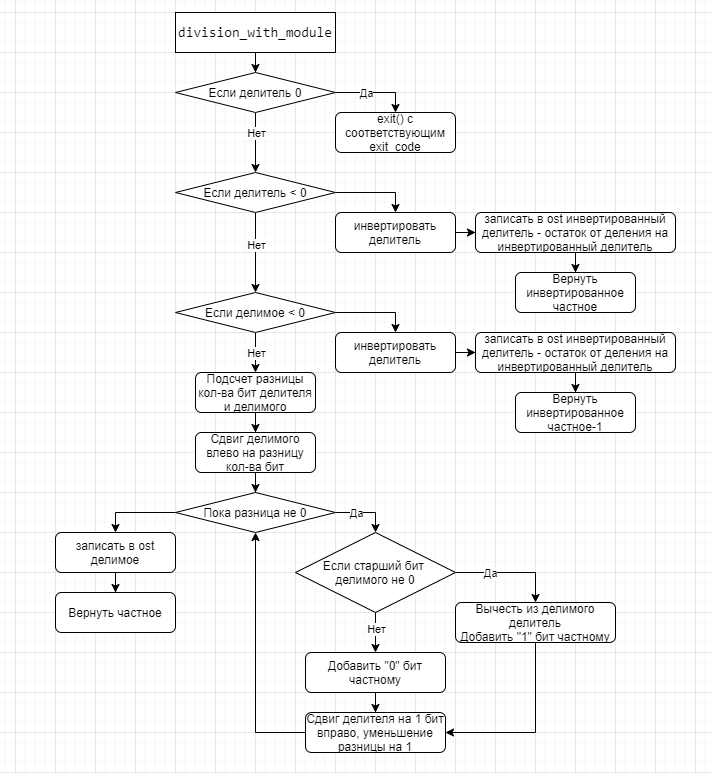


Рисунок 9 – Блок-схема функции «division\_with\_module»

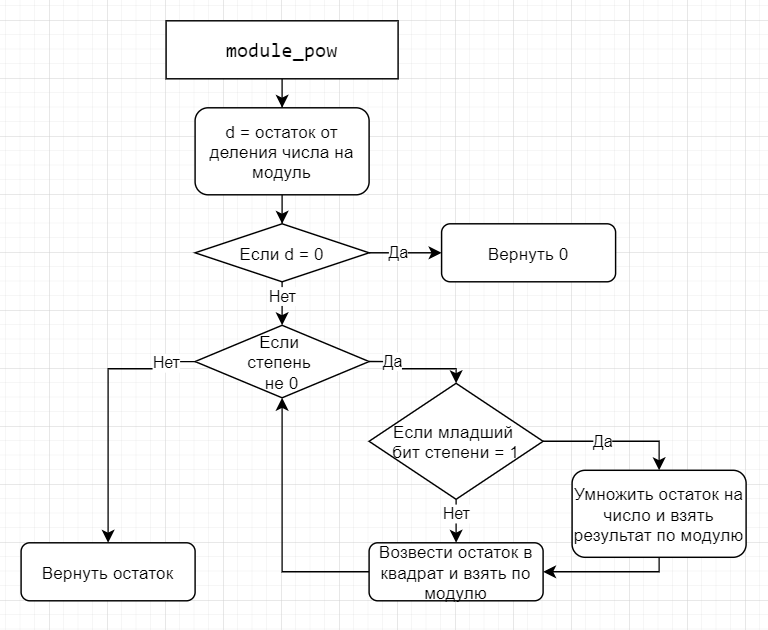


Рисунок 10 – Блок-схема функции «module\_pow»

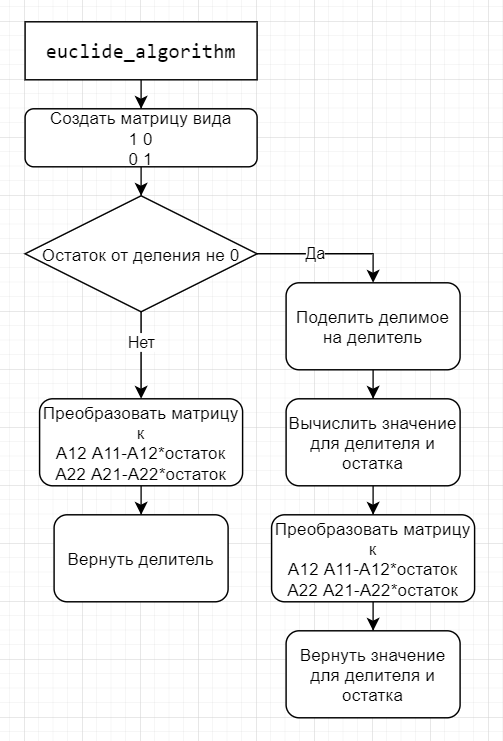


Рисунок 11 – Блок-схема функции «euclide\_algorithm\_modifyed»

Также в программе реализованы функции и алгоритмы RSA шифрования:

1. Функция для генерации открытого и закрытого ключей для RSA алгоритма и их запись в файл (рисунок 12);
2. Функция для проверки подписи файла (рисунок 13);
3. Функция для создания подписи файла (рисунок 14);
4. Функция для шифрования файла (рисунок 15);
5. Функция для дешифрования файла (рисунок 16).

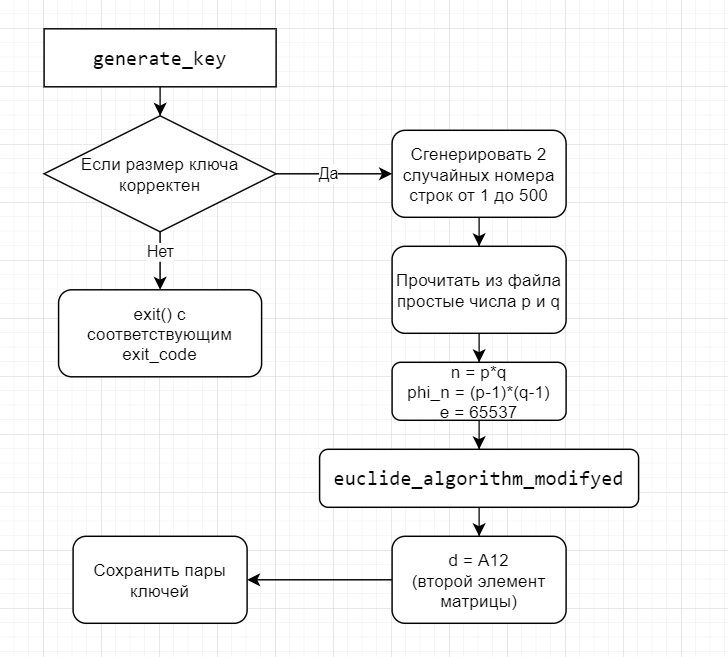


Рисунок 12 – Блок-схема функции «generate\_key»

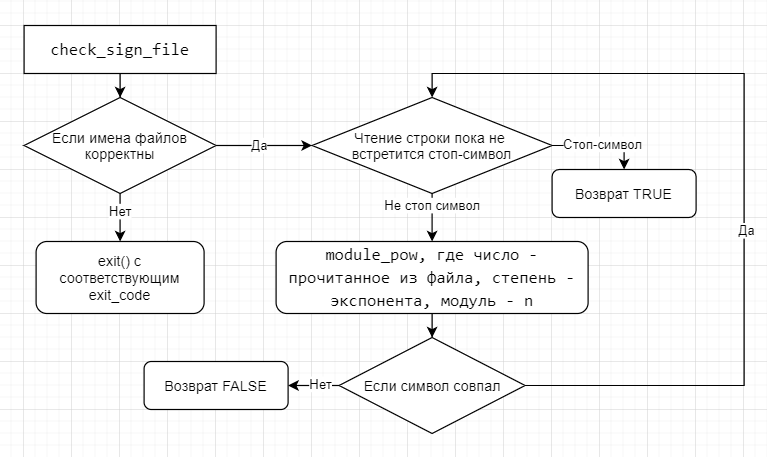


Рисунок 13 – Блок-схема функции «check\_sign\_file»

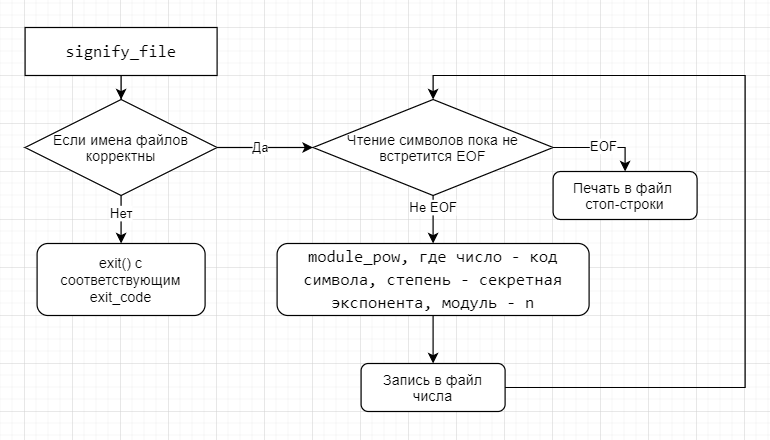


Рисунок 14 – Блок-схема функции «signify\_file»

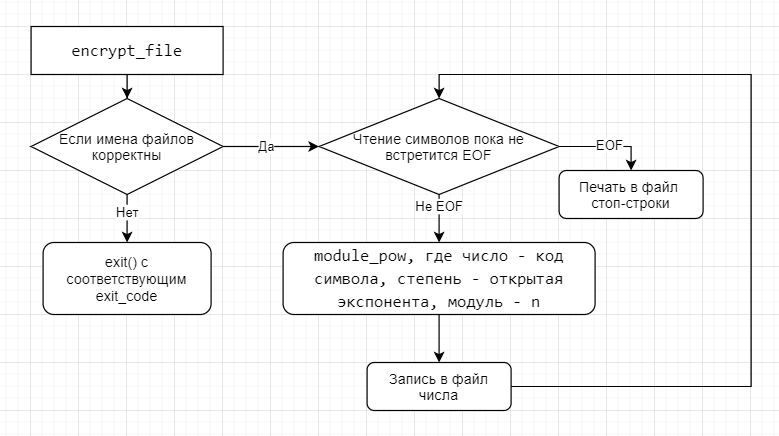


Рисунок 15 – Блок-схема функции «encrypt\_file»

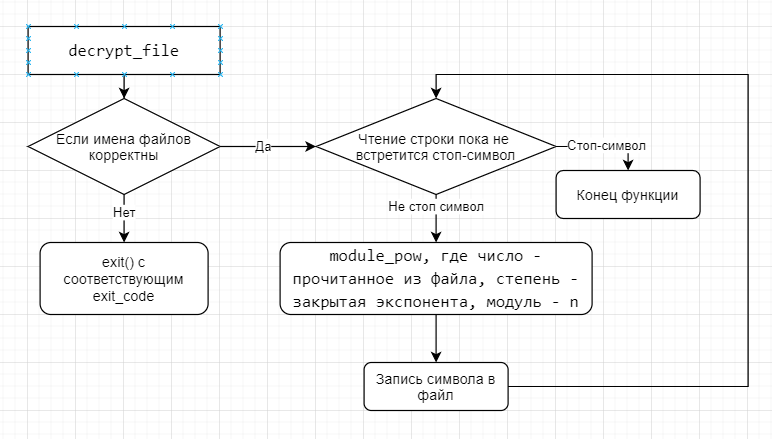


Рисунок 16 – Блок-схема функции «decrypt\_file»

Для ускорения производимых расчетов был использован алгоритм умножения по методу Карацубы с асимптотической сложностью O(n^1.58), алгоритм быстрого возведения в степень с асимптотической сложностью O(log2(n)), алгоритм умножения числа по модулю.

Блок-схема представлена на рисунке 17.

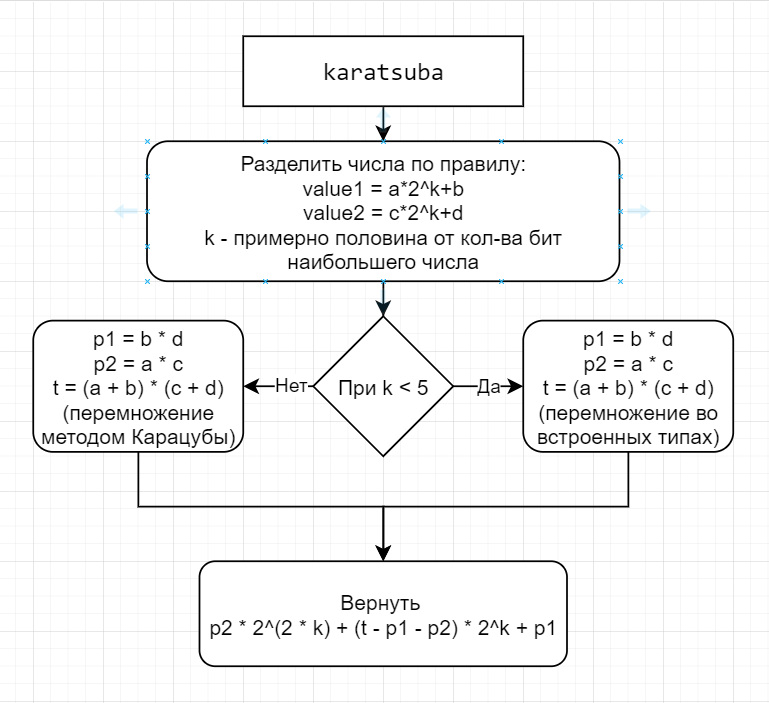


Рисунок 17 – Блок-схема функции «karatsuba»

## Реализация ввода данных

Данная программа позволяет обрабатывать файлы «.txt» расширения, а также использует файлы «.blackleague» расширения для хранения заранее сгенерированных простых чисел с заданным количеством бит.

При вызове из консоли без параметров программа предложит пользователю воспользоваться функцией помощи «-h» или «--help» (Рисунок 18).

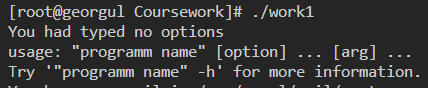


Рисунок 18 – Неправильный вызов программы

При вызове программы с флагом -h или --help пользователю будет выдана полная справка о функционале программы (Рисунок 19).

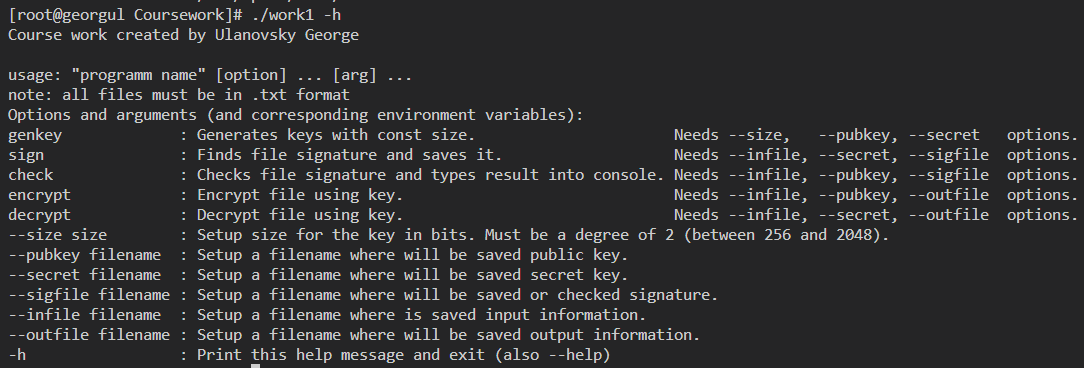


Рисунок 19 – Запуск программы с флагом “-h”

При вызове программы с неправильными флагами пользователю будет выдано соответствующее сообщение (Рисунок 20).

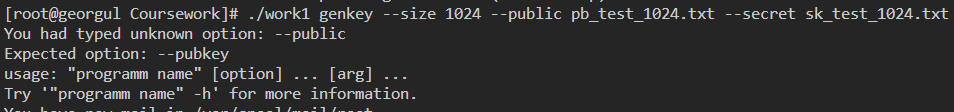


Рисунок 20 – Запуск программы с неправильным флагом

В программе реализована защита от сбоев:

1. Все сгенерированные файлы с «большими» числами имеют свой формат хранения данных, при изменении которого программа выдаст в log-файл сообщение об ошибке чтения и штатно завершит работу;
2. При вводе неправильного флага программа предупредит пользователя о неправильном флаге, предложит ожидаемый вариант и штатно завершит работу;
3. При вводе недостаточного или избыточного количества аргументов программа предупредит пользователя о неправильном вводе, предложит воспользоваться флагом «-h» и штатно завершит работу;
4. При вводе файлов с одинаковыми названиями программа выдаст пользователю соответствующее сообщение и штатно завершит работу;
5. При вводе файлов с неправильным расширением программа выдаст пользователю соответствующее сообщение и штатно завершит работу;
6. При сбое при попытке чтения или записи в файл программа выдаст в log-файл сообщение об ошибке чтения и штатно завершит работу;
7. При сбое при выделении динамической памяти программа выдаст в log-файл сообщение об ошибке чтения и штатно завершит работу;

## Логирование

Программа автоматически заносит все события в log-файл в папке «logs» в каталог, в котором находится программа. Все файлы в данной папке имеют название по правилу: «системное время в формате ЧЧ:ММ:СС» + «пробел» + «системная дата в формате ДД:ММ:ГГ». По такому же правилу начинаются все строки сообщения в log-файлах. Файлы имеют расширение “.log”. Логировщик заносит в файлы всю отладочную информацию, которая создается в процессе выполнения программы.

# Результат работы

В случае правильных входных данных и успешного завершения работы программа не выдаст никакого сообщения (Рисунок 21).

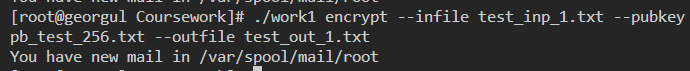


Рисунок 21 – Правильный вызов программы

По завершению работы программа создает или перезаписывает один или несколько выходных файлов в зависимости от выбранного режима:

1. Файлы с парами открытых и закрытых ключей при флаге «genkey»
2. Файл с расшифрованным сообщением при флаге «decrypt»
3. Файл с зашифрованным сообщением при флаге «encrypt»
4. Файл с подписью файла при флаге «sign»

В случае режима проверки подписи результат проверки будет выведен в консоль (Рисунок 22, Рисунок 23).

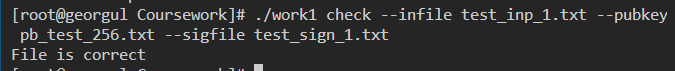


Рисунок 22 – Файл, его правильная подпись и ключ

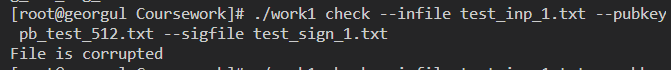


Рисунок 23 – Файл, его правильная подпись и неправильный ключ

# Тест на быстродействие

Для замеров быстродействия программы были сгенерированы тестовые ключи всех размеров, которые предоставляет программа (256, 512, 1024 и 2048 бит), а также с помощью log-файлов получено время работы по шифрованию, дешифрованию, созданию подписи и проверки подписи файлов.

Как следует из графика на Рисунке 24, скорость работы алгоритмов на одном и том же по размеру ключе линейно зависит от количества символов, поэтому для проверки были использованы файлы с одним символом.

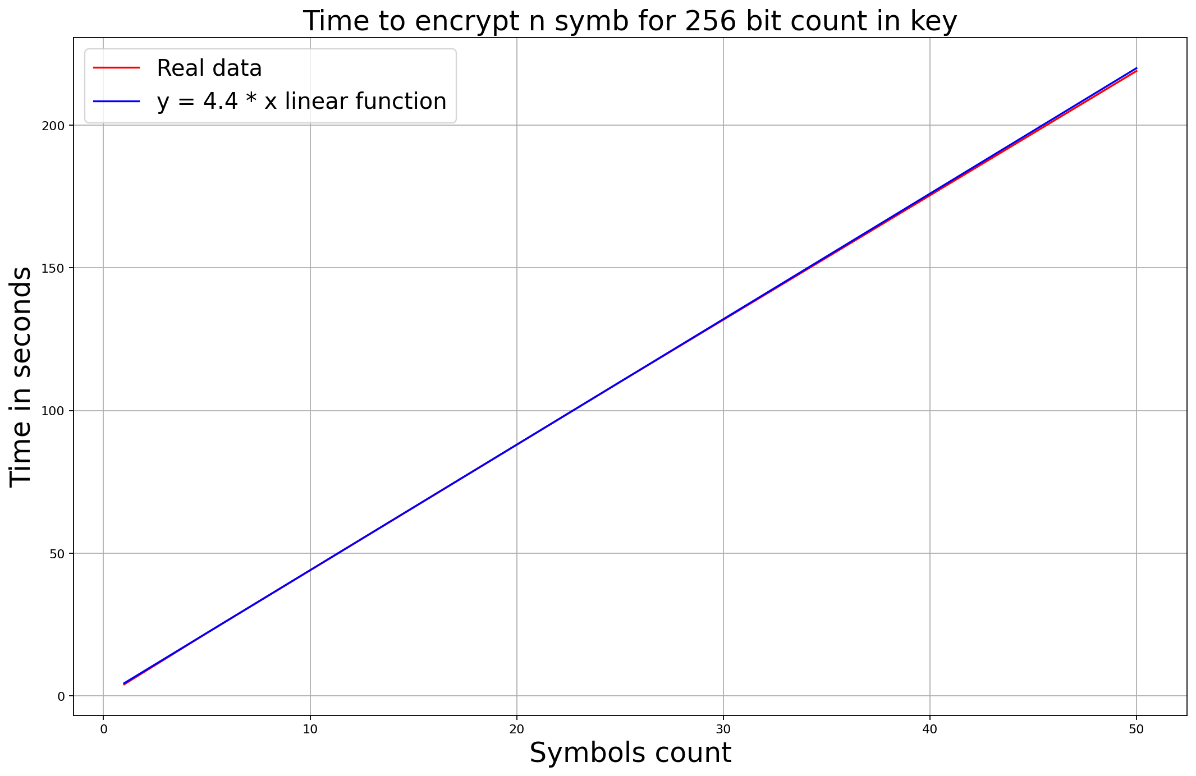


Рисунок 24 – Зависимость скорости работы от количества исходных данных в файле

График на Рисунке 25 отражает зависимость времени зашифровки 1 символа от длины ключа в битах.

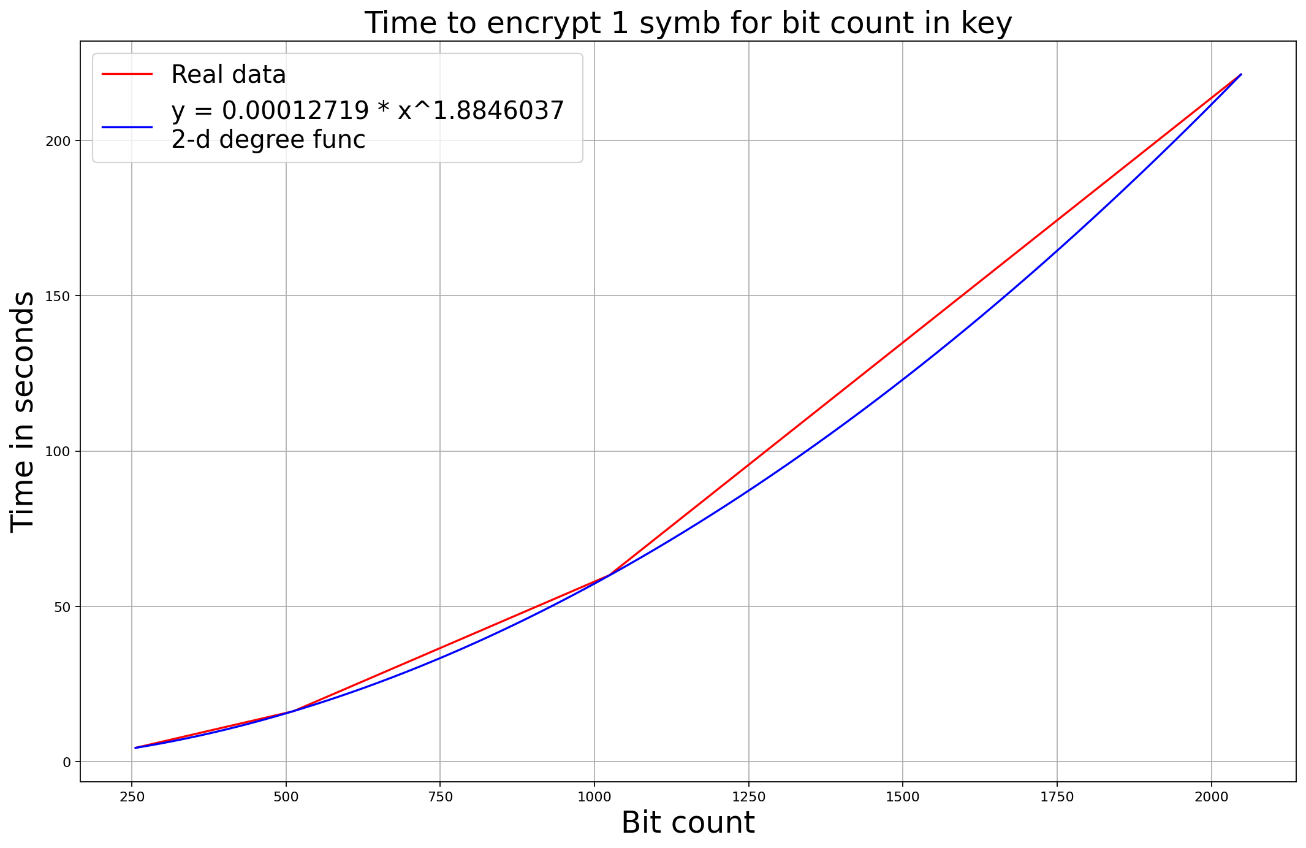


Рисунок 25 – Зависимость скорости работы от количества бит в ключе

График на Рисунке 26 отражает зависимость времени дешифровки 1 символа от длины ключа в битах.

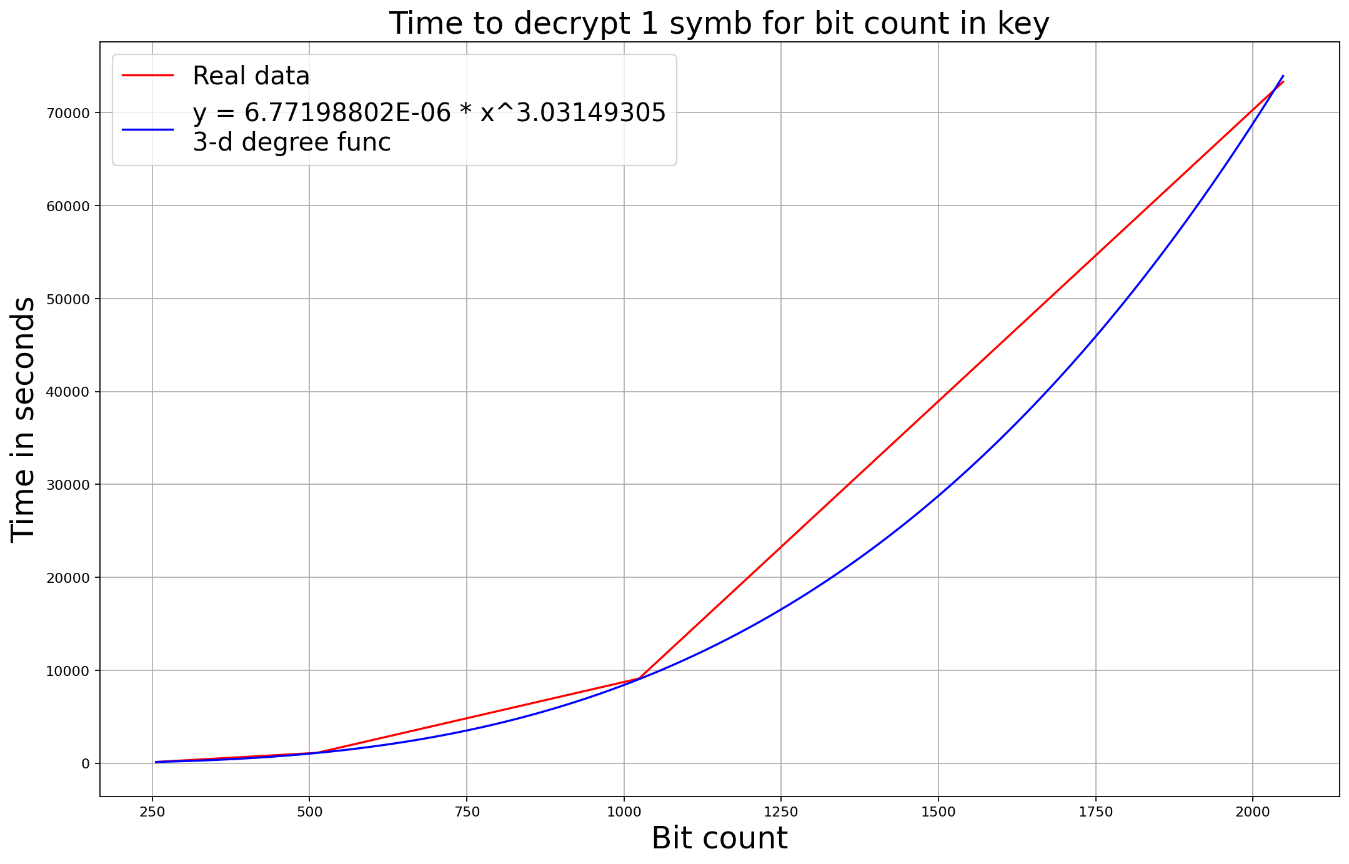


Рисунок 26 – Зависимость скорости работы от количества бит в ключе

Графики на Рисунках 27 и 28 отражают зависимость времени создания и проверки подписи для 1 символа от длины ключа в битах.

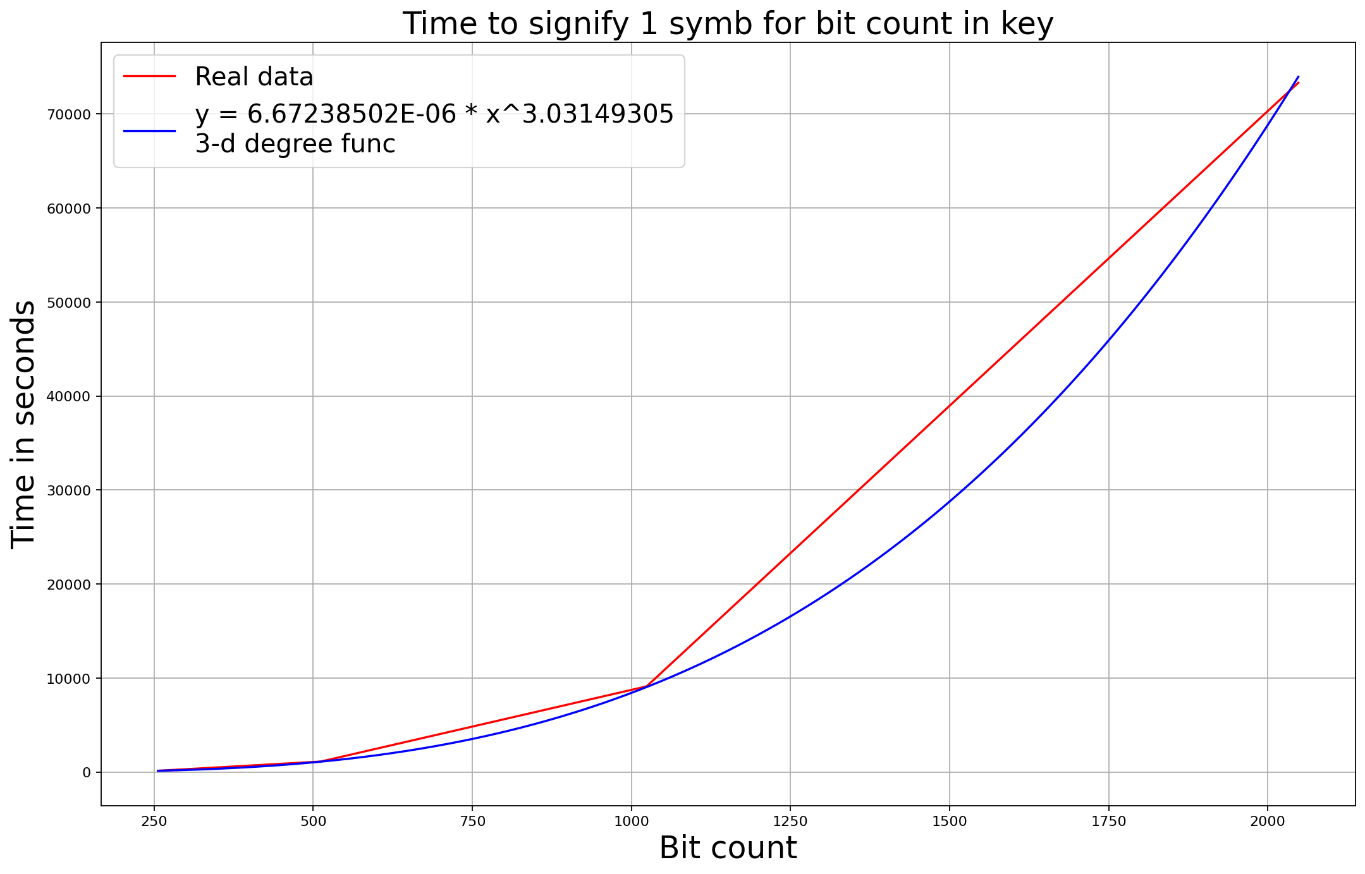


Рисунок 27 – Зависимость скорости работы от количества бит в ключе

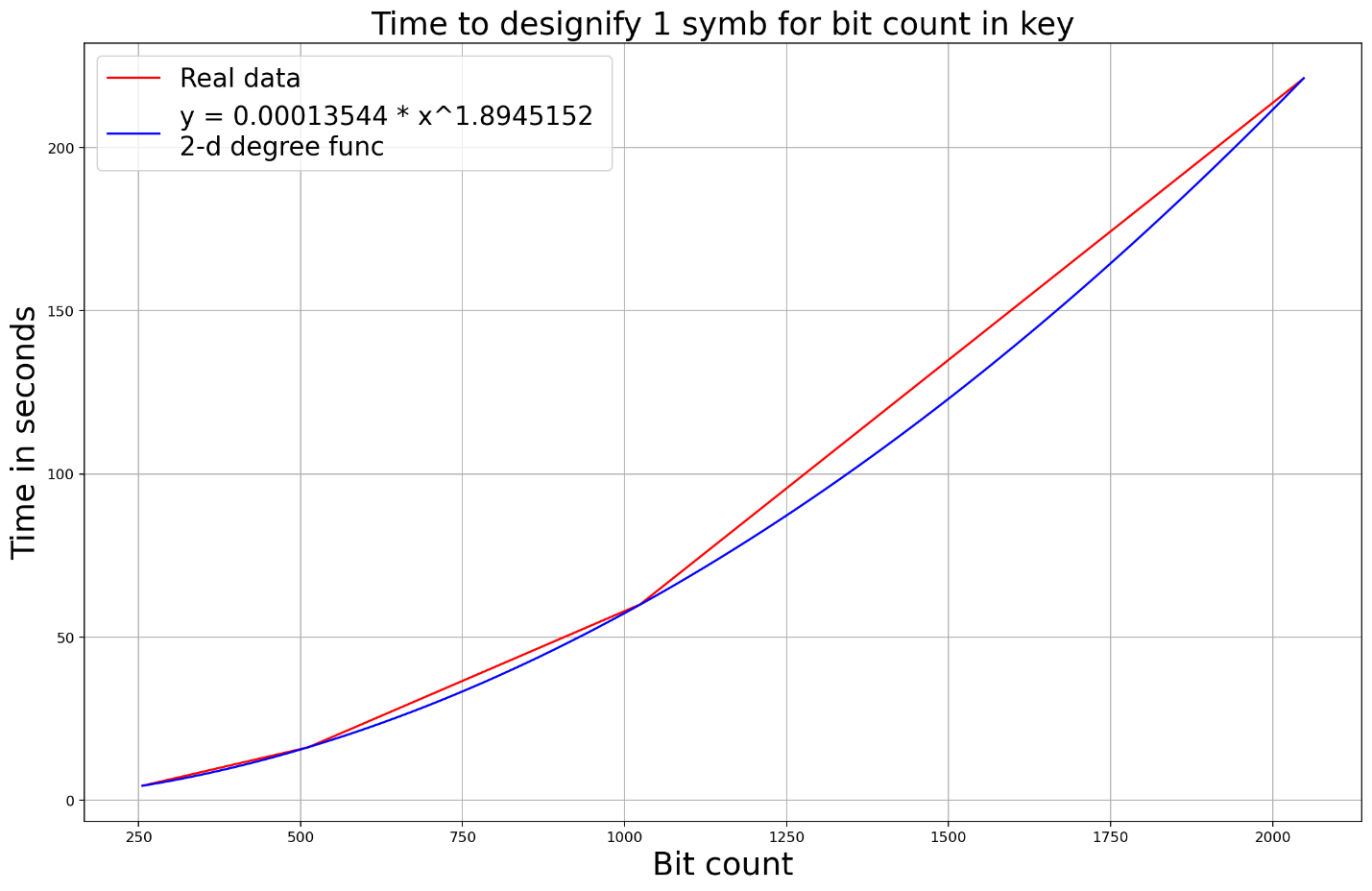


Рисунок 28 – Зависимость скорости работы от количества бит в ключе

Как видно из графиков, временная сложность операций создания подписи и дешифровки сообщения совпадают (порядка O(n^3)), как и временная сложность операций проверки подписи и шифрования сообщения (порядка O(n^1.88)). Разница сложностей связана со сложностью алгоритма быстрого возведения в степень, которая равна log2(n). Соотношение скоростей будет равно log2(bit\_size(d))/log2(bit\_size(e)). Так как открытая экспонента – константа, то при увеличении количества бит в d в n раз, затраченное время также увеличивается в n раз больше, чем при операциях шифрования и проверки подписи.

# Заключение

В ходе работы было создано консольное приложение, позволяющее шифровать, дешифровать, подписывать, проверять подпись для файлов и генерировать ключи для RSA алгоритма. Были оточены навыки работы со структурами данных, поиск нужной информации в глобальной паутине. Был проведен анализ различных алгоритмов и способов хранения данных по временной сложности и сложности их реализации.

Также были освоены основы правильного логирования событий и отточены навыки работы с вводимыми пользователем данными и системой контроля версий GitHub.

# Приложение А

Листинг файла «bit\_la.h»

#ifndef BIT\_LA\_H

#define BIT\_LA\_H

#include "\_log\_.h"

#define NUMBER\_SYSTEM\_BASE 2

#define MILLERS\_METHOD\_ITERATIONS\_NUMBER 20

/// <summary>

/// Структура, описывающая целочисленный тип с изменяемым размером памяти

/// </summary>

typedef struct

{

int current\_count; // текущее количество элементов

int size; // реальный размер массива

uint8\_t\* mas; // массив данных

}number;

/// <summary>

/// Побитовое исключающее �Л�

/// </summary>

/// <param name="a">Первый бит</param>

/// <param name="b">�'торой бит</param>

/// <returns>Результат операции</returns>

inline uint8\_t XOR(uint8\_t a, uint8\_t b);

/// <summary>

/// Побитовое НЕ

/// </summary>

/// <param name="a">�'ит числа</param>

/// <returns>Результат операции</returns>

inline uint8\_t NOT(uint8\_t a);

/// <summary>

/// Побитовое �

/// </summary>

/// <param name="a">Первый бит</param>

/// <param name="b">�'торой бит</param>

/// <returns>Результат операции</returns>

inline uint8\_t AND(uint8\_t a, uint8\_t b);

/// <summary>

/// Перевод числа в дополнительный код

/// </summary>

/// <param name="value"></param>

void additional\_code(number\* value);

/// <summary>

/// Перевод числа из дополнительного в прямой код

/// </summary>

/// <param name="value"></param>

void nonadditional\_code(number\* value);

/// <summary>

/// �нициализирует обьект структуры с пустым массивом и положительным negative

/// </summary>

/// <returns>Обьект структуры</returns>

number init();

/// <summary>

/// �'озвращает структуру идентичную данной

/// </summary>

/// <param name="value">Ссылка на структуру</param>

/// <returns>Обьект структуры, идентичный данному</returns>

number copy(number\* value);

/// <summary>

/// �'озвращает структуру, в которой записано данное число в диапазоне int

/// </summary>

/// <param name="value">Целое число</param>

/// <returns>Обьект структуры, содержащий данное число</returns>

number int\_to\_number(int value);

/// <summary>

/// �'озвращает int, в котором записано данное число

/// </summary>

/// <param name="value">Number число</param>

/// <returns>Целое число</returns>

int number\_to\_int(number\* value);

/// <summary>

/// �'озвращает number-обьект без незначащих нулей

/// </summary>

/// <param name="value">Ссылка на обьект</param>

void normalize(number\* value);

/// <summary>

/// Освобождение динамически выделенной памяти

/// </summary>

void clear\_mem(number\* value);

/// <summary>

/// �"обавляет char-элемент в конец массива структуры (новый разряд слева)

/// </summary>

/// <param name="object">Ссылка на структуру</param>

/// <param name="value">�-начение</param>

void add\_digit(number\* object, uint8\_t value);

/// <summary>

/// Удаление последней цифры (смещение на один разряд вправо)

/// </summary>

void offset\_right(number\* object);

/// <summary>

/// �"обавление нуля в конец (смещение на один разряд влево)

/// </summary>

void offset\_left(number\* object);

/// <summary>

/// Разворачивает число в данном обьекте

/// </summary>

/// <param name="value">Ссылка на число</param>

void reverse(number\* value);

/// <summary>

/// Печатает обьект с учетом знака так, как он хранится в памяти

/// </summary>

/// <param name="value">Ссылка на число</param>

void print\_number\_as\_is(number\* value);

/// <summary>

/// Печатает обьект с учетом знака в десятичной СС

/// </summary>

/// <param name="value">Ссылка на число</param>

void print\_number\_decimal(number\* value);

/// <summary>

/// Печатает обьект с учетом знака

/// </summary>

/// <param name="value">Ссылка на число</param>

void print\_number(number\* value);

/// <summary>

/// Печатает число в лог

/// </summary>

/// <param name="value">Ссылка на число</param>

void debug\_log(number\* value);

/// <summary>

/// Сравнение числа с 0 (Равно нулю/не равно нулю)

/// </summary>

/// <param name="object">Число, которое необходимо проверить</param>

/// <returns>Результат сравнения</returns>

BOOL is\_zero(number\* object);

/// <summary>

/// Сравнение числа с другим число (Равны/не равны)

/// </summary>

/// <param name="value1">Первое число, которое необходимо сравнить</param>

/// <param name="value2">�'торое число, которое необходимо сравнить</param>

/// <returns>Результат сравнения</returns>

BOOL is\_equal(number\* value1, number\* value2);

/// <summary>

/// Прибавляет одно число к другому, с учетом внутреннего знака данных на вход чисел, и возвращает результат типа number

/// </summary>

/// <param name="value1">Первое число</param>

/// <param name="value2">�'торое число</param>

/// <returns>Результат сложения</returns>

number addition(number\* value1, number\* value2);

/// <summary>

/// �'ычитает одно число из другого, с учетом внутреннего знака данных на вход чисел, и возвращает результат типа number

/// </summary>

/// <param name="value1">Первое число</param>

/// <param name="value2">�'торое число</param>

/// <returns>Результат вычитания</returns>

number difference(number\* value1, number\* value2);

/// <summary>

/// Умножает одно число на другое и возвращает результат типа number

/// </summary>

/// <param name="value1">Первое число</param>

/// <param name="value2">�'торое число</param>

/// <returns>Результат умножения</returns>

number multiplication(number\* value1, number\* value2);

/// <summary>

/// �"еление с остатком

/// </summary>

/// <param name="value1">�"елимое</param>

/// <param name="value2">�"елитель</param>

/// <param name="ost">Остаток</param>

/// <returns>Частное</returns>

number division\_with\_module(number\* value1, number\* value2, number\* ost);

/// <summary>

/// �'озвращает модуль от деление числа a в степени t на b

/// </summary>

/// <param name="a">�"елимое</param>

/// <param name="t">Степень</param>

/// <param name="b">�"елитель</param>

/// <returns>Остаток</returns>

number module\_pow(number\* a, number\* t, number\* b);

/// <summary>

/// �'озвращает НО�" двух чисел

/// </summary>

/// <param name="value1">Первое число</param>

/// <param name="value2">�'торое число</param>

/// <returns>НО�"</returns>

number euclide\_algorithm(number\* value1, number\* value2);

/// <summary>

/// �'озвращает НО�" двух чисел и матрицу множетелей

/// </summary>

/// <param name="value1">Первое число</param>

/// <param name="value2">�'торое число</param>

/// <param name="values">Массив с матрицей</param>

/// <returns>НО�"</returns>

number euclide\_algorithm\_modifyed(number\* value1, number\* value2, number\* values);

/// <summary>

/// �"енерация случайного числа с заданным количеством бит

/// </summary>

/// <param name="bit\_count">Кол-во бит</param>

/// <returns>Сгенерированное число</returns>

number generate\_random(int bit\_count);

/// <summary>

/// Проверка числа на простоту

/// </summary>

/// <param name="value">Число</param>

/// <returns>1 - если простое, 0 - если составное</returns>

BOOL Millers\_method(number\* value);

#endif // !BIT\_LA\_H

# Приложение Б

Листинг файла «bit\_la.c»

//

// Created by Georgul on 02.11.2021.

//

#include "bit\_LA.h"

inline uint8\_t XOR(uint8\_t a, uint8\_t b)

{

return a ^ b;

}

inline uint8\_t NOT(uint8\_t a)

{

return (a) ? 0 : 1;

}

inline uint8\_t AND(uint8\_t a, uint8\_t b)

{

return a & b;

}

void additional\_code(number\* value)

{

if (!is\_zero(value)) {

uint8\_t addit\_digit = 1;

int iter;

for (iter = 0; iter < value->current\_count; iter++)

{

value->mas[iter] = NOT(value->mas[iter]);

}

for (iter = 0; iter < value->current\_count; iter++)

{

value->mas[iter] = XOR(value->mas[iter], addit\_digit);

if (value->mas[iter])

addit\_digit = 0;

}

}

}

void nonadditional\_code(number\* value)

{

if (!is\_zero(value)) {

uint8\_t addit\_digit = 1;

int iter;

for (iter = 0; iter < value->current\_count; iter++)

{

value->mas[iter] = NOT(value->mas[iter]);

}

for (iter = 0; iter < value->current\_count; iter++)

{

value->mas[iter] = XOR(value->mas[iter], addit\_digit);

if (value->mas[iter])

addit\_digit = 0;

}

}

}

number init() {

number result = { 1, 1 };

result.mas = (uint8\_t\*)malloc(sizeof(uint8\_t));

if (result.mas == NULL)

{

\_log("Memory allocation failure in init() function");

exit(MEMORY\_ALLOCATION\_FAILURE);

}

result.mas[0] = 0;

return result;

}

number copy(number\* value) {

number result = init();

int i; // iterator

for (i = 0; i < value->current\_count - 1; i++)

add\_digit(&result, value->mas[i]);

result.mas[result.current\_count - 1] = value->mas[value->current\_count - 1];

return result;

}

number int\_to\_number(int value) {

int ostatok;

number result = init();

if (value < 0)

{

value \*= -1;

while (value > 0) {

ostatok = value % NUMBER\_SYSTEM\_BASE;

add\_digit(&result, ostatok);

value >>= 1;

}

if (result.current\_count == 0)

add\_digit(&result, 0);

additional\_code(&result);

}

else

{

while (value > 0) {

ostatok = value % NUMBER\_SYSTEM\_BASE;

add\_digit(&result, ostatok);

value >>= 1;

}

if (result.current\_count == 1)

{

add\_digit(&result, 0);

}

}

return result;

}

int number\_to\_int(number\* value)

{

int res = 0, i, zn = 1;

int x = 1; // ������� ������

if (value->mas[value->current\_count - 1])

{

zn = -1;

nonadditional\_code(value);

for (i = 0; i < value->current\_count - 1; i++)

{

res += value->mas[i] \* x;

x <<= 1;

}

}

else

{

for (i = 0; i < value->current\_count - 1; i++)

{

res += value->mas[i] \* x;

x <<= 1;

}

}

res \*= zn;

return res;

}

void normalize(number\* value) {

number result = init();

int i; //iterator

int end = value->current\_count - 2;

if (value->mas[value->current\_count - 1])

{

for (i = value->current\_count - 2; i > 0; i--) {

end = i;

if (value->mas[i] == 0 || value->mas[i - 1] == 0) {

break;

}

}

}

else

{

for (i = value->current\_count - 2; i >= 0; i--) {

end = i;

if (value->mas[i] != 0) {

break;

}

}

}

for (i = 0; i <= end; i++) {

add\_digit(&result, value->mas[i]);

}

result.mas[result.current\_count - 1] = value->mas[value->current\_count - 1];

value->current\_count = 1;

value->size = 1;

free(value->mas);

value->mas = (uint8\_t\*)malloc(sizeof(uint8\_t));

if (value->mas == NULL)

{

\_log("Memory allocation failure in normalize() function");

exit(MEMORY\_ALLOCATION\_FAILURE);

}

for (i = 0; i <= end; i++) {

add\_digit(value, result.mas[i]);

}

value->mas[value->current\_count - 1] = result.mas[result.current\_count - 1];

clear\_mem(&result);

}

void clear\_mem(number\* value)

{

free(value->mas);

value->mas = NULL;

}

void add\_digit(number\* object, uint8\_t value) {

int iter;

uint8\_t\* buff;

if (object->current\_count < object->size) {

object->mas[object->current\_count] = object->mas[object->current\_count - 1];

object->mas[object->current\_count - 1] = value;

object->current\_count += 1;

}

else {

buff = (uint8\_t\*)malloc(sizeof(uint8\_t) \* (object->current\_count));

if (buff == NULL)

{

\_log("Memory allocation failure in add\_digit() function (buffer)");

exit(MEMORY\_ALLOCATION\_FAILURE);

}

for (iter = 0; iter < object->current\_count; ++iter)

{

buff[iter] = object->mas[iter];

}

clear\_mem(object);

object->mas = (uint8\_t\*)malloc(sizeof(uint8\_t) \* (object->size) \* 2);

if (object->mas == NULL)

{

\_log("Memory allocation failure in add\_digit() function (object)");

exit(MEMORY\_ALLOCATION\_FAILURE);

}

for (iter = 0; iter < object->current\_count; ++iter)

{

object->mas[iter] = buff[iter];

}

free(buff);

object->mas[object->current\_count] = object->mas[object->current\_count - 1];

object->mas[object->current\_count - 1] = value;

object->current\_count += 1;

object->size \*= 2;

}

}

void offset\_right(number\* object)

{

if (object->current\_count == 2)

{

\*object = int\_to\_number(0);

}

else

{

reverse(object);

object->mas[object->current\_count - 2] = object->mas[object->current\_count - 1];

object->current\_count -= 1;

reverse(object);

normalize(object);

}

}

void offset\_left(number\* object)

{

reverse(object);

add\_digit(object, 0);

swap(object->mas[object->current\_count - 2], object->mas[object->current\_count - 1]);

reverse(object);

normalize(object);

}

void reverse(number\* value)

{

number prom = init();

int i; // iterator

for (i = value->current\_count - 2; i >= 0; i--)

{

add\_digit(&prom, value->mas[i]);

}

for (i = 0; i <= prom.current\_count - 2; i++)

{

value->mas[i] = prom.mas[i];

}

clear\_mem(&prom);

}

void print\_number\_as\_is(number\* value) {

int i; //iterator

for (i = 0; i <= value->current\_count - 2; i++)

{

printf("%d", (int)value->mas[i]);

if (i % 4 == 3)

printf(" ");

}

printf(" %d", (int)value->mas[value->current\_count - 1]);

printf("\n");

}

void print\_number\_decimal(number\* value) {

int y; //iterator

int x = 1; // stepen 2

int s = 0; // summa

if (value->mas[value->current\_count - 1])

{

printf("-");

nonadditional\_code(value);

for (y = 0; y < value->current\_count - 1; y++)

{

s += value->mas[y] \* x;

x <<= 1;

}

printf("%d\n", s);

additional\_code(value);

}

else

{

for (y = 0; y < value->current\_count - 1; y++)

{

s += value->mas[y] \* x;

x <<= 1;

}

printf("%d\n", s);

}

}

void print\_number(number\* value) {

int i; //iterator

printf("%d ", (int)value->mas[value->current\_count - 1]);

while (((value->current\_count) - 1) % 4 != 0)

{

add\_digit(value, 0);

}

for (i = value->current\_count - 2; i >= 0; i--)

{

printf("%d", (int)value->mas[i]);

if (i % 4 == 0)

printf(" ");

}

normalize(value);

printf("\n");

}

void debug\_log(number\* value)

{

int i, n = value->current\_count;

char buff[4000];

buff[0] = value->mas[n - 1]+'0';

buff[1] = ' ';

// |

//100101 cc=6

//1 01001

// |

for (i = n - 2; i >= 0; i--)

{

buff[n - i] = value->mas[i] + '0';

}

buff[n + 1] = '\0';

//printf("%s\n", buff);

\_log(buff);

}

BOOL is\_zero(number\* object)

{

int iterator = 0;

for (iterator = 0; iterator < object->current\_count; iterator++)

{

if (object->mas[iterator] != 0)

{

return FALSE;

}

}

return TRUE;

}

BOOL is\_equal(number\* value1, number\* value2)

{

short iterator = 0;

normalize(value1);

normalize(value2);

if (value1->current\_count != value2->current\_count)

return FALSE;

for (iterator = 0; iterator < value1->current\_count; iterator++)

{

if (value1->mas[iterator] != value2->mas[iterator])

return FALSE;

}

return TRUE;

}

number addition(number\* value1, number\* value2) {

number summand, addend;

number carry = init();

int real\_symb = 0;

int oper\_sign = 0;

int iter = 0;

int max\_symb = 0;

normalize(value1); normalize(value2);

if ((value1->current\_count) >= (value2->current\_count)) {

summand = copy(value1);

addend = copy(value2);

}

else {

summand = copy(value2);

addend = copy(value1);

}

if (summand.mas[summand.current\_count - 1] && addend.mas[addend.current\_count - 1])

{

clear\_mem(&carry);

nonadditional\_code(&summand);

nonadditional\_code(&addend);

carry = addition(&summand, &addend);

additional\_code(&carry);

clear\_mem(&addend);

clear\_mem(&summand);

normalize(&carry);

return carry;

}

oper\_sign = summand.mas[summand.current\_count - 1] + addend.mas[addend.current\_count - 1];

real\_symb = max(summand.current\_count, addend.current\_count);

while (!is\_zero(&addend))

{

max\_symb = max(summand.current\_count, addend.current\_count);

max\_symb = max(max\_symb, carry.current\_count);

if (max\_symb == 2) { max\_symb++; real\_symb++; }

for (iter = max\_symb - summand.current\_count; iter > 0; iter--)

{

if (summand.mas[summand.current\_count - 1])

add\_digit(&summand, 1);

else

add\_digit(&summand, 0);

}

for (iter = max\_symb - addend.current\_count; iter > 0; iter--)

{

if (addend.mas[addend.current\_count - 1])

add\_digit(&addend, 1);

else

add\_digit(&addend, 0);

}

for (iter = max\_symb - carry.current\_count; iter > 0; iter--)

{

if (carry.mas[carry.current\_count - 1])

add\_digit(&carry, 1);

else

add\_digit(&carry, 0);

}

//carry = (summand & addend);

for (iter = 0; iter < max\_symb; iter++)

{

carry.mas[iter] = AND(summand.mas[iter], addend.mas[iter]);

}

//summand = summand ^ addend;

for (iter = 0; iter < max\_symb; iter++)

{

summand.mas[iter] = XOR(summand.mas[iter], addend.mas[iter]);

}

//addend = (carry << 1);

clear\_mem(&addend);

addend = copy(&carry);

offset\_left(&addend);

if (oper\_sign == 1)

{

if (summand.current\_count > real\_symb)

{

summand.mas[real\_symb] = 0;

}

}

}

clear\_mem(&addend);

clear\_mem(&carry);

normalize(&summand);

return summand;

}

number difference(number\* value1, number\* value2) {

number b = copy(value2), buff;

if (value2->mas[value2->current\_count - 1])

nonadditional\_code(&b);

else

additional\_code(&b);

// \_b = -b

// a-b = a + (-b) = a + \_b

buff = addition(value1, &b);

clear\_mem(&b);

return buff;

}

number easy\_mult(number\* value1, number\* value2)

{

number result;

int a = number\_to\_int(value1);

int b = number\_to\_int(value2);

result = int\_to\_number(a\*b);

return result;

}

number karatsuba(number\* value1, number\* value2)

{

int n; int k; int iter;

number res = init(), buff1, buff2;

number a, b, c, d, p1, p2, t, v1, v2;

// k = n/2

// v1 = a \* (2^k) + b, v2 = c \* (2^k) + d

// p1 = b \* d

// p2 = a \* c

// t = (a+b)\*(c+d) - p1 - p2

// res = p2 \* 2^n + t \* 2^k + p1

n = max(value1->current\_count, value2->current\_count) - 1;

k = n / 2;

if (n <= 5)

{

clear\_mem(&res);

#ifdef DEBUG

printf("Start easyMult\n");

printf("n = %d\n", n);

printf("k = %d\n", k);

printf("v1 = ");

print\_number(value1);

printf("v2 = ");

print\_number(value2);

#endif // DEBUG

return easy\_mult(value1, value2);

}

else

{

v1 = copy(value1); v2 = copy(value2);

a = init(); b = init(); c = init(); d = init();

for (iter = n - v1.current\_count; iter > 0; iter--)

{

add\_digit(&v1, 0);

}

for (iter = n - v2.current\_count; iter > 0; iter--)

{

add\_digit(&v2, 0);

}

for (iter = 0; iter < k; iter++)

{

add\_digit(&b, v1.mas[iter]);

add\_digit(&d, v2.mas[iter]);

}

for (iter = k; iter < n; iter++)

{

add\_digit(&a, v1.mas[iter]);

add\_digit(&c, v2.mas[iter]);

}

#ifdef DEBUG

printf("Before all\n");

printf("n = %d\n", n);

printf("k = %d\n", k);

printf("v1 = ");

print\_number(&v1);

printf("v2 = ");

print\_number(&v2);

printf("a = ");

print\_number(&a);

printf("b = ");

print\_number(&b);

printf("c = ");

print\_number(&c);

printf("d = ");

print\_number(&d);

#endif // DEBUG

p1 = karatsuba(&b, &d);

p2 = karatsuba(&a, &c);

buff1 = addition(&a, &b);

buff2 = addition(&c, &d);

t = karatsuba(&buff1, &buff2);

clear\_mem(&buff1); clear\_mem(&buff2); clear\_mem(&a); clear\_mem(&b); clear\_mem(&c); clear\_mem(&d); clear\_mem(&v1); clear\_mem(&v2);

buff1 = copy(&t);

clear\_mem(&t);

t = difference(&buff1, &p1);

clear\_mem(&buff1);

buff1 = copy(&t);

clear\_mem(&t);

t = difference(&buff1, &p2);

clear\_mem(&buff1);

#ifdef DEBUG

printf("Berofe offset\np2 = ");

print\_number(&p2);

printf("t = ");

print\_number(&t);

printf("p1 = ");

print\_number(&p1);

#endif // DEBUG

for (iter = 0; iter < 2 \* k; iter++)

{

offset\_left(&p2);

}

for (iter = 0; iter < k; iter++)

{

offset\_left(&t);

}

#ifdef DEBUG

printf("After all\np2 = ");

print\_number(&p2);

printf("t = ");

print\_number(&t);

printf("p1 = ");

print\_number(&p1);

#endif // DEBUG

buff1 = copy(&res);

clear\_mem(&res);

res = addition(&buff1, &p2);

clear\_mem(&buff1);

buff1 = copy(&res);

clear\_mem(&res);

res = addition(&buff1, &p1);

clear\_mem(&buff1);

buff1 = copy(&res);

clear\_mem(&res);

res = addition(&buff1, &t);

clear\_mem(&buff1);

#ifdef DEBUG

printf("res = ");

print\_number(&res);

#endif // DEBUG

clear\_mem(&p1);

clear\_mem(&p2);

clear\_mem(&t);

////////////////////////////////////////////////////////////////////////////////////////////////

return res;

}

}

number multiplication(number\* value1, number\* value2) {

number result, a, b;

a = copy(value1); b = copy(value2);

int sign = a.mas[a.current\_count - 1] + b.mas[b.current\_count - 1];

if (a.mas[a.current\_count - 1])

{

additional\_code(&a);

}

if (b.mas[b.current\_count - 1])

{

additional\_code(&b);

}

result = karatsuba(&a, &b);

clear\_mem(&a); clear\_mem(&b);

if (sign == 1) {

additional\_code(&result);

}

normalize(&result);

return result;

}

number division\_with\_module(number\* value1, number\* value2, number\* ost)

{

number mod = int\_to\_number(0), rem = copy(value1), sub = copy(value2);

number add = int\_to\_number(1);

number buff;

short shifts = 1;

if (is\_zero(value2))

{

\_log("Divider was 0 in division\_with\_module() function");

exit(FAILURE);

}

if (value2->mas[value2->current\_count - 1])

{

clear\_mem(&mod);

clear\_mem(&rem);

buff = copy(value2);

nonadditional\_code(&buff);

mod = division\_with\_module(value1, &buff, &rem);

clear\_mem(&buff);

\*ost = rem;

additional\_code(&mod);

clear\_mem(&add);

clear\_mem(&sub);

return mod;

}

if (value1->mas[value1->current\_count - 1])

{

clear\_mem(&mod);

clear\_mem(&rem);

// b - ( |a| % b )

buff = copy(value1);

nonadditional\_code(&buff);

mod = division\_with\_module(&buff, value2, &rem); // ost = ( |a| % b )

clear\_mem(&buff);

if (is\_zero(&rem))

{

buff = int\_to\_number(0);

clear\_mem(&rem);

rem = copy(&buff);

clear\_mem(&buff);

}

else {

buff = difference(value2, &rem);

clear\_mem(&rem);

rem = copy(&buff);

clear\_mem(&buff);

buff = addition(&mod, &add);

clear\_mem(&mod);

mod = copy(&buff);

clear\_mem(&buff);

}

\*ost = rem;

additional\_code(&mod);

clear\_mem(&add);

clear\_mem(&sub);

return mod;

}

buff = difference(&sub, &rem);

while (buff.mas[buff.current\_count - 1])

{

offset\_left(&sub);

offset\_left(&add);

clear\_mem(&buff);

buff = difference(&sub, &rem);

shifts++;

}

clear\_mem(&buff);

while (shifts)

{

buff = difference(&rem, &sub);

while (!buff.mas[buff.current\_count - 1])

{

clear\_mem(&buff);

buff = difference(&rem, &sub);

clear\_mem(&rem);

rem = copy(&buff);

clear\_mem(&buff);

buff = addition(&add, &mod);

clear\_mem(&mod);

mod = copy(&buff);

clear\_mem(&buff);

buff = difference(&rem, &sub);

}

clear\_mem(&buff);

offset\_right(&sub);

offset\_right(&add);

shifts--;

}

clear\_mem(&sub);

clear\_mem(&add);

normalize(&rem);

normalize(&mod);

\*ost = rem;

return mod;

}

number module\_pow(number\* a, number\* t, number\* b)

{

number d, ost, iterator = init(), buff, buff2, buff3;

add\_digit(&iterator, 1);

buff3 = division\_with\_module(a, b, &d);

clear\_mem(&buff3);

ost = copy(&d);

if (is\_zero(&d))

{

clear\_mem(&d);

clear\_mem(&iterator);

return ost;

}

else

{

clear\_mem(&iterator);

iterator = copy(t);

clear\_mem(&ost);

ost = int\_to\_number(1);

buff2 = copy(a);

while (!is\_zero(&iterator))

{

if (iterator.mas[0] % 2 == 1)

{

buff = multiplication(&ost, &buff2);

clear\_mem(&ost);

buff3 = division\_with\_module(&buff, b, &ost);

clear\_mem(&buff3);

clear\_mem(&buff);

}

buff = multiplication(&buff2, &buff2);

clear\_mem(&buff2);

buff3 = division\_with\_module(&buff, b, &buff2);

clear\_mem(&buff3);

clear\_mem(&buff);

offset\_right(&iterator);

}

buff = copy(&ost);

clear\_mem(&ost);

buff3 = division\_with\_module(&buff, b, &ost);

clear\_mem(&buff);

clear\_mem(&buff2);

clear\_mem(&buff3);

}

clear\_mem(&d);

clear\_mem(&iterator);

normalize(&ost);

return ost;

}

number euclide\_algorithm(number\* value1, number\* value2)

{

number buff, a, b, mod;

a = copy(value1);

if (a.mas[a.current\_count - 1])

{

nonadditional\_code(&a);

}

b = copy(value2);

if (b.mas[b.current\_count - 1])

{

nonadditional\_code(&b);

}

buff = difference(&a, &b);

if (buff.mas[buff.current\_count - 1])

{

clear\_mem(&buff);

clear\_mem(&a);

clear\_mem(&b);

a = copy(value2);

b = copy(value1);

}

else

{

clear\_mem(&buff);

clear\_mem(&a);

clear\_mem(&b);

a = copy(value1);

b = copy(value2);

}

if (a.mas[a.current\_count - 1])

{

nonadditional\_code(&a);

}

if (b.mas[b.current\_count - 1])

{

nonadditional\_code(&b);

}

//a = b \* q\_0 + r\_1

buff = division\_with\_module(&a, &b, &mod);

clear\_mem(&buff);

if (!is\_zero(&mod))

{

buff = euclide\_algorithm(&b, &mod);

clear\_mem(&mod);

clear\_mem(&a);

clear\_mem(&b);

normalize(&buff);

return buff;

}

else

{

clear\_mem(&mod);

clear\_mem(&a);

normalize(&b);

return b;

}

}

number euclide\_algorithm\_modifyed(number\* value1, number\* value2, number\* values)

{

number buff, a, b, mod, div, GCD;

number \_a, \_b, \_c, \_d;

a = copy(value1);

b = copy(value2);

buff = difference(&a, &b);

if (buff.mas[buff.current\_count - 1])

{

clear\_mem(&buff);

clear\_mem(&a);

clear\_mem(&b);

a = copy(value2);

b = copy(value1);

}

else

{

clear\_mem(&buff);

clear\_mem(&a);

clear\_mem(&b);

a = copy(value1);

b = copy(value2);

}

//a = b \* q\_0 + r\_1

div = division\_with\_module(&a, &b, &mod);

if (!is\_zero(&mod))

{

buff = euclide\_algorithm\_modifyed(&b, &mod, values);

GCD = copy(&buff);

clear\_mem(&buff);

\_a = copy(&values[0]);

\_b = copy(&values[1]);

\_c = copy(&values[2]);

\_d = copy(&values[3]);

clear\_mem(&values[0]);

values[0] = copy(&\_b);

buff = multiplication(&\_b, &div);

clear\_mem(&values[1]);

values[1] = difference(&\_a, &buff);

clear\_mem(&buff);

clear\_mem(&values[2]);

values[2] = copy(&\_d);

buff = multiplication(&\_d, &div);

clear\_mem(&values[3]);

values[3] = difference(&\_c, &buff);

clear\_mem(&buff);

clear\_mem(&\_a);

clear\_mem(&\_b);

clear\_mem(&\_c);

clear\_mem(&\_d);

clear\_mem(&div);

clear\_mem(&mod);

clear\_mem(&a);

clear\_mem(&b);

normalize(&GCD);

return GCD;

}

else

{

GCD = copy(&b);

\_a = copy(&values[0]);

\_b = copy(&values[1]);

\_c = copy(&values[2]);

\_d = copy(&values[3]);

clear\_mem(&values[0]);

values[0] = copy(&\_b);

buff = multiplication(&\_b, &div);

clear\_mem(&values[1]);

values[1] = difference(&\_a, &buff);

clear\_mem(&buff);

clear\_mem(&values[2]);

values[2] = copy(&\_d);

buff = multiplication(&\_d, &div);

clear\_mem(&values[3]);

values[3] = difference(&\_c, &buff);

clear\_mem(&buff);

clear\_mem(&\_a);

clear\_mem(&\_b);

clear\_mem(&\_c);

clear\_mem(&\_d);

clear\_mem(&div);

clear\_mem(&mod);

clear\_mem(&a);

clear\_mem(&b);

normalize(&GCD);

return GCD;

}

}

number generate\_random(int bit\_count)

{

int i;

number res = init();

for (i = 0; i < bit\_count - 1; i++)

{

add\_digit(&res, rand() % 2);

}

return res;

}

BOOL Millers\_method(number\* value)

{

number t, buff, buff2, a, N\_minus\_1;

BOOL fl1, fl2;

number step = int\_to\_number(1);

number step2 = int\_to\_number(2);

number N\_3 = int\_to\_number(3);

int n = MILLERS\_METHOD\_ITERATIONS\_NUMBER;

int s = 0;

int i, k, j; // iterators

if (is\_equal(value, &step2))

{

clear\_mem(&step);

clear\_mem(&step2);

clear\_mem(&N\_3);

return TRUE;

}

if (is\_equal(value, &N\_3))

{

clear\_mem(&step);

clear\_mem(&step2);

clear\_mem(&N\_3);

return TRUE;

}

if (!value->mas[0])

{

clear\_mem(&step);

clear\_mem(&step2);

clear\_mem(&N\_3);

return FALSE;

}

//N-1 = 2^s \* t, t - �������

N\_minus\_1 = difference(value, &step);

t = copy(&N\_minus\_1);

while (t.mas[0] == 0)

{

s++;

offset\_right(&t);

}

srand((unsigned int)time(NULL));

for (i = 0; i < n; i++)

{

printf("i is %d\n", i);

fl1 = TRUE;

fl2 = TRUE;

a = generate\_random(value->current\_count - 1);

buff = difference(&a, &step2);

while (buff.mas[buff.current\_count - 1])

{

clear\_mem(&buff);

clear\_mem(&a);

a = generate\_random(value->current\_count - 1);

buff = difference(&a, &step2);

}

clear\_mem(&buff);

buff = euclide\_algorithm(value, &a);

if (!is\_equal(&buff, &step)) //�������� 1

{

fl1 = FALSE;

}

clear\_mem(&buff);

buff = module\_pow(&a, &t, value);

if (!is\_equal(&buff, &step)) // �������� 2

{

fl2 = FALSE;

for (k = 1; k <= s; k++)

{

clear\_mem(&buff);

buff2 = copy(&t);

for (j = 0; j < k - 1; j++)

{

offset\_left(&buff2);

}

buff = module\_pow(&a, &buff2, value);

clear\_mem(&buff2);

if (is\_equal(&buff, &N\_minus\_1))

{

//printf("= is %d\n", is\_equal(&buff, &N\_minus\_1));

fl2 = TRUE;

}

}

}

clear\_mem(&buff);

if (!fl1 || !fl2)

{

clear\_mem(&step);

clear\_mem(&step2);

clear\_mem(&N\_3);

clear\_mem(&t);

clear\_mem(&a);

clear\_mem(&N\_minus\_1);

return FALSE;

}

clear\_mem(&a);

}

clear\_mem(&step);

clear\_mem(&step2);

clear\_mem(&N\_3);

clear\_mem(&t);

clear\_mem(&N\_minus\_1);

return TRUE;

}

# Приложение В

Листинг файла «RSA.h»

#ifndef RSA\_H

#define RSA\_H

#include "config.h"

#include "\_file\_.h"

/// <summary>

/// Function, checking sign of file

/// </summary>

/// <param name="input\_filename">char array contains filename</param>

/// <param name="pubkey\_filename">char array contains filename</param>

/// <param name="sign\_filename">char array contains filename</param>

/// <returns>1 if sign is correct, else 0 </returns>

BOOL check\_sign\_file(char\* input\_filename, char\* pubkey\_filename, char\* sign\_filename);

/// <summary>

/// Function, creating a sign file to a input file

/// </summary>

/// <param name="input\_filename">char array contains filename</param>

/// <param name="seckey\_filename">char array contains filename</param>

/// <param name="sign\_filename">char array contains filename</param>

void signify\_file(char\* input\_filename, char\* seckey\_filename, char\* sign\_filename);

/// <summary>

/// Function, creating decrypted file of input file with key

/// </summary>

/// <param name="input\_filename">char array contains filename</param>

/// <param name="seckey\_filename">char array contains filename</param>

/// <param name="output\_filename">char array contains filename</param>

void decrypt\_file(char\* input\_filename, char\* seckey\_filename, char\* output\_filename);

/// <summary>

/// Function, creating encrypted file of input file with key

/// </summary>

/// <param name="input\_filename">char array contains filename</param>

/// <param name="pubkey\_filename">char array contains filename</param>

/// <param name="output\_filename">char array contains filename</param>

void encrypt\_file(char\* input\_filename, char\* pubkey\_filename, char\* output\_filename);

/// <summary>

/// Function, generating a pair of keys to en/decrypt

/// </summary>

/// <param name="key\_size\_str">char array contains bit size of keys</param>

/// <param name="pubkey\_filename">char array contains filename</param>

/// <param name="seckey\_filename">char array contains filename</param>

void generate\_key(char\* key\_size\_str, char\* pubkey\_filename, char\* seckey\_filename);

#endif // !RSA\_H

# Приложение Г

Листинг файла «RSA.c»

#include "RSA.h"

BOOL check\_sign\_file(char\* input\_filename, char\* pubkey\_filename, char\* sign\_filename)

{

number n, e, msg, msg\_ci;

int c = 0;

int c2 = 0;

FILE\* input, \* sign;

read\_key(pubkey\_filename, &n, &e, 'e');

msg\_ci = init();

input = check\_file\_exist\_read(input\_filename);

sign = check\_file\_exist\_read(sign\_filename);

while (read\_num\_from\_file(sign, &msg\_ci))

{

msg = module\_pow(&msg\_ci, &e, &n);

c = number\_to\_int(&msg) - 100;

c2 = fgetc(input);

if (c2 == EOF || c2 != c)

{

clear\_mem(&msg);

clear\_mem(&msg\_ci);

clear\_mem(&e);

clear\_mem(&n);

fclose(input);

fclose(sign);

return 0;

}

clear\_mem(&msg);

clear\_mem(&msg\_ci);

msg\_ci = init();

}

clear\_mem(&e);

clear\_mem(&n);

clear\_mem(&msg\_ci);

fclose(input);

fclose(sign);

return 1;

}

void signify\_file(char\* input\_filename, char\* seckey\_filename, char\* sign\_filename)

{

number n, d, msg, msg\_ci;

int c = 0;

FILE\* input, \* sign;

read\_key(seckey\_filename, &n, &d, 'd');

input = check\_file\_exist\_read(input\_filename);

sign = check\_file\_exist\_write(sign\_filename);

while ((c = fgetc(input)) != EOF)

{

msg = int\_to\_number(c + 100);

msg\_ci = module\_pow(&msg, &d, &n);

save\_num\_to\_file(sign, &msg\_ci);

clear\_mem(&msg);

clear\_mem(&msg\_ci);

}

fprintf(sign, "EOF");

clear\_mem(&d);

clear\_mem(&n);

fclose(input);

fclose(sign);

\_log("Signify complete");

}

void decrypt\_file(char\* input\_filename, char\* seckey\_filename, char\* output\_filename)

{

number n, d, msg, msg\_ci;

int c = 0;

FILE\* input, \* output;

read\_key(seckey\_filename, &n, &d, 'd');

msg\_ci = init();

input = check\_file\_exist\_read(input\_filename);

output = check\_file\_exist\_write(output\_filename);

while (read\_num\_from\_file(input, &msg\_ci))

{

msg = module\_pow(&msg\_ci, &d, &n);

c = number\_to\_int(&msg) - 100;

fprintf(output, "%c", (char)c);

clear\_mem(&msg);

clear\_mem(&msg\_ci);

msg\_ci = init();

}

// cleanup

clear\_mem(&d);

clear\_mem(&n);

clear\_mem(&msg\_ci);

fclose(input);

fclose(output);

}

void encrypt\_file(char\* input\_filename, char\* pubkey\_filename, char\* output\_filename)

{

number n, e, msg, msg\_ci;

int c = 0;

FILE \*input, \*output;

read\_key(pubkey\_filename, &n, &e, 'e');

input = check\_file\_exist\_read(input\_filename);

output = check\_file\_exist\_write(output\_filename);

while ((c = fgetc(input)) != EOF)

{

msg = int\_to\_number(c+100);

msg\_ci = module\_pow(&msg, &e, &n);

save\_num\_to\_file(output, &msg\_ci);

clear\_mem(&msg);

clear\_mem(&msg\_ci);

}

fprintf(output, "EOF");

clear\_mem(&e);

clear\_mem(&n);

fclose(input);

fclose(output);

\_log("Encrypt complete");

}

void generate\_key(char\* key\_size\_str, char\* pubkey\_filename, char\* seckey\_filename)

{

KEY\_BIT\_SIZE keysize;

int key\_s\_buf = 0, rand\_num\_p = -1, rand\_num\_q = -1;

number p, q, n, phi\_n, e, d, buff1, buff2, numb\_one, numb\_zero;

number\* arr;

key\_s\_buf = atoi(key\_size\_str);

if (key\_s\_buf == 0)

{

printf("Wrong key size: %s.\nType -h for help", key\_size\_str);

\_log("Error when try to make string \"key size\" to integer");

exit(FAILURE);

}

else

switch (key\_s\_buf)

{

case KEY\_256:

keysize = KEY\_256;

break;

case KEY\_512:

keysize = KEY\_512;

break;

case KEY\_1024:

keysize = KEY\_1024;

break;

case KEY\_2048:

keysize = KEY\_2048;

break;

default:

printf("Wrong key size: %s.\nType -h for help", key\_size\_str);

\_log("Error: key size doesnt belong to keySizes list");

exit(FAILURE);

break;

}

srand((unsigned int)time(NULL));

while (rand\_num\_p == rand\_num\_q) {

rand\_num\_p = rand() % 500 + 1;

rand\_num\_q = rand() % 500 + 1;

}

switch (keysize)

{

case KEY\_256:

p = get\_prime\_from\_file(primes\_128\_bit\_filename, rand\_num\_p, KEY\_256 / 2);

q = get\_prime\_from\_file(primes\_128\_bit\_filename, rand\_num\_q, KEY\_256 / 2);

break;

case KEY\_512:

p = get\_prime\_from\_file(primes\_256\_bit\_filename, rand\_num\_p, KEY\_512 / 2);

q = get\_prime\_from\_file(primes\_256\_bit\_filename, rand\_num\_q, KEY\_512 / 2);

break;

case KEY\_1024:

p = get\_prime\_from\_file(primes\_512\_bit\_filename, rand\_num\_p, KEY\_1024 / 2);

q = get\_prime\_from\_file(primes\_512\_bit\_filename, rand\_num\_q, KEY\_1024 / 2);

break;

case KEY\_2048:

p = get\_prime\_from\_file(primes\_1024\_bit\_filename, rand\_num\_p, KEY\_2048 / 2);

q = get\_prime\_from\_file(primes\_1024\_bit\_filename, rand\_num\_q, KEY\_2048 / 2);

break;

default:

break;

}

numb\_zero = int\_to\_number(0);

numb\_one = int\_to\_number(1);

n = multiplication(&p, &q);

buff1 = difference(&p, &numb\_one);

buff2 = difference(&q, &numb\_one);

phi\_n = multiplication(&buff1, &buff2);

clear\_mem(&buff1);

clear\_mem(&buff2);

e = int\_to\_number(65537);

\_log("genkey: const done");

arr = (number\*)malloc(4 \* sizeof(number)); // = { 1,0,0,1 };

arr[0] = copy(&numb\_one);

arr[1] = copy(&numb\_zero);

arr[2] = copy(&numb\_zero);

arr[3] = copy(&numb\_one);

\_log("genkey: arr created");

buff1 = euclide\_algorithm\_modifyed(&phi\_n, &e, arr);

clear\_mem(&buff1);

d = copy(&arr[1]);

\_log("genkey: all counts ready");

clear\_mem(&p);

clear\_mem(&q);

clear\_mem(&phi\_n);

clear\_mem(&numb\_one);

clear\_mem(&numb\_zero);

clear\_mem(&arr[0]);

clear\_mem(&arr[1]);

clear\_mem(&arr[2]);

clear\_mem(&arr[3]);

\_log("N");

debug\_log(&n);

\_log("E");

debug\_log(&e);

\_log("D");

debug\_log(&d);

save\_key(pubkey\_filename, &n, &e, 'e');

save\_key(seckey\_filename, &n, &d, 'd');

clear\_mem(&n);

clear\_mem(&e);

clear\_mem(&d);

exit(DEBUG\_EXIT\_CODE);

}

# Приложение Д

Листинг файла «\_log\_.h»

#ifndef \_LOG\_\_H

#define \_LOG\_\_H

#include "config.h"

#include "help.h"

void \_log\_start();

void \_log(char\* message);

#endif // !\_LOG\_\_H

# Приложение Е

Листинг файла «\_log\_.c»

#include "\_log\_.h"

char date[18];

char filename[29];

void \_log\_start()

{

FILE\* file;

const time\_t time\_programm\_started = time(NULL);

struct tm\* u = localtime(&time\_programm\_started);

#ifdef \_\_linux\_\_

mkdir("logs", S\_IRWXU);

#elif \_WIN32

mkdir("logs");

#endif

strftime(date, 18, "%H.%M.%S %d.%m.%y", u);

snprintf(filename, 29, "./logs/%s.log", date);

file = fopen(filename, "w");

if (file == NULL)

{

exit(LOG\_FAILURE);

}

else

{

fprintf(file, "%s: CREATED LOG\n", date);

}

fclose(file);

}

void \_log(char\* message)

{

FILE\* file;

const time\_t time\_programm\_started = time(NULL);

struct tm\* u = localtime(&time\_programm\_started);

strftime(date, 18, "%H.%M.%S %d.%m.%y", u);

file = fopen(filename, "a");

if (file == NULL)

{

exit(LOG\_FAILURE);

}

else

{

fprintf(file, "%s: %s\n", date, message);

}

fclose(file);

}

# Приложение Ё

Листинг файла «\_file\_.h»

#ifndef \_FILE\_\_H

#define \_FILE\_\_H

#include "config.h"

#include "bit\_LA.h"

#define PREFIX\_SIZE 3

#define PRIME\_PREFIX\_SIZE 5

/// <summary>

/// Проверяет имена файлов на правильность формата и совпадение

/// </summary>

/// <param name="fn1">�мя первого файла</param>

/// <param name="fn2">�мя второго файла</param>

/// <returns>1, если все проверки пройдены. 0, если хотя бы одна из проверок не пройдена</returns>

BOOL check\_filenames\_2(char\* fn1, char\* fn2);

/// <summary>

/// Проверяет имена файлов на правильность формата и совпадение

/// </summary>

/// <param name="fn1">�мя первого файла</param>

/// <param name="fn2">�мя второго файла</param>

/// <param name="fn3">�мя третьего файла</param>

/// <returns>1, если все проверки пройдены. 0, если хотя бы одна из проверок не пройдена</returns>

BOOL check\_filenames\_3(char\* fn1, char\* fn2, char\* fn3);

/// <summary>

/// Проверка на существование файла (запись)

/// </summary>

/// <param name="filename">�мя файла в виде строки</param>

/// <returns>Указатель на файл</returns>

FILE\* check\_file\_exist\_write(char\* filename);

/// <summary>

/// Проверка на существование файла (чтение)

/// </summary>

/// <param name="filename">�мя файла в виде строки</param>

/// <returns>Указатель на файл</returns>

FILE\* check\_file\_exist\_read(char\* filename);

/// <summary>

/// �-апись number в файл

/// </summary>

/// <param name="file">указатель на файл</param>

/// <param name="value">число</param>

void save\_num\_to\_file(FILE\* file, number\* value);

/// <summary>

/// Чтение number из файла

/// </summary>

/// <param name="file">указатель на файл</param>

/// <param name="value">число</param>

int read\_num\_from\_file(FILE\* file, number\* value);

/// <summary>

/// Сохраняет ключ

/// </summary>

/// <param name="filename">имя файла</param>

/// <param name="mod">N</param>

/// <param name="subkey">второй параметр (e или d)</param>

void save\_key(char\* filename, number\* mod, number\* subkey, char log);

/// <summary>

/// Читает ключ из файла

/// </summary>

/// <param name="filename">имя файла</param>

/// <param name="mod">N</param>

/// <param name="subkey">второй параметр (e или d)</param>

void read\_key(char\* filename, number\* mod, number\* subkey, char log);

/// <summary>

/// Function, returning a prime number from file.

/// </summary>

/// <param name="filename">char array contains filename</param>

/// <param name="line\_number">number of prime number. can be one from 1-500</param>

/// <param name="bit\_size">bit size of prime number</param>

/// <returns></returns>

number get\_prime\_from\_file(char\* filename, int line\_number, int bit\_size);

#endif // !\_FILE\_\_H

# Приложение Ж

Листинг файла «\_file\_.c»

#include "\_file\_.h"

BOOL check\_filenames\_2(char\* fn1, char\* fn2)

{

int strln;

int i; // iterator

char resol[5];

resol[4] = '\0';

if (!strcmp(fn1, fn2))

{

printf(ERROR\_FILE\_IDENT);

\_log("Files had same names");

exit(FILE\_FORMAT\_FAILURE);

}

if (strlen(fn1) < 6 || strlen(fn2) < 6)

{

printf(ERROR\_FILE\_EXTENSION);

\_log("1 or more files had less then 5 symbols in filename");

exit(FILE\_FORMAT\_FAILURE);

}

strln = strlen(fn1);

for (i = strln - 4; i < strln; i++)

{

resol[4 - (strln - i)] = fn1[i];

}

if (strcmp(".txt", resol))

{

printf(ERROR\_FILE\_EXTENSION);

\_log("File has wrong extension");

exit(FILE\_FORMAT\_FAILURE);

}

strln = strlen(fn2);

for (i = strln - 4; i < strln; i++)

{

resol[4 - (strln - i)] = fn2[i];

}

if (strcmp(".txt", resol))

{

printf(ERROR\_FILE\_EXTENSION);

\_log("File has wrong extension");

exit(FILE\_FORMAT\_FAILURE);

}

\_log("Filenames had good format");

return 1;

}

BOOL check\_filenames\_3(char\* fn1, char\* fn2, char\* fn3)

{

int strln;

int i; // iterator

char resol[5];

resol[4] = '\0';

if (!strcmp(fn1, fn2) || !strcmp(fn2, fn3) || !strcmp(fn1, fn3))

{

printf(ERROR\_FILE\_IDENT);

\_log("Files had same names");

exit(FILE\_FORMAT\_FAILURE);

}

if (strlen(fn1) < 6 || strlen(fn2) < 6 || strlen(fn3) < 6)

{

printf(ERROR\_FILE\_EXTENSION);

\_log("1 or more files had less then 5 symbols in filename");

exit(FILE\_FORMAT\_FAILURE);

}

strln = strlen(fn1);

for (i = strln - 4; i < strln; i++)

{

resol[4 - (strln - i)] = fn1[i];

}

if (strcmp(".txt", resol))

{

printf(ERROR\_FILE\_EXTENSION);

\_log("File has wrong extension");

exit(FILE\_FORMAT\_FAILURE);

}

strln = strlen(fn2);

for (i = strln - 4; i < strln; i++)

{

resol[4 - (strln - i)] = fn2[i];

}

if (strcmp(".txt", resol))

{

printf(ERROR\_FILE\_EXTENSION);

\_log("File has wrong extension");

exit(FILE\_FORMAT\_FAILURE);

}

strln = strlen(fn3);

for (i = strln - 4; i < strln; i++)

{

resol[4 - (strln - i)] = fn3[i];

}

if (strcmp(".txt", resol))

{

printf(ERROR\_FILE\_EXTENSION);

\_log("File has wrong extension");

exit(FILE\_FORMAT\_FAILURE);

}

\_log("Filenames had good format");

return 1;

}

FILE\* check\_file\_exist\_write(char\* filename)

{

FILE\* file;

file = fopen(filename, "w");

if (file == NULL)

{

\_log("Error while saving file");

exit(FILE\_OPEN\_FAILURE);

}

return file;

}

FILE\* check\_file\_exist\_read(char\* filename)

{

FILE\* file;

file = fopen(filename, "r");

if (file == NULL)

{

\_log("Error while reading file");

exit(FILE\_OPEN\_FAILURE);

}

return file;

}

void save\_num\_to\_file(FILE\* file, number\* value)

{

int i, c, j;

fprintf(file, "\_c\_");

while (((value->current\_count) - 1) % 4 != 0)

{

add\_digit(value, 0);

}

for (i = 0; i < value->current\_count - 1; i += 4)

{

c = 0;

for (j = i + 3; j >= i; j--)

{

c = c \* 2 + value->mas[j];

}

fprintf(file, "%c", (char)(c + (int)'a'));

}

fprintf(file, "#\n");

}

void check\_readed\_num(char\* string)

{

char buff[PREFIX\_SIZE + 1];

int i; // iterator

int str\_len;

char check\_str\_min = 'a', check\_str\_max = 'a' + 15;

BOOL result = FALSE;

buff[PREFIX\_SIZE] = '\0';

for (i = 0; i < PREFIX\_SIZE; i++)

{

buff[i] = string[i];

}

if (!strcmp("\_n\_", buff) || !strcmp("\_d\_", buff) || !strcmp("\_e\_", buff) || !strcmp("\_c\_", buff))

result = TRUE;

if (!result)

{

\_log("Wrong number prefix");

exit(GET\_NUMBER\_FAILURE);

}

#ifdef \_WIN32

str\_len = strlen(string) - 2;

#else

str\_len = strlen(string) - 2;

#endif

if (string[str\_len] != '#')

{

\_log("End of number doesnt exist");

exit(GET\_NUMBER\_FAILURE);

}

for (i = 3; i < str\_len; i++)

{

if (string[i]<check\_str\_min || string[i] > check\_str\_max)

{

\_log("There's exist some unexpected symbol");

exit(GET\_NUMBER\_FAILURE);

}

}

}

int read\_num\_from\_file(FILE\* file, number\* value)

{

int i, j; // iterators

int c;

int ms[4];

char buff[2048];

fgets(buff, 2048, file);

if (!strcmp("EOF", buff))

{

return 0;

}

check\_readed\_num(buff);

i = 3;

while ((c = (int)buff[i]) != (int)'#')

{

c = (int)(buff[i] - 'a');

for (j = 0; j < 4; j++)

{

ms[j] = c % 2;

c >>= 1;

}

for (j = 0; j < 4; j++)

{

add\_digit(value, ms[j]);

}

i++;

}

return 1;

}

void save\_key(char\* filename, number\* mod, number\* subkey, char log)

{

FILE\* file;

char str[2];

int i, j; // iterators

int c;

file = check\_file\_exist\_write(filename);

fprintf(file, "\_n\_");

while (((mod->current\_count) - 1) % 4 != 0)

{

add\_digit(mod, 0);

}

for (i = 0; i < mod->current\_count - 1; i += 4)

{

c = 0;

for (j = i + 3; j >= i; j--)

{

c = c \* 2 + mod->mas[j];

}

fprintf(file, "%c", (char)(c + (int)'a'));

}

fprintf(file, "#\n\_%c\_", log);

while (((subkey->current\_count) - 1) % 4 != 0)

{

add\_digit(subkey, 0);

}

for (i = 0; i < subkey->current\_count - 1; i += 4)

{

c = 0;

for (j = i + 3; j >= i; j--)

{

c = c \* 2 + subkey->mas[j];

}

fprintf(file, "%c", (char)(c + (int)'a'));

}

fprintf(file, "#\n&");

fclose(file);

\_log("Key saved succesfully. Subkey was: ");

str[0] = log;

str[1] = '\0';

\_log(str);

}

void read\_key(char\* filename, number\* mod, number\* subkey, char log)

{

FILE\* file;

char str[2];

int i, j; // iterators

int c;

int ms[4];

\*mod = init(); \*subkey = init();

file = check\_file\_exist\_read(filename);

char buff[2048];

fgets(buff, 2048, file);

i = 3;

while ((c = (int)buff[i]) != (int)'#')

{

c = (int)(buff[i] - 'a');

for (j = 0; j < 4; j++)

{

ms[j] = c % 2;

c >>= 1;

}

for (j = 0; j < 4; j++)

{

add\_digit(mod, ms[j]);

}

i++;

}

fgets(buff, 2048, file);

i = 3;

while ((c = (int)buff[i]) != (int)'#')

{

c = (int)(buff[i] - 'a');

for (j = 0; j < 4; j++)

{

ms[j] = c % 2;

c >>= 1;

}

for (j = 0; j < 4; j++)

{

add\_digit(subkey, ms[j]);

}

i++;

}

normalize(mod);

normalize(subkey);

fclose(file);

\_log("Key parced succesfully. We read: ");

str[0] = log;

str[1] = '\0';

\_log(str);

}

void check\_readed\_prime(char\* string)

{

char buff[PRIME\_PREFIX\_SIZE + 1];

int i; // iterator

int str\_len;

char check\_str\_min = '0', check\_str\_max = '0' + 1;

char check\_prx\_min = '0', check\_prx\_max = '0' + 9;

BOOL result = FALSE;

if (strcmp("&", string))

{

buff[PRIME\_PREFIX\_SIZE] = '\0';

for (i = 0; i < PRIME\_PREFIX\_SIZE; i++)

{

buff[i] = string[i];

}

if (buff[0] != '\_')

{

\_log("Wrong prime number prefix:");

\_log(buff);

exit(GET\_PRIME\_FAILURE);

}

i--;

while (buff[i] != '\_')

{

buff[i] = '\0';

i--;

}

str\_len = strlen(buff);

if (str\_len < 3)

{

\_log("Wrong prime number prefix:");

\_log(buff);

snprintf(buff, 5, "%d", str\_len);

\_log(buff);

exit(GET\_PRIME\_FAILURE);

}

for (i = 1; i < str\_len - 1; i++)

{

if (string[i]<check\_prx\_min || string[i] > check\_prx\_max)

{

\_log("There's exist some unexpected symbol");

\_log(buff);

buff[1] = '\0';

buff[0] = string[i];

\_log(buff);

snprintf(buff, 5, "%d", i);

\_log(buff);

exit(GET\_PRIME\_FAILURE);

}

}

#ifdef \_WIN32

str\_len = strlen(string) - 2;

#else

str\_len = strlen(string) - 3;

#endif

if (string[str\_len] != '#')

{

\_log("End of number doesnt exist");

buff[1] = '\0';

buff[0] = string[str\_len];

\_log(buff);

snprintf(buff, 5, "%d", str\_len);

\_log(buff);

snprintf(buff, 5, "%d", (int)string[str\_len]);

\_log(buff);

exit(GET\_PRIME\_FAILURE);

}

for (i = strlen(buff); i < str\_len; i++)

{

if (string[i]<check\_str\_min || string[i] > check\_str\_max)

{

\_log("There's exist some unexpected symbol");

exit(GET\_PRIME\_FAILURE);

}

}

}

}

number get\_prime\_from\_file(char\* filename, int line\_number, int bit\_size)

{

FILE\* file;

int c = 0, line\_num, i=0;

char str[3];

number res = init();

char buff[4096];

file = check\_file\_exist\_read(filename);

fgets(buff, 4096, file);

check\_readed\_prime(buff);

while (strcmp("&", buff))

{

i = 1;

c = buff[i];

line\_num = 0;

while (c != (int)'\_')

{

line\_num \*= 10;

line\_num += (int)(c - '0');

i++;

c = buff[i];

}

i++;

c = buff[i];

if (line\_num == line\_number)

{

while (c != (int)'#')

{

add\_digit(&res, (int)(c - '0'));

i++;

if (i > 2100)

{

\_log("Doesnt got prime number succesfully. Bit size doesnt match");

fclose(file);

exit(GET\_PRIME\_FAILURE);

}

c = buff[i];

}

if (res.current\_count - 1 != bit\_size)

{

\_log("Doesnt got prime number succesfully. Bit size doesnt match");

fclose(file);

exit(GET\_PRIME\_FAILURE);

}

\_log("Got prime number succesfully");

fclose(file);

return res;

}

fgets(buff, 4096, file);

check\_readed\_prime(buff);

}

sprintf(str, "%d", line\_number);

\_log("Doesnt got prime number succesfully. No prime. Line number was: ");

\_log(str);

fclose(file);

exit(GET\_PRIME\_FAILURE);

}

# Приложение З

Листинг файла «help.h»

#ifndef HELP\_H

#define HELP\_H

#define HELP\_TEXT "\

Course work created by Ulanovsky George\n\n\

usage: \"programm name\" [option] ... [arg] ...\n\

note: all files must be in .txt format\n\

Options and arguments (and corresponding environment variables):\n\

genkey : Generates keys with const size. Needs --size, --pubkey, --secret options.\n\

sign : Finds file signature and saves it. Needs --infile, --secret, --sigfile options.\n\

check : Checks file signature and types result into console. Needs --infile, --pubkey, --sigfile options.\n\

encrypt : Encrypt file using key. Needs --infile, --pubkey, --outfile options.\n\

decrypt : Decrypt file using key. Needs --infile, --secret, --outfile options.\n\

--size size : Setup size for the key in bits. Must be a degree of 2 (between 256 and 2048).\n\

--pubkey filename : Setup a filename where will be saved public key.\n\

--secret filename : Setup a filename where will be saved secret key.\n\

--sigfile filename : Setup a filename where will be saved or checked signature.\n\

--infile filename : Setup a filename where is saved input information.\n\

--outfile filename : Setup a filename where will be saved output information.\n\

-h : Print this help message and exit (also --help) \n"

#define ERROR\_FIRST\_PARAMETER "\

Unknown option : %s\n\

usage: \"programm name\" [option] ... [arg] ...\n\

Try '\"programm name\" -h' for more information.\n"

#define ERROR\_NO\_PARAMETERS "\

You had typed no options\n\

usage: \"programm name\" [option] ... [arg] ...\n\

Try '\"programm name\" -h' for more information.\n"

#define WRONG\_PARAMETER "\

You had typed unknown option: %s\n\

Expected option: %s\n\

usage: \"programm name\" [option] ... [arg] ...\n\

Try '\"programm name\" -h' for more information.\n"

#define ERROR\_NOT\_ENOUGH\_PARAMETERS "\

You had typed not enough options\n\

usage: \"programm name\" [option] ... [arg] ...\n\

Try '\"programm name\" -h' for more information.\n"

#define ERROR\_DEFAULT "\

You had typed wrong options\n\

usage: \"programm name\" [option] ... [arg] ...\n\

Try '\"programm name\" -h' for more information.\n"

#define ERROR\_FILE\_IDENT "\

You entered same filenames\n\

usage: \"programm name\" [option] ... [arg] ...\n\

Try '\"programm name\" -h' for more information.\n"

#define ERROR\_FILE\_EXTENSION "\

You entered wrong filename (symbols count or extension)\n\

usage: \"programm name\" [option] ... [arg] ...\n\

Try '\"programm name\" -h' for more information.\n"

#endif // !HELP\_H

# Приложение И

Листинг файла «config.h»

#ifndef CONFIG\_H

#define CONFIG\_H

#define \_CRT\_SECURE\_NO\_WARNINGS

//#define DEBUG

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <string.h>

#include <malloc.h>

#include <time.h>

#include <stdint.h>

#include <locale.h>

#include <sys/stat.h>

// input console checking

#define HELP\_CONSOLE\_OPTION\_1 "-h"

#define HELP\_CONSOLE\_OPTION\_2 "--help"

#define GENKEY\_CONSOLE\_OPTION "genkey"

#define SIGNATURE\_CONSOLE\_OPTION "sign"

#define CHECK\_CONSOLE\_OPTION "check"

#define ENCRYPT\_CONSOLE\_OPTION "encrypt"

#define DECRYPT\_CONSOLE\_OPTION "decrypt"

#define SIZE\_CONSOLE\_OPTION "--size"

#define PUBKEY\_CONSOLE\_OPTION "--pubkey"

#define SECRET\_CONSOLE\_OPTION "--secret"

#define SIGFILE\_CONSOLE\_OPTION "--sigfile"

#define INFILE\_CONSOLE\_OPTION "--infile"

#define OUTFILE\_CONSOLE\_OPTION "--outfile"

// boolean

#define BOOL int

#define TRUE 1

#define FALSE 0

// RSA

#define KEYSIZE\_MODULE 256

typedef enum {

KEY\_256 = 256,

KEY\_512 = 512,

KEY\_1024 = 1024,

KEY\_2048 = 2048

} KEY\_BIT\_SIZE;

// file

#define FILE\_OPENING\_ERROR "Cannot open the %s file.\n"

#define primes\_128\_bit\_filename "primes\_128\_bit.blackleague"

#define primes\_256\_bit\_filename "primes\_256\_bit.blackleague"

#define primes\_512\_bit\_filename "primes\_512\_bit.blackleague"

#define primes\_1024\_bit\_filename "primes\_1024\_bit.blackleague"

/// <summary>

/// Exit codes for exit()

/// </summary>

enum EXIT\_CODE {

SUCCESS,

FAILURE,

NO\_ARGUMENTS\_FAILURE,

WRONG\_ARGUMENT\_FAILURE,

NOT\_ENOUGH\_ARGUMENTS\_FAILURE,

FILE\_OPEN\_FAILURE,

FILE\_FORMAT\_FAILURE,

LOG\_FAILURE,

MEMORY\_ALLOCATION\_FAILURE,

GET\_PRIME\_FAILURE,

GET\_NUMBER\_FAILURE,

DEBUG\_EXIT\_CODE = 100

};

// fast funcs

#define swap(a,b); b = a+b; a = b-a; b = b-a;

#ifdef \_WIN32

#else

#define max(a,b) (a>b)? a : b

#endif

#endif // !CONFIG\_H

# Приложение Й

Листинг файла «Coursework.с»

#include "RSA.h"

/// <summary>

/// Function that is checking for parameters for work mode

/// </summary>

/// <param name="argv">Input parameters from console</param>

/// <param name="work\_mode">Expected work mode, that will be checked</param>

/// <param name="arg\_1">Expected parameter 1</param>

/// <param name="arg\_2">Expected parameter 2</param>

/// <param name="arg\_3">Expected parameter 3</param>

/// <returns>0 if work mode isnt expected, 1 if parameters are right</returns>

int check\_console\_input\_format(char\*\* argv, char\* work\_mode, char\* arg\_1, char\* arg\_2, char\* arg\_3)

{

if (!strcmp(argv[1], work\_mode)){

if (!strcmp(argv[2], arg\_1)){

if (!strcmp(argv[4], arg\_2)){

if (!strcmp(argv[6], arg\_3)){

return 1;

}

else {

printf(WRONG\_PARAMETER, argv[6], arg\_3);

exit(WRONG\_ARGUMENT\_FAILURE);

}

}

else {

printf(WRONG\_PARAMETER, argv[4], arg\_2);

exit(WRONG\_ARGUMENT\_FAILURE);

}

}

else {

printf(WRONG\_PARAMETER, argv[2], arg\_1);

exit(WRONG\_ARGUMENT\_FAILURE);

}

}

else

return 0;

}

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

{

setlocale(LC\_ALL, "rus");

\_log\_start();

if (argc == 1)

{

printf(ERROR\_NO\_PARAMETERS);

\_log("User typed no parameters");

exit(NO\_ARGUMENTS\_FAILURE);

}

else if (argc == 2)

{

if (!strcmp(argv[1], HELP\_CONSOLE\_OPTION\_1) || !strcmp(argv[1], HELP\_CONSOLE\_OPTION\_2))

{

printf(HELP\_TEXT);

\_log("User typed -h or --help parameter");

exit(WRONG\_ARGUMENT\_FAILURE);

}

else if (strcmp(argv[1], GENKEY\_CONSOLE\_OPTION) && strcmp(argv[1], CHECK\_CONSOLE\_OPTION) && strcmp(argv[1], SIGNATURE\_CONSOLE\_OPTION) && strcmp(argv[1], ENCRYPT\_CONSOLE\_OPTION) && strcmp(argv[1], DECRYPT\_CONSOLE\_OPTION))

{

printf(ERROR\_FIRST\_PARAMETER, argv[1]);

\_log("User typed wrong 1 parameter");

exit(WRONG\_ARGUMENT\_FAILURE);

}

else

{

printf(ERROR\_NOT\_ENOUGH\_PARAMETERS);

\_log("User typed not enough parameters");

exit(NOT\_ENOUGH\_ARGUMENTS\_FAILURE);

}

}

else if (argc == 8)

{

// progname workmode arg1 -- arg2 -- arg3 --

// argv0 argv1 argv2 argv3 argv4 argv5 argv6 argv7

if (check\_console\_input\_format (argv, GENKEY\_CONSOLE\_OPTION, SIZE\_CONSOLE\_OPTION, PUBKEY\_CONSOLE\_OPTION, SECRET\_CONSOLE\_OPTION))

{

\_log("User chose generate key option");

check\_filenames\_2(argv[5], argv[7]);

generate\_key(argv[3], argv[5], argv[7]);

}

else if (check\_console\_input\_format(argv, SIGNATURE\_CONSOLE\_OPTION, INFILE\_CONSOLE\_OPTION, SECRET\_CONSOLE\_OPTION, SIGFILE\_CONSOLE\_OPTION))

{

\_log("User chose signify file option");

check\_filenames\_3(argv[3], argv[5], argv[7]);

signify\_file(argv[3], argv[5], argv[7]);

}

else if (check\_console\_input\_format(argv, CHECK\_CONSOLE\_OPTION, INFILE\_CONSOLE\_OPTION, PUBKEY\_CONSOLE\_OPTION, SIGFILE\_CONSOLE\_OPTION))

{

\_log("User chose check files sign option");

check\_filenames\_3(argv[3], argv[5], argv[7]);

check\_sign\_file(argv[3], argv[5], argv[7]);

}

else if (check\_console\_input\_format(argv, ENCRYPT\_CONSOLE\_OPTION, INFILE\_CONSOLE\_OPTION, PUBKEY\_CONSOLE\_OPTION, OUTFILE\_CONSOLE\_OPTION))

{

\_log("User chose to encrypt file option");

check\_filenames\_3(argv[3], argv[5], argv[7]);

encrypt\_file(argv[3], argv[5], argv[7]);

}

else if (check\_console\_input\_format(argv, DECRYPT\_CONSOLE\_OPTION, INFILE\_CONSOLE\_OPTION, SECRET\_CONSOLE\_OPTION, OUTFILE\_CONSOLE\_OPTION))

{

\_log("User chose to decrypt file option");

check\_filenames\_3(argv[3], argv[5], argv[7]);

decrypt\_file(argv[3], argv[5], argv[7]);

}

else

{

printf(ERROR\_FIRST\_PARAMETER, argv[1]);

\_log("User typed wrong parameters");

exit(WRONG\_ARGUMENT\_FAILURE);

}

}

else

{

printf(ERROR\_DEFAULT);

\_log("User typed wrong number of parameters");

exit(WRONG\_ARGUMENT\_FAILURE);

}

return 0;

}