ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ**

**ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

(НИУ «БелГУ»)

ИНСТИТУТ ИНЖЕНЕРНЫХ И ЦИФРОВЫХ ТЕХНОЛОГИЙ

КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

**Отчет по лабораторной работе №1**

**по дисциплине: «Методы распараллеливания программ»**

**Тема: «Изучение свойств и структуры алгоритма»**

студента очного отделения

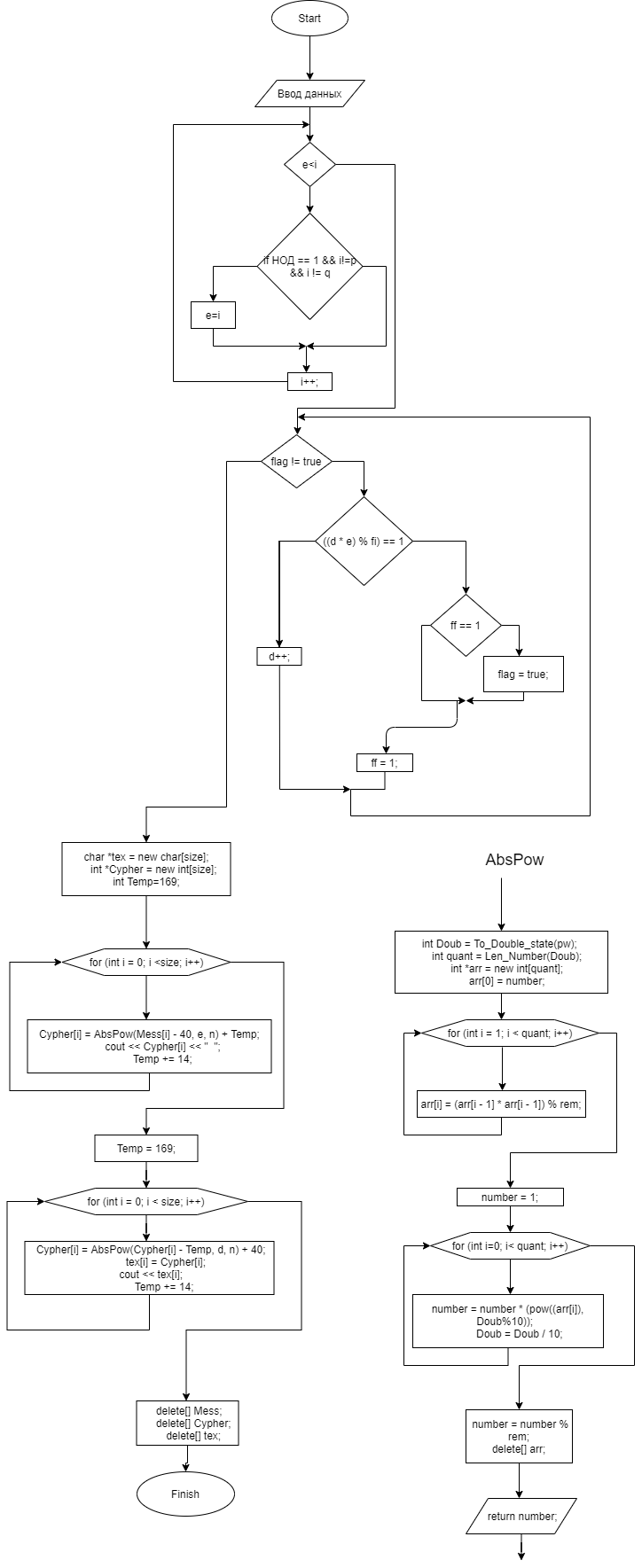
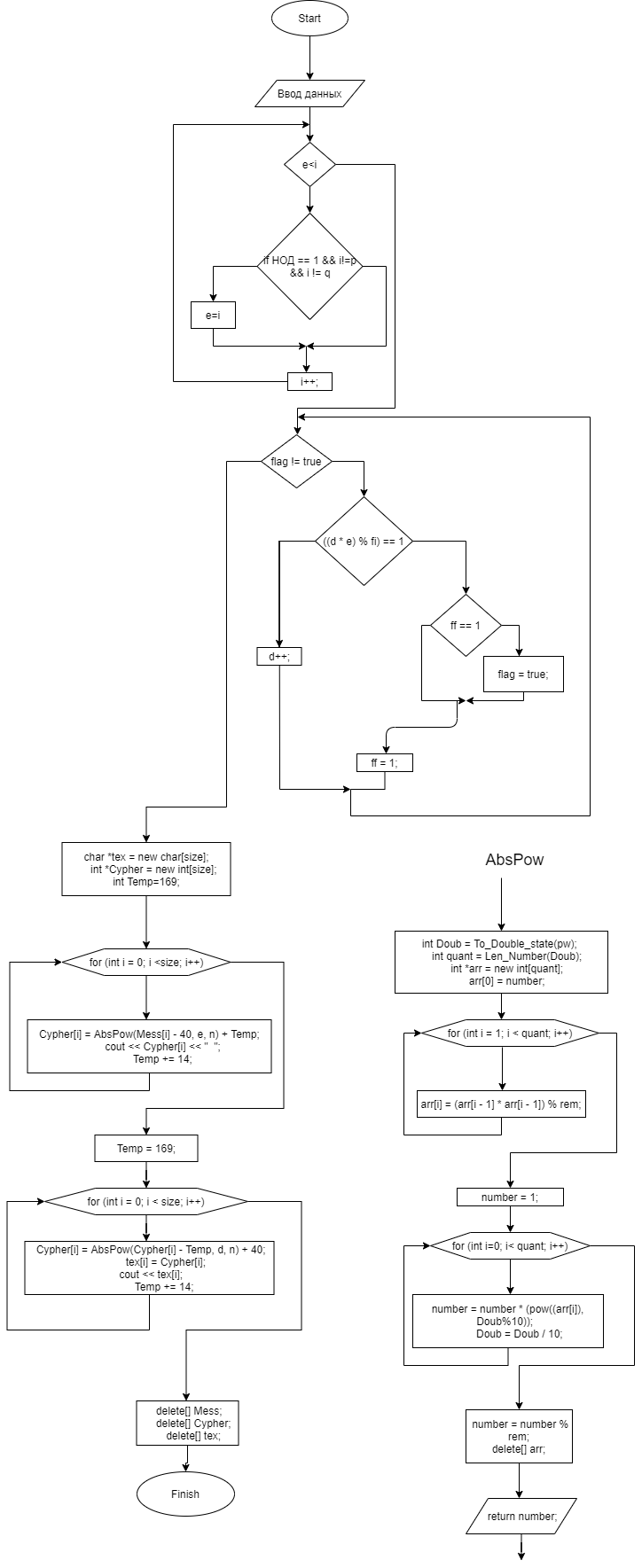
3 курса 12001801 группы

Капустина Виктора Сергеевича

Проверил(а):

Петров Денис Васильевич

Белгород 2020

**16 Вариант  
Блок схема****

**Общее описание алгоритма**

*RSA шифрование* - криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел. Криптосистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи. Алгоритм используется в большом числе криптографических приложений.

Перед шифрованием необходимо произвести дополнительные расчеты, такие как:   
-Выбор двух простых чисел (p,q)

-Нахождение модуля их произведения (n)

-Вычисление функции Эйлера (φ)

-Выбор числа e, соответствующего трем критериям: (I) оно должно быть простое, (II) оно должно быть меньше φ (III) оно должно быть взаимно простое с φ.

-Вычисление числа d, обратное e по модулю φ

Теперь считается, что {e,n} - открытый ключ, по которому шифруется сообщение. Замечательно, что по нему практически нереально расшифровать полученное сообщение. Для этого будет использоваться полученный закрытый ключ {d,n}, который рекомендуется никому не сообщать.

Вводимое пользователем сообщение разбивается посимвольно и каждый из них кодируется по открытому ключу. Мы получаем зашифрованную последовательность, что и высылается. Уже на приеме идет дешифровка по закрытому ключу каждого элемента сообщения, где мы и получаем наши символы обратно.

**Математическое описание алгоритма**

Необходимые расчеты:

-Нахождение модуля произведения P и Q (n): n=|P \* Q|

-Вычисление функции Эйлера (φ) : φ=(P-1)\*(Q-1)

-Проверка взаимной простоты φ и е - Алгоритм Евклида:

1. Большее число делим на меньшее.
2. Если делится без остатка, то меньшее число и есть НОД (следует выйти из цикла).
3. Если есть остаток, то большее число заменяем на остаток от деления.
4. Переходим к пункту 1.

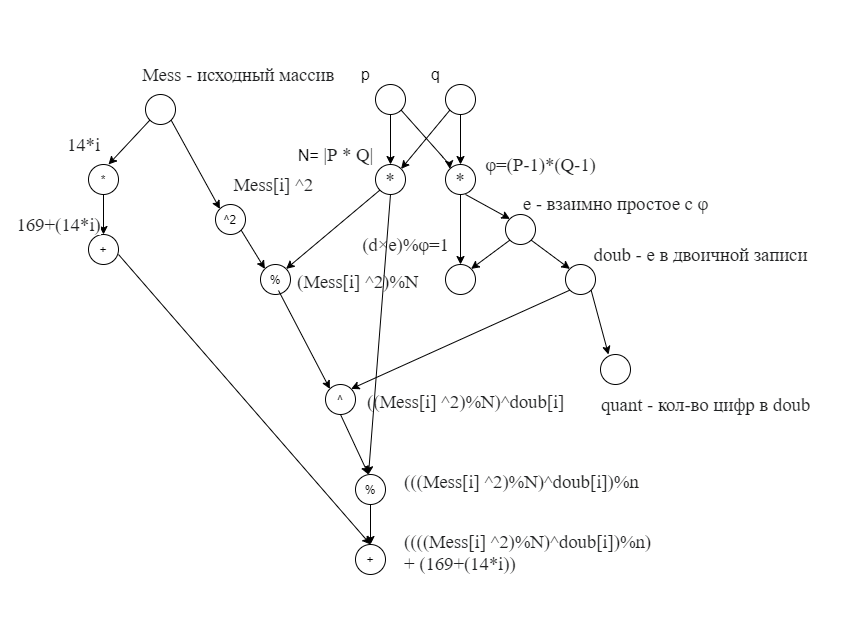
-Вычисление d: (d×е)%φ=1 ,где е и φ были получены в предыдущих шагах.

-Шифрование происходит путем возведения каждого элемента в степень по модулю. Так, чтобы чтобы возвести ***ax mod p*** воспользуемся алгоритмом быстрого возведения в степень по модулю, где *a* соответствует элементу массива **arr,** *x*соответствует **e,** и *p* соответствует **n**:

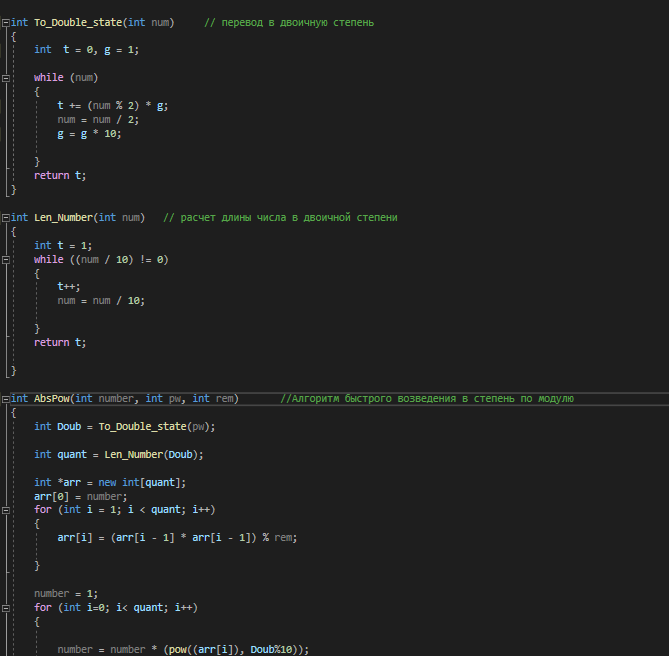
1. Перевод степень числа **x** в двоичную систему (Doub)
2. Определить **quant**, которое равно количеству всех цифр степени в двоичной системе.
3. Составить массив данных, где первый элемент равен исходному числу **a,** а каждый последующий вычисляется так: Возводим в квадрат текущее значение и находим остаток от деления на ***p.*** По итогу получим массив значений **arr**.
4. Вычислить произведение элементов массива **arr**, которые возведены в соответствующие степени из двоичной записи числа на 1 шаге. Требуется каждый элемент массива возвести в степень (либо в первую, либо в нулевую), затем перемножить то, что получилось. *Степени соответствуют обратному порядку двоичной системы из 1 шага.*
5. Вычислить остаток от деления полученного произведения.

Для дополнительной защиты шифрования, после кодирования символа в число, к его значение прибавляется 169 + (14\*n) где n-номер итерации. Для первого символа n будет равно 0, т.е. его закодированному значению добавится 169. Во время дешифровки вначале отнимается 169 + (14\*n) по такому же принципу и только потом декодируется в символ.

**Граф информационной зависимости**

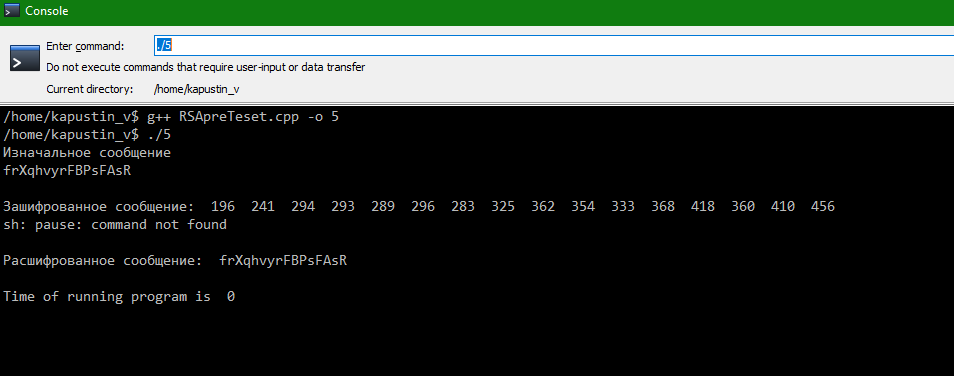
****

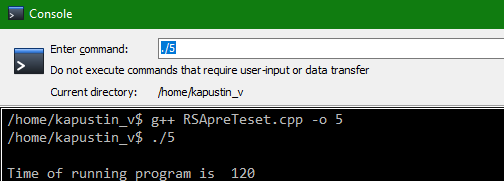
**Копии экрана иллюстрирующие процесс разработки**

****Рис. 1 Процесс разработки

## Таблица 1. Зависимость времени вычислений от объема исходных данных

|  |  |  |
| --- | --- | --- |
| Номер расчета | Объем исходных данных | Время расчета, мс |
| 1 | 16 | 0 |
| 2 | 16\*100 | 0 |
| 3 | 16\*10000 | 120 |
| 4 | 16\*1000000 | 8250 |
| 5 | 16\*10000000 | 83220 |

**Копии экрана, иллюстрирующие работу приложения.  
**Рис. 2 Пример реализации программы в тестовом режиме

  
Рис. 3 Пример реализации программы в рабочем режиме  
  
  
Рис. 4 График зависимости времени расчетов от объема исходных данных  
  
  
**Листинг программы**

#include <iostream>

#include <cstdlib>

#include <cctype>

#include <string>

#include <bitset>

#include <cmath>

#include <time.h>

#include <ctime>

#include <string>

#include <iomanip>

using namespace std;

int gcd(int x, int y)

{

return y ? gcd(y, x % y) : x;

}

int To\_Double\_state(int num) // перевод в двоичную степень

{

int t = 0, g = 1;

while (num)

{

t += (num % 2) \* g;

num = num / 2;

g = g \* 10;

}

return t;

}

int Len\_Number(int num) // расчет длины числа в двоичной степени

{

int t = 1;

while ((num / 10) != 0)

{

t++;

num = num / 10;

}

return t;

}

int AbsPow(int number, int pw, int rem) //Алгоритм быстрого возведения в степень по модулю

{

int Doub = To\_Double\_state(pw);

int quant = Len\_Number(Doub);

int \*arr = new int[quant];

arr[0] = number;

for (int i = 1; i < quant; i++)

{

arr[i] = (arr[i - 1] \* arr[i - 1]) % rem;

}

number = 1;

for (int i=0; i< quant; i++)

{

number = number \* (pow((arr[i]), Doub%10));

Doub = Doub / 10;

}

number = number % rem;

delete[] arr;

return number;

}

int main()

{

clock\_t start, end;

start = clock();

string str;

srand(time(0));

setlocale(LC\_ALL, "Russian");

int p, q, n, fi;

int e = 0;

int rnd = rand() % 2;

int size = 160000000;

char \*Mess = new char[size];

for (int i = 0; i < size; i++)

{

Mess[i] = 65 + rand() % 57;

}

switch (rnd)

{

case 0: p = 7; break;

case 1: p = 13; break;

}

q = 17;

n = abs(p \* q);

fi = (p - 1) \* (q - 1);

int i = 2;

while (e < 1) {

if ((gcd(i, fi) == 1) && (i != p) && (i != q))

{

e = i;

}

i++;

}

bool flag = false;

int ff = 0;

int d = 1;

while (flag != true)

{

if (((d \* e) % fi) == 1)

{

if (ff == 1)

{

flag = true;

}

ff = 1;

}

else d++;

}

char \*tex = new char[size];

int \*Cypher = new int[size];

int Temp=169;

for (int i = 0; i <size; i++)

{

Cypher[i] = AbsPow(Mess[i] - 40, e, n) + Temp;

Temp += 14;

}

Temp = 169;

for (int i = 0; i < size; i++)

{

Cypher[i] = AbsPow(Cypher[i] - Temp, d, n) + 40;

tex[i] = Cypher[i];

Temp += 14;

}

delete[] Mess;

delete[] Cypher;

delete[] tex;

end = clock();

double time = ((double)(end - start)) / double( CLOCKS\_PER\_SEC ) \* 1000;

// CLOCKS\_PER\_SEC;

//double time = double(end - start) / double( CLOCKS\_PER\_SEC ) \* 1000;

cout << endl;

cout << "Time of running program is " << time << endl;

system("pause");

return 0;

}