A description...

НТУУ «Київський Політехнічний Інститут»

Спеціальні розділи обчислювальної математики

Комп’ютерний практикум №3

**Виконав:**

Студент 3 курсу ФТІ

групи ФИ-83

Паршин О.Ю.

**Прийняла:**

Пекарчук Н.А.

1. **Мета роботи**

Одержання практичних навичок програмної реалізації обчислень у полі Галуа характеристики 2 в поліноміальному базисі; ознайомлення з прийомами ефективної реалізації критичних по часу ділянок програмного коду та методами оцінки їх ефективності.

1. **Теоретичні відомості**

**2.1. Деякі відомості про скінченні поля**

*Полем* називається множина елементів з двома заданими на ній бінарними операціями,додаванням та множенням (+ та ×, інколи позначаються Å та Ä ) для яких виконуються умови:

а) щодо операції додавання елементи поля утворюють абелеву групу з нейтральним елементом 0;

б) щодо операції множення всі елементи, окрім 0, також утворюють абелеву групу з нейтральним елементом 1;

в) додавання та множення пов’язані між собою законом дистрибутивності: для будь-яких елементів поля *x, y, z* виконується *x*( *y* + *z*) = *xy* + *xz* .

Число елементів поля називається *порядком* поля. Поле називається *скінченним* (або *полем* *Галуа*),якщо воно має скінченну кількість елементів.Скінченне поле порядку *q* позначається

*GF*(*q*)або *Fq* .Порядок скінченного поля завжди є степенем деякого простого числа, *q* = *p m* ,число *m* називається *степенем* поля, а просте число *p* – його *характеристикою*.

Абелева група ненульових елементів поля з операцією множення називається *мультиплікативною групою поля* (позначається *GF*\*(*q*)або *Fq*\*)Мультиплікативна група

скінченного поля є циклічною групою порядку *p m* -1,її твірний елемент(або генератор)

називається *примітивним* елементом поля.

Скінченне поле степеня 1 називається *простим*. Просте скінченне поле можна ототожнити

* множиною класів лишків за модулем числа *p* з операціями додавання та множення за модулем *p.* Наприклад,скінченне поле *GF*(2)складається з двох елементів0і1.У цьому полі операції

додавання й множення виконуються наступним чином: 0+0=0, 0+1=1+0=1, 1+1=0, 0·0=1·0=0·1=0, 1·1=1, тобто за модулем 2.

*Многочленом f*(*t*)степеня *m* над полем *GF*(*p*)є вираз вигляду

*f* (*t*)= *amt m* + *am*-1*t m*-1+ + *a*1*t* + *a*0 ,

де коефіцієнти многочлена *ai* Î*GF*(*p*), *i* = 0, *m* , а *t* – змінна, деякий символ, що не належить полю.

Операції над такими многочленами виконуються як операції над звичайними многочленами, тільки операції над коефіцієнтами здійснюються в полі *GF*(*p*). Зокрема, многочлен *g*(*t*) ділиться з залишком *r*(*t*) на многочлен *f* (*t*) , *f* (*t*) ¹ 0 , якщо *g*(*t*) = *h*(*t*) *f* (*t*) + *r*(*t*) , де степінь многочлена *r*(*t*) менша за степінь многочлена *f* (*t*) . Операція обчислення залишку від ділення многочлена *g*(*t*) на многочлен *f* (*t*) називається *зведенням* (або *редукцією*) многочлена *g*(*t*)за модулем *f* (*t*),а залишок *r*(*t*)позначається *g*(*t*) mod *f* (*t*).Якщо *r*(*t*)º0,то многочлен *g*(*t*)ділиться на многочлен *f* (*t*)без залишку.

Многочлен *f* (*t*)ненульового степеня називається *незвідним* над полем *GF*(*p*),якщо він

|  |  |  |  |
| --- | --- | --- | --- |
| ділиться без залишку над цим полем тільки на самого себе і на многочлени нульового степеня. | | | |
| Елемент *x* | скінченного поля | *GF*(*pm* )називається коренем многочлена *f* (*t*), | якщо *f* (*x*) = 0 . |
| Незвідний | многочлен *f* (*t*) | називається *примітивним*, якщо його корені є | примітивними |

елементами поля.

Будь-яке скінченне поле *GF*(*pm* ) є *m*-вимірним векторним простором над полем *GF*(*p*). Якщо *x* - корінь незвідного многочлена *f* (*t*) степеня *m* над *GF*(*p*), то елементи

{*xm*-1 , , *x*, 1} утворюють *базис* скінченного поля *GF*(*pm* ) як векторного простору над полем *GF*(*p*).Цей базис називається *поліноміальним*.Будь-який елемент основного поля однозначновиражається через елементи поліноміального базису. Найзручніше поліноміальний базис задавати примітивним многочленом *f* (*t*) степеня *m* . Тоді елементи поля задаються

многочленами степеня не вище *m* -1, їх сума задається сумою цих многочленів, а добуток – добутком цих многочленів по модулю *f* (*t*) .

Для будь-яких елементів *х*, *у* скінченного поля *GF*(*pm* ) мають місце рівності:

*x pm* = *x* , (*x* + *y*) *p* = *x* *p* + *y* *p* .

Таким чином, операція піднесення до степеня *р* у полі *GF*(*pm* ) лінійна над *GF*(*p*):

(*ax* + *by*) *p* = *ax* *p* + *by* *p* ,

для довільних *a*, *b* Î*GF*(*p*), *x*, *y* Î*GF*(*p* *m* ).

**2.2. Виконання операцій у поліноміальному базисі полів характеристики 2**

У поліноміальному зображенні елементи поля *GF*(2*m* ) являють собою многочлени степеня, що не перевищує *m* -1 , над *GF*(2) . Елементи *GF*(2*m* ) зображуються двійковими

векторами, що відповідають їх розкладу за базисними елементами (тобто, коефіцієнтам відповідного полінома), причому крайній лівий розряд зображення елемента поля відповідає

степеню *x*2*m* -1,а крайній правий–степеню *x*0;таким чином,елемент1має зображення(0,0,0,...,0,1) або просто 0000…01, а елемент *x* – зображення (0,0,0,...,1,0) або 0000…010.

***2.2.1. Додавання у поліноміальному базисі***

Додавання у *GF*(2*m* ) є звичайним додаванням поліномів над *GF*(2) , що відповідає покомпонентному додаванню за модулем 2 відповідних векторів.

***2.2.2. Множення у поліноміальному базисі***

При множенні елементів *GF*(2*m* ) відповідні їм многочлени перемножуються, з наступним зведенням результату за модулем незвідного многочлена *f* (*t*) , який використовується для побудови *GF*(2*m* ) як розширення *GF*(2) .

***2.2.3. Піднесення до квадрату в поліноміальному базисі***

Піднесення елементу поля *GF*(2*m* ) до квадрату можна зробити як звичайне множення

цього елементу сам на себе. Втім, із використанням властивості лінійності піднесення до квадрату, можна зробити цю операцію більш ефективно.

Нехай дано елемент *a* Î*GF*(2*m* ):

*a* =(*am* , *am*-1,..., *a*1, *a*0)= *amt m* + *am*-1*t m*-1+ + *a*1*t* + *a*0 ;

тоді його квадрат буде виглядати таким чином:

*a* =(*amt m* + *am*-1*t m*-1+ + *a*1*t* + *a*0 )2 mod *f* (*t*) = *amt* 2*m* + *am*-1*t* 2*m*-2 + + *a*1*t* 2 + *a*0 mod *f* (*t*) ,

або, в бітовому записі,

*a*2=(*am* ,0, *am*-1,0,..., *a*1,0*a*0) mod *f* (*t*).

Отже, для того, щоб обчислити квадрат елементу *a*, треба виконати такі дії:

1. «Прорідити» бітовий запис, вставляючи 0 після кожного біту, окрім останнього.
2. Отриманий бітовий вектор довжини 2*m* +1 біт представити як поліном та звести за модулем *f* (*t*)

***2.2.4. Знаходження оберненого елемента поліноміальному базисі***

Обернений елемент до *а* ( *a* ¹ 0 ) можна знайти як *a* 2*m* -2 . Обчислення правої частини цієї формули ефективно виконується за схемою Горнера: оскільки 2*m* - 2 = 2*m*-1 + 2*m*-2 + ... + 2 , то

*a*-1 = *a*2*m*-1 *a*2*m*-2 ...*a*2 = (*a*2*m*-2 *a*2*m*-3 ...*a*20 )2 .

Більш швидкий спосіб: за допомогою розширеного алгоритму Евкліда знайти поліном, обернений до поліному *a* за модулем *f* (*x*) .

***2.2.5. Приклади***

Розглянемо поле *GF*(23 ), для побудови якого використаємо примітивний многочлен 3-го степеня *f* (*x*) = *x*3 + *x* +1.

Елементами *GF*(23 ) є поліноми 0, 1, *х*, *x* +1, *x* 2, *x*2+1, *x*2+ *x* , *x*2+ *x* +1або,інакше,

відповідні їм вектори (000), (001), (010), (011), (100), (101), (110), (111).

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

**Завдання до комп’ютерного практикуму**

А) Реалізувати поле Галуа характеристики 2 степеня *m* в поліноміальному базисі з операціями:

1. знаходження константи **0** – нейтрального елемента по операції «+»;
2. знаходження константи **1** – нейтрального елемента по операції «×»;
3. додавання елементів;
4. множення елементів;
5. обчислення сліду елементу;
6. піднесення елемента поля до квадрату;
7. піднесення елемента поля до довільного степеня (не вище 2*m* -1, де *m* – розмірність розширення);
8. знаходження оберненого елемента за множенням;
9. конвертування (переведення) елемента поля в *m* -бітний рядок (строкове зображення) і навпаки, де *m* – розмірність розширення;

Мова програмування, семантика функцій, спосіб реалізації можуть обиратись довільно. Під час конвертування елементів поля у бітові рядки потрібно враховувати конвенції щодо зображень елементів поля (зокрема, порядок бітів).

**Варіант 13:**

|  |  |  |
| --- | --- | --- |
| **Номер** | ***m* (розмірність** | *p*(*x*)**(генератор поля)** |
| **варіанта** | **поля)** |  |
| 13 | 419 | *p*(*x*)= *x*419+ *x*21+ *x*14+ *x* +1 |

Б) Проконтролювати коректність реалізації поля для кожної операції; наприклад, для декількох *a*, *b*, *c*, *d* перевірити тотожності (*a* + *b*) × *c* = *b* × *c* + *c* × *a* , *d* 2*m* -1 = 1 ( *d* ¹ 0 ) та ін.

В) Визначити середній час виконання операцій у полі. Підрахувати кількість тактів процесора (або інших одиниць виміру часу) на кожну операцію. Результати подати у вигляді таблиць або діаграм.

**Результати**

*Тестові дані:*

*A:* 10101101010110010011011001010101101110011110100011000111010000110000010001011110111001010111111000011110101011010101010111101011100110000001101100001010111011111101011101100101101100111001100001111111100101110000110110100001001010101100000111100010000101111100111000110011011011001011000011111001000111101101010001000111100111011101011011010010010010001001011011001101010001011011101001001011100101111010101000001100111

*B:*

10101101010110010011011001010101101110011110100011000111010000110000010001011110111001010111111000011110101011010101010111101011100110000001101100001010111011111101011101100101101100111001100001111111100101110000110110100001001010101100000111100010000101111100111000110011011011001011000011111001000111101101010001000111100111011101011011010010010010001001011011001101010001011011101001001011100101111010101000001100111

*N:*

00100001011001000000111000100110111011111101001100101111011110010110110101111111100100001000111011011000111011010011010100111110111000111100110001000001100100101001100100111010010110010010101111000000000001000001011010010101000101010011001001011010101000010011010001000100110101000011010000110001101110111110101101101001100100100000001100011010010011100111001111111000011010001111011111010110100010001001100010100111001

A+B:

11100111010000100000001101001010000110010000100010010111101100001110101010000111010100110101101001111010111100001111100100011011110111010001100100101110011111010011011011010011111111010000110101100101001100111101101101111011010000001011011001111111100010010101000001001010110011000011001111000110110101001011011010101111111110101101100111010110111000010011011001010011001101110111101001010000000110011101000001101011100

Середній час виконання: 1.18e-05 s

A\*B:

11110000001001110000101111100101101001111010000001111001011001011111011000100100010000111011010101000011110011011001001101111111100011110101001100000111100000010110011001001110101010100101001100100101111101000111111001101011100110000011101001000000101000100110110001011001111010001000011011011100110010110101010100000011010110101101010011101110010010001010110001111100110100101101011000010001001010111011010100011011000

Середній час виконання: 0.0057702 s

A2:

10100010000010011111000000001100000010110111001001001010010000111011001011111001111000101100110100110110001111100101101000100001100011000110110110001100110110001011100111101110000111111000001100010000010010100010101000110111111001000000000000001100001111010110001100010001110010010101011010010111110100010000100010010000101101001001100111100111111110100011010101110111011001001011010101000111001110000000010000010110111

Середній час виконання: 0.0019756 s

A-1:

11100101001000101100110101000011001000000110100111101110000001010000100101000011011011001111110001101001111101110110000000111110010110011110001111001100001000000011110100000101111101111010011001111010111101111100111011101111011101111111110000100100100000001111011010000101101111101010000010110011111111011011110000010100101110000110111101100011001110011001111100111111111011000101100110001100011110011111011001101100000

Середній час виконання: 2.50205 s

A^N:

String representation:

11100011001011101110101011101010001000110110001100001001001110010101011110111011011111100101100001001000010100101110111100010000000001010000110000010110110000100110100101001100011110110110111000100011110000110001000111011010000100010100001001111111101010110001111000110110001111111100101010100011000100111011010001100100100010110000101001111101011100000011110101000010110100000101001001100101100011110010110001111111100

Середній час виконання: 1.51214 s

Tr(A):

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001

Середній час виконання: 0.60036 s

**Результати тестових перевірок**

*Тест Б.1*

*A:* 10101101010110010011011001010101101110011110100011000111010000110000010001011110111001010111111000011110101011010101010111101011100110000001101100001010111011111101011101100101101100111001100001111111100101110000110110100001001010101100000111100010000101111100111000110011011011001011000011111001000111101101010001000111100111011101011011010010010010001001011011001101010001011011101001001011100101111010101000001100111

*B:*

01001010000110110011010100011111101000001110000001010000111100111110111011011001101101100010010001100100010111011010110011110000010001010000001000100100100100101110000110110110010011101001010100011010101001001101011011011010011010100111011110011101100111101001111001111001101000001000001100111111110010100110001011101000011001110000111100000100101010011010000010011110011100101100000000011011100011100111101001100111011

*C:*

00100001011001000000111000100110111011111101001100101111011110010110110101111111100100001000111011011000111011010011010100111110111000111100110001000001100100101001100100111010010110010010101111000000000001000001011010010101000101010011001001011010101000010011010001000100110101000011010000110001101110111110101101101001100100100000001100011010010011100111001111111000011010001111011111010110100010001001100010100111001

(A + B) \* C:

10101110101100100010010101001100011101110001111001100001101101100110000010110101010011101000110010010001000001001011110000000000110010111000011011000111001001011011001001010101101001011011100110010111110001111100100001110001101010100110100111010110110010011111101011110000100101111001011100010101111110111110101010111101011001111001001100010010011011001010101011111100011010010011010111000011101100111100011111010010001

(A \* C + B \* C):

10101110101100100010010101001100011101110001111001100001101101100110000010110101010011101000110010010001000001001011110000000000110010111000011011000111001001011011001001010101101001011011100110010111110001111100100001110001101010100110100111010110110010011111101011110000100101111001011100010101111110111110101010111101011001111001001100010010011011001010101011111100011010010011010111000011101100111100011111010010001

*Тест Б.2 (num = 15)*

A:

01001010000110110011010100011111101000001110000001010000111100111110111011011001101101100010010001100100010111011010110011110000010001010000001000100100100100101110000110110110010011101001010100011010101001001101011011011010011010100111011110011101100111101001111001111001101000001000001100111111110010100110001011101000011001110000111100000100101010011010000010011110011100101100000000011011100011100111101001100111011

A^(2^m-1):

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001

# Код програми

#include <iostream>

#include <string>

#include <sstream>

#include <array>

#include <chrono>

using namespace std;

const int degree = 419;

struct LongNumber

{

unsigned int\* value;

int size;

string str\_value;

public:

void PrintParameters();

};

LongNumber\* GetValue(string str)

{

LongNumber \*result = new LongNumber;

result->value = new unsigned int[str.length()];

for (int i = str.length() - 1; i >= 0; i--)

{

char temp = str[i];

int number = strtol(&temp, NULL, 2);

result->value[str.length() - 1 - i] = number;

}

result->size = str.length();

result->str\_value = str;

return result;

}

LongNumber\* GetValue(unsigned int A[], int size)

{

LongNumber \*result = new LongNumber;

result->value = A;

result->size = size;

return result;

}

void GetStringValue(LongNumber \*A)

{

string str = "";

for (int i = 0; i < A->size; i++)

{

str.insert(0, to\_string(A->value[i]));

}

A->str\_value = str;

}

int GetFirstNotNullBit(unsigned int A[], int size)

{

int counter = 0;

while (A[size - 1 - counter] == 0)

{

counter++;

}

return size - counter - 1;

}

void LongNumber::PrintParameters() // No memory leak

{

cout << "Parameters: " << endl;

cout << "Size: " << this->size << endl;

cout << "String representation: " << endl << this->str\_value << endl;

}

void KillZeros(LongNumber \*A) // No memory leak

{

int k = GetFirstNotNullBit(A->value, A->size);

unsigned int\* C = new unsigned int[k + 1];

for (int i = 0; i < k + 1; i++)

{

C[i] = A->value[i];

}

delete[] A->value;

A->value = C;

A->size = k + 1;

}

void AddZeros(LongNumber \*A, int size) // No memory leak

{

unsigned int \*C = new unsigned int[size];

for (int i = 0; i < A->size; i++)

{

C[i] = A->value[i];

}

for (int i = A->size; i < size; i++)

{

C[i] = 0;

}

delete A->value;

A->value = C;

A->size = size;

}

LongNumber\* GetModule()

{

string str = "";

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

{

str.insert(0, "0");

}

str[degree - 419] = '1';

str[degree - 21] = '1';

str[degree - 14] = '1';

str[degree - 1] = '1';

str[degree - 0] = '1';

LongNumber \*result = GetValue(str);

return result;

}

LongNumber\* GetZero()

{

string str = "";

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

{

str.insert(0, "0");

}

LongNumber \*result = GetValue(str);

return result;

}

LongNumber\* GetOne()

{

string str = "";

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

{

str.insert(0, "0");

}

str[str.length() - 1] = '1';

LongNumber\* result = GetValue(str);

return result;

}

LongNumber\* CopyLongNumber(LongNumber \*A)

{

LongNumber\* B = new LongNumber;

B->size = A->size;

B->str\_value = A->str\_value;

B->value = A->value;

return B;

}

int LongCmp(LongNumber \*A, LongNumber \*B)

{

int a\_len = GetFirstNotNullBit(A->value, A->size);

int b\_len = GetFirstNotNullBit(B->value, B->size);

if (a\_len - b\_len > 0) return 1;

if (a\_len - b\_len < 0) return -1;

int i = a\_len;

while (A->value[i] == B->value[i] && i >= 0)

{

i--;

}

if (i == -1)

{

return 0;

}

else

{

if (A->value[i] > B->value[i]) return 1;

else return -1;

}

}

LongNumber \*mod = GetModule();

LongNumber\* ShiftBitsToHigh(LongNumber \*A, int index, bool selfculc)

{

unsigned int\* C = new unsigned int[A->size + index];

for (int i = A->size + index - 1; i >= index; i--)

{

C[i] = A->value[i - index];

}

for (int i = 0; i <= index - 1; i++)

{

C[i] = 0;

}

LongNumber \*B = GetValue(C, A->size + index);

if (selfculc)

{

delete A;

A = NULL;

return B;

}

return B;

}

unsigned int\* CopyArray(unsigned int arr[], int size)

{

unsigned int\* temp = new unsigned int[size];

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

{

temp[i] = arr[i];

}

return temp;

}

unsigned int\* LongShiftBitsToHigh(unsigned int A[], int size, int index, bool selfculc)

{

unsigned int\* C = new unsigned int[size + index];

for (int i = size + index - 1; i >= index; i--)

{

C[i] = A[i - index];

}

for (int i = 0; i <= index - 1; i++)

{

C[i] = 0;

}

if (selfculc)

{

delete[] A;

return C;

}

return C;

}

unsigned int\* AddArray(unsigned int A[], unsigned int B[], int size\_A, int size\_B, bool selfculc)

{

unsigned int\* C = new unsigned int[size\_A];

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

{

C[i] = (A[i] + B[i]) % 2;

}

for (int i = size\_B; i < size\_A; i++)

{

C[i] = A[i];

}

if (selfculc)

{

free(A);

return C;

}

return C;

}

// ----------------------- OPERATIONS ZONE -----------------------

LongNumber\* Addition(LongNumber \*A, LongNumber \*B, bool selfculc)

{

int big = max(A->size, B->size);

int small = min(A->size, B->size);

unsigned int\* C = new unsigned int[big];

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

{

C[i] = (A->value[i] + B->value[i]) % 2;

}

if (big == A->size)

{

for (int i = small; i < big; i++)

{

C[i] = A->value[i];

}

}

else

{

for (int i = small; i < big; i++)

{

C[i] = B->value[i];

}

}

LongNumber \*result = GetValue(C, big);

return result;

}

LongNumber\* TakeModule(LongNumber \*A, bool selfculc)

{

int k = GetFirstNotNullBit(mod->value, mod->size);

LongNumber \*R = CopyLongNumber(A);

while (R->size >= mod->size)

{

LongNumber \*C = new LongNumber;

int t = GetFirstNotNullBit(R->value, R->size);

C = ShiftBitsToHigh(mod, t - k, false);

if (LongCmp(R, C) == -1)

{

t = t - 1;

free(C);

C = ShiftBitsToHigh(mod, t - k, false);

}

R = Addition(R, C, true);

KillZeros(R);

free(C);

}

AddZeros(R, degree);

if (selfculc)

{

free(A);

return R;

}

return R;

}

LongNumber\* Multiplication(LongNumber \*A, LongNumber \*B, bool selfculc)

{

int sz = 2 \* max(A->size, B->size);

unsigned int\* C = new unsigned int[sz];

fill(&C[0], &C[sz], 0);

for (int i = 0; i < B->size; i++)

{

if (B->value[i] == 1)

{

unsigned int\* temp = CopyArray(A->value, A->size);

temp = LongShiftBitsToHigh(temp, B->size, i, true);

C = AddArray(C, temp, sz, B->size + i, true);

delete[] temp;

}

}

LongNumber \*result = GetValue(C, sz);

KillZeros(result);

result = TakeModule(result, true);

return result;

}

LongNumber\* Square(LongNumber \*A, bool selfculc)

{

unsigned int\* C = new unsigned int[2 \* A->size - 1];

for (int i = 0; i < A->size - 1; i++)

{

C[2 \* i] = A->value[i];

C[2 \* i + 1] = 0;

}

C[2 \* A->size - 2] = A->value[A->size - 1];

LongNumber \*result = GetValue(C, 2 \* A->size - 1);

KillZeros(result);

result = TakeModule(result, true);

if (selfculc)

{

free(A);

return result;

}

return result;

}

LongNumber\* LongPower(LongNumber \*A, LongNumber \*N, bool selfculc)

{

LongNumber \*C = GetOne();

LongNumber \*temp = CopyLongNumber(A);

for (int i = 0; i < N->size; i++)

{

if (N->value[i] == 1)

{

C = Multiplication(C, temp, true);

}

temp = Square(temp, true);

}

if (selfculc)

{

free(A);

return C;

}

return C;

}

LongNumber\* Trace(LongNumber \*A)

{

LongNumber \*temp = CopyLongNumber(A);

unsigned int \*C = CopyArray(A->value, A->size);

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

{

temp = Square(temp, true);

C = AddArray(C, temp->value, A->size, temp->size, true);

}

free(temp);

LongNumber \*result = GetValue(C, A->size);

return result;

}

LongNumber\* Reversed(LongNumber \*A)

{

LongNumber \*temp = Square(A, false);

LongNumber \*result = GetOne();

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

{

result = Multiplication(result, temp, true);

temp = Square(temp, true);

}

free(temp);

return result;

}

// ----------------------- TEST ZONE -----------------------

void Test1()

{

string s\_a;

cout << "Insert number a: " << endl;

cin >> s\_a;

string s\_b;

cout << "Insert number b: " << endl;

cin >> s\_b;

string s\_c;

cout << "Insert number c: " << endl;

cin >> s\_c;

LongNumber \*a = GetValue(s\_a);

LongNumber \*b = GetValue(s\_b);

LongNumber \*c = GetValue(s\_c);

LongNumber \*sum1 = Addition(a, b, false);

LongNumber \*leftResult = Multiplication(sum1, c, false);

LongNumber \*mul1 = Multiplication(b, c, false);

LongNumber \*mul2 = Multiplication(c, a, false);

LongNumber \*rightResult = Addition(mul1, mul2, false);

GetStringValue(leftResult);

GetStringValue(rightResult);

leftResult->PrintParameters();

rightResult->PrintParameters();

}

void Test2()

{

string s\_a;

cout << "Insert number a: " << endl;

cin >> s\_a;

LongNumber \*a = GetValue(s\_a);

unsigned int \*m = new unsigned int[degree];

fill(&m[0], &m[degree], 1);

LongNumber \*deg = GetValue(m, degree);

LongNumber \*result = LongPower(a, deg, false);

GetStringValue(result);

result->PrintParameters();

}

// ----------------------- MAIN ZONE -----------------------

int main()

{

//Test1();

//Test2();

cout << "Module: " << endl;

mod->PrintParameters();

auto start = std::chrono::system\_clock::now();

auto end = std::chrono::system\_clock::now();

std::chrono::duration<double> diff = end - start;

end = std::chrono::system\_clock::now();

diff = end - start;

string s\_A;

cout << "Insert number A: " << endl;

cin >> s\_A;

string s\_B;

cout << "Insert number B: " << endl;

cin >> s\_B;

string s\_N;

cout << "Insert number N: " << endl;

cin >> s\_N;

LongNumber \*A = GetValue(s\_A);

LongNumber \*B = GetValue(s\_B);

LongNumber \*N = GetValue(s\_N);

cout << endl << "A + B: " << endl;

start = std::chrono::system\_clock::now();

LongNumber \*C = Addition(A, B, false);

end = std::chrono::system\_clock::now();

diff = end - start;

GetStringValue(C);

C->PrintParameters();

cout << "Time: " << diff.count() << " sec" << endl;

cout << endl << "A \* B: " << endl;

start = std::chrono::system\_clock::now();

LongNumber \*D = Multiplication(A, B, false);

end = std::chrono::system\_clock::now();

diff = end - start;

GetStringValue(D);

D->PrintParameters();

cout << "Time: " << diff.count() << " sec" << endl;

cout << endl << "A ^ 2: " << endl;

start = std::chrono::system\_clock::now();

LongNumber \*E = Square(A, false);

end = std::chrono::system\_clock::now();

diff = end - start;

GetStringValue(E);

E->PrintParameters();

cout << "Time: " << diff.count() << " sec" << endl;

cout << endl << "A ^ N: " << endl;

start = std::chrono::system\_clock::now();

LongNumber \*F = LongPower(A, N, false);

end = std::chrono::system\_clock::now();

diff = end - start;

GetStringValue(F);

F->PrintParameters();

cout << "Time: " << diff.count() << " sec" << endl;

cout << endl << "Trace A: " << endl;

start = std::chrono::system\_clock::now();

LongNumber \*G = Trace(A);

end = std::chrono::system\_clock::now();

diff = end - start;

GetStringValue(G);

G->PrintParameters();

cout << "Time: " << diff.count() << " sec" << endl;

cout << endl << "A ^ -1: " << endl;

start = std::chrono::system\_clock::now();

LongNumber \*H = Reversed(A);

end = std::chrono::system\_clock::now();

diff = end - start;

GetStringValue(H);

H->PrintParameters();

cout << "Time: " << diff.count() << " sec" << endl;

}