МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**Национальный исследовательский ядерный университет «МИФИ»**

**Отчёт к проекту**

**на тему «ГПСЧ в поле Галуа »**

**Выполнили:**

Студенты группы Б18-503

Чуркин К.Ю.

Григорьев М.П.

Хисамутдинов М.А.

**Преподаватель:**

Иванов Михаил Александрович

**Москва 2021**

**Оглавление**

[1 Введение в поля Галуа 3](#_Toc72529941)

[1.1 Общее понятие поля Галуа 3](#_Toc72529942)

[1.2 Арифметика в поле Галуа 3](#_Toc72529943)

[1.2.1 Операция сложения 3](#_Toc72529944)

[1.2.2 Операция вычитания 4](#_Toc72529945)

[1.2.3 Операция умножения 4](#_Toc72529946)

[2 Программная реализация поля Галуа на языке С++ 5](#_Toc72529947)

[2.1 Особенности размещения кода 5](#_Toc72529948)

[2.2 Модульная арифметика 5](#_Toc72529949)

[2.2.1 Сложение 5](#_Toc72529950)

[2.2.2 Вычитание 5](#_Toc72529951)

[2.2.3 Умножение 6](#_Toc72529952)

[2.2.4 Инверсия 6](#_Toc72529953)

[2.2.5 Деление 7](#_Toc72529954)

[2.3 Арифметика полиномов 7](#_Toc72529955)

[2.3.1 Представление полиномов 7](#_Toc72529956)

[2.3.2 Немного про “контракты” 8](#_Toc72529957)

[2.3.3 Сложение 8](#_Toc72529958)

[2.3.4 Вычитание 9](#_Toc72529959)

[2.3.5 Умножение 10](#_Toc72529960)

[2.3.6 Определение степени полинома 11](#_Toc72529961)

[2.3.7 Деление 12](#_Toc72529962)

[2.4 Поле Галуа и его элементы 13](#_Toc72529963)

[2.4.1 Элементы поля 15](#_Toc72529964)

[2.4.2 Поле Галуа 20](#_Toc72529965)

[3 ГПСЧ в поле Галуа 22](#_Toc72529966)

[4 Реализация ГПСЧ на С++ 23](#_Toc72529967)

[5 Графический интерфейс для ГПСЧ 30](#_Toc72529968)

[5.1 Приложение и его описание 30](#_Toc72529969)

[5.2 Реализация на С++ 32](#_Toc72529970)

[6 Тесты 41](#_Toc72529971)

# Введение в поля Галуа

## Общее понятие поля Галуа

Поле Галуа — это поле из конечного числа элементов. Поле Галуа обычно обозначают , где – число элементов поля, это число называется ***порядком*** поля. Так же установлено, что , где – простое число (***характеристика*** поля), а – любое натуральное число (***степень*** поля). Таким образом получаем ещё один вариант записи поля – .

Элементами поля являются полиномы вида:

Так же существует так называемый ***примитивный*** (***порождающий***) полином поля вида:

Все элементы поля являются степенями примитивного полинома:

## Арифметика в поле Галуа

В конечном поле определены операции сложения, вычитания, умножения и деления.

Так как для создания ГПСЧ операция деления элементов поля не используется, то мы не будем её рассматривать.

### Операция сложения

Операция сложения полиномов выполняется с каждым коэффициентом отдельно.

Исходные элементы поля:

По правилу сложения:

Получаем:

### Операция вычитания

Операция вычитания полиномов так же выполняется с каждым коэффициентом отдельно.

Исходные элементы поля:

По правилу вычитания:

Получаем:

### Операция умножения

Исходные элементы поля:

Перемножив их, а так же приведя соответствующие слагаемые и получим новый полином :

В зависимости от коэффициентов степень существует две ситуации:

1. Степень полинома меньше полученный полином принадлежит
2. Степень полинома больше, либо равна полученный полином не принадлежит

# Программная реализация поля Галуа на языке С++

## Особенности размещения кода

В файлах с расширением .h хранятся так называемые определения *типов* и *функций*. В файлах с расширением .cpp хранятся реализации этих *типов* и *функций*.

## Модульная арифметика

Модульная арифметика является основой всех последующих операций с полиномами в поле Галуа, поэтому нужно начать именно с её реализации.

Для начала обозначим целочисленный *тип*, с которым мы будем работать. Это будет целое число, которое занимает 64 бита.

Файл ModularArithmetic.h:

using Int = long long int;

### Сложение

Функция модульного сложения принимает три аргумента: левый операнд, правый операнд и сам модуль соответственно. Возвращает результат модульного сложения.

Файл ModularArithmetic.h:

Int modular\_add(const Int left, const Int right, const Int mod);

Файл ModularArithmetic.cpp:

Int modular\_add(const Int left, const Int right, const Int mod)

{

return (left + right) % mod;

}

### Вычитание

Функция модульного вычитания принимает три аргумента: левый операнд, правый операнд и модуль соответственно. Возвращает результат модульного вычитания.

Файл ModularArithmetic.h:

Int modular\_sub(const Int left, const Int right, const Int mod);

Файл ModularArithmetic.cpp:

Int modular\_sub(const Int left, const Int right, const Int mod)

{

return (left + (mod - right)) % mod;

}

### Умножение

Функция модульного умножения принимает три аргумента: левый операнд, правый операнд и модуль соответственно. Возвращает результат модульного умножения.

Файл ModularArithmetic.h:

Int modular\_mul(const Int left, const Int right, const Int mod);

Файл ModularArithmetic.cpp:

Int modular\_mul(const Int left, const Int right, const Int mod)

{

return (left \* right) % mod;

}

### Инверсия

Функция модульной инверсии принимает два аргумента: операнд и модуль соответственно. Возвращает результат модульной инверсии. Так же функция использует функцию быстрого возведения в степень.

Файл ModularArithmetic.h:

Int fast\_power(Int base, Int power, Int mod);

Int modular\_inv(const Int left, const Int mod);

Файл ModularArithmetic.cpp:

Int fast\_power(Int base, Int power, Int mod) {

Int result = 1;

while (power > 0) {

if ((power & 1) == 1) {

result = (result \* base) % mod;

}

base = (base \* base) % mod;

power = power >> 1;

}

return result;

}

Int modular\_inv(const Int left, const Int mod)

{

return fast\_power(left, mod - 2, mod);

}

### Деление

Функция модульного деления использует мольную инверсию числа и выполняется как модульное умножение левого операнда и инверсии правого.

Принимает три аргумента: левый операнд, правый операнд и модуль соответственно. Возвращает результат модульного деления. Если правый операнд(делитель) равен 0, то функция “кидает” соответствующее *исключение*.

Файл ModularArithmetic.h:

Int modular\_mul(const Int left, const Int right, const Int mod);

Файл ModularArithmetic.cpp:

Int modular\_mul(const Int left, const Int right, const Int mod)

{

if (right == 0) {

throw std::exception("divide by zero");

}

return (left \* modular\_inv(right, mod)) % mod;

}

## Арифметика полиномов

Арифметика полиномов является промежуточной ступенью между арифметикой элементов поля и модульной арифметикой, поэтому следующей стадией станет реализация функций для полиномов.

### Представление полиномов

Так как в полиномах обозначает лишь позицию, то можно хранить только коэффициенты полиномов. При этом, если в полиноме нет какого-то , то будет соответственно указан коэффициент равный 0.

Теперь нужно установить тип, которым будут представлены наборы коэффициентов. Это будет std::vector<Int>. Этот тип является реализацией концепции динамического массива (***вектора***) и находится в Стандартной Библиотеке Шаблонов (STL) языка С++.

Файл PolynomialArithmetic.h:

using CoefficientsVector = std::vector<Int>;

### Немного про “контракты”

Существует практика установления определённых “контрактов” для функций. Контракты обозначаются лишь на словах и устанавливают определённые требования для входных данных

Воспользуемся этим и скажем, что два входных вектора коэффициентов обязаны быть одинакового размера. Это очень важное уточнение, так как оно позволяет существенно упростить функции и отказаться от специфичных реализаций. Естественно, перед вызовом функции нужно убедиться, что данные соответствуют контракту.

### Сложение

Сложение полиномов производится с помощью модульного сложения коэффициентов у соответствующего .

Функция сложения полиномов принимает три аргумента: вектор коэффициентов первого (левого) операнда, вектор коэффициентов первого (правого) операнда и модуль. Возвращает вектор коэффициентов результирующего полинома.

Файл PolynomialArithmetic.h:

CoefficientsVector polynomial\_add(

const CoefficientsVector& left,

const CoefficientsVector& right,

const Int mod);

Файл PolynomialArithmetic.cpp:

CoefficientsVector polynomial\_add(

const CoefficientsVector& left,

const CoefficientsVector& right,

const Int mod

)

{

auto size = left.size();

if (size != right.size()) {

throw std::exception(

"dimensions of the polynomials are not equal");

}

CoefficientsVector result(size);

for (size\_t i = 0; i < size; ++i) {

result[i] = modular\_add(left[i], right[i], mod);

}

return result;

}

### Вычитание

Вычитание полиномов производится с помощью модульного вычитания коэффициентов у соответствующего .

Функция вычитания полиномов принимает три аргумента: вектор коэффициентов первого (левого) операнда, вектор коэффициентов первого (правого) операнда и модуль. Возвращает вектор коэффициентов результирующего полинома.

Файл PolynomialArithmetic.h:

CoefficientsVector polynomial\_sub(

const CoefficientsVector& left,

const CoefficientsVector& right,

const Int mod);

Файл PolynomialArithmetic.cpp:

CoefficientsVector polynomial\_sub(

const CoefficientsVector& left,

const CoefficientsVector& right,

const Int mod

)

{

auto size = left.size();

if (size != right.size()) {

throw std::exception(

"dimensions of the polynomials are not equal");

}

CoefficientsVector result(size);

for (size\_t i = 0; i < size; ++i) {

result[i] = modular\_sub(left[i], right[i], mod);

}

return result;

}

### Умножение

Умножение полиномов производится обычным раскрытием скобок с помощью модульного умножения коэффициентов.

Функция умножения полиномов принимает три аргумента: вектор коэффициентов первого (левого) операнда, вектор коэффициентов первого (правого) операнда и модуль. Возвращает вектор коэффициентов двойной длины результирующего полинома.

Файл PolynomialArithmetic.h:

CoefficientsVector polynomial\_mul(

const CoefficientsVector& left,

const CoefficientsVector& right,

const Int mod);

Файл PolynomialArithmetic.cpp:

CoefficientsVector polynomial\_mul(

const CoefficientsVector& left,

const CoefficientsVector& right,

const Int mod

)

{

CoefficientsVector result(left.size() + right.size(), 0);

for (size\_t i = 0; i < result.size(); ++i) {

for (size\_t j = 0; j <= i; ++j) {

if (i - j < right.size() && j < left.size()) {

decltype(auto) temp =

modular\_mul(

left[j],

right[i - j],

mod

);

result[i] = modular\_add(result[i], temp, mod);

}

}

}

return result;

}

### Определение степени полинома

Определение степени полинома является довольно полезной функцией при установленном ранее представлении полиномов. Эта функция требуется для алгоритмов, в которых нужно знать степень полинома именно на данный момент, например, в делении.

Функция принимает один аргумент – вектор коэффициентов.

Возвращает индекс первого старшего коэффициента не равного 0.   
Если все коэффициенты равны 0, то возвращает -1.

Файл ModularArithmetic.h:

Int polynomial\_degree(

const CoefficientsVector& polynomial\_coefficients

);

Файл ModularArithmetic.cpp:

Int polynomial\_degree(

const CoefficientsVector& polynomial\_coefficients

)

{

for (Int i = polynomial\_coefficients.size() - 1; i >= 0; --i) {

if (polynomial\_coefficients[i]) return i;

}

return -1;

}

### Деление

Деление полиномов производится обычным делением столбиком с использованием модульной арифметики.

Функция деления полиномов принимает три аргумента: вектор коэффициентов первого (левого) операнда, вектор коэффициентов первого (правого) операнда и модуль. Возвращает пару векторов-коэффициентов из частного и остатка соответственно.

Файл PolynomialArithmetic.h:

std::pair<CoefficientsVector, CoefficientsVector> polynomial\_div(

const CoefficientsVector& left,

const CoefficientsVector& right,

const Int mod);

Файл PolynomialArithmetic.cpp:

std::pair<CoefficientsVector, CoefficientsVector> polynomial\_div(

const CoefficientsVector& left,

const CoefficientsVector& right,

const Int mod

)

{

decltype(auto) right\_degree = polynomial\_degree(right);

if (right\_degree == -1) throw std::exception("divide by zero");

CoefficientsVector quotient(left.size(),0);

CoefficientsVector remainder = left;

CoefficientsVector qRight(left.size(), 0);

decltype(auto) remainder\_degree = polynomial\_degree(remainder);

while (remainder\_degree >= right\_degree) {

decltype(auto) q = modular\_div(

remainder[remainder\_degree],

right[right\_degree],

mod);

decltype(auto) diff = remainder\_degree - right\_degree;

quotient[diff] = q;

for (Int i = right\_degree; i >= 0; --i)

qRight[i + diff] = modular\_mul(right[i], q, mod);

for (Int i = 0; i < diff; ++i)

qRight[i] = 0;

for (Int i = 0; i <= remainder\_degree; ++i) {

remainder[i] = modular\_sub(

remainder[i], qRight[i], mod);

remainder\_degree = polynomial\_degree(remainder);

}

return { quotient, remainder };

}

## Поле Галуа и его элементы

Поле описывает *класс* (*тип*) GaloisField, элемент поля – GFElement.

Функции, принадлежащие классу – *методы класса*.

Данные, принадлежащие классу – *поля класса*.

Так как эти два класса имеют сильную внутреннюю связь, то сперва нужно определить оба класса, и только после переходить к их реализации.

Файл GFElement.h:

class GFElement {

private:

CoefficientsVector \_coefficients;

GaloisField& \_field;

private:

void field\_assert(const GFElement& other) const;

public:

GFElement operator+(const GFElement& other) const;

GFElement operator-(const GFElement& other) const;

GFElement operator\*(const GFElement& other) const;

bool operator == (const GFElement& other) const noexcept;

bool operator != (const GFElement& other) const noexcept;

bool operator < (const GFElement& other) const noexcept;

bool operator > (const GFElement& other) const noexcept;

bool operator <= (const GFElement& other) const noexcept;

bool operator >= (const GFElement& other) const noexcept;

public:

GFElement mod\_add(const GFElement& other) const;

public:

GFElement() = delete;

GFElement(const GFElement& other);

GFElement(GFElement&& other) noexcept;

GFElement(const CoefficientsVector& coefficients,

GaloisField& field);

GFElement(CoefficientsVector&& coefficients, GaloisField& field);

GFElement(const GFElement& left, const GFElement& right,

decltype(polynomial\_add) polynomial\_operator);

GFElement& operator=(const GFElement& other);

GFElement& operator=(GFElement&& other) noexcept;

public:

friend std::ostream& operator << (std::ostream& out,

const GFElement& element);

public:

Int to\_int() const;

};

Элемент поля определяют его коэффициенты (*поле класса* \_coefficients) и поле (*поле класса* \_field), к которому оно принадлежит.

Файл GaloisField.h:

class GaloisField {

private:

Int \_characteristic;

Int \_dimension;

CoefficientsVector \_reduction\_polynomial;

public:

GaloisField(Int characteristic, Int dimension,

CoefficientsVector reduction\_polynomial);

public:

Int characteristic() const noexcept;

Int dimension() const noexcept;

const CoefficientsVector& reduction\_polynomial() const noexcept;

GFElement create\_polynomial(

const CoefficientsVector& coefficients);

GFElement create\_polynomial(CoefficientsVector&& coefficients);

std::vector<GFElement> all\_field\_elements();

public:П

bool operator == (const GaloisField& other) const noexcept;

bool operator != (const GaloisField& other) const noexcept;

};

Поле определяет его ***характеристика*** (*поле класса* \_characteristic), ***степень*** поля (*поле класса* \_dimension) и ***порождающий*** полином (*поле класса* \_reduction\_polynomial).

Здесь поле является своеобразной меткой для элементов поля, имеют ли они право производить арифметические операции друг с другом или нет. В то время как основное взаимодействие берут на себя элементы поля.

### Элементы поля

Конструкторы:

GFElement::GFElement(const GFElement& other) :

\_coefficients(other.\_coefficients),

\_field(other.\_field)

{}

GFElement::GFElement(GFElement&& other) noexcept :

\_coefficients(std::move(other.\_coefficients)),

\_field(other.\_field)

{}

GFElement::GFElement(

const CoefficientsVector& coefficients,

GaloisField& field

) :

\_coefficients(coefficients),

\_field(field)

{}

GFElement::GFElement(

CoefficientsVector&& coefficients,

GaloisField& field

) :

\_coefficients(coefficients),

\_field(field)

{}

GFElement::GFElement(

const GFElement& left,

const GFElement& right,

decltype(polynomial\_add) polynomial\_operator

) :

\_field(left.\_field),

\_coefficients(polynomial\_operator(

left.\_coefficients,

right.\_coefficients,

left.\_field.characteristic()))

{}

Нужны для удобного создания объектов.

Операторы присваивания:

GFElement& GFElement::operator=(const GFElement& other)

{

if (this == &other) return \*this;

\_field = other.\_field;

\_coefficients = other.\_coefficients;

return \*this;

}

GFElement& GFElement::operator=(GFElement&& other) noexcept

{

if (this == &other) return \*this;

\_field = other.\_field;

\_coefficients = std::move(other.\_coefficients);

return \*this;

}

Функция field\_assert(…) проверяет совпадают ли поля. Она используется во всех арифметических и логических операциях.

void GFElement::field\_assert(const GFElement& other) const

{

if (\_field != other.\_field)

throw std::exception("the fields are not compatible");

}

Перегруженные операторы сложения, вычитания и умножения используют рассмотренные выше арифметические функции для работы с полиномами.

Сложение - polynomial\_add(…)

GFElement GFElement::operator+(const GFElement& other) const

{

field\_assert(other);

return GFElement(\*this, other, polynomial\_add);

}

Вычитание - polynomial\_sub(…)

GFElement GFElement::operator-(const GFElement& other) const

{

field\_assert(other);

return GFElement(\*this, other, polynomial\_sub);

}

Умножение – сначала polynomial\_mul(…), после чего берётся остаток от деления на порождающий полином, получаемый через polynomial\_div(…).

GFElement GFElement::operator\*(const GFElement& other) const

{

field\_assert(other);

decltype(auto) product = polynomial\_mul(

\_coefficients,

other.\_coefficients,

\_field.characteristic());

decltype(auto) reduction\_polynomial =

\_field.reduction\_polynomial();

decltype(auto) reminder = polynomial\_div(

product,

reduction\_polynomial,

\_field.characteristic()).second;

return GFElement(CoefficientsVector(

reminder.begin(),

reminder.begin() + \_field.dimension()),

\_field);

}

Операторы сравнения используют последовательное сравнение, начиная со старших разрядов.

bool GFElement::operator==(const GFElement& other) const noexcept

{

field\_assert(other);

for (size\_t i = 0; i < \_coefficients.size(); ++i) {

if (\_coefficients[i] != other.\_coefficients[i])

return false;

}

return true;

}

bool GFElement::operator!=(const GFElement& other) const noexcept

{

return !(\*this == other);

}

bool GFElement::operator<(const GFElement& other) const noexcept

{

field\_assert(other);

for (Int i = \_coefficients.size() - 1; i >= 0; --i) {

if (\_coefficients[i] < other.\_coefficients[i]) return true;

if (\_coefficients[i] > other.\_coefficients[i])

return false;

}

return false;

}

bool GFElement::operator>(const GFElement& other) const noexcept

{

field\_assert(other);

for (size\_t i = \_coefficients.size() - 1; i > 0; --i) {

if (\_coefficients[i] > other.\_coefficients[i]) return true;

if (\_coefficients[i] < other.\_coefficients[i])

return false;

}

return false;

}

bool GFElement::operator<=(const GFElement& other) const noexcept

{

return !(\*this > other);

}

bool GFElement::operator>=(const GFElement& other) const noexcept

{

return !(\*this < other);

}

Оператор “<<” используется для вывода информации о элементе поля.

std::ostream& operator<<(std::ostream& out, const GFElement& element)

{

decltype(auto) coefficients = element.\_coefficients;

decltype(auto) coefficients\_it = coefficients.rbegin();

out << "(";

out << \*coefficients\_it++;

for (; coefficients\_it != coefficients.rend(); ++coefficients\_it)

{

out << "," << \*coefficients\_it;

}

out << ")";

out << "[" << element.to\_int() << "]";

return out;

}

Функция to\_int() используется для перевода поразрядного представления числа в десятичное.

Int GFElement::to\_int() const

{

Int result = 0;

Int two\_powers = 1;

for (size\_t i = 0; i < \_coefficients.size(); ++i) {

result += (\_coefficients[i] \* two\_powers);

two\_powers \*= 2;

}

return result;

}

Особого внимания заслуживает функция mod\_add(…), так как она используется для поразрядного модульного сложения с модулем , а не сложения в поле и нужна для ГПСЧ.

GFElement GFElement::mod\_add(const GFElement& other) const

{

field\_assert(other);

CoefficientsVector new\_coefficients(\_coefficients);

Int carry = 0;

for (size\_t i = 0; i < new\_coefficients.size(); ++i) {

auto sum = new\_coefficients[i] +

other.\_coefficients[i] + carry;

new\_coefficients[i] = sum % \_field.characteristic();

carry = sum / \_field.characteristic();

}

return GFElement(std::move(new\_coefficients), \_field);

}

### Поле Галуа

Как и говорилось ничего особенного здесь нет.

Конструктор:

GaloisField::GaloisField(

Int characteristic,

Int dimension,

CoefficientsVector reduction\_polynomial

) :

\_characteristic(characteristic),

\_dimension(dimension),

\_reduction\_polynomial(reduction\_polynomial)

{}

Функции characteristic(), dimension() и reduction\_polynomial() используются для получения соответствующих параметров поля.

Int GaloisField::characteristic() const noexcept

{

return \_characteristic;

}

Int GaloisField::dimension() const noexcept

{

return \_dimension;

}

const CoefficientsVector& GaloisField::reduction\_polynomial() const

noexcept

{

return \_reduction\_polynomial;

}

Функция create\_polynomial(…) нужна для создания элементов поля.

GFElement GaloisField::create\_polynomial(

const CoefficientsVector& coefficients)

{

return GFElement(coefficients, \*this);

}

GFElement GaloisField::create\_polynomial(

CoefficientsVector&& coefficients)

{

return GFElement(coefficients, \*this);

}

Перегруженные операторы сравнения используются для проверки, являются ли два объекта одним и тем же полем.

bool GaloisField::operator==(const GaloisField& other) const noexcept

{

return (\_characteristic == other.\_characteristic)

&& (\_dimension == other.\_dimension);

}

bool GaloisField::operator!=(const GaloisField& other) const noexcept

{

return !(\*this == other);

}

Функция all\_field\_elements() возвращает вектор с набором всех элементов, принадлежащих полю.

std::vector<GFElement> GaloisField::all\_field\_elements()

{

std::vector<GFElement> elements;

CoefficientsVector coefficients(\_dimension, 0);

CoefficientsVector max\_coefficients(\_dimension,

\_characteristic - 1);

while (coefficients != max\_coefficients)

{

elements.emplace\_back(coefficients, \*this);

coefficients[0] += 1;

for (size\_t i = 0; i < coefficients.size() - 1; ++i) {

Int over\_mod = coefficients[i] / \_characteristic;

coefficients[i] = coefficients[i] % \_characteristic;

coefficients[i + 1] += over\_mod;

}

}

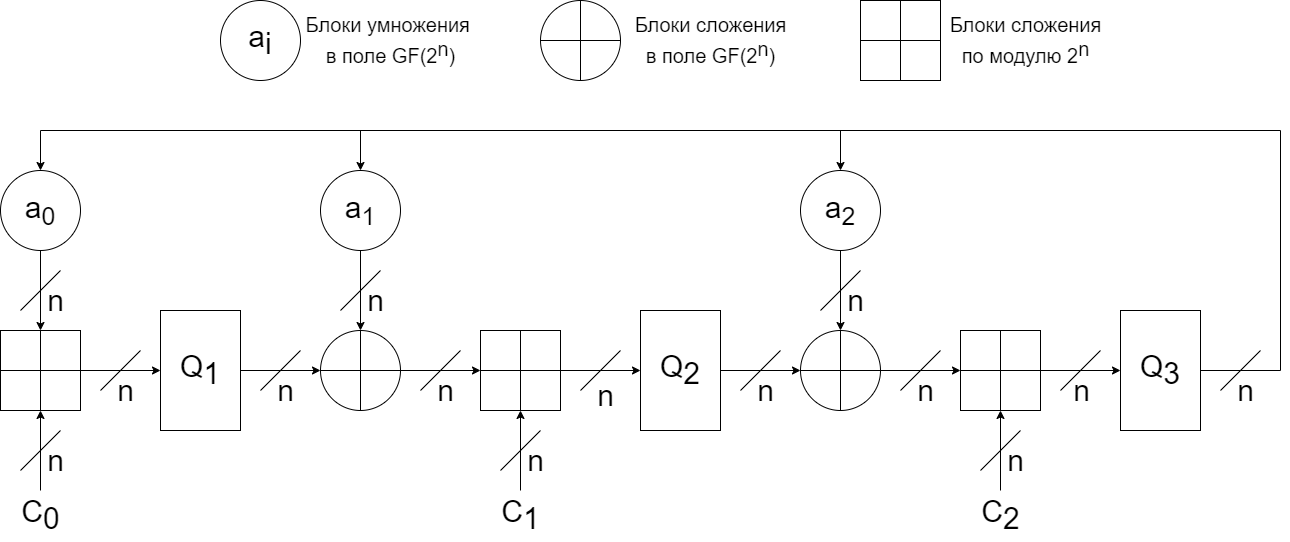
elements.emplace\_back(coefficients, \*this);

return elements;

}

# ГПСЧ в поле Галуа

Генератор состоит из регистров, которые хранят значения для текущего состояния генератора, блоков умножения в поле , блоков сложения в поле и блоков сложения по модулю . Схема генератора для N = 3 приведена на рисунке 3.1.



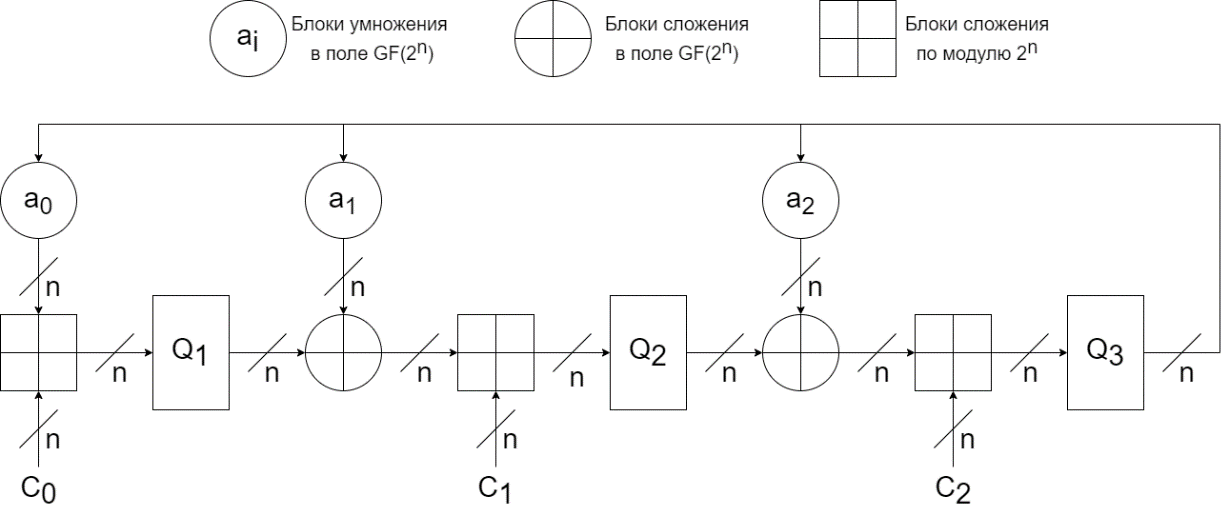


Рис 3.1 – схема ГПСЧ в поле

В общем случае значения образуют вектор обратный связей , а значения образуют вектор управляющих сигналов .

# Реализация ГПСЧ на С++

class Generator {

private:

GaloisField \_field;

std::vector<GFElement> \_even\_A;

std::vector<GFElement> \_even\_C;

std::vector<GFElement> \_odd\_A;

std::vector<GFElement> \_odd\_C;

std::vector<GFElement> \_Q;

std::vector<GFElement> \_start\_Q;

Int \_N;

bool \_even\_iteration = false;

private:

void reset();

void even\_iteration(std::vector<GFElement>& new\_Q);

void odd\_iteration(std::vector<GFElement>& new\_Q);

void check\_endless\_loop(std::vector<GFElement>& Q);

public:

Generator(GaloisField& field, Int N,

std::vector<GFElement> A,

std::vector<GFElement> C,

std::vector<GFElement> star\_Q = std::vector<GFElement>());

public:

std::vector<GFElement> generate();

void print\_state(std::ostream& out = std::cout);

void print\_all\_current\_cycle\_states(

std::ostream& out = std::cout);

std::vector<std::vector<std::vector<GFElement>>>

generate\_all\_cycles();

void print\_all\_cycles(std::ostream& out = std::cout,

bool show\_states = false);

public:

static GFElement field\_add(const GFElement& left,

const GFElement& right);

static GFElement mod\_add(const GFElement& left,

const GFElement& right);

void set\_even\_iteration\_add(decltype(&mod\_add) add);

void set\_odd\_iteration\_add(decltype(&mod\_add) add);

private:

decltype(&mod\_add) \_even\_iteration\_add = mod\_add;

decltype(&mod\_add) \_odd\_iteration\_add = mod\_add;

public:

void set\_even\_A(std::vector<GFElement> A);

void set\_even\_C(std::vector<GFElement> C);

void set\_odd\_A(std::vector<GFElement> A);

void set\_odd\_C(std::vector<GFElement> C);

};

Описание полей генератора:

\_field – поле Галуа, в котором работает ГПСЧ

\_even\_A – вектор обратных связей для чётных итераций

\_even\_C – вектор управляющих сигналов для чётных итераций

\_odd\_A – вектор обратных связей для нечётных итераций

\_odd\_C – вектор управляющих сигналов для нечётных итераций

\_Q – вектор значений генератора

\_start\_Q – вектор начальных значений генератора, по умолчанию состоит из нулей

\_N – количество генерируемых значений

\_even\_iteration – флаг чётной\нечётной итерации

\_even\_iteration\_add – блок сложения для чётного цикла

\_odd\_iteration\_add – блок сложения для нечётного цикла

Конструктор:

Generator::Generator(GaloisField& field, Int N,

std::vector<GFElement> A,

std::vector<GFElement> C,

std::vector<GFElement> star\_Q

) :

\_field(field),

\_N(N),

\_even\_A(A), \_odd\_A(A),

\_even\_C(C), \_odd\_C(C)

{

if (A.size() != N || C.size() != N) {

throw std::exception("dimensions of the model do not match

each other");

}

if (star\_Q.size() == N) \_start\_Q = star\_Q;

else \_start\_Q = std::vector(N, GFElement(CoefficientsVector(field.dimension(), 0), field));

reset();

}

Функция reset() используется для сброса генератора в стартовое состояние.

void Generator::reset()

{

\_Q = \_start\_Q;

\_even\_iteration = false;

}

При инициализации генератора, векторы обратных связей и управляющих сигналов имеют одинаковые значения для чётных и нечётных итераций. Для их изменения используются функции:

void Generator::set\_even\_A(std::vector<GFElement> A)

{

if (A.size() != \_N) {

throw std::exception("dimensions of the vectors do not match each other");

}

\_even\_A = A;

}

void Generator::set\_even\_C(std::vector<GFElement> C)

{

if (C.size() != \_N) {

throw std::exception("dimensions of the vectors do not match each other");

}

\_even\_C = C;

}

void Generator::set\_odd\_A(std::vector<GFElement> A)

{

if (A.size() != \_N) {

throw std::exception("dimensions of the vectors do not match each other");

}

\_odd\_A = A;

}

void Generator::set\_odd\_C(std::vector<GFElement> C)

{

if (C.size() != \_N) {

throw std::exception("dimensions of the vectors do not match each other");

}

\_odd\_C = C;

}

А блоки сложения являются блоками сложения по модулю . Для изменения блоков сложения для чётных и нечётных итераций существуют функции:

void Generator::set\_even\_iteration\_add(decltype(&mod\_add) add)

{

\_even\_iteration\_add = add;

}

void Generator::set\_odd\_iteration\_add(decltype(&mod\_add) add)

{

\_odd\_iteration\_add = add;

}

Сами блоки сложения есть двух видов: по модулю и в поле Галуа. Их описывают следующие функции:

GFElement Generator::field\_add(

const GFElement& left,

const GFElement& right)

{

return left + right;

}

GFElement Generator::mod\_add(

const GFElement& left,

const GFElement& right)

{

return left.mod\_add(right);

}

Функция generate() совершает одну итерацию генератора и возвращает вектор новых значений.

std::vector<GFElement> Generator::generate()

{

std::vector<GFElement> new\_Q;

if (\_even\_iteration) even\_iteration(new\_Q);

else odd\_iteration(new\_Q);

\_even\_iteration = !\_even\_iteration;

\_Q = std::move(new\_Q);

return \_Q;

}

Эта функция использует even\_iteration(...) и odd\_iteration(…) для генерации новых значений, основываясь на предыдущем состоянии и чётности итерации.

void Generator::even\_iteration(std::vector<GFElement>& new\_Q)

{

new\_Q.emplace\_back(\_even\_iteration\_add(

\_even\_A[0] \* \_Q[\_Q.size() - 1],

\_even\_C[0]));

for (size\_t i = 1; i < \_Q.size(); ++i) {

new\_Q.emplace\_back(\_even\_iteration\_add(

\_even\_A[i] \* \_Q[\_Q.size() - 1] + \_Q[i - 1],

\_even\_C[i]));

}

}

void Generator::odd\_iteration(std::vector<GFElement>& new\_Q)

{

new\_Q.emplace\_back(\_odd\_iteration\_add(

\_odd\_A[0] \* \_Q[\_Q.size() - 1],

\_odd\_C[0]));

for (size\_t i = 1; i < \_Q.size(); ++i) {

new\_Q.emplace\_back(\_odd\_iteration\_add(

\_odd\_A[i] \* \_Q[\_Q.size() - 1] + \_Q[i - 1],

\_odd\_C[i]));

}

}

Функция generate\_all\_cycles() используется для получения вектора из всех возможных циклов генератора.

std::vector<std::vector<std::vector<GFElement>>> Generator::generate\_all\_cycles()

{

decltype(auto) prev\_Q = \_Q;

decltype(auto) prev\_even\_flag = \_even\_iteration;

decltype(auto) all\_field\_polynomials =

\_field.all\_field\_elements();

std::set<std::vector<GFElement>>

all\_polynomials\_combinations\_set;

{

auto combination = std::vector<GFElement>(

\_N, all\_field\_polynomials.front());

auto max\_combination = std::vector<GFElement>(

\_N, all\_field\_polynomials.back());

while (combination != max\_combination) {

all\_polynomials\_combinations\_set.insert(combination);

for (auto&& q : combination) {

if (q == all\_field\_polynomials.back())

q = all\_field\_polynomials.front();

else {

q = all\_field\_polynomials[q.to\_int() + 1];

break;

}

}

}

all\_polynomials\_combinations\_set.insert(combination);

}

std::vector<std::vector<std::vector<GFElement>>> all\_cycles;

while (!all\_polynomials\_combinations\_set.empty()) {

auto start\_Q = \*all\_polynomials\_combinations\_set.begin();

\_Q = start\_Q;

\_even\_iteration = false;

all\_cycles.emplace\_back(

std::vector<std::vector<GFElement>>());

decltype(auto) current\_cycle = all\_cycles.back();

current\_cycle.emplace\_back(\_Q);

all\_polynomials\_combinations\_set.erase(\_Q);

while (generate() != start\_Q) {

check\_endless\_loop(\_Q);

current\_cycle.emplace\_back(\_Q);

all\_polynomials\_combinations\_set.erase(\_Q);

}

}

\_Q = prev\_Q;

\_even\_iteration = prev\_even\_flag;

return all\_cycles;

}

Функция check\_endless\_loop(…) проверяет, является ли последовательность текущих итераций бесконечным циклом.

void Generator::check\_endless\_loop(std::vector<GFElement>& Q)

{

static int repeat = 0;

static const int max\_repeat = 10;

static std::vector<GFElement> prev\_Q;

if (Q == prev\_Q) ++repeat;

else repeat = 0;

if (repeat == max\_repeat) throw std::exception("endless loop");

prev\_Q = Q;

}

Остались только функции вывода.

Функция print\_state(…) используется для вывода текущего состояния.

void Generator::print\_state(std::ostream& out)

{

for (auto&& q : \_Q) {

out << q.to\_int() << " ";

}

out << std::endl;

}

Функция print\_all\_current\_cycle\_states(..) используется для вывода всех состояний в текущем цикле.

void Generator::print\_all\_current\_cycle\_states(std::ostream& out)

{

decltype(auto) prev\_Q = \_Q;

decltype(auto) prev\_even\_flag = \_even\_iteration;

reset();

print\_state(out);

while (generate() != \_start\_Q) {

print\_state(out);

}

\_Q = prev\_Q;

\_even\_iteration = prev\_even\_flag;

}

Функция print\_all\_cycles(…) используется для вывода всех возможных циклов генератора. Используя флаг show\_states можно вывести состояния в каждом цикле.

void Generator::print\_all\_cycles(std::ostream& out, bool show\_states)

{

out << "n = " << \_field.dimension() << ", N = "

<< \_N << std::endl;

auto print\_ch = [&out](

std::string name,

const std::vector<GFElement>& elements)

{

out << name << " -> { ";

auto it = elements.begin();

out << (\*it++).to\_int();

for (; it != elements.end(); ++it) {

out << "," << (\*it).to\_int();

}

out << " }" << std::endl;

};

print\_ch("even A", \_even\_A);

print\_ch(" odd A", \_odd\_A);

print\_ch("even C", \_even\_C);

print\_ch(" odd C", \_odd\_C);

out << std::endl;

decltype(auto) all\_cycles = generate\_all\_cycles();

Int cycle\_counter = 0;

for (auto&& cycle : all\_cycles) {

++cycle\_counter;

out << "Cycle -> " << cycle\_counter << std::endl;

out << "States count -> " << cycle.size() << std::endl;

if (show\_states) {

for (auto&& combination : cycle) {

for (auto&& q : combination) {

out << q.to\_int() << " ";

}

out << std::endl;

}

}

out << std::endl;

}

}

# Графический интерфейс для ГПСЧ

## Приложение и его описание

Приложение для ГПСЧ было разработано при помощи библиотеки для графических приложений Qt. Интерфейс приведён на рисунке 5.1.1.

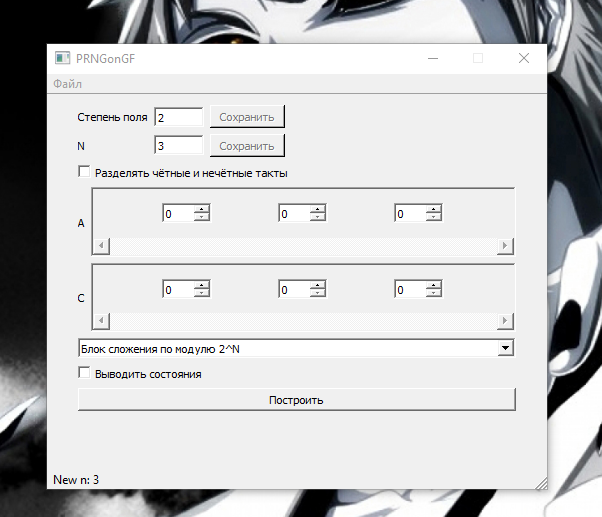


Рис 5.1.1 – графический интерфейс приложения для ГПСЧ

Поле “Степень поля” используется для задания степени для .

Поле “N” используется для задания количества генерируемых значений в одной итерации генератора.

Флаг “Разделять чётные и нечётные такты” нужен, чтобы использовать различные значения вектора обратных значений и управляющих сигналов для чётных и нечётных итераций генератора. При включении флага добавляются дополнительные поля для ВОЗ и УС (рис. 5.1.2).

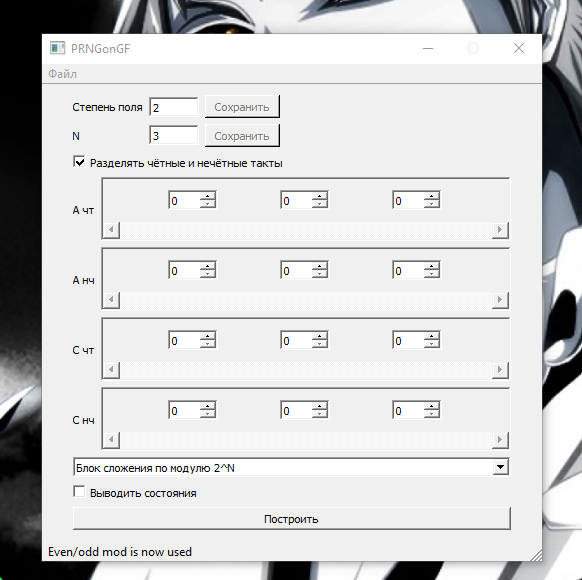


Рис 5.1.2 – интерфейс приложения при разделении ВОЗ и УС

на чётные и нечётные такты

Список блоков сложения, которые будут использоваться при прибавлении УС, предлагает на выбор блоки сложения по модулю или блоки сложения в заданном ранее поле Галуа.

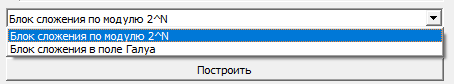


Рис 5.1.3 – список выбора блоков сложения

Флаг “Выводить состояния” нужен для вывода состояний у каждого цикла, так как по умолчанию приложение строит и выводит только циклы с количеством состояний в них. Кнопка “Построить” отвечает за построение и вывод всех состояний генератора.

## Реализация на С++

Файл PRNGonGF.h:

#pragma once

#include <QtWidgets/QMainWindow>

#include <QtWidgets/QSpinBox>

#include <string>

#include "ui\_PRNGonGF.h"

#include "Dialog.h"

#include "Galois Field/Generator/Generator.h"

#include <sstream>

#include <QString>

#include <QMessageBox>

#include <QScrollArea>

class PRNGonGF : public QMainWindow

{

Q\_OBJECT

private slots:

void on\_degree\_edit\_returnPressed();

void on\_n\_edit\_returnPressed();

void on\_even\_checkbox\_stateChanged();

void on\_build\_prng\_clicked();

void on\_n\_edit\_textChanged();

void on\_degree\_edit\_textChanged();

void on\_n\_save\_clicked();

void on\_degree\_save\_clicked();

public:

PRNGonGF(QWidget \*parent = Q\_NULLPTR);

private:

Ui::PRNGonGFClass ui;

QHBoxLayout\* a\_first = nullptr;

QHBoxLayout\* c\_first = nullptr;

QHBoxLayout\* a\_second = nullptr;

QHBoxLayout\* c\_second = nullptr;

QLabel\* a\_label\_first = new QLabel("A");

QLabel\* c\_label\_first = new QLabel("C");

QLabel\* a\_label\_second = nullptr;

QLabel\* c\_label\_second = nullptr;

void remove\_last\_widget\_from\_layout(QBoxLayout\* parent);

void remove\_layout(QBoxLayout\* layout);

void set\_spinbox\_max\_value(QBoxLayout\* parent, int max\_degree);

QSpinBox\* create\_new\_spinbox(int max\_degree);

};

Файл PRNGonGF.cpp:

#include "PRNGonGF.h"

PRNGonGF::PRNGonGF(QWidget \*parent)

: QMainWindow(parent)

{

ui.setupUi(this);

QWidget\* central12 = new QWidget;

QWidget\* central22 = new QWidget;

a\_first = new QHBoxLayout();

c\_first = new QHBoxLayout();

central12->setLayout(a\_first);

central22->setLayout(c\_first);

QScrollArea\* scroll1 = new QScrollArea;

scroll1->setWidget(central12);

scroll1->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOn);

scroll1->setWidgetResizable(true);

ui.a\_even\_layout->addWidget(scroll1);

QScrollArea\* scroll2 = new QScrollArea;

scroll2->setWidget(central22);

scroll2->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOn);

scroll2->setWidgetResizable(true);

ui.c\_even\_layout->addWidget(scroll2);

ui.a\_label\_layout->addWidget(a\_label\_first);

ui.c\_label\_layout->addWidget(c\_label\_first);

ui.degree\_edit->setText("1");

}

void PRNGonGF::remove\_last\_widget\_from\_layout(QBoxLayout\* parent) {

auto i = parent->count();

auto wItem = parent->takeAt(i - 1);

if (wItem->widget())

wItem->widget()->setParent(NULL);

delete wItem;

}

void PRNGonGF::remove\_layout(QBoxLayout\* layout) {

QLayoutItem\* item;

QLayout\* sublayout;

QWidget\* widget;

while ((item = layout->takeAt(0))) {

if ((sublayout = item->layout()) != 0) {}

else if ((widget = item->widget()) != 0) {

widget->hide(); delete widget; }

else { delete item; }

}

delete layout;

}

QSpinBox\* PRNGonGF::create\_new\_spinbox(int max\_degree) {

auto newSpinBox = new QSpinBox();

newSpinBox->setMaximum(std::pow(2, max\_degree) - 1);

newSpinBox->setMaximumWidth(50);

newSpinBox->setMinimumWidth(50);

newSpinBox->setMinimum(0);

return newSpinBox;

}

void PRNGonGF::on\_n\_edit\_returnPressed() {

bool ok;

int max\_degree = ui.degree\_edit->text().toInt();

int new\_n = ui.n\_edit->text().toInt(&ok);

bool use\_even\_odd = ui.even\_checkbox->isChecked();

if (ok == true) {

if(new\_n < 1)

ui.statusBar->showMessage("N can't be less than 1");

else {

ui.statusBar->showMessage("New n: " +

QString::number(new\_n));

int current\_degree = a\_first->count();

for (size\_t i = current\_degree; i < new\_n; i++)

{

a\_first->addWidget(create\_new\_spinbox(max\_degree));

c\_first->addWidget(create\_new\_spinbox(max\_degree));

if (use\_even\_odd) {

a\_second->addWidget(

create\_new\_spinbox(max\_degree));

c\_second->addWidget(

create\_new\_spinbox(max\_degree));

}

}

for (size\_t i = current\_degree; i > new\_n; i--) {

remove\_last\_widget\_from\_layout(a\_first);

remove\_last\_widget\_from\_layout(c\_first);

if (use\_even\_odd) {

remove\_last\_widget\_from\_layout(a\_second);

remove\_last\_widget\_from\_layout(c\_second);

}

}

}

ui.n\_save->setEnabled(false);

}

else{

ui.statusBar->showMessage("Degree can't be a letter");

}

}

void PRNGonGF::on\_n\_edit\_textChanged() {

ui.n\_save->setEnabled(true);

}

void PRNGonGF::on\_degree\_edit\_textChanged() {

ui.degree\_save->setEnabled(true);

}

void PRNGonGF::set\_spinbox\_max\_value(QBoxLayout\* parent, int max\_degree) {

QLayoutItem\* item;

QLayout\* sublayout;

QSpinBox\* widget;

auto count = parent->count();

for (size\_t i = 0; i < count; i++)

{

item = parent->itemAt(i);

widget = dynamic\_cast<QSpinBox\*>(item->widget());

widget->setMaximum(std::pow(2, max\_degree) - 1);

}

}

void PRNGonGF::on\_degree\_edit\_returnPressed() {

bool ok;

int new\_degree = ui.degree\_edit->text().toInt(&ok);

bool use\_even\_odd = ui.even\_checkbox->isChecked();

if (ok == true) {

ui.statusBar->showMessage("New degree: " +

QString::number(new\_degree));

set\_spinbox\_max\_value(a\_first, new\_degree);

set\_spinbox\_max\_value(c\_first, new\_degree);

if (use\_even\_odd) {

set\_spinbox\_max\_value(a\_second, new\_degree);

set\_spinbox\_max\_value(c\_second, new\_degree);

}

ui.degree\_save->setEnabled(false);

}

else {

ui.statusBar->showMessage("Degree can't be a letter");

}

}

void PRNGonGF::on\_degree\_save\_clicked() {

bool ok;

int new\_degree = ui.degree\_edit->text().toInt(&ok);

bool use\_even\_odd = ui.even\_checkbox->isChecked();

if (ok == true) {

ui.statusBar->showMessage("New degree: " +

QString::number(new\_degree));

set\_spinbox\_max\_value(a\_first, new\_degree);

set\_spinbox\_max\_value(c\_first, new\_degree);

if (use\_even\_odd) {

set\_spinbox\_max\_value(a\_second, new\_degree);

set\_spinbox\_max\_value(c\_second, new\_degree);

}

ui.degree\_save->setEnabled(false);

}

void PRNGonGF::on\_n\_save\_clicked() {

bool ok;

int max\_degree = ui.degree\_edit->text().toInt();

int new\_n = ui.n\_edit->text().toInt(&ok);

bool use\_even\_odd = ui.even\_checkbox->isChecked();

if (ok == true) {

if (new\_n < 1)

ui.statusBar->showMessage("N can't be less than 1");

else {

ui.statusBar->showMessage("New n: " +

QString::number(new\_n));

int current\_degree = a\_first->count();

for (size\_t i = current\_degree; i < new\_n; i++)

{

a\_first->addWidget(create\_new\_spinbox(max\_degree));

c\_first->addWidget(create\_new\_spinbox(max\_degree));

if (use\_even\_odd) {

a\_second->addWidget(

create\_new\_spinbox(max\_degree));

c\_second->addWidget(

create\_new\_spinbox(max\_degree));

}

}

for (size\_t i = current\_degree; i > new\_n; i--) {

remove\_last\_widget\_from\_layout(a\_first);

remove\_last\_widget\_from\_layout(c\_first);

if (use\_even\_odd) {

remove\_last\_widget\_from\_layout(a\_second);

remove\_last\_widget\_from\_layout(c\_second);

}

}

}

ui.n\_save->setEnabled(false);

}

else {

ui.statusBar->showMessage("Degree can't be a letter");

}

}

else {

ui.statusBar->showMessage("Degree can't be a letter");

}

}

void PRNGonGF::on\_even\_checkbox\_stateChanged() {

bool use\_even\_odd = ui.even\_checkbox->isChecked();

int max\_degree = ui.degree\_edit->text().toInt();

if (use\_even\_odd) {

ui.statusBar->showMessage("Even/odd mod is now used");

a\_label\_second = new QLabel(QString::fromUtf8(u8"A нч"));

c\_label\_second = new QLabel(QString::fromUtf8(u8"C нч"));

a\_label\_first->setText(QString::fromUtf8(u8"A чт"));

c\_label\_first->setText(QString::fromUtf8(u8"C чт"));

QWidget\* central1 = new QWidget;

QWidget\* central2 = new QWidget;

a\_second = new QHBoxLayout();

c\_second = new QHBoxLayout();

central1->setLayout(a\_second);

central2->setLayout(c\_second);

a\_first->setSizeConstraint(QLayout::SetMinimumSize);

a\_second->setSizeConstraint(QLayout::SetMinimumSize);

c\_first->setSizeConstraint(QLayout::SetMinimumSize);

c\_second->setSizeConstraint(QLayout::SetMinimumSize);

QScrollArea\* scroll1 = new QScrollArea;

scroll1->setWidget(central1);

scroll1->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOn);

scroll1->setWidgetResizable(true);

ui.a\_even\_layout->addWidget(scroll1);

QScrollArea\* scroll2 = new QScrollArea;

scroll2->setWidget(central2);

scroll2->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOn);

scroll2->setWidgetResizable(true);

ui.c\_even\_layout->addWidget(scroll2);

ui.a\_label\_layout->addWidget(a\_label\_second);

ui.c\_label\_layout->addWidget(c\_label\_second);

int current\_degree = a\_first->count();

int old\_degree = a\_second->count();

for (size\_t i = old\_degree; i < current\_degree; i++)

{

a\_second->addWidget(create\_new\_spinbox(max\_degree));

c\_second->addWidget(create\_new\_spinbox(max\_degree));

}

}

else {

ui.statusBar->showMessage("Single mod is now used");

a\_label\_first->setText("A");

c\_label\_first->setText("C");

ui.a\_even\_layout->removeItem(a\_second);

ui.c\_even\_layout->removeItem(c\_second);

remove\_layout(a\_second);

remove\_layout(c\_second);

remove\_last\_widget\_from\_layout(ui.a\_even\_layout);

remove\_last\_widget\_from\_layout(ui.c\_even\_layout);

remove\_last\_widget\_from\_layout(ui.a\_label\_layout);

remove\_last\_widget\_from\_layout(ui.c\_label\_layout);

}

}

void PRNGonGF::on\_build\_prng\_clicked() {

bool ok1;

bool ok2;

int n = ui.degree\_edit->text().toInt(&ok1);

int N = ui.n\_edit->text().toInt(&ok2);

bool use\_even\_odd = ui.even\_checkbox->isChecked();

int sum\_type = ui.field\_box->currentIndex(); // 0 - 2^N, 1 - G

if (ok1 && ok2) {

try{

int p = 2;

auto primitives = prim\_poly::find\_prim(n);

auto field = GaloisField(p, n, primitives.front());

std::vector<GFElement> A;

std::vector<GFElement> C;

int a\_count = a\_first->count();

for (size\_t i = 0; i < a\_count; i++)

{

auto current\_spin\_A\_even = dynamic\_cast<QSpinBox\*>(

a\_first->itemAt(i)->widget());

auto current\_spin\_C\_even = dynamic\_cast<QSpinBox\*>(

c\_first->itemAt(i)->widget());

bool ok1;

bool ok2;

int value\_a = current\_spin\_A\_even->text().toInt(&ok1);

int value\_c = current\_spin\_C\_even->text().toInt(&ok2);

if (ok1 && ok2) {

A.push\_back(field.create\_polynomial(

prim\_poly::detail::to\_binary(value\_a, n)));

C.push\_back(field.create\_polynomial(

prim\_poly::detail::to\_binary(value\_c, n)));

}

else {

QMessageBox msgBox;

msgBox.setText("Wrong A/C parameters");

msgBox.exec();

throw std::exception("Wrong A/C parameters");

}

}

auto generator = Generator(field, N, A, C);

if (use\_even\_odd) {

std::vector<GFElement> A\_odd;

std::vector<GFElement> C\_odd;

int a\_count = a\_first->count();

for (size\_t i = 0; i < a\_count; i++)

{

auto current\_spin\_A\_odd = dynamic\_cast<QSpinBox\*>(

a\_second->itemAt(i)->widget());

auto current\_spin\_C\_odd = dynamic\_cast<QSpinBox\*>(

c\_second->itemAt(i)->widget());

bool ok1;

bool ok2;

int value\_a =

current\_spin\_A\_odd->text().toInt(&ok1);

int value\_c =

current\_spin\_C\_odd->text().toInt(&ok2);

if (ok1 && ok2) {

A\_odd.push\_back(field.create\_polynomial(

prim\_poly::detail::to\_binary(value\_a, n)));

C\_odd.push\_back(field.create\_polynomial(

prim\_poly::detail::to\_binary(value\_c, n)));

}

else {

QMessageBox msgBox;

msgBox.setText("Wrong A/C parameters");

msgBox.exec();

throw std::exception("Wrong A/C parameters");

}

}

generator.set\_odd\_A(A\_odd);

generator.set\_odd\_C(C\_odd);

}

// 0 - 2^N, 1 - G

if (sum\_type) {

generator.set\_even\_iteration\_add(Generator::field\_add);

generator.set\_odd\_iteration\_add(Generator::field\_add);

}

std::ostringstream result\_string;

generator.print\_all\_cycles(result\_string,

ui.display\_states->isChecked());

Dialog\* mDialog = new Dialog(this,

result\_string.str().c\_str());

mDialog->show();

}

catch(std::exception& exc){

QMessageBox msgBox;

msgBox.setText("Invalid data");

msgBox.exec();

ui.statusBar->showMessage(exc.what());

}

}

else {

QMessageBox msgBox;

msgBox.setText("Invalid data");

msgBox.exec();

ui.statusBar->showMessage("Invalid data");

}

}

# Тесты

Были проведены следующие тесты:

1-я группа тестов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № |  |  |  |  | Кол-во состояний в циклах |
| 1 | 2 | 2 | {2, 3} | {0, 0} | 1-3-3-3-3-1-1-1 |
| 2 | 2 | 2 | {2, 3} | {0, 1} | 14-2 |
| 3 | 2 | 2 | {2, 3} | {1, 2} | 14-2 |
| 4 | 2 | 2 | {2, 3} | {2, 3} | 14-2 |
| 5 | 2 | 2 | {2, 1} | {0, 0} | 1-15 |
| 6 | 2 | 2 | {2, 1} | {0, 1} | 7-7-1-1 |

2-я группа тестов

Часть 1 (блоки сложения в поле Галуа)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № |  |  | Чётность такта |  |  | Кол-во состояний в циклах |
| 1 | 2 | 3 | чётный | {2, 3, 0} | {0, 0, 1} | 60-60 |
| нечётный | {0, 0, 3} |
| 2 | 2 | 3 | чётный | {2, 3, 0} | {0, 0, 2} | 60-60 |
| нечётный | {0, 0, 3} |
| 3 | 2 | 3 | — | {2, 3, 0} | {0, 0, 0} | 1-15-15-15-15-1-1-1 |
| 4 | 2 | 3 | — | {2, 3, 0} | {1, 0, 1} | 15-15-15-15-1-1-1-1 |
| 5 | 2 | 3 | — | {2, 3, 0} | {1, 0, 2} | 30-30-2-2 |

Часть 2 (блоки сложения по модулю )

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № |  |  |  |  | Кол-во состояний в циклах |
| 1 | 2 | 3 | {2, 3, 0} | {0, 0, 0} | 1-15-15-15-15-1-1-1 |
| 2 | 2 | 3 | {2, 3, 0} | {0, 0, 1} | 62-2 |
| 3 | 2 | 3 | {2, 3, 0} | {0, 0, 3} | 62-2 |
| 4 | 2 | 3 | {2, 3, 0} | {1, 1, 1} | 8-8-8-8-8-8-8-8 |
| 5 | 2 | 3 | {2, 3, 0} | {1, 2, 1} | 31-31-1-1 |
| 6 | 2 | 3 | {2, 3, 0} | {1, 3, 2} | 31-31-1-1 |

Часть 3 (блоки сложения по модулю )

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № |  |  |  |  | Кол-во состояний в циклах |
| 1 | 2 | 2 | {2, 3} | {0, 0} | 1-3-3-3-3-1-1-1 |
| 2 | 2 | 2 | {2, 3} | {0, 1} | 14-2 |
| 3 | 2 | 2 | {2, 3} | {0, 3} | 14-2 |
| 4 | 2 | 2 | {2, 3} | {1, 2} | 14-2 |
| 5 | 2 | 2 | {2, 1} | {1, 0} | 14-2 |
| 6 | 2 | 2 | {2, 1} | {1, 3} | 4-2-4-1-4-1 |

Часть 4 (блоки сложения по модулю )

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № |  |  |  |  | Кол-во состояний в циклах |
| 1 | 3 | 2 | {2, 3} | {0, 0} | 1-7-7-7-7-7-7-7-7-1-1-1-1-1-1-1 |
| 2 | 3 | 2 | {2, 3} | {0, 1} | 34-2-12-12-2-2 |
| 3 | 3 | 2 | {2, 3} | {0, 3} | 62-2 |
| 4 | 3 | 2 | {2, 3} | {1, 2} | 36-24-2-2 |
| 5 | 3 | 2 | {2, 3} | {1, 1} | 5-28-15-4-2-1-3-4-1-1 |
| 6 | 3 | 2 | {2, 3} | {2, 3} | 12-32-8-4-6-2 |
| 7 | 3 | 2 | {2, 3} | {5, 0} | 16-20-4-20-2-2 |
| 8 | 3 | 2 | {2, 3} | {7, 0} | 4-28-28-4 |
| 9 | 3 | 2 | {2, 3} | {2, 5} | 6-32-12-2-4-8 |