

Правительство Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
(НИУ ВШЭ)

Московский институт электроники и математики им. А.Н. Тихонова

ОТЧЕТ
О ПРАКТИЧЕСКОЙ РАБОТЕ № 1
по дисциплине «Криптографические методы защиты информации»

Построение криптографических операций в полях Галуа

Студент гр. БИБ233

Д.А. Гутников

«24» декабря 2024 г.

Руководитель

Заведующий кафедрой информационной
безопасности киберфизических систем
канд. техн. наук, доцент

_____ О.О. Евсютин

«__» _____ 2024 г.

Москва – 2024 г.

1 ЦЕЛЬ РАБОТЫ

Целью данной работы является приобретение навыков программной реализации операций над многочленами в полях Галуа для построения криптографических преобразований.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1 Поля Галуа

Поле Галуа называется поле F_{p^n} , полученное расширением простого конечного поля F_p посредством неприводимого многочлена $f \in F_p[X]$ степени n . Мощность поля Галуа составляет p^n . Элементами поля Галуа являются многочлены, принадлежащие кольцу многочленов над полем F_p , степень которых строго меньше n . Принадлежность многочлена кольцу многочленов $F_p[X]$ означает, что коэффициенты при степенях данного многочлена являются элементами поля F_p . Таким образом, поле Галуа F_{p^n} состоит из всевозможных остатков от деления многочленов, заданных над полем F_p , на неприводимый многочлен $f \in F_p[X]$ степени n , и в общем случае может быть записано так: $F_{p^n} = \{0, 1, \dots, p-1, x, x+1, \dots, x+(p-1), \dots, (p-1)x^{n-1} + (p-1)x^{n-2} + \dots + (p-1)\}$.

В поле Галуа определены две операции: сложение и умножение.

Чтобы сложить два многочлена–элемента поля Галуа F_{p^n} , необходимо сложить их коэффициенты при соответствующих степенях и привести полученные значения по модулю p .

Чтобы перемножить два многочлена–элемента поля Галуа F_{p^n} , необходимо каждый член одного многочлена умножить на каждый член второго многочлена, привести подобные, привести коэффициенты при степенях полученного многочлена по модулю p и выполнить деление полученного многочлена, степень которого может быть выше $n-1$, на неприводимый многочлен $f \in F_p[X]$ степени n . Остаток от такого деления и будет представлять собой результат перемножения двух исходных элементов поля Галуа.

Известно, что мультипликативная группа $F_{p^n}^*$ поля Галуа F_{p^n} , включающая все ненулевые элементы поля, представляет собой циклическую группу порядка $q = p^n - 1$. Это означает, что каждый элемент данной группы может быть представлен в виде некоторой степени образующего элемента $\alpha \in F_{p^n}^*$: $F_{p^n}^* = \langle \alpha \rangle = \{1, \alpha, \alpha^2, \dots, \alpha^{q-1}\}$. Количество образующих элементов группы $F_{p^n}^*$ может быть определено через функцию Эйлера как $\varphi(q)$. Разложение элементов мультипликативной группы поля Галуа по степеням образующего позволяет реализовать операцию умножения многочленов в поле

Галуа более простым образом. Пусть даны два многочлена $u, v \in F_p^* = \langle \alpha \rangle$, причем известно, что $u = \alpha^k$, $v = \alpha^l$. Тогда умножение данных многочленов может быть выполнено по следующей формуле:

$$u \cdot v = \alpha^k \alpha^l = \alpha^{(k+l) \bmod q}. \quad (1)$$

2.2 Построение аффинного шифра над полем Галуа

Математический аппарат полей Галуа широко используется для конструирования криптографических операций в современных симметричных шифрах. В качестве основных примеров можно привести шифры AES и Кузнечик. В шифре AES один из этапов основного криптографического преобразования состоит в замене байтов мультипликативно обратными значениями в группе $F_{2^8}^*$. В шифре Кузнечик в качестве одного из базовых преобразований используется свертка 16-байтового слова в один байт посредством линейного преобразования в поле Галуа F_{2^8} .

Наиболее простым примером шифра, который может быть построен над полем Галуа, является аффинный шифр, основанный на так называемом аффинном преобразовании.

Пусть A – это алфавит, используемый для представления сообщений, подлежащих шифрованию. Символы данного алфавита представляются в виде элементов поля Галуа F_{p^n} одним из двух возможных способов:

- Параметры поля Галуа F_{p^n} выбираются таким образом, чтобы выполнялось равенство $|A| = p^n$. Это не всегда возможно для алфавитов естественных языков, поэтому можно работать с усеченным или расширенным алфавитом.

- Независимо от используемого алфавита естественного языка сообщение представляется в виде двоичной последовательности, которая разбивается на n -разрядные блоки. Отдельно взятый блок рассматривается как символ сообщения, подлежащий замене с помощью аффинного шифра. Алфавит, составленный из таких символов, будет иметь мощность 2^n , поэтому он может быть представлен в виде поля Галуа F_{2^n} .

Тогда открытый текст после перехода от исходного алфавита A к полю Галуа F_{p^n} может быть обозначен $x = (x_1, \dots, x_l)$, соответствующий шифртекст – $y = (y_1, \dots, y_l)$, где $x_i, y_i \in F_{p^n}$, $i = \overline{1, l}$. В качестве ключа аффинного шифра, построенного над полем Галуа F_{p^n} , выступает пара значений $k = (\alpha, \beta)$, $\alpha \in F_{p^n}^*$, $\beta \in F_{p^n}$, и ключевое пространство имеет вид $K = F_{p^n}^* \times F_{p^n}$.

Зашифрование отдельного символа открытого текста осуществляется по следующей формуле:

$$y_i = \alpha x_i + \beta, i = \overline{1, l}. \quad (2)$$

Расшифрование символа шифртекста осуществляется по формуле

$$x_i = (y_i - \beta)\alpha^{-1}, i = \overline{1, l}, \quad (3)$$

где $\alpha^{-1} \in F_{p^n}^*$ – элемент поля Галуа, мультипликативно обратный к α .

3 ЗАДАНИЕ

- 1) написать программную реализацию инструмента, позволяющего строить и исследовать поле Галуа F_{p^n} ;
- 2) написать программную реализацию аффинного шифра над полем Галуа F_{p^n} ;
- 3) подготовить отчет о выполнении работы.

Инструмент для построения и исследования полей Галуа должен обладать следующей функциональностью:

- 1) принимать на вход значения p и n , определяющие поле Галуа, и строить и отображать соответствующее поле Галуа F_{p^n} ;
- 2) принимать на вход неприводимый многочлен $f \in F_p[X]$ степени n либо генерировать такие многочлены для заданных значений p и n ;
- 3) принимать на вход многочлены–элементы поля Галуа F_{p^n} и осуществлять их сложение или умножение по выбору пользователя;
- 4) находить образующие элементы мультипликативной группы поля Галуа $F_{p^n}^*$ и выполнять разложение элементов данной группы по степеням выбранного образующего.

Программная реализация аффинного шифра должна обладать следующей функциональностью:

- 1) принимать на вход произвольную последовательность символов, вводимую пользователем в качестве открытого текста или шифртекста;
- 2) принимать на вход секретный ключ $k = (\alpha, \beta)$, $\alpha \in F_{p^n}^*$, $\beta \in F_{p^n}$;
- 3) осуществлять зашифрование или расшифрование введенного текста по выбору пользователя.

Отчет должен содержать следующие составные части:

- 1) раздел с заданием;
- 2) раздел с описанием особенностей программных реализаций;
- 3) раздел с демонстрацией примеров работы программных реализаций и объяснением полученных результатов;
- 4) раздел с выводами о проделанной работе.

4 ОПИСАНИЕ ОСОБЕННОСТЕЙ ПРОГРАММНЫХ РЕАЛИЗАЦИЙ

Поле Галуа – математическая структура с ресурсоёмкими операциями, поэтому в данной практической работе предпочтительнее выбрать быстрый язык программирования. Хорошим выбором будет язык C#.

В данной реализации логика разделена на несколько пространств имён, которые представлены в таблице ниже (таблица 1).

Таблица 1 – Пространства имён программной реализации.

Пространство имён	Идея
Exceptions	В данном пространстве располагаются объявления исключений, возникающих в любой части программы.
Galois_Field	Данное пространство имён содержит в себе класс <code>Galois_Field</code> , содержащий атрибуты <code>p</code> , <code>n</code> , <code>fx</code> (неприводимый многочлен), <code>forming_member</code> (образующий элемент), <code>member</code> (элемент поля) – все они представлены в виде целых чисел, где число для элементов является результатом подстановки в многочлены $X=p$. В классе располагаются методы по работе с элементами и определены операторы сложения, разности, произведения и частного.
Numbers	Здесь располагаются статические элементы для работы с числами: возведение в степень, деление многочленов, сумма, разность, произведение и другие.
Dictionaries	В данном пространстве располагаются самостоятельно созданные алфавиты для зашифрования и расшифрования.
Ciphers	Пространство шифров содержит в себе методы для зашифрования и расшифрования входной строки по паре ключей, основанных на поле Галуа.
ConsoleProgram	Содержит консольную программу, предоставляющую удобное взаимодействие с инструментом.
Program	Здесь расположен класс с методом <code>Main()</code> , запускающим программу.

В данной практической работе имеет смысл рассмотрение пространств `Galois_Field`, `Numbers`, `Ciphers`, поскольку вся логика теоретической части заложена именно там.

В пространстве `Galois_Field` объявлен конструктор, позволяющий создать экземпляр класса по известным параметрам `p`, `n`, `member`. Также можно указать конкретные `forming_member` и `fx`, если такие известны, чтобы ускорить создание экземпляра. Также объявлены методы, позволяющие изучить элемент поля Галуа, и перегружены операторы для простой работы с классами.

В таблице 2 приведено описание методов данного класса.

Таблица 2 – Методы класса `Galois_Field`.

Метод	Идея
<code>get_p()</code>	Возвращает <code>p</code> .
<code>get_n()</code>	Возвращает <code>n</code> .
<code>get_fx()</code>	Возвращает неприводимый многочлен <code>fx</code> в виде числа.
<code>get_forming_element()</code>	Возвращает образующий элемент.
<code>set_member(int member)</code>	Записывает в <code>this.member</code> значение <code>member</code> .
<code>set_forming_element(int suggested_forming_member)</code>	Записывает <code>suggested_forming_member</code> в <code>this.forming_member</code> , предварительно проверяя, является ли оно образующим элементом данного поля.
<code>check_if_fx_irreducible(int fx)</code>	Возвращает <code>true</code> , если многочлен <code>fx</code> неприводим, иначе <code>false</code> . Проверка осуществляется перебором всех возможных бинарных произведений.
<code>find_fx()</code>	Ищет и возвращает неприводимый многочлен.
<code>find_reversed_member()</code>	Ищет и возвращает обратный элемент с учётом неприводимого многочлена.
<code>check_if_forming(int member)</code>	Возвращает <code>true</code> , если <code>member</code> является образующим элементом, иначе <code>false</code> . Проверка осуществляется последовательным произведением до того момента, пока в итоге не получится 1.
<code>find_forming_element()</code>	Возвращает первый <code>forming_element</code> , найденный перебором.
<code>find_forming_element(int random)</code>	Возвращает <code>forming_element</code> под номером <code>random+1</code> .
<code>find_member_degree()</code>	Возвращает такое <code>degree</code> , в которое нужно возвести <code>forming_member</code> , чтобы получить <code>member</code> .
<code>find_member_degree(int suggested_forming_element)</code>	Возвращает такое <code>degree</code> , в которое нужно возвести <code>suggested_forming_member</code> , чтобы получить <code>member</code> . Осуществляется проверка того, что аргумент является образующим элементом.
<code>find_member_order()</code>	Возвращает порядок элемента.
<code>visualizer(int current, int base_number)</code>	Возвращает строку-визуализацию многочлена.
<code>visualize_member()</code>	Возвращает строку-визуализацию элемента.
<code>visualize_irreducible_fx()</code>	Возвращает строку-визуализацию неприводимого многочлена.
<code>visualize_forming_element()</code>	Возвращает строку-визуализацию образующего элемента.

В таблице 3 три приведено пояснение для всех перегрузок операторов для данной реализации поля Галуа.

Таблица 3 – Перегрузки операторов `Galois_Field`.

Перегрузка оператора	Идея
$+$	Сложение. Сложением двух элементов в поле Галуа будет являться элемент из того же поля, коэффициенты перед степенями которого были получены в результате поразрядного сложения с приведением по модулю p .
$-$	Разность. Разностью двух элементов в поле Галуа будет являться элемент из того же поля, коэффициенты перед степенями которого были получены в результате поразрядной разности с приведением по модулю p .
$*$	Произведение. Произведением двух элементов в поле Галуа будет являться элемент из того же поля, полученный в результате перемножения многочленов друг на друга с выделением остатка от деления на неприводимый многочлен.
$/$	Частное. Частного в классическом понимании в поле Галуа нет, но его можно добавить как произведение первого элемента на обратный второму элемент.

В классе `Number_Operations` пространства имён `Numbers` приведены простейшие операции над числами, необходимые для работы с полем Галуа: сумма и разность по модулю, деление одного многочлена на другой, возведение в степень числа, возведение многочлена в степень с использованием неприводимого многочлена, операции по извлечению или замене коэффициентов, нахождение наибольшего общего делителя и другие.

Данные статические методы имели широкое применение в классе `Galois_Field`. Их описание представлено в таблице 4.

Таблица 4 – Методы класса `Number_Operations`. Часть 1.

Метод	Идея
<code>greatest_common_divisor(int a, int b)</code>	Возвращает наибольший общий делитель по алгоритму Евклида.
<code>get_power(int number, int base_number)</code>	Возвращает наибольшую степень из разрядов числа в СС с основанием в <code>base_number</code> .
<code>pow(int number, int degree)</code>	Возвращает степень <code>degree</code> для числа <code>number</code> .
<code>fx_pow(int member, int fx, int degree, int base_number, int n_limit)</code>	Возводит многочлен, представленный в виде числа по основанию <code>base_number</code> в степень <code>degree</code> , с использованием неприводимого многочлена. Возвращает полученное число.

Таблица 4 – Методы класса Number_Operations. Часть 2.

Метод	Идея
<code>get_coef_on_place(int number, int base_number, int position)</code>	Представляет число в виде числа в СС <code>base_number</code> и возвращает разряд с индексом <code>position</code> .
<code>set_coef_on_place(int number, int base_number, int position, int replace_coef)</code>	Представляет число в виде числа в СС <code>base_number</code> и заменяет разряд с индексом <code>position</code> на <code>replace_coef</code> . Возвращает полученное число.
<code>plus(int a, int b, int base_number)</code>	Возвращает поразрядную сумму двух чисел по модулю <code>base_number</code> .
<code>minus(int a, int b, int base_number)</code>	Возвращает поразрядную разность двух чисел по модулю <code>base_number</code> .
<code>mul(int a, int b, int base_number)</code>	Возвращает произведение двух многочленов, представляя их в виде чисел, по модулю <code>base_number</code> .
<code>normalizer(int member, int fx, int n_limit, int base_number)</code>	Возвращает остаток многочлена <code>member</code> от деления на <code>fx</code> в виде числа с основанием <code>base_number</code> .
<code>polynomial_maker(int number, int base_number)</code>	Возвращает ссылку на массив, в котором лежат разряды числа <code>number</code> , представленного в СС по основанию <code>base_number</code> .

Поле Галуа готово, теперь можно приступить к Аффинному шифру. Он реализован в классе `AffineCipher`, содержащем в себе методы зашифрования и расшифрования, а также метод поиска элемента в алфавите.

Таблица 5 – Методы класса AffineCipher. Часть 1.

Метод	Идея
<code>find_position_in_dict(char symbol, char[] dict)</code>	Возвращает индекс символа <code>symbol</code> в массиве <code>dict</code> , если находит его, иначе возвращает -1.
<code>encrypt_default(string input, Galois_Field.Galois_Field key_a, Galois_Field.Galois_Field key_b)</code>	Возвращает шифр-текст для текста <code>input</code> на основе ключей <code>key_a</code> и <code>key_b</code> и Unicode таблицы.
<code>decrypt_default(string input, Galois_Field.Galois_Field key_a, Galois_Field.Galois_Field key_b)</code>	Возвращает исходный текст для шифр-текста <code>input</code> на основе ключей <code>key_a</code> и <code>key_b</code> и Unicode таблицы.
<code>encrypt_with_dict(string input, Galois_Field.Galois_Field key_a, Galois_Field.Galois_Field key_b, char[] dict)</code>	Возвращает шифр-текст для текста <code>input</code> на основе ключей <code>key_a</code> и <code>key_b</code> и алфавита <code>dict</code> .

Таблица 5 – Методы класса AffineCipher. Часть 2.

Метод	Идея
decrypt_with_dict(string input, Galois_Field.Galois_Field key_a, Galois_Field.Galois_Field key_b, char[] dict)	Возвращает исходный текст для шифр-текста input на основе ключей key_a и key_b и алфавита dict.

Остальные классы реализованы для удобного взаимодействия с пользователем.

5 ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

5.1 Инструмент для работы с полем Галуа

При запуске программа встречает нас удобным консольным интерфейсом (рисунок 1). В главном меню можно выбрать, инструмент, с которым работать: 1) инструмент для работы с полем Галуа, 2) инструмент для шифрования.

```

Выберите режим работы программы:
  1) Инструмент для работы с полем Галуа;
  2) Инструмент для шифрования;
  3) Прекратить.

Выбранный режим:
  
```

Рисунок 1. Главное меню программы.

Запустим инструмент для работы с полем Галуа, введя число 1. Чтобы создать поле Галуа на следующем этапе нужно ввести необходимые параметры: 1) p , 2) n , 3) выбранный элемент, 4) неприводимый многочлен, 5) образующий элемент (рисунок 2). После ввода нас встретит меню выбора дальнейших действий

```

Выбран инструмент для работы с полем Галуа!
Введите основные параметры поля Галуа:

Введите основание p: 3
Введите степень n: 4
Подставьте в многочлен поля X=3 и впишите результат: 7
Аналогично впишите число для неприводимого многочлена
Если нужно выбрать автоматически, то впишите 0: 0
Точно так же введите образующий элемент
Если без него, то 0: 0
  
```

Рисунок 2. Ввод параметров поля Галуа.

Отлично! Вы добавили элемент в поле Галуа!

Визуализация элемента A: $2 \cdot X + 1$

В виде числа: 7

Выберите дальнейшие действия:

- 1) Вывести образующий элемент;
- 2) Вывести неприводимый многочлен;
- 3) Вычислить степень элемента;
- 4) Вычислить порядок элемента;
- 5) Найти обратный элемент;
- 6) Возвести в степень;
- 7) Изменить A;
- 8) Изменить B;
- 9) + B;
- 10) - B;
- 11) * B;
- 12) / B;
- 13) Выйти в главное меню.

Выбранный режим:

Рисунок 3. Выбор дальнейших действий.

Пройдёмся по пунктам 1-12 чтобы убедиться в работоспособности данной программы. На рисунке 4-5 выведены образующий элемент и неприводимый многочлен, а в таблице 6 представлено возведение в степень проверяемого многочлена.

Образующий элемент: X
В виде числа: 3
Повторить? Нажмите Enter ☐

Рисунок 4. Образующий элемент.

Неприводимый многочлен: $X^4 + X + 2$
В виде числа: 86
Повторить? Нажмите Enter_

Рисунок 5. Неприводимый многочлен.

Таблица 6 – Поиск порядка элемента x .

$ F_{3^4}^* = 3^4 - 1 = 80$ Возможные порядки: 1, 2, 4, 8, 10, 16, 20, 40, 80.	
Степень	Результат
1	$x^1 = x$
2	$x^2 = x^2$
4	$x^4 = 2x + 1$
5	$x^5 = x^4 * x = (2x + 1) * x = 2x^2 + x$
8	$x^8 = (x^4)^2 = (2x + 1)^2 = 4x^2 + 4x + 1 = x^2 + x + 1$
10	$x^{10} = x^8 * x^2 = x^4 + x^3 + x^2 = (2x + 1) + x^3 + x^2 =$ $= x^3 + x^2 + 2x + 1$
16	$x^{16} = (x^4)^4 = (2x+1)^4 = 16x^4 + 32x^3 + 24x^2 + 8x + 1 =$ $= x^4 + 2x^3 + 2x + 1 = 2x^3 + x + 2$
20	$x^{20} = x^{16} * x^4 = (2x^3 + x + 2) * (2x + 1) =$ $= x^4 + 2x^2 + x + 2x^3 + x + 2 = (2x + 1) + 2x^3 + 2x^2 +$ $+ 2x + 2 = 2x^3 + 2x^2 + x$
40	$x^{40} = (x^{20})^2 = (2x^3 + 2x^2 + x)^2 = x^6 + x^4 + x^2 +$ $+ 4x^4 + 4x^3 + 8x^5 = x^6 + 2x^5 + 2x^4 + x^3 + x^2 =$ $= x^5 * x + 2 * (2x^2 + x) + 2 * (2x + 1) + x^3 + x^2 =$ $= (2x^2 + x) * x + (x^2 + 2x) + (x + 2) + x^3 + x^2 =$ $= (2x^3 + x^3) + (x^2 + x^2 + x^2) + (2x + x) + 2 = 2$

В результате перебора возможных порядков от 2 до 40 остался последний, и действительно: $x^{80} = (x^{40})^2 = (2)^2 = 1$, то есть порядок данного элемента – 80, или же многочлен x является образующим данной группы. Выходит, что программа отработала здесь верно.

Определим, в какую степень нужно возвести образующий элемент, чтобы получить введённый ранее элемент $2x + 1$ (рисунок 6).

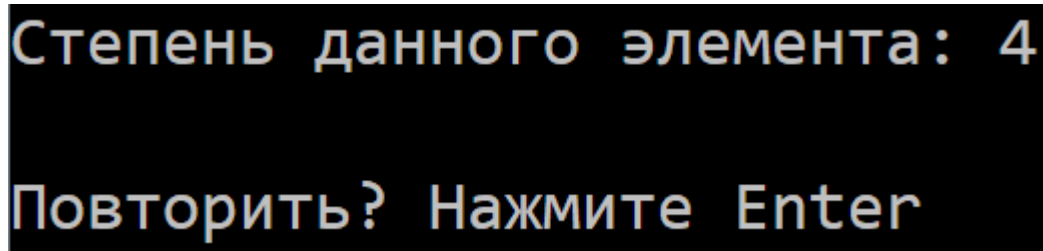


Рисунок 6. Степень элемента.

Действительно! Был же определён неприводимый многочлен $x^4 + x + 2$, другими словами: $x^4 + x + 2 = 0 \Leftrightarrow x^4 = 2x + 1$ –, что как раз и нужно было найти.

Найдём порядок элемента с помощью программы (рисунок 7) и сравним результат с ручными вычислениями.

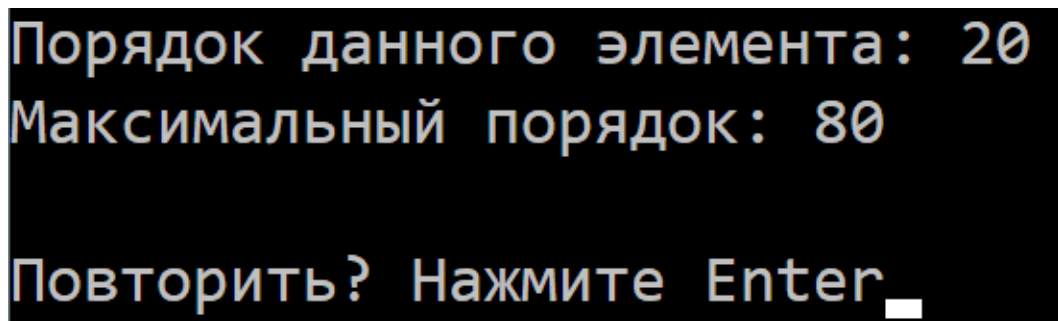


Рисунок 7. Вычисление порядка элемента.

Порядок элемента можно найти по формуле: $O(x_{\text{обр.}}^i) = \frac{|F_p^{*n}|}{\text{НОД}(i, |F_p^{*n}|)}$. В данном случае $O(x^4) = \frac{80}{\text{НОД}(4, 80)} = \frac{80}{4} = 20$, что и было получено на практике.

Найдём обратный элемент (рисунок 8).

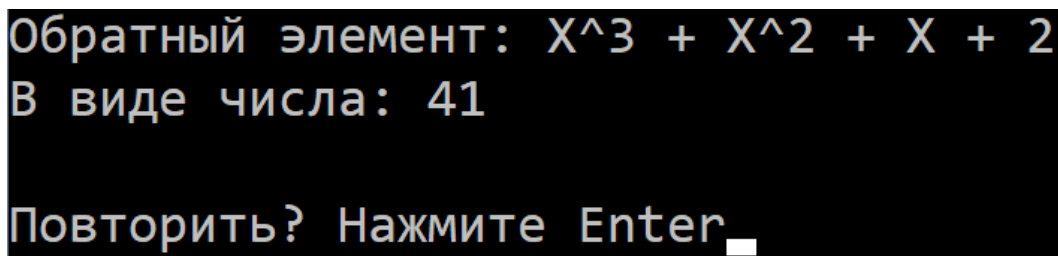


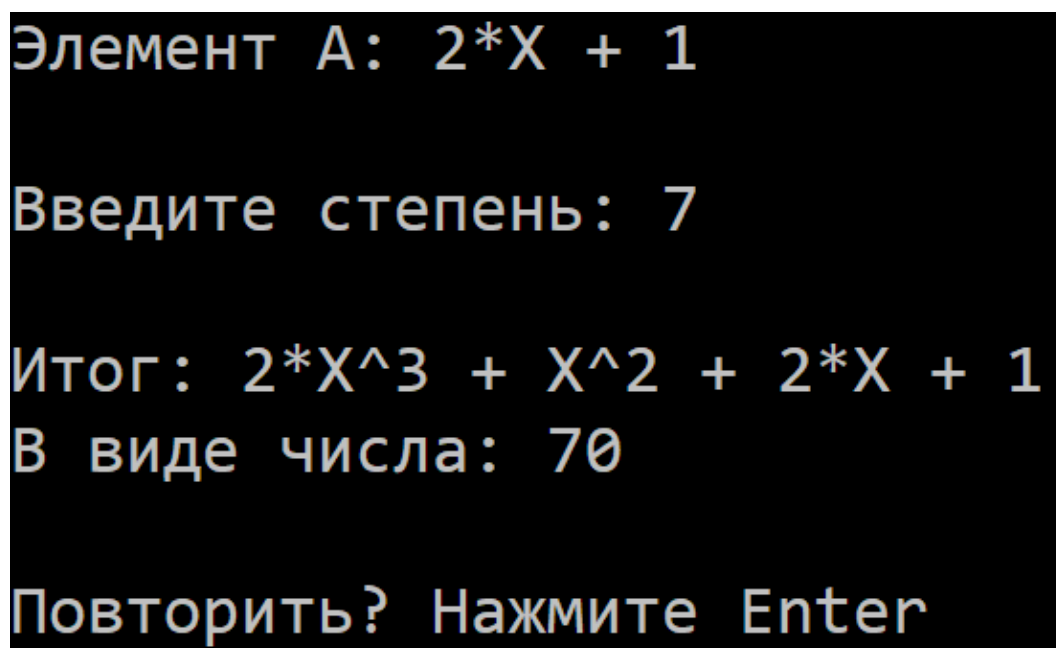
Рисунок 8. Обратный элемент.

Проверим произведение исходного и обратного элементов:

$$(2x + 1) * (x^3 + x^2 + x + 2) = 2x^4 + 2x^3 + 2x^2 + x + x^3 + x^2 + x + 2 = 2 * (2x + 1) + 2x + 2 = 1.$$

Да, произведение исходного и обратного равно 1, что и должно было произойти в теории.

Попробуем возвести в конкретную степень (рисунок 9).



```
Элемент А: 2*X + 1
Введите степень: 7
Итог: 2*X^3 + X^2 + 2*X + 1
В виде числа: 70
Повторить? Нажмите Enter
```

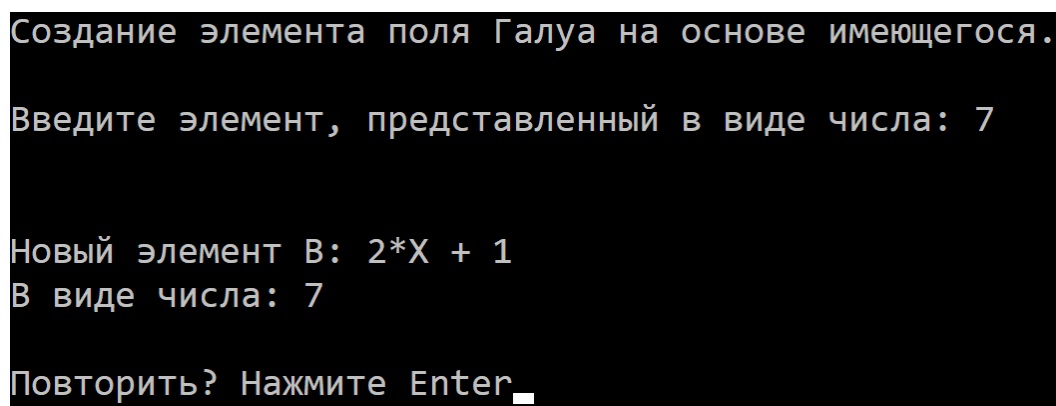
Рисунок 9. Возведение в степень.

Возведём аналогично вручную элемент $2x + 1$:

$$\begin{aligned} (2x + 1)^7 &= (2x + 1) * ((2x + 1)^3)^2 = (2x + 1) * (8x^3 + 12x^2 + 6x + 1)^2 = (2x + 1) * (2x^3 + 1)^2 = \\ &= (2x + 1) * (4x^6 + 4x^3 + 1) = (2x + 1) * (x^4 * x^2 + x^3 + 1) = (2x + 1) * (2x^3 + x^2 + x^3 + 1) = \\ &= (2x + 1) * (x^2 + 1) = 2x^3 + x^2 + 2x + 1. \end{aligned}$$

Теоретические и практические значения совпали, всё верно.

Попробуем создать копию данного элемента в В, найдём сумму, разность, произведение и частное (рисунок 10-14):



```
Создание элемента поля Галуа на основе имеющегося.
Введите элемент, представленный в виде числа: 7
Новый элемент В: 2*X + 1
В виде числа: 7
Повторить? Нажмите Enter
```

Рисунок 10. Создание копии элемента.

Сумма $A + B$: $X + 2$
В виде числа: 5
Повторить? Нажмите Enter

Рисунок 11. Сумма A и B.

Разность $A - B$:
В виде числа: 0
Повторить? Нажмите Enter

Рисунок 12. Разность A и B.

Произведение $A * B$: $X^2 + X + 1$
В виде числа: 13
Повторить? Нажмите Enter_

Рисунок 13. Произведение A и B.

Частное A / B : 1
В виде числа: 1
Повторить? Нажмите Enter_

Рисунок 14. Частное A и B.

Сумма и разность соответствуют действительности, произведение можно проверить по уже готовым вычислениям (в таблице 6 возведение в 8 степень эквивалентно

произведению А и В), частное двух равных элементов равно 1, что и наблюдается на практике.

Инструмент для работы с полем Галуа работает исправно, можно вернуться в главное меню и выбрать инструмент для шифрования.

5.2 Инструмент для шифрования

Данный инструмент может работать с 4 видами алфавитов:

1. *Default* – алфавитом являются все символы кодировки Unicode, ограничения выставляются пользователем при вводе параметров поля Галуа n и p . Копирование в буфер обмена шифр-текста для данного словаря не может гарантировать целостность информации!
2. *dict_base2_degree5* – данный алфавит содержит в себе 26 прописных букв латинского алфавита и 6 специальных символов. Его преимущество состоит в скорости работы, а недостаток – в маленькой мощности.
3. *dict_base2_degree6* – данный алфавит содержит в себе 26 прописных и 26 строчных букв латинского алфавита, а также 12 специальных символов. Данный алфавит хорошо балансирует между большим выбором символов и невысокой сложностью алгоритмов.
4. *dict_base2_degree7* – данный алфавит содержит строчные и прописные буквы русского и латинского алфавита, а также 10 специальных символов. Данный алфавит удобно использовать в силу своих размеров и большого охвата символов.

Для проверки инструмента выберем *dict_base2_degree6* и *Default* варианты алфавитов (рисунок 15).

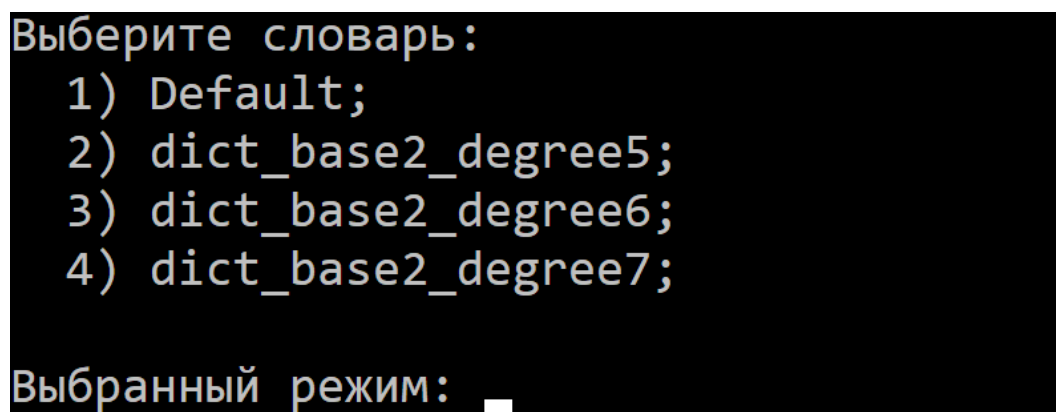
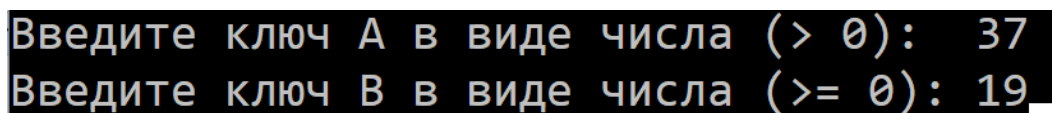


Рисунок 15. Выбор алфавита.

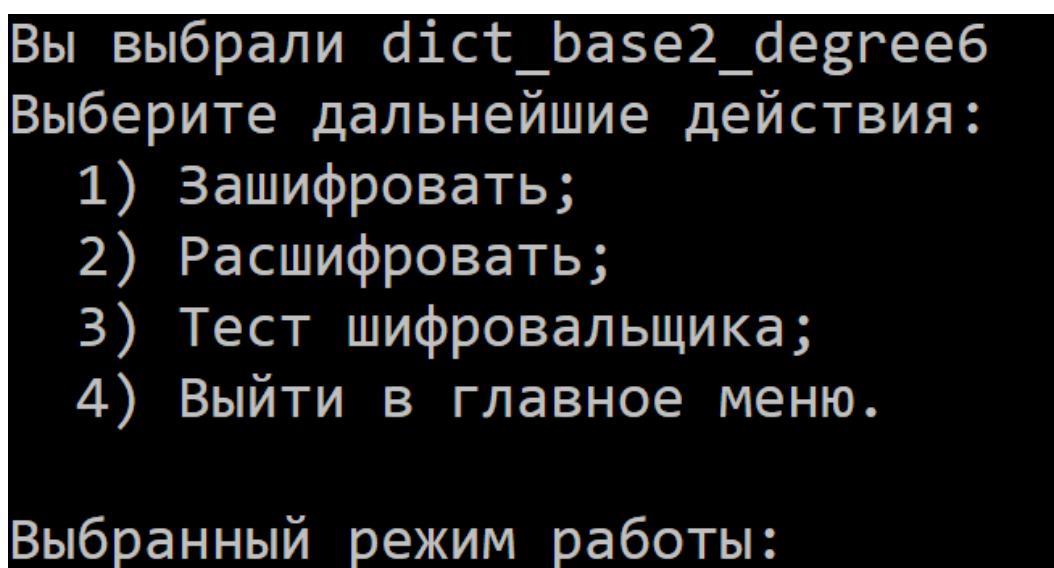
Если был выбран алфавит Default, то программа потребует ввести дополнительные параметры поля Галуа: p и n . В противном случае они будут выбраны автоматически исходя из размеров алфавита.

Дальше нужно выбрать ключи для данного шифра: $key_a \geq 1$, $key_b \geq 0$ (рисунок 16). После выбора ключей появляется возможность выбрать режим работы программы (рисунок 17).



```
Введите ключ А в виде числа (> 0): 37
Введите ключ В в виде числа (>= 0): 19
```

Рисунок 16. Ввод ключей.

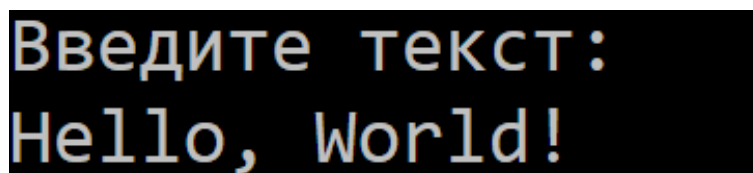


```
Вы выбрали dict_base2_degree6
Выберите дальнейшие действия:
1) Зашифровать;
2) Расшифровать;
3) Тест шифровальщика;
4) Выйти в главное меню.

Выбранный режим работы:
```

Рисунок 17. Выбор режима работы инструмента.

Выберем «3) Тест шифровальщика» и протестируем программу на примере известной строки «Hello, World!» (рисунок 18). Если алгоритм реализован правильно, то в консоли будет написано, что программа отработала верно, иначе будет сказано, что что-то пошло не так.



```
Введите текст:
Hello, World!
```

Рисунок 18. Ввод текста.

К счастью, программа отработала успешно (рисунок 19).


```
Используемый словарь: dict_base2_degree6

Тест работы программы:
Введённый текст: "Hello, World!"
Шифр-текст:      "ocffQqHAQkf?z"
Расшифровка:     "Hello, World!"

Программа отработала верно!

Повторить? Нажмите Enter_
```

Рисунок 19. Успешное выполнение программы.

Попробуем зашифровать данную строку с помощью алфавита Default, взяв, например, значения $p = 3$ и $n = 6$, ключи выберем те же (рисунок 20). Проверим работу программы (рисунок 21).

```
Введите необходимые параметры поля:
  p = 3
  n = 6
Введите ключ А в виде числа (> 0): 37
Введите ключ В в виде числа (>= 0): 19
```

Рисунок 20. Ввод параметров для шифра.

```
Используемый словарь: Default

Тест работы программы:
Введённый текст: "Hello, World!"
Шифр-текст:      "ŮòŮŮŷŸŷGŷŷŮîŮ"
Расшифровка:     "Hello, World!"

Программа отработала верно!

Повторить? Нажмите Enter_
```

Рисунок 21. Успешное выполнение программы для алфавита Default.

Несмотря на странный вид шифр-текста, программа смогла успешно расшифровать данную замысловатую последовательность символов. Отдельно можно проверить, сможет ли скопированная в буфер обмена запись расшифроваться (рисунок 22).

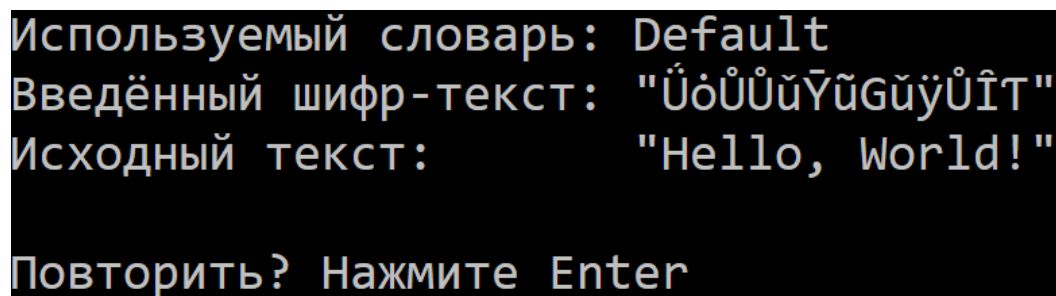
A screenshot of a terminal window with a black background and yellow text. It displays the results of a decryption process. The text is as follows:
Используемый словарь: Default
Введённый шифр-текст: "ŮòŮŮŷŸŷGŷŷŮÎT"
Исходный текст: "Hello, World!"
Повторить? Нажмите Enter

Рисунок 22. Проверка возможности расшифровки после копирования.

В этот раз расшифровка была успешно выполнена, но это не гарантирует успешную расшифровку и в других случаях при других параметрах n , p , key_a , key_b .

6 ВЫВОД ПО ПРОДЕЛАННОЙ РАБОТЕ

В результате проделанной работы была написана программная реализация для полей Галуа с различными методами, позволяющими всеобъемлюще изучить элементы конкретного поля. Был реализован аффинный шифр на основе полей Галуа, была добавлена возможность выбора между различными алфавитами. На практике было получено, что расширение аффинного шифра с помощью полей Галуа делает его более криптостойким, так как увеличивается сложность подбора конкретного ключа.

7 ИСПОЛЬЗОВАННЫЕ МАТЕРИАЛЫ

Презентации Smart LMS курса «Криптографические методы защиты информации».

8 ПРИЛОЖЕНИЕ

Ниже представлен код программы, написанный в результате выполнения данной практической работы.

```
using Exceptions;
using System;
using System.Windows.Forms;
using System.Text;

namespace Exceptions
{
    public class IncompatibleGaloisFieldMembers : Exception
    {
        public IncompatibleGaloisFieldMembers() : base("Несоответствие n или p в полях Галуа.") { }
    }
    public class CurrentMemberIsNotForming : Exception
    {
        public CurrentMemberIsNotForming() : base("Выбранный элемент не является образующим данной группы.") { }
    }
    public class WrongIrreduciblePolynomial : Exception
    {
        public WrongIrreduciblePolynomial() : base("Данный многочлен не является неприводимым.") { }
    }
    public class IncompatibleCharAndField : Exception
    {
        public IncompatibleCharAndField() : base("Значение char не влезает в выбранное поле.") { }
    }
    public class NonexistingCharInDict : Exception
    {
        public NonexistingCharInDict() : base("Данного символа нет в выбранном алфавите.") { }
    }
    public class WrongDictForField : Exception
    {
        public WrongDictForField() : base("У поля и словаря разные размеры.") { }
    }
    public class WrongInput : Exception
    {
        public WrongInput() : base("Неправильный ввод данных.") { }
    }
    public class NoValidNumbersFound : Exception
    {
        public NoValidNumbersFound() : base("Не найдены параметры поля Галуа.") { }
    }
}

namespace Galois_Field
{
    public class Galois_Field
    {
        private int p;
        private int n;
        private int fx;

        private int forming_element;
        private int member;
```

```

public Galois_Field(int p, int n, int member)
{
    this.p = p;
    this.n = n;

    this.fx = find_fx();
    this.forming_element = find_forming_element();

    this.member = Numbers.Number_Operations.normalizer(member, fx, n, p);
}
public Galois_Field(int p, int n, int member, int fx)
{
    this.p = p;
    this.n = n;

    if (check_if_fx_irreducible(fx)) { this.fx = fx; }
    else { throw new Exceptions.WrongIrreduciblePolynomial(); }

    this.forming_element = find_forming_element();

    this.member = Numbers.Number_Operations.normalizer(member, fx, n, p);
}
public Galois_Field(int p, int n, int member, int fx, int forming_element)
{
    this.p = p;
    this.n = n;

    if (check_if_fx_irreducible(fx)) { this.fx = fx; }
    else { throw new Exceptions.WrongIrreduciblePolynomial(); }

    if (check_if_forming(forming_element)) { this.forming_element = forming_element; }
    else { throw new Exceptions.CurrentMemberIsNotForming(); }

    this.member = Numbers.Number_Operations.normalizer(member, fx, n, p);
}

public int get_p() { return p; }
public int get_n() { return n; }
public int get_fx() { return fx; }
public int get_member() { return member; }
public int get_forming_element() { return forming_element; }
public void set_member(int member) { this.member = member; }

public void set_forming_element(int suggested_forming_member)
{
    if (check_if_forming(member) == false)
    {
        throw new Exceptions.CurrentMemberIsNotForming();
    }
    this.forming_element = suggested_forming_member;
}

public bool check_if_fx_irreducible(int fx)
{
    if (fx < Numbers.Number_Operations.pow(p, n) + 1) { return false; }
    for (int member_a = p; member_a <= Numbers.Number_Operations.pow(p, n) - 1; member_a++)
    {
        for (int member_b = member_a; member_b <= Numbers.Number_Operations.pow(p, n) - 1; member_b++)
        {

```

```

        if (fx == Numbers.Number_Operations.mul(member_a, member_b, p))
        {
            return false;
        }
    }
}
return true;
}
private int find_fx()
{
    int max_member = Numbers.Number_Operations.pow(p, n + 1);
    int min_member = Numbers.Number_Operations.pow(p, n);
    int out_fx = -1;

    for (int current_fx = min_member; current_fx < max_member; current_fx++)
    {
        bool fx_found = true;

        for (int member_a = p; member_a <= Numbers.Number_Operations.pow(p, n) - 1; member_a++)
        {
            for (int member_b = member_a; member_b <= Numbers.Number_Operations.pow(p, n) - 1;
member_b++)
            {
                if (current_fx == Numbers.Number_Operations.mul(member_a, member_b, p))
                {
                    fx_found = false;
                    break;
                }
            }
            if (fx_found == false)
            {
                break;
            }
        }

        if (fx_found)
        {
            out_fx = current_fx;
            break;
        }
    }

    return out_fx;
}
public int find_reversed_member()
{
    if (member == 0)
    {
        return 0;
    }
    int reversed_degree = Numbers.Number_Operations.pow(p, n) - 1 - find_member_degree();
    int out_member = Numbers.Number_Operations.fx_pow(forming_element, fx, reversed_degree, p, n);
    return out_member;
}
public bool check_if_forming(int member)
{
    bool is_forming = true;
    int current_member_powered = member;
    int degree = 1;

```

```

int max_degree = Numbers.Number_Operations.pow(p, n);

while (current_member_powered != 1)
{
    current_member_powered = Numbers.Number_Operations.mul(current_member_powered, member, p);
    current_member_powered = Numbers.Number_Operations.normalizer(current_member_powered, fx, n,
p);
    degree++;
    if (degree > max_degree)
    {
        throw new Exceptions.NoValidNumbersFound();
    }
}
if (degree != Numbers.Number_Operations.pow(p, n) - 1)
{
    is_forming = false;
}
return is_forming;
}
private int find_forming_element()
{
    int forming_element = 0;
    int desired_degree = Numbers.Number_Operations.pow(p, n) - 1;           // Максимальный порядок
соответствует числу элементов
    for (int current_element = 1; current_element <= desired_degree; current_element++)
    {
        if (check_if_forming(current_element))
        {
            forming_element = current_element;
            break;
        }
    }
    return forming_element;
}
// Если на входе 0, то возвращает первый образующий элемент,
// Если 1 - то второй и т.д.
public int find_forming_element(int random)
{
    int random_steps = 0;
    int forming_element = 0;
    int desired_degree = Numbers.Number_Operations.pow(p, n) - 1;           // Максимальный порядок
соответствует числу элементов
    for (int current_element = 1; current_element <= desired_degree; current_element++)
    {
        if (check_if_forming(current_element))
        {
            forming_element = current_element;
            if (random_steps == random)
            {
                break;
            }
            random_steps++;
        }
    }
    return forming_element;
}
public int find_member_degree()
{
    if (member == 0)

```

```

    {
        return 0;
    }
    int current_member = 1;
    int degree = 0;
    while (current_member != member)
    {
        current_member = Numbers.Number_Operations.mul(current_member, forming_element, p);
        current_member = Numbers.Number_Operations.normalizer(current_member, fx, n, p);
        degree++;
    }
    return degree;
}

public int find_member_degree(int suggested_forming_element)
{
    if (!check_if_forming(suggested_forming_element))
    {
        throw new Exceptions.CurrentMemberIsNotForming();
    }
    int current_member = 1;
    int degree = 0;
    while (current_member != member)
    {
        current_member = Numbers.Number_Operations.mul(current_member, suggested_forming_element, p);
        current_member = Numbers.Number_Operations.normalizer(current_member, fx, n, p);
        degree++;
    }
    return degree;
}

public int find_member_order()
{
    if (member == 0)
    {
        return 0;
    }
    int degree = find_member_degree(forming_element);
    int max_degree = Numbers.Number_Operations.pow(p, n) - 1;
    int GCD = Numbers.Number_Operations.greatest_common_divisor(degree, max_degree);

    return max_degree / GCD;
}

public static Galois_Field operator +(Galois_Field a, Galois_Field b)
{
    if (a.get_p() != b.get_p()
        || a.get_n() != b.get_n()
        || a.get_fx() != b.get_fx())
    {
        throw new Exceptions.IncompatibleGaloisFieldMembers();
    }

    int p = a.get_p();
    int n = a.get_n();
    int fx = a.get_fx();
    int forming_element = a.get_forming_element();

    int member = Numbers.Number_Operations.plus(a.get_member(), b.get_member(), p);

    Galois_Field c = new Galois_Field(p, n, member, fx, forming_element);

```

```

        return c;
    }
    public static Galois_Field operator -(Galois_Field a, Galois_Field b)
    {
        if (a.get_p() != b.get_p()
            || a.get_n() != b.get_n()
            || a.get_fx() != b.get_fx())
        {
            throw new Exceptions.IncompatibleGaloisFieldMembers();
        }

        int p = a.get_p();
        int n = a.get_n();
        int fx = a.get_fx();
        int forming_element = a.get_forming_element();

        int member = Numbers.Number_Operations.minus(a.get_member(), b.get_member(), p);

        Galois_Field c = new Galois_Field(p, n, member, fx, forming_element);

        return c;
    }
    public static Galois_Field operator *(Galois_Field a, Galois_Field b)
    {
        if (a.get_p() != b.get_p()
            || a.get_n() != b.get_n()
            || a.get_fx() != b.get_fx())
        {
            throw new Exceptions.IncompatibleGaloisFieldMembers();
        }

        int member = Numbers.Number_Operations.mul(a.get_member(), b.get_member(), a.get_p());
        member = Numbers.Number_Operations.normalizer(member, a.get_fx(), a.get_n(), a.get_p());

        Galois_Field c = new Galois_Field(a.get_p(), a.get_n(), member, a.get_fx(), a.get_forming_element());

        return c;
    }
    public static Galois_Field operator /(Galois_Field a, Galois_Field b)
    {
        if (a.get_p() != b.get_p()
            || a.get_n() != b.get_n()
            || a.get_fx() != b.get_fx())
        {
            throw new Exceptions.IncompatibleGaloisFieldMembers();
        }

        int member = Numbers.Number_Operations.mul(a.get_member(), b.find_reversed_member(), a.get_p());
        member = Numbers.Number_Operations.normalizer(member, a.get_fx(), a.get_n(), a.get_p());

        Galois_Field c = new Galois_Field(a.get_p(), a.get_n(), member, a.get_fx(), a.get_forming_element());

        return c;
    }
    private string visualizer(int current, int base_number)
    {
        string out_text = "";
        int max_power = Numbers.Number_Operations.get_power(current, base_number);

```



```

for (int i = max_power; i > 0; i--)
{
    int digit = Numbers.Number_Operations.get_coef_on_place(current, p, i);
    if (digit == 0)
    {
        continue;
    }
    else if (i != max_power)
    {
        out_text += " + ";
    }
    if (i != 1)
    {
        if (digit == 1)
        {
            out_text += string.Format("X^{0}", i);
        }
        else
        {
            out_text += string.Format("{0}*X^{1}", digit, i);
        }
    }
    else
    {
        if (digit == 1)
        {
            out_text += "X";
        }
        else
        {
            out_text += string.Format("{0}*X", digit, i);
        }
    }
}
int last_digit = Numbers.Number_Operations.get_coef_on_place(current, p, 0);
if (last_digit != 0)
{
    if (max_power == 0)
    {
        out_text += string.Format("{0}", last_digit);
    }
    else
    {
        out_text += string.Format(" + {0}", last_digit);
    }
}

return out_text;
}
public string visualize_member()
{
    return visualizer(member, p);
}
public string visualize_irreducible_fx()
{
    return visualizer(fx, p);
}
public string visualize_forming_element()

```

```

        {
            return visualizer(forming_element, p);
        }
    }
}

namespace Numbers
{
    public class Number_Operations
    {
        public static int greatest_common_divisor(int a, int b)
        {
            if (a == b)
            {
                return a;
            }
            if (a < b)
            {
                int c = a;
                a = b;
                b = c;
            }

            int r;
            do
            {
                r = a % b;
                a = b;
                b = r;
            }
            while(r != 0);
            return a;
        }

        public static int get_power(int number, int base_number)
        {
            int power = 0;
            int current_powered_number = 1;
            do
            {
                current_powered_number *= base_number;
                power++;
            }
            while (number >= current_powered_number);

            power--;

            return power;
        }

        public static int pow(int number, int degree)
        {
            if (degree == 0)
            {
                return 1;
            }
            else if (degree > 0)
            {
                int current_number = 1;
                for (int i = 0; i < degree; i++)
                {

```

```

        current_number *= number;
    }
    return current_number;
}
else
{
    return 0;
}
}
public static int fx_pow(int member, int fx, int degree, int base_number, int n_limit)
{
    if (degree == 0)
    {
        return 1;
    }
    else if (degree == 1)
    {
        return member;
    }
    else
    {
        int new_member = 1;
        for (int i = 0; i < degree; i++)
        {
            new_member = mul(new_member, member, base_number);
            new_member = normalizer(new_member, fx, n_limit, base_number);
        }

        return new_member;
    }
}
// Возвращает позицию от 0 до max_row, а дальше нули.
public static int get_coef_on_place(int number, int base_number, int position)
{
    int current_coef = 0;
    for (int i = 0; i <= position; i++)
    {
        current_coef = number % base_number;
        number /= base_number;
    }
    return current_coef;
}
public static int set_coef_on_place(int number, int base_number, int position, int replace_coef)
{
    int number_copy = number;
    int current_coef = 0;
    for (int i = 0; i <= position; i++)
    {
        current_coef = number % base_number;
        number /= base_number;
    }
    number_copy = number_copy
        - (current_coef % base_number) * pow(base_number, position)
        + (replace_coef % base_number) * pow(base_number, position);

    return number_copy;
}
public static int plus(int a, int b, int base_number)
{

```

```

int out_number = 0;
int remainder_a = 0;
int remainder_b = 0;
int degree = 0;
while (a != 0 || b != 0)
{
    remainder_a = a % base_number;
    remainder_b = b % base_number;

    a /= base_number;
    b /= base_number;

    int int_plus = (remainder_a + remainder_b) % base_number;
    out_number += int_plus * pow(base_number, degree);

    degree++;
}
return out_number;
}
public static int minus(int a, int b, int base_number)
{
    int out_number = 0;
    int remainder_a = 0;
    int remainder_b = 0;
    int degree = 0;
    while (a != 0 || b != 0)
    {
        remainder_a = a % base_number;
        remainder_b = b % base_number;

        a /= base_number;
        b /= base_number;

        int int_plus = (base_number + remainder_a - remainder_b) % base_number;
        out_number += int_plus * pow(base_number, degree);

        degree++;
    }
    return out_number;
}
public static int mul(int a, int b, int base_number)
{
    int a_size = get_power(a, base_number) + 1;
    int b_size = get_power(b, base_number) + 1;

    int[] a_digits = new int[a_size];
    int i = 0;
    while (a > 0)
    {
        a_digits[i] = a % base_number;
        a /= base_number;
        i++;
    }

    int[] b_digits = new int[b_size];
    int j = 0;
    while (b > 0)
    {
        b_digits[j] = b % base_number;

```

```

        b /= base_number;
        j++;
    }

    int[] resulted_digits = new int[a_size + b_size - 1];
    for (i = 0; i < a_size + b_size - 1; i++)
    {
        for (j = 0; j <= i; j++)
        {
            if (j < a_size && i - j < b_size)
            {
                int addable = a_digits[j] * b_digits[i - j] % base_number;
                resulted_digits[i] = Numbers.Number_Operations.plus(resulted_digits[i],
                    addable,
                    base_number);
            }
        }
    }

    int out_number = 0;
    int degree = 0;
    foreach (int digit in resulted_digits)
    {
        out_number += digit * pow(base_number, degree);
        degree++;
    }

    return out_number;
}
// Это деление на многочлен fx
//
// member - член поля Галуа
// fx - неприводимый многочлен (Например  $x^4 + x + 1$ , где  $n\_limit = 4$ )
// n_limit - ограничение степени
// base_number - основание p
public static int normalizer(int member, int fx, int n_limit, int base_number)
{
    int current_power = get_power(member, base_number);
    if (current_power < n_limit)
    {
        return member;
    }
    else
    {
        for (int i = current_power; i >= n_limit; i--)
        {
            int digit = get_coef_on_place(member, base_number, i);
            int multiplied_digit = mul(digit, fx, base_number);
            multiplied_digit = mul(multiplied_digit,
                pow(base_number, i - n_limit),
                base_number);

            member = minus(member,
                multiplied_digit,
                base_number);
        }
        return member;
    }
}
}

```

```

public static int[] polynomial_maker(int number, int base_number)
{
    int degree = Numbers.Number_Operations.get_power(number, base_number);
    int[] digits = new int[degree + 1];

    int i = 0;
    while (number != 0)
    {
        digits[i++] = number % base_number;
        number /= base_number;
    }

    return digits;
}
}

```

```
namespace Dictionaries
{
    public class Dictionaries
    {
        public static char[] dict_base2_degree5 = [' ', ',', '!', '?', '-', 'A', 'B', 'C', 'D',
            'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
            'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
            'Y', 'Z'];

        public static char[] dict_base2_degree6 = [' ', ',', '!', '?', '-', '(', ')', ';', ':',
            '_', '*', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
            'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R',
            'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b',
            'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
            'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
            'w', 'x', 'y', 'z'];

        public static char[] dict_base2_degree7 = [' ', ',', '!', '?', '-', '(', ')', ';', ':',
            'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
            'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
            'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd',
            'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
            'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x',
            'y', 'z', 'А', 'Б', 'В', 'Г', 'Д', 'Е', 'Ё', 'Ж',
            'З', 'И', 'Й', 'К', 'Л', 'М', 'Н', 'О', 'П', 'Р',
            'С', 'Т', 'У', 'Ф', 'Х', 'Ц', 'Ч', 'Ш', 'Щ', 'Ъ',
            'Ы', 'Ь', 'Э', 'Ю', 'Я', 'а', 'б', 'в', 'г', 'д',
            'е', 'ё', 'ж', 'з', 'и', 'й', 'к', 'л', 'м', 'н',
            'о', 'п', 'р', 'с', 'т', 'у', 'ф', 'х', 'ц', 'ч',
            'ш', 'щ', 'ъ', 'ы', 'ь', 'э', 'ю', 'я'];
    }
}
```

```
namespace Ciphers
{
    public class AffineCipher
    {
        private static int find_position_in_dict(char symbol, char[] dict)
        {
            int i = 0;
            foreach(char c in dict)
            {
```

```

        if (c == symbol)
        {
            return i;
        }
        else
        {
            i++;
        }
    }
    return -1;
}

public static string encrypt_default(string input, Galois_Field.Galois_Field key_a, Galois_Field.Galois_Field key_b)
{
    if (key_a.get_p() != key_b.get_p()
        || key_a.get_n() != key_b.get_n()
        || key_a.get_fx() != key_b.get_fx())
    {
        throw new Exceptions.IncompatibleGaloisFieldMembers();
    }

    string output = "";
    int p = key_a.get_p();
    int n = key_b.get_n();
    int fx = key_a.get_fx();
    int max_number = Numbers.Number_Operations.pow(p, n);

    foreach(char c in input)
    {
        int number = (int)c;
        if (number >= max_number)
        {
            throw new Exceptions.IncompatibleCharAndField();
        }
        Galois_Field.Galois_Field member = new Galois_Field.Galois_Field(p, n, number, fx);
        member = member * key_a + key_b;
        char member_char = (char)member.get_member();
        output += member_char;
    }
    return output;
}

public static string decrypt_default(string input, Galois_Field.Galois_Field key_a, Galois_Field.Galois_Field key_b)
{
    if (key_a.get_p() != key_b.get_p()
        || key_a.get_n() != key_b.get_n()
        || key_a.get_fx() != key_b.get_fx())
    {
        throw new Exceptions.IncompatibleGaloisFieldMembers();
    }

    string output = "";
    int p = key_a.get_p();
    int n = key_b.get_n();
    int fx = key_a.get_fx();
    int max_number = Numbers.Number_Operations.pow(p, n);

    foreach (char c in input)
    {
        int number = (int)c;
        if (number >= max_number)

```

```

        {
            throw new Exceptions.IncompatibleCharAndField();
        }
        Galois_Field.Galois_Field member = new Galois_Field.Galois_Field(p, n, number, fx);
        member = (member - key_b) / key_a;
        char member_char = (char)member.get_member();
        output += member_char;
    }
    return output;
}

public static string encrypt_with_dict(string input, Galois_Field.Galois_Field key_a, Galois_Field.Galois_Field
key_b, char[] dict)
{
    if (key_a.get_p() != key_b.get_p()
        || key_a.get_n() != key_b.get_n()
        || key_a.get_fx() != key_b.get_fx())
    {
        throw new Exceptions.IncompatibleGaloisFieldMembers();
    }

    string output = "";
    int p = key_a.get_p();
    int n = key_b.get_n();
    int fx = key_a.get_fx();

    int member_count = Numbers.Number_Operations.pow(p, n);
    if (member_count != dict.Length)
    {
        throw new Exceptions.WrongDictForField();
    }

    foreach (char c in input)
    {
        int number = find_position_in_dict(c, dict);
        if (number == -1)
        {
            throw new Exceptions.NonexistingCharInDict();
        }

        Galois_Field.Galois_Field member = new Galois_Field.Galois_Field(p, n, number, fx);
        member = member * key_a + key_b;

        output += dict[member.get_member()];
    }
    return output;
}

public static string decrypt_with_dict(string input, Galois_Field.Galois_Field key_a, Galois_Field.Galois_Field
key_b, char[] dict)
{
    if (key_a.get_p() != key_b.get_p()
        || key_a.get_n() != key_b.get_n()
        || key_a.get_fx() != key_b.get_fx())
    {
        throw new Exceptions.IncompatibleGaloisFieldMembers();
    }

    string output = "";
    int p = key_a.get_p();
    int n = key_b.get_n();

```



```

int fx = key_a.get_fx();

int member_count = Numbers.Number_Operations.pow(p, n);
if (member_count != dict.Length)
{
    throw new Exceptions.WrongDictForField();
}

foreach (char c in input)
{
    int number = find_position_in_dict(c, dict);
    if (number == -1)
    {
        throw new Exceptions.NonexistingCharInDict();
    }

    Galois_Field.Galois_Field member = new Galois_Field.Galois_Field(p, n, number, fx);
    member = (member - key_b) / key_a;

    output += dict[member.get_member()];
}
return output;
}
}

namespace ConsoleProgram
{
    public class ConsoleProgram
    {
        public static Galois_Field.Galois_Field get_galois_field_member()
        {
            Galois_Field.Galois_Field Element;

            while (true)
            {
                try
                {
                    Console.WriteLine("Выбран инструмент для работы с полем Галуа!");
                    Console.WriteLine("Введите основные параметры поля Галуа:\n");

                    Console.WriteLine("Введите основание p: ");
                    int p = Int32.Parse(Console.ReadLine());
                    if (p < 2)
                    {
                        throw new Exceptions.WrongInput();
                    }

                    Console.WriteLine("Введите степень n: ");
                    int n = Int32.Parse(Console.ReadLine());
                    if (n < 2)
                    {
                        throw new Exceptions.WrongInput();
                    }

                    Console.WriteLine("Подставьте в многочлен поля X={0} и впишите результат: ", p);
                    int member = Int32.Parse(Console.ReadLine());
                    if (member < 0)

```

```

    {
        throw new Exceptions.WrongInput();
    }

    Console.WriteLine("Аналогично впишите число для неприводимого многочлена\nЕсли нужно выбрать
автоматически, то впишите 0: ");
    int fx = Int32.Parse(Console.ReadLine());
    if (fx < Numbers.Number_Operations.pow(p, n) && fx != 0)
    {
        throw new Exceptions.WrongInput();
    }

    Element = new Galois_Field.Galois_Field(p, n, member);

    if (fx != 0)
    {
        if (Element.check_if_fx_irreducible(fx))
        {
            Element = new Galois_Field.Galois_Field(p, n, member, fx);
        }
        else
        {
            throw new Exceptions.WrongIrreduciblePolynomial();
        }
    }

    Console.WriteLine("Точно так же введите образующий элемент\nЕсли без него, то 0: ");
    int forming_element = Int32.Parse(Console.ReadLine());
    if (forming_element < 1 && forming_element != 0)
    {
        throw new Exceptions.WrongInput();
    }

    if (forming_element != 0)
    {
        if (Element.check_if_forming(forming_element))
        {
            Element = new Galois_Field.Galois_Field(p, n, member, Element.get_fx(), forming_element);
        }
        else
        {
            throw new Exceptions.CurrentMemberIsNotForming();
        }
    }

}
catch (WrongIrreduciblePolynomial)
{
    show_wrong_irreducible_polynomial_error();

    continue;
}
catch (CurrentMemberIsNotForming)
{
    show_current_member_is_not_forming_error();

    continue;
}
catch

```

```

        {
            show_wrong_input_error();

            continue;
        }

        break;
    }

    return Element;
}
public static Galois_Field.Galois_Field form_element_based_on_existing(Galois_Field.Galois_Field existing)
{
    Console.WriteLine("Создание элемента поля Галуа на основе имеющегося.\n");
    Console.Write("Введите элемент, представленный в виде числа: ");
    int member = Int32.Parse(Console.ReadLine());
    Console.WriteLine("\n");

    return new Galois_Field.Galois_Field(existing.get_p(),
        existing.get_n(),
        member,
        existing.get_fx(),
        existing.get_forming_element());
}
public static void show_wrong_input_error()
{
    Console.Clear();

    MessageBox.Show(text: "Введено неправильное значение!",
        caption: "Ошибка",
        buttons: MessageBoxButtons.OK,
        icon: MessageBoxIcon.Error);
}
public static void show_wrong_irreducible_polynomial_error()
{
    Console.Clear();

    MessageBox.Show(text: "Введён приводимый многочлен!",
        caption: "Ошибка",
        buttons: MessageBoxButtons.OK,
        icon: MessageBoxIcon.Error);
}
public static void show_current_member_is_not_forming_error()
{
    Console.Clear();

    MessageBox.Show(text: "Введён не образующий элемент!",
        caption: "Ошибка",
        buttons: MessageBoxButtons.OK,
        icon: MessageBoxIcon.Error);
}
public ConsoleProgram()
{
    int chosen_program;
    while (true)
    {
        try
        {

```

```

        Console.WriteLine("Выберите режим работы программы:");
        Console.WriteLine(" 1) Инструмент для работы с полем Галуа;");
        Console.WriteLine(" 2) Инструмент для шифрования;");
        Console.WriteLine(" 3) Прекратить.\n");
        Console.Write("Выбранный режим: ");

        chosen_program = Int32.Parse(Console.ReadLine());
        if (chosen_program != 1 && chosen_program != 2 && chosen_program != 3)
        {
            throw new Exceptions.WrongInput();
        }
        Console.Clear();
        break;
    }
    catch
    {
        show_wrong_input_error();

        continue;
    }
}

if (chosen_program == 1)
{
    GaloisFieldTool tool = new GaloisFieldTool();
}
if (chosen_program == 2)
{
    CipherTool tool = new CipherTool();
}
}
public class GaloisFieldTool
{
    public GaloisFieldTool()
    {
        Galois_Field.Galois_Field member_a = get_galois_field_member();
        Galois_Field.Galois_Field member_b = null;

        Console.Clear();
        Console.WriteLine("Отлично! Вы добавили элемент в поле Галуа!");

        int chosen_program;

        while (true)
        {
            while (true)
            {
                try
                {
                    Console.WriteLine("\nВизуализация элемента A: " + member_a.visualize_member());
                    Console.WriteLine("В виде числа: " + member_a.get_member() + "\n");
                    if (member_b != null)
                    {
                        Console.WriteLine("Визуализация элемента B: " + member_b.visualize_member());
                        Console.WriteLine("В виде числа: " + member_b.get_member() + "\n");
                    }
                }
                Console.WriteLine("Выберите дальнейшие действия:");
                Console.WriteLine(" 1) Вывести образующий элемент;");
                Console.WriteLine(" 2) Вывести неприводимый многочлен;");
            }
        }
    }
}

```

```

        Console.WriteLine(" 3) Вычислить степень элемента;");
        Console.WriteLine(" 4) Вычислить порядок элемента;");
        Console.WriteLine(" 5) Найти обратный элемент;");
        Console.WriteLine(" 6) Возвести в степень;");
        Console.WriteLine(" 7) Изменить A;");
        Console.WriteLine(" 8) Изменить B;");
        Console.WriteLine(" 9) + B;");
        Console.WriteLine(" 10) - B;");
        Console.WriteLine(" 11) * B;");
        Console.WriteLine(" 12) / B;");
        Console.WriteLine(" 13) Выйти в главное меню.\n");
        Console.Write("Выбранный режим: ");

        chosen_program = Int32.Parse(Console.ReadLine());
        if (chosen_program < 1 || chosen_program > 13)
        {
            throw new Exceptions.WrongInput();
        }
        Console.Clear();
        break;

    }
    catch
    {
        show_wrong_input_error();

        Console.Clear();
        continue;
    }
}
bool want_to_restart = false;
switch (chosen_program)
{
    case 1:
        Console.WriteLine("Образующий элемент: " + member_a.visualize_forming_element());
        Console.WriteLine("В виде числа: " + member_a.get_forming_element());
        break;
    case 2:
        Console.WriteLine("Неприводимый многочлен: " + member_a.visualize_irreducible_fx());
        Console.WriteLine("В виде числа: " + member_a.get_fx());
        break;
    case 3:
        Console.WriteLine("Степень данного элемента: " + member_a.find_member_degree());
        break;
    case 4:
        Console.WriteLine("Порядок данного элемента: {0}\nМаксимальный порядок: {1}",
            member_a.find_member_order(),
            Numbers.Number_Operations.pow(member_a.get_p(), member_a.get_n() - 1);
        break;
    case 5:
        Galois_Field.Galois_Field reversed = new Galois_Field.Galois_Field(member_a.get_p(),
            member_a.get_n(),
            member_a.find_reversed_member(),
            member_a.get_fx(),
            member_a.get_forming_element());

        Console.WriteLine("Обратный элемент: " + reversed.visualize_member());
        Console.WriteLine("В виде числа: " + reversed.get_member());
        break;
}

```

```

case 6:
    while (true)
    {
        try
        {
            Console.WriteLine("Элемент A: " + member_a.visualize_member() + "\n");
            Console.Write("Введите степень: ");
            int degree = Int32.Parse(Console.ReadLine());
            degree = degree % (Numbers.Number_Operations.pow(member_a.get_p(), member_a.get_n())
- 1);

            if (degree == 0)
            {
                Console.WriteLine("\nИтог: 1");
            }
            if (degree > 0)
            {
                Galois_Field.Galois_Field out_member = new Galois_Field.Galois_Field(member_a.get_p(),
                    member_a.get_n(),
                    1,
                    member_a.get_fx(),
                    member_a.get_forming_element());

                for (int i = 0; i < degree; i++)
                {
                    out_member *= member_a;
                }

                Console.WriteLine("\nИтог: " + out_member.visualize_member());
                Console.WriteLine("В виде числа: " + out_member.get_member());
            }
            if (degree < 0)
            {
                Galois_Field.Galois_Field out_member = new Galois_Field.Galois_Field(member_a.get_p(),
                    member_a.get_n(),
                    1,
                    member_a.get_fx(),
                    member_a.get_forming_element());

                for (int i = 0; i > degree; i--)
                {
                    out_member /= member_a;
                }

                Console.WriteLine("\nИтог: " + out_member.visualize_member());
                Console.WriteLine("В виде числа: " + out_member.get_member());
            }

            break;
        }
        catch
        {
            show_wrong_input_error();

            Console.Clear();
        }
    }
    break;
case 7:

```

```

        member_a = form_element_based_on_existing(member_a);
        Console.WriteLine("Новый элемент A: " + member_a.visualize_member());
        Console.WriteLine("В виде числа: " + member_a.get_member());
        break;
    case 8:
        member_b = form_element_based_on_existing(member_a);
        Console.WriteLine("Новый элемент B: " + member_b.visualize_member());
        Console.WriteLine("В виде числа: " + member_b.get_member());
        break;
    case 9:
        if (member_b == null) { member_b = form_element_based_on_existing(member_a); }
        Galois_Field.Galois_Field result9 = member_a + member_b;
        Console.WriteLine("Сумма A + B: " + result9.visualize_member());
        Console.WriteLine("В виде числа: " + result9.get_member());
        break;
    case 10:
        if (member_b == null) { member_b = form_element_based_on_existing(member_a); }
        Galois_Field.Galois_Field result10 = member_a - member_b;
        Console.WriteLine("Разность A - B: " + result10.visualize_member());
        Console.WriteLine("В виде числа: " + result10.get_member());
        break;
    case 11:
        if (member_b == null) { member_b = form_element_based_on_existing(member_a); }
        Galois_Field.Galois_Field result11 = member_a * member_b;
        Console.WriteLine("Произведение A * B: " + result11.visualize_member());
        Console.WriteLine("В виде числа: " + result11.get_member());
        break;
    case 12:
        if (member_b == null) { member_b = form_element_based_on_existing(member_a); }
        Galois_Field.Galois_Field result12 = member_a / member_b;
        Console.WriteLine("Частное A / B: " + result12.visualize_member());
        Console.WriteLine("В виде числа: " + result12.get_member());
        break;
    case 13:
        want_to_restart = true;
        break;
    }
    if (!want_to_restart)
    {
        Console.Write("\nПовторить? Нажмите Enter");
        string stop_string = Console.ReadLine();
        if (!stop_string.Equals(""))
        {
            break;
        }
        Console.Clear();
    }
    else
    {
        break;
    }
}
Console.Clear();
ConsoleProgram program = new ConsoleProgram();

}
}
public class CipherTool
{

```

```

public CipherTool()
{
    int chosen_dict;
    char[] dict = null;
    string output_chosen_dict = "Вы выбрали ";

    int n = 0;
    int p = 0;
    Galois_Field.Galois_Field member_a, member_b;
    int number_a, number_b;

    while (true)
    {
        try
        {
            Console.Clear();
            Console.WriteLine("Выберите словарь:");
            Console.WriteLine(" 1) Default;");
            Console.WriteLine(" 2) dict_base2_degree5;");
            Console.WriteLine(" 3) dict_base2_degree6;");
            Console.WriteLine(" 4) dict_base2_degree7;\n");
            Console.Write("Выбранный режим: ");

            chosen_dict = Int32.Parse(Console.ReadLine());
            if (chosen_dict != 1 && chosen_dict != 2 && chosen_dict != 3 && chosen_dict != 4)
            {
                throw new Exceptions.WrongInput();
            }
            Console.Clear();

            if (chosen_dict == 1)
            {
                Console.WriteLine("Введите необходимые параметры поля:");
                Console.Write(" p = ");
                p = Int32.Parse(Console.ReadLine());
                Console.Write(" n = ");
                n = Int32.Parse(Console.ReadLine());
                output_chosen_dict += "Default";
            }
            if (chosen_dict == 2)
            {
                p = 2;
                n = 5;
                output_chosen_dict += "dict_base2_degree5";
                dict = Dictionaries.Dictionaries.dict_base2_degree5;
            }
            if (chosen_dict == 3)
            {
                p = 2;
                n = 6;
                output_chosen_dict += "dict_base2_degree6";
                dict = Dictionaries.Dictionaries.dict_base2_degree6;
            }
            if (chosen_dict == 4)
            {
                p = 2;
                n = 7;
                output_chosen_dict += "dict_base2_degree7";
                dict = Dictionaries.Dictionaries.dict_base2_degree7;
            }
        }
        catch { }
    }
}

```



```

    }

    Console.WriteLine("Введите ключ А в виде числа (> 0): ");
    number_a = Int32.Parse(Console.ReadLine());
    if (number_a < 1)
    {
        throw new Exceptions.WrongInput();
    }

    Console.WriteLine("Введите ключ В в виде числа (>= 0): ");
    number_b = Int32.Parse(Console.ReadLine());
    if (number_b < 0)
    {
        throw new Exceptions.WrongInput();
    }
    Console.Clear();
    break;
}
catch
{
    ConsoleProgram.show_wrong_input_error();

    continue;
}
}

int chosen_program;
member_a = new Galois_Field.Galois_Field(p, n, number_a);
member_b = new Galois_Field.Galois_Field(p, n, number_b);

while (true)
{
    try
    {
        bool want_to_restart = false;
        Console.WriteLine(output_chosen_dict);
        Console.WriteLine("Выберите дальнейшие действия:");
        Console.WriteLine(" 1) Зашифровать;");
        Console.WriteLine(" 2) Расшифровать;");
        Console.WriteLine(" 3) Тест шифровальщика;");
        Console.WriteLine(" 4) Выйти в главное меню.\n");
        Console.WriteLine("Выбранный режим работы: ");

        chosen_program = Int32.Parse(Console.ReadLine());
        if (chosen_dict != 1 && chosen_dict != 2 && chosen_dict != 3 && chosen_dict != 4)
        {
            throw new Exceptions.WrongInput();
        }
        Console.Clear();

        if (chosen_program == 4)
        {
            break;
        }

        Console.WriteLine("Введите текст:");
        string input = Console.ReadLine();
        string output = "";
        Console.Clear();
    }
    catch
    {
        ConsoleProgram.show_wrong_input_error();
        continue;
    }
}

```

```

Console.WriteLine("Используемый словарь: " + output_chosen_dict.Remove(0, 11));
if (chosen_program == 1)
{
    Console.WriteLine("Введённый текст: \"\" + input + "\"");
    if (chosen_dict == 1)
    {
        output = Ciphers.AffineCipher.encrypt_default(input, member_a, member_b);
    }
    else
    {
        output = Ciphers.AffineCipher.encrypt_with_dict(input, member_a, member_b, dict);
    }
    Console.WriteLine("Шифр-текст: \"\" + output + "\"");
}
if (chosen_program == 2)
{
    Console.WriteLine("Введённый шифр-текст: \"\" + input + "\"");
    if (chosen_dict == 1)
    {
        output = Ciphers.AffineCipher.decrypt_default(input, member_a, member_b);
    }
    else
    {
        output = Ciphers.AffineCipher.decrypt_with_dict(input, member_a, member_b, dict);
    }
    Console.WriteLine("Исходный текст: \"\" + output + "\"");
}
if (chosen_program == 3)
{
    string output2;
    Console.WriteLine("\nТест работы программы:");
    Console.WriteLine("Введённый текст: \"\" + input + "\"");
    if (chosen_dict == 1)
    {
        output = Ciphers.AffineCipher.encrypt_default(input, member_a, member_b);
        Console.WriteLine("Шифр-текст: \"\" + output + "\"");
        output2 = Ciphers.AffineCipher.decrypt_default(output, member_a, member_b);
    }
    else
    {
        output = Ciphers.AffineCipher.encrypt_with_dict(input, member_a, member_b, dict);
        Console.WriteLine("Шифр-текст: \"\" + output + "\"");
        output2 = Ciphers.AffineCipher.decrypt_with_dict(output, member_a, member_b, dict);
    }
    Console.WriteLine("Расшифровка: \"\" + output2 + "\"");

    if (input.Equals(output2))
    {
        Console.WriteLine("\nПрограмма отработала верно!");
    }
    else
    {
        Console.WriteLine("\nЧто-то пошло не так...");
    }
}

Console.WriteLine("\nПовторить? Нажмите Enter");
string stop_string = Console.ReadLine();

```

```

        if (!stop_string.Equals(""))
        {
            break;
        }
        Console.Clear();
    }
    catch
    {
        ConsoleProgram.show_wrong_input_error();
        Console.Clear();
        continue;
    }
}
Console.Clear();
ConsoleProgram program = new ConsoleProgram();
}
}
}

namespace Program
{
    static class Program
    {
        static void Main()
        {
            Console.OutputEncoding = Encoding.Unicode;
            Console.InputEncoding = Encoding.Unicode;

            ConsoleProgram.ConsoleProgram program = new ConsoleProgram.ConsoleProgram();
        }
    }
}

```