

МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт (факультет) Институт информационных технологий

Кафедра Кафедра математического и программного
обеспечения ЭВМ

Задание на лабораторную работу №1

Дисциплина: С#-программирование

Темы:	<u>Классы, ассоциация (композиция, агрегация); поля, свойства и методы; статические поля, свойства и методы; операторы и перегрузка операторов; обработка строк и работа с массивами</u>
--------------	--

Среда разработки: Microsoft Visual Studio 2022

Язык программирования: С# 9.0

Тип проекта: Библиотека классов

ЗАДАНИЕ

Разработать библиотеку классов для работы со штрихкодами.

Библиотека позволяет:

- Формировать QR-код на основе текстовой информации;
- Выводить информацию по QR-коду;
- Кодировать текстовую информацию по 3 алгоритмам: цифровому буквенно-цифровому и универсальному побайтовому.

ТРЕБОВАНИЯ К РАЗРАБОТКЕ

1. Запрещается использовать обработку исключительных ситуаций и генерировать исключения.
2. Каждый класс должен быть оформлен в отдельном файле.
3. **Придерживайтесь принципа DRY (Don't repeat yourself).**
4. Обязательно наличие комментариев и xml-комментариев.

Создать в **решении**¹ новую библиотеку классов для QR-кода и консольное приложение терминала для отладки всего функционала по QR-коду и другого функционала в последующих лабораторных работах.

ЧАСТЬ 1

QR-код описать в виде класса, без виртуальных методов:

- Класс содержит в себе информацию:
 - заданный текст для кодирования,
 - версия QR-кода,
 - уровень коррекции,
 - маска,
 - строка кода («QR-код») в текстовом виде (см. часть 2, как формируется текстовый код) и примеры вывода кода (см. приложение 1);
- Информацию по тексту для кодирования можно изменять, следовательно необходимо обеспечить обновление и самого кода;
 - Информацию по строке кода изменять вне класса запрещено;
 - Способ вывода кода, задается один для всех объектов класса и может быть изменен в процессе работы программы:
 - Только исходный текст;
 - Только код;
 - Код + текст (желательно текст выравнивать по центру)
- При создании объекта класса достаточно передать информацию о тексте для кодирования, код же должен сформироваться автоматически (версия, маска, уровень коррекции ошибок);
- Перегрузить функцию «***ToString()***» для получения информации по тексту и коду, учитывая способ вывода самого кода;

¹ Решение – это сборник нескольких проектов, его название должно соответствовать общей теме, не стоит использовать названия привязанные к одному проекту

С#-программирование

- Класс не должен содержать дополнительных открытых методов и полей («Класс-Модель²»);
- Логика формирования QR-кода (в части 2 и 3) должна быть отдельно находится в «Классе-Сервисе»³;
- Перенесите все классы из приложения 2 в свою библиотеку, следуя принципу каждый класс или перечисление в отдельный файл с его именем.
- Прежде чем приступать к следующей части, проверьте сначала работоспособность класса, создав в решении консольное приложение (терминал), настройте зависимости – подключив библиотеку к консольному приложению.
- На данном этапе вместо строкового QR-кода консоль должна выводить просто исходный текст.
- Далее можно изучить статью: <https://habr.com/ru/articles/172525/>
- После изучения статьи, приступить к части 2.

² Задача класса в первую очередь хранить данные, бизнес-логика класса закрыта и в целом нужна только для формирования связей между данными в классе

³ Задача класса в первую очередь обрабатывать данные, все необходимые данные поступают в качестве параметров функций или методов

ЧАСТЬ 2

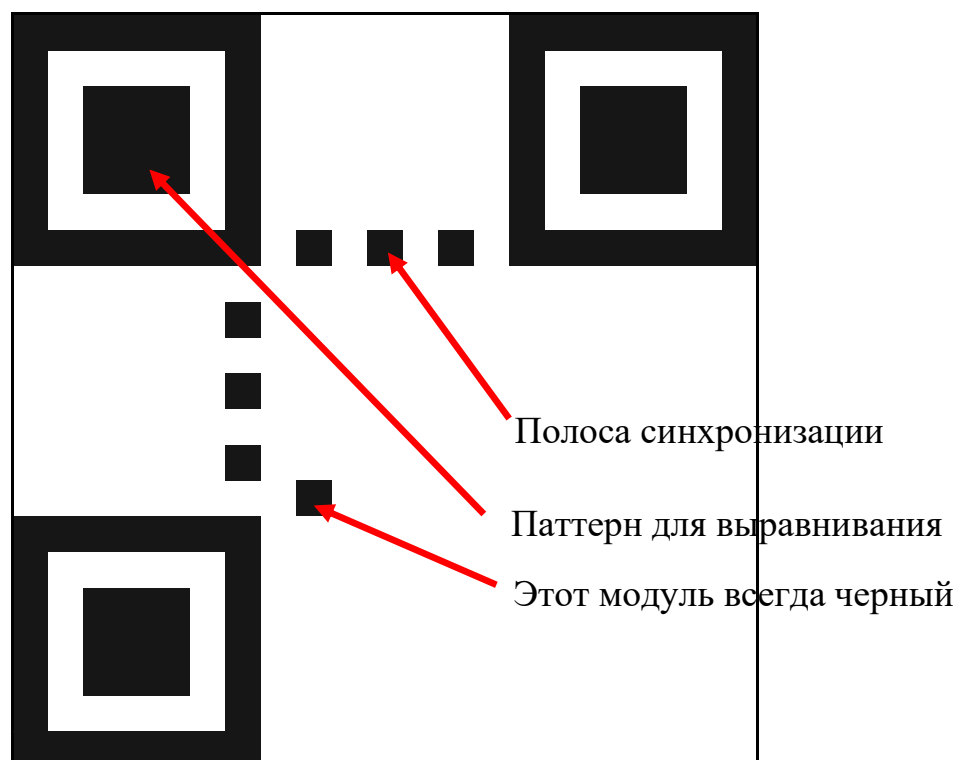
Для начала рассмотрим структуру штрихкода подробнее, для примера возьмем готовый сформированный текстовый код из строки «Example»:



Структура QR-кода:

Если обратить внимание, то данные кодируются в виде квадрата из пикселей (далее пиксель будем называть модулем, а сам квадрат представим в виде матрицы).

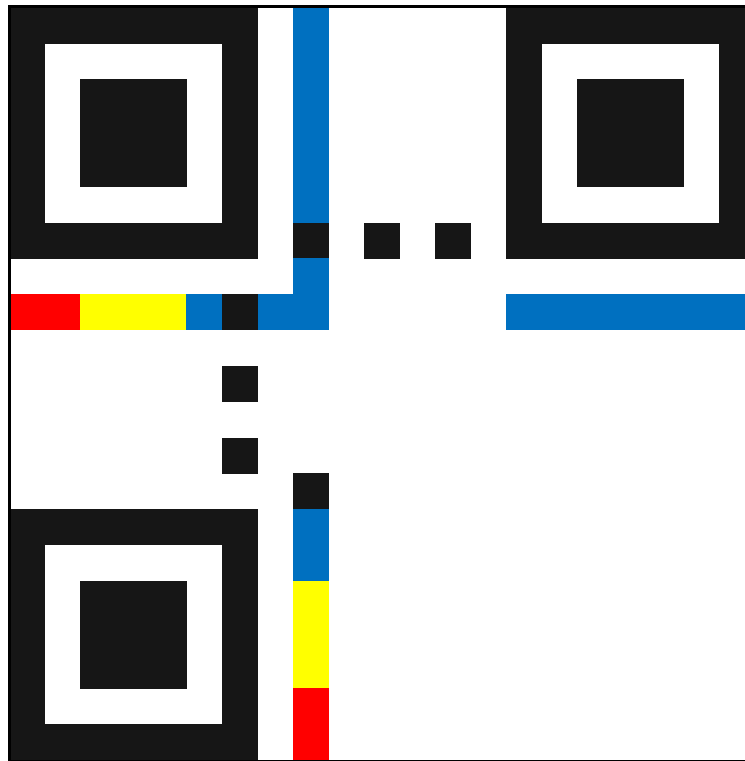
Исходные данные для кодирования в QR-коде занимают лишь часть модулей, помимо нее любой QR-код содержит 3 паттерна позиционирования, и полосы синхронизации:



С#-программирование

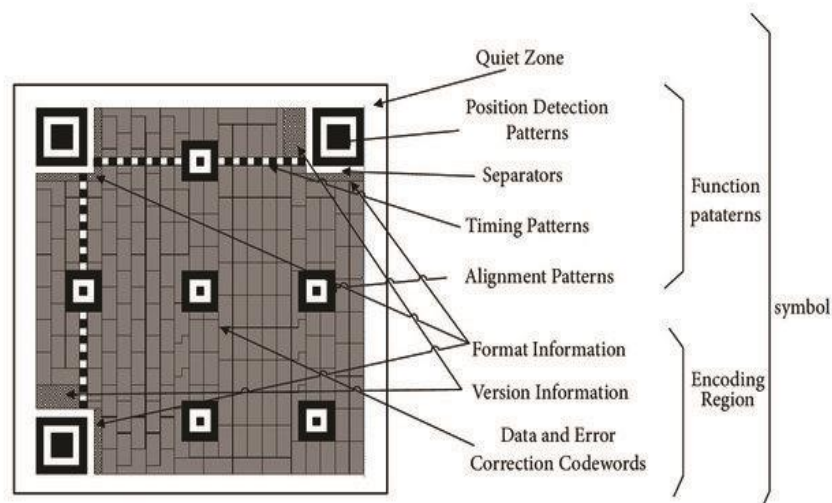
Следующая обязательная информация, дублируемая в двух местах:

- Уровень коррекции ошибок (УКО) два модуля;
- Маска в 3 модуля (всего 8 различных масок);
- Паттерн , уникальный для каждой комбинации УКО и Маски.



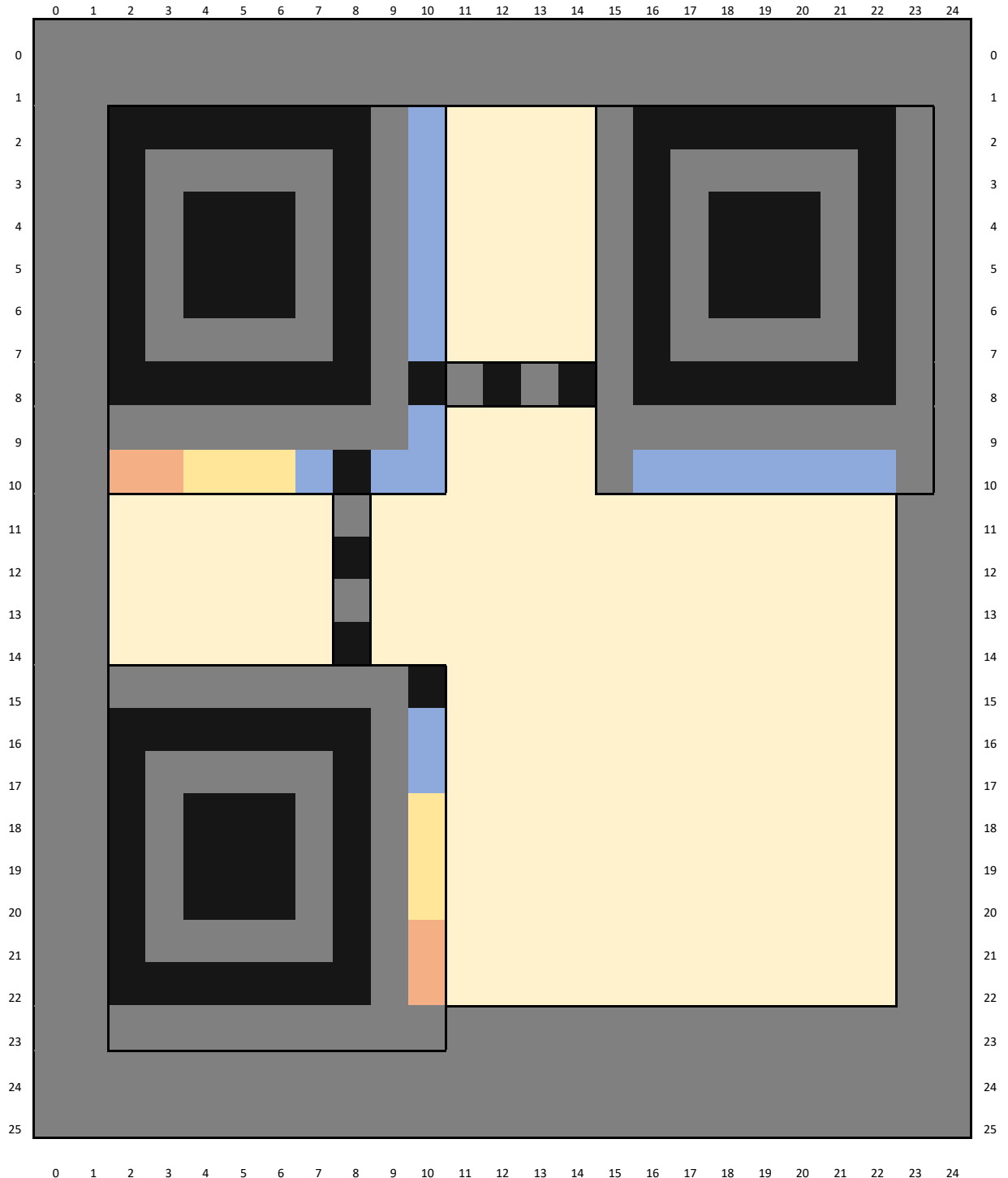
Дополнительно QR-код может содержать в себе:



- Версию, продублированную в двух местах;
- Выравнивающие паттерны.



С#-программирование

Из обязательных требований, вокруг штрих кода должна быть пустая рамка в два модуля, в итоге для первой версии QR-кода, размером 21 на 21 модуль (каждая следующая версия QR-кода увеличивает свой размер на 4 модуля) получаем матрицу размером 25 на 25 модулей следующего вида:



Получается, что на исходной матрице только в части  модули содержат в себе данные, остальное — это служебная информация:  всегда пусто, либо задействовано для паттернов.

С#-программирование

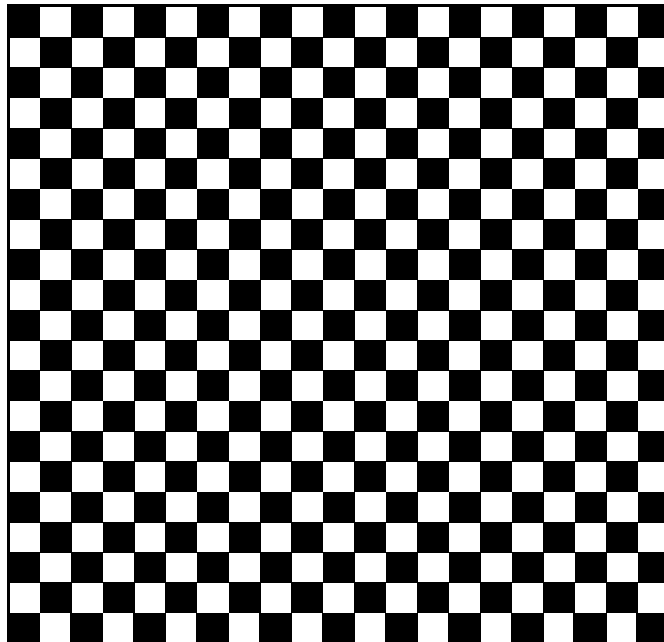
Однако, стоит иметь в виду, что каждая версия QR кода может быть распознана даже при повреждении данных, это обеспечивается 4-мя уровнями коррекции ошибок:

УКО	Представление в модулях	Название	Допустимое повреждение данных
L	Два белых	Low	7 %
M	Белый-черный	Medium	15 %
Q	Черный-белый	Quartile	25 %
H	Черный-черный	High	30 %

И да, когда вы видите на QR-коде в центре какой-нибудь рисунок, это, по сути, преднамеренное повреждение данных.

В части данных QR-кода, очевидно, что есть какая-то информация для восстановления при их повреждении, и да, эта информация идет сразу же после размещенных данных, называется она блоками коррекции.

Так же для лучшего сканирования QR-кода на размещенные данные накладывается маска, например:



Правило наложения маски простое, если модуль в маске черный, то модуль данных в этой позиции QR-кода инвертируется, если белый не трогается. Важно лишь только один момент, маска не применяется на служебную информацию, и на вспомогательные паттерны.

ЧАСТЬ 3

Текст преобразовывается в строковый код («QR-код») по следующему алгоритму (доп. см. приложение 1 с описанием используемых структур):

1. Формирование полного блока с данными + подходящий уровень коррекции ошибок + нужная версия QR-кода.
2. Формирование блоков с данными + байты коррекции.
3. Создание матрицы QR кода с лучшей маской.
4. Формирование полного многострочного QR-кода по сформированной матрице.

Этап 1

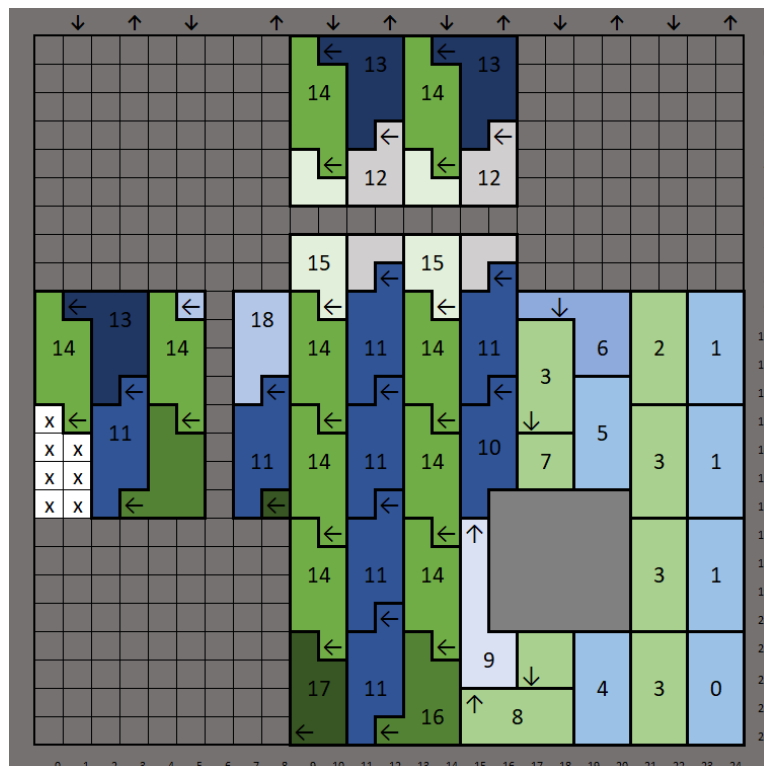
- Определяемся с текстовой информацией и выбираем алгоритм кодирования:
 - Цифровой - если текстовая строка содержит только цифры.
 - Буквенно-цифровой – если текстовая строка должна быть из заглавных латинских букв, цифр с специальных символов.
 - Байтовый, просто если надо закодировать любую информацию, например строку в формате UTF-8.
- Формируем на основе алгоритма кодирования строку из «0» и «1».
- Выбираем версию QR-кода и уровень коррекции, в прил. 1 найдем словарь с максимальным размером в битах, включая служебную информацию в зависимости от уровня коррекции и версии QR-кода.
- Формируем полную строку с данными:
 - Алгоритм кодирования +
 - Длина данных (для байтов – количество байт, для остального – количество символов)
 - Закодированная строка данных из «0» и «1», полученная ранее.
 -

Этап 2

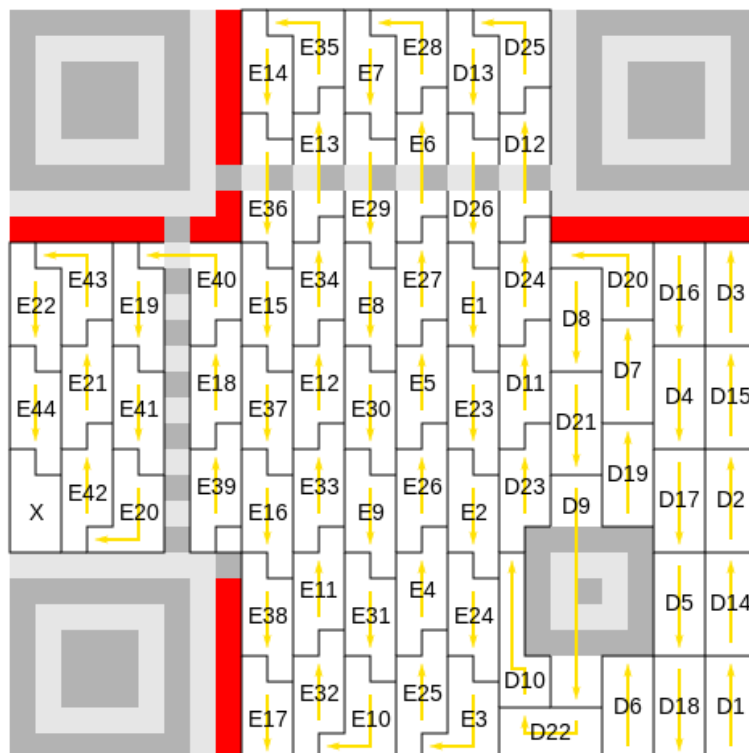
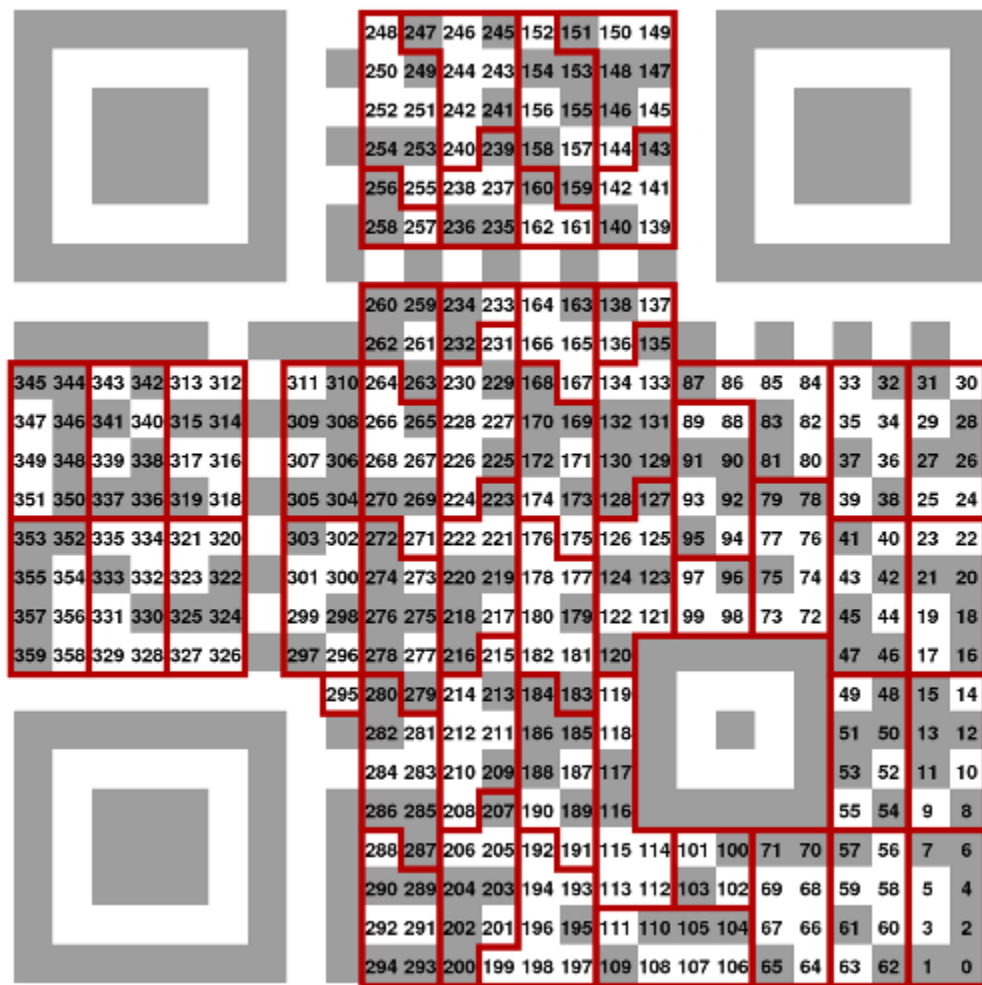
- Строка данных дополняется до максимального размера данных QR-кода последовательностью, чередуя «11101100» и «00010001».
- Если длина строки не кратна 8, в конце дополняем ее «0».
- Используя информацию по количеству блоков для коррекции, разбиваем строку с данными на блоки, их количество в зависимости от версии и уровня коррекции ошибок можно найти в прил. 1;
- Для каждого блока с данными вычисляется свой блок коррекции.
- После чего, блоки с данными и блоками коррекции объединяются в один большой блок, подробно алгоритм объединения приведен в статье.

Этап 3

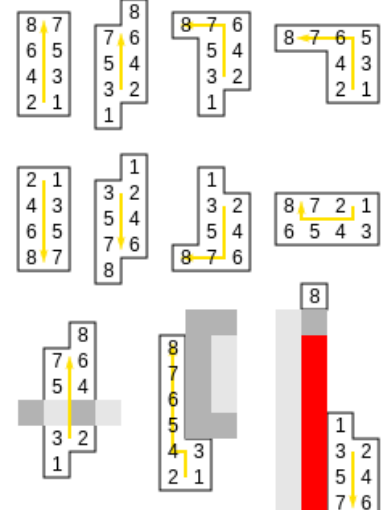
- Формируется матрица для QR-кода
- На матрице размещается вся служебная информация
- Далее на матрице размещаются данные, полученные из части 2 следующим образом (версия QR-кода - 2):



Другие примеры расположения данных:



Fixed Patterns (gray squares) and Format Info (red squares) are shown.
 D: Data, E: Error Correction, X: Unused
 Error Correction Level H is shown
 Block 1 Codewords: D1–D13, E1–E22
 Block 2 Codewords: D14–D26, E23–E44
 Message Data: D1–D13, D14–D26
 Bit order (1 is the most significant bit):



- Выбирается лучшая матрица по 8 маскам, оценивается соотношение белых и черных модулей 1 к 1, слишком длинные последовательности белых и черных модулей. Выбирается матрица с наименьшей оценкой, она же и пойдет на формирование строки.
- Однако, никто не мешает использовать определенную маску, в большинстве случаев QR-код будет нормально распознаваться.

Этап 4

- Преобразуем каждые 2 строки из «0» и «1» матрицы в строку модулей. Символ можно печатать, зажимая Alt и печатая на цифровой клавиатуре номера кода, отпуская Alt появится нужный символ.

Пара	Символ	Код
0 0	« »	32
0 1	«■»	220
1 0	«■»	223
1 1	«■»	219

- Фиксируем длину полученной строки, для выравнивания текста под QR-кодом.

Этап вывода готовой информации в перегруженной функции ToString():

- В зависимости от способа вывода могут быть варианты:
 - Только текст;
 - Только QR-код из Этапа 2;
 - Все вместе, сначала QR-код, под ним текст.
- По умолчанию достаточно выводить QR-код.

Пример:

Съешь ещё этих мягких французских булок, да выпей же чаю
V8 Binary Q M3



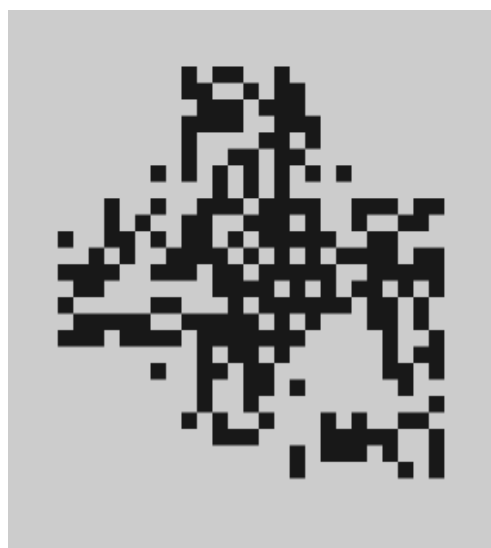
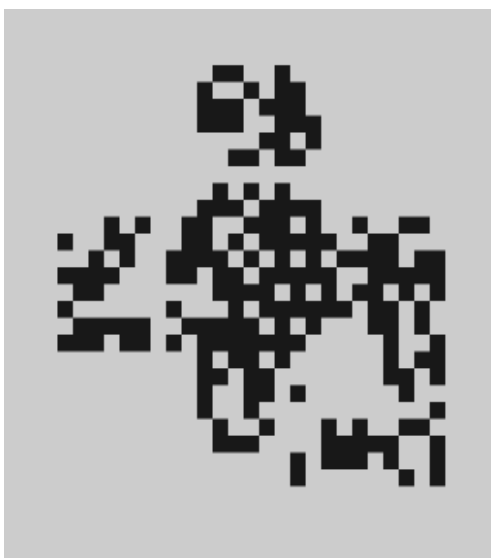
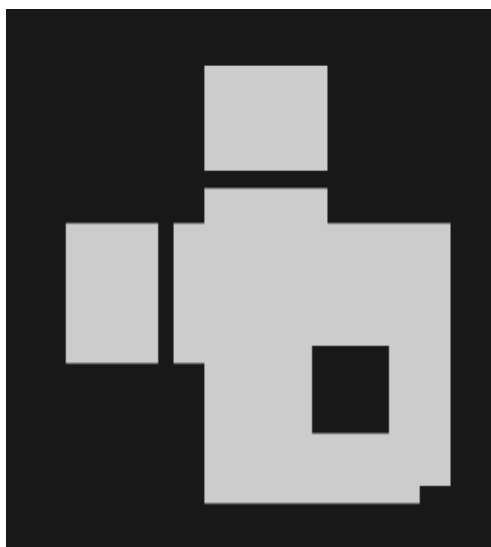
Сгенерированная строка (инвертированная для сравнения):



Hello world!
V1 Binary M M0



V2 AlphaNumeric H M011



Используемые структуры данных

Файл **QrCodeType.cs**

```

/// <summary>
///     Формат вывода QR-кода, должен
///     быть в отдельном файле
/// </summary>
public enum QrCodeType
{
    /// <summary>
    ///     Текстовая информация
    /// </summary>
    Text,

    /// <summary>
    ///     QR-код
    /// </summary>
    QrCode,
    /// <summary>
    ///     Полная информация: QR-код
    ///     + текст
    /// </summary>
    Full
}

```

Файл **EccLevel.cs**

```

/// <summary>
/// Error Correction Level
/// </summary>
public enum EccLevel : byte
{
    /// <summary>
    /// 7 % Low 11
    /// </summary>
    L = 0,
    /// <summary>
    /// 15 % Medium 10
    /// </summary>
    M = 1,
    /// <summary>
    /// 25 % Quartile 01
    /// </summary>
    Q = 2,
    /// <summary>
    /// 30 % High 00
    /// </summary>
    H = 3
}

```

Файл **EncodingMode.cs**

```

/// <summary>
/// QR Code encoding mode
/// </summary>
public enum EncodingMode : byte
{
    /// <summary>
    /// Numbers only
    /// </summary>
    Numeric,
    /// <summary>
    /// Caps english + numbers
    /// </summary>
    AlphaNumeric,
    /// <summary>
    /// Everything
    /// </summary>
    Binary,
    /// <summary>
    /// だってばよ
    /// </summary>
    Kanji
}

```

Файл **Pair.cs**

```

public record Pair(int X, int Y)
{
    public static implicit operator Pair((int x, int y) pos) => new(pos.x, pos.y);
}

```

Файл **QrCodeData.cs**

```

public record QrCodeData
{
    public QR Version {get; init;}
    public EccLevel CorrectionLevel {get; init;}
    public string Data {get; init;}
}

```

Файл Mask.cs

Файл QR.cs

Файл QrCodeBuilder.cs

15

C#-программирование

```

/// Активный модуль
/// </summary>
private const byte ACTIVE = 1;

/// <summary>
/// Неактивный модуль
/// </summary>
private const byte ZERO = 0;

/// <summary>
/// Недопустимый параметр
/// </summary>
private const byte NA = 0;

/// <summary>
/// Magic
/// </summary>
private static string Magic(this
List<byte[]> a, bool b)
{
    var sb = new StringBuilder();
    var length = a.Count % 2 == 1 ?
a.Count + 1 : a.Count;

    for (int row = 0; row < length;
row+=2)
    {
        for(int column = 0; column <
a[0].Length; column++)
        {
            byte scanModule1 =
a[row][column];
            byte scanModule2 = row <
a.Count - 1 ? a[row+1][column] : ACTIVE;

            var c = b
                ? Magic((scanModule1,
scanModule2))
                : Magic([scanModule1,
scanModule2]);

            sb.Append(c);
        }
        sb.AppendLine();
    }
    return sb.ToString();
}

/// <summary>
/// Magic
/// </summary>
private static char Magic(byte[] a)
=> (a[0], a[1]) switch
{
    (0, 0) => ' ',
    (0, 1) => '■',
    (1, 0) => '■',
    (1, 1) => '■',
    _ => throw new
NotImplementedException(),
};

/// <summary>
/// Magic
/// </summary>
private static char Magic((byte a, byte
b)c)
=> (c.a, c.b) switch
{
    (1, 1) => ' ',
    (1, 0) => '■',
    (0, 1) => '■',
    (0, 0) => '■',
    _ => throw new
NotImplementedException(),
};

/// <summary>

```

```

/// Создание болванки матрицы,
заполненной <see cref="ACTIVE"/>
/// </summary>
private static List<byte[]> Magic(int a,
byte b = ACTIVE)
{
    List<byte[]> qrCodeMatrix = [];
    for (int i = 0; i < a; i++)
    {
        qrCodeMatrix.Add(new byte[a]);
    }
    return qrCodeMatrix.Magic(b);
}

/// <summary>
/// Размер матрицы в зависимости от
версии QR-кода
/// </summary>
private static int Magic(byte a, int b)
{
    return 17 + 4 * a + b * 2;
}

#endregion

#region Magic

private static List<byte[]> Magic(this
List<byte[]> a, int b, int c, int d, byte e,
byte f)
{
    Magic(a, b - d + 0, c - d + 0, 9, f);
    Magic(a, b - d + 1, c - d + 1, 7, e);
    Magic(a, b - d + 2, c - d + 2, 5, f);
    Magic(a, b - d + 3, c - d + 3, 3, e);
    return a;
}

private static List<byte[]> Magic(this
List<byte[]> a, byte b)
{
    for (int i = 0; i < a.Count; i++)
    {
        for (int j = 0; j < a.Count; j++)
        {
            a[i][j] = b;
        }
    }
    return a;
}

private static List<byte[]> Magic(this
List<byte[]> a, int b, int y)
{
    Magic(a, b - 2, y - 2, 5, 0);
    Magic(a, b - 1, y - 1, 3, 1);
    a[b][y] = 0;
    return a;
}

private static List<byte[]> Magic(this
List<byte[]> a, bool b, byte c)
{
    var offset = BORDER + 6;
    for (int i = offset; i < a.Count -
offset; i++)
    {
        a[i][offset] = a[offset][i] = !b
? (byte)((i - offset) % 2) : c;
    }
    return a;
}

private static List<byte[]> Magic(this
List<byte[]> a, int b, int c, int d, byte e)
{
    for(int i = 0; i < d; i++)
    {
        for(int j = 0; j < d; j++)

```


C#-программирование

```

        {
            a[i + b][j + c] = e;
        }
    }
    return a;
}

private static List<byte[]> Magic(this
List<byte[]> a, int b, int c, byte d)
{
    a[b][c] = d;
    return a;
}

private static readonly Dictionary<QR,
int[]> _alignmentsPosition = new()
{
    {QR.V1, []},
    {QR.V2, [18]},
    {QR.V3, [22]},
    {QR.V4, [26]},
    {QR.V5, [30]},
    {QR.V6, [34]},
    {QR.V7, [6, 22, 38]},
    {QR.V8, [6, 24, 42]},
    {QR.V9, [6, 26, 46]},
    {QR.V10, [6, 28, 50]},
    {QR.V11, [6, 30, 54]},
    {QR.V12, [6, 32, 58]},
    {QR.V13, [6, 34, 62]},
    {QR.V14, [6, 26, 46, 66]},
    {QR.V15, [6, 26, 48, 70]},
    {QR.V16, [6, 26, 50, 74]},
    {QR.V17, [6, 30, 54, 78]},
    {QR.V18, [6, 30, 56, 82]},
    {QR.V19, [6, 30, 58, 86]},
    {QR.V20, [6, 34, 62, 90]},
};

private static readonly Dictionary<QR,
string> _versionCodes = new()
{
    // 000010
    // 111101
    {QR.V7, "000010011110100110"},
    // 010001
    // 011100
    {QR.V8, "010001011100111000"},
    // 110111
    // 011000
    {QR.V9, "110111011000000100"},
    // 101001
    // 111110
    {QR.V10, "101001111110000000"},
    // 001111
    // 111010
    {QR.V11, "001111111010111100"},
    // 001101
    // 100100
    {QR.V12, "001101100100011010"},
    // 101011
    // 100000
    {QR.V13, "101011100000100110"},
    // 110101
    // 000110
    {QR.V14, "110101000110100010"},
    // 010011
    // 000010
    {QR.V15, "010011000010011110"},
    // 011100
    // 010001
    {QR.V16, "011100010001011100"},
    // 111010
    // 010101
    {QR.V17, "111010010101100000"},
    // 100100
    // 110011
    {QR.V18, "100100110011100100"},
    // 000010
    // 110111
    {QR.V19, "000010101001111110"},
    // 000000
    // 101001
    {QR.V20, "000000101001111110"},
};

private static List<byte[]> Magic(this
List<byte[]> a, QR b, bool c = false)
{
    if ((byte)b < 7) return a;

    int pos = 0;
    var version = _versionCodes[b];
    int offsetColumn = BORDER;
    int offsetRow = a.Count - BORDER -
        POSITION_DETECTION - 3;

    for (int row = 0; row < 3; row++)
    {
        for (int column = 0; column < 6;
            column++)
        {
            byte value = c ?
            version[pos++] == '1' ? ZERO : ACTIVE;
            a[offsetColumn +
            column][offsetRow + row] = value;
            a[offsetRow +
            row][offsetColumn + column] = value;
        }
    }
    return a;
}

private static List<byte[]> Magic(this
QRCodeData a, Mask b)
{
    var size = Magic((byte)a.Version,
        BORDER);
    var tmp = Magic(size)
        .Magic(a.Data, a.Version)
        .Magic(Magic(b))
        .Magic(a.Version, b,
            a.CorrectionLevel);

    int posX1 = BORDER + 3;
    int posX2 = tmp.Count - BORDER - 4;
    int posY = BORDER + 3;

    tmp.Magic(false, ZERO)
        .Magic(posX1, posY, 4, 0, 1)
        .Magic(posX1, tmp[0].Length -
            BORDER - 4, 4, 0, 1)
        .Magic(posX2, posY, 4, 0, 1);

    foreach (var x in
        _alignmentsPosition[a.Version])
        foreach (var y in
            _alignmentsPosition[a.Version].Where(y =>
                CanMagic(x + BORDER, y + BORDER, tmp)))
            tmp.Magic(x + BORDER, y +
                BORDER);

    tmp.Magic(posX2 - 4, posY + 5, 0)
        .Magic(a.Version);

    return tmp;
}

private static List<byte[]> Magic(this
QRCodeData a, Mask b)
{
    var size = Magic((byte)a.Version,
        BORDER);
    var tmp = Magic(size)
        .Magic(a.Data, a.Version)
        .Magic(Magic(b))
        .Magic(a.Version, b,
            a.CorrectionLevel);

    int posX1 = BORDER + 3;
    int posX2 = tmp.Count - BORDER - 4;
    int posY = BORDER + 3;

    tmp.Magic(false, ZERO)
        .Magic(posX1, posY, 4, 0, 1)
        .Magic(posX1, tmp[0].Length -
            BORDER - 4, 4, 0, 1)
        .Magic(posX2, posY, 4, 0, 1);

    foreach (var x in
        _alignmentsPosition[a.Version])
        foreach (var y in
            _alignmentsPosition[a.Version].Where(y =>
                CanMagic(x + BORDER, y + BORDER, tmp)))
            tmp.Magic(x + BORDER, y +
                BORDER);

    tmp.Magic(posX2 - 4, posY + 5, 0)
        .Magic(a.Version);

    return tmp;
}

```

C#-программирование

```
private static List<byte[]> Magic(QR a)
{
    int matrixSize = Magic((byte)a,
BORDER);
    var size = matrixSize - BORDER * 2;

    var tmp = Magic(matrixSize, ZERO)
        .Magic(BORDER, BORDER, size,
ACTIVE);

    int cubeSize = 9;
    int posX1 = BORDER;
    int posY1 = BORDER;
    int posX2 = BORDER + cubeSize +
(int)a * 4;
    int posY2 = BORDER + cubeSize +
(int)a * 4;

    tmp.Magic(posX1, posY1, cubeSize,
ZERO)
        .Magic(posX1, posY2, cubeSize,
ZERO)
        .Magic(posX2, posY1, cubeSize,
ZERO);

    foreach (var x in
_alignmentsPosition[a])
        foreach (var y in
_alignmentsPosition[a].Where(y => CanMagic(x
+ BORDER, y + BORDER, tmp)))
            tmp.Magic(x + BORDER - 2, y +
BORDER - 2, 5, 0);

    tmp.Magic(true, ZERO)
        .Magic(a, true);

    return tmp;
}

#endregion

#region Magic

/// <summary>
/// Magic
/// </summary>
private static bool CanMagic(int x, int
y, List<byte[]> matrix)
=> !(x < POSITION_DETECTION + BORDER
+ 1 && y < POSITION_DETECTION + BORDER + 1 ||
x < POSITION_DETECTION +
BORDER + 1 && y > matrix.Count -
POSITION_DETECTION - BORDER ||
x > matrix.Count -
POSITION_DETECTION - BORDER && y <
POSITION_DETECTION + BORDER + 1);

/// <summary>
/// Magic
/// </summary>
private static List<byte[]> Magic(this
List<byte[]> a, string b, QR c)
{
    var blockedModules = Magic(c);

    var size = a.Count - BORDER * 2;
    var up = true;
    var index = 0;
    var count = b.Length;

    for (var column = size + BORDER - 1;
column >= BORDER; column -= 2)
    {
        if (column == 8) column--;

        for (var i = 0; i < size; i++)
        {
            var row = up ? size + BORDER
- i - 1 : i + BORDER;
```

```
        if (index < count &&
!blockedModules.IsMagic(row, column))
            Magic(a, blockedModules,
row, column, b[index++]);

        if (index < count && column >
0 && !blockedModules.IsMagic(row, column -
1))
            Magic(a, blockedModules,
row, column - 1, b[index++]);
        }
        up = !up;
    }

    return a;
}

/// <summary>
/// Magic
/// </summary>
private static bool IsMagic(this
List<byte[]> a, int b, int c)
{
    return a[b][c] == ZERO;
}

/// <summary>
/// Magic
/// </summary>
private static void Magic(List<byte[]> a,
List<byte[]> b, int c, int d, char e)
{
    b[c][d] = ZERO;
    a[c][d] = e != '1' ? ACTIVE : ZERO;
}

#endregion

#region Magic

private const string END_OF_DATA =
"0000";

private static string Magic(string text,
ref EncodingMode? codeType)
{
    if (string.IsNullOrEmpty(text))
        throw new
InvalidDataException("No data to encode!");

    if (!codeType.HasValue)
    {
        if (text.All(x =>
char.IsDigit(x)))
        {
            codeType =
EncodingMode.Numeric;
        }
        else if (text.All(x =>
letterNumberArray.Any(y => x == y)))
        {
            codeType =
EncodingMode.AlphaNumeric;
        }
        else
        {
            codeType =
EncodingMode.Binary;
        }
    }
    var sb = codeType switch
    {
        EncodingMode.Numeric =>
Magic(text, (10, 7, 4)),
        EncodingMode.AlphaNumeric =>
Magic(text.ToUpper(), 0, 45, 11),
```

C#-программирование

```

        EncodingMode.Binary =>
        Magic(text, [8]),
        _ => throw new
        NotSupportedException("Current type of
        encoding not supported!"),
        };

        return
        sb.Append(END_OF_DATA).ToString();
    }

    private static readonly char[]
    letterNumberArray = {

        '0','1','2','3','4','5','6','7','8','9',

        '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',' ','$','%','*',

        '+','-','.','/',':','

    };

    private static void Magic(StringBuilder
    sb, int num, byte lengthType)
    {
        var str = Convert.ToString(num, 2);
        sb.Append(str.PadLeft(lengthType,
        '0'));
    }

    private static StringBuilder Magic(string
    a, int b, int c, byte d)
    {
        StringBuilder sb = new();

        while (b <= a.Length-2)
        {
            var number1 =
            Array.IndexOf(letterNumberArray, a[b]);
            var number2 =
            Array.IndexOf(letterNumberArray, a[b + 1]);
            if (number1 == -1) throw new
            InvalidDataException($"Not supported
            character {a[b]}!");
            if (number2 == -1) throw new
            InvalidDataException($"Not supported
            character {a[b + 1]}!");
            var number = number1 * c +
            number2;

            Magic(sb, number, d);
            b += 2;
        }
        if (a.Length % 2 == 1)
        {
            var number =
            Array.IndexOf(letterNumberArray, a[^1]);
            if (number == -1) throw new
            InvalidDataException($"Not supported
            character {a[^1]}!");
            Magic(sb, number, 6);
        }
        return sb;
    }

    private static StringBuilder Magic(string
    text, (byte a, byte b, byte c) b)
    {
        StringBuilder sb = new();
        var pos = 0;
        while (pos <= text.Length-3)
        {
            var number =
            Convert.ToInt32(text.Substring(pos, 3));
            Magic(sb, number, b.a);
            pos += 3;

```

```

        }
        if (text.Length % 3 == 2)
        {
            var number =
            Convert.ToInt32(text.Substring(pos, 2));
            Magic(sb, number, b.b);
        }
        else if (text.Length % 3 == 1)
        {
            var number =
            Convert.ToInt32(text.Substring(pos, 1));
            Magic(sb, number, b.c);
        }
        return sb;
    }

    private static StringBuilder Magic(string
    text, byte[] b)
    {
        StringBuilder sb = new();
        var bytes =
        Encoding.UTF8.GetBytes(text);
        foreach (var bt in bytes)
        {
            Magic(sb, Convert.ToInt32(bt),
            b[0]);
        }
        return sb;
    }

    private static readonly
    Dictionary<EncodingMode, string>
    _codeTypeMode = new()
    {
        {EncodingMode.Numeric, "0001"},
        {EncodingMode.AlphaNumeric, "0010"},
        {EncodingMode.Binary, "0100"}
    };

    private static byte Magic(EncodingMode a,
    QR b)
    {
        return ((int)b, a) switch
        {
            (< 10, EncodingMode.Numeric) =>
            10,
            (< 10, EncodingMode.AlphaNumeric)
            => 9,
            (< 10, EncodingMode.Binary) => 8,
            (< 27, EncodingMode.Numeric) =>
            12,
            (< 27, EncodingMode.AlphaNumeric)
            => 11,
            (< 27, EncodingMode.Binary) =>
            16,
            (< 27, _) => 10,
            (_, EncodingMode.Numeric) => 14,
            (_, EncodingMode.AlphaNumeric) =>
            13,
            (_, EncodingMode.Binary) => 16,
            (_, _) => 12,
        };
    }

    private static string Magic(EncodingMode
    a, QR b, string c)
    {
        var length = a switch
        {
            EncodingMode.Binary =>
            Encoding.UTF8.GetBytes(c).Length,
            _ => c.Length,
        };
        var size = Magic(a, b);
        var str = Convert.ToString(length,
        2).PadLeft(size, '0');
        return str;
    }

```

C#-программирование

```
private static StringBuilder Magic(this
StringBuilder a, EncodingMode b, QR c, string
d)
{
    return a.Append(Magic(b, c, d,
_codeTypeMode));
}

private static string Magic(EncodingMode
a, QR b, string c, Dictionary<EncodingMode,
string> d)
{
    return d[a] + Magic(a, b, c);
}

private static readonly string[]
_magicTextArray = ["11101100", "00010001"];

private static StringBuilder Magic(this
StringBuilder a)
{
    while (a.Length % 8 != 0)
        a.Append('0');
    return a;
}

private static string Magic(string a, int
b)
{
    var sb = new StringBuilder(a);

    var cnt = (b - a.Length) / 8;
    for (int i = 0; i < cnt; i++)
    {
        sb.Append(_magicTextArray[i %
2]);
    }
    return sb.ToString();
}

public static List<byte[]> Magic(string
a, int b, int c)
{
    List<byte> tmp = [];
    var str = Enumerable.Range(0,
a.Length / 8).Select(i => a.Substring(i * 8,
8));

    foreach (var line in str)
    {
        tmp.Add(Convert.ToByte(line,2));
    }

    var size = a.Length / 8 / b;
    var extraSize = a.Length / 8 % b;

    List<byte[]> list = [];
    for (int i = b - 1; i >= 0; i--)
    {
        var currentSize = size +
(extraSize-- > 0 ? 1 : 0);
        list.Insert(0, new
byte[currentSize]);
    }

    var index = c;
    foreach (var block in list)
    {
        for (int i = 0; i < block.Length;
i++)
        {
            block[i] = tmp[index++];
        }
    }

    return list;
}

#endregion
```

```
#region Magic

private static readonly
Dictionary<EccLevel, byte[]>
_countOfErrorCorrectionCodeWords = new()
{
    {EccLevel.L,
[NA,07,10,15,20,26,18,20,24,30,18,20,24,26,30
,22,24,28,30,28,28]},
    {EccLevel.M,
[NA,10,16,26,18,24,16,18,22,22,26,30,22,22,24
,24,28,28,26,26,26]},
    {EccLevel.Q,
[NA,13,22,18,26,18,24,18,22,20,24,28,26,24,20
,30,24,28,28,26,30]},
    {EccLevel.H,
[NA,17,28,22,16,22,28,26,26,24,28,24,28,22,24
,24,30,28,28,26,28]}
};

private static readonly
Dictionary<EccLevel, byte[]>
_correctionLevelBlocksCount = new()
{
    {EccLevel.L,
[NA,01,01,01,01,01,02,02,02,04,04,04,04,04
,06,06,06,06,07,08]},
    {EccLevel.M,
[NA,01,01,01,02,02,04,04,04,05,05,05,08,09,09
,10,10,11,13,14,16]},
    {EccLevel.Q,
[NA,01,01,02,02,04,04,06,06,08,08,08,10,12,16
,12,17,16,18,21,20]},
    {EccLevel.H,
[NA,01,01,02,04,04,04,05,06,08,08,11,11,16,16
,18,16,19,21,25,25]}
};

private static readonly Dictionary<byte,
byte[]> _correctionLevelGeneratingPolynomial
= new()
{
    {
        // x^7 + α^87x^6 + α^229x^5 +
α^146x^4 +
        // α^149x^3 + α^238x^2 + α^102x^ +
α^21
        {7, [87, 229, 146, 149, 238, 102,
21]}},
    {
        // x^10 + α^251x^9 + α^67x^8 +
α^46x^7 + α^61x^6 +
        // α^118x^5 + α^70x^4 + α^64x^3 +
α^94x^2 + α^32x^ + α^45
        {10, [251, 67, 46, 61, 118, 70, 64,
94, 32, 45]}},
    {
        // x^13 + α^74x^12 + α^152x^11 +
α^176x^10 + α^100x^9 + α^86x^8 +
        // α^100x^7 + α^106x^6 + α^104x^5 +
α^130x^4 + α^218x^3 + α^206x^2 + α^140x^ +
α^78
        {13, [74, 152, 176, 100, 86, 100,
106, 104, 130, 218, 206, 140, 78]}},
    {
        // x^16 + α^120x^15 + α^104x^14 +
α^107x^13 + α^109x^12 + α^102x^11 + α^161x^10
+ α^76x^9 +
        // α^3x^8 + α^91x^7 + α^191x^6
+α^147x^5 + α^169x^4 + α^182x^3 + α^194x^2 +
α^225x^ + α^120
        {16, [120, 104, 107, 109, 102, 161,
76, 3, 91, 191, 147, 169, 182, 194, 225,
120]}},
    {
        // x^17 + α^43x^16 + α^139x^15 +
α^206x^14 + α^78x^13 + α^43x^12 + α^239x^11
+ α^123x^10 + α^206x^9 + α^214x^8 + α^147x^7
+ α^24x^6 +
        // α^99x^5 + α^150x^4 + α^39x^3 +
α^243x^2 + α^163x^ + α^136
    }
};
```

C#-программирование

```
{17, [43, 139, 206, 78, 43, 239, 123,
206, 214, 147, 24, 99, 150, 39, 243, 163,
136]}},
// x^18 + α^215x^17 + α^234x^16 +
α^158x^15 + α^94x^14 + α^184x^13 + α^97x^12 +
α^118x^11 + α^170x^10 + α^79x^9 + α^187x^8 +
α^152x^7 +
// α^148x^6 + α^252x^5 + α^179x^4 +
α^5x^3 + α^98x^2 + α^96x^1 + α^153
{18, [215, 234, 158, 94, 184, 97,
118, 170, 79, 187, 152, 148, 252, 179, 5, 98,
96, 153]}},
{20, [17, 60, 79, 50, 61, 163, 26,
187, 202, 180, 221, 225, 83, 239, 156, 164,
212, 212, 188, 190]}},
// x^22 + α^210x^21 + α^171x^20 +
α^247x^19 + α^242x^18 + α^93x^17 + α^230x^16
+ α^14x^15 + α^109x^14 + α^221x^13 + α^53x^12
+
// α^200x^11 + α^74x^10 + α^8x^9 +
α^172x^8 + α^98x^7 + α^80x^6 + α^219x^5 +
α^134x^4 + α^160x^3 + α^105x^2 + α^165x^1
+ α^231
{22, [210, 171, 247, 242, 93, 230,
14, 109, 221, 53, 200, 74, 8, 172, 98, 80,
219, 134, 160, 105, 165, 231]}},
{24, [229, 121, 135, 48, 211, 117,
251, 126, 159, 180, 169, 152, 192, 226, 228,
218, 111, 0, 117, 232, 87, 96, 227, 21]}},
{26, [173, 125, 158, 2, 103, 182,
118, 17, 145, 201, 111, 28, 165, 53, 161, 21,
245, 142, 13, 102, 48, 227, 153, 145, 218,
70]}},
// x^28 + α^168x^27 + α^223x^26 +
α^200x^25 + α^104x^24 + α^224x^23 + α^234x^22
+ α^108x^21 + α^180x^20 + α^110x^19 +
α^190x^18 + α^195x^17 +
// α^147x^16 + α^205x^15 + α^27x^14 +
α^232x^13 + α^201x^12 + α^21x^11 + α^43x^10 +
α^245x^9 + α^87x^8 + α^42x^7 + α^195x^6 +
α^212x^5 + α^119x^4 +
// α^242x^3 + α^37x^2 + α^9x^1 + α^123
{28, [168, 223, 200, 104, 224, 234,
108, 180, 110, 190, 195, 147, 205, 27, 232,
201, 21, 43, 245, 87, 42, 195, 212, 119, 242,
37, 9, 123]}},
{30, [41, 173, 145, 152, 216, 31,
179, 182, 50, 48, 110, 86, 239, 96, 222, 125,
42, 173, 226, 193, 224, 130, 156, 37, 251,
216, 238, 40, 192, 180]}},
};

private static readonly byte[]
_galoisField = [1, 2, 4, 8, 16, 32, 64, 128,
29, 58, 116, 232, 205, 135, 19, 38,

76, 152, 45, 90, 180, 117, 234, 201, 143, 3,
6, 12, 24, 48, 96, 192,

157, 39, 78, 156, 37, 74, 148, 53, 106, 212,
181, 119, 238, 193, 159, 35,

70, 140, 5, 10, 20, 40, 80, 160, 93, 186,
105, 210, 185, 111, 222, 161,

95, 190, 97, 194, 153, 47, 94, 188, 101, 202,
137, 15, 30, 60, 120, 240,

253, 231, 211, 187, 107, 214, 177, 127, 254,
225, 223, 163, 91, 182, 113, 226,

217, 175, 67, 134, 17, 34, 68, 136, 13, 26,
52, 104, 208, 189, 103, 206,

129, 31, 62, 124, 248, 237, 199, 147, 59,
118, 236, 197, 151, 51, 102, 204,

133, 23, 46, 92, 184, 109, 218, 169, 79, 158,
33, 66, 132, 21, 42, 84,

168, 77, 154, 41, 82, 164, 85, 170, 73, 146,
57, 114, 228, 213, 183, 115,

230, 209, 191, 99, 198, 145, 63, 126, 252,
229, 215, 179, 123, 246, 241, 255,

227, 219, 171, 75, 150, 49, 98, 196, 149, 55,
110, 220, 165, 87, 174, 65,

130, 25, 50, 100, 200, 141, 7, 14, 28, 56,
112, 224, 221, 167, 83, 166,

81, 162, 89, 178, 121, 242, 249, 239, 195,
155, 43, 86, 172, 69, 138, 9,

18, 36, 72, 144, 61, 122, 244, 245, 247, 243,
251, 235, 203, 139, 11, 22,

44, 88, 176, 125, 250, 233, 207, 131, 27, 54,
108, 216, 173, 71, 142, 1

];

private static readonly byte[]
_backGaloisField = [NA, 0, 1, 25, 2, 50, 26,
198, 3, 223, 51, 238, 27, 104, 199, 75,

4, 100, 224, 14, 52, 141, 239, 129, 28, 193,
105, 248, 200, 8, 76, 113,

5, 138, 101, 47, 225, 36, 15, 33, 53, 147,
142, 218, 240, 18, 130, 69,

29, 181, 194, 125, 106, 39, 249, 185, 201,
154, 9, 120, 77, 228, 114, 166,

6, 191, 139, 98, 102, 221, 48, 253, 226, 152,
37, 179, 16, 145, 34, 136,

54, 208, 148, 206, 143, 150, 219, 189, 241,
210, 19, 92, 131, 56, 70, 64,

30, 66, 182, 163, 195, 72, 126, 110, 107, 58,
40, 84, 250, 133, 186, 61,

202, 94, 155, 159, 10, 21, 121, 43, 78, 212,
229, 172, 115, 243, 167, 87,

7, 112, 192, 247, 140, 128, 99, 13, 103, 74,
222, 237, 49, 197, 254, 24,

227, 165, 153, 119, 38, 184, 180, 124, 17,
68, 146, 217, 35, 32, 137, 46,

55, 63, 209, 91, 149, 188, 207, 205, 144,
135, 151, 178, 220, 252, 190, 97,

242, 86, 211, 171, 20, 42, 93, 158, 132, 60,
57, 83, 71, 109, 65, 162,

31, 45, 67, 216, 183, 123, 164, 118, 196, 23,
73, 236, 127, 12, 111, 246,

108, 161, 59, 82, 41, 157, 85, 170, 251, 96,
134, 177, 187, 204, 62, 90,

203, 89, 95, 176, 156, 169, 160, 81, 11, 245,
22, 235, 122, 117, 44, 215,

79, 174, 213, 233, 230, 231, 173, 232, 116,
214, 244, 234, 168, 80, 88, 175

];

/// <summary>
/// Magic
/// </summary>
```

C#-программирование

```

private static byte[] Magic(byte[] a,
byte b)
{
    var size = Math.Max(a.Length, b);
    var m = new List<byte>(a);
    var g =
_correctionLevelGeneratingPolynomial[b];
    var n = g.Length;
    while (m.Count != size)
        m.Add(0);

    for (int i = 0; i < a.Length; i++)
    {
        byte e = m[0];

        m.RemoveAt(0);
        m.Add(0);

        if (e == 0) continue;

        byte bb = _backGaloisField[e];
        for (int x = 0; x < g.Length;
x++)
        {
            var c = (g[x] + bb) % 255;
            var d = _galoisField[c];
            m[x] = (byte) (m[x] ^ d);
        }

        return m.Take(n).ToArray();
    }

    private static void Magic(List<byte[]> a,
StringBuilder b)
    {
        if (a.Count == 1)
        {
            foreach (var c in a[0])
            {
                b.Append(Convert.ToString(c,
2).PadLeft(8, '0'));
            }
            return;
        }

        var size = a.Max(x => x.Length);
        for (int i = 0; i < size; i++)
        {
            foreach (var bytes in a)
            {
                if (i < bytes.Length)
b.Append(Convert.ToString(bytes[i],
2).PadLeft(8, '0'));
            }
        }

        private static string Magic(List<byte[]>
a, List<byte[]> b)
        {
            var sb = new StringBuilder();
            Magic(a, sb);
            Magic(b, sb);
            return sb.ToString();
        }

        private static readonly
Dictionary<(EccLevel correctionLevel, QR
version), int> _maxData = new()
        {
            {(EccLevel.H, QR.V1), 72},
            {(EccLevel.Q, QR.V1), 104}, {(EccLevel.M,
QR.V1), 128}, {(EccLevel.L, QR.V1), 152},
            {(EccLevel.H, QR.V2), 128},
            {(EccLevel.Q, QR.V2), 176}, {(EccLevel.M,
QR.V2), 224}, {(EccLevel.L, QR.V2), 272},

```

```

            {(EccLevel.H, QR.V3), 208},
            {(EccLevel.Q, QR.V3), 272}, {(EccLevel.M,
QR.V3), 352}, {(EccLevel.L, QR.V3), 440},
            {(EccLevel.H, QR.V4), 288},
            {(EccLevel.Q, QR.V4), 384}, {(EccLevel.M,
QR.V4), 512}, {(EccLevel.L, QR.V4), 640},
            {(EccLevel.H, QR.V5), 368},
            {(EccLevel.Q, QR.V5), 496}, {(EccLevel.M,
QR.V5), 688}, {(EccLevel.L, QR.V5), 864},
            {(EccLevel.H, QR.V6), 480},
            {(EccLevel.Q, QR.V6), 608}, {(EccLevel.M,
QR.V6), 864}, {(EccLevel.L, QR.V6), 1088},
            {(EccLevel.H, QR.V7), 528},
            {(EccLevel.Q, QR.V7), 704}, {(EccLevel.M,
QR.V7), 992}, {(EccLevel.L, QR.V7), 1248},
            {(EccLevel.H, QR.V8), 688},
            {(EccLevel.Q, QR.V8), 880}, {(EccLevel.M,
QR.V8), 1232}, {(EccLevel.L, QR.V8), 1552},
            {(EccLevel.H, QR.V9), 800},
            {(EccLevel.Q, QR.V9), 1056}, {(EccLevel.M,
QR.V9), 1456}, {(EccLevel.L, QR.V9), 1856},
            {(EccLevel.H, QR.V10), 976},
            {(EccLevel.Q, QR.V10), 1232}, {(EccLevel.M,
QR.V10), 1728}, {(EccLevel.L, QR.V10), 2192},
            {(EccLevel.H, QR.V11), 1120},
            {(EccLevel.Q, QR.V11), 1440}, {(EccLevel.M,
QR.V11), 2032}, {(EccLevel.L, QR.V11), 2592},
            {(EccLevel.H, QR.V12), 1264},
            {(EccLevel.Q, QR.V12), 1648}, {(EccLevel.M,
QR.V12), 2320}, {(EccLevel.L, QR.V12), 2960},
            {(EccLevel.H, QR.V13), 1440},
            {(EccLevel.Q, QR.V13), 1952}, {(EccLevel.M,
QR.V13), 2672}, {(EccLevel.L, QR.V13), 3424},
            {(EccLevel.H, QR.V14), 1576},
            {(EccLevel.Q, QR.V14), 2088}, {(EccLevel.M,
QR.V14), 2920}, {(EccLevel.L, QR.V14), 3688},
            {(EccLevel.H, QR.V15), 1784},
            {(EccLevel.Q, QR.V15), 2360}, {(EccLevel.M,
QR.V15), 3320}, {(EccLevel.L, QR.V15), 4184},
            {(EccLevel.H, QR.V16), 2024},
            {(EccLevel.Q, QR.V16), 2600}, {(EccLevel.M,
QR.V16), 3624}, {(EccLevel.L, QR.V16), 4712},
            {(EccLevel.H, QR.V17), 2264},
            {(EccLevel.Q, QR.V17), 2936}, {(EccLevel.M,
QR.V17), 4056}, {(EccLevel.L, QR.V17), 5176},
            {(EccLevel.H, QR.V18), 2504},
            {(EccLevel.Q, QR.V18), 3176}, {(EccLevel.M,
QR.V18), 4504}, {(EccLevel.L, QR.V18), 5768},
            {(EccLevel.H, QR.V19), 2728},
            {(EccLevel.Q, QR.V19), 3560}, {(EccLevel.M,
QR.V19), 5016}, {(EccLevel.L, QR.V19), 6360},
            {(EccLevel.H, QR.V20), 3080},
            {(EccLevel.Q, QR.V20), 3880}, {(EccLevel.M,
QR.V20), 5352}, {(EccLevel.L, QR.V20), 6888},
        };

        /// <summary>
        /// Magic
        /// </summary>
        private static (string a, EccLevel b, QR
c) Magic(string a, string b, EncodingMode c,
QR d, EccLevel? e = null)
        {
            if (d == NA)
                throw new
NotSupportedException("QR-code version start
with 1!");

            var sb = new StringBuilder();
            sb.Magic(c, d, a)
                .Append(b)
                .Magic();

            var length = sb.Length;

            if ((int)d > 20)
                throw new
NotSupportedException($"Current QR-code does
not support version {d} yet!");

```

C#-программирование

```

        if (e.HasValue)
        {
            foreach (var found in _maxData
                .Where(v => v.Key.version >=
d && v.Key.correctionLevel >= e.Value)
                .Where(l => length <
1.Value))
            {
                return (sb.ToString(),
found.Key.correctionLevel,
found.Key.version);
            }

            foreach (var found in _maxData
                .Where(v => v.Key.version == d)

.OrderByDescending(x=>x.Key.correctionLevel)
                .Where(x => length < x.Value))
            {
                return (sb.ToString(),
found.Key.correctionLevel,
found.Key.version);
            }

            if ((int)d > 20)
                throw new
NotSupportedException($"Current QR-code does
not support data length {length} yet!");

            return Magic(a, b, c, d + 1, e);
        }
    }

    #endregion

    #region Magic

    /// <summary>
    /// Format information
    /// </summary>
    private static readonly
Dictionary<(EccLevel correctionLevel, Mask
maskNum), string> _masksAndCorrectionLevel =
new()
    {
        { (EccLevel.L, Mask.M101),
"111011111000100"},
        { (EccLevel.L, Mask.M100),
"111001011110011"},
        { (EccLevel.L, Mask.M111),
"111110110101010"},
        { (EccLevel.L, Mask.M110),
"111100010011101"},
        { (EccLevel.L, Mask.M001),
"110011000101111"},
        { (EccLevel.L, Mask.M000),
"110001100011000"},
        { (EccLevel.L, Mask.M011),
"110110001000001"},
        { (EccLevel.L, Mask.M010),
"110100101110110"},
        { (EccLevel.M, Mask.M101),
"101010000010010"},
        { (EccLevel.M, Mask.M100),
"101000100100101"},
        { (EccLevel.M, Mask.M111),
"101111001111100"},
        { (EccLevel.M, Mask.M110),
"101101101001011"},
        { (EccLevel.M, Mask.M001),
"100010111111001"},
        { (EccLevel.M, Mask.M000),
"100000011001110"},
        { (EccLevel.M, Mask.M011),
"100111110010111"},
        { (EccLevel.M, Mask.M010),
"100101010100000"},
        { (EccLevel.Q, Mask.M101),
"011010101011111"},
        { (EccLevel.Q, Mask.M100),
"011000001101000"},
        { (EccLevel.Q, Mask.M111),
"011111100110001"},
        { (EccLevel.Q, Mask.M110),
"011101000000110"},
        { (EccLevel.Q, Mask.M001),
"010010010101010"},
        { (EccLevel.Q, Mask.M000),
"010000110000011"},
        { (EccLevel.Q, Mask.M011),
"010111011011010"},
        { (EccLevel.Q, Mask.M010),
"010101111101101"},
        { (EccLevel.H, Mask.M101),
"001011010001001"},
        { (EccLevel.H, Mask.M100),
"001001110111110"},
        { (EccLevel.H, Mask.M111),
"001110011100111"},
        { (EccLevel.H, Mask.M110),
"001100111010000"},
        { (EccLevel.H, Mask.M001),
"000011101100010"},
        { (EccLevel.H, Mask.M000),
"000001001010101"},
        { (EccLevel.H, Mask.M011),
"000110100001100"},
        { (EccLevel.H, Mask.M010),
"000100000111011"},
    };

    /// <summary>
    /// Информация о маске и уровне коррекции
    /// </summary>
    private static List<byte[]> Magic(this
List<byte[]> a, QR b, Mask c, EccLevel d)
    {
        var maskNumAndCorrectionLevel =
_masksAndCorrectionLevel[(d, c)];
        Magic(a, b,
maskNumAndCorrectionLevel);
        return a;
    }

    /// <summary>
    /// Format information - порядок
    размещения возле верхнего левого паттерна
    позиционирования
    /// </summary>
    private static readonly Pair[]
_masksAndCorrectionLevelTopLeftTemplate = [
    (02, 10), (03, 10), (04, 10),
    (05, 10), (06, 10), (07, 10),
    (09, 10), (10, 10), (10, 09),
    (10, 07), (10, 06), (10, 05),
    (10, 04), (10, 03), (10, 02),
];

    /// <summary>
    /// Magic
    /// </summary>
    private static Pair[] Magic(this QR a,
int b)
    {
        var offset = 11 + (int)a * 4;
        return [(b, offset + 7), (b, offset
+6 ), (b, offset + 5),
                (b, offset + 4), (b, offset +
3), (b, offset + 2),
                (b, offset + 1), (b, offset),
                (offset + 1, b),
                (offset + 2, b), (offset + 3,
b), (offset + 4, b),
    ];
    }

```

C#-программирование

```

        (offset + 5, b), (offset + 6,
b), (offset + 7, b)];
    }

    /// <summary>
    /// Magic
    /// </summary>
    private static void Magic(Pair[] a,
List<byte[]> b, int c, char d)
    {
        (var x, var y) = (a[c].X, a[c].Y);
        b[y][x] = d != '1' ? ACTIVE : ZERO;
    }

    /// <summary>
    /// Magic
    /// </summary>
    private static List<byte[]> Magic(this
List<byte[]> a, QR b, string c)
    {
        for (int i = 0; i < c.Length; i++)
        {
            char letter = c[i];

Magic(_masksAndCorrectionLevelTopLeftTemplate
, a, i, letter);
            Magic(b.Magic(10), a, i, letter);
        }
        return a;
    }

    #endregion

    #region Magic

    /// <summary>
    /// Magic
    /// </summary>
    private static Predicate<(int,int)>
Magic(Mask a)
    => a switch
    {
        Mask.M000 => ((int x, int y) m)
=> (m.x * m.y) % 2 + (m.x * m.y) % 3 == 0,
        Mask.M001 => ((int x, int y) m)
=> (m.x/3 + m.y/2) % 2 == 0,
        Mask.M010 => ((int x, int y) m)
=> ((m.x * m.y) % 3 + (m.x + m.y) % 2) % 2 ==
0,
        Mask.M011 => ((int x, int y) m)
=> ((m.x * m.y) % 2 + (m.x * m.y) % 3) % 2 ==
0,
        Mask.M100 => ((int x, int y) m)
=> m.y % 2 == 0,
        Mask.M101 => ((int x, int y) m)
=> (m.x + m.y) % 2 == 0,
        Mask.M110 => ((int x, int y) m)
=> (m.x + m.y) % 3 == 0,
        Mask.M111 => ((int x, int y) m)
=> m.x % 3 == 0,
        _ => throw new
ArgumentOutOfRangeException("Incorrect Mask
Number")
    };

    /// <summary>
    /// Magic
    /// </summary>
    private static List<byte[]> Magic(this
List<byte[]> a, Predicate<(int,int)> b)
    {
        for (int x = BORDER; x < a.Count -
BORDER; x++)
        {
            for (int y = BORDER; y < a.Count
- BORDER; y++)
            {
                if (b((y - BORDER, x -
BORDER)))

```

```

                a[x][y] = (byte) (ACTIVE -
a[x][y]);
            }
        }
        return a;
    }

    /// <summary>
    /// Magic
    /// </summary>
    private static (int a, List<byte[]>b)
Magic(this (int a, List<byte[]>b) a)
    {
        var length = 5;
        var s = 0;

        int cnt;
        int current;
        for (int x = BORDER; x < a.b.Count -
BORDER; x++)
        {
            cnt = 0;
            current = 3;
            for (int y = BORDER; y <
a.b.Count - BORDER; y++)
            {
                if (a.b[x][y] == current)
                    cnt++;
                else
                {
                    if (cnt >= length)
                        s += cnt - 2;
                    current = a.b[x][y];
                    cnt = 0;
                }
            }
        }

        for (int y = BORDER; y<a.b.Count -
BORDER; y++)
        {
            cnt = 0;
            current = 3;
            for (int x = BORDER; x<a.b.Count
- BORDER; x++)
            {
                if (a.b[x][y] == current)
                    cnt++;
                else
                {
                    if (cnt>=length)
                        s += cnt - 2;
                    current = a.b[x][y];
                    cnt = 0;
                }
            }
        }

        return (s + a.a, a.b);
    }

    /// <summary>
    /// Magic
    /// </summary>
    private static (int a, List<byte[]>b)
Magic(this (int a, List<byte[]>b) a, double
b)
    {
        var cntTotal = 0.0;
        for (int x = BORDER; x < a.b.Count -
BORDER; x++)
        {
            for (int y = BORDER; y <
a.b.Count - BORDER; y++)
            {
                if (a.b[x][y] == ACTIVE) b++;
                cntTotal++;
            }
        }
    }

```


С#-программирование

```
        double s = b / cntTotal;
        s = s * 100 - 50;
        return (Math.Abs((int)s) * BORDER +
a.a, a.b);
    }

    /// <summary>
    /// Magic
    /// </summary>
    private static List<byte[]> Magic(this
QrCodeData a, ref Mask? b)
    {
        var res = Enumerable
            .Range(0, 8)
            .Select(maskNumber =>
(maskNumber, (0, a.Magic((Mask)maskNumber))
                .Magic()
                .Magic(0.0)))
            .MinBy(x=>x.Item2.a);

        b = (Mask)res.maskNumber;

        return res.Item2.b;
    }

    #endregion
}
```