プロジェクト実習1
～第4回（大東担当分、2024/5/10）～

東海大学　情報通信学部 情報通信学科
大東　俊博
ohigashi@tokai.ac.jp

QRコード

- 1994年に株式会社デンソーによって開発された2次元バーコード

例：www.u-tokai.ac.jp のQRコード

- 1次元のバーコードと比べて扱える情報量が多い
- 汚れや撮影画像の歪みへの耐性（誤り訂正機能）
- 広く使われている
  - 情報の提示：URLを掲載してWebサイトへ誘導
  - 航空機の搭乗手続きの簡略化
  - 電子決済：LINE Pay, PayPay, etc.

QRコードの潜在的な脆弱性

- 埋め込まれている情報が可視化されていないため、画像を見ただけでは正しい情報が埋め込まれているかを確認できない
  ⇒　悪性サイト（ウイルスに感染してしまうようなサイト）に誘導されるURLが含まれていても気づけない

- ただし、通常の偽造ではいつも悪性サイトに誘導されるため、掲示されたQRコードを剥がすなどで回避できる

実際にあった事件

- QR決済でカネをだまし取る「ステッカー型」詐欺
  - 毎日新聞Webサイト
    https://mainichi.jp/premier/business/articles/20190924/biz/00m/020/018000c

もっと危ないQRコード（偽装QRコード）

- 2018年6月ICSS研究会にて大熊らにより提案[1]
  - QRコードの誤り訂正符号の性質に注目した方法
  - 悪性サイトへ確率的に誘導可能
    - 0.6%の確率で誘導した例もあり

偽装QRコードの例

正規サイト:
www.u-tokai.ac.jp

高確率

低確率

悪性サイト:
www.u-yokai.ac.jp

[1] 大熊浩也，瀧田　愼，森井昌克：悪性サイトに誘導するQR コードの存在とそれを利用した偽造攻撃，電子情報通信学会技術研究報告，ICSS，Vol. 118, No. 109, pp. 33–38 (2018).

もっと危ないQRコード（偽装QRコード）

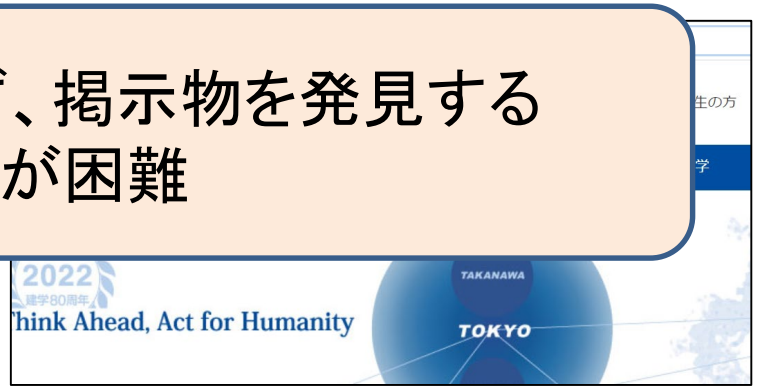
- 2018年6月ICSS研究会にて大熊らにより提案[1]
  - QRコードの誤り訂正符号の性質に注目した方法
  - 悪性サイトへ確率的に誘導可能
    - ◆0.6%の確率で誘導した例もあり

⇒ たまにしか被害が発生せず、掲示物を発見するのが遅れてしまうため対応が困難

偽装QRコードの例

低確率

悪性サイト:
www.u-yokai.ac.jp



[1] 大熊浩也, 瀧田　愼, 森井昌克：悪性サイトに誘導するQR コードの存在とそれを利用した偽造攻撃, 電子情報通信学会技術研究報告, ICSS, Vol. 118, No. 109, pp. 33–38 (2018).

まずは偽装QRコードを撮影してみよう！

Webページ1: http://www.u-tokai.ac.jp/
Webページ2: http://www.u-yokai.ac.jp/

大東担当分（第4回、第5回）でやること

- 第4回　QRコードを作ってみよう
  - QRコードがどういうフォーマットで作られているかの理解
  - 自分で決めたURLにアクセスさせるデータを作成
  - リードソロモン符号による冗長化
    → 山本先生分のときに作ったプログラムを利用
  - 画像として印字
    → Webシステムとして用意したのでそれを使う

- 第5回　偽装QRコードを理解しよう
  - 撮影ごとに振る舞いが変化する（ことがある）
    悪性QRコードの原理を理解する
  - 実際に自分で作ってみる
    → 第4回のプログラムをベースに改造

QRコードの説明

- モジュール（正方形の□や■）
  - 白は 0、黒は1の数字を表現
  - 0と1を8個分で1文字を表現している
    例）「a」の文字 ← 01100001 (10進数で書くと97)

文字と数値の関係（ASCII文字コード）

文字 a ＝97（10進数）
＝01100001（2進数）

ASCII文字コード

文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進
NUL	0	00	DLE	16	10	SP	32	20	0	48	30	@	64	40	P	80	50	`	96	60	p	112	70
SOH	1	01	DC1	17	11	!	33	21	1	49	31	A	65	41	Q	81	51	a	97	61	q	113	71
STX	2	02	DC2	18	12	"	34	22	2	50	32	B	66	42	R	82	52	b	98	62	r	114	72
ETX	3	03	DC3	19	13	#	35	23	3	51	33	C	67	43	S	83	53	c	99	63	s	115	73
EOT	4	04	DC4	20	14	$	36	24	4	52	34	D	68	44	T	84	54	d	100	64	t	116	74
ENQ	5	05	NAK	21	15	%	37	25	5	53	35	E	69	45	U	85	55	e	101	65	u	117	75
ACK	6	06	SYN	22	16	&	38	26	6	54	36	F	70	46	V	86	56	f	102	66	v	118	76
BEL	7	07	ETB	23	17	'	39	27	7	55	37	G	71	47	W	87	57	g	103	67	w	119	77
BS	8	08	CAN	24	18	(	40	28	8	56	38	H	72	48	X	88	58	h	104	68	x	120	78
HT	9	09	EM	25	19	)	41	29	9	57	39	I	73	49	Y	89	59	i	105	69	y	121	79
LF*	10	0a	SUB	26	1a	*	42	2a	:	58	3a	J	74	4a	Z	90	5a	j	106	6a	z	122	7a
VT	11	0b	ESC	27	1b	+	43	2b	;	59	3b	K	75	4b	[	91	5b	k	107	6b	{	123	7b
FF*	12	0c	FS	28	1c	,	44	2c	<	60	3c	L	76	4c	\¥	92	5c	l	108	6c	|	124	7c
CR	13	0d	GS	29	1d	-	45	2d	=	61	3d	M	77	4d	]	93	5d	m	109	6d	}	125	7d
SO	14	0e	RS	30	1e	.	46	2e	>	62	3e	N	78	4e	^	94	5e	n	110	6e	~	126	7e
SI	15	0f	US	31	1f	/	47	2f	?	63	3f	O	79	4f	_	95	5f	o	111	6f	DEL	127	7f

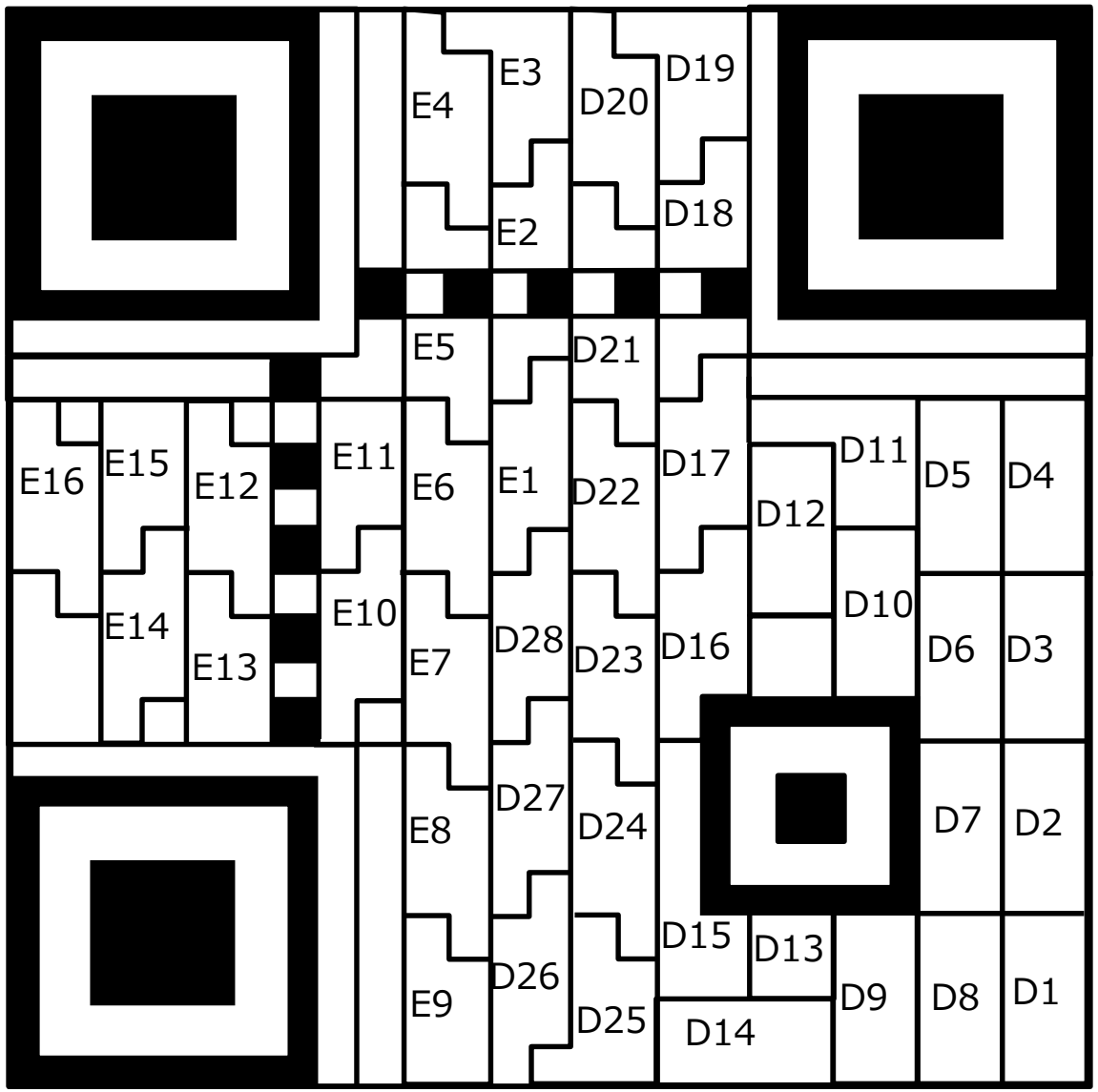
* LFはNL、FFはNPと呼ばれることもある。
* 赤字は制御文字、SPは空白文字(スペース)、黒字と緑字は図形文字。
* 緑字はISO 646で割り当ての変更が認められており、例えば日本ではバックスラッシュが円記号になってい

http://e-words.jp/p/r-ascii.html

QRコードの説明

- モジュール（正方形の□や■）
  - 白は 0、黒は1の数字を表現
  - 0と1を8個分で1文字を表現している
    例）「a」の文字 ← 01100001 (10進数で書くと97)
- モジュールのバージョン
  - 配置できるモジュール数を指定するもの
    - どれだけデータを埋め込められるかが決まる
    - version 2では 25x25のモジュールを配置

QRコードのモジュール配置（2-M型QRコード）



1モジュール
（白黒の四角の最小単位）
で1ビットを表現

QRコードの説明

- モジュール（正方形の□や■）
  - 白は 0、黒は1の数字を表現
  - 0と1を8個分で1文字を表現している
    例）「a」の文字 ← 01100001 (10進数で書くと97)
- モジュールのバージョン
  - 配置できるモジュール数を指定するもの
    - どれだけデータを埋め込められるかが決まる
    - version 2では 25x25のモジュールを配置
- 誤り訂正機能
  - 汚れや影などで少しぐらい白黒判定を誤っても
    正しく読み取れるようにする機能
  - どれくらい判定を間違っても読めるかも設定できる

誤り訂正符号

- 送受信されるデータにノイズが入ってエラーが起こったとしても検出・訂正を可能にする技術

何もしない場合（Aという文字を送りたい場合）

（ノイズ無しの通信） A → A

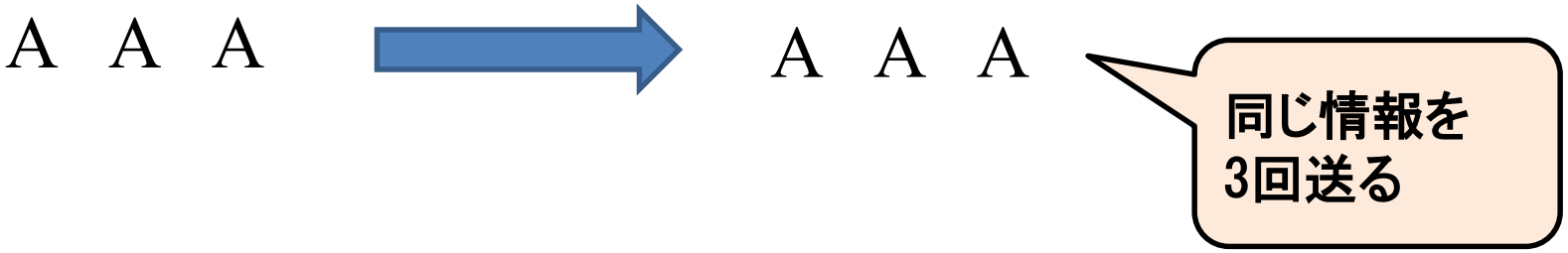
（通信路上でノイズが発生） A → F

誤りが生じる（気づけない）

誤り訂正符号

- 送受信されるデータにノイズが入ってエラーが起こったとしても検出・訂正を可能にする技術
- 例）多数決法

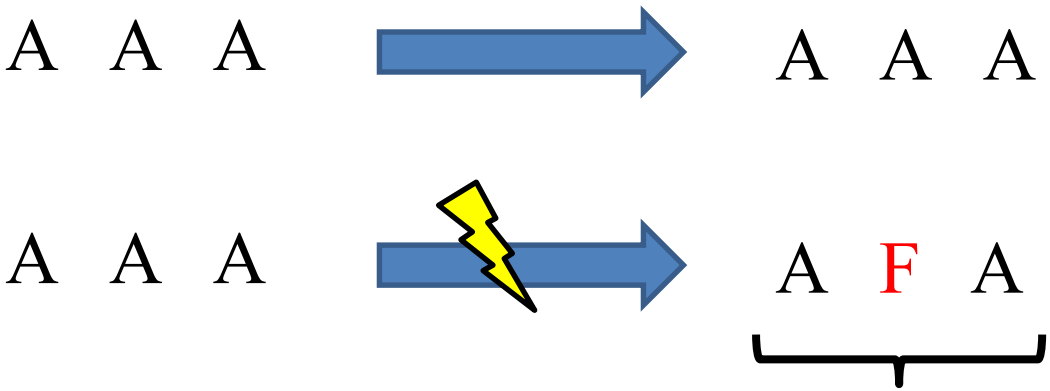
誤り訂正符号（多数決法）（Aという文字を送りたい場合）



誤り訂正符号

- 送受信されるデータにノイズが入ってエラーが起こったとしても検出・訂正を可能にする技術
- 例）多数決法

誤り訂正符号（多数決法）（Aという文字を送りたい場合）

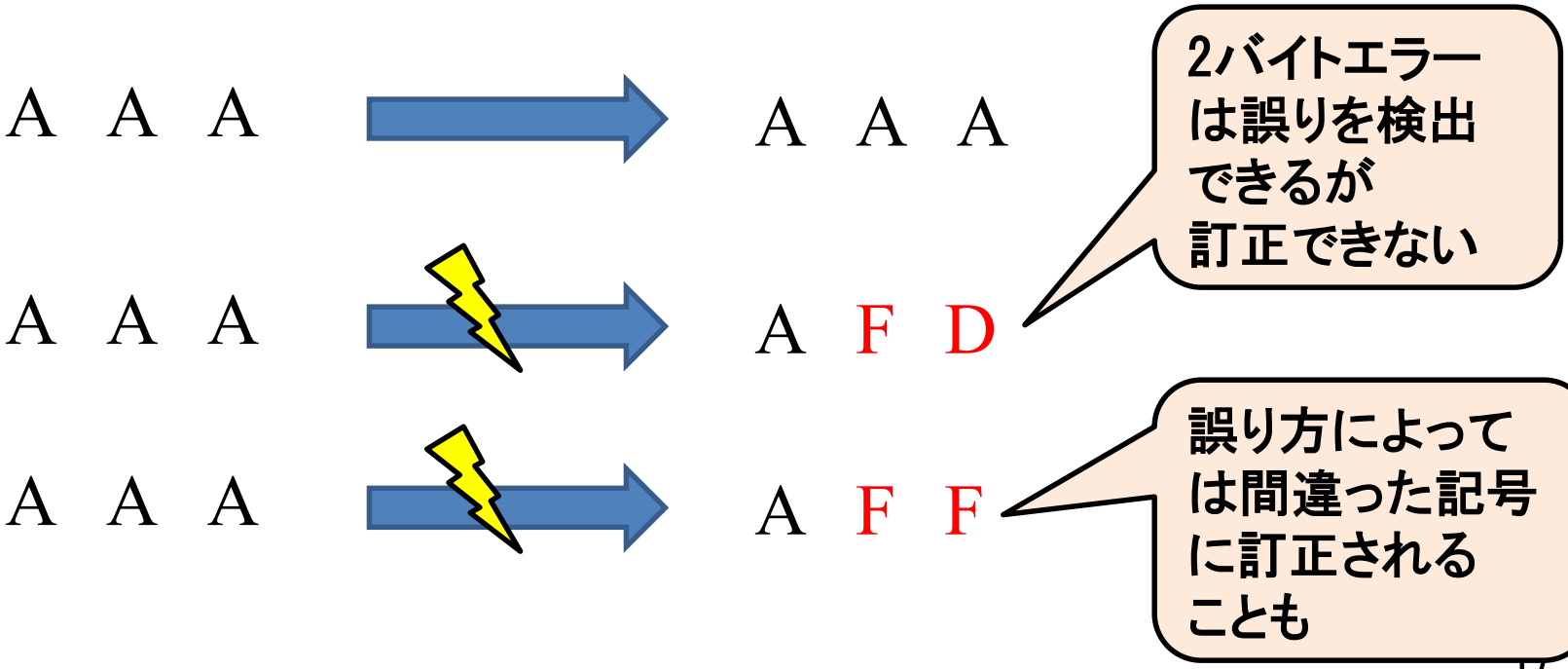


多く受信された「A」を正しい値と判定
（1バイトのエラーまでは検出＆訂正可能）

誤り訂正符号

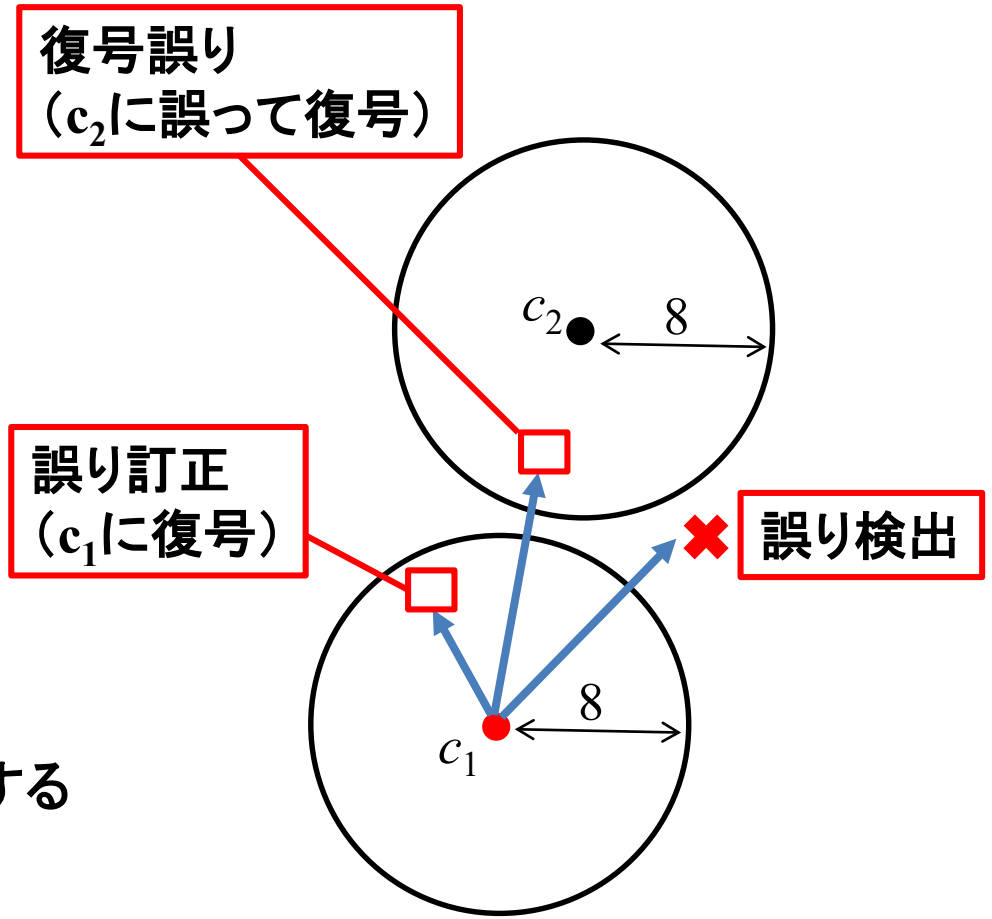
- 送受信されるデータにノイズが入ってエラーが起こったとしても検出・訂正を可能にする技術
- 例）多数決法

誤り訂正符号（多数決法）（Aという文字を送りたい場合）



QRコードの誤り訂正符号の能力

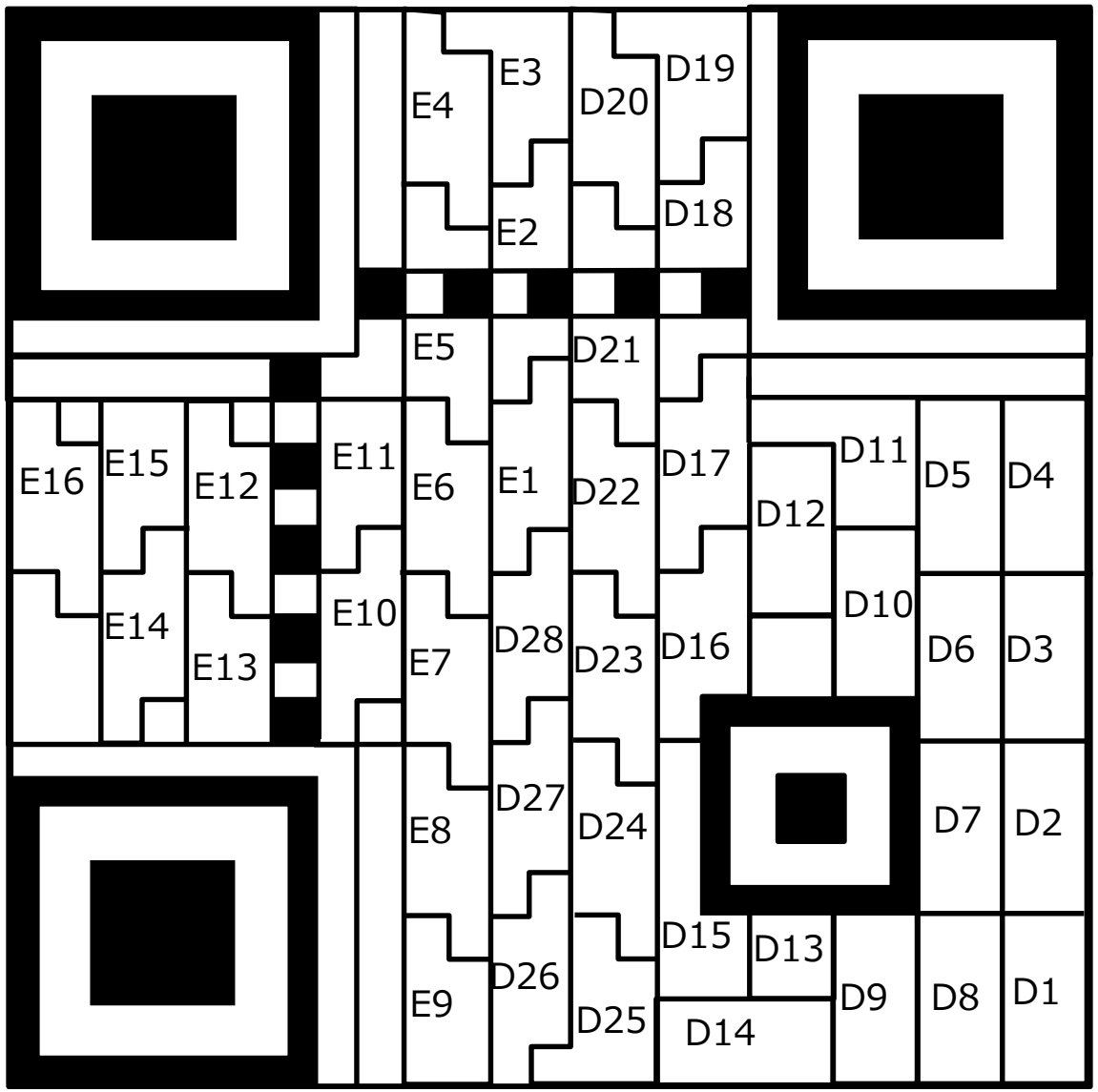
- 2-M型のQRコード
  - 誤り訂正能力：8バイトまでの誤りなら正しく読み込める



c_1：正規のURLから生成したQRコードのデータ

c_2：正規URL以外のURLに対応するQRコードのデータ

QRコードのモジュール配置（2-M型QRコード）



1モジュール
（白黒の四角の最小単位）
で1ビットを表現

- D1～D28は
  データ本体
- E1～E16は
  誤り訂正のために
  追加したデータ

実習1: QRコードの描画

- QRコード用データ(D1～D28、E1～E16)を入力すると対応する2-M型QRコードが描画されるシステム
  - 本来はモジュール(正方形の□や■)を自身で打ってもらいたかったが、マスク処理がややこしいのでやめた
  - http://150.7.136.43/~ocsl1501/phpgenqr/
  - フォームにカンマ区切り(csv形式)で44バイト分のデータを入力するとQRコードが描画される

http://www.u-tokai.ac.jp/ のQRコードのデータ(カンマ区切り)

65,150,135,71,71,3,162,242,247,119,119,114,231,82,215,70,246,182,22,146,230,22,50,230,167,2,240,236,36,111,113,168,84,82,21,223,143,148,4,29,86,247,145,151

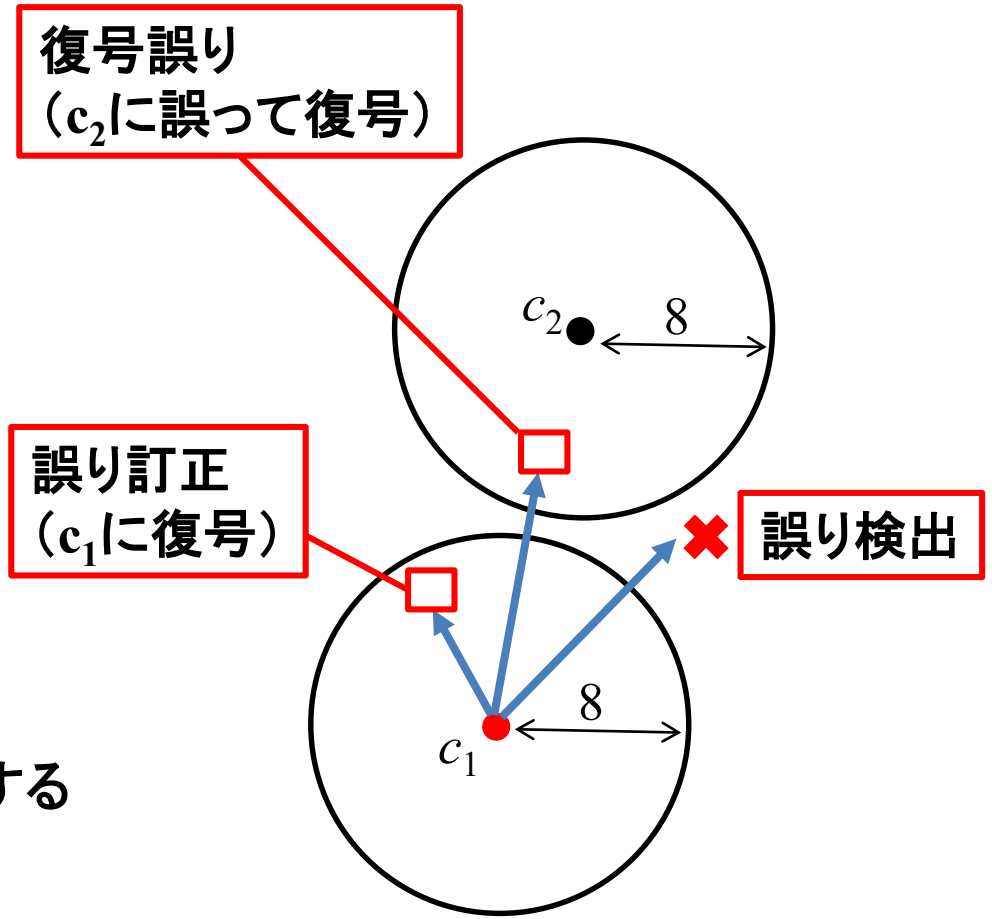
実習1: QRコードの描画

- QRコード用データ(D1～D28、E1～E16)を入力すると対応する2-M型QRコードが描画されるシステム
  - http://150.7.136.43/~ocsl1501/phpgenqr/

65,150,135,71,71,3,162,242,247,119,119,
114,231,82,215,70,246,182,22,146,230,22
,50,230,167,2,240,236,36,111,113,168,84,
82,21,223,143,148,4,29,86,247,145,151

QRコードの誤り訂正符号の能力

- 2-M型のQRコード
  - 誤り訂正能力：8バイトまでの誤りなら正しく読み込める

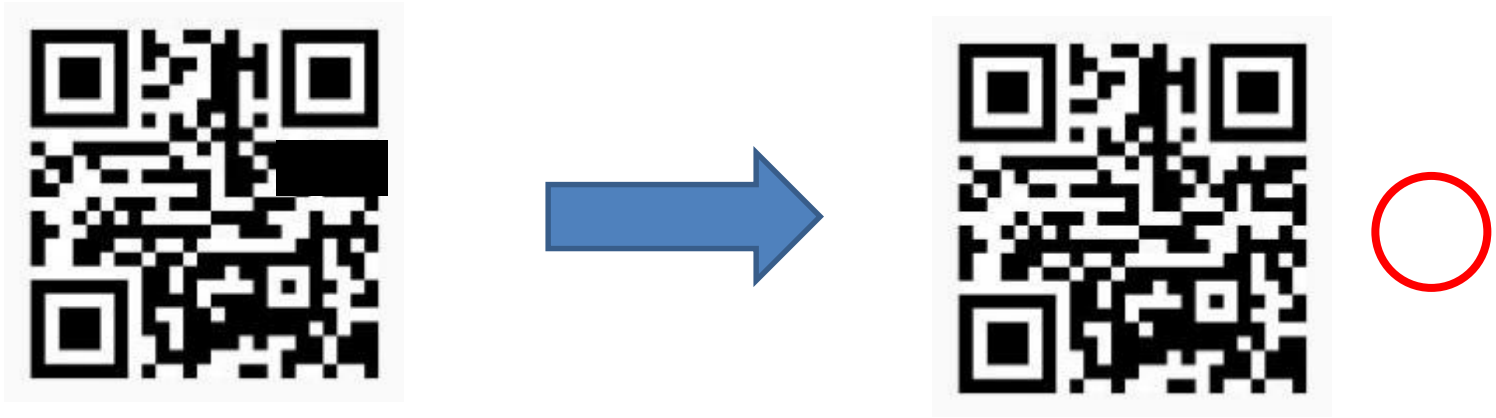


c_1：正規のURLから生成したQRコードのデータ

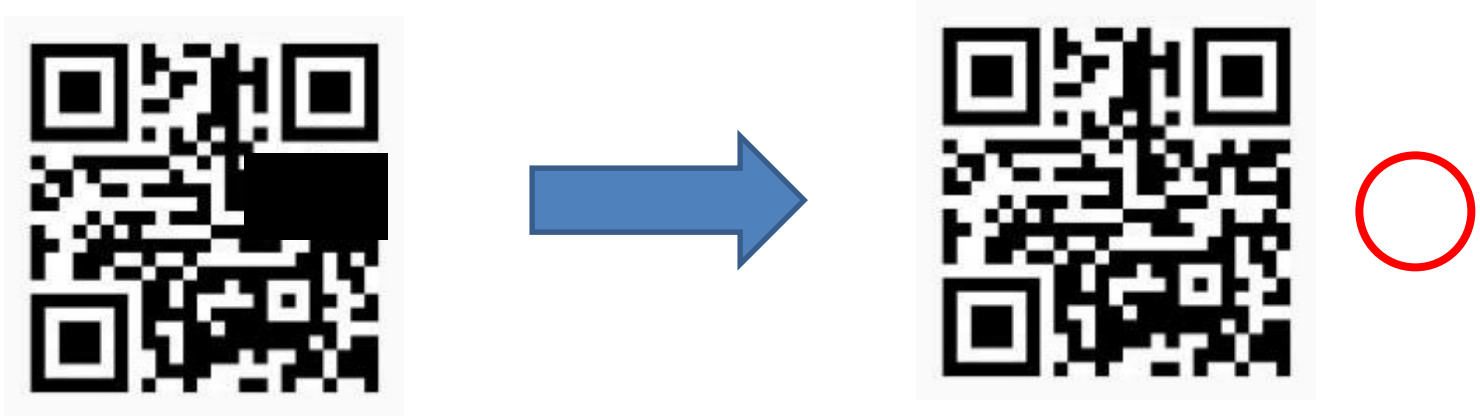
c_2：正規URL以外のURLに対応するQRコードのデータ

QRコードの誤り訂正能力

汚れ小

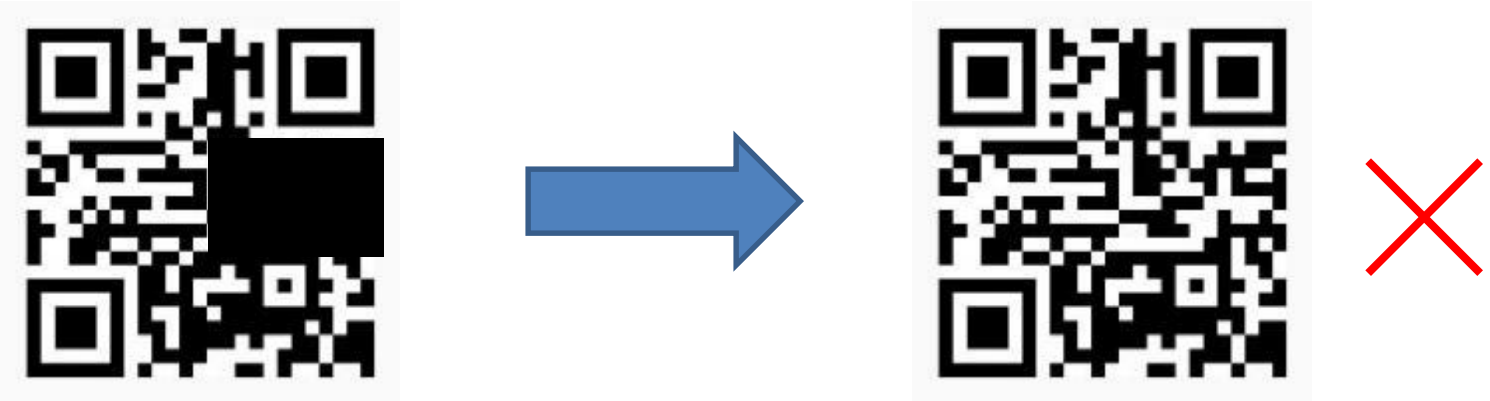


汚れ中



QRコードの誤り訂正能力

汚れ大



実習2: QRコードの誤り訂正能力

- 以下のQRコード用データ(D1～D28、E1～E16)ををQRコード描画システムへコピー＆ペーストした後、
  - 4バイト分誤り(違うデータ)に変更してQRコードを作り、読み込んでみよ
  - 8バイト分誤り(違うデータ)に変更してQRコードを作り、読み込んでみよ
  - 12バイト分誤り(違うデータ)に変更してQRコードを作り、読み込んでみよ

http://www.u-tokai.ac.jp/ のQRコードのデータ(カンマ区切り)

65,150,135,71,71,3,162,242,247,119,119,114,231,82,215,70,246,182,22,146,230,22,50,230,167,2,240,236,36,111,113,168,84,82,21,223,143,148,4, 29,86,247,145,151

大東担当分（第4回、第5回）でやること

- 第4回　QRコードを作ってみよう
  - QRコードがどういうフォーマットで作られているかの理解
  - 自分で決めたURLにアクセスさせるデータを作成
  - リードソロモン符号による冗長化
    → 山本先生分のときに作ったプログラムを利用
  - 画像として印字
    → Webシステムとして用意したのでそれを使う

- 第5回　偽装QRコードを理解しよう
  - 撮影ごとに振る舞いが変化する（ことがある）
    悪性QRコードの原理を理解する
  - 実際に自分で作ってみる
    → 第4回のプログラムをベースに改造

2-M型QRコードの符号語の例（25文字のデータの場合）

- モード識別子：0100（入力データが8bitバイトの場合）
- 文字数指示子：00011001 (25文字, 2進数1バイトで表記)
- データ本体 + 終端パターン(0000)
- 埋め草コード：誤り訂正ブロック以外のデータサイズが28バイトになるまで 236, 17 を繰り返す
- 誤り訂正ブロック：16バイトの冗長データ（モード識別子から埋め草コードまでの情報から計算）

http://www.u-tokai.ac.jp/ のQRコードのデータ

65 150 135 71 71 3 162 242 247 119 119 114
231 82 215 70 246 182 22 146 230 22 50 230
167 2 240 236 36 111 113 168 84 82 21 223
143 148 4 29 86 247 145 151

モード識別子

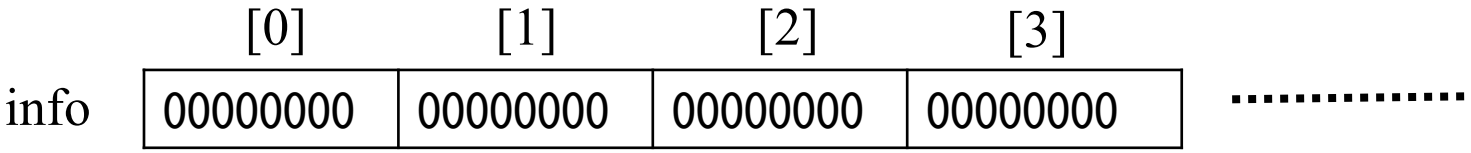
- データ本体がどのモードで符号化されているかを示す4ビットの識別子
  - 0001: 数字モード　　[3文字あたり10ビットで表現]
  - 0010: 英数字モード　　[2文字あたり11ビットで表現]
  - 0100: 8ビットバイトモード　[1文字あたり 8ビットで表現]
  - 1000: 漢字モード　　[漢字1文字あたり13ビットで表現]

モード識別子を配列に格納

- モード識別子：0100（入力データが8bitバイトの場合）

28バイト分のデータ用配列info[28]に値を詰めていく

```
// データ用配列の宣言（バイト単位，0で初期化）
unsigned char info[28] = {0};
```

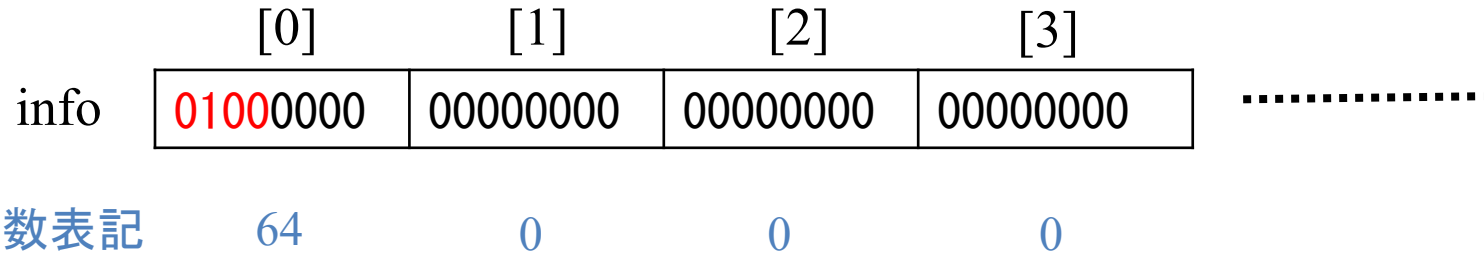


モード識別子を配列に格納

- モード識別子：0100（入力データが8bitバイトの場合）

28バイト分のデータ用配列info[28]に値を詰めていく

```
// データ用配列の宣言（バイト単位，0で初期化）
unsigned char info[28] = {0};
// infoの1バイト目に 0100 (10進数で4)を4ビット左シフトしてから加算
info[0] = info[0] + (4 << 4);
```



モード識別子を配列に格納

- モード識別子：0100（入力データが8bitバイトの場合）

28バイト分のデータ用配列info[28]に値を詰めていく

```
int i;
unsigned char info[28] = {0};
info[0] = info[0] + (4 << 4);
for(i = 0; i < 28; i++){
  printf("info[%d]=%d¥n", i, info[i]);
}
```

	[0]	[1]	[2]	[3]	
info	01000000	00000000	00000000	00000000	.....
10進数表記	64	0	0	0	

2-M型QRコードの符号語の例（25文字のデータの場合）

- モード識別子：0100（入力データが8bitバイトの場合）
- 文字数指示子：00011001 (25文字, 2進数1バイトで表記)
- データ本体 + 終端パターン(0000)
- 埋め草コード：誤り訂正ブロック以外のデータサイズが28バイトになるまで 236, 17 を繰り返す
- 誤り訂正ブロック：16バイトの冗長データ（モード識別子から埋め草コードまでの情報から計算）

http://www.u-tokai.ac.jp/ のQRコードのデータ

65 150 135 71 71 3 162 242 247 119 119 114
231 82 215 70 246 182 22 146 230 22 50 230
167 2 240 236 36 111 113 168 84 82 21 223
143 148 4 29 86 247 145 151

文字数指示子を配列に格納

- 文字数指示子: 00011001 (25文字, 2進数1バイトで表記)
  - 違うモードでは文字数指示子のビット数が異なる場合もある

28バイト分のデータ用配列info[28]に値を詰めていく

```
int i;
unsigned char info[28] = {0};
info[0] = info[0] + (4 << 4);
info[0] = info[0] + (25 >> 4);
info[1] = info[1] + (25 << 4);
for(i = 0; i < 28; i++){
  printf("info[%d]=%d¥n", i, info[i]);
}
```

	[0]	[1]	[2]	[3]	
info	01000001	10010000	00000000	00000000	.....
10進数表記	65	144	0	0	

文字数指示子を配列に格納

- 文字数指示子: 00011001 (25文字, 2進数1バイトで表記)
  - 違うモードでは文字数指示子のビット数が異なる場合もある

28バイト分のデータ用配列info[28]

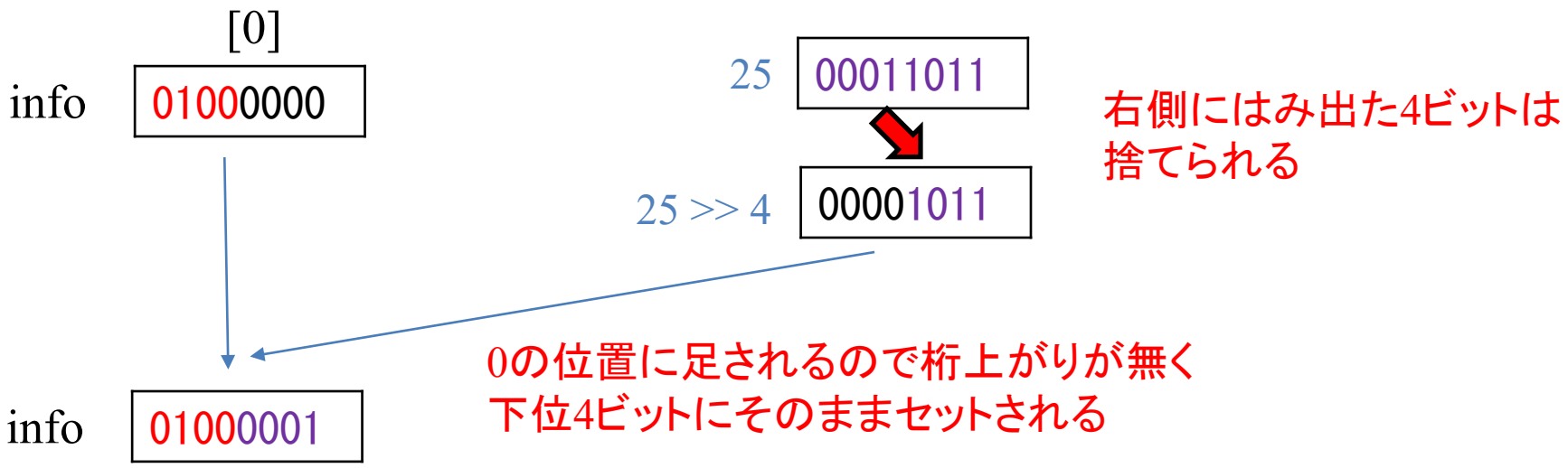
```
int i;
unsigned char info[28] = {0};
info[0] = info[0] + (4 << 4);
info[0] = info[0] + (25 >> 4);
info[1] = info[1] + (25 << 4);
for(i = 0; i < 28; i++){
  printf("info[%d]=%d¥n", i, info[i]);
}
```

例としてデータ本体の長さを25にしているが、実際はstrlen()関数やstrnlen()関数などで求める必要がある

	[0]	[1]	[2]	[3]	
info	01000001	10010000	00000000	00000000	.....
10進数表記	65	144	0	0	

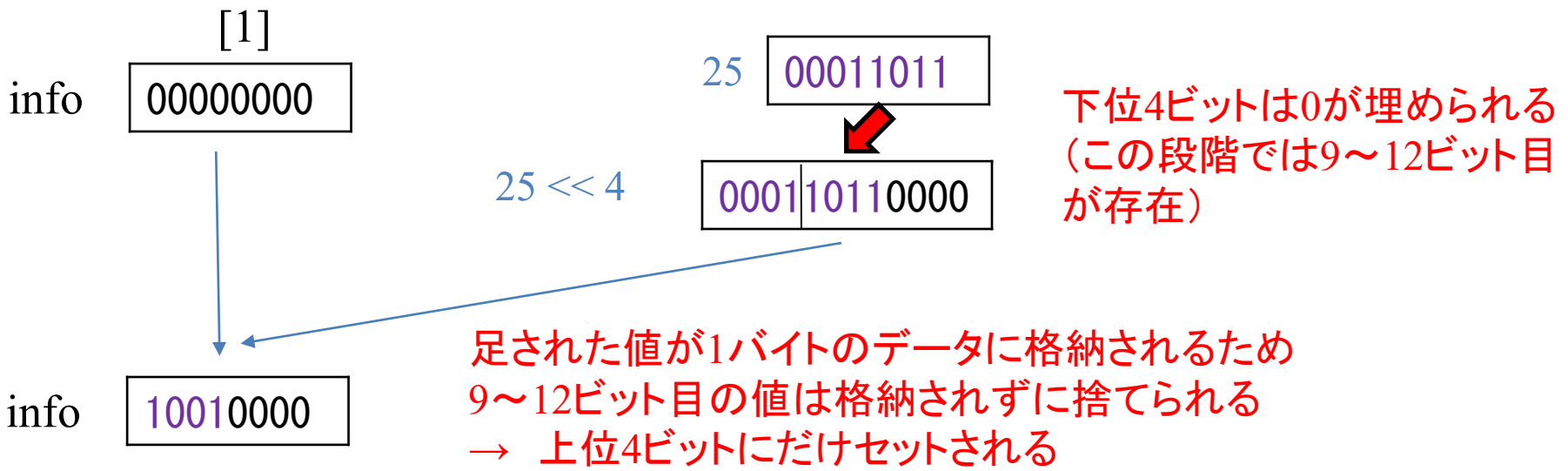
ずれた場所にビットをセットする際のテクニック

```
info[0] = info[0] + (25 >> 4);
```



ずれた場所にビットをセットする際のテクニック

```
info[1] = info[1] + (25 << 4);
```



文字数指示子を配列に格納

- 文字数指示子: 00011001 (25文字, 2進数1バイトで表記)
  - 違うモードでは文字数指示子のビット数が異なる場合もある

28バイト分のデータ用配列info[28]に値を詰めていく

```
int i;
unsigned char info[28] = {0};
info[0] = info[0] + (4 << 4);
info[0] = info[0] + (25 >> 4);
info[1] = info[1] + (25 << 4);
for(i = 0; i < 28; i++){
  printf("info[%d]=%d¥n", i, info[i]);
}
```

このあと、同様に1バイトのデータを4ビットずつセットするところがあるが同様にやれば良い

	[0]	[1]	[2]	[3]	
info	01000001	10010000	00000000	00000000	.....
10進数表記	65	144	0	0	

2-M型QRコードの符号語の例（25文字のデータの場合）

- モード識別子：0100（入力データが8bitバイトの場合）
- 文字数指示子：00011001 (25文字, 2進数1バイトで表記)
- データ本体 + 終端パターン(0000)
- 埋め草コード：誤り訂正ブロック以外のデータサイズが28バイトになるまで 236, 17 を繰り返す
- 誤り訂正ブロック：16バイトの冗長データ（モード識別子から埋め草コードまでの情報から計算）

http://www.u-tokai.ac.jp/ のQRコードのデータ

65 150 135 71 71 3 162 242 247 119 119 114
231 82 215 70 246 182 22 146 230 22 50 230
167 2 240 236 36 111 113 168 84 82 21 223
143 148 4 29 86 247 145 151

データ本体（URL）の配列

- URLの文字列をunsigned char型の配列で宣言
  - 文字列も同時にセットしておく
  - strlen関数でurlのサイズも取得
    (string.hをincludeしておく必要あり)

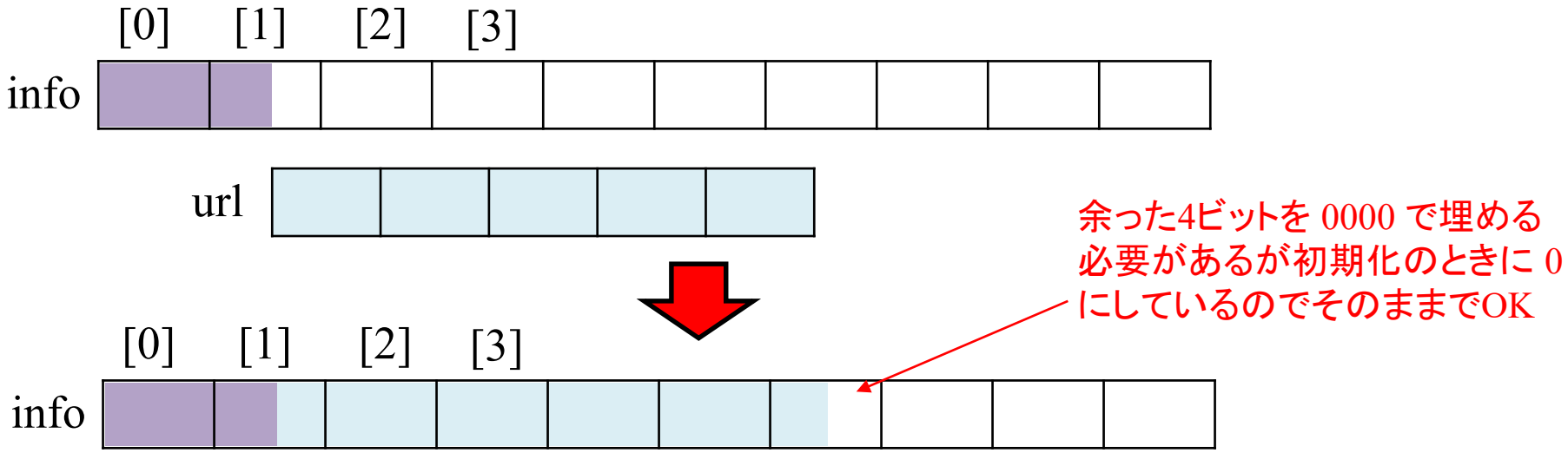
```
#include<stdio.h>
#include<string.h>

int main(void){
  unsigned char url[] = "http://www.u-tokai.ac.jp/";
  int size;
  size = strlen(url);
  printf("%d byte: %s¥n", size, url);

  return 0;
}
```

データ本体を配列に格納

- 1.5バイト目以降にデータ本体を格納する
  - URLの各バイトを4ビットずつセットしていかないとならない
  - 文字数指示子の要領で繰り返し命令（for文）を使って埋め込む



url[] = “http://www.u-tokai.ac.jp/” の場合のデータ

65 150 135 71 71 3 162 242 247 119 119 114
231 82 215 70 246 182 22 146 230 22 50 230
167 2 240

2-M型QRコードの符号語の例（25文字のデータの場合）

- モード識別子：0100（入力データが8bitバイトの場合）
- 文字数指示子：00011001 (25文字, 2進数1バイトで表記)
- データ本体 + 終端パターン(0000)
- 埋め草コード：誤り訂正ブロック以外のデータサイズが28バイトになるまで 236, 17 を繰り返す
- 誤り訂正ブロック：16バイトの冗長データ（モード識別子から埋め草コードまでの情報から計算）

http://www.u-tokai.ac.jp/ のQRコードのデータ

65 150 135 71 71 3 162 242 247 119 119 114
231 82 215 70 246 182 22 146 230 22 50 230
167 2 240 236 36 111 113 168 84 82 21 223
143 148 4 29 86 247 145 151

2-M型QRコードの符号語の例（25文字のデータの場合）

- モード識別子：0100（入力データが8bitバイトの場合）
- 文字数指示子：00011001 (25文字, 2進数1バイトで表記)
- データ本体 + 終端パターン(0000)
- 埋め草コード：誤り訂正ブロック以外のデータサイズが28バイトになるまで 236, 17 を繰り返す
- 誤り訂正ブロック：16バイトの冗長データ（モード識別子から埋め草コードまでの情報

1バイト余っている場合 236 をセット
2バイト余っている場合 236 17
3バイト余っている場合 236 17 236
以下同様に

http://www.u-tokai.ac.jp/

65 150 135 71 71 3 [illegible]
231 82 215 70 [illegible] 182 22 146 230 22 50 230
167 2 240 236 36 111 113 168 84 82 21 223
143 148 4 29 86 247 145 151

2-M型QRコードの符号語の例(25文字のデータの場合)

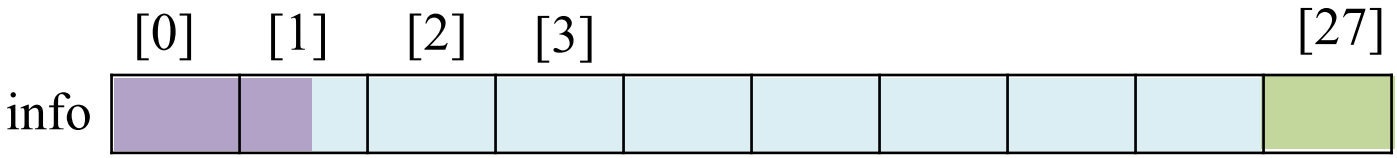
- モード識別子: 0100 (入力データが8bitバイトの場合)
- 文字数指示子: 00011001 (25文字, 2進数1バイトで表記)
- データ本体 + 終端パターン(0000)
- 埋め草コード: 誤り訂正ブロック以外のデータサイズが28バイトになるまで 236, 17 を繰り返す
- 誤り訂正ブロック: 16バイトの冗長データ (モード識別子から埋め草コードまでの情報から計算)

http://www.u-tokai.ac.jp/ のQRコードのデータ

65 150 135 71 71 3 162 242 247 119 119 114
231 82 215 70 246 182 22 146 230 22 50 230
167 2 240 236 36 111 113 168 84 82 21 223
143 148 4 29 86 247 145 151

QRコード用の符号語を作成

- 第3回授業の課題で作ったプログラムを利用して
  誤り訂正ブロックを作成し完成させる



$i(x)$

65 150 135 71 71 3 162 242 247 119 119 114
231 82 215 70 246 182 22 146 230 22 50 230
167 2 240 236

$r(x)$

36 111 113 168 84 82 21 223
143 148 4 29 86 247 145 151

2-M型QRコードの符号語の例（25文字のデータの場合）

- モード識別子：0100（入力データが8bitバイトの場合）
- 文字数指示子：00011001 (25文字, 2進数1バイトで表記)
- データ本体 + 終端パターン(0000)
- 埋め草コード：誤り訂正ブロック以外のデータサイズが28バイトになるまで 236, 17 を繰り返す
- 誤り訂正ブロック：16バイトの冗長データ（モード識別子から埋め草コードまでの情報から計算）

http://www.u-tokai.ac.jp/ のQRコードのデータ

65 150 135 71 71 3 162 242 247 119 119 114
231 82 215 70 246 182 22 146 230 22 50 230
167 2 240 236 36 111 113 168 84 82 21 223
143 148 4 29 86 247 145 151

実習3: QRコードのデータの作成

- QRコードのフォーマットに従ってデータ生成プログラムを作成せよ
  - 前回の課題が出来なかった人は山本先生提供のRSencode_M2()まで書いてある encode.c を使ってmain関数内で前ページまでで説明したデータを作れば良い
- データの各バイトは10進数でカンマ区切りで出力せよ
- テスト用のURL http://www.u-tokai.ac.jp/ に対して以下の出力が出ることを確認せよ

http://www.u-tokai.ac.jp/ のQRコードのデータ(カンマ区切り)

65,150,135,71,71,3,162,242,247,119,119,114,231,82,215,70,246,182,22,146,230,22,50,230,167,2,240,236,36,111,113,168,84,82,21,223,143,148,4,29,86,247,145,151

実習4: オリジナルのQRコードの作成

- 作成したプログラムにより、
  以下の2種類のURLのQRコードを作成し画像を
  保存せよ
- URL1
  - http://www.学生番号.org/
  - 例: http://www.1cje3456.org/
- URL2
  - http://苗字のローマ字表記.net/
  - 例: http://ohigashi.net/

課題

- 実習3で作成したプログラムを提出せよ
  ファイル名: 学生番号_kadai42.c
  例) 1CJE3456_kadai42.c
- 実習4で作成したQRコードを提出せよ
  - URL1のQRコード: 学生番号_kadai43url1.jpg
  - URL2のQRコード: 学生番号_kadai43url2.jpg
    ※ 上記の画像フォーマットはjpegにしているが pngなど
    別のフォーマットでも良い
    例) 1CJE3456_kadai43url1.jpg
    1CJE3456_kadai43url2.jpg
- Teamsから提出
  締切: 2024/5/16(木)　23:59 (JST) 厳守

付録:
手動でQRコードを作るために

マスク処理

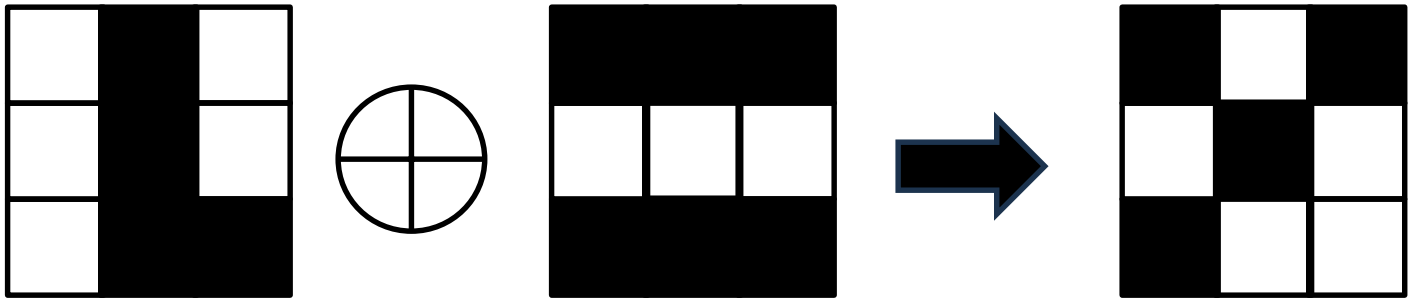
●マスク処理

- QRコードの生成時，読み取りの確実性を高めるために，明暗（白黒）のモジュールをシンボル内にバランスよく配置する処理

- マスク処理前のモジュールパターンとマスクパターンを重ね合わせ，XOR演算を行い，マスクパターンと重なった部分のモジュールパターンの明暗を反転させる

マスク処理実行例

処理前のモジュールパターン　　マスクパターン



マスクパターン

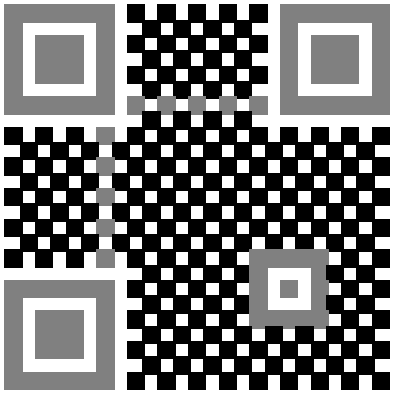
- 参照子0~7の8通り存在し，式で表せる
- iは行位置，jは列位置
- (i, j) = (0, 0)は左上のモジュール

0	(i＋j) mod 2＝0
1	i mod 2＝0
2	j mod 3＝0
3	(i＋j) mod 3＝0
4	((i div 2)＋(j div 3)) mod 2＝0
5	(i j) mod 2＋(i j) mod 3＝0
6	((i j) mod 2＋(i j) mod 3) mod 2＝0
7	((i＋j) mod 2＋(i j) mod 3) mod 2＝0

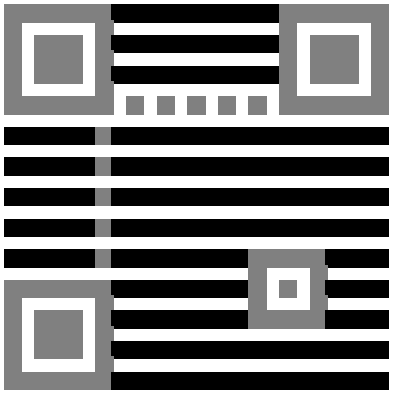
マスクパターンとマスク処理

- 黒のモジュールと重なった部分をXORして白黒を反転
  → 白（黒）のモジュールの塊が生まれないようにする

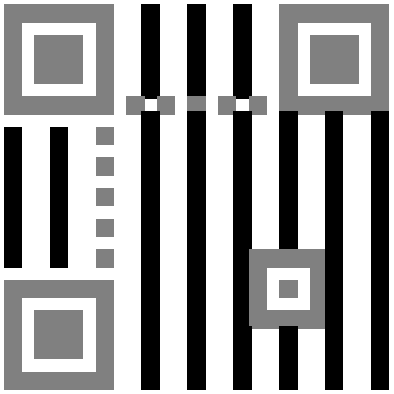
0



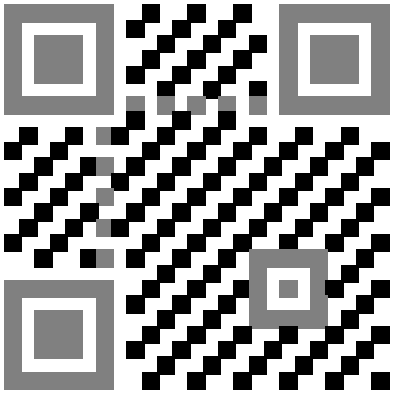
1



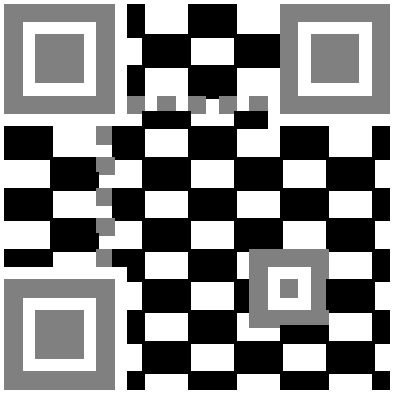
2



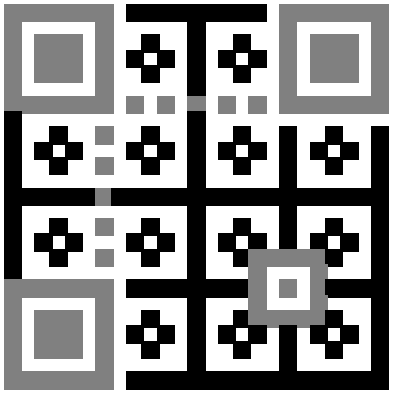
3



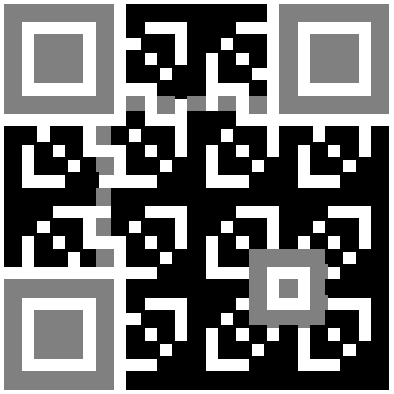
4



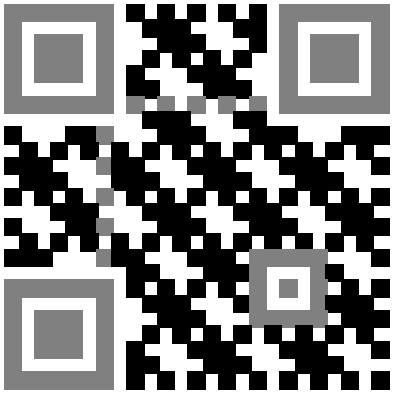
5



6

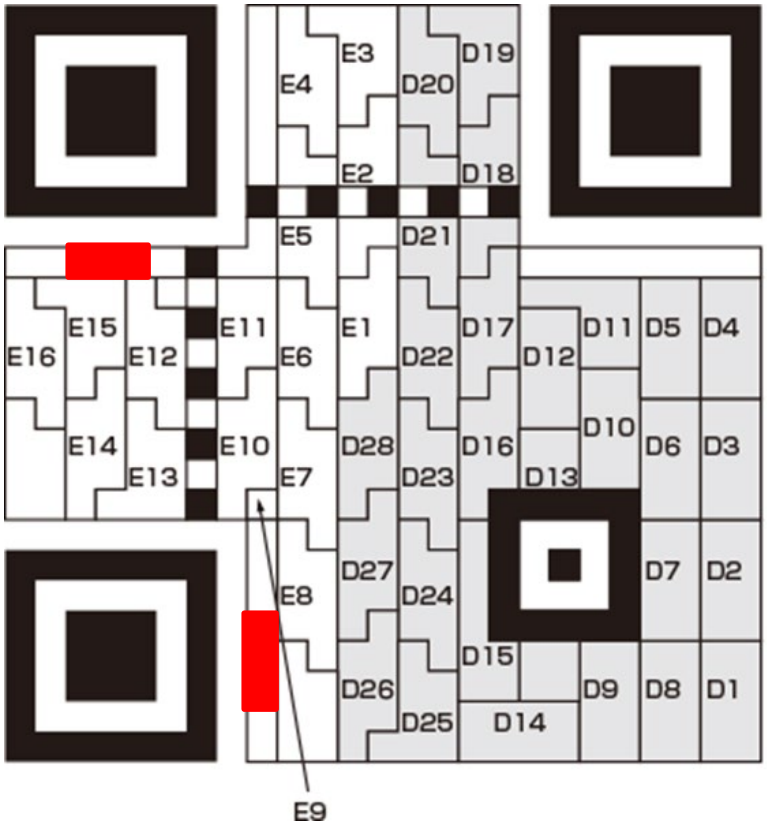


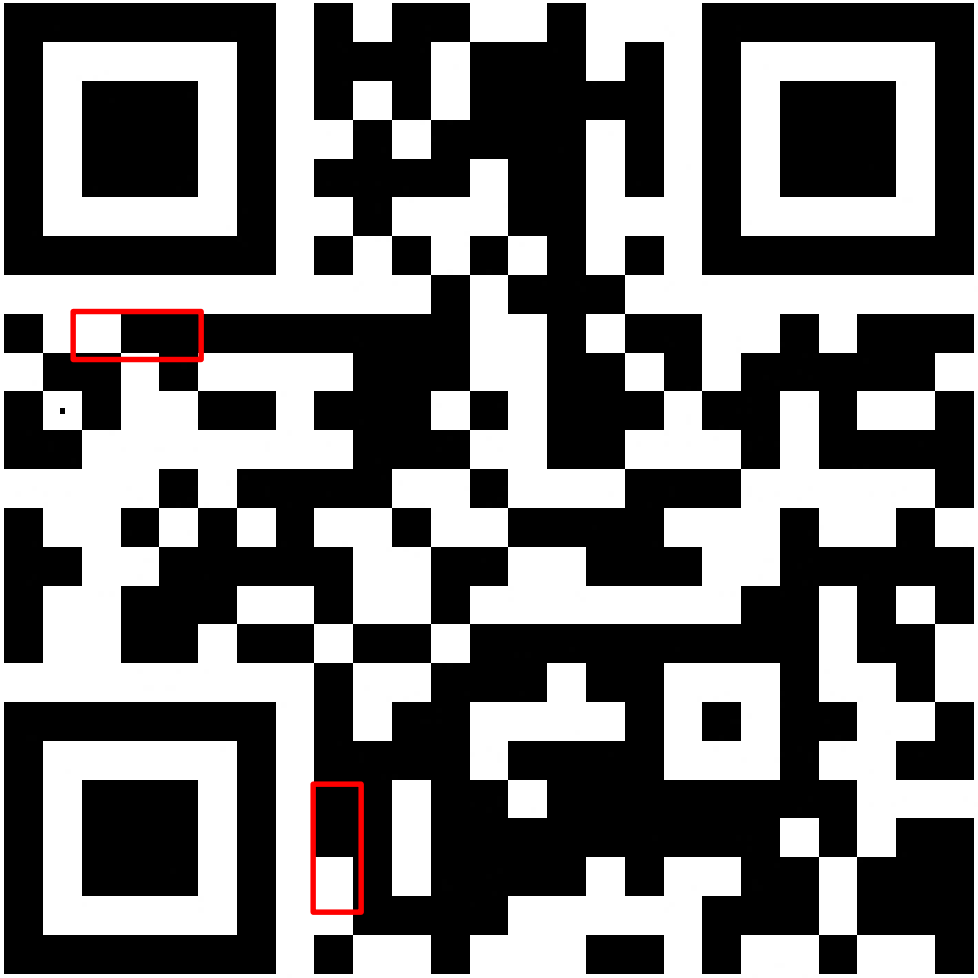
7



マスクパターンの判定

- マスクパターン情報の格納位置，変更方法
  - 読み込み時にマスクパターンの情報に従ってマスクを解除
  - 図中の赤部分にマスク情報が格納
  - 上の赤色の部分をビットに変換して 101 とXORするとマスクパターンが出てくる
    → □■■ (011)の場合
    101 とXORすると
    110 (マスクパターン6)





エクセル上で描画したビット位置