



ビットとビット演算

プログラマのためのC言語 第4回

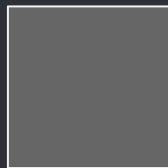
● 目次

- - ✓ ビットと情報量
 - ✓ ビット、ニブル、バイト
 - ✓ よくある表現方法
 - ✓ 論理演算
 - ✓ ビット演算
 - ✓ ビット演算の利用例

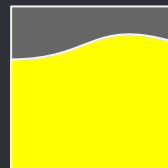
“

ビットと情報量

● ビット



電気蓄えてない



電気を蓄えてる

コンピュータの中には電気を蓄えたり放出したりできる箱が沢山ある

- ビット



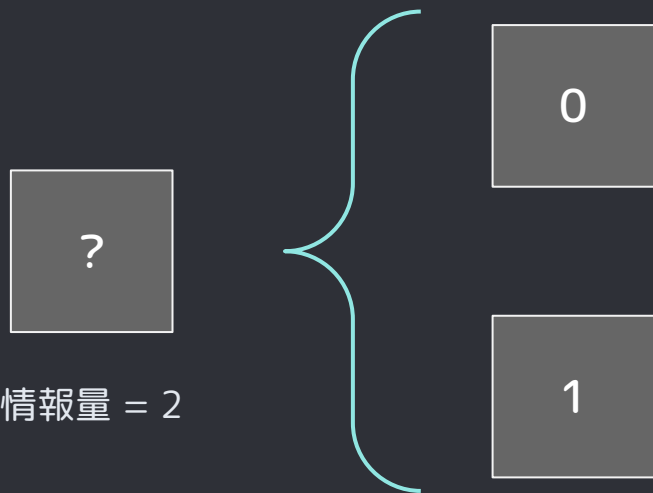
電気が空の状態を0、電気を蓄えている状態を1と表すことにする

- ビット



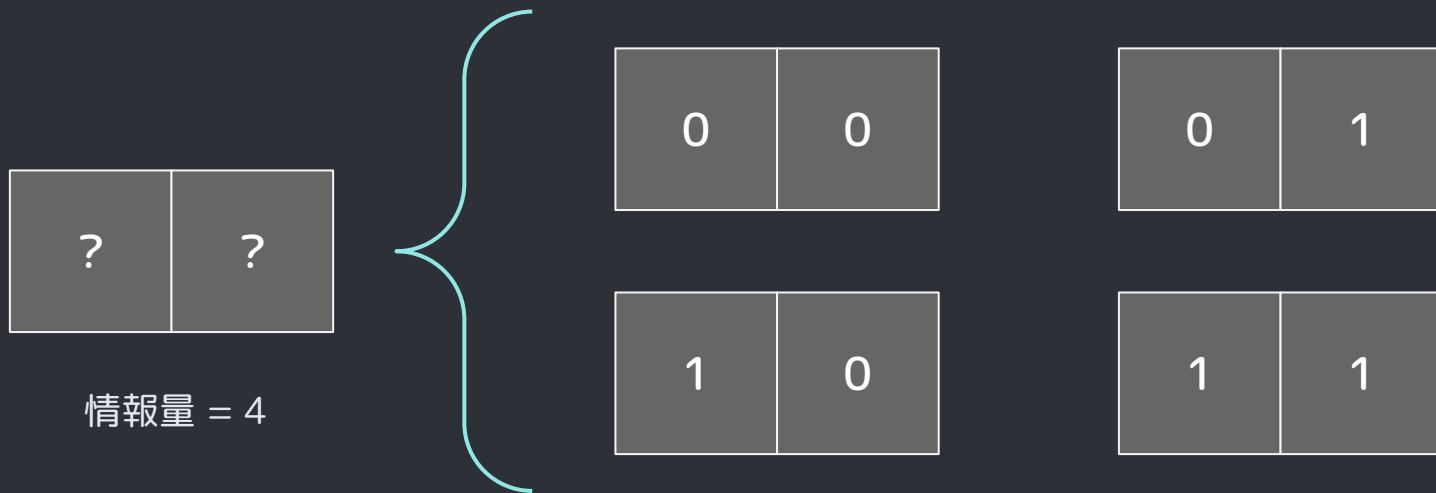
この箱が1つあると0と1の**2パターン**を表すことができる

● ビット



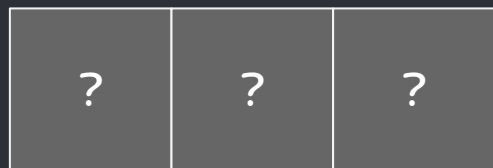
あるかないか、YESかNOか、0か1かなど
2通りの情報量を **ビット** といい、コンピュータが扱う情報の最小単位

情報量



箱を2つに増やすと扱える情報量は4パターンに増える

情報量



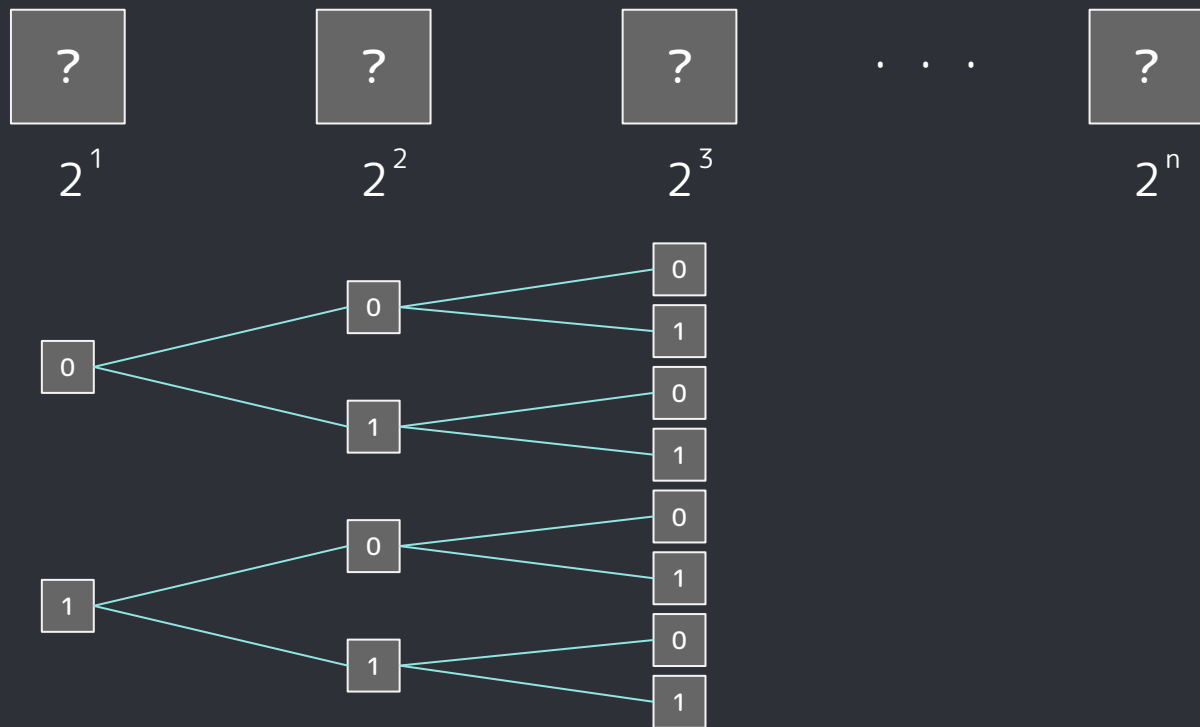
情報量 = 8



0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	1	1	0
0	1	1	1	1	1

箱を3つに増やすと扱える情報量は8パターンに増える

情報量



箱が増えるたびに扱える情報量は指数的(倍々)に増える

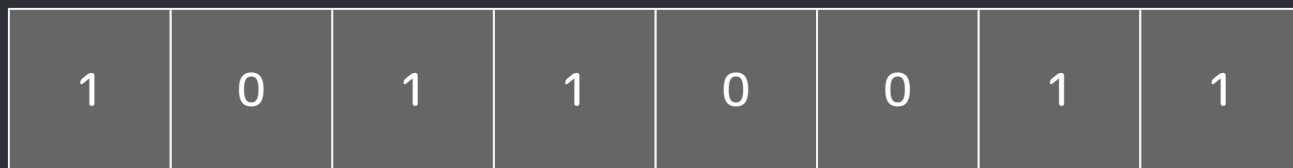
“

ビット、ニブル、バイト

- ビット、ニブル、バイト



- ビット、ニブル、バイト



ビット

情報量 = 2

ニブル

ニブル

情報量 = 16

バイト

バイト

情報量 = 256

● ビット、ニブル、バイト

最上位ビット

最下位ビット

7ビット

6ビット

5ビット

4ビット

3ビット

2ビット

1ビット

0ビット

1	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

ビット

情報量 = 2

ニブル

情報量 = 16

バイト

情報量 = 256

“

よくある表現方法

● よくある表現方法

対応表

0000 = 0	1000 = 8
0001 = 1	1001 = 9
0010 = 2	1010 = A
0011 = 3	1011 = B
0100 = 4	1100 = C
0101 = 5	1101 = D
0110 = 6	1110 = E
0111 = 7	1111 = F

これならニブルは1文字で表せる

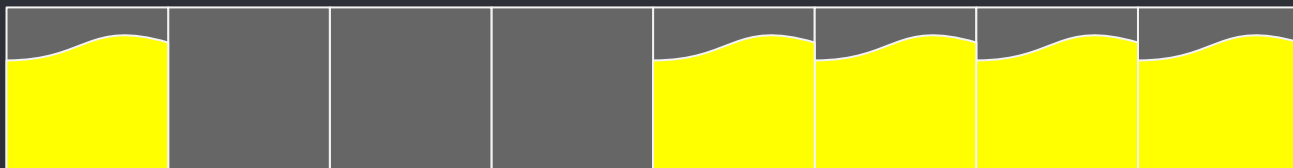
1	0	0	0	1	1	1	1
							
8				F			

0と1で表すと長いのでニブルを0~Fの1文字で表現するのが一般的

よくある表現方法

メモリ
アドレス: 0x00E71750

0x00E71750	55	8b	ec	81	ec	cc	00	00	00	53	56	57	8d	bd	3
0x00E71760	ff	ff	b9	33	00	00	00	b8	cc	cc	cc	cc	f3	ab	b
0x00E71770	c0	e7	00	e8	8f	fb	ff	ff	c7	45	f8	00	00	00	0
0x00E71780	c0	5f	5e	5b	81	c4	cc	00	00	00	3b	ec	e8	9f	f
0x00E71790	ff	8b	e5	5d	c3	cc	cc	cc	cc	cc	cc	cc	cc	cc	c



メモリをより細かくみたイメージ

バイト

ニブル

ビット

電気

“

論理演算

● 論理演算

true false を使った演算

1

0

真偽値(true, false)を使った演算
コンピュータだとtrueを1、falseを0と考えてやることが多い

● 論理演算

論理演算 四天王



NOT



AND



OR





XOR

コンピューターは論理演算だけで全てをこなす

● 論理演算



○ NOT (論理否定)

NOT 0		1
NOT 1		0
単項演算		

否定 0は1、1は0に

● 論理演算


○ NOT (論理否定)

NOT 0		1
NOT 1		0
単項演算		

◆ 単項演算と二項演算の補足

-2



この - 演算は2をマイナスにするという
1つの項に対する演算  単項演算





1 + 2



この + 演算は2つの項を加えるという
2項を扱う演算  二項演算

● 論理演算

○ AND (論理積)



0 AND 0		0
0 AND 1		0
1 AND 0		0
1 AND 1		1

二項演算

両方 1 なら 1

● 論理演算

○ OR (論理和)

0 OR 0		0
0 OR 1		1
1 OR 0		1
1 OR 1		1

二項演算

どっちなか 1 なら 1

● 論理演算

○ XOR (排他的論理和)

0 XOR 0		0
0 XOR 1		1
1 XOR 0		1
1 XOR 1		0

二項演算

双方が異なるときだけ 1

“

ビット演算

- ビット演算

- ビット演算の種類

- ✓ NOT
- ✓ AND
- ✓ OR
- ✓ XOR
- ✓ 左シフト
- ✓ 右シフト

- ビット演算

- NOT

```
char a = 0b10001010;  
char not = ~a;
```

```
// ~ 10001010
```

```
// -----
```

```
// .. 01110101
```

bit単位で否定(反転)、記号は ~ (チカダ)を使う

- ビット演算

AND

```
char a = 0b10001010;  
char b = 0b11110000;  
char and = a & b;  
  
// ... 1000 1010  
// & 1111 0000  
// -----  
// ... 1000 0000 → 0x80
```

ビット単位で論理積、記号は & (アンド)を使う

- ビット演算

- OR

```
char a = 0b10001010;  
char b = 0b11110000;  
char or = a | b;  
  
// 1000 1010  
// | 1111 0000  
// -----  
// 1111 1010 → 0xFA
```

ビット単位で論理和、記号は | (パイプ) を使う

- ビット演算

XOR

```
char a = 0b10001010;  
char b = 0b11110000;  
char xor = a ^ b;  
  
// ... 1000 1010  
// ^ 1111 0000  
// -----  
// ... 0111 1010 → 0x7A
```

ビット単位で排他的論理和、記号は ^ (ハット)を使う

- ビット演算

- 左シフト

```
char a = 1 << 0; // 00000001  
char b = 1 << 1; // 00000010  
char c = 1 << 2; // 00000100  
char d = 1 << 3; // 00001000
```



ビット単位で左に動かす、記号は << を使う

- ビット演算

- 右シフト

```
char a = 0b00001000 >> 0; // 00001000  
char b = 0b00001000 >> 1; // 00000100  
char c = 0b00001000 >> 2; // 00000010  
char d = 0b00001000 >> 3; // 00000001
```



ビット単位で右に動かす、記号は >> を使う

“

ビット演算の利用例

- ビット演算の利用例

- - ✓ ビットフラグ

- ・ ゲームの攻撃属性の判定とか
- ・ Linuxのアクセス権 rwx とか

- ✓ ビットマスク

- ・ IPアドレスのサブネットマスクとか

- ✓ etc...

- ビット演算の利用例

ビットフラグ

				土	風	水	火
0	0	0	0	0	0	0	0

各ビットを属性フラグとして使う

- ビット演算の利用例

ビットフラグ

弱点属性

				土	風	水	火
0	0	0	0	0	1	0	1

例えば弱点属性が火と風の2つならこんな感じ

- ビット演算の利用例

ビットフラグ

弱点属性

				土	風	水	火
0	0	0	0	0	1	0	1

攻撃属性

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

攻撃属性に火と水をセット

- ビット演算の利用例

ビットフラグ

弱点属性

0

0

0

0

0

1

0

1

攻撃属性

&

0

0

0

0

0

1

1

0 <

0

0

0

0

0

0

1

土

風

水

火

攻撃属性に弱点が含まれる場合、論理積の結果は0より大きくなる

- ビット演算の利用例

ビットフラグ



Linuxのアクセス権限とか？

- ビット演算の利用例

ビットマスク

IPアドレス

192.168.1.5

マスク

255.255.255.0

ネットワーク部

&

11000000 10101000 00000001 00000101

11111111 11111111 11111111 00000000

11000000 10101000 00000001 00000000

ビット列中の一部分を取得する

- ビット演算の利用例

- ビットマスク

IPアドレス 192.168.1.5	11000000 10101000 00000001 00000101
NOT マスク ~ 255.255.255.0	& 00000000 0000 000 00000000 11111111
ホスト部	00000000 00000000 00000000 00000101

NOTしたマスクを使えばホスト部がとれる

● ビット演算の利用例

- ✓ 部分和を求める
- ✓ 探索アルゴリズムに使う
- ✓ 乱数生成

...

などなど探せばいろんな活用方法があります



おしまい