入力の文字としての数字をどう"整数"に変換するか?

char型データ　（1byte）のサイズの数　（~~文字型~~）

1byte ← 2進8ケタ / 16進2ケタ

```
#include <stdio.h>
int main() {
    char a;
    char b;
    a=2;
    b=a*a*a*a;          printf("%d\n", b)
    b=a*a*a*a*a*a*a;    printf("%d\n", b);
    b= 8回かける ;         "
}
```

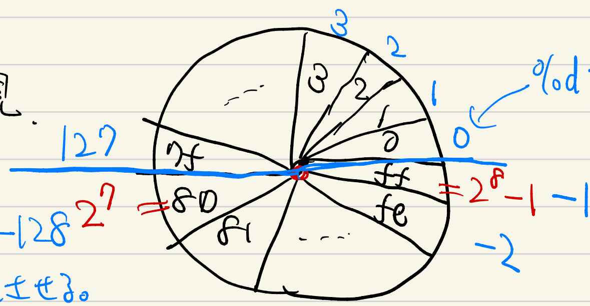
$16=2^4$ → 10進で表示 → 出力 16

$128=2^7$ → −128

$256=2^8$ → 0

2の補数表現

mod計算の結果で、10進計算のふりをさせる。



printf("%c",b); は putchar(b) と同じ。

bをそのまま画面におくる。画面は文字コードとおもい文字を表示。 bに0x41（AのアスキーコードA）が入ってると A と表示

```
#include <stdio.h>
int main() {
    int i;
```

char型データ[10] 配列
in[0], in[1], …, in[9]

```
char in[10];
fgets(in, 10, stdin);
for (i=0; i<10; i++){
  printf("%x ", in[i]);
}
```

10 byte 以上はよまない。

← キーボードから文字列をよみこむ。おしたキーのアスキーコードが順に in に入る。

1 2 とキーをおしたら、in[0] に 0x31 (文字1のアスキーコード) in[1] に 0x32 (文字2のアスキーコード) が入る。

アスキーコードがわかる

%c にどうなるか？

修正。

```
int result;
result = 0;
for (i=0; i<10; i++){
  result = result*10 + (in[i]-0x30);
}
printf("%d \n", result);
```

in[i] が 0xa なら、for をぬける。

```
if (in[i]==0xa)
  break;
```

プログラム = 関数のあつまりとして書く。

```
int str2int(char in[]){
  int i;
  ☆
  return result;
}
```

↖ 中身はわすれてよい。

関数 str2int は、文字列 in を、数字の文字列とおもって、int 型に変換。

```
int main(){
```

```c
char in[10];
fgets(in, 10, stdin);
printf("%d \n", str2int(in));
}
```

⇒ コウギーページのmyinput.c

ケタ数の大きい数の素ぼくな扱い

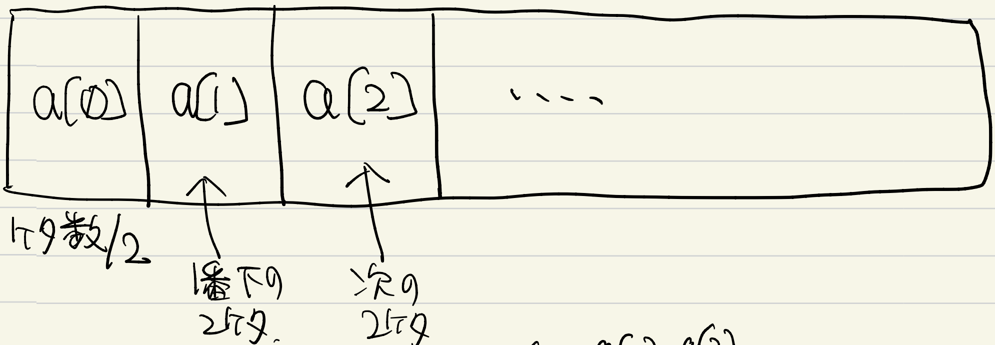
```c
unsigned char a[10];
```

a[i]には 0~255 ($=2^8-1$) を格納できる。

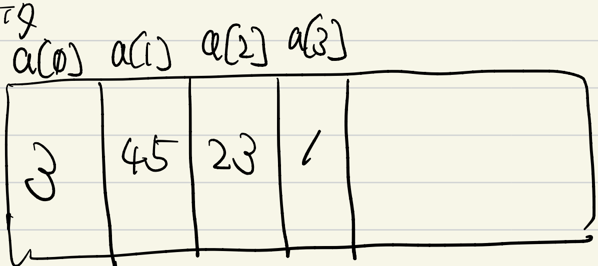
(10) → 大きくすれば、ケタをふやせる。

(unsigned) → 補数表現しない。

a[i]には0~99を入れる。



例1. 12345



Q この形式の整数表現に対してたし算をやりたい。

```c
unsigned char a[10];
unsigned char b[10];
unsigned char c[10];
c[0] = 1;
c[1] = 0;
```

仮定: aのケタ数 ≥ bのケタ数

c[10] ← a+b の結果

```
for (i=1; i<=a[0]; i++) {
    if (i<=b[0]) bb=b[i]; else bb=0;
    c[i]=c[i]+a[i]+bb;
    if (c[i]>=100) {
        c[i+1]=1; c[i]=c[i]%100;
        c[0]=i+1;
    } else {
        c[0]=i; c[i+1]=0;
    }
}
```

くり上がりを あらかじめ 入れておく。

c[i]を100で わった 余り。

くりあがりが あるときの処理。

したの 小学生で やった、筆算法。

→ 本質的に これより 早い方法は 知られていない。

かけ算については、~~筆算法~~。

Fast Fourier Transformation (FFT) による
$\mathbb{Z}$ でのかけ算.

多項式のかけ算

$$F = c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1}$$

$$G = d_0 + d_1 x + d_2 x^2 + \cdots + d_{n-1} x^{n-1}$$

$0 \le c_i, d_i < x-1$

$\Rightarrow$ x を数とみれば x 進数の表現.

かけ算のくりあがりがなければかけ算そのもの.

$F(x)G(x)$ を計算したい.

$$FG = c_0 d_0 + (c_0 d_1 + c_1 d_0) x + (c_0 d_2 + c_1 d_1 + c_2 d_0) x^2 + \cdots$$

係数 c_i, d_i のかけ算の回数は n^2 回.

係数のかけ算の回数をもっとへらしたい.

ω 1の n 乗根. $\omega = \exp\left(\frac{2\pi\sqrt{-1}}{n}\right)$

$$\omega^n = 1.$$

$n \times n$ 行列 $A = (\omega^{ij})_{0 \le i,j < n}$ $B = (\omega^{-ij})_{0 \le i,j < n}$

$n=3$

$$A = \begin{pmatrix} \omega^{0\cdot0} & \omega^{0\cdot1} & \omega^{0\cdot2} \\ \omega^{1\cdot0} & \omega^{1\cdot1} & \omega^{1\cdot2} \\ \omega^{2\cdot0} & \omega^{2\cdot1} & \omega^{2\cdot2} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \omega & \omega^2 \\ 1 & \omega^2 & \omega \end{pmatrix}$$

Prop. $AB = nE_n$, E_n は $n \times n$ 単位行列.

AB の ij 成分は. $\sum_{k=0}^{n-1} \omega^{ik} \omega^{-kj} = \sum_{k=0}^{n-1} \omega^{k(i-j)}$ ……①

$i=j$ のとき $\omega^{k(i-j)} = \omega^0 = 1$. $\therefore$ ① $= n$

$i \neq j$ のとき. $\sum_{k=0}^{n-1} (\omega^{(i-j)})^k = \dfrac{1-(\omega^{(i-j)})^n}{1-\omega^{(i-j)}}$

$(\omega^{(i-j)})^n = (\omega^n)^{i-j} = 1^{i-j} = 1$ $\Longrightarrow$ $\bigcirc$ $\therefore$ ① $= 0$ //

A の ij 成分

$$F(\omega^i) = \sum_{j=0}^{n-1} C_j (\omega^i)^j = \sum_{j=0}^{n-1} \omega^{ij} C_j .$$

$$\begin{pmatrix} F(\omega^0) \\ F(\omega^1) \\ \vdots \\ F(\omega^{n-1}) \end{pmatrix} = A \begin{pmatrix} C_0 \\ C_1 \\ \vdots \\ C_{n-1} \end{pmatrix}$$

Prop. より

$$\frac{B}{n} \begin{pmatrix} F(\omega^0) \\ F(\omega^1) \\ \vdots \\ F(\omega^n) \end{pmatrix} = \begin{pmatrix} C_0 \\ C_1 \\ \vdots \\ C_{n-1} \end{pmatrix}$$

F の (ω^i) での値から、係数 C_i を求める.

Def. Discrete Fourier Transformation (DFT)

$$\mathcal{F}: \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} \longmapsto A \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix}$$

Inverse DFT.

$$\mathcal{F}^{-1}: \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-1} \end{pmatrix} \longmapsto \frac{B}{n} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-1} \end{pmatrix}$$

Propより

$\mathcal{F}^{-1} \circ \mathcal{F} = id$.

$f=F$, $g=G$. とする.

DFTを用いた $F(x)G(x)$ の計算法.

① $\mathcal{F}\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} f(w^0) \\ f(w^1) \\ \vdots \\ f(w^{n-1}) \end{pmatrix}$, $\mathcal{F}\begin{pmatrix} d_0 \\ d_1 \\ \vdots \\ d_{n-1} \end{pmatrix} = \begin{pmatrix} g(w^0) \\ g(w^1) \\ \vdots \\ g(w^{n-1}) \end{pmatrix}$ を計算.

② $\begin{pmatrix} f(w^0)g(w^0) \\ f(w^1)g(w^1) \\ \vdots \\ f(w^{n-1})g(w^{n-1}) \end{pmatrix}$ 成分毎の積を計算.

注. $f(x)g(x)$ が n 次になるように, $f(x)$

③ ②を $\mathcal{F}^{-1}$ で変換すれば.

$f(x)g(x)$ の x^i の係数を得る。

$h(x) = f(x)g(x)$. $\mathcal{F}^{-1}$ で. $(h(w^0), h(w^1), \cdots, h(w^{n-1}))$ より h の係数は求まる。

何故この方法は高速なのか?

係数のかけ算の回数に注目

② の計算には、かけ算は n 回でよい。

① DFTの計算で、かけ算の回数をつぜないか?

そぼくには、n^2 回

上下のうまいやり方で $n \log n$ 回くらいにできる。

$n=2^N$ とする。

$n=8$ で説明。　$f(\omega^0), f(\omega^1), \cdots, f(\omega^7)$ を計算したい。

$$f(x)=c_0+c_2x^2+c_4x^4+c_6x^6+c_1x+c_3x^3+c_5x^5+c_7x^7$$

2つにおける。

$$p(y)=c_0+c_2y+c_4y^2+c_4y^3$$
$$q(y)=c_1+c_3y+c_5y^2+c_7y^3$$

とおくと、

$$f(x)=p(x^2)+xq(x^2) \quad \text{となる。}$$

$$f(\omega^i)=p((\omega^2)^i)+\omega^i q((\omega^2)^i) \qquad \omega^8=1$$

$$f(\omega^0)=p(\omega^0)+\omega^0 q(\omega^0)$$
$$f(\omega^1)=p(\omega^2)+\omega^1 q(\omega^2)$$
$$f(\omega^2)=p(\omega^4)+\omega^2 q(\omega^4)$$
$$f(\omega^3)=p(\omega^6)+\omega^3 q(\omega^6)$$

$$f(\omega^4)=p(\omega^0)+\omega^4 q(\omega^0)$$
$$f(\omega^5)=p(\omega^2)+\omega^5 q(\omega^2)$$
$$f(\omega^6)=p(\omega^4)+\omega^6 q(\omega^4)$$
$$f(\omega^7)=p(\omega^6)+\omega^7 q(\omega^6)$$

同じ　　同じ

$\omega^8=1=\omega^0$

$\omega^{10}=\omega^2$

$\omega^{12}=\omega^4$

n 次多項式に ω^i を代入して値を計算する加算の回数を $T(n)$ とする。

$i=0,1,\cdots,n-1$

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$ なる漸化式が成立。

○○の計算、つまり、p, q の計算

w^2 q(w^2) の計算

これをみたす T(n) は？

$T(n) = n\log_2 n$ がみたす。

$T\left(\frac{n}{2}\right) = \frac{n}{2}\log_2\frac{n}{2} = \frac{n}{2}\log_2 n - \frac{n}{2}$ これを漸化式に代入。

$$2T\left(\frac{n}{2}\right) + n = 2 \times \frac{n}{2}\log_2 n - n + n$$

$$= n\log_2 n.$$

$\therefore\ T(n) = n\log_2 n.$

かけ算回数は、
$n + n\log_2 n$ 回。
これは n^2 より、はるかに小さい。

---

x を数とすると、ケタ上がりが生じる。 準備なし。