主題: Big numbers

- 基礎
- 應用
- 作業與自我挑戰

基礎

- What are big numbers ?
- Addition
- Multiplication
- Subtraction
- Division

What are big numbers ?

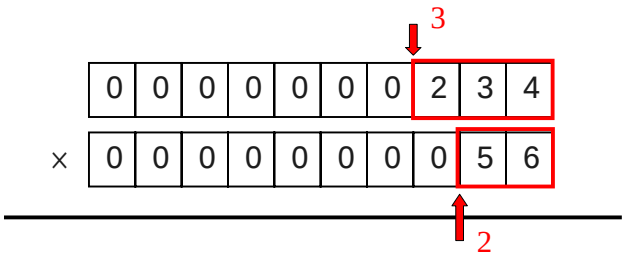
- 怎麼存下面的整數
  - 100000 : int ($-2^{31} \sim 2^{31}-1$) ($\approx 10^9$)
  - 10000000000 : __int64 ($-2^{63} \sim 2^{63}-1$) ($\approx 10^{19}$)
    - 在工作站上：long long
  - 100000000000000000000000 ($= 10^{23}$) : 怎麼存？
- 目前 C 只能用 64 bit 儲存整數，64 bit 存不下的整數就稱為 big numbers (大數)

How to store big numbers?

- 正確評估 max_len
  - 最大數要多少位元 (10 進位)，保證不會 overflow
  - 5 位數 + 4 位數 <= 6 位數
  - 5 位數 × 4 位數 <= 9 位數

How to store big numbers? (cont.)

- 用整數陣列來儲存 (最方便)　**684597315274**
  - int num[max_len] = { 4, 7, 2, 5, 1, 3, 7, 9, 5, 4, 8, 6, 0, 0, 0}
    - 要記得補 0　個位數　(max_len = 15)
  - 不補 0 要記有效位元長度，速度快但程式稍不好寫



Addition

- 方法
  - 模擬加法的動作
  - 注意進位的處理

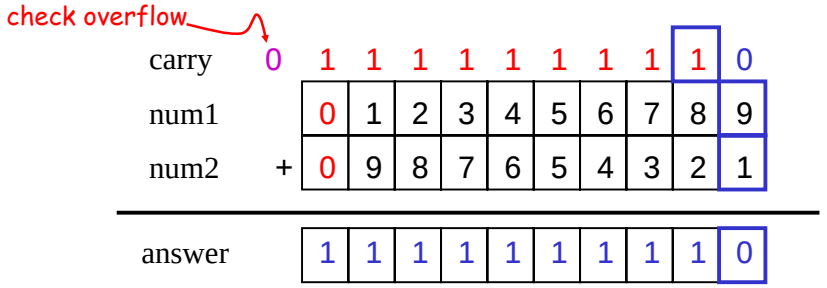
```
    0  1  1
    4  1  7  8
+      4  2  5
-----------------
    4  6  0  3
```


Variables

- int num1[max_len], num2[max_len]
  - 儲存運算數字
- int answer[max_len]
  - 儲存結果
- int carry
  - 儲存進位

Example

- 123456789 + 987654321 ？
  - 把 123456789 存在 num1 之中，987654321 存在 num2，結果存在 answer 中

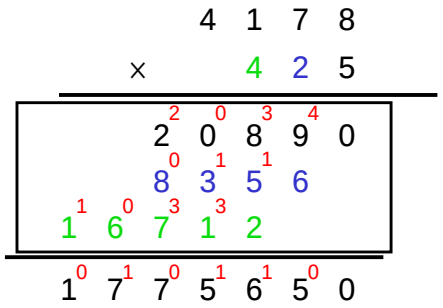


Pseudo code

```
void big_add( int *num1, int *num2, int *answer ) {
int i;
    carry = 0;
    for  ( i = 0 ; i < max_len ; i++) {
          answer[ i ] = (num1[ i ] + num2[ i ] + carry) % 10;
          carry = (num1[ i ] + num2[ i ] + carry) / 10;
    }
    if (carry != 0)  printf("overflow!!!\n");
}
```


Multiplication

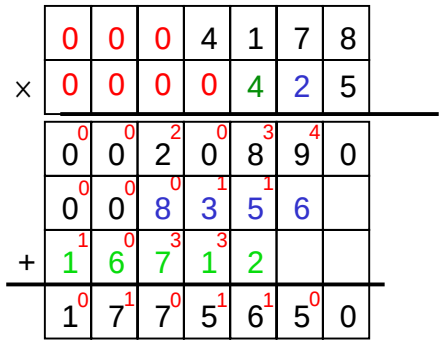
- 方法
  - 利用一位數乘法和加法完成
  - 注意移位

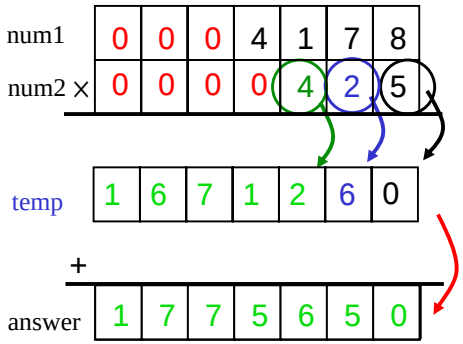


Variables

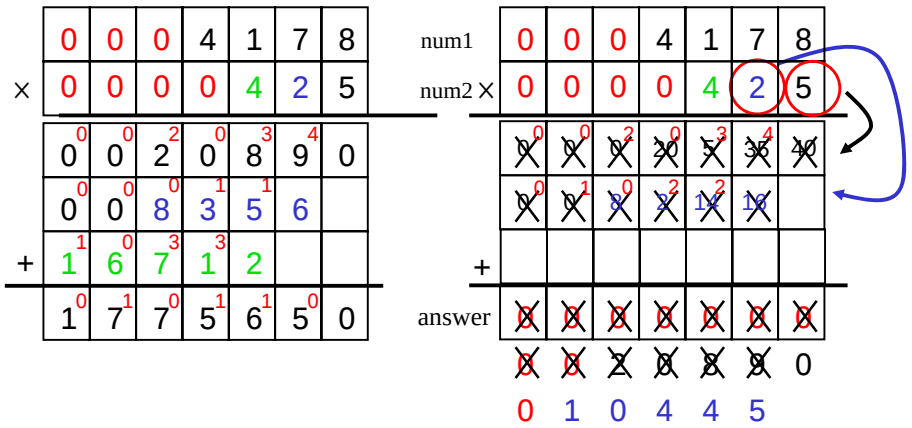
- num1, num2, answer, carry 和加法一樣
- 中間的過程是否要用二維陣列來儲存？
  - 不需要
  - 每次有新的值乘出來後，作適當的移位再相加

Example





Example



Pseudo code

```
void big_mul( int *num1, int *num2, int *answer ) {
int i, j, t;
    for (i = 0; i < max_len ; i++)  answer[i] = 0;

    for ( i = 0; i < max_len ; i++ ) {
        carry = 0;
        for ( j = 0; j < max_len ; j++ ) {
            if (( i + j ) < max_len) {  /* not overflow
                t = answer[ i + j ] + num2[ i ] * num1[ j ] + carry;
                answer[ i + j ] = t % 10;
                carry = t / 10;          }
        }
    }
}
```

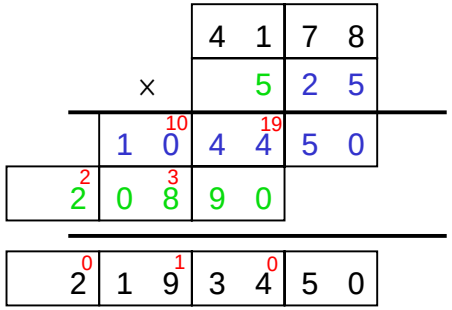
將一位數的乘法與加法合併在一個 loop 中完成

移位

/* 注意 overflow!

Speedup

- 可以將好幾個位數當成一個位數來運算
  - 之前的例子都是十進位，也可以換成百進位



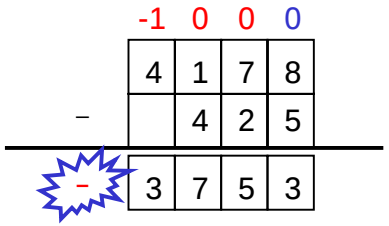
- 當每一次運算所使用的位數增加時，總共所需的計算次數就會減少，所要開的陣列大小也跟著變小
- 若陣列的型態為 32-bit int，最多能以多少進位來計算???

Subtraction

- Variables
  - int num1[maxlen+1], num2[maxlen+1];
  - 加一個 int 存 sign (+1, -1)
- 方法 (假設是兩個正數相減)
  - 模擬減法的動作
  - 注意借位
  - 先判斷減數或被減數大，拿大的去減小的，最後再補上正負號

Example

- 425 – 4178 = – (4178 – 425) = –3753



C code

```
/* 假設 num1 ≥ num2 而且兩個都是正數*/
void big_sub( int * num1, int *num2, int *answer ) {
  int i;
  borrow = 0;     // borrow = 0 or -1
  for (i = 0; i < max_len  ; i++) {
      answer[ i ] = num1[ i ] – num2 [ i ] + borrow;
      if (answer[ i ] < 0) {    // 需借位
              answer[ i ] = answer[ i ] + 10;   // 若是百進位則加 100
              borrow = -1;   // 借位
        }
      else      borrow = 0;
   }
   if (borrow != 0) printf("underflow!!!\n");      // 確定 num1 ≥ num2
}
```


Division

- Method 1: 寫程式模擬除法動作
  - 程式速度快
  - 程式較難寫
- Method 2: 不停的作減法
  - 程式速度最慢
  - 程式最好寫，只需要大數減法和大數比較
- Method 3: 利用 binary search
  - 需要大數乘法、大數加法、大數比較、和大數除以 2 的動作，就可以得到除法的結果

Time complexity???

應用

- 應用一: H.91.4 連分數
- 應用二: A.113 Power of Cryptography

應用一: H.91.4 連分數

http://www.cs.tku.edu.tw/info_race2002/codeq.htm

- $[a_0, a_1, \ldots, a_n]$，其中每一個 a_i 都是 0 到 9 之間的整數，而且 $n \le 50$ (最多有 51 個數字)
- 將連分數轉成分數

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_n}}}}$$

Example

- n = 3, $[a_0, a_1, a_2, a_3] = [4, 3, 1, 2]$

$\frac{1}{2}$ → (+1 後取倒數) → $\frac{2}{3}$ → (+3 後取倒數) → $\frac{3}{11}$ → (+4 後取倒數) → $\frac{11}{47}$

$$\cfrac{1}{4+\cfrac{1}{3+\cfrac{1}{1+\cfrac{1}{2}}}} = \cfrac{1}{4+\cfrac{1}{3+\cfrac{2}{3}}} = \cfrac{1}{4+\cfrac{3}{11}} = \frac{11}{47}$$

(a_0) (a_1) (a_2) (a_3)

Observation

$\frac{1}{2}$ → (+1 後取倒數) → $\frac{2}{3}$ → (+3 後取倒數) → $\frac{3}{11}$ → (+4 後取倒數) → $\frac{11}{47}$

$\frac{y}{x} = \frac{1}{a_3}$ → ($+a_2$ 後取倒數) → $\frac{y'}{x'}$ → ($+a_1$ 後取倒數) → $\frac{y''}{x''}$ → ($+a_0$ 後取倒數) → $\frac{y'''}{x'''}$

$\frac{y}{x}$ → (+z 後取倒數) → $\frac{new\_y}{new\_x}$

Solution

- 所需的動作
  - 乘法
  - 加法
  - 分子分母對調

$\frac{y}{x}$ + z 後取倒數 $\frac{new\_y}{new\_x}$

$$\frac{1}{z+\frac{y}{x}} = \frac{1}{\frac{x \times z + y}{x}} = \frac{x}{x \times z + y}$$

new_y

new_x

- 約分 ??? (題目並沒有要求約分 - 陷阱嗎 ???)
- Easy ???

Program structure

- **variables**
  - int a[51] // 儲存 a_0 到 a_n， $n \le 50$
  - int x[maxlen] // 分母
  - int y[maxlen] // 分子
  - int maxlen = ???
- **program structure**

```
initial_x_y();
for (i = n – 1;  i >= 0; i– –) {
      compute_new_x_y();
}
```


Pseudo code

$a_{i-1} + \frac{1}{a_i + \frac{y}{x}} \Rightarrow a_{i-1} + \frac{new\_y}{new\_x}$

z, new_z

```
int x[51], y[51];
int newx[51], newy[51];

x ← a_n; y ← 1;
for (i = n – 1; i >= 0; i– –) {          /* ( x , y , z ) => ( newx , new_y )
    newx ← big_add ( big_mul (x , a_i) , y) ;      /* newx = x × z + y
    newy ← x;
    x ← newx;
    y ← newy;
}
```

a_i

這裡的大數乘法是一個大數乘以一個整數，較簡單

應用二: A.113 Power of Cryptography

- 給兩個整數 n, p ，已知 k= $\sqrt[n]{p}$ 為整數，求 $\sqrt[n]{p}$ 為多少?
- $1 \le n \le 200$
- $1 \le p \le 10^{101}$
- $1 \le k \le 10^9$

Solution (I)

- 將題目轉換為 $k^n = p$，用 binary search 求 k
  - 以 n = 2, p = 6538249 為例 (開根號)
    - $1^2 < 6538249 < 10000^2$

      $1^2 < 6538249 < 5000^2$

      ⋮
  - 需要大數乘法 (因 $k \le 10^9$，(L + U) / 2 不是大數運算)
  - 計算 k^n 所需的大數乘法是一個大數乘以一個整數，較簡單
- Time complexity ???

Solution (II)

- Since $1 \le k \le 10^9$, we can also solve this problem as follows.

```
#include <math.h>
long n, k;
long double p, r;  (19 位有效數字)
scanf("%Lf", &p)
r = pow(p, 1.0/n);
k = "the integer closest to r"
```

- Correct or not ???
- How to make r an integer ???

作業與自我挑戰

- 作業
  - 練習題
    - A. 623 500!
      http://uva.onlinejudge.org/external/6/623.html
  - 挑戰題
    - A.10023 Square Root (binary search does not work!)
      http://uva.onlinejudge.org/external/100/10023.html
- 自我挑戰
  - A. 288 Arithmetic Operation with Large Integers
    http://uva.onlinejudge.org/external/2/288.html (with power)

- 其它有趣的題目:
  - A. 619 Numerically Speaking (base 26 to decimal)
    http://uva.onlinejudge.org/external/6/619.html
  - A. 495 Fibonacci Freeze
    - http://uva.onlinejudge.org/external/4/495.html
  - A. 338 Long Multiplication
    - http://uva.onlinejudge.org/external/3/338.html
  - 2004兩岸清華學生程式比賽 Problem D
    - http://venus.cs.nthu.edu.tw/~adv_prog/NTHU2004-Problems.doc