大整數運算

丁培毅

大整數運算

丁培毅

這個程式不算太難, 可是有很多地方需要做出適當的設計決策,

大整數運算

丁培毅

這個程式不算太難, 可是有很多地方需要做出適當的設計決策,
很容易讓你有想要重寫的衝動,

大整數運算

丁培毅

這個程式不算太難, 可是有很多地方需要做出適當的設計決策,
很容易讓你有想要重寫的衝動,
拋開競賽程式的束縛,

大整數運算

丁培毅

這個程式不算太難, 可是有很多地方需要做出適當的設計決策,
很容易讓你有想要重寫的衝動,
拋開競賽程式的束縛,
你的考量點不能只是通過測資就好 (AC),

大整數運算

丁培毅

這個程式不算太難, 可是有很多地方需要做出適當的設計決策,
很容易讓你有想要重寫的衝動,
拋開競賽程式的束縛,
你的考量點不能只是通過測資就好 (AC),
也不能只有 CPU 的效率 (TLE),

大整數運算

丁培毅

這個程式不算太難, 可是有很多地方需要做出適當的設計決策,
很容易讓你有想要重寫的衝動,
拋開競賽程式的束縛,
你的考量點不能只是通過測資就好 (AC),
也不能只有 CPU 的效率 (TLE),
你自己的效率呢?

大整數運算

丁培毅

這個程式不算太難, 可是有很多地方需要做出適當的設計決策,
很容易讓你有想要重寫的衝動,
拋開競賽程式的束縛,
你的考量點不能只是通過測資就好 (AC),
也不能只有 CPU 的效率 (TLE),
你自己的效率呢?
程式未來的維護性呢?

大整數運算

丁培毅

這個程式不算太難, 可是有很多地方需要做出適當的設計決策,

很容易讓你有想要重寫的衝動,

拋開競賽程式的束縛,

你的考量點不能只是通過測資就好 (AC),

也不能只有 CPU 的效率 (TLE),

你自己的效率呢?

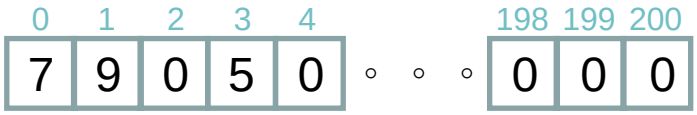
程式未來的維護性呢?

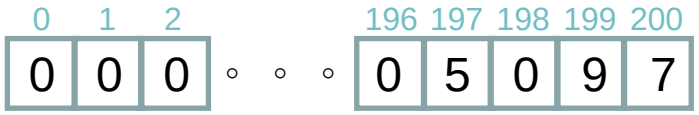
測試的時候也很需要加上自動的單元測試

資料表示方法

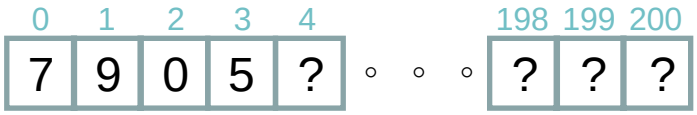
1. 固定大小字元陣列 char x[201];　固定 201 位的十進位數字

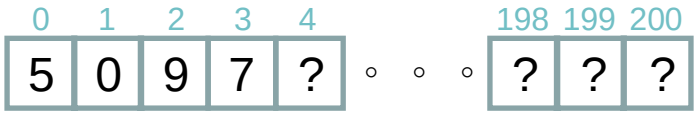
5097,個位數放在 x[0], 反著放　5097, 個位數放在 x[200]





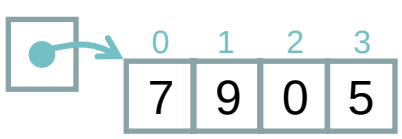
任意 len 位的十進位數字　len 4





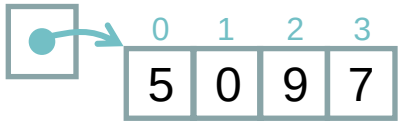
2. 動態配置大小的字元陣列 char *x;　任意 len 位的十進位數字

個位數放在 x[0], 反著放



len 4

最高位放在 x[0]



len 4

結構

加法

9 7 3 8

6 9 4

+

加法

	9	7	3	8
		6	9	4
進位	+		1	
				2

加法

	9	7	3	8
		6	9	4
進位	+	1	1	
			3	2

加法

	9	7	3	8
		6	9	4
進位	1	1	1	
+				
		4	3	2

加法

		9	7	3	8
			6	9	4
進位	+	1	1	1	1
		0	4	3	2

加法

a[]		9	7	3	8
b[]			6	9	4
進位	+ 1	1	1	1	
c[]	1	0	4	3	2

加法

a[]		9	7	3	8
b[]			6	9	4
進位	+ 1	1	1	1	
c[]	1	0	4	3	2

固定每個數字都有 200 位數

加法

a[]		9	7	3	8
b[]			6	9	4
進位	+ 1	1	1	1	
c[]	1	0	4	3	2

固定每個數字都有 200 位數

```
for (c[0]=i=0; i<201; i++) {


}
```

加法

a[]		9	7	3	8
b[]			6	9	4
進位	+ 1	1	1	1	
c[]	1	0	4	3	2

固定每個數字都有 200 位數

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] + b[i]) / 10;

}
```

加法

a[]		9	7	3	8
b[]			6	9	4
進位	+ 1	1	1	1	
c[]	1	0	4	3	2

固定每個數字都有 200 位數

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] + b[i]) / 10;
    c[i] += (a[i] + b[i]) % 10;
}
```

加法

a[]		9	7	3	8
b[]			6	9	4
進位	+ 1	1	1	1	
c[]	1	0	4	3	2

固定每個數字都有 200 位數

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] + b[i]) / 10;
    c[i] += (a[i] + b[i]) % 10;
}
```

省一點時間只加到兩者之間較大者的位數, 需要紀錄位數也要算出和的位數

加法

a[]		9	7	3	8
b[]			6	9	4
進位	+ 1	1	1	1	
c[]	1	0	4	3	2

固定每個數字都有 200 位數

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] + b[i]) / 10;
    c[i] += (a[i] + b[i]) % 10;
}
```

省一點時間只加到兩者之間較大者的位數, 需要紀錄位數也要算出和的位數

```
lenc = lena>lenb ? lena : lenb;
for (c[0]=i=0; i<lenc; i++) {
    c[i+1] = (a[i] + b[i]) / 10;
    c[i] += (a[i] + b[i]) % 10;
}
if (c[lenc]>0) lenc++;
```

加法

a[]		9	7	3	8
b[]			6	9	4
進位	+ 1	1	1	1	
c[]	1	0	4	3	2

省一點空間, 陣列的大小就是幾位數; 只能加到兩者之間較大者的位數(也省一點時間)

固定每個數字都有 200 位數

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] + b[i]) / 10;
    c[i] += (a[i] + b[i]) % 10;
}
```

省一點時間只加到兩者之間較大者的位數, 需要紀錄位數也要算出和的位數

```
lenc = lena>lenb ? lena : lenb;
for (c[0]=i=0; i<lenc; i++) {
    c[i+1] = (a[i] + b[i]) / 10;
    c[i] += (a[i] + b[i]) % 10;
}
if (c[lenc]>0) lenc++;
```

加法

a[]		9	7	3	8
b[]			6	9	4
進位	+ 1	1	1	1	
c[]	1	0	4	3	2

省一點空間, 陣列的大小就是幾位數; 只能加到兩者之間較大者的位數(也省一點時間)

輸出還不知道確切位數之前, 先用固定大小的陣列 char x[200];

固定每個數字都有 200 位數

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] + b[i]) / 10;
    c[i] += (a[i] + b[i]) % 10;
}
```

省一點時間只加到兩者之間較大者的位數, 需要紀錄位數也要算出和的位數

```
lenc = lena>lenb ? lena : lenb;
for (c[0]=i=0; i<lenc; i++) {
    c[i+1] = (a[i] + b[i]) / 10;
    c[i] += (a[i] + b[i]) % 10;
}
if (c[lenc]>0) lenc++;
```

加法

a[]		9	7	3	8
b[]			6	9	4
進位	+ 1	1	1	1	
c[]	1	0	4	3	2

省一點空間, 陣列的大小就是幾位數; 只能加到兩者之間較大者的位數(也省一點時間)

輸出還不知道確切位數之前, 先用固定大小的陣列 char x[200];

假設 b[] 比較小, lenb ≤ lena

固定每個數字都有 200 位數

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] + b[i]) / 10;
    c[i] += (a[i] + b[i]) % 10;
}
```

省一點時間只加到兩者之間較大者的位數, 需要紀錄位數也要算出和的位數

```
lenc = lena>lenb ? lena : lenb;
for (c[0]=i=0; i<lenc; i++) {
    c[i+1] = (a[i] + b[i]) / 10;
    c[i] += (a[i] + b[i]) % 10;
}
if (c[lenc]>0) lenc++;
```

加法

a[]		9	7	3	8
b[]			6	9	4
進位	+ 1	1	1	1	
c[]	1	0	4	3	2

省一點空間, 陣列的大小就是幾位數; 只能加到兩者之間較大者的位數(也省一點時間)

輸出還不知道確切位數之前, 先用固定大小的陣列 char x[200];

假設 b[] 比較小, lenb ≤ lena

```
lenx = lena;
memcpy(x, b, lenb);
memset(&x[lenb], 0, lenx-lenb+1);
```

固定每個數字都有 200 位數

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] + b[i]) / 10;
    c[i] += (a[i] + b[i]) % 10;
}
```

省一點時間只加到兩者之間較大者的位數, 需要紀錄位數也要算出和的位數

```
lenc = lena>lenb ? lena : lenb;
for (c[0]=i=0; i<lenc; i++) {
    c[i+1] = (a[i] + b[i]) / 10;
    c[i] += (a[i] + b[i]) % 10;
}
if (c[lenc]>0) lenc++;
```

加法

a[]		9	7	3	8
b[]			6	9	4
進位	+ 1	1	1	1	
c[]	1	0	4	3	2

省一點空間, 陣列的大小就是幾位數; 只能加到兩者之間較大者的位數(也省一點時間)

輸出還不知道確切位數之前, 先用固定大小的陣列 char x[200];

假設 b[] 比較小, lenb ≤ lena

```
lenx = lena;
memcpy(x, b, lenb);
memset(&x[lenb], 0, lenx-lenb+1);
```

固定每個數字都有 200 位數

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] + b[i]) / 10;
    c[i] += (a[i] + b[i]) % 10;
}
```

省一點時間只加到兩者之間較大者的位數, 需要紀錄位數也要算出和的位數

```
lenc = lena>lenb ? lena : lenb;
for (c[0]=i=0; i<lenc; i++) {
    c[i+1] = (a[i] + b[i]) / 10;
    c[i] += (a[i] + b[i]) % 10;
}
if (c[lenc]>0) lenc++;
```

```
for (i=0; i<lenx; i++) {
    x[i+1] += (a[i] + x[i]) / 10;
    x[i] = (a[i] + x[i]) % 10;
}
if (x[lenx]>0) lenx++;
```

配置陣列 c, 由 x 陣列拷貝資料

- 用前面的加法實作乘法時, 會覺得有點浪費 CPU 時間

- 用前面的加法實作乘法時, 會覺得有點浪費 CPU 時間

$$\begin{array}{rrrrr} & 9 & 7 & 3 & 8 \\ \times & & 6 & 9 & 4 \\ \hline \end{array}$$

- 用前面的加法實作乘法時, 會覺得有點浪費 CPU 時間

$$\begin{array}{r} 9\ 7\ 3\ 8 \\ \times\quad 6\ 9\ 4 \\ \hline 3\ 8\ 9\ 5\ 2 \end{array}$$

- 用前面的加法實作乘法時, 會覺得有點浪費 CPU 時間

$$
\begin{array}{rrrrrr}
 & & & 9 & 7 & 3 & 8 \\
\times & & & & 6 & 9 & 4 \\
\hline
 & & 3 & 8 & 9 & 5 & 2 \\
 & 8 & 7 & 6 & 4 & 2 & \\
\end{array}
$$

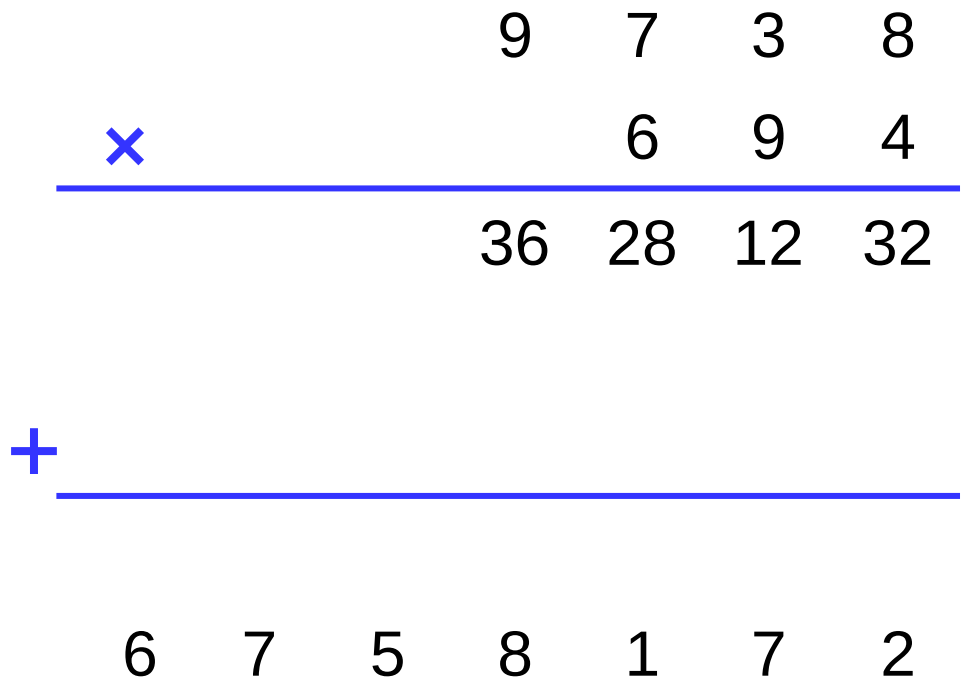
- 用前面的加法實作乘法時, 會覺得有點浪費 CPU 時間

$$
\begin{array}{rrrrrrr}
 & & & 9 & 7 & 3 & 8 \\
\times & & & & 6 & 9 & 4 \\
\hline
 & & 3 & 8 & 9 & 5 & 2 \\
 & 8 & 7 & 6 & 4 & 2 & \\
5 & 8 & 4 & 2 & 8 & &
\end{array}
$$

- 用前面的加法實作乘法時, 會覺得有點浪費 CPU 時間

$$
\begin{array}{rrrrrrr}
 & & & 9 & 7 & 3 & 8 \\
\times & & & & 6 & 9 & 4 \\
\hline
 & & 3 & 8 & 9 & 5 & 2 \\
 & 8 & 7 & 6 & 4 & 2 & \\
+ & 5 & 8 & 4 & 2 & 8 & \\
\hline
 & 6 & 7 & 5 & 8 & 1 & 7 & 2
\end{array}
$$

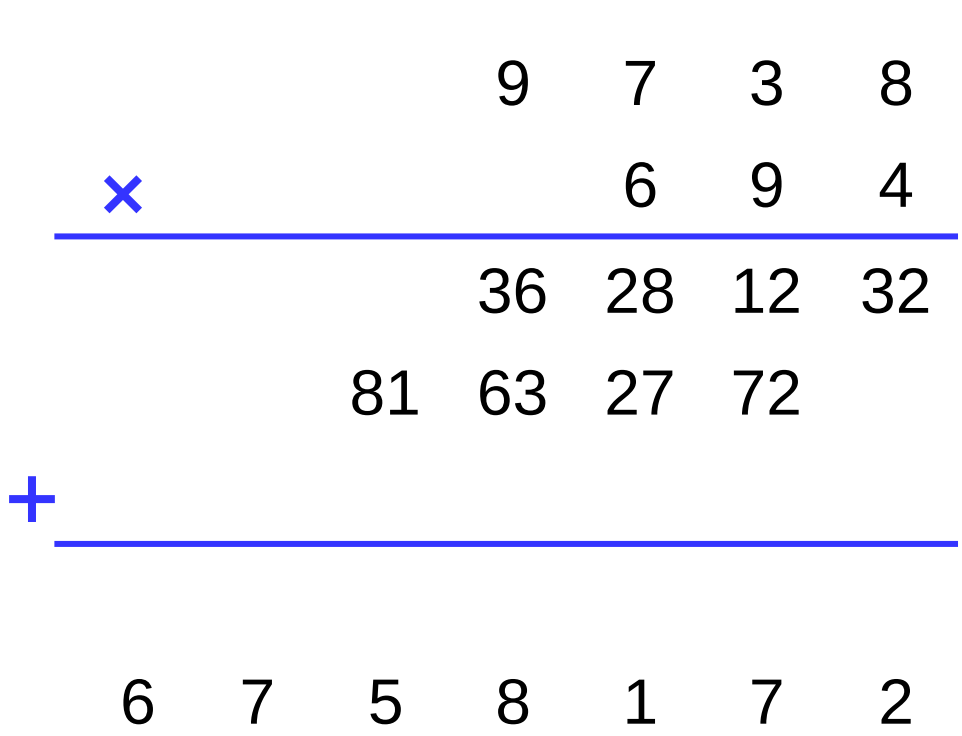
- 用前面的加法實作乘法時, 會覺得有點浪費 CPU 時間



中間不需要是合法的十進位

進位的動作不需要重複做

- 用前面的加法實作乘法時, 會覺得有點浪費 CPU 時間



中間不需要是合法的十進位

進位的動作不需要重複做

- 用前面的加法實作乘法時, 會覺得有點浪費 CPU 時間

				9	7	3	8
×					6	9	4
				36	28	12	32
			81	63	27	72	
+		54	42	18	48		
	6	7	5	8	1	7	2

中間不需要是合法的十進位

進位的動作不需要重複做

- 用前面的加法實作乘法時, 會覺得有點浪費 CPU 時間

				9	7	3	8
×					6	9	4
				36	28	12	32
			81	63	27	72	
+		54	42	18	48		
x[]		54	123	117	103	84	32
	6	7	5	8	1	7	2

中間不需要是合法的十進位

進位的動作不需要重複做

假設都已經加總在陣列 x 中

- 用前面的加法實作乘法時, 會覺得有點浪費 CPU 時間

				9	7	3	8
×					6	9	4
				36	28	12	32
			81	63	27	72	
+		54	42	18	48		
x[]		54	123	117	103	84	32
	6	7	5	8	1	7	2

中間不需要是合法的十進位

進位的動作不需要重複做

假設都已經加總在陣列 x 中

進位的數字有可能超過 9

- 用前面的加法實作乘法時, 會覺得有點浪費 CPU 時間

				9	7	3	8
×					6	9	4
				36	28	12	32
			81	63	27	72	
+		54	42	18	48		
x[]		54	123	117	103	84	32
	6	7	5	8	1	7	2

中間不需要是合法的十進位

進位的動作不需要重複做

假設都已經加總在陣列 x 中

進位的數字有可能超過 9

- 這個想法很實際, 程式也很容易製作, 刪除重複的進位動作, 也刪除重複的記憶體配置動作

- 用前面的加法實作乘法時, 會覺得有點浪費 CPU 時間

				9	7	3	8
×					6	9	4
				36	28	12	32
			81	63	27	72	
+		54	42	18	48		
x[]		54	123	117	103	84	32
	6	7	5	8	1	7	2

中間不需要是合法的十進位

進位的動作不需要重複做

假設都已經加總在陣列 x 中

進位的數字有可能超過 9

- 這個想法很實際, 程式也很容易製作, 刪除重複的進位動作, 也刪除重複的記憶體配置動作
- 有誰像這樣另外寫一個適合實作乘法的加法函式?

- 用前面的加法實作乘法時, 會覺得有點浪費 CPU 時間

				9	7	3	8
×					6	9	4
				36	28	12	32
			81	63	27	72	
+		54	42	18	48		
x[]		54	123	117	103	84	32
	6	7	5	8	1	7	2

中間不需要是合法的十進位

進位的動作不需要重複做

假設都已經加總在陣列 x 中

進位的數字有可能超過 9

- 這個想法很實際, 程式也很容易製作, 刪除重複的進位動作, 也刪除重複的記憶體配置動作
- 有誰像這樣另外寫一個適合實作乘法的加法函式?
- 管他的, 在乘法裡面用迴圈直接加就好了

- 用前面的加法實作乘法時, 會覺得有點浪費 CPU 時間

			9	7	3	8	
×				6	9	4	
			36	28	12	32	
		81	63	27	72		
+	54	42	18	48			
x[]	54	123	117	103	84	32	
	6	7	5	8	1	7	2

中間不需要是合法的十進位

進位的動作不需要重複做

假設都已經加總在陣列 x 中

進位的數字有可能超過 9

進位

```
for (i=0; i<lenx; i++) {
    x[i+1] += x[i] / 10;
    x[i] %= 10;
}
while (x[lenx]>0) {
    x[i+1] = x[i] / 10;
    x[i] %= 10;
    lenx++;
}
```

- 這個想法很實際, 程式也很容易製作, 刪除重複的進位動作, 也刪除重複的記憶體配置動作
- 有誰像這樣另外寫一個適合實作乘法的加法函式?
- 管他的, 在乘法裡面用迴圈直接加就好了

- 用前面的加法實作乘法時, 會覺得有點浪費 CPU 時間

			9	7	3	8	
×				6	9	4	
			36	28	12	32	
		81	63	27	72		
+	54	42	18	48			
x[]	54	123	117	103	84	32	
	6	7	5	8	1	7	2

中間不需要是合法的十進位

進位的動作不需要重複做

假設都已經加總在陣列 x 中

進位的數字有可能超過 9

進位

```
for (i=0; i<lenx; i++) {
    x[i+1] += x[i] / 10;
    x[i] %= 10;
}
while (x[lenx]>0) {
    x[i+1] = x[i] / 10;
    x[i] %= 10;
    lenx++;
}
```

- 這個想法很實際, 程式也很容易製作, 刪除重複的進位動作, 也刪除重複的記憶體配置動作
- 有誰像這樣另外寫一個適合實作乘法的加法函式?
- 管他的, 在乘法裡面用迴圈直接加就好了
  - **好?**

- 用前面的加法實作乘法時, 會覺得有點浪費 CPU 時間

				9	7	3	8
×					6	9	4
				36	28	12	32
			81	63	27	72	
+		54	42	18	48		
x[]		54	123	117	103	84	32
	6	7	5	8	1	7	2

中間不需要是合法的十進位

進位的動作不需要重複做

假設都已經加總在陣列 x 中

進位的數字有可能超過 9

進位

```
for (i=0; i<lenx; i++) {
    x[i+1] += x[i] / 10;
    x[i] %= 10;
}
while (x[lenx]>0) {
    x[i+1] = x[i] / 10;
    x[i] %= 10;
    lenx++;
}
```

- 這個想法很實際, 程式也很容易製作, 刪除重複的進位動作, 也刪除重複的記憶體配置動作
- 有誰像這樣另外寫一個適合實作乘法的加法函式?
- 管他的, 在乘法裡面用迴圈直接加就好了
  - 好? 不好!?

- 你有沒有在**加法**和**乘法**裡面看到很相似的程式碼?

- 你有沒有在**加法**和**乘法**裡面看到很相似的程式碼?
- 你有沒有運氣很不好地遇見一些測資在加法的程式錯掉, 在乘法裡面的「加」和「進位」裡也錯掉?

- 你有沒有在**加法**和**乘法**裡面看到很相似的程式碼?
- 你有沒有運氣很不好地遇見一些測資在加法的程式錯掉, 在乘法裡面的「加」和「進位」裡也錯掉?
- 為什麼不盡量重複使用? 如果用同一個加法函式就不會錯兩次了

- 你有沒有在**加法**和**乘法**裡面看到很相似的程式碼?
- 你有沒有運氣很不好地遇見一些測資在加法的程式錯掉, 在乘法裡面的「加」和「進位」裡也錯掉?
- 為什麼不盡量重複使用? 如果用同一個加法函式就不會錯兩次了
- 不完美?

- 你有沒有在**加法**和**乘法**裡面看到很相似的程式碼?
- 你有沒有運氣很不好地遇見一些測資在加法的程式錯掉, 在乘法裡面的「加」和「進位」裡也錯掉?
- 為什麼不盡量重複使用? 如果用同一個加法函式就不會錯兩次了
- 不完美?
- 不會吧, CPU 一秒鐘做 10 億 (10^9) 個指令, 不要只是從 CPU 的角度看事情, 不一定要那麼低階!!

- 你有沒有在**加法**和**乘法**裡面看到很相似的程式碼?
- 你有沒有運氣很不好地遇見一些測資在加法的程式錯掉, 在乘法裡面的「加」和「進位」裡也錯掉?
- 為什麼不盡量重複使用? 如果用同一個加法函式就不會錯兩次了
- 不完美?
- 不會吧, CPU 一秒鐘做 10 億 (10^9) 個指令, 不要只是從 CPU 的角度看事情, 不一定要那麼低階!!
- 不同的應用有不同的要求

- 你有沒有在**加法**和**乘法**裡面看到很相似的程式碼?
- 你有沒有運氣很不好地遇見一些測資在加法的程式錯掉, 在乘法裡面的「加」和「進位」裡也錯掉?
- 為什麼不盡量重複使用? 如果用同一個加法函式就不會錯兩次了
- 不完美?
- 不會吧, CPU 一秒鐘做 10 億 (10^9) 個指令, 不要只是從 CPU 的角度看事情, 不一定要那麼低階!!
- 不同的應用有不同的要求
- 工程的應用裡你還需要把程式開發的速度, 程式除錯的速度, 程式的可維護性一起考量進來

比較大小

- 這是標準的多欄位資料比對 (multi-field comparison)

比較大小

- 這是標準的多欄位資料比對 (multi-field comparison)
- 在相同位數時是字典順序 (lexicographic order) 的比較

比較大小

- 這是標準的多欄位資料比對 (multi-field comparison)
- 在相同位數時是字典順序 (lexicographic order) 的比較

1. 比對兩個數字的位數, 比較多位的數值較大

比較大小

- 這是標準的多欄位資料比對 (multi-field comparison)
- 在相同位數時是字典順序 (lexicographic order) 的比較

1. 比對兩個數字的位數, 比較多位的數值較大
2. 位數相等時, 由最高位開始逐位比對

比較大小

- 這是標準的多欄位資料比對 (multi-field comparison)
- 在相同位數時是字典順序 (lexicographic order) 的比較

1. 比對兩個數字的位數, 比較多位的數值較大

2. 位數相等時, 由最高位開始逐位比對

2.1 某一位數不等時, 該位數較大的就是數值較大的

比較大小

- 這是標準的多欄位資料比對 (multi-field comparison)
- 在相同位數時是字典順序 (lexicographic order) 的比較

1. 比對兩個數字的位數, 比較多位的數值較大

2. 位數相等時, 由最高位開始逐位比對

2.1 某一位數不等時, 該位數較大的就是數值較大的

2.2 某一位數相等時, 比較下一位數

比較大小

- 這是標準的多欄位資料比對 (multi-field comparison)
- 在相同位數時是字典順序 (lexicographic order) 的比較

1. 比對兩個數字的位數, 比較多位的數值較大
2. 位數相等時, 由最高位開始逐位比對
    2.1 某一位數不等時, 該位數較大的就是數值較大的
    2.2 某一位數相等時, 比較下一位數
3. 每一位數都相等時, 代表兩個數字相等

減法

9　3　3　2

8　6　9　4

減法

	9	3	3	2
	8	6	9	4
借位	−		1	
				8

減法

	9	3	3	2
	8	6	9	4
借位	−	1	1	
			3	8

減法

		9	3	3	2
		8	6	9	4
借位	−	1	1	1	
			6	3	8

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

減法

固定每個數字都有 200 位數

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

減法

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {


}
```

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;

}
```

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

程式碼和加法很像, 要求達成!

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

程式碼和加法很像, 要求達成!

是這樣嗎??!! 很像耶!!

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

程式碼和加法很像, 要求達成!

是這樣嗎??!! 很像耶!!

CPU 都只有加法電路沒有減法電路喔!!

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

程式碼和加法很像, 要求達成!

是這樣嗎??!! 很像耶!!

CPU 都只有加法電路沒有減法電路喔!!

有沒有同學減法直接呼叫加法函式的?

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

程式碼和加法很像, 要求達成!

是這樣嗎??!! 很像耶!!

CPU 都只有加法電路沒有減法電路喔!!

有沒有同學減法直接呼叫加法函式的?

9 3 3 2

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

程式碼和加法很像, 要求達成!

是這樣嗎??!! 很像耶!!

CPU 都只有加法電路沒有減法電路喔!!

有沒有同學減法直接呼叫加法函式的?

	9	3	3	2
−	8	6	9	4

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

程式碼和加法很像, 要求達成!

是這樣嗎??!! 很像耶!!

CPU 都只有加法電路沒有減法電路喔!!

有沒有同學減法直接呼叫加法函式的?

	9	3	3	2
10's 補數	1	3	0	5
				1

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

程式碼和加法很像, 要求達成!

是這樣嗎??!! 很像耶!!

CPU 都只有加法電路沒有減法電路喔!!

有沒有同學減法直接呼叫加法函式的?

		9	3	3	2
10's 補數		1	3	0	5
	+				1

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

程式碼和加法很像, 要求達成!

是這樣嗎??!! 很像耶!!

CPU 都只有加法電路沒有減法電路喔!!

有沒有同學減法直接呼叫加法函式的?

		9	3	3	2
10's 補數		1	3	0	5
進位	+			0	1
					8

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

程式碼和加法很像, 要求達成!

是這樣嗎??!! 很像耶!!

CPU 都只有加法電路沒有減法電路喔!!

有沒有同學減法直接呼叫加法函式的?

		9	3	3	2
10's 補數		1	3	0	5
進位	+		0	0	1
				3	8

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

程式碼和加法很像, 要求達成!

是這樣嗎??!! 很像耶!!

CPU 都只有加法電路沒有減法電路喔!!

有沒有同學減法直接呼叫加法函式的?

		9	3	3	2
10's 補數		1	3	0	5
進位	+	0	0	0	1
			6	3	8

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

程式碼和加法很像, 要求達成!

是這樣嗎??!! 很像耶!!

CPU 都只有加法電路沒有減法電路喔!!

有沒有同學減法直接呼叫加法函式的?

		9	3	3	2
10's 補數		1	3	0	5
進位	+ 1	0	0	0	1
		0	6	3	8

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

程式碼和加法很像, 要求達成!

是這樣嗎??!! 很像耶!!

CPU 都只有加法電路沒有減法電路喔!!

有沒有同學減法直接呼叫加法函式的?

		9	3	3	2
10's 補數		1	3	0	5
進位	+ 1	0	0	0	1
		0	6	3	8

好吧, 有點麻煩, 最高位要忽略!

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

程式碼和加法很像, 要求達成!

是這樣嗎??!! 很像耶!!

CPU 都只有加法電路沒有減法電路喔!!

有沒有同學減法直接呼叫加法函式的?

		9	3	3	2
10's 補數		1	3	0	5
進位	+ 1	0	0	0	1
		0	6	3	8

好吧, 有點麻煩, 最高位要忽略!

算了, 重寫, 不用剛才的加法了!!

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

程式碼和加法很像, 要求達成!

是這樣嗎??!! 很像耶!!

CPU 都只有加法電路沒有減法電路喔!!

有沒有同學減法直接呼叫加法函式的?

		9	3	3	2
10's 補數		1	3	0	5
進位	+ 1	0	0	0	1
		0	6	3	8

好吧, 有點麻煩, 最高位要忽略!

算了, 重寫, 不用剛才的加法了!!

這不是自相矛盾嗎????????

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

程式碼和加法很像, 要求達成!

是這樣嗎??!! 很像耶!!

CPU 都只有加法電路沒有減法電路喔!!

有沒有同學減法直接呼叫加法函式的?

		9	3	3	2
10's 補數		1	3	0	5
進位	+ 1	0	0	0	1
		0	6	3	8

好吧, 有點麻煩, 最高位要忽略!

算了, 重寫, 不用剛才的加法了!!

這不是自相矛盾嗎????????

這是設計時期的決策

減法

a[]		9	3	3	2
b[]		8	6	9	4
借位	−	1	1	1	
c[]		0	6	3	8

固定每個數字都有 200 位數

假設 a[] 比 b[] 大 (需要比較)

```
for (c[0]=i=0; i<201; i++) {
    c[i+1] = (a[i] - b[i]) / 10;
    c[i] += (a[i] - b[i]) % 10;
}
```

程式碼和加法很像, 要求達成!

是這樣嗎??!! 很像耶!!

CPU 都只有加法電路沒有減法電路喔!!

有沒有同學減法直接呼叫加法函式的?

		9	3	3	2
10's 補數		1	3	0	5
進位	+ 1	0	0	0	1
		0	6	3	8

好吧, 有點麻煩, 最高位要忽略!

算了, 重寫, 不用剛才的加法了!!

這不是自相矛盾嗎????????

這是設計時期的決策

安啦, 你有考慮過就好

乗法

		9	7	3	8
×			6	9	4

乗法

	9	7	3	8
×		6	9	4
	36	28	12	32

乗法

		9	7	3	8
×			6	9	4
		36	28	12	32
	81	63	27	72	

乗法

			9	7	3	8
×				6	9	4
			36	28	12	32
		81	63	27	72	
+	54	42	18	48		

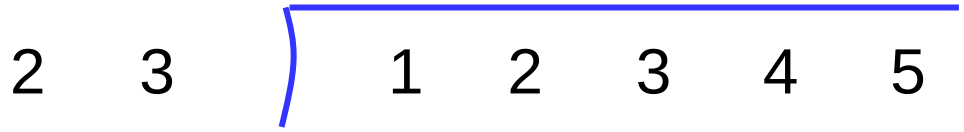
乗法

			9	7	3	8
×				6	9	4
			36	28	12	32
		81	63	27	72	
+	54	42	18	48		
x[]	54	123	117	103	84	32

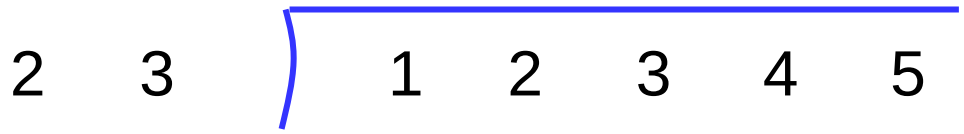
乗法

				9	7	3	8
×					6	9	4
				36	28	12	32
			81	63	27	72	
+		54	42	18	48		
x[]		54	123	117	103	84	32
	6	7	5	8	1	7	2

除法

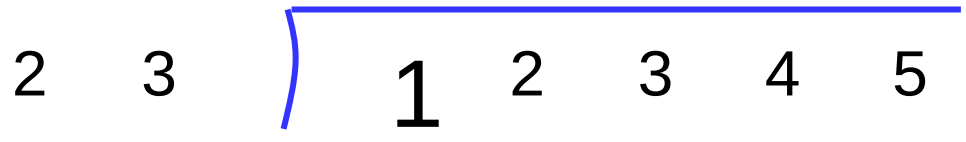


除法



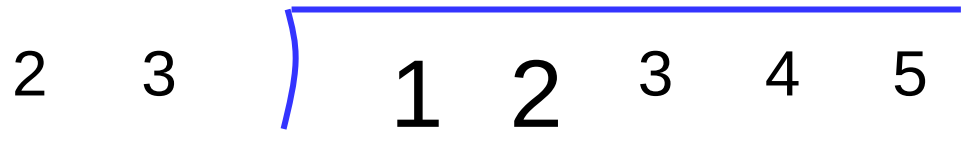
長除法

除法



長除法

除法

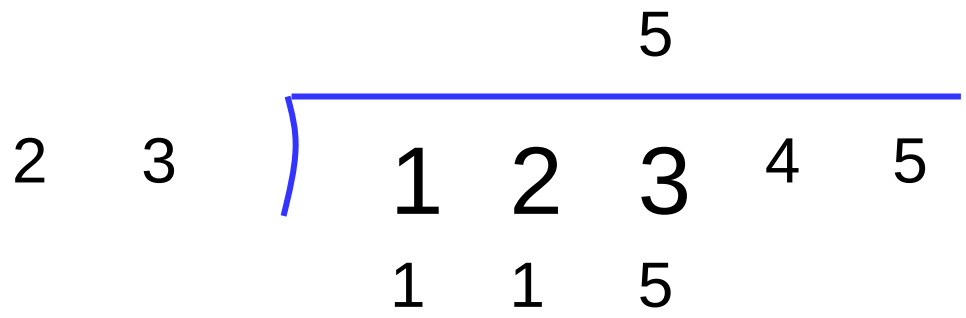


長除法

除法

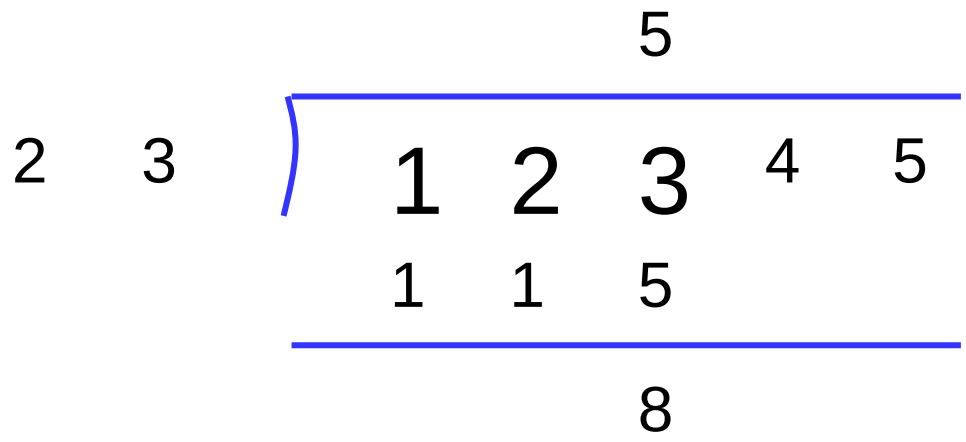
長除法

除法



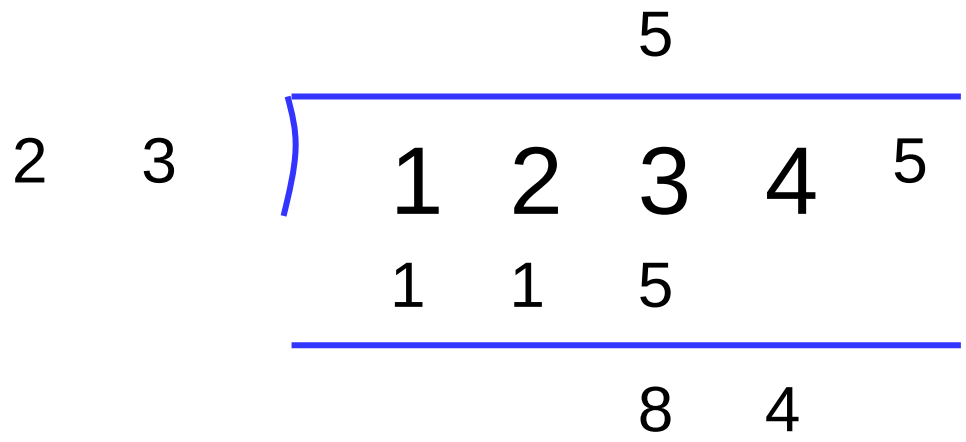
長除法

除法



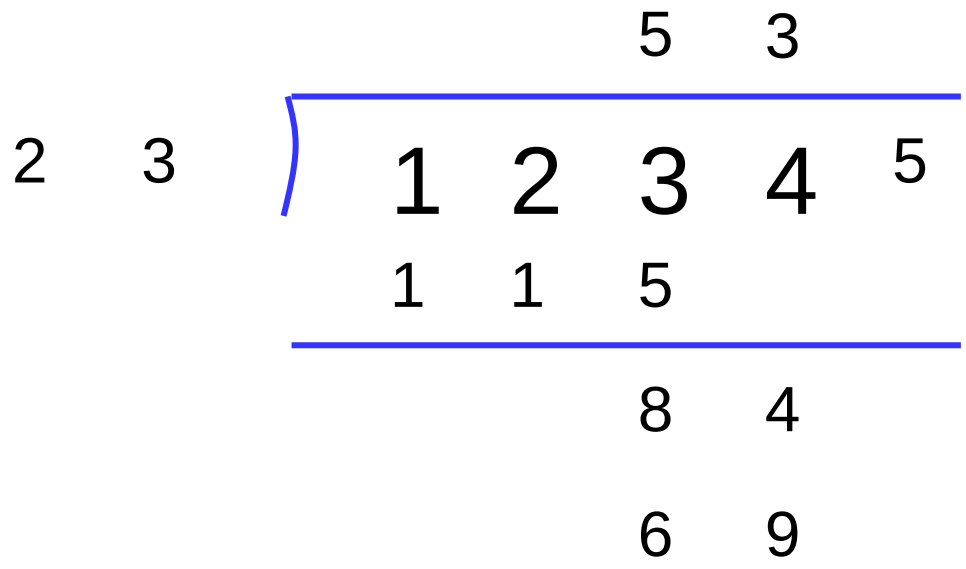
長除法

除法



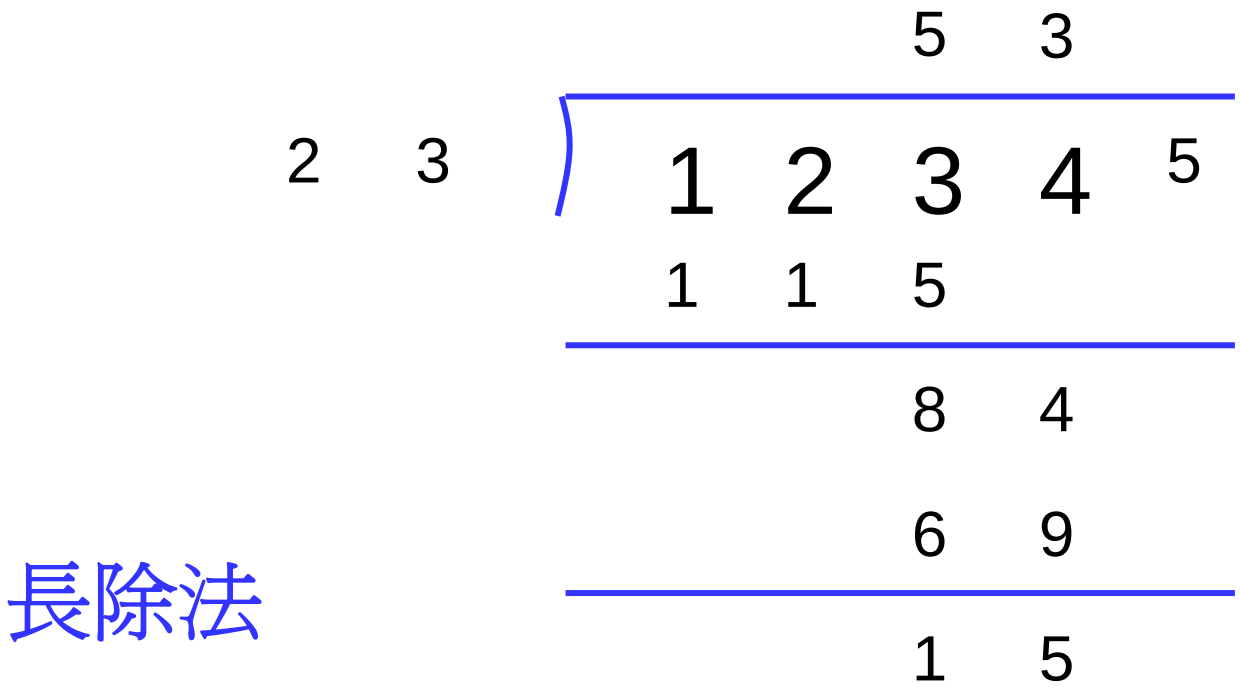
長除法

除法



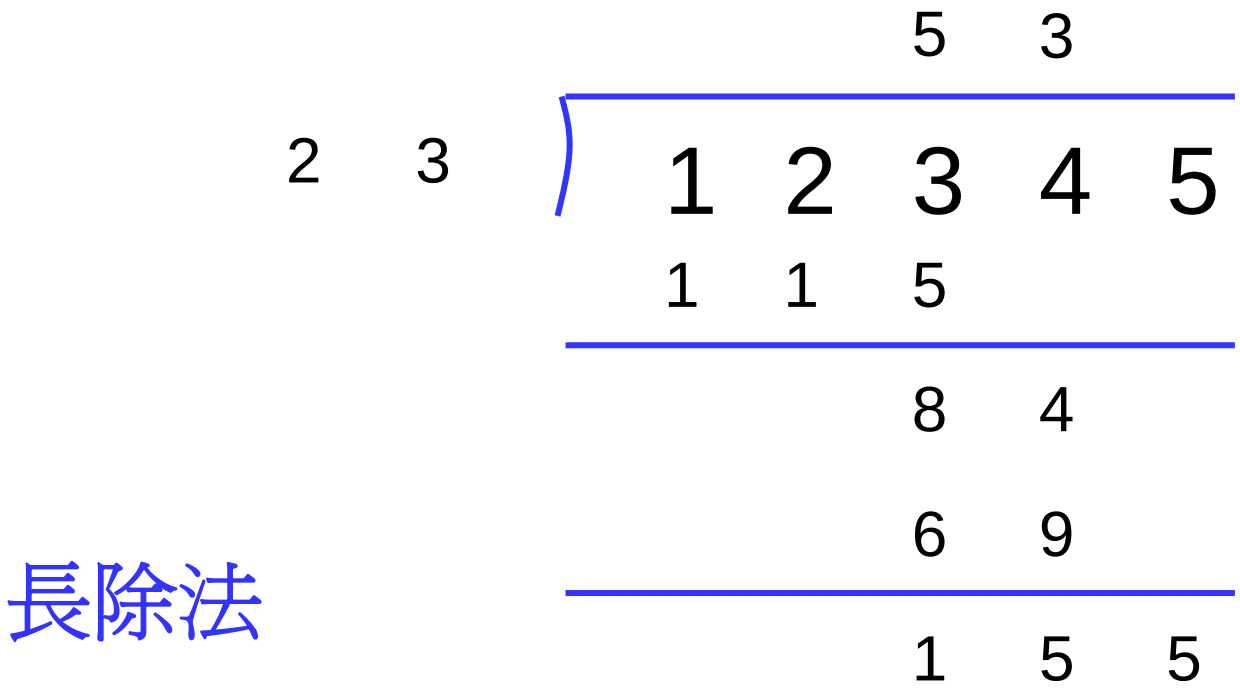
長除法

除法



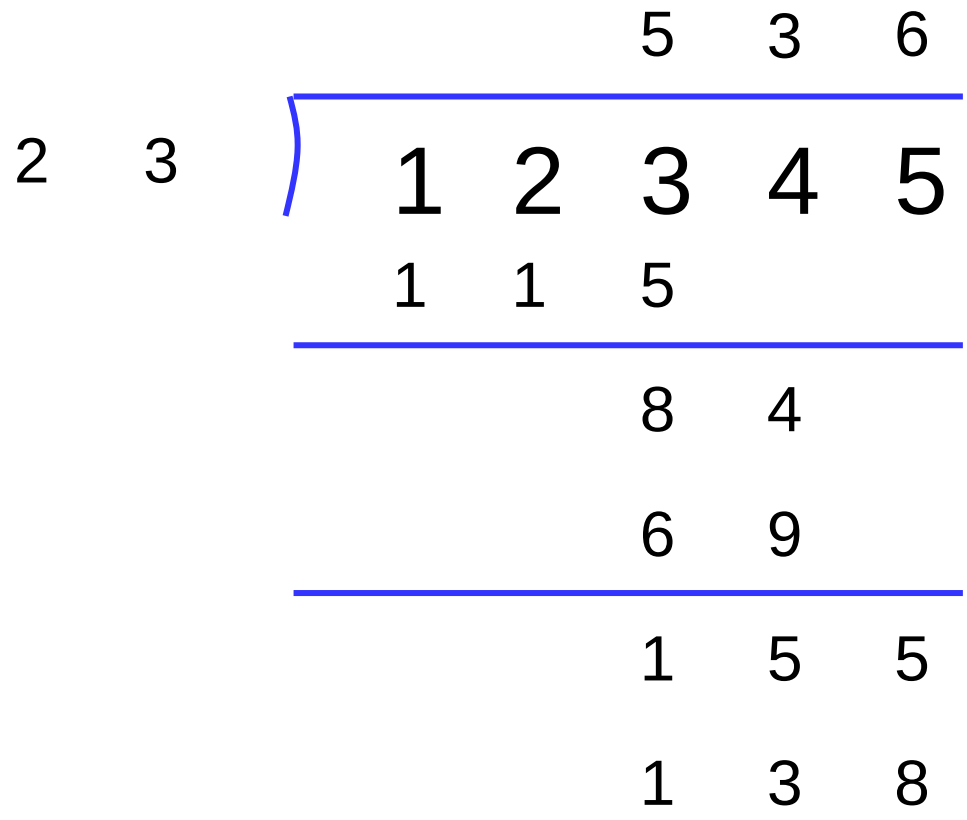
長除法

除法



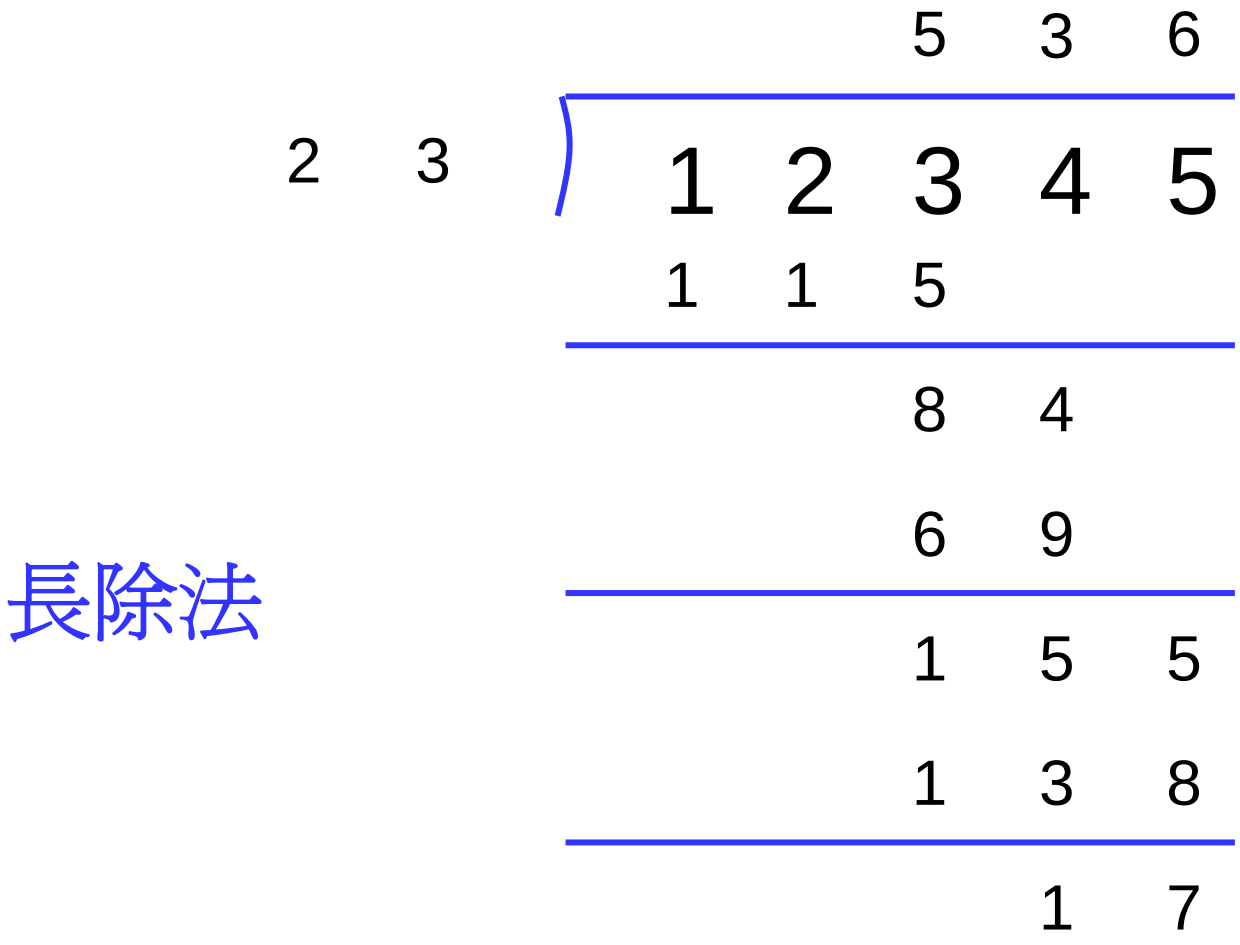
長除法

除法



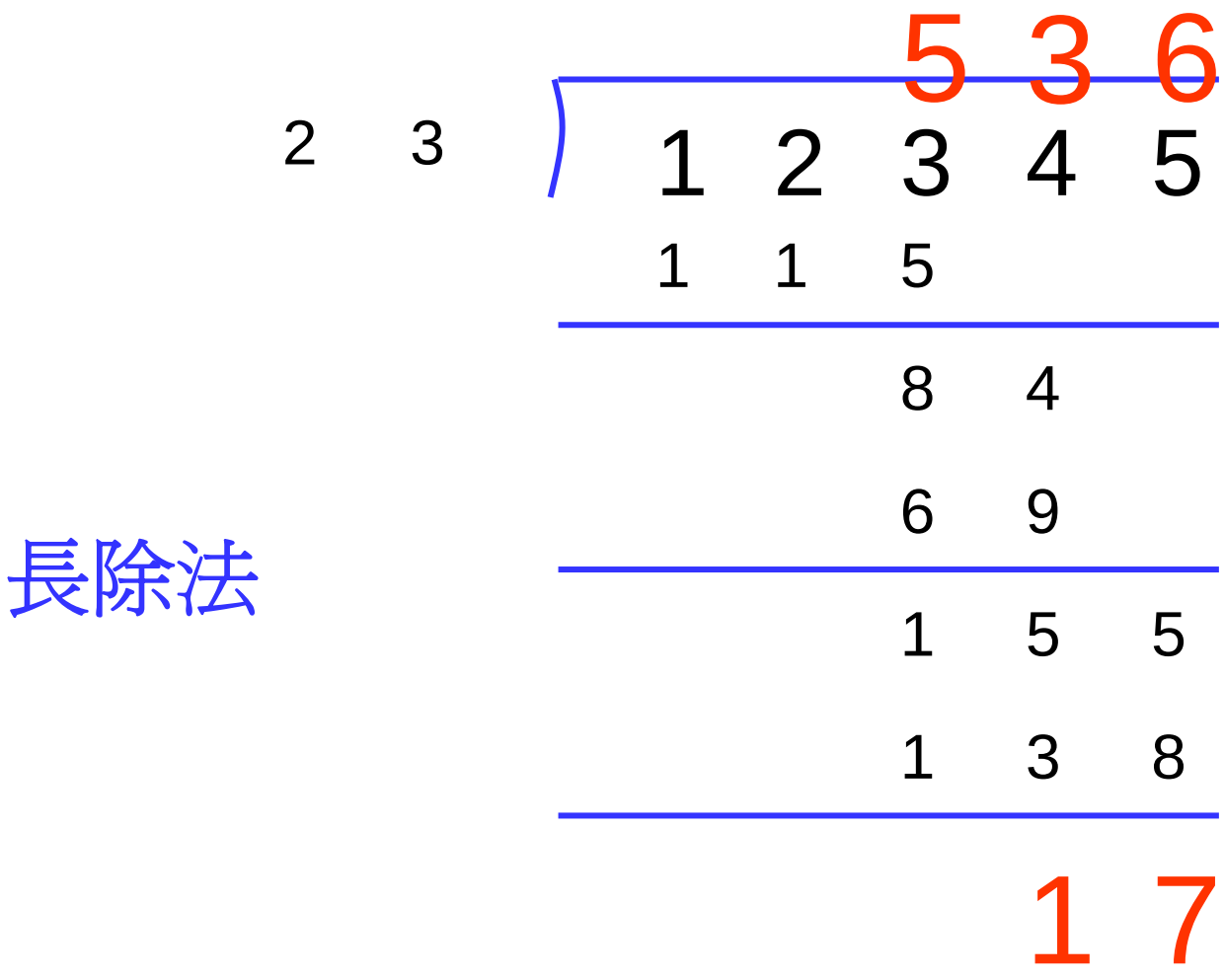
長除法

除法



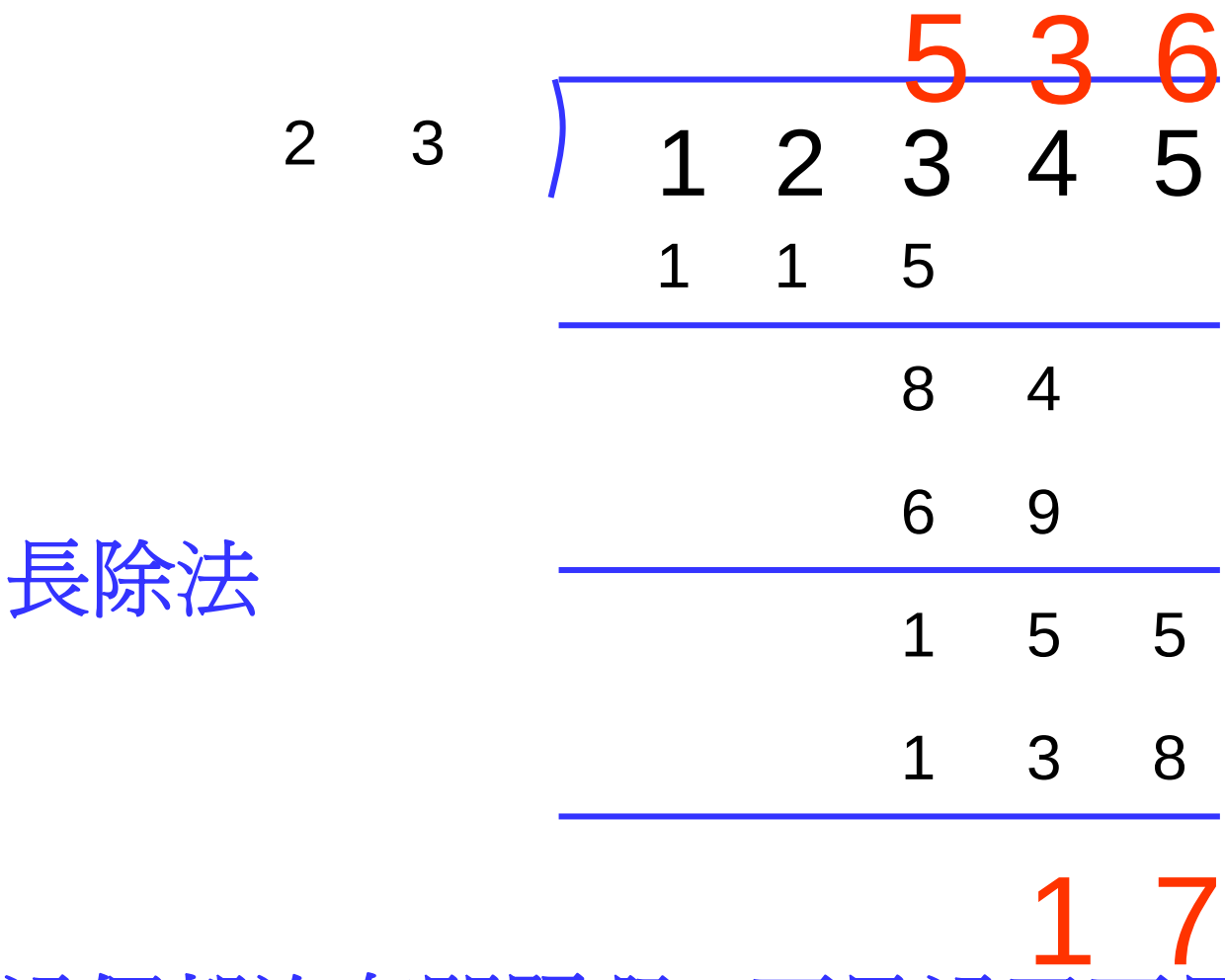
長除法

除法



長除法

除法

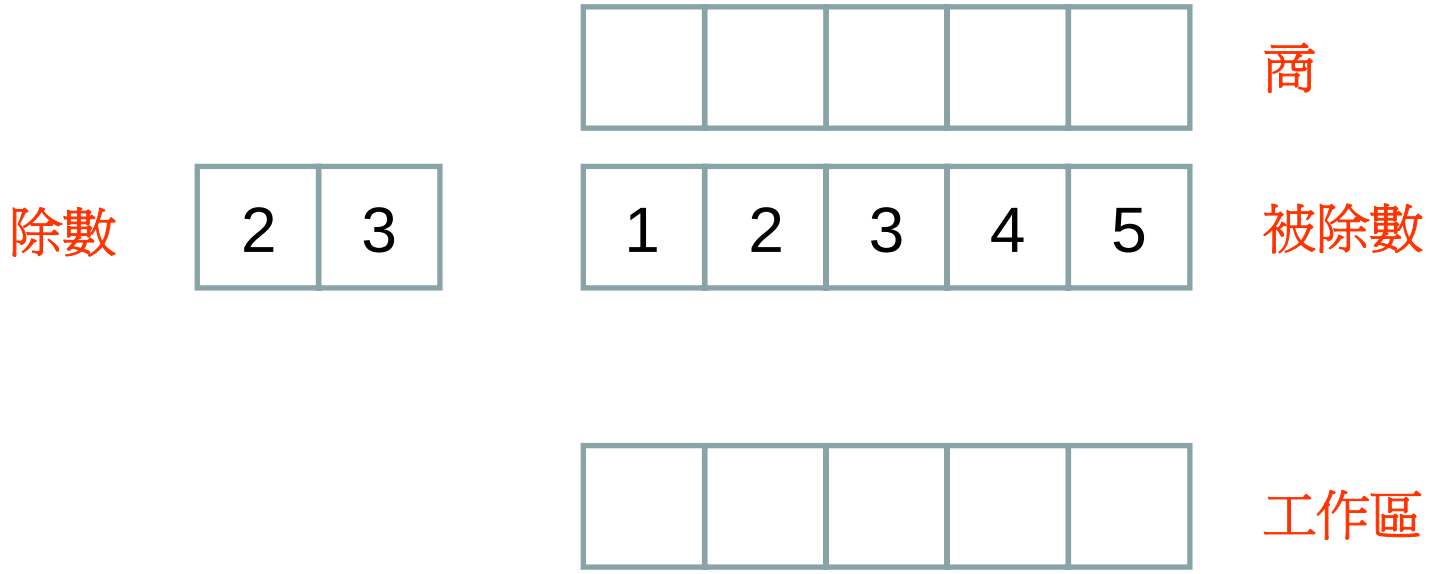


長除法

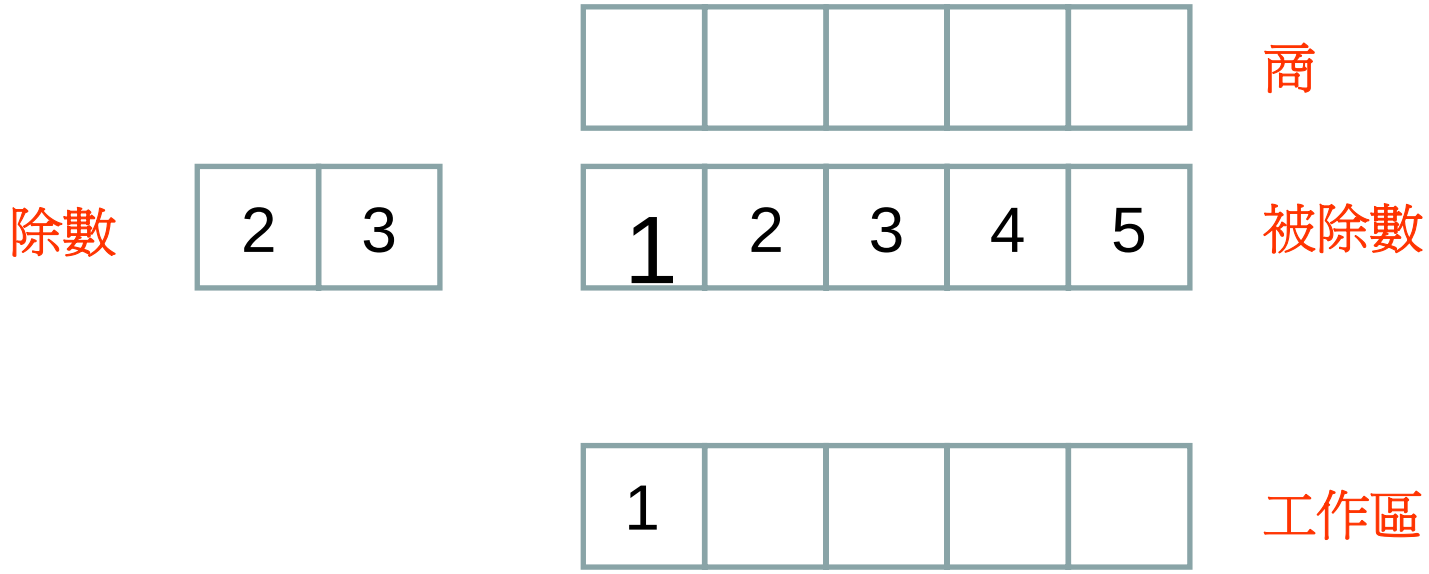
這個都沒有問題啊!! 可是這又不是程式?

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以

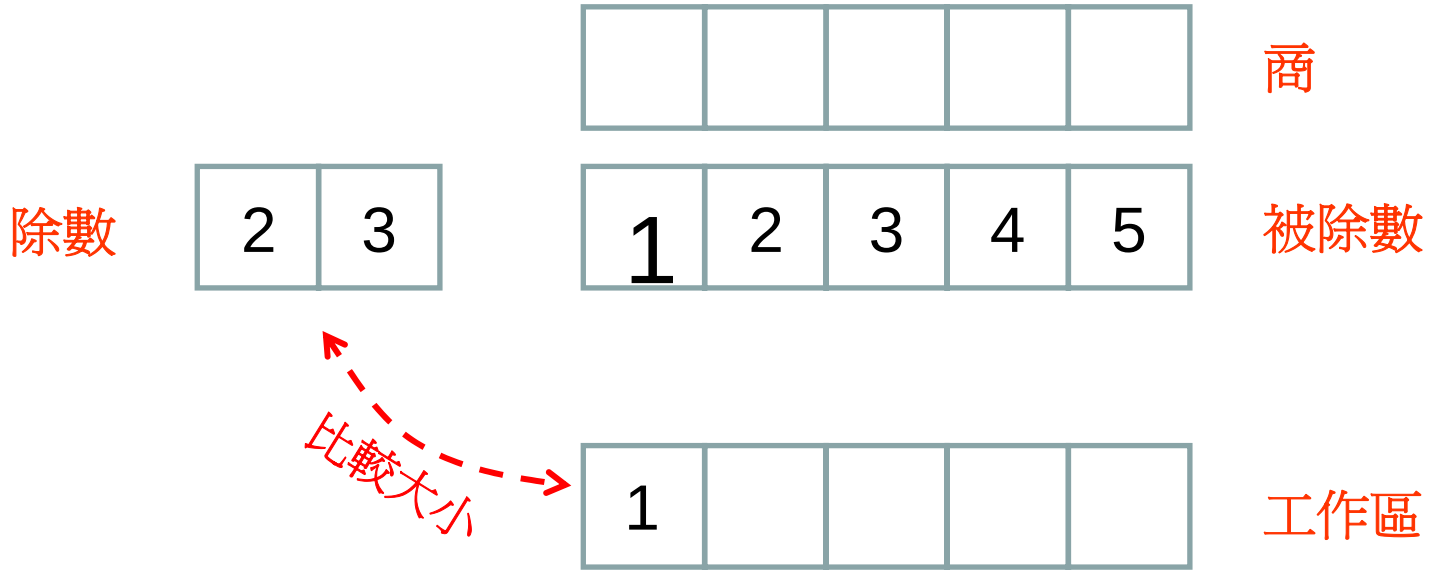
- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



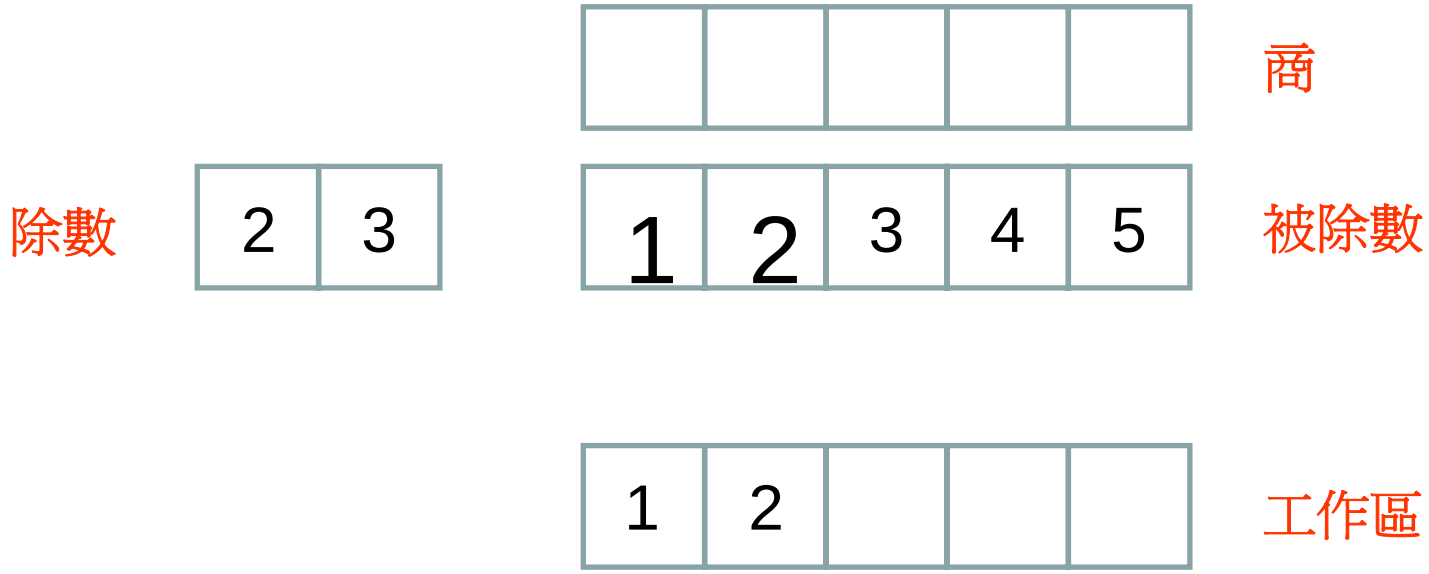
- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



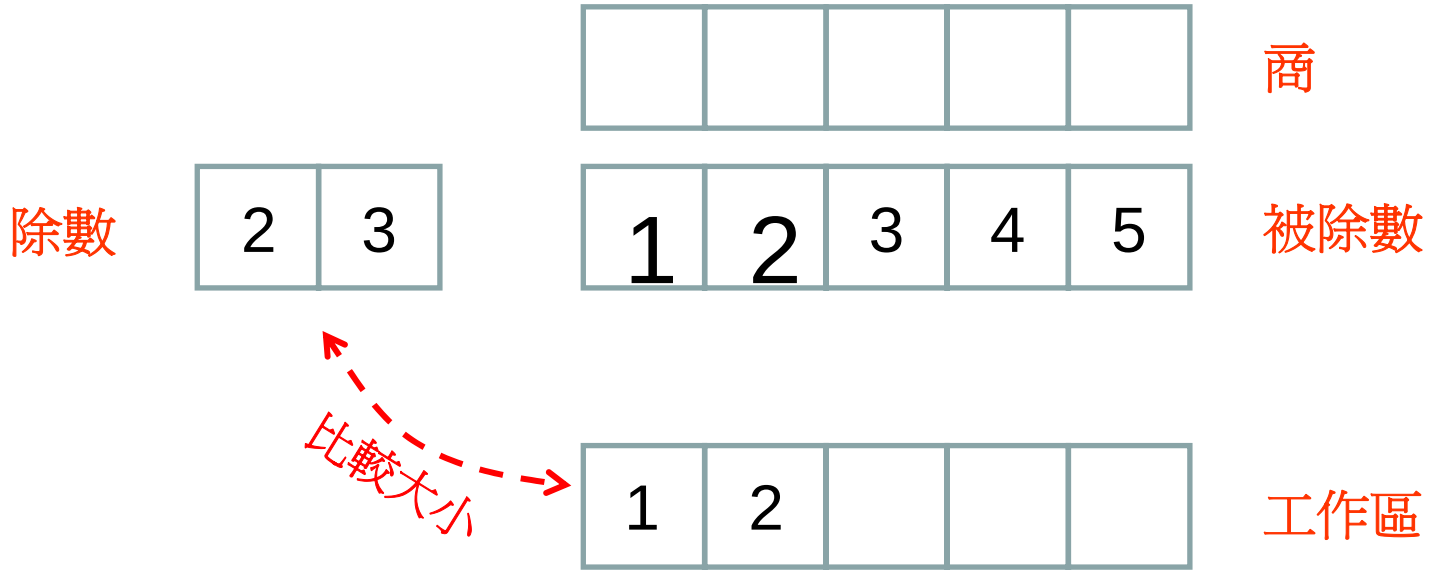
- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



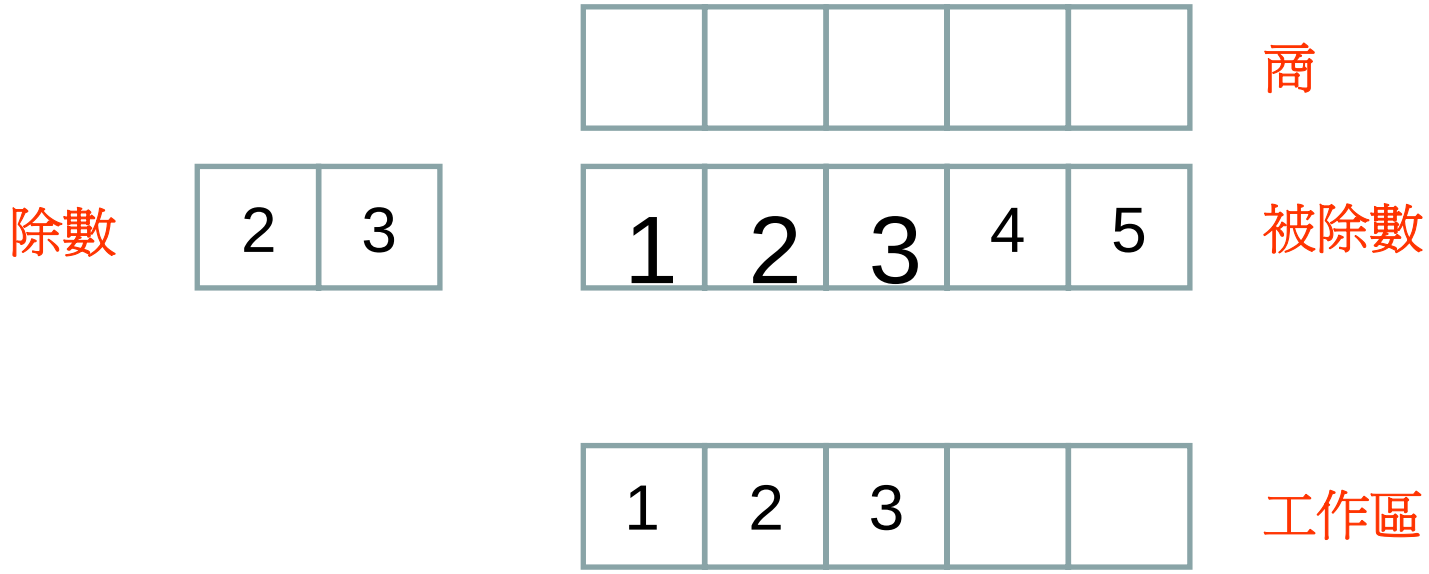
- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



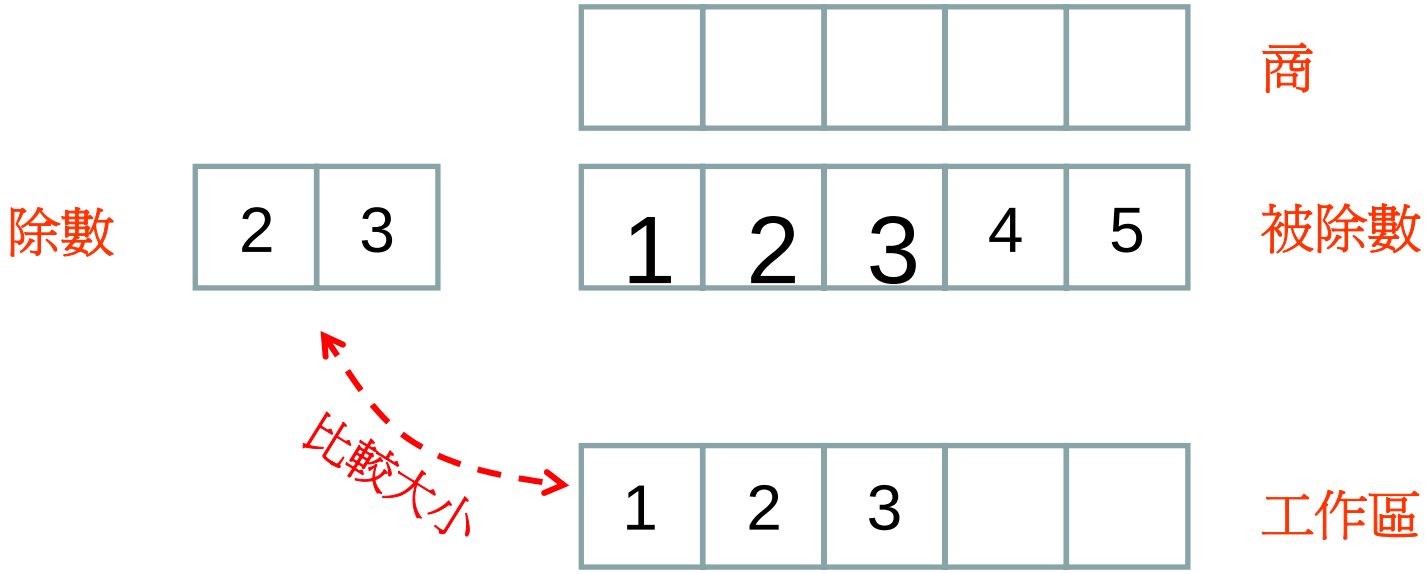
- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



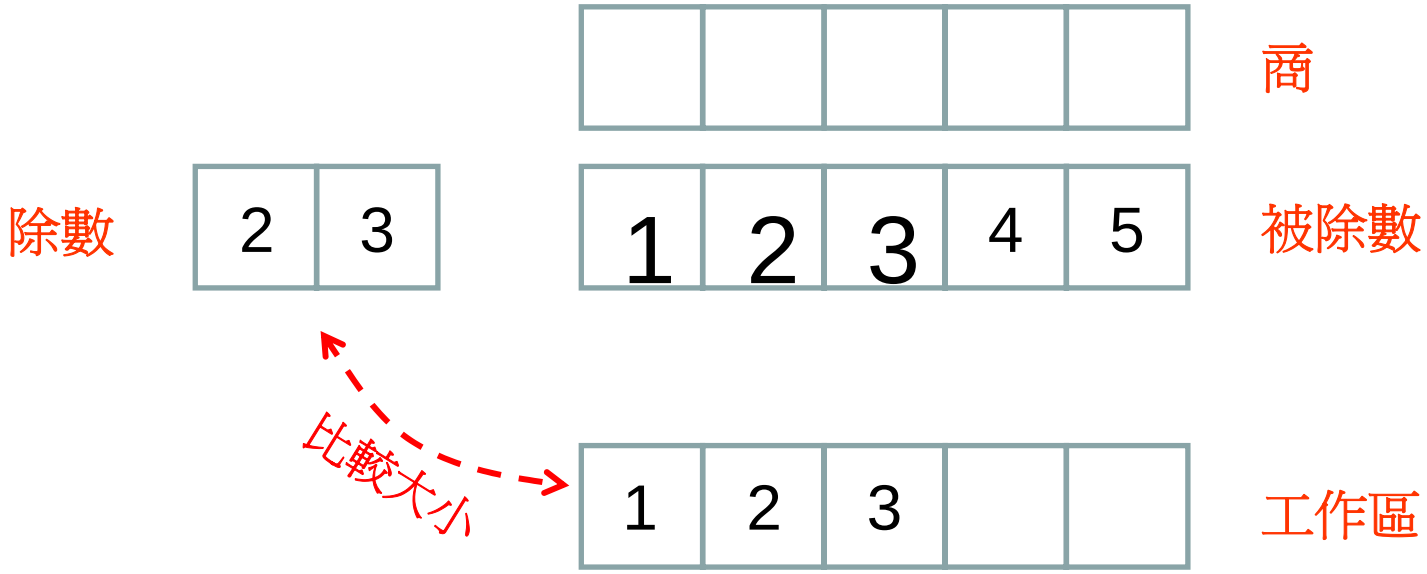
- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以

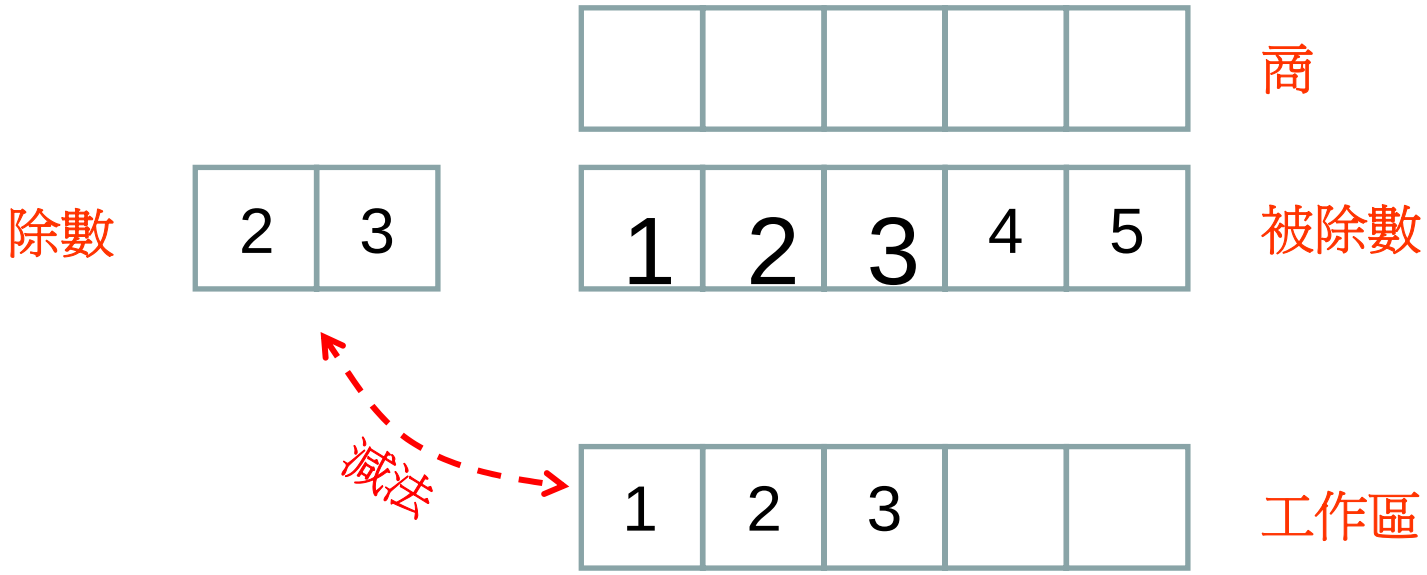


for (i=0; i<被除數位數; i++) {

拷貝一個位元到工作區

}

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以

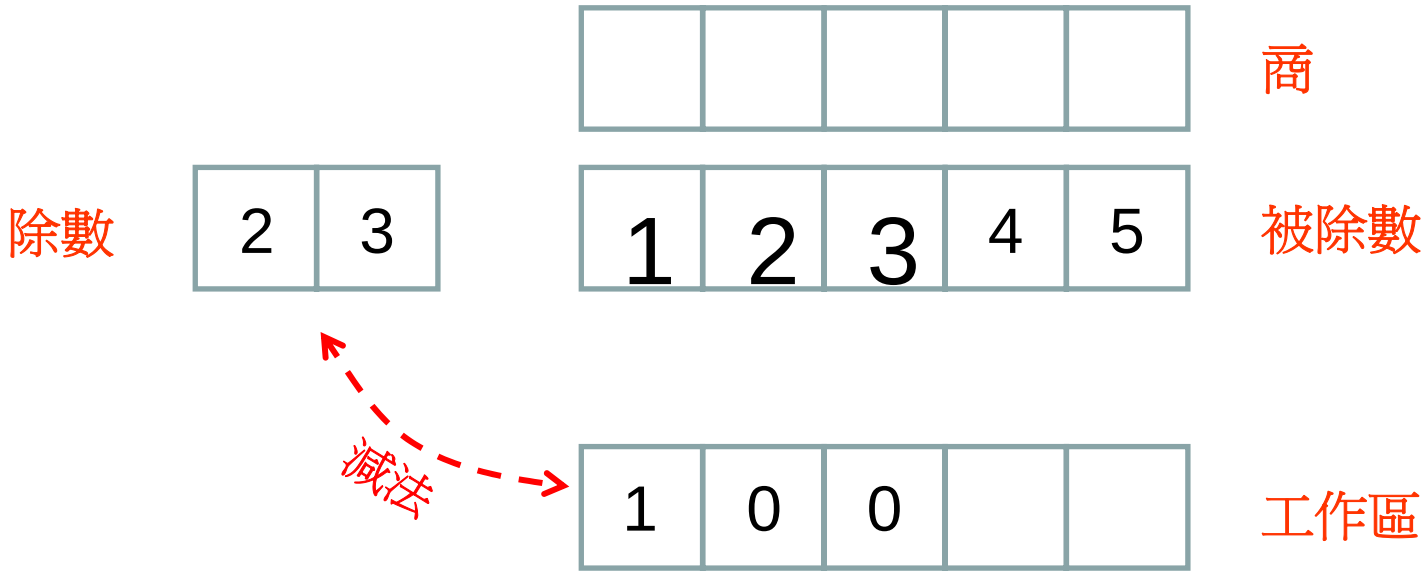


for (i=0; i<被除數位數; i++) {

拷貝一個位元到工作區

}

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以

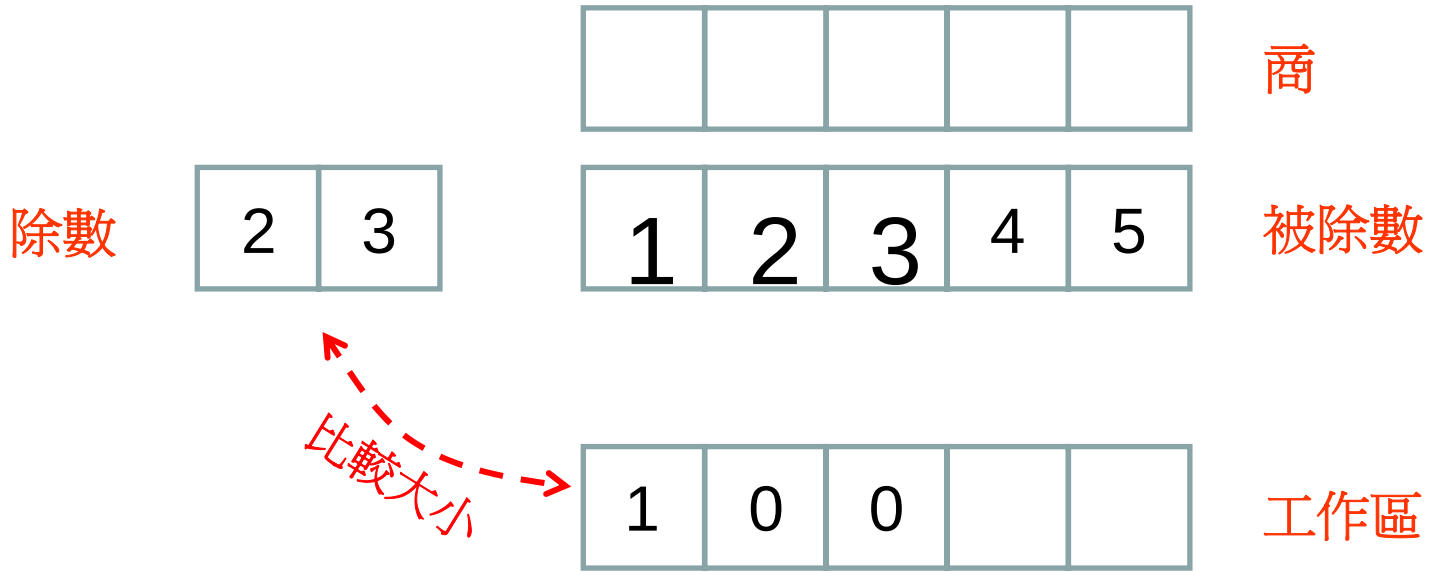


for (i=0; i<被除數位數; i++) {

拷貝一個位元到工作區

}

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以

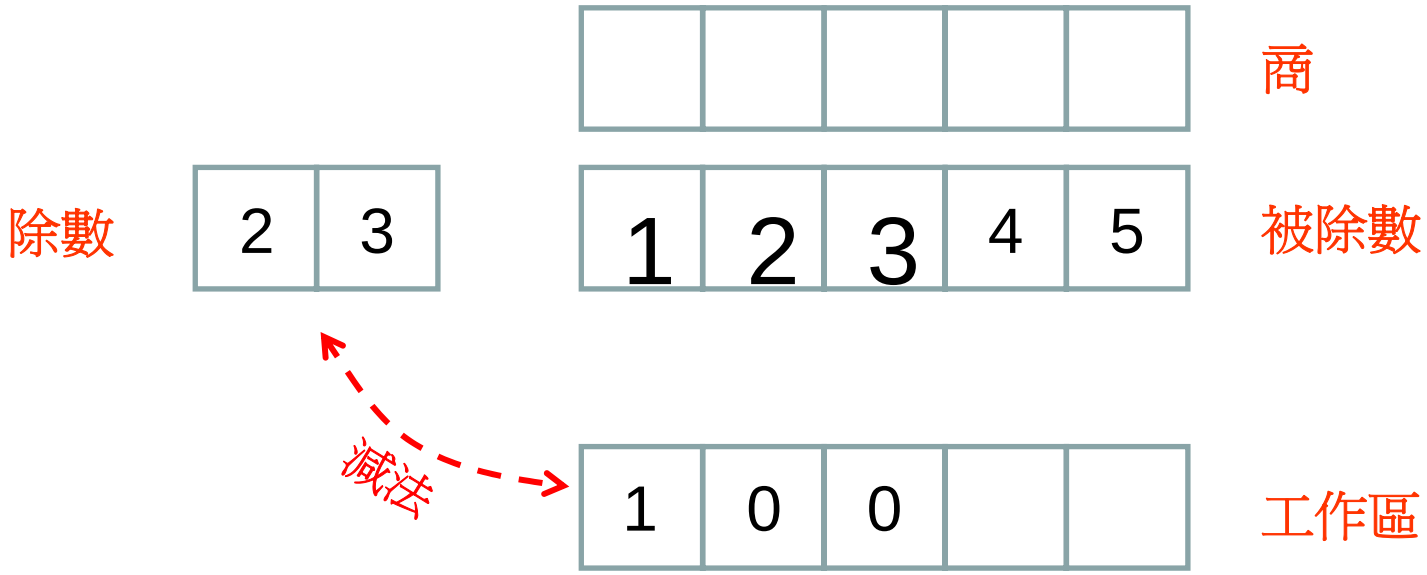


for (i=0; i<被除數位數; i++) {

拷貝一個位元到工作區

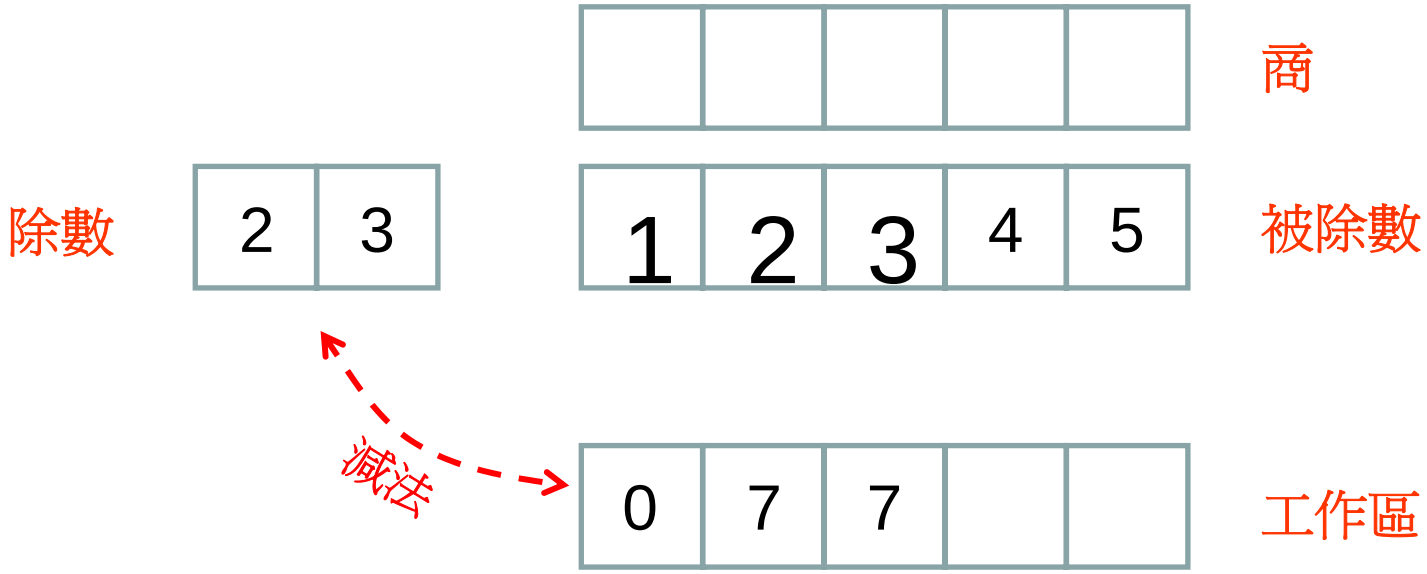
}

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以




```
for  (i=0; i<被除數位數; i++) {
    拷貝一個位元到工作區



}
```

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以

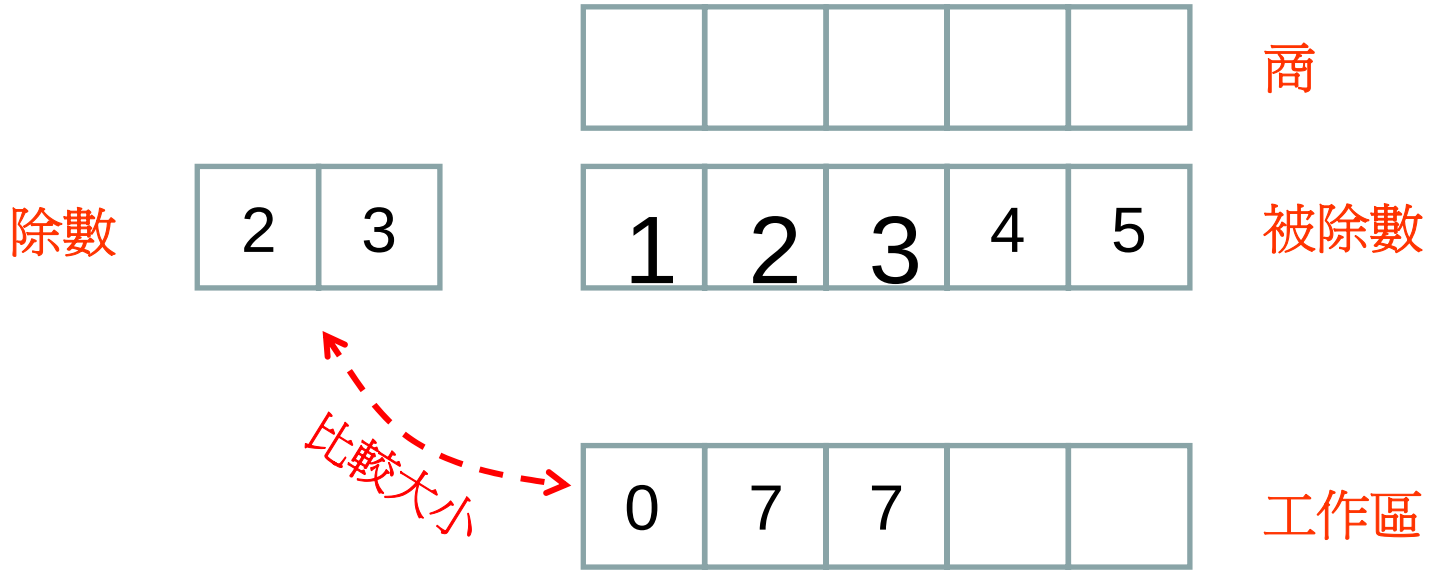


for (i=0; i<被除數位數; i++) {

拷貝一個位元到工作區

}

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以

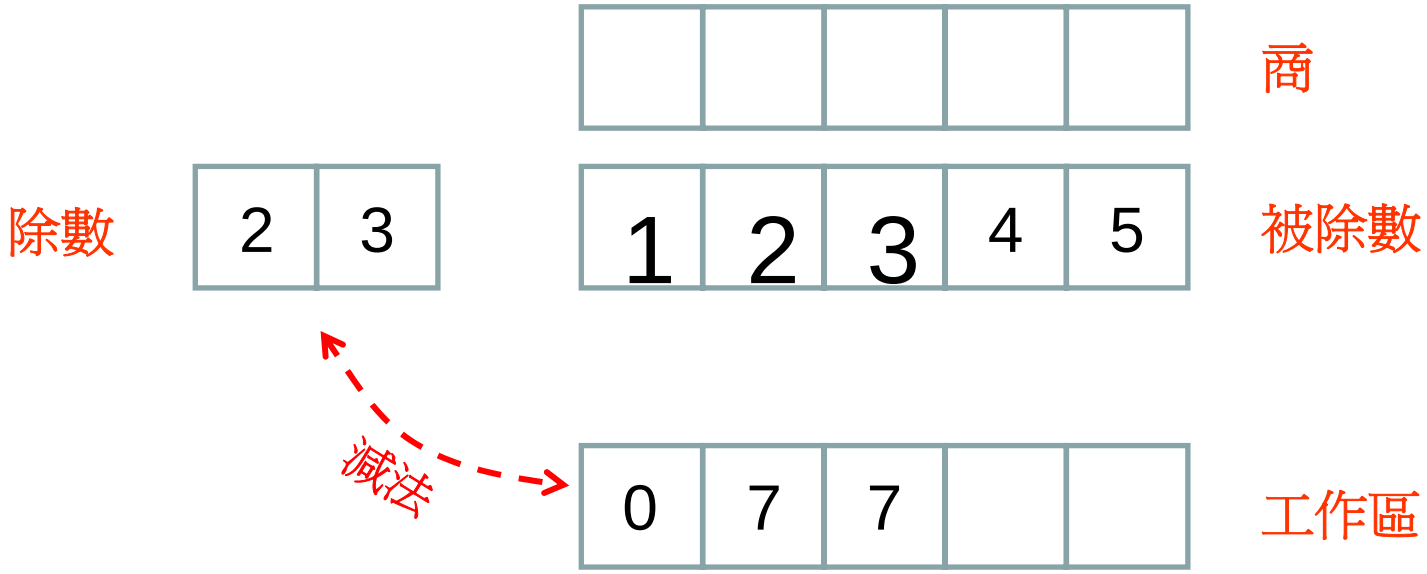


for (i=0; i<被除數位數; i++) {

拷貝一個位元到工作區

}

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以

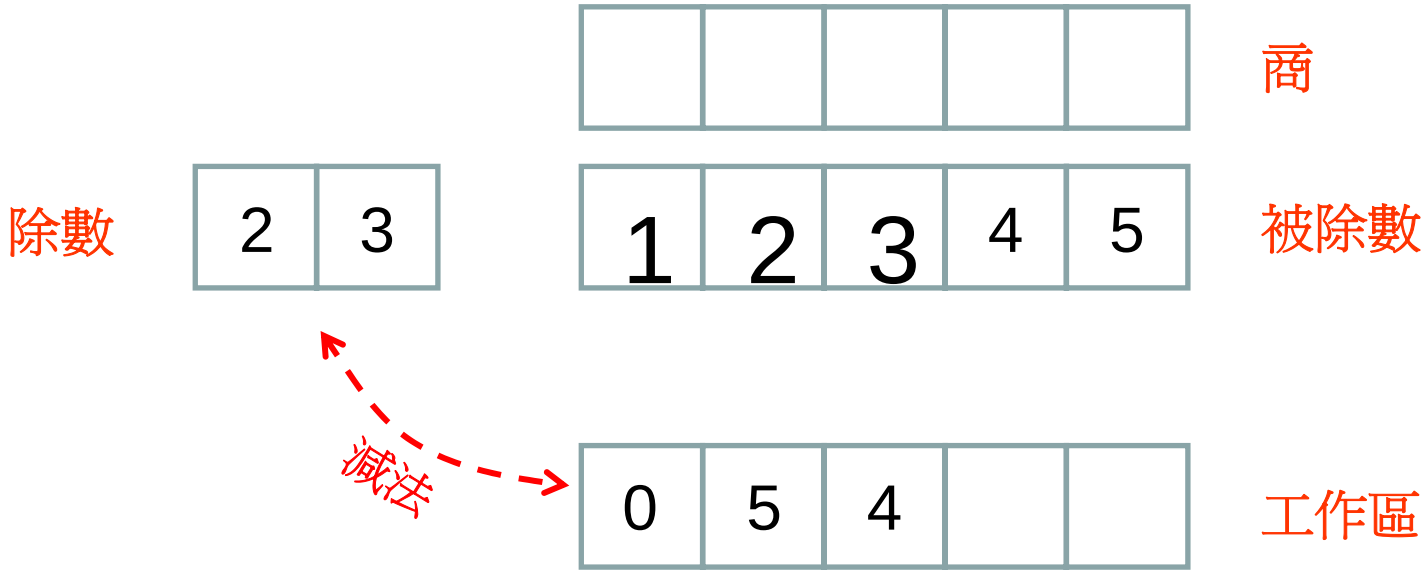


for (i=0; i<被除數位數; i++) {

拷貝一個位元到工作區

}

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以

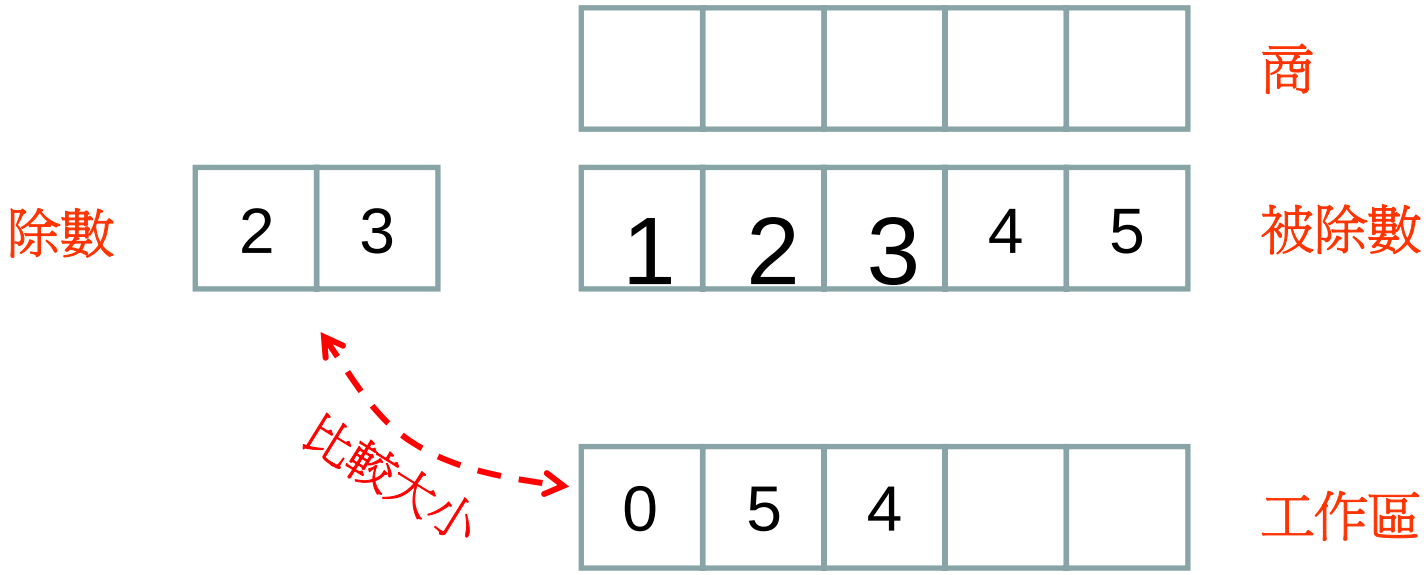


for (i=0; i<被除數位數; i++) {

拷貝一個位元到工作區

}

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以

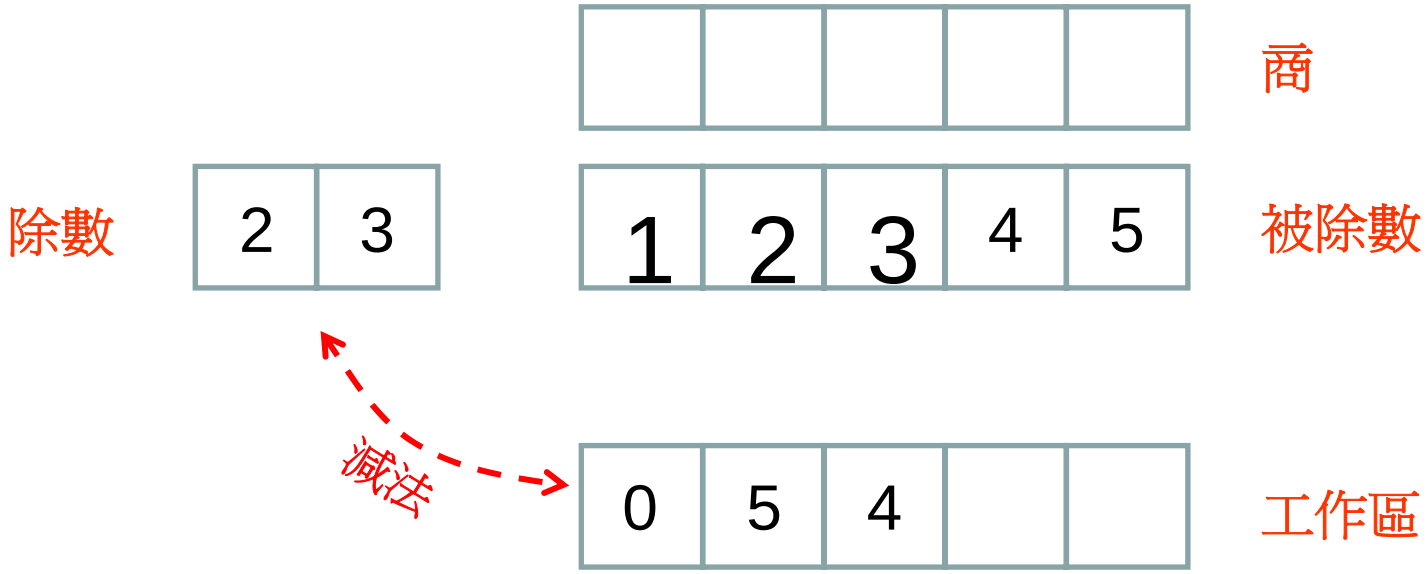


for (i=0; i<被除數位數; i++) {

拷貝一個位元到工作區

}

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以

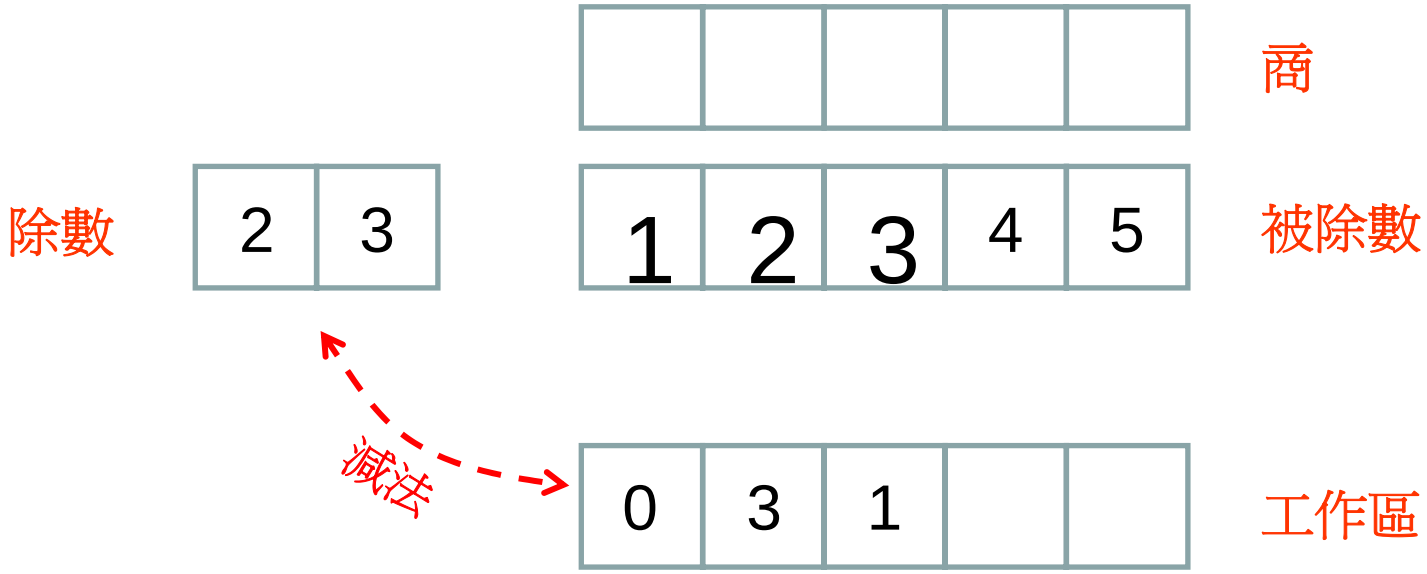


for (i=0; i<被除數位數; i++) {

拷貝一個位元到工作區

}

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以

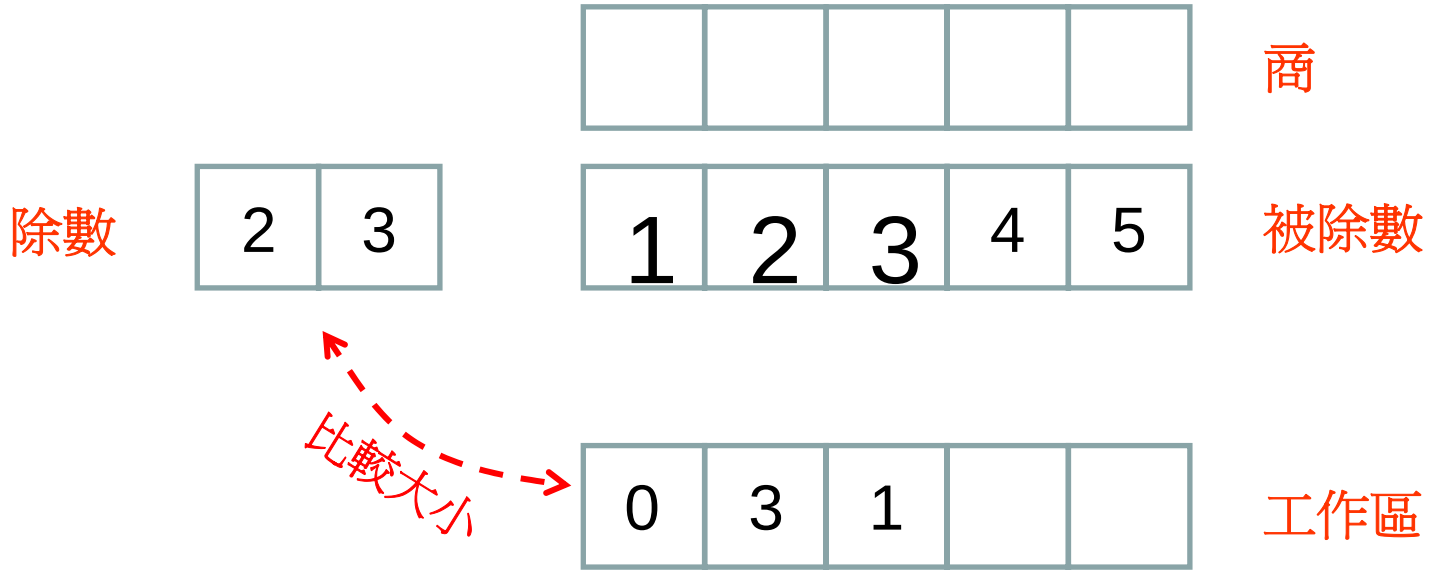


for (i=0; i<被除數位數; i++) {

拷貝一個位元到工作區

}

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以

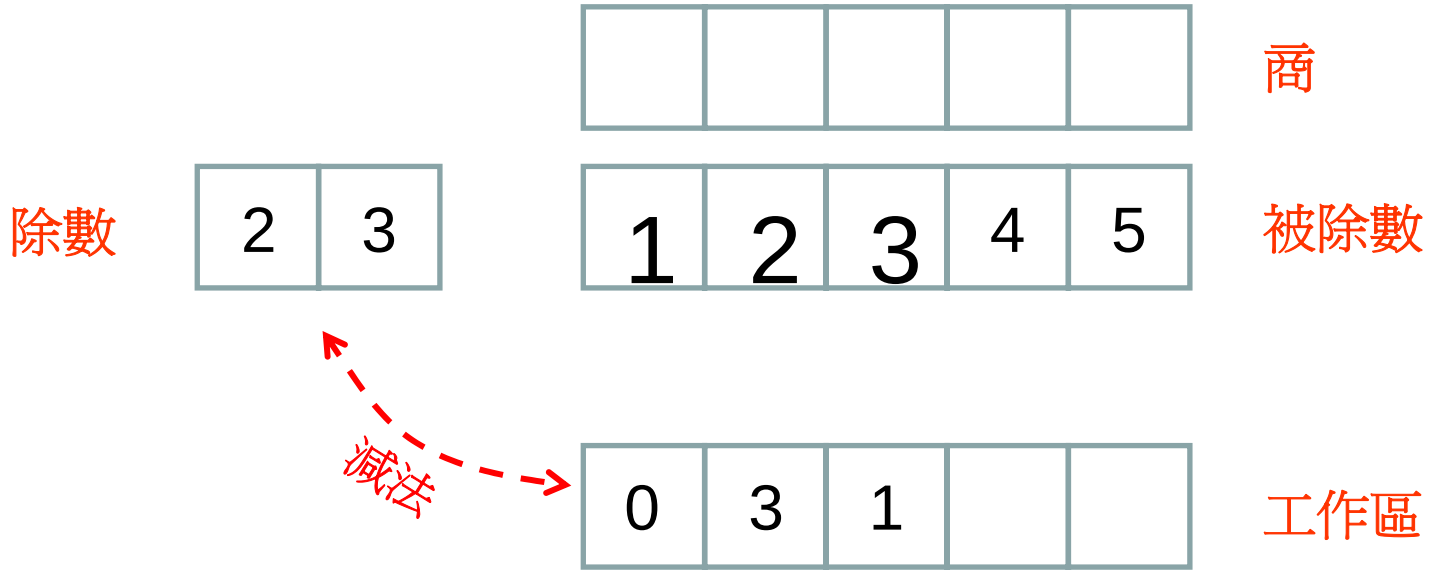


for (i=0; i<被除數位數; i++) {

拷貝一個位元到工作區

}

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以

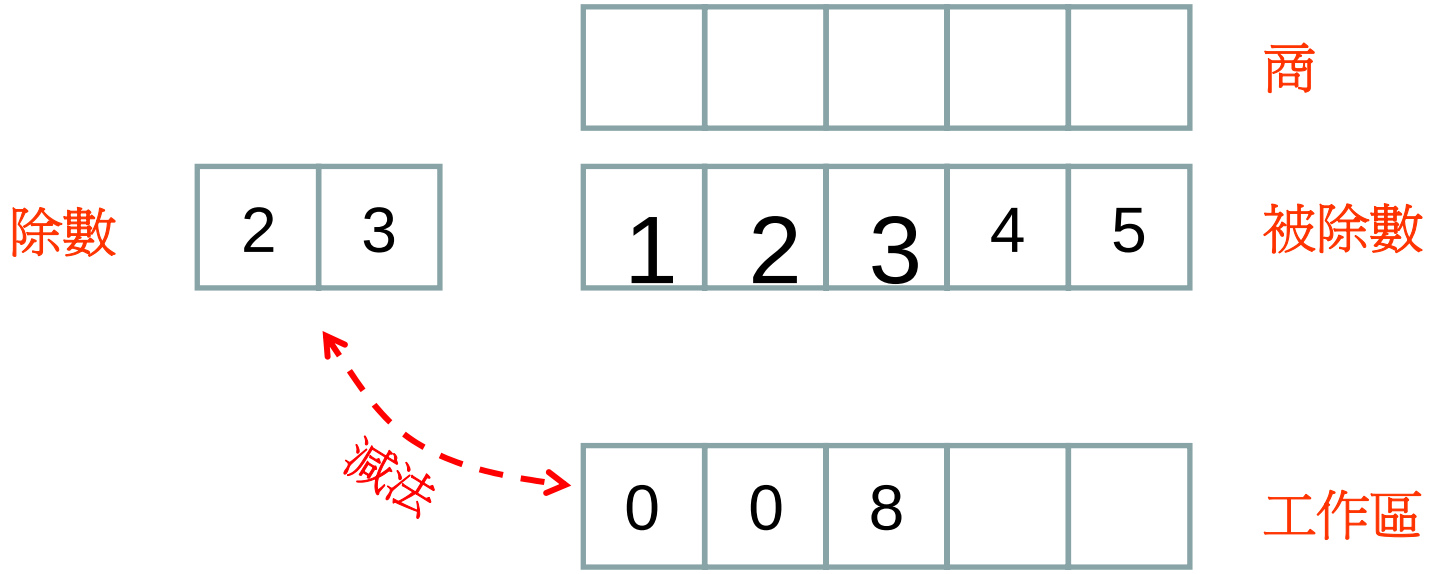


for (i=0; i<被除數位數; i++) {

拷貝一個位元到工作區

}

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以

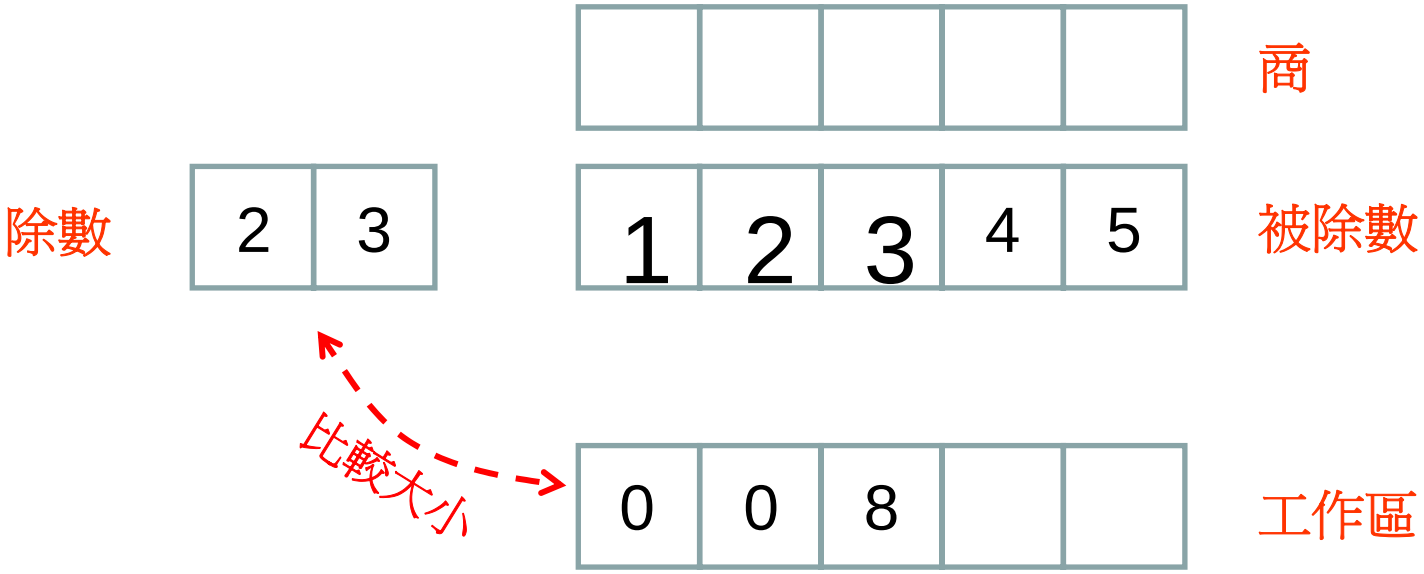


for (i=0; i<被除數位數; i++) {

拷貝一個位元到工作區

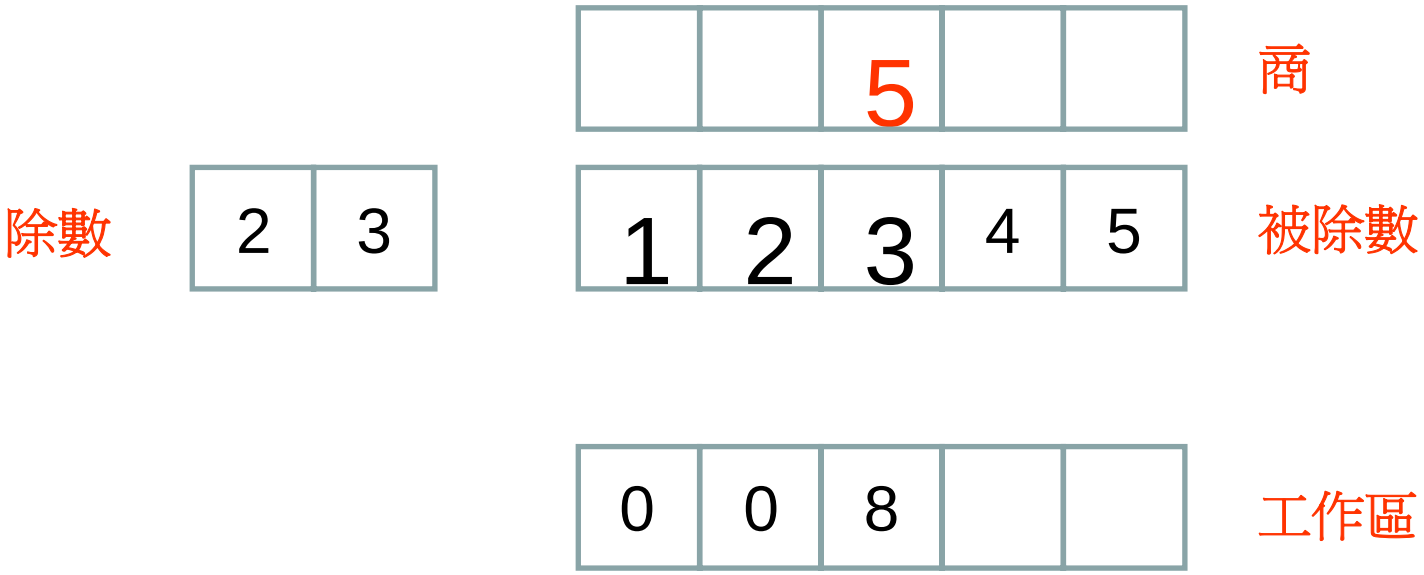
}

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



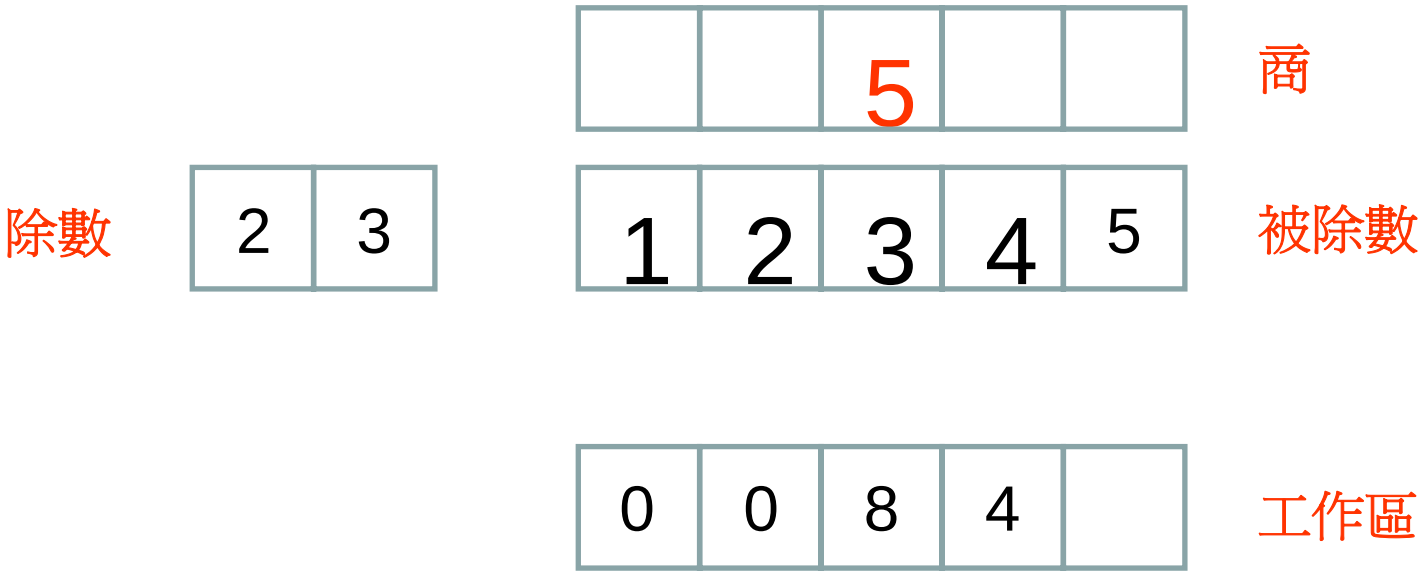

```
for  (i=0; i<被除數位數; i++) {
    拷貝一個位元到工作區
    while (工作區資料>=除數) {
        將 工作區資料 減 除數

    }
}
```

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



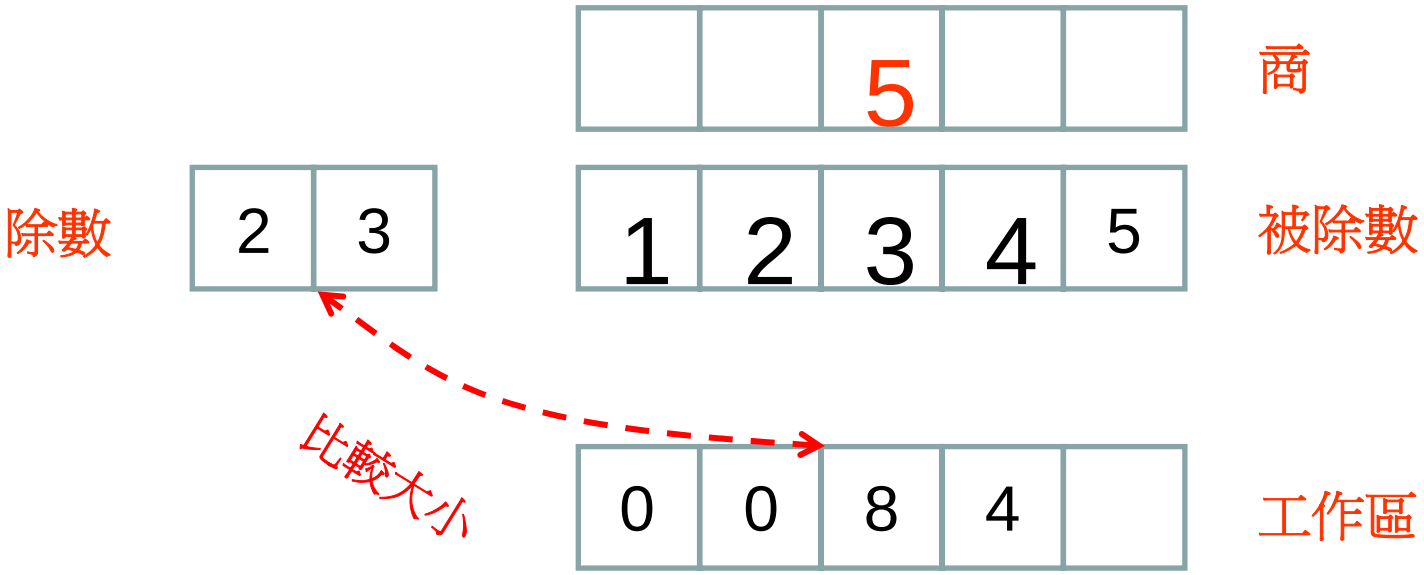

```
for  (i=0; i<被除數位數; i++) {
    拷貝一個位元到工作區
    while (工作區資料>=除數) {
        將 工作區資料 減 除數
        計算 while 迴圈執行次數
    }
}
```

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



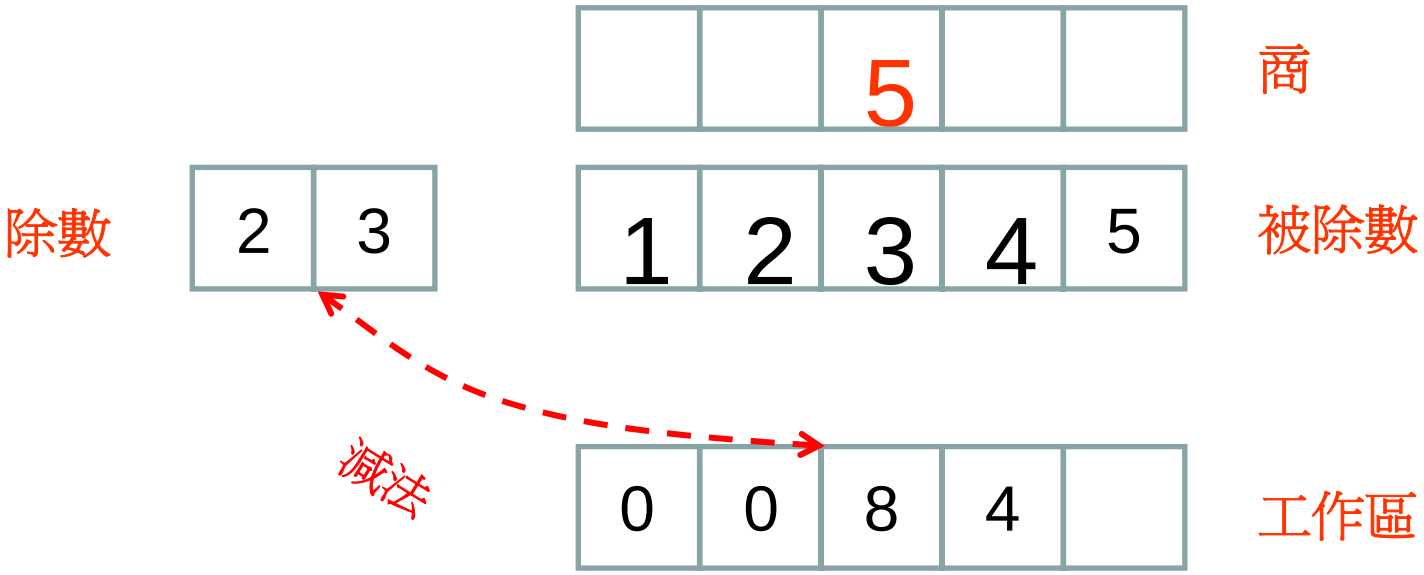

```
for  (i=0; i<被除數位數; i++) {
    拷貝一個位元到工作區
    while (工作區資料>=除數) {
        將 工作區資料 減 除數
        計算 while 迴圈執行次數
    }
}
```

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



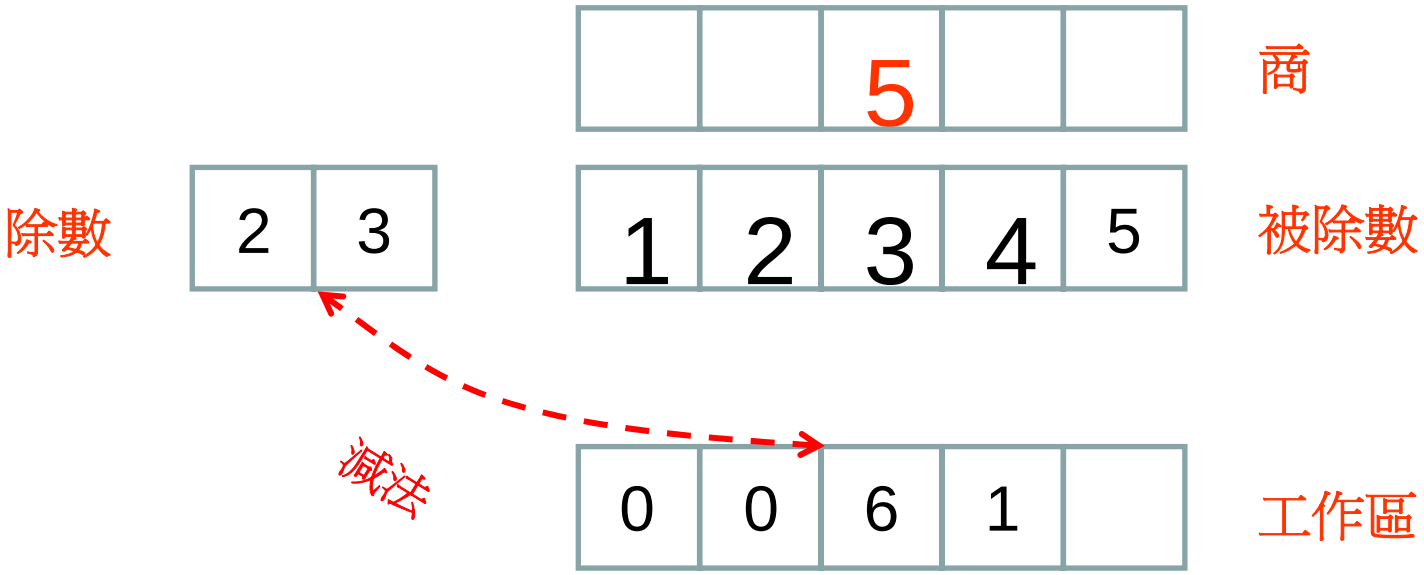

```
for  (i=0; i<被除數位數; i++) {
    拷貝一個位元到工作區
    while (工作區資料>=除數) {
        將 工作區資料 減 除數
        計算 while 迴圈執行次數
    }
}
```

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



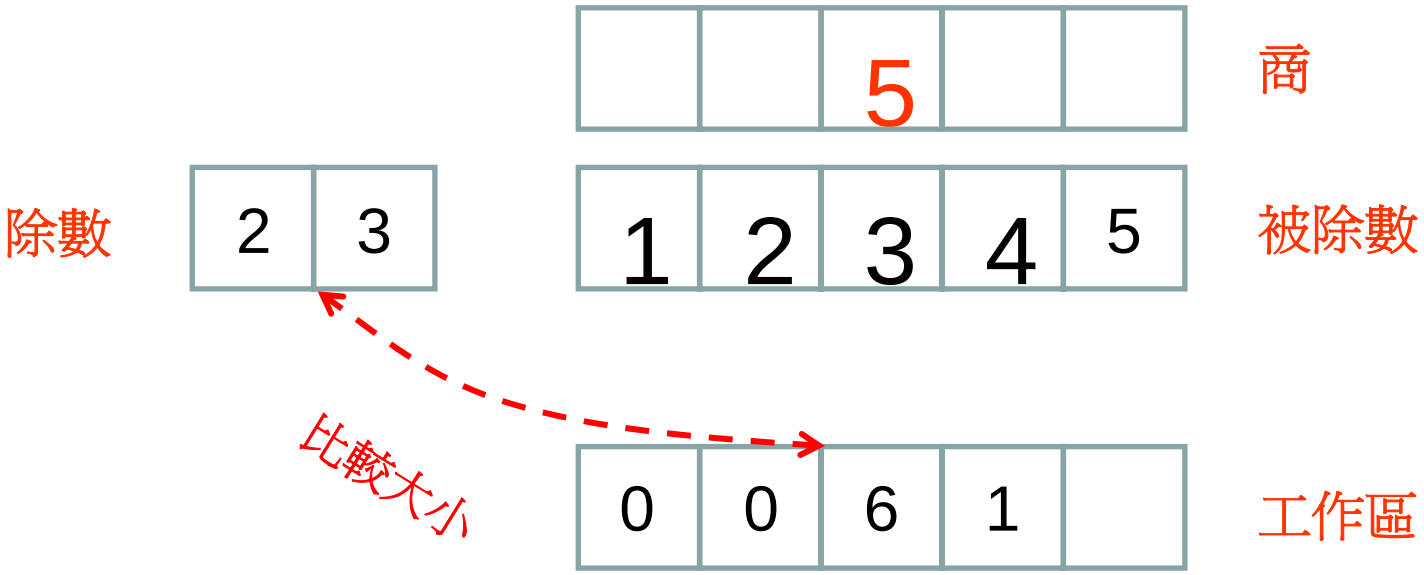

```
for  (i=0; i<被除數位數; i++) {
    拷貝一個位元到工作區
    while (工作區資料>=除數) {
        將 工作區資料 減 除數
        計算 while 迴圈執行次數
    }
}
```

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



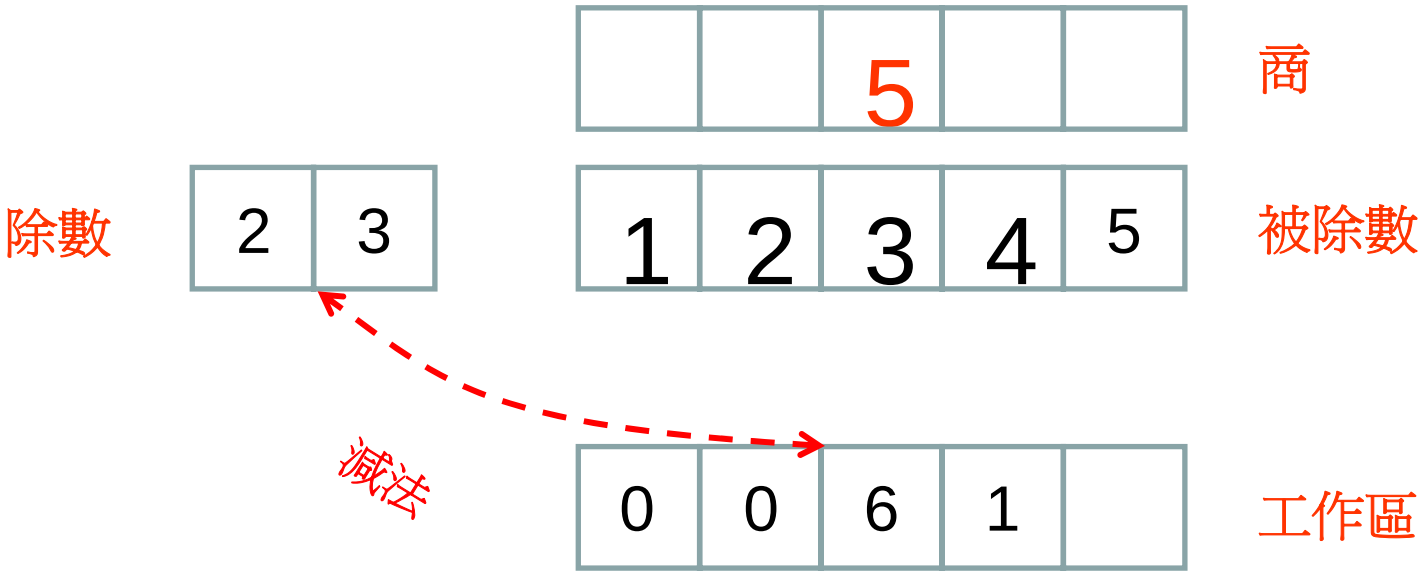

```
for  (i=0; i<被除數位數; i++) {
    拷貝一個位元到工作區
    while (工作區資料>=除數) {
        將 工作區資料 減 除數
        計算 while 迴圈執行次數
    }
}
```

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



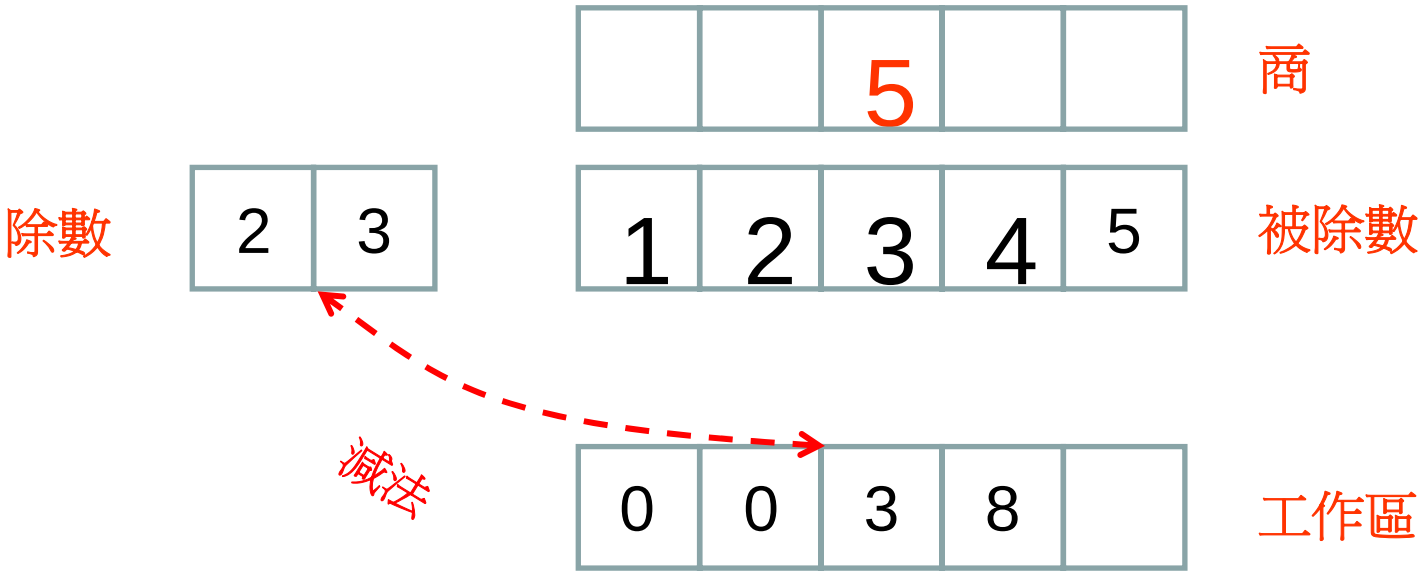

```
for  (i=0; i<被除數位數; i++) {
    拷貝一個位元到工作區
    while (工作區資料>=除數) {
        將 工作區資料 減 除數
        計算 while 迴圈執行次數
    }
}
```

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



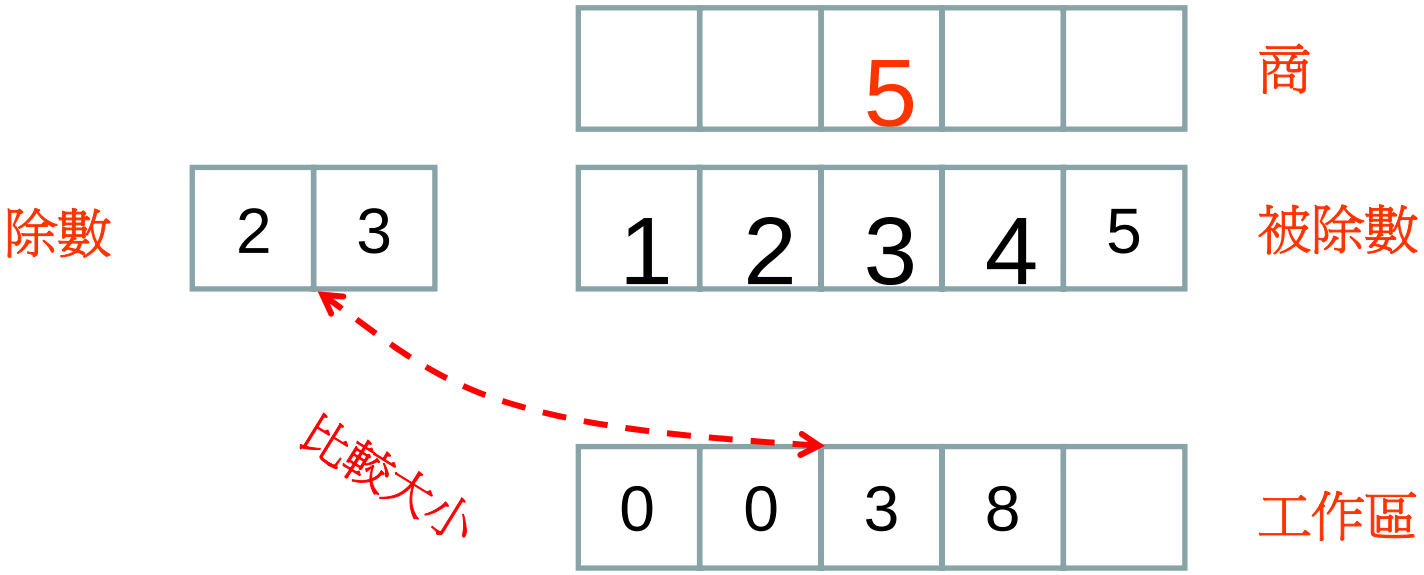

```
for  (i=0; i<被除數位數; i++) {
    拷貝一個位元到工作區
    while (工作區資料>=除數) {
        將 工作區資料 減 除數
        計算 while 迴圈執行次數
    }
}
```

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



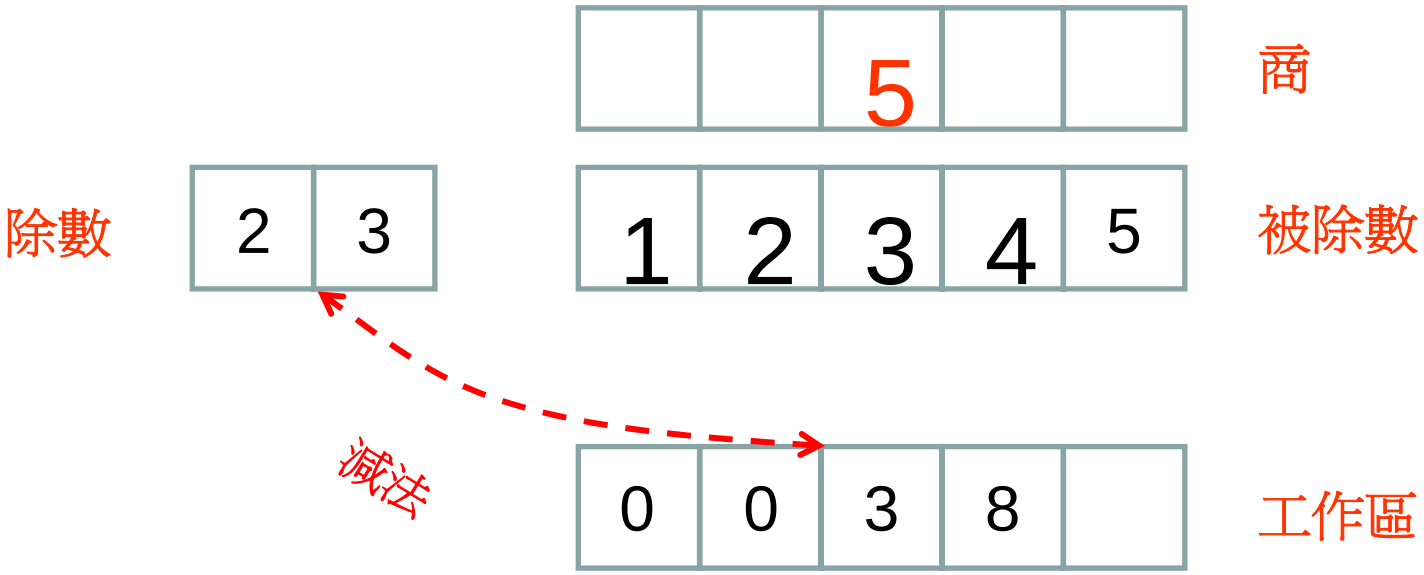

```
for  (i=0; i<被除數位數; i++) {
    拷貝一個位元到工作區
    while (工作區資料>=除數) {
        將 工作區資料 減 除數
        計算 while 迴圈執行次數
    }
}
```

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



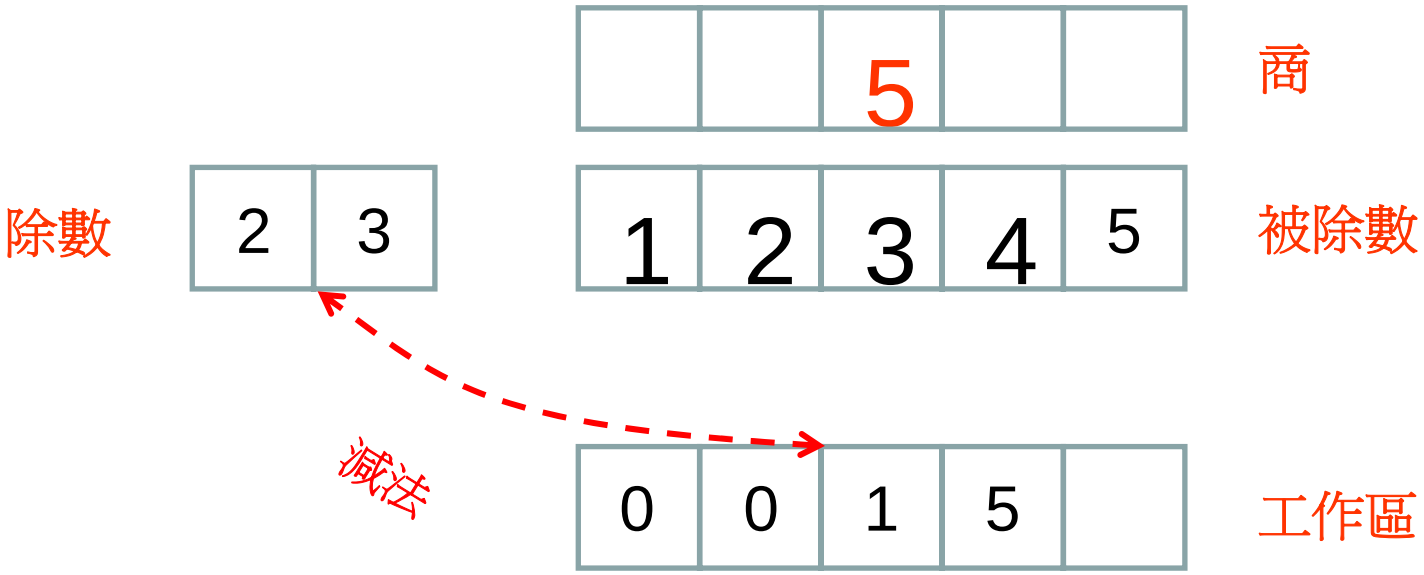

```
for  (i=0; i<被除數位數; i++) {
    拷貝一個位元到工作區
    while (工作區資料>=除數) {
        將 工作區資料 減 除數
        計算 while 迴圈執行次數
    }
}
```

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



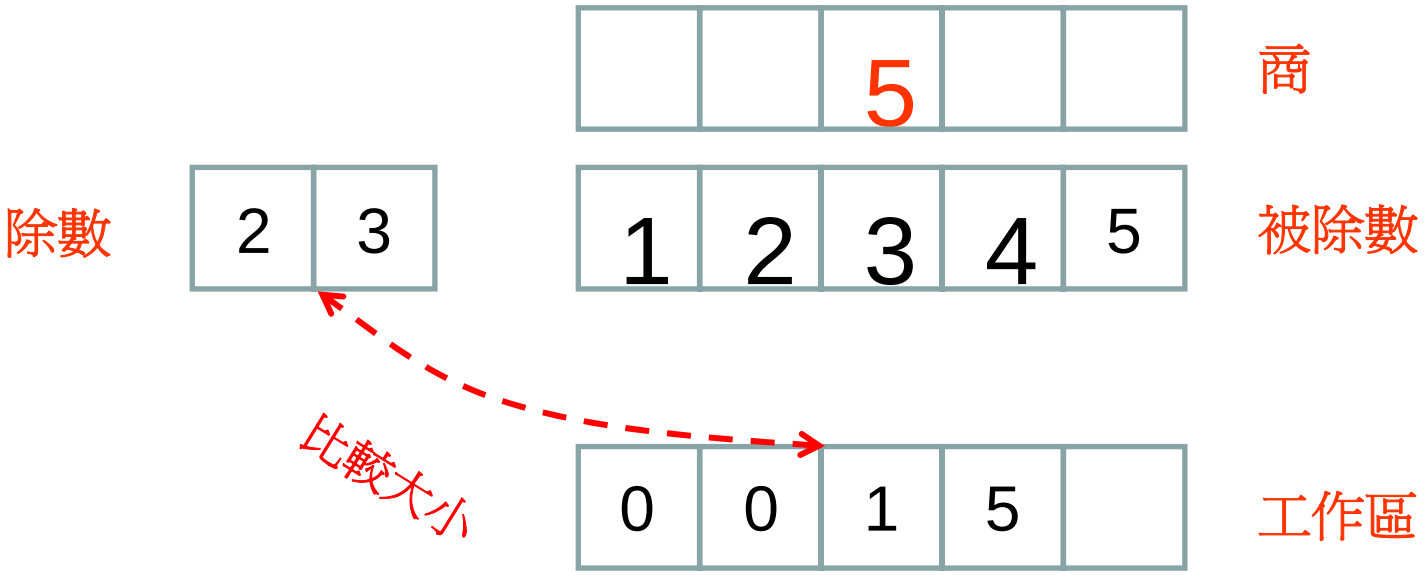

```
for  (i=0; i<被除數位數; i++) {
    拷貝一個位元到工作區
    while (工作區資料>=除數) {
        將 工作區資料 減 除數
        計算 while 迴圈執行次數
    }
}
```

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



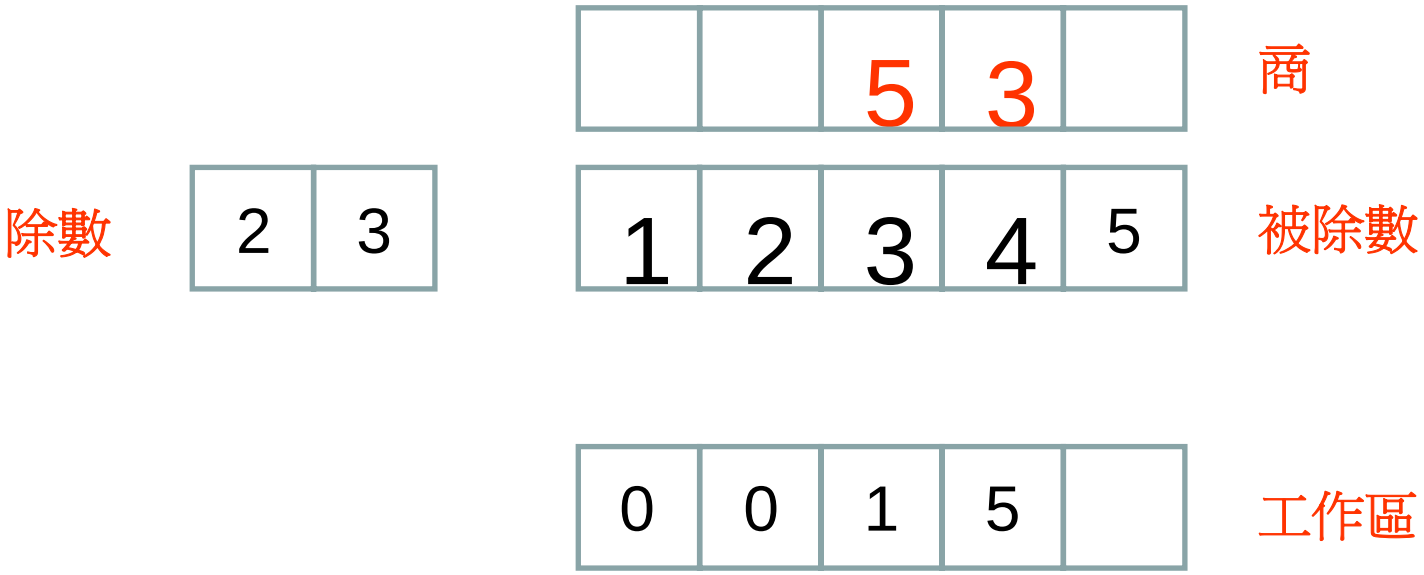

```
for  (i=0; i<被除數位數; i++) {
    拷貝一個位元到工作區
    while (工作區資料>=除數) {
        將 工作區資料 減 除數
        計算 while 迴圈執行次數
    }
}
```

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



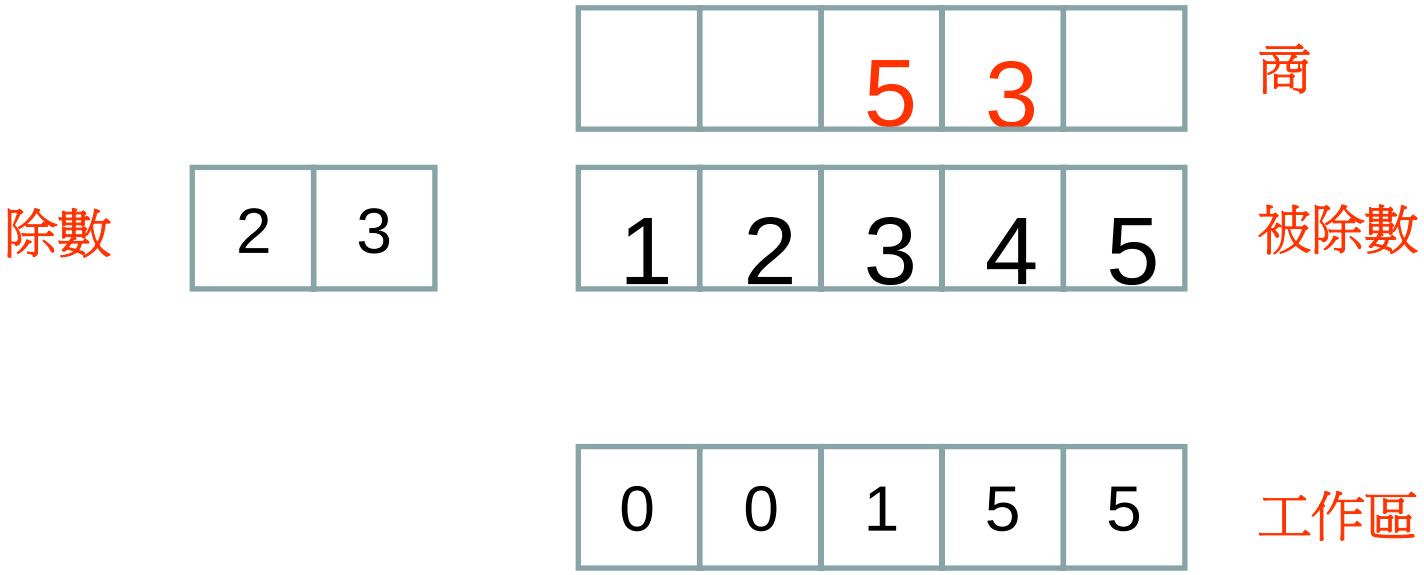

```
for  (i=0; i<被除數位數; i++) {
    拷貝一個位元到工作區
    while (工作區資料>=除數) {
        將 工作區資料 減 除數
        計算 while 迴圈執行次數
    }
}
```

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以



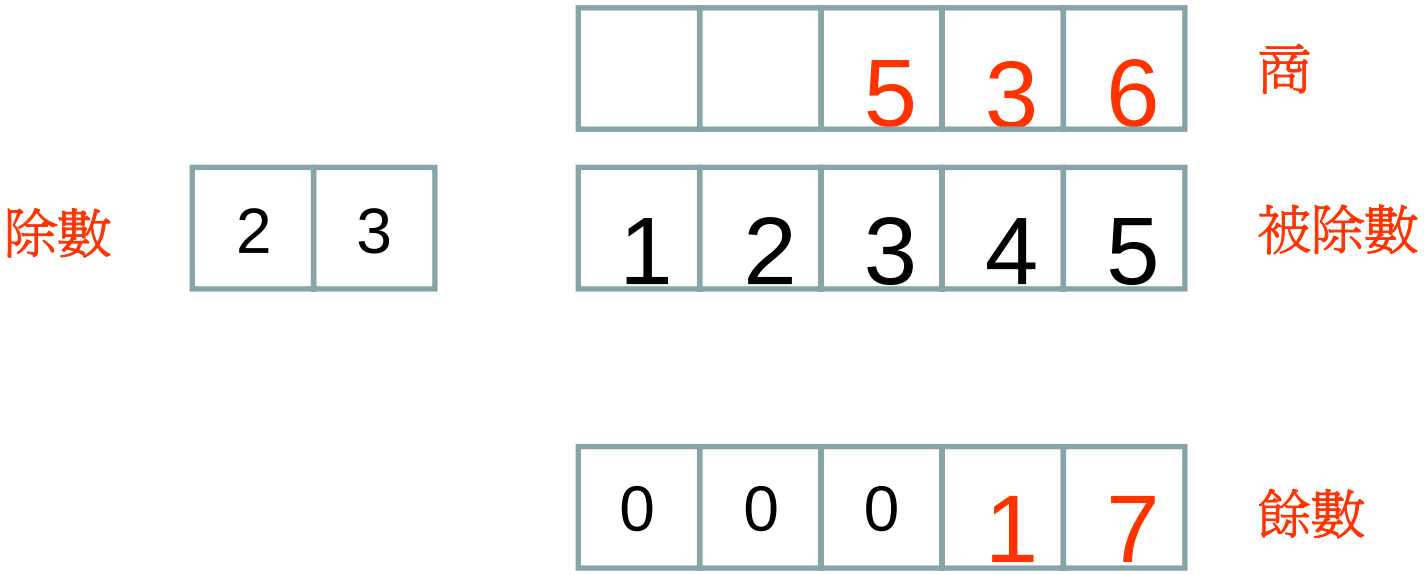

```
for  (i=0; i<被除數位數; i++) {
    拷貝一個位元到工作區
    while (工作區資料>=除數) {
        將 工作區資料 減 除數
        計算 while 迴圈執行次數
    }
}
```

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以




```
for  (i=0; i<被除數位數; i++) {
    拷貝一個位元到工作區
    while (工作區資料>=除數) {
        將 工作區資料 減 除數
        計算 while 迴圈執行次數
    }
}
```

- 好喔, 大家一起來, 看到這麼規律化的動作, 要看到**陣列**, 要看到**迴圈**, 要看到**演算法**, 要看到**程式**才可以




```
for  (i=0; i<被除數位數; i++) {
    拷貝一個位元到工作區
    while (工作區資料>=除數) {
        將 工作區資料 減 除數
        計算 while 迴圈執行次數
    }
}
```

函式的運用

- 如果你在寫程式的時候還在不斷地掙扎要不要寫函式?

函式的運用

- 如果你在寫程式的時候還在不斷地掙扎要不要寫函式?

  這有點糟, 你應該早就已經過了那個掙扎期了啊?

函式的運用

- 如果你在寫程式的時候還在不斷地掙扎要不要寫函式?

  這有點糟, 你應該早就已經過了那個掙扎期了啊?

- 使用函式的目的是為了在程式碼重複出現時少寫一點嗎?

函式的運用

- 如果你在寫程式的時候還在不斷地掙扎要不要寫函式?

  這有點糟, 你應該早就已經過了那個掙扎期了啊?

- 使用函式的目的是為了在程式碼重複出現時少寫一點嗎?

  那麼只出現一次的程式碼不應該寫成函式囉?

函式的運用

- 如果你在寫程式的時候還在不斷地掙扎要不要寫函式?

  這有點糟, 你應該早就已經過了那個掙扎期了啊?

- 使用函式的目的是為了在程式碼重複出現時少寫一點嗎?

  那麼只出現一次的程式碼不應該寫成函式囉?

- 對資訊系的同學來說, **函式**應該是

函式的運用

- 如果你在寫程式的時候還在不斷地掙扎要不要寫函式?

  這有點糟, 你應該早就已經過了那個掙扎期了啊?

- 使用函式的目的是為了在程式碼重複出現時少寫一點嗎?

  那麼只出現一次的程式碼不應該寫成函式囉?

- 對資訊系的同學來說, **函式**應該是
  - **抽象化的工具:** 把下層細節藏起來的工具, 你應該可以有信心地假設函式內都正常運作, 運用個別函式組合出你需要的功能

函式的運用

- 如果你在寫程式的時候還在不斷地掙扎要不要寫函式?

  這有點糟, 你應該早就已經過了那個掙扎期了啊?

- 使用函式的目的是為了在程式碼重複出現時少寫一點嗎?

  那麼只出現一次的程式碼不應該寫成函式囉?

- 對資訊系的同學來說, **函式**應該是
  - **抽象化的工具:** 把下層細節藏起來的工具, 你應該可以有信心地假設函式內都正常運作, 運用個別函式組合出你需要的功能
  - **模組化的工具:** 給定有意義的名稱把相關功能的程式碼綁起來工具

函式的運用

- 如果你在寫程式的時候還在不斷地掙扎要不要寫函式?

  這有點糟, 你應該早就已經過了那個掙扎期了啊?

- 使用函式的目的是為了在程式碼重複出現時少寫一點嗎?

  那麼只出現一次的程式碼不應該寫成函式囉?

- 對資訊系的同學來說, **函式**應該是
  - **抽象化的工具:** 把下層細節藏起來的工具, 你應該可以有信心地假設函式內都正常運作, 運用個別函式組合出你需要的功能
  - **模組化的工具:** 給定有意義的名稱把相關功能的程式碼綁起來工具
  - **預設查核點的工具:** 運用函式切割流程, 清楚掌握資料的改變狀況

函式的運用

- 如果你在寫程式的時候還在不斷地掙扎要不要寫函式?

  這有點糟, 你應該早就已經過了那個掙扎期了啊?

- 使用函式的目的是為了在程式碼重複出現時少寫一點嗎?

  那麼只出現一次的程式碼不應該寫成函式囉?

- 對資訊系的同學來說, **函式**應該是
  - **抽象化的工具:** 把下層細節藏起來的工具, 你應該可以有信心地假設函式內都正常運作, 運用個別函式組合出你需要的功能
  - **模組化的工具:** 給定有意義的名稱把相關功能的程式碼綁起來工具
  - **預設查核點的工具:** 運用函式切割流程, 清楚掌握資料的改變狀況
  - **拆解程式與分工的工具:** 一個函式是一個小程式, 有自己的要求

函式的運用

- 如果你在寫程式的時候還在不斷地掙扎要不要寫函式?

  這有點糟, 你應該早就已經過了那個掙扎期了啊?

- 使用函式的目的是為了在程式碼重複出現時少寫一點嗎?

  那麼只出現一次的程式碼不應該寫成函式囉?

- 對資訊系的同學來說, **函式**應該是
  - **抽象化的工具:** 把下層細節藏起來的工具, 你應該可以有信心地假設函式內都正常運作, 運用個別函式組合出你需要的功能
  - **模組化的工具:** 給定有意義的名稱把相關功能的程式碼綁起來工具
  - **預設查核點的工具:** 運用函式切割流程, 清楚掌握資料的改變狀況
  - **拆解程式與分工的工具:** 一個函式是一個小程式, 有自己的要求
  - **防止資料耦合的工具:** 在程式與不相關變數之間劃出界線的工具

函式的運用

- 如果你在寫程式的時候還在不斷地掙扎要不要寫函式?

  這有點糟, 你應該早就已經過了那個掙扎期了啊?

- 使用函式的目的是為了在程式碼重複出現時少寫一點嗎?

  那麼只出現一次的程式碼不應該寫成函式囉?

- 對資訊系的同學來說, **函式**應該是
  - **抽象化的工具:** 把下層細節藏起來的工具, 你應該可以有信心地假設函式內都正常運作, 運用個別函式組合出你需要的功能
  - **模組化的工具:** 給定有意義的名稱把相關功能的程式碼綁起來工具
  - **預設查核點的工具:** 運用函式切割流程, 清楚掌握資料的改變狀況
  - **拆解程式與分工的工具:** 一個函式是一個小程式, 有自己的要求
  - **防止資料耦合的工具:** 在程式與不相關變數之間劃出界線的工具

  在**設計階段**自然浮出協助你的, 不是程式要繳交時才分割出來的