ZeroJudge e283
小威的特殊編碼

ZeroJudge e283 小威的特殊編碼

https://zerojudge.tw/ShowProblem?problemid=e283

107/06/09 APCS #1 近似題

丁培毅

110/08

問題描述

- 任何文字或數字儲存在電腦中都是使用 0 與 1 構成的二元編碼序列，A~F 的編碼方式如下表：

字元	A	B	C	D	E	F
編碼	0 1 0 1	0 1 1 1	0 0 1 0	1 1 0 1	1 0 0 0	1 1 0 0

問題描述

- 任何文字或數字儲存在電腦中都是使用 0 與 1 構成的二元編碼序列，A~F 的編碼方式如下表：

字元	A	B	C	D	E	F
編碼	0 1 0 1	0 1 1 1	0 0 1 0	1 1 0 1	1 0 0 0	1 1 0 0

- 請寫一個程式辨識 0 1 編碼序列，輸出對應的字元

問題描述

- 任何文字或數字儲存在電腦中都是使用 0 與 1 構成的二元編碼序列，A~F 的編碼方式如下表：

字元	A	B	C	D	E	F
編碼	0 1 0 1	0 1 1 1	0 0 1 0	1 1 0 1	1 0 0 0	1 1 0 0

- 請寫一個程式辨識 0 1 編碼序列，輸出對應的字元
- 輸入包括多組測資，每一組測資的第一列是一個整數 N, 1≤N≤4, 接著有 N 列，每一列有 4 個以空白隔開的 0 或 1，一定是上述 6 種編碼序列之一

Sample Input
```
1
0 1 0 1
1
0 0 1 0
2
1 0 0 0
1 1 0 0
4
1 1 0 1
1 0 0 0
0 1 1 1
1 1 0 1
```

問題描述

- 任何文字或數字儲存在電腦中都是使用 0 與 1 構成的二元編碼序列，A~F 的編碼方式如下表：

字元	A	B	C	D	E	F
編碼	0 1 0 1	0 1 1 1	0 0 1 0	1 1 0 1	1 0 0 0	1 1 0 0

- 請寫一個程式辨識 0 1 編碼序列，輸出對應的字元
- **輸入**包括多組測資，每一組測資的第一列是一個整數 N, 1≤N≤4, 接著有 N 列，每一列有 4 個以空白隔開的 0 或 1，一定是上述 6 種編碼序列之一
- 對於每一組輸入測資，**輸出**一列輸入編碼序列所代表的 N 個字元，字元之間不需要空白。

Sample Input

```
1
0 1 0 1
1
0 0 1 0
2
1 0 0 0
1 1 0 0
4
1 1 0 1
1 0 0 0
0 1 1 1
1 1 0 1
```

Sample Output

```
A
C
EF
DEBD
```

問題描述

- 任何文字或數字儲存在電腦中都是使用 0 與 1 構成的二元編碼序列，A~F 的編碼方式如下表：

字元	A	B	C	D	E	F
編碼	0 1 0 1	0 1 1 1	0 0 1 0	1 1 0 1	1 0 0 0	1 1 0 0

- 請寫一個程式辨識 0 1 編碼序列，輸出對應的字元
- **輸入**包括多組測資，每一組測資的第一列是一個整數 N, 1≤N≤4, 接著有 N 列，每一列有 4 個以空白隔開的 0 或 1，一定是上述 6 種編碼序列之一
- 對於每一組輸入測資，**輸出**一列輸入編碼序列所代表的 N 個字元，字元之間不需要空白。
- 時間限制: 1 sec

Sample Input

```
1
0 1 0 1
1
0 0 1 0
2
1 0 0 0
1 1 0 0
4
1 1 0 1
1 0 0 0
0 1 1 1
1 1 0 1
```

Sample Output

```
A
C
EF
DEBD
```

問題分析與基本解法

Sample Input

```
1
0 1 0 1
1
0 0 1 0
2
1 0 0 0
1 1 0 0
```

Sample Output

```
A
C
EF
```

問題分析與基本解法

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C
  ◦ ◦ ◦

Sample Input
1
0 1 0 1
1
0 0 1 0
2
1 0 0 0
1 1 0 0

Sample Output
A
C
EF

問題分析與基本解法

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C
  。 。 。
- 簡化一點，如果輸入是 1 輸出 A
  輸入是 2 輸出 B
  。 。 。

Sample Input
1
0 1 0 1
1
0 0 1 0
2
1 0 0 0
1 1 0 0

Sample Output
A
C
EF

問題分析與基本解法

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C
  。 。 。

- 簡化一點，如果輸入是 1 輸出 A
  輸入是 2 輸出 B
  。 。 。

咦，題目還可以改喔！

Sample Input
```
1
0 1 0 1
1
0 0 1 0
2
1 0 0 0
1 1 0 0
```

Sample Output
```
A
C
EF
```

問題分析與基本解法

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C
  。 。 。
- 簡化一點，如果輸入是 1 輸出 A
  輸入是 2 輸出 B
  。 。 。

咦，題目還可以改喔！

對啊，山不轉路轉，先找到一個熟悉的運算模型再說

Sample Input

```
1
0 1 0 1
1
0 0 1 0
2
1 0 0 0
1 1 0 0
```

Sample Output

```
A
C
EF
```

問題分析與基本解法

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C
  。 。 。
- 簡化一點，如果輸入是 1 輸出 A
  輸入是 2 輸出 B
  。 。 。
- 此時看到一個資料轉換問題，基本作法是檢查與比對輸入，針對不同的輸入來輸出對應的結果

Sample Input

```
1
0 1 0 1
1
0 0 1 0
2
1 0 0 0
1 1 0 0
```

Sample Output

```
A
C
EF
```

問題分析與基本解法

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C
  ◦ ◦ ◦
- 簡化一點，如果輸入是 1 輸出 A
  輸入是 2 輸出 B
  ◦ ◦ ◦
- 此時看到一個資料轉換問題，基本作法是檢查與比對輸入，針對不同的輸入來輸出對應的結果

```
1 if (x==1) printf("A");
2 else if (x==2) printf("B");
3 else if …
```

Sample Input

```
1
0 1 0 1
1
0 0 1 0
2
1 0 0 0
1 1 0 0
```

Sample Output

```
A
C
EF
```

問題分析與基本解法

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C
  ◦ ◦ ◦
- 簡化一點，如果輸入是 1 輸出 A
  輸入是 2 輸出 B
  ◦ ◦ ◦
- 此時看到一個資料轉換問題，基本作法是檢查與比對輸入，針對不同的輸入來輸出對應的結果

```
1 if (x==1) printf("A");
2 else if (x==2) printf("B");
3 else if …
```

- 回到原來的問題，只是檢查的地方複雜一點，如果輸入資料存放在變數 d0, d1, d2, d3 裡面

Sample Input
1
0 1 0 1
1
0 0 1 0
2
1 0 0 0
1 1 0 0

Sample Output
A
C
EF

問題分析與基本解法

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C
  ◦ ◦ ◦
- 簡化一點，如果輸入是 1 輸出 A
  輸入是 2 輸出 B
  ◦ ◦ ◦
- 此時看到一個資料轉換問題，基本作法是檢查與比對輸入，針對不同的輸入來輸出對應的結果

```
1 if (x==1) printf("A");
2 else if (x==2) printf("B");
3 else if …
```

- 回到原來的問題，只是檢查的地方複雜一點，如果輸入資料存放在變數 d0, d1, d2, d3 裡面

```
1 if (d0==0&&d1==1&&d2==0&&d3==1) printf("A");
2 else if (d0==0&&d1==0&&d2==1&&d3==0) printf("B");
3 else if …
```

Sample Input

```
1
0 1 0 1
1
0 0 1 0
2
1 0 0 0
1 1 0 0
```

Sample Output

```
A
C
EF
```

變數設計與輸入

- 輸入是 1 輸出 A
  輸入是 2 輸出 B
  。 。 。

變數設計與輸入

- 輸入是 1 輸出 A
  輸入是 2 輸出 B
  。 。 。

```
1 int x;
2 scanf("%d", &x);
3 if (x==1) printf("A");
4 else if (x==2) printf("B");
5 else if …
```

變數設計與輸入

- 輸入是 1 輸出 A
  輸入是 2 輸出 B
  ∘ ∘ ∘
- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C
  ∘ ∘ ∘

```
1 int x;
2 scanf("%d", &x);
3 if (x==1) printf("A");
4 else if (x==2) printf("B");
5 else if …
```

Sample Input
1
0 1 0 1
1
0 0 1 0
2
1 0 0 0
1 1 0 0

Sample Output
A
C
EF

變數設計與輸入

- 輸入是 1 輸出 A
  輸入是 2 輸出 B
  。 。 。
- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C
  。 。 。

```
1 int x;
2 scanf("%d", &x);
3 if (x==1) printf("A");
4 else if (x==2) printf("B");
5 else if …
```

```
1 int d0, d1, d2, d3;
2 scanf("%d%d%d%d", &d0, &d1, &d2, &d3);
3 if (d0==0&&d1==1&&d2==0&&d3==1) printf("A");
4 else if (d0==0&&d1==0&&d2==1&&d3==0) printf("B");
5 else if …
```

Sample Input

```
1
0 1 0 1
1
0 0 1 0
2
1 0 0 0
1 1 0 0
```

Sample Output

```
A
C
EF
```

變數設計與輸入

- 輸入是 1 輸出 A
  輸入是 2 輸出 B
  ◦ ◦ ◦
- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C
  ◦ ◦ ◦

```
1 int x;
2 scanf("%d", &x);
3 if (x==1) printf("A");
4 else if (x==2) printf("B");
5 else if …
```

```
1 int d0, d1, d2, d3;
2 scanf("%d%d%d%d", &d0, &d1, &d2, &d3);
3 if (d0==0&&d1==1&&d2==0&&d3==1) printf("A");
4 else if (d0==0&&d1==0&&d2==1&&d3==0) printf("B");
5 else if …
```

```
1 char d0, d1, d2, d3;
2 scanf("□%c□%c□%c□%c", &d0, &d1, &d2, &d3);
3 if (d0=='0'&&d1=='1'&&d2=='0'&&d3=='1') printf("A");
4 else if (d0=='0'&&d1=='0'&&d2=='1'&&d3=='0') printf("B");
5 else if …
```

Sample Input

```
1
0 1 0 1
1
0 0 1 0
2
1 0 0 0
1 1 0 0
```

Sample Output

```
A
C
EF
```

多重 if ⇒ switch

- 輸入是 1 輸出 A
  輸入是 2 輸出 B
  。 。 。

多重 if ⇒ switch

- 輸入是 1 輸出 A
  輸入是 2 輸出 B
  。 。 。

```
1 if (x==1) printf("A");
2 else if (x==2) printf("B");
3 else if …
```

多重 if ⇒ switch

- 輸入是 1 輸出 A
  輸入是 2 輸出 B
  。 。 。

```
1 if (x==1) printf("A");
2 else if (x==2) printf("B");
3 else if …
```

```
1 switch (x) {
2 case 1: printf("A"); break;
3 case 2: printf("B"); break;
    …
4 }
```

多重 if ⇒ switch

- 輸入是 1 輸出 A
  輸入是 2 輸出 B
  。 。 。

```
1 if (x==1) printf("A");
2 else if (x==2) printf("B");
3 else if …
```

```
1 switch (x) {
2 case 1: printf("A"); break;
3 case 2: printf("B"); break;
   …
4 }
```

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C
  。 。 。

```
1 if (d0==0&&d1==1&&d2==0&&d3==1) printf("A");
2 else if (d0==0&&d1==0&&d2==1&&d3==0) printf("B");
3 else if …
```

多重 if ⇒ switch

- 輸入是 1 輸出 A
  輸入是 2 輸出 B
  。 。 。

```
1 if (x==1) printf("A");
2 else if (x==2) printf("B");
3 else if …
```

```
1 switch (x) {
2 case 1: printf("A"); break;
3 case 2: printf("B"); break;
   …
4 }
```

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C
  。 。 。

```
1 if (d0==0&&d1==1&&d2==0&&d3==1) printf("A");
2 else if (d0==0&&d1==0&&d2==1&&d3==0) printf("B");
3 else if …
```

```
1 switch (????) {
2 case ?: printf("A"); break;
3 case ?: printf("B"); break;
   …
4 }
```

多重 if ⇒ switch

- 輸入是 1 輸出 A
  輸入是 2 輸出 B
  。 。 。

```
1 if (x==1) printf("A");
2 else if (x==2) printf("B");
3 else if …
```

```
1 switch (x) {
2 case 1: printf("A"); break;
3 case 2: printf("B"); break;
    …
4 }
```

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C
  。 。 。

```
1 if (d0==0&&d1==1&&d2==0&&d3==1) printf("A");
2 else if (d0==0&&d1==0&&d2==1&&d3==0) printf("B");
3 else if …
```

```
1 switch (????) {
2 case ?: printf("A"); break;
3 case ?: printf("B"); break;
    …
4 }
```

1. 編碼 0101→5 0010→2

多重 if ⇒ switch

- 輸入是 1 輸出 A
  輸入是 2 輸出 B
  。 。 。

```
1 if (x==1) printf("A");
2 else if (x==2) printf("B");
3 else if …
```

```
1 switch (x) {
2 case 1: printf("A"); break;
3 case 2: printf("B"); break;
   …
4 }
```

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C
  。 。 。

```
1 if (d0==0&&d1==1&&d2==0&&d3==1) printf("A");
2 else if (d0==0&&d1==0&&d2==1&&d3==0) printf("B");
3 else if …
```

```
1 switch (????) {
2 case ?: printf("A"); break;
3 case ?: printf("B"); break;
   …
4 }
```

1. 編碼 0101→5 0010→2
2. 函式

```
1 int enc(int d0, int d1, int d2, int d3) {
2    return (((d0*2)+d1)*2+d2)*2+d3;
3 }
```

多重 if ⇒ switch

- 輸入是 1 輸出 A
  輸入是 2 輸出 B
  。 。 。

```
1 if (x==1) printf("A");
2 else if (x==2) printf("B");
3 else if …
```

```
1 switch (x) {
2 case 1: printf("A"); break;
3 case 2: printf("B"); break;
   …
4 }
```

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C
  。 。 。

```
1 if (d0==0&&d1==1&&d2==0&&d3==1) printf("A");
2 else if (d0==0&&d1==0&&d2==1&&d3==0) printf("B");
3 else if …
```

```
1 switch (????) {
2 case ?: printf("A"); break;
3 case ?: printf("B"); break;
   …
4 }
```

1. 編碼 0101→5 0010→2
2. 函式

```
1 int enc(int d0, int d1, int d2, int d3) {
2    return (((d0*2)+d1)*2+d2)*2+d3;
3 }
```

$d_0 * 2^3 + d_1 * 2^2 + d_2 * 2 + d_3$

多重 if ⇒ switch

- 輸入是 1 輸出 A
  輸入是 2 輸出 B
  。 。 。

```
1 if (x==1) printf("A");
2 else if (x==2) printf("B");
3 else if …
```

```
1 switch (x) {
2 case 1: printf("A"); break;
3 case 2: printf("B"); break;
   …
4 }
```

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C
  。 。 。

```
1 if (d0==0&&d1==1&&d2==0&&d3==1) printf("A");
2 else if (d0==0&&d1==0&&d2==1&&d3==0) printf("B");
3 else if …
```

```
1 switch (????) {
2 case ?: printf("A"); break;
3 case ?: printf("B"); break;
   …
4 }
```

1. 編碼 0101→5 0010→2
2. 函式
```
1 int enc(int d0, int d1, int d2, int d3) {
2   return (((d0*2)+d1)*2+d2)*2+d3;
3 }
```
$d_0 * 2^3 + d_1 * 2^2 + d_2 * 2 + d_3$

運用
```
switch(enc(d0,d1,d2,d3))
case 5: …
```

這個設計的根本問題

- 輸入是 1 輸出 A
  輸入是 2 輸出 B

這個設計的根本問題

- 輸入是 1 輸出 A
  輸入是 2 輸出 B

。 。 。

```
1 int x;
2 scanf("%d", &x);
3 if (x==1) printf("A");
4 else if (x==2)
5     printf("B");
6 else if …
7 else if …
```

這個設計的根本問題

- 輸入是 1 輸出 A
  輸入是 2 輸出 B

。 。 。

```
1 int x;
2 scanf("%d", &x);
3 if (x==1) printf("A");
4 else if (x==2)
5     printf("B");
6 else if …
7 else if …
```

程式不斷複製而變得很長

這個設計的根本問題

- 輸入是 1 輸出 A
  輸入是 2 輸出 B

  ∘ ∘ ∘

```
1 int x;
2 scanf("%d", &x);
3 if (x==1) printf("A");
4 else if (x==2)
5     printf("B");
6 else if …
7 else if …
```

程式不斷複製而變得很長

公式計算

```
printf("%c", 'A'+x-1);
```

這個設計的根本問題

- 輸入是 1 輸出 A
  輸入是 2 輸出 B

。 。 。

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C

。 。 。

```
1 int x;
2 scanf("%d", &x);
3 if (x==1) printf("A");
4 else if (x==2)
5     printf("B");
6 else if …
7 else if …
```

程式不斷複製而變得很長

公式計算

```
printf("%c", 'A'+x-1);
```

這個設計的根本問題

- 輸入是 1 輸出 A
  輸入是 2 輸出 B

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C

```
1 int x;
2 scanf("%d", &x);
3 if (x==1) printf("A");
4 else if (x==2)
5     printf("B");
6 else if …
7 else if …
```

程式不斷複製而變得很長

公式計算

```
printf("%c", 'A'+x-1);
```

```
1 int d0, d1, d2, d3;
2 scanf("%d%d%d%d", &d0, &d1, &d2, &d3);
3 if (d0==0&&d1==1&&d2==0&&d3==1)
4     printf("A");
5 else if (d0==0&&d1==0&&d2==1&&d3==0)
6     printf("B");
7 else if …
8 else if …
```

這個設計的根本問題

- 輸入是 1 輸出 A
  輸入是 2 輸出 B

。 。 。

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C

。 。 。

```
1 int x;
2 scanf("%d", &x);
3 if (x==1) printf("A");
4 else if (x==2)
5     printf("B");
6 else if …
7 else if …
```

程式不斷複製而變得很長

公式計算

```
printf("%c", 'A'+x-1);
```

```
1 int d0, d1, d2, d3;
2 scanf("%d%d%d%d", &d0, &d1, &d2, &d3);
3 if (d0==0&&d1==1&&d2==0&&d3==1)
4     printf("A");
5 else if (d0==0&&d1==0&&d2==1&&d3==0)
6     printf("B");
7 else if …
8 else if …
```

字元	A	B	C	D	E	F
編碼	0101	0111	0010	1101	1000	1100

這個設計的根本問題

- 輸入是 1 輸出 A
  輸入是 2 輸出 B

。 。 。

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C

。 。 。

```
1 int x;
2 scanf("%d", &x);
3 if (x==1) printf("A");
4 else if (x==2)
5     printf("B");
6 else if …
7 else if …
```

程式不斷複製而變得很長

公式計算

```
printf("%c", 'A'+x-1);
```

```
1 int d0, d1, d2, d3;
2 scanf("%d%d%d%d", &d0, &d1, &d2, &d3);
3 if (d0==0&&d1==1&&d2==0&&d3==1)
4     printf("A");
5 else if (d0==0&&d1==0&&d2==1&&d3==0)
6     printf("B");
7 else if …
8 else if …
```

除了用 **if / switch** 對照或是公式計算 之外，還有一種就是 – 陣列表列

字元	A	B	C	D	E	F
編碼	0101	0111	0010	1101	1000	1100

這個設計的根本問題

- 輸入是 1 輸出 A
  輸入是 2 輸出 B

。 。 。

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C

。 。 。

```
1 int x;
2 scanf("%d", &x);
3 if (x==1) printf("A");
4 else if (x==2)
5     printf("B");
6 else if …
7 else if …
```

程式不斷複製而變得很長

公式計算

```
printf("%c", 'A'+x-1);
```

```
1 int d0, d1, d2, d3;
2 scanf("%d%d%d%d", &d0, &d1, &d2, &d3);
3 if (d0==0&&d1==1&&d2==0&&d3==1)
4     printf("A");
5 else if (d0==0&&d1==0&&d2==1&&d3==0)
6     printf("B");
7 else if …
8 else if …
```

除了用 **if / switch** 對照或是公式計算之外，還有一種就是 – 陣列表列

字元	A	B	C	D	E	F
編碼	0101	0111	0010	1101	1000	1100
d	**5**	**7**	**2**	**13**	**8**	**12**

這個設計的根本問題

- 輸入是 1 輸出 A
  輸入是 2 輸出 B

。 。 。

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C

。 。 。

```
1 int x;
2 scanf("%d", &x);
3 if (x==1) printf("A");
4 else if (x==2)
5     printf("B");
6 else if …
7 else if …
```

程式不斷複製而變得很長

公式計算

```
printf("%c", 'A'+x-1);
```

```
1 int d0, d1, d2, d3;
2 scanf("%d%d%d%d", &d0, &d1, &d2, &d3);
3 if (d0==0&&d1==1&&d2==0&&d3==1)
4     printf("A");
5 else if (d0==0&&d1==0&&d2==1&&d3==0)
6     printf("B");
7 else if …
8 else if …
```

除了用 **if / switch** 對照或是公式計算 之外，還有一種就是 – 陣列表列

字元	A	B	C	D	E	F
編碼	0101	0111	0010	1101	1000	1100
d	**5**	**7**	**2**	**13**	**8**	**12**

tab

0	1	2	3	4	5	6	7	8	9	10	11	12	13
?	?	C	?	?	A	?	B	E	?	?	?	F	D

這個設計的根本問題

- 輸入是 1 輸出 A
  輸入是 2 輸出 B

。 。 。

- 輸入 0 1 0 1 ⇒ 輸出 A
  輸入 0 0 1 0 ⇒ 輸出 C

。 。 。

```
1 int x;
2 scanf("%d", &x);
3 if (x==1) printf("A");
4 else if (x==2)
5     printf("B");
6 else if …
7 else if …
```

程式不斷複製而變得很長

公式計算

```
printf("%c", 'A'+x-1);
```

除了用 **if / switch** 對照或是公式計算 之外，還有一種就是 – 陣列表列

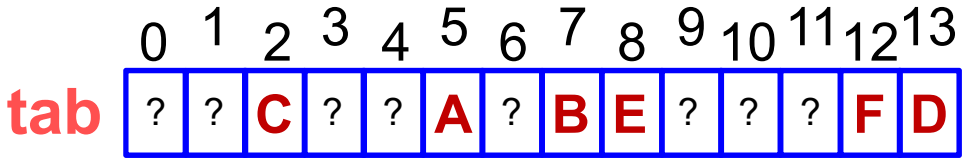
```
1 int d0, d1, d2, d3;
2 scanf("%d%d%d%d", &d0, &d1, &d2, &d3);
3 if (d0==0&&d1==1&&d2==0&&d3==1)
4     printf("A");
5 else if (d0==0&&d1==0&&d2==1&&d3==0)
6     printf("B");
7 else if …
8 else if …
```

```
1 int d, tab[]={0,0,'C',0,0,'A',0,
2                'B','E',0,0,0,'F','D'};
3 d = ((d0*2+d1)*2+d2)*2+d3;
4 printf("%c", tab[d]);
```

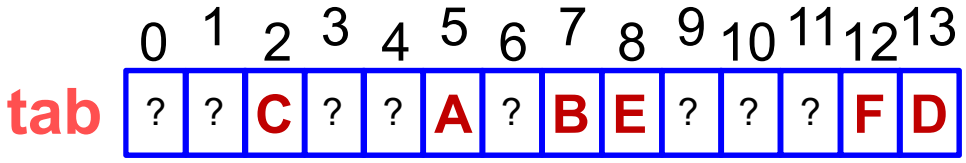
字元	A	B	C	D	E	F
編碼	0101	0111	0010	1101	1000	1100
d	**5**	**7**	**2**	**13**	**8**	**12**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
tab	?	?	C	?	?	A	?	B	E	?	?	?	F	D

建立對照表



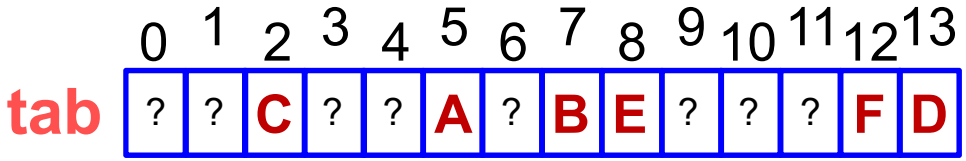
建立對照表



- 以陣列建立對照表的前提是這些編碼要相當密集，也就是上圖中的？不能太多，否則就太浪費記憶體空間了，基本方法如下：

```
int tab[]={0,0,'C',0,0,'A',0,'B','E',0,0,0,'F','D'};
```

建立對照表



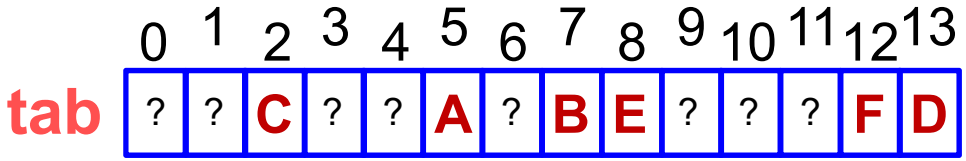
- 以陣列建立對照表的前提是這些編碼要相當密集，也就是上圖中的？不能太多，否則就太浪費記憶體空間了，基本方法如下：

```
int tab[]={0,0,'C',0,0,'A',0,'B','E',0,0,0,'F','D'};
```

- 如果自己由鍵盤打一次，就會發現上面這個初始化陣列的敘述還蠻繁瑣的，可以考慮下面的寫法

```
1 int tab[14]={0};
2 tab[2]='C';
3 tab[5]='A';
4 tab[7]='B';
5 tab[8]='E';
6 tab[12]='F';
7 tab[13]='D';
```

建立對照表



- 以陣列建立對照表的前提是這些編碼要相當密集，也就是上圖中的？不能太多，否則就太浪費記憶體空間了，基本方法如下：

```
int tab[]={0,0,'C',0,0,'A',0,'B','E',0,0,0,'F','D'};
```

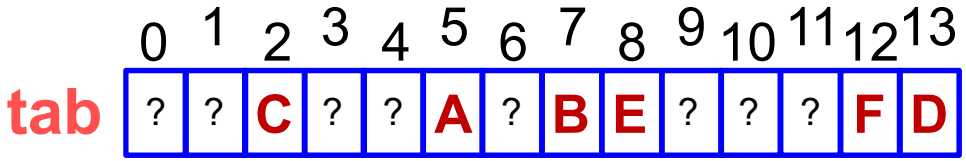
- 如果自己由鍵盤打一次，就會發現上面這個初始化陣列的敘述還蠻繁瑣的，可以考慮下面的寫法

```
1 int tab[14]={0};
2 tab[2]='C';
3 tab[5]='A';
4 tab[7]='B';
5 tab[8]='E';
6 tab[12]='F';
7 tab[13]='D';
```

- 另外一種特別簡潔的設定方法，可以改用字串常數來初始化字元陣列

```
char tab[]="??C??A?BE???FD";
```

建立對照表



- 以陣列建立對照表的前提是這些編碼要相當密集，也就是上圖中的？不能太多，否則就太浪費記憶體空間了，基本方法如下：

```
int tab[]={0,0,'C',0,0,'A',0,'B','E',0,0,0,'F','D'};
```

- 如果自己由鍵盤打一次，就會發現上面這個初始化陣列的敘述還蠻繁瑣的，可以考慮下面的寫法

```
1 int tab[14]={0};
2 tab[2]='C';
3 tab[5]='A';
4 tab[7]='B';
5 tab[8]='E';
6 tab[12]='F';
7 tab[13]='D';
```

- 另外一種特別簡潔的設定方法，可以改用字串常數來初始化字元陣列

```
char tab[]="??C??A?BE???FD";
```

- 最後，如果對照表裡面的資料非常稀疏該怎麼辦? (例如有100個字母散佈在0~10000間)

稀疏的對照表

- 5/**14**

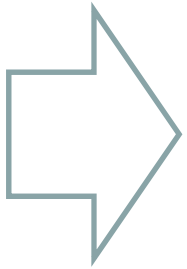
```
1 int tab[14]={0};
2 tab[2]='C';
3 tab[5]='A';
4 tab[7]='B';
5 tab[8]='E';
6 tab[12]='F';
7 tab[13]='E';
```

稀疏的對照表

- 5/**14**

```
1 int tab[14]={0};
2 tab[2]='C';
3 tab[5]='A';
4 tab[7]='B';
5 tab[8]='E';
6 tab[12]='F';
7 tab[13]='E';
```

- 5/**20000**



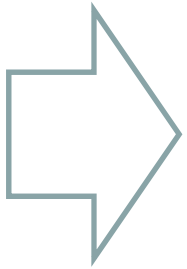
```
1 int tab[20000]={0};
2 tab[227]='C';
3 tab[5310]='A';
4 tab[7980]='B';
5 tab[8773]='E';
6 tab[12145]='F';
7 tab[14930]='E';
```

稀疏的對照表

- 5/**14**

```
1 int tab[14]={0};
2 tab[2]='C';
3 tab[5]='A';
4 tab[7]='B';
5 tab[8]='E';
6 tab[12]='F';
7 tab[13]='E';
```

- 5/**20000**



```
1 int tab[20000]={0};
2 tab[227]='C';
3 tab[5310]='A';
4 tab[7980]='B';
5 tab[8773]='E';
6 tab[12145]='F';
7 tab[14930]='E';
```

- 平行陣列

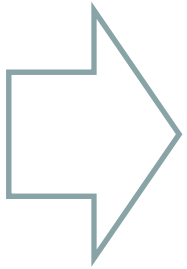
```
1 int idx[]={227,5310,7980,…,8773,12145,14930};
2 int tab[]= {'C','A','B',…,'E','F','E'};
```

稀疏的對照表

- 5/**14**

```
1 int tab[14]={0};
2 tab[2]='C';
3 tab[5]='A';
4 tab[7]='B';
5 tab[8]='E';
6 tab[12]='F';
7 tab[13]='E';
```

- 5/**20000**



```
1 int tab[20000]={0};
2 tab[227]='C';
3 tab[5310]='A';
4 tab[7980]='B';
5 tab[8773]='E';
6 tab[12145]='F';
7 tab[14930]='E';
```

- 平行陣列

```
1 int idx[]={227,5310,7980,…,8773,12145,14930};
2 int tab[]= {'C','A','B',…,'E','F','E'};
```

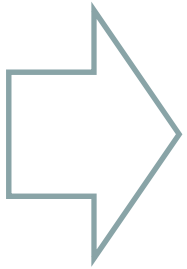
7980 $\xrightarrow{?}$ B

稀疏的對照表

- 5/**14**

```
1 int tab[14]={0};
2 tab[2]='C';
3 tab[5]='A';
4 tab[7]='B';
5 tab[8]='E';
6 tab[12]='F';
7 tab[13]='E';
```

- 5/**20000**



```
1 int tab[20000]={0};
2 tab[227]='C';
3 tab[5310]='A';
4 tab[7980]='B';
5 tab[8773]='E';
6 tab[12145]='F';
7 tab[14930]='E';
```

- 平行陣列

```
1 int idx[]={227,5310,7980,…,8773,12145,14930};
2 int tab[]= {'C','A','B',…,'E','F','E'};
```

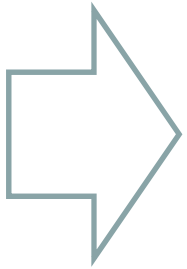
7980 $\xrightarrow{?}$ B　　12145 $\xrightarrow{?}$ F

稀疏的對照表

- 5/**14**

```
1 int tab[14]={0};
2 tab[2]='C';
3 tab[5]='A';
4 tab[7]='B';
5 tab[8]='E';
6 tab[12]='F';
7 tab[13]='E';
```

- 5/**20000**



```
1 int tab[20000]={0};
2 tab[227]='C';
3 tab[5310]='A';
4 tab[7980]='B';
5 tab[8773]='E';
6 tab[12145]='F';
7 tab[14930]='E';
```

- 平行陣列

```
1 int idx[]={227,5310,7980,…,8773,12145,14930};
2 int tab[]= {'C','A','B',…,'E','F','E'};
```

7980 $\xrightarrow{?}$ B　　12145 $\xrightarrow{?}$ F

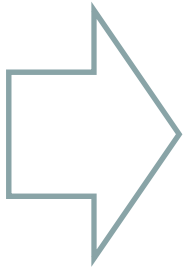
資料轉換：以迴圈比對 idx 陣列, 找到 7980 存放在 idx[2], 對應的 tab[2] 就是 B

稀疏的對照表

- 5/**14**

```
1 int tab[14]={0};
2 tab[2]='C';
3 tab[5]='A';
4 tab[7]='B';
5 tab[8]='E';
6 tab[12]='F';
7 tab[13]='E';
```

- 5/**20000**



```
1 int tab[20000]={0};
2 tab[227]='C';
3 tab[5310]='A';
4 tab[7980]='B';
5 tab[8773]='E';
6 tab[12145]='F';
7 tab[14930]='E';
```

- 平行陣列

```
1 int idx[]={227,5310,7980,…,8773,12145,14930};
2 int tab[]= {'C','A','B',…,'E','F','E'};
```

7980 $\xrightarrow{?}$ B　　12145 $\xrightarrow{?}$ F

資料轉換：以迴圈比對 idx 陣列, 找到 7980 存放在 idx[2], 對應的 tab[2] 就是 B

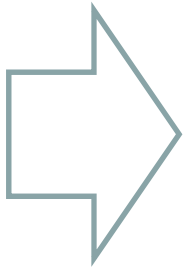
```
1 j=0; while (j<m&&idx[j]!=d) j++;
2 printf("%c", tab[j]);
```

稀疏的對照表

- 5/**14**

```
1 int tab[14]={0};
2 tab[2]='C';
3 tab[5]='A';
4 tab[7]='B';
5 tab[8]='E';
6 tab[12]='F';
7 tab[13]='E';
```

- 5/**20000**



```
1 int tab[20000]={0};
2 tab[227]='C';
3 tab[5310]='A';
4 tab[7980]='B';
5 tab[8773]='E';
6 tab[12145]='F';
7 tab[14930]='E';
```

- 平行陣列

```
1 int idx[]={227,5310,7980,…,8773,12145,14930};
2 int tab[]= {'C','A','B',…,'E','F','E'};
```

7980 $\xrightarrow{?}$ B　　12145 $\xrightarrow{?}$ F

資料轉換：以迴圈比對 idx 陣列, 找到 7980 存放在 idx[2], 對應的 tab[2] 就是 B

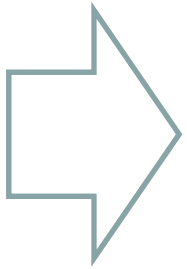
```
1 j=0; while (j<m&&idx[j]!=d) j++;
2 printf("%c", tab[j]);
```

- 結構陣列

稀疏的對照表

- 5/**14**

```
1 int tab[14]={0};
2 tab[2]='C';
3 tab[5]='A';
4 tab[7]='B';
5 tab[8]='E';
6 tab[12]='F';
7 tab[13]='E';
```



- 5/**20000**

```
1 int tab[20000]={0};
2 tab[227]='C';
3 tab[5310]='A';
4 tab[7980]='B';
5 tab[8773]='E';
6 tab[12145]='F';
7 tab[14930]='E';
```

- 平行陣列

```
1 int idx[]={227,5310,7980,…,8773,12145,14930};
2 int tab[]= {'C','A','B',…,'E','F','E'};
```

7980 $\xrightarrow{?}$ B　　12145 $\xrightarrow{?}$ F

資料轉換：以迴圈比對 idx 陣列, 找到 7980 存放在 idx[2], 對應的 tab[2] 就是 B

```
1 j=0; while (j<m&&idx[j]!=d) j++;
2 printf("%c", tab[j]);
```

- 結構陣列

```
1 typedef struct { int idx, ch; } data_t;
2 data_t tab[] = {{227, 'C'}, {5310,'A'}, {7980,'B'}, …,
3                 {8773,'E'},{12145,'F'},{14930,'E'}};
```