Josephus Problem

參考資料:

http://en.wikipedia.org/wiki/Josephus_problem

http://mathworld.wolfram.com/JosephusProblem.html

“THE ART OF COMPUTER PROGRAMMING” DONALD E. KNUTH

主題: 演算法 Algorithms、資料結構 Data Structures

有 n 個人排成一圈，從個位置開始數，往前每數到第 m 個人，就將那個人從圈子中移除。將被移除的順位列出，就構成所謂的 Josephus Permutation。以 n=8, m=4 為例，假設從編號 1 的人開始數，則最早被移除的會是編號 4 那個人。請看底下的過程，編號加底線表示從那個人開始數，紅色粗體的編號代表要被移除的人：

第 1 次:

1 2 3 4 5 6 7 8 → 1 2 3 **4** 5 6 7 8 然後移除變成 1 2 3 5 6 7 8 所以下次從編號 5 開始數

第 2 次:

1 2 3 5 6 7 8 → 1 2 3 5 6 7 **8** 然後移除變成 1 2 3 5 6 7 下次從編號 1 開始 (因為繞成一圈)

這樣依序做下去

第 3 次: 1 2 3 5 6 7 → 1 2 3 **5** 6 7

第 4 次: 1 2 3 6 7 → 1 **2** 3 6 7

第 5 次: 1 3 6 7 → **1** 3 6 7

第 6 次: 3 6 7 → **3** 6 7

第 7 次: 6 7 → 6 **7**

第 8 次: 6 → **6**

所以編號 1 到 8 的人被移除的順位是 5 4 6 1 3 8 7 2，這就是 n=8, m=4 的 Josephus Permutation，我們可以寫成 JP(8, 4) = [5 4 6 1 3 8 7 2]，它的意思是編號 1 的人會是第五順位被移除，編號 2 的人會在第四順位被移除，而編號 6 的人會是第八順位，也就是最後一個存留下的人。另外我們也可以列出 Inverse Josephus Permutation，以上面 n=8, m=4 的例子會是 IJP(8, 4) = [4 8 5 2 1 3 7 6] 也就是依照被移除的順序將編號列出。

接下來我們就來寫程式，輸入 n 和 m，然後讓程式自動列出 Inverse Josephus Permutation。開始寫程式之前，我們要先想好 (1) 如何用程式碼表達問題所描述的資料形式 (2) 如何用程式碼處理資料。這兩個問題牽涉到 資料結構 (Data Structures) 和 演算法 (Algorithms) 的設計。我們將用幾個範例程式來介紹不同的設計方式。

Method 1:

演算法採用模擬的方式，資料結構則採用 Array。

做法是先產生一個陣列，陣列的元素是每個人的編號。接下來就逐步模擬移除的過程，移除了某個人之後，就將後面的陣列元素往前挪。

```
// Method 1 for Inverse Josephus Permutation
#include <stdio.h>
#define N 41
#define M 3
int a[N];
int main(void)
{
    int CP, SS, i;

    CP = 0;     // CP 用來記住從哪個位置開始數
    SS = N;     // SS 用來記錄目前殘餘的人數， 一開始有 N 個人

    for (i = 0; i < N; i++) a[i] = i + 1;  // 設定編號
    while (SS > 0) {
        CP += M-1;    // 算出下一個要被移除的人的位置
        if (CP >= SS) CP = CP % SS;  // 因為繞成一圈， 算出循環後的對應位置
        printf(" %d", a[CP]);
        for (i = CP; i < SS-1; i++) {  // 往前挪一格
            a[i] = a[i+1];
        }
        SS--;
    }

    return 0;
}
```

範例程式的執行結果會是

```
3 6 9 12 15 18 21 24 27 30 33 36 39 1 5 10 14 19 23 28 32 37 41 7 13 20 26 34 40 8 17 29 38 11
25 2 22 4 35 16 31
```

Method 2:

和 Method 1 一樣，演算法也是採用模擬的方式，資料結構採用 Array。

和 Method 1 不同的地方在於，移除了某個人之後，只將原本的陣列元素標記為 -1，不移動其他陣列元素。

```
//Method 2
#include <stdio.h>
#define N 41
#define M 3
int a[N];
int main(void)
{
    int CP, SS, i;

    CP = -1;   // CP 用來記住開始數的位置
    SS = N;    // SS 用來記錄目前殘餘的人數， 一開始有 N 個人

    for (i = 0; i < N; i++) a[i] = i + 1;    // 設定每個人的編號

    while (SS > 0) {
        i = 0;  // 用 i 來數到 M
        do {
            do {
                CP = (CP + 1) % N; // 持續移到下一個位置 直到找到還留著的人
            } while (a[CP] == -1); // 才會結束迴圈
            i++;                   // 找到了還留下的人 所以 i++
        } while (i < (M-1)%SS+1);  // 當 i 數到 M 就會結束迴圈
        printf(" %d", a[CP]);  // 把要移除的人的編號輸出
        a[CP] = -1;            // 移除之後要標記為 -1
        SS--;
    }

    return 0;
}
```

Method 3:

演算法也是採用模擬的方式，資料結構採用 Circular Linked List。

需要自定下面的資料結構

```
typedef struct  _node {
    int id;
    struct _node *next;
} Node;
```

Circular Linked List 由上面的 Node 構成，每個 Node 包含兩個欄位，id 是用來記住編號，next 則是指向下一個 Node。搭配 Circular Linked List 我們需要寫兩個函數，分別是用來 insert 和 delete，讓 Linked List 可以動態改變內容。

```
void insertNext(Node* p, int id)
{
    Node *q = (Node *) malloc(sizeof(Node));  // 產生一個新的 Node
    q->id = id;          // 設定新的 Node 的 id
    q->next = p->next;  // 本來是 p --> r  變成 p, q --> r (也就是 p, q 後面都接著 r)
    p->next = q;         // 然後最後變成   p --> q --> r
}
Node* deleteNext(Node* p)
{
    Node *q;
    if (p->next == p) { // 如果只剩下一個 Node
        free(p);         // 就直接將它刪除
        p = NULL;
    } else {
        q = p->next;        // 如果原本是 p --> q --> r
        p->next = q->next; // 就先改成  p, q --> r
        free(q);            // 然後把 q 刪掉之後就變成  p --> r
    }
    return p;
}
```

主程式如下

```
int main(void)
{
    int i, SS;
    Node *last, *CP;
    last = (Node *) malloc(sizeof(Node)); // 先產生最後一個 Node
    last->id = N;               // 設定 id
    last->next = last;          // 自己指向自己
    for (i=N-1; i>0; i--) {     // 逐一把 Node 加入 Circular Linked List
        insertNext(last, i);    // N --> N 變成 N --> N-1 --> N  變成  N --> N-2 --> N-1 --> N
    }
    CP = last; SS = N;
    while (SS > 0) {
        for (i=0; i<(M-1)%SS; i++) {  // 往後找 M-1 個
            CP = CP->next;
        }
        printf(" %d", CP->next->id);  // 下一個人要被移除
        CP = deleteNext(CP);          // 原本是 N --> ... --> CP --> X --> Y --> ... --> N
                                      // 移除後 N --> ... --> CP --> Y --> ... --> N
        SS--;
    }
    return 0;
}
```

Method 4:

演算法也是採用模擬的方式，資料結構採用 Array 加上 Node struct。

需要自定下面的資料結構，要特別注意的是其中 next 欄位的型別是 int。

```
typedef struct  _node {
    int id;
    int next;
} Node;
```

主要的想法是利用 Array 模擬 Circular Linked List，自己維持 next 的連接順序。從這個例子也可以看出 Array 與 指標之間相通之處。此外，使用 Array 模仿 Linked List 與真正的 Linked List 還有一個不同的地方：Linked List 可以動態產生和移除 Node，如果是用 Array 模仿則會占用固定大小的空間，沒用到的 Node 其實還在原位，沒有真的被移除掉，只是靠 next 跳過而已。

主程式如下

```
int main(void)
{
    Node circle[N];   // 固定大小的 Array
    int i, CP;

    for (i=0; i<N; i++) {          // 設定每個人的編號 以及 位置順序
        circle[i].id = i+1;
        circle[i].next = (i+1)%N;  // 一開始是按照編號順序排列  因為 %N 所以會指回第一個人
    }
    CP = N-1;  // CP 的值會是 0 到 N-1， 0 代表第一個人  N-1 是最後一個人
    while (CP != circle[CP].next) { // 迴圈做到還剩下一個人為止
        for (i=0; i<(M-1); i++) {   // 數到 M-1 就停下來，下一個就是要被移除的人
            CP = circle[CP].next;   // 不斷往後數
        }
        printf(" %d", circle[circle[CP].next].id);       // 印出要被移除的人的編號
        circle[CP].next = circle[circle[CP].next].next;  // 所謂的移除其實是改變 next 所指向的人
    }
    printf(" %d", circle[CP].id);  // 把最後存留下來的那個人的編號印出來
    return 0;
}
```

Method 5:

最後來看遞迴的做法。底下的寫法只能列出最後一個留下來的人，不能列出 Inverse Josephus Permutation。

程式如下

```
#include <stdio.h>
int F(int n, int m)
{
    if (n==1) {
        return 1;
    } else {
        return (F(n-1, m)+(m-1)) % n + 1;
    }
}
int main(void)
{
    printf("%d\n", F(41, 3));
    return 0;
}
```

上面的函數 F(n, m) 傳回的值是最後能夠存留下來的人的編號，也就是如果總共 n 個人， 每 m 個移除掉一人，最後能夠剩下的人他的編號會是 F(n, m)。同樣地， F(n-1, m) 則是總共 n-1 個人、每 m 個移除掉一人，最後留下的人的編號。

遞迴的關鍵在於找出 F(n, m) 和 F(n-1, m) 之間的關聯。

原本有 n 個人，移除掉一個之後，剩下 n-1 個人。假如我們有辦法知道 F(n-1, m) 是多少，那麼我們該如何推算出 F(n, m)？ 如果我們知道 F(n-1, m)， 也就是最後留下的那個人，我們就能換算出他在原本的 n 個人的相對位置。

以 F(41, 3) 為例，如果我們知道 F(40, 3) 的值是 28，對應到原本 41 人的情況中，F(40, 3) 其實是從原本的編號 4 的人開始數，因為原本 41 人如果是從編號 1 開始數，一開始第一個被移除的人編號應該是 3，所以接下來剩下的 40 個人中的第一個人 (新的編號 1)，在原本 41 人中的編號是 4。這樣我們就可以看得出新舊編號的對應關係 (1→4, 2→5, ⋯, 28→31)，所以如果 F(40, 3) 是 28，則 F(41,3) 應該是 31。

寫成遞迴關係式

F(n, m) = (F(n-1, m) + m – 1) % n + 1

F(1, m) = 1