程式設計(一)-HW04

Due to 12/1 PM 11:59／授課老師：紀博文

一、基本資料

姓名：林育辰
系級：資工 111
學號：40771131H

二、檔案有哪些？

1) hw0401-1.c
2) hw0401-2.c
3) hw0401_header.c
4) hw0401_header.h
5) hw0402.c
6) hw0403.c
7) hw0404.c
8) hw0405.c
9) hw0405_header.c
10) hw0405_header.h
11) hw0406.c
12) Makefile
13) README.pdf

◎每個.c 檔皆有詳細註解！問題的回答寫在 README 每題詳細說明中！

三、如何執行？

請輸入 make→編譯 hw0401.c~hw0406.c→產生 hw0401~hw0406 檔
指令如下：

```
$ make
$ ./hw0401
$ ./hw0402
…
```

以此類推，即可分別執行 hw0401~hw0406

四、索引

第一題——P.2-5
第二題——**P.5-8**
第三題——P.8-14
第四題——**P.14-18**
第五題——P.18-21
第六題——**P.22-24**

說明

1 Tower of Hanoi (20 pts)

The Tower of Hanoi is a mathematical game or puzzle. It consists of three rods and a number of disks of different sizes, which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape. Figure 1 is an example with 8 disks.

The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
- No larger disk may be placed on top of a smaller disk.

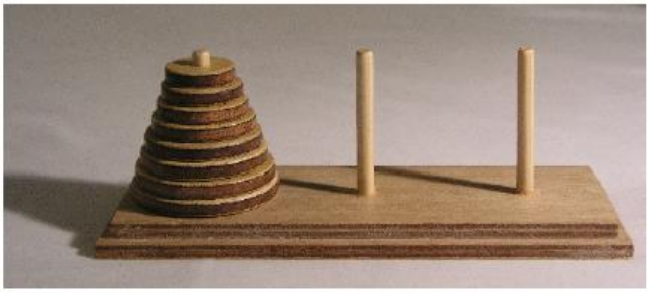


Figure 1: Tower of Hanoi

Please write a program to list the procedure of moving n disks from one rod to another. We assume there are only 3 rods, and all disks are placed on the first rod with the ascending order of size. Our target is to move these n disks from the first rod to the second rod. The disks are labeled $1, 2, \ldots, n$ from top to bottom.

◎題意說明→寫出一個程式，讓使用者輸入 n，並一一輸出盤子移動的過程，盤子由上至下必須由小至大放置，一次只能移動一個盤子到另一個桿上，最終要全部在**第二個**桿子上

Wait! I have told you that every recursive program can be converted to an iterative program (loop). So this time, I ask you to write TWO programs. hw0401-1 is the recursive version and hw0401-2 is the iterative function. Good Luck.

You need to implement these with two different functions in another C code and prepare a header.

※注意：必須分成標頭檔方式撰寫，並寫成**遞迴**、**迴圈**兩種方式分別為 hw0401-1 和 hw0401-2

◎輸入格式

```
$ ./hw0401-1
Please enter the disk number (2-20): 2
move disk 1 to rod 3
move disk 2 to rod 2
move disk 1 to rod 2
```

編譯後，執行"$ ./hw0401-1"或執行"$ ./hw0401-2"
並輸入一個在[2,20]內的數

◎檢查

1) 若輸入的數值不在[2,20]內的話=>提示使用者重新輸入

◎輸出格式

1) 輸出盤子移動的過程
2) "move disk [number] to rod [number]"

◎程式設計思路

1) 寫一個二維陣列存放每根桿子目前盤子的順序為何，並利用類似堆疊的方式製作，紀錄目前想移動的盤子在哪根桿，以及目的地，和 disk 為何，再以 cur 紀錄上一次移動到了哪裡
2) 當第二根桿子上的盤子不等於 n 的數值就一直跑迴圈
3) 每次記錄自己移動到哪了，下次 loop 時，則從除了該位置以外的桿子，找到盤子最小所在的桿子，存為 ptr(目前要移動的位置)
4) 發現目前所在位置若為**第一根或第二根桿子，則該桿子上盤子總數為奇數則往右移動，偶數則往左移動**，若為**第三根桿子，則盤子總數為奇數會往左移動，偶數會往右移動**
5) 若移動超出邊界則從另一邊繼續開始，按照這樣的規律我們可以找出 dir(目的地)到哪，接著只要輸出移動的過程，並改變好 rod 目前三根桿子的狀態，以及各個桿子的數目，最終將 cur 設為 dir 即可

◎各情形範例

1) Ex: ”1”

Tip: 輸入數不在[2,20]範圍內=>提示使用者重新輸入

```
┌─yuchen0515@NTNUMATHLIN ~/程式設計hw4
└─$ ./hw0401-2
Please enter the disk number (2-20): 1
Input error! your valus isn't on the interval [2,20]
Please enter the disk number (2-20):
```

2) Ex: ”2”

Tip: test case #1

```
Please enter the disk number (2-20): 2
move disk 1 to rod 3
move disk 2 to rod 2
move disk 1 to rod 2
```

3) EX: 3

```
Please enter the disk number (2-20): 3
move disk 1 to rod 2
move disk 2 to rod 3
move disk 1 to rod 3
move disk 3 to rod 2
move disk 1 to rod 1
move disk 2 to rod 2
move disk 1 to rod 2
```

4) EX: 4

```
Please enter the disk number (2-20): 4
move disk 1 to rod 3
move disk 2 to rod 2
move disk 1 to rod 2
move disk 3 to rod 3
move disk 1 to rod 1
move disk 2 to rod 3
move disk 1 to rod 3
move disk 4 to rod 2
move disk 1 to rod 2
move disk 2 to rod 1
move disk 1 to rod 1
move disk 3 to rod 2
move disk 1 to rod 3
move disk 2 to rod 2
move disk 1 to rod 2
```

說明

2 Polynomials (20 pts)

You have learned a lot about polynomials, right? Now please write a program to calculate the derivatives of a given equation. The following example is for the equation $f(x) = x^2+2x+3$.

◎題意說明→想必你們一定學了很多關於多項式的數學吧~ 請寫出一個程式，讓使用者輸入次方數，各項係數，欲代入的值，並依序微分並輸出函數的數值。

◎檢查

1) 次方數必須非負→若不符合則提示使用者重新輸入

◎輸入格式

```
$ ./hw0402
Please enter the degree: 2
Please enter the coefficients: 1 2 3
Please enter x: 1
f(1) = 6
f^1(1) = 4
f^2(1) = 2
```

編譯後，執行"$ ./hw0402"，輸入：

1) 最高次方數(需大於等於 0)
2) 各項係數 (共有最高次方數+1 項)
3) 欲代入的數

◎輸出格式

1) 先以 x 的值代入原函式，並輸出數值
2) 微分後再代入函式，輸出數值
3) …一路微分到最高次方數為 0

◎程式設計思路

1) 先依序儲存使用者所輸入的數，分別為 degree, coe 陣列, x

※其中次方數必須記得檢查為非負

2) 外迴圈跑(次方數+1)次，依序輸出 f(x), f^1(x), … f^n(x)的數值
3) 內圈依序跑(degree-i)次，此處為簡單起見將 f^n(x)中分成 f／^n／(x)寫，在內圈中將係數改存放(係數*次方)，並在 res 中加進該數*(次方-1)
4) 因為剛開始會代入原函式，尚未微分，因此檢查(if (i!=0))

※最難的在於變數的次方該以什麼代入，此處係數乘以(degree -i-j+1)，i 代表為分到第幾階，j 代表到第幾項，次方則到(degree-i-j)

※i 作為紀錄目前**已微分次數**，j 則是**項次**

※特別的地方是 pow 函式不使用 math.h 的函式改自己寫，並且用**快速冪**的原理

Ex. Pow_(2, 16) the same as 2^{16}

→x = 2, y = 16

→若 y 為偶數我可以將 2^{16} 重新表達成： Pow_(2,16/2) * Pow_(2,16/2)

一直跑遞迴下去，會遇到 pow_(2,1) 此時

→若 y 為奇數我可以將 2^1 重新表達成： Pow_(2,1>>1) * x

小舉例：$2^{15} = 2^7 * 2^7 * 2$

→直到跑到 y = 0 時回傳 1

因此整體函式時間複雜度為 $\log_2 N => \log_2 16 => 4$ 就可以計算完 2^{16} 的數值

◎各情形範例

1) Ex: ”-1”

Tip: 次方數為負數→重新輸入

```
┌─yuchen0515@NTNUMATHLIN ~/程式設計hw4
└─$ ./hw0402
Please enter a non-negative drgree: -1
Input error! your value isn't over the zero.
Please enter a non-negative drgree:
```

2) Ex: ”2／1 2 3／1”

Tip: **f(x) = x^2+2x+3**，代入 1 以及微分後的各項數值

```
Please enter a non-negative drgree: 2
Plese enter the coefficients ( 3 number ): 1 2 3
Please enter x: 1
f(1) = 6
f^1(1) = 4
f^2(1) = 2
```

→f(1) = 1^2+2*1+3 = 1+2+3 = **6**

→f’(x) = 2x+2 => f’(1) = 2*1+2= 2+2 = **4**

→f’’(x) = 2 => f’’(1) = **2**

3) Ex: ”3／8 4 5 7／2”

Tip: **f(x) = $8x^3$+$4x^2$+5x+7**，代入 2 以及微分後的各項數值

```
Please enter a non-negative drgree: 3
Plese enter the coefficients ( 4 number ): 8 4 5 7
Please enter x: 2
f(2) = 97
f^1(2) = 117
f^2(2) = 104
f^3(2) = 48
```

驗證：

→f(2) = 8*2^3+4*2^2+5*2+7 = 8*8 + 4*4 + 5*2 + 7 = 64+16+10+7= **97**

→f’(x) = $24x^2$+8x+5 => f’(2) = 24*2^2+8*2+5 = 96+16+5 = **117**

→f’’(x) = 48x + 8 => f’’(2) = 48*2+8 = 96+8 = **104**

→f’’’(x)=48 => f’’’(3) = **48**

4) Ex: ”0／7／11”

Tip: **f(x) = 7**，代入 11 以及微分後的各項數值

```
Please enter a non-negative drgree: 0
Plese enter the coefficients ( 1 number ): 5
Please enter x: 2
f(2) = 5
```

驗證：

→f(11) = 7

→此測資主要為驗證當 f(x)為**常數函數**時，程式是否能正常運行

說明

3 Redo and Undo (20pts)

I believe that you all have used Redo and Undo functions in your text editor. Now, let's implement a program to simulate their behaviors. Suppose your editor can memorize 10 actions. Each action is represented by a positive integer. The simulator is an infinite loop and works as follows:

- When a user inputs a positive integer, add this integer to your action buffer and it will be the newest action. That is, there will no action that you can redo. If the buffer is full, kick off the oldest one.
- When a user inputs -1, it means the user wants to undo an action. Note that the action that is undone is still in your memory buffer since you may redo the action again.
- When a user inputs -2, it means the user wants to redo an action. There will be no new coming action. If nothing can be redone, just skip this command.
- When a user input 0, terminate the program and dump all actions in the buffer according to their coming order. You should also point out the current action. Note that the output action number may be less than 10. 0, -1, -2 are not actions and should not be printed.

◎題意說明→設計一個程式可以模擬文字編輯器的復原和取消復原，其中妳的程式應該要能存 10 個動作，並使用無限迴圈模擬以下狀況：

(1) 每輸入一個正整數，就將此數加進「動作暫存」內，並且視為最新的操作，如果滿十個的話，必須淘汰最舊的那一個

(2) 當輸入-1 時，意味著想使用「復原」，但後面那些數仍然要持續存在 buffer 內，因為你可能會需要使用到「取消復原」

(3) 當輸入-2 時，意味著想使用「取消復原」，如果之前沒做過「復原」但先輸入了這指令請忽略他

(4) 當輸入 0 時，意味著輸入終止了，請按順序輸出 buffer 內存的值，並以(*)標註目前所在位置，0,-1,-2 不用印出來

◎檢查

1) 檢查輸入的數值應大於等於-2，-2,-1,0 為操作，其餘正整數為存放的值，此外的數(小於-2)不應存放，因此若使用者輸入小於-2 的值=>提示重新輸入

◎輸入格式

```
$ ./hw0403
Please enter your action: 3
Please enter your action: 2
Please enter your action: 1
Please enter your action: 4
Please enter your action: 5
Please enter your action: 0
3 2 1 4 5(*)
```

編譯後，執行"$ ./hw0403"

依提示輸入整數或-1(復原)或-2(取消復原)，輸入 0 程式終止

◎輸出格式

1) 依照 buffer 存的順序輸出，並以(*)標註目前所在位置

◎程式設計思路

1) 設置一長度為 10 的 buffer 陣列、index(目前所在位置的索引)和 check(檢查使用者輸入的值)

2) 陣列初始化為-3，跑一無限迴圈，讀到值後存進 check 並進行判斷：

→如果小於-2 則重新輸入

→如果等於 0 則索引值-1 並跳出迴圈

→如果索引值等於 10，且數值為正整數，將**最舊的汰換掉**[索引值==10 代表存放的已經超過 10 個動作了]，並且將索引值調至 9[最末]

→如果要復原，而且索引值大於 0，則將索引值-1 [表示索引值為 0，當然不能復原]

→如果要取消復原而且索引值的下一位數不等於-3(也就是**有值**的意思)，表示操作過復原，則 index+1

→如果數值為正整數，則存放在陣列的 index 位置上，並將索引值往下調

3) 跑迴圈，當 buffer 第 i 個等於-3 或 i>=10 就跳出[如此便能讀到沒值就停]，另當 i==index 時，輸出(*)以告知目前所在位置

◎各情形範例

1) Ex: ”-3”

Tip: 小於-2 的數

```
┌─yuchen0515@NTNUMATHLIN ~/程式設計hw4
└─$ ./hw0403
Please enter your action: -3
Input error! your number both isn't a action and positive number!
Please enter your action:
```

2) Ex: ”3 2 1 4 5 0”

Tip: 第三題 範例#1

```
┌─yuchen0515@NTNUMATHLIN ~/程式設計hw4
└─$ ./hw0403
Please enter your action: 3
Please enter your action: 2
Please enter your action: 1
Please enter your action: 4
Please enter your action: 5
Please enter your action: 0
3 2 1 4 5(*)
```

3) Ex: ”3 2 1 -1 -1 0”

Tip: 第三題 範例#2

```
Please enter your action: 3
Please enter your action: 2
Please enter your action: 1
Please enter your action: -1
Please enter your action: -1
Please enter your action: 0
3(*) 2 1
```

4) Ex: “3 2 1 -1 -1 -2 100 0”

Tip: 第三題 範例#3

```
Please enter your action: 3
Please enter your action: 2
Please enter your action: 1
Please enter your action: -1
Please enter your action: -1
Please enter your action: -2
Please enter your action: 100
Please enter your action: 0
3 2 100(*)
```

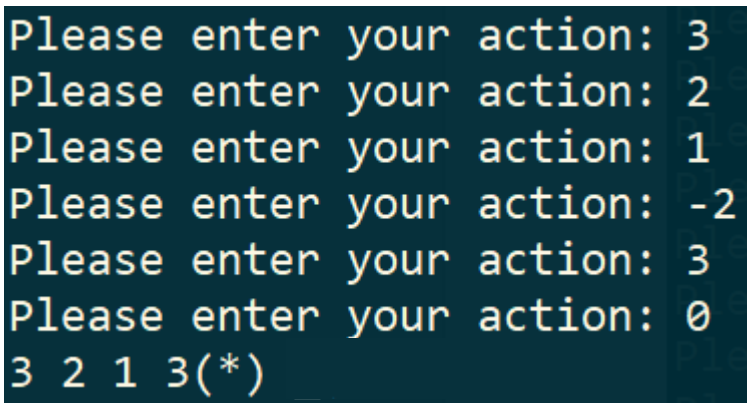
5) Ex: “3 2 1 4 5 3 2 1 -1 -1 3 2 1 -1 -1 -2 100 0”

Tip: 第三題 範例#1+#2+#3

```
Please enter your action: 3
Please enter your action: 2
Please enter your action: 1
Please enter your action: 4
Please enter your action: 5
Please enter your action: 3
Please enter your action: 2
Please enter your action: 1
Please enter your action: -1
Please enter your action: -1
Please enter your action: 3
Please enter your action: 2
Please enter your action: 1
Please enter your action: -1
Please enter your action: -1
Please enter your action: -2
Please enter your action: 100
Please enter your action: 0
3 2 1 4 5 3 3 2 100(*)
```

6) Ex: “3 2 1 -2 3 0”

Tip: 無復原 用取消復原



7) Ex: “1 2 3 4 5 6 7 8 9 10 11 12 13 -1 -1 -2 14 15 16 17 18 19 20 -2 -1 0”
Tip: buffer 超出 10 個

```
Please enter your action: 1
Please enter your action: 2
Please enter your action: 3
Please enter your action: 4
Please enter your action: 5
Please enter your action: 6
Please enter your action: 7
Please enter your action: 8
Please enter your action: 9
Please enter your action: 10
Please enter your action: 11
Please enter your action: 12
Please enter your action: 13
Please enter your action: -1
Please enter your action: -1
Please enter your action: -2
Please enter your action: 14
Please enter your action: 15
Please enter your action: 16
Please enter your action: 17
Please enter your action: 18
Please enter your action: 19
Please enter your action: 20
Please enter your action: -2
Please enter your action: -1
Please enter your action: 0
10 11 12 14 15 16 17 18 19 20(*)
```

8) Ex: “-2 -2 -1 -1 -2 2 -2 3 -1 6 -2 0”

Tip: 很多的復原、取消復原

```
Please enter your action: -2
Please enter your action: -2
Please enter your action: -1
Please enter your action: -1
Please enter your action: -2
Please enter your action: 2
Please enter your action: -2
Please enter your action: 3
Please enter your action: -1
Please enter your action: 6
Please enter your action: -2
Please enter your action: 0
2 6(*)
```

說明

4 Linear regression (20 pts)

In statistics, linear regression is a linear approach to modeling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression. If you do not know what linear regression is, please check it on the wikipedia.

In this homework, I want you to implement **Simple linear regression**. That is, I want you to find a linear function (a non-vertical straight line) that, as accurately as possible, predicts the dependent variable values as a function of the independent variables. The accuracy of each predicted value is measured by its squared residual (vertical distance between the point of the data set and the fitted line), and the goal is to make the sum of these squared deviations as small as possible. Figure 2 is an example. Users

◎題意說明→使用者輸入座標數量，以及各座標(x,y)值，請求出整體的迴歸直線為何(題目給定 請找出 **非垂直**的迴歸直線)

◎檢查

1) 使用者輸入的座標數量必須大於 0，否則提示使用者重新輸入

2) 計算標準差、相關係數推出迴歸直線 y = ax+b 時，要確保 a 不等於 0

→否則輸出 迴歸直線 不存在 (即使存在也是垂直的，不符合題意要我們找的)

◎輸入格式

```
$ ./hw0404
Please enter the point number: 4
Please enter Point 1: 80 11
Please enter Point 2: 90 12
Please enter Point 3: 110 8
Please enter Point 4: 120 9
Regression Equation: y = -0.08 x + 18
```

編譯後，執行"$ ./hw0404"
首先，輸入座標數，接下來依序輸入個別座標的 x,y 值

◎輸出格式

1) 依據輸入的點，輸出一個整體的迴歸直線，若為垂直(即斜率未定義)則輸出不存在(我們要找的是符合題意的迴歸直線)

◎參考公式

一、標準差

基本定義 [編輯]

$$SD = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

μ為平均值（$\overline{x}$）。

二、相關係數

相關係數的公式如下:[2]

$$r = \frac{\sigma xy}{\sigma_x \sigma_y} \quad (1)$$

$$\sigma xy = \sigma^2 xy = \frac{\sum(x-\overline{x})(y-\overline{y})}{n}$$

$$\sigma_x = \sqrt{\frac{\sum(x-\overline{x})^2}{n}}$$

$$\sigma_y = \sqrt{\frac{\sum(y-\overline{y^2})}{n}}$$

三、迴歸直線

$$y = r \times \frac{\sigma_y}{\sigma_x}(x-\mu_x)+\mu_y$$

$$\Rightarrow y = r \times \frac{\sigma_y}{\sigma_x}x + \left(\mu_y - r \times \frac{\sigma_y}{\sigma_x}\mu_x\right) \quad ....(3)$$

我們可以看出公式固定有一些結構重複出現：

(1) 離均差平方

(2) 離均差相乘

(3) 平均

因此只要算出這三者，就能進而計算 x,y 標準差以及 xy 標準差

→推出相關係數→再運用前述所計算的結果計算迴歸直線

◎程式設計思路

1) 儲存座標數並檢查是否大於 0=>若否則提示使用者重新輸入
2) 根據上述數值開 x,y 陣列儲存 (讀數值時，一邊計算平均)
3) 利用前述所推出的基本三結構 平均、離均差平方、離均差相乘計算式子
4) 使用三結構計算出三種標準差(x,y,xy)→推出相關係數→迴歸直線
5) 設定 check_lower，檢查分母是否為 0，若為 0 就不要進行除法以免出現錯誤，並且若為 0 則迴歸直線會無法計算(可能為垂直)
6) 印出到小數點後第二位的方程式

※pow 部分一樣使用快速幕做，而 sqrt 則引用 math.h 的函式，自己寫會需使用數次二分逼近會有點麻煩

◎各情形範例

1) Ex: ”-1”
Tip: 座標數必須大於 0

```
┌─yuchen0515@NTNUMATHLIN ~/程式設計hw4
└─$ ./hw0404
Please enter the point number: -1
Input error! your number isn't over 1
Please enter the point number:
```

2) Ex: ”4／80 11／90 12／110 8／120 9”
Tip: 作業基本測資

```
Please enter the point number: 4
Please enter Point 1: 80 11
Please enter Point 2: 90 12
Please enter Point 3: 110 8
Please enter Point 4: 120 9
Regression Equation: y = -0.08 x + 18.00
```

3) Ex: ” 4／80 12／70 24／60 1／50 90”

Tip: 較無關係的四個座標值

```
Please enter the point number: 4
Please enter Point 1: 80 12
Please enter Point 2: 70 24
Please enter Point 3: 60 1
Please enter Point 4: 50 90
Regression Equation: y = -2.11 x + 168.90
```

4) Ex: ” 4／20 50／20 98／20 134／20 23”

Tip: 迴歸直線可能不存在或垂直

```
Please enter the point number: 4
Please enter Point 1: 20 50
Please enter Point 2: 20 98
Please enter Point 3: 20 134
Please enter Point 4: 20 23
Regression Equation isn't exist
```

說明

5 Modular Multiplicative Inverse (20 pts)

In this class, I ask you to implement extended Extended Euclidean algorithm yourself at home. I believe you know how to run this algorithm. Actually, Extended Euclidean algorithm can also help you to calculate the modular multiplicative inverse. A modular multiplicative inverse of an integer a is an integer x such that the product ax is congruent to 1 with respect to the modulus m. Note that a, x are between 0 and $m-1$. For example, given a modulus 7, $2^{-1} = 4$ since $2 \times 4 = 8 \equiv 1 (\text{mod } 7)$.

Extended Euclidean algorithm tells you that given two integers a and b, you can find integers x and y such that

$$ax + by = \gcd(a, b).$$

Suppose a and b are coprime, we can find integers x and y such that

$$ax + by = 1.$$

So if the modular multiplicative inverse of b is y when modulo a since

$$ax + by (\text{mod } a) = by (\text{mod } a) = 1 (\text{mod } a).$$

Now, please implement a function for calculating the modular multiplicative inverse.

◎題意說明→擴展歐幾里得(輾轉相除)已告訴你給定兩整數 a,b，你能找出 x,y 使得 ax + by = gcd(a,b) 進一步推展後能用來計算模倒數，請利用此式子，讓使用者輸入模數、整數後，輸出該整數的模倒數，若不存在請輸出沒有

You need to implement these with two different functions in another C code and prepare a header. Do not forget to check errors.

※請使用函式並**以標頭檔實作**

◎輸入格式

```
$ ./hw0405
Please enter the modulus: 7
Please enter the number: 3
The modular multiplicative inverse of 3 is 5.
```

編譯後，執行"$ ./hw0405"
依提示輸入一個模數(modulus)，再輸入一個整數(number)

◎檢查

1) 模數必須大於 0，且輸入的整數亦必須大於 0=>若否提示使用者重新輸入

◎輸出格式

1) 輸出模倒數為何

◎推導

根據題目知 ax + by = gcd(a,b)→若 a,b 為互質則 ax + by = 1
→取模為 a：ax(mod a) + by(mod a) = 1 (mod a)
→能化簡並將 mod 寫成另種形式：by(mod a) = 1 + ak
→因模倒數為 0~m-1，也就是 0~a-1：by = 1 + ak
→y = (1+ak)/b 即為模倒數
→若 gcd(1+ak, b) = b => 可整除 => 模倒數存在

◎程式設計思路

1) 根據題目可知，若模倒數存在 a,b 必互質，因此以輾轉相除法檢驗是否互質，若否或著模數等於 1 則傳回-1(也就是不存在的意思)
2) 若互質則跑一個迴圈不斷累加 k 的值，若 1+ak 和 b 公因數為 b=>可整除，即回傳 1+ak/b，若一直無法整除則直到(1+ak)/b > a 時就跳出
3) 在輾轉相除法的設計上，除遇偶化簡(>>1)之外，兩者為奇數則運用更相減損法，在整體上能大幅提升速度

◎各情形範例

1) Ex: ”-1”

Tip: 輸入負數

```
┌─yuchen0515@NTNUMATHLIN ~/程式設計hw4
└─$ ./hw0405
Please enter the modulus: -1
Input error! please enter a number over zero
Please enter the modulus: 1
Please enter the number: -1
Input error! please enter a number over zero
Please enter the modulus:
```

2) Ex: “7／3”

Tip: test case#1

```
Please enter the modulus: 7
Please enter the number: 3
The modular multiplicative inverse of 3 is 5
```

驗證：

(1) (3*5) mod 7 = 15 mod 7 = 1

(2) 0<= 5 <= 7

3) Ex: “7／2”

```
Please enter the modulus: 7
Please enter the number: 2
The modular multiplicative inverse of 2 is 4
```

驗證：

(1) (2*4) mod 7 = 8 mod 7 = 1

(2) 0<= 4 <= 7

4) Ex: “2／7”

```
Please enter the modulus: 2
Please enter the number: 7
The modular multiplicative inverse of 7 is 1
```

驗證：

(1) (7*1) mod 2 = 7 mod 2 = 1

(2) 0<= 1 <= 2

5) Ex: “1／7”

Tip: mod 1 時沒有模倒數

```
Please enter the modulus: 1
Please enter the number: 7
The modular multiplicative inverse of 7 isn't exist
```

6) Ex: “4／2”

Tip: 兩者不互質時，模倒數不存在

```
Please enter the modulus: 4
Please enter the number: 2
The modular multiplicative inverse of 2 isn't exist
```

說明

6 Bonus: Code (10 pts)

Please explain the following code:

```
int p(int i, int N) {
    return (i < N && printf("%d\n", i) && !p(i + 1, N))
           || printf("%d\n", i);
}
```

Undoubtedly, please use Chinese. Do not tell me that I run this code and I find the output rule. Please explain the code itself.

◎題意說明→

(1)閱讀以下程式碼，並就「程式**本身**」解釋他

◎範例參考

```
//解釋請見README第六題

#include <stdio.h>
#include <stdint.h>

int p(int i, int N) {
    return (i < N && printf("%d\n", i) && !p(i + 1, N)) || printf("%d\n", i);
}

int main(){

    int i = 0 , N = 0;

    printf("Please enter two integer: ");
    scanf("%d %d", &i, &N);
    printf("%d\n", p(i,N));

    return 0;
}
```

```
┌─yuchen0515@NTNUMATHLIN ~/程式設計hw4
└─$ ./hw0406
Please enter two integer: 3 5
3
4
5
4
3
1
```

我們都知道，輸出結果為 3,4,5,4,3,1，不過究竟為何會輸出這樣的結果呢？

◎**解釋**

1) 首先，該函式回傳值內共分為兩個部分
一為(i<N **&&** printf(“%d\n”,i) **&&** !p(i+1,N))
二為 printf(“%d\n”,i)

2) 此函式最重要的觀念在於 printf 的回傳值，以及電腦對於&&運算的邏輯為何

3) 在一內，我們又能分成三個部分 **i<N**、**printf(“%d\n”,i)**、**!p(i+1,N)**

4) 電腦判斷 and 時，若無論 ture 以及 false，直接三個部分都判讀的話，在!p(i+1,N)會將 i 值不斷增加，並且會因第二部分而不斷輸出 i 值，陷入無限迴圈，但很顯然執行結果沒有出現這樣的情況

5) 根據結果，我們能知道輸出結果是**有限**的，也就是電腦判斷 and 時，是依據一個一個部份看，而且遇到 false 時，就會停止判斷下一部份，而電腦判斷 or 時，則兩邊都會判斷

6) 試著解釋上方 p(3,5)的輸出結果

一、起初 p(3,5) return 的是 (3<5)[True]，因此繼續判斷
printf(“%d\n”,i)[True／回傳值為 1，並且**輸出 i 值 3**]，繼續判斷!p(4,5)

二、p(4,5) return 的是 (4<5)[True]→printf(“%d\n”,i)[True／值為 1／**輸出 4**]
→!p(5,5)

三、p(5,5) return 的是 (5<5)[False]，因此不再判斷另兩個部分，此部分整體為 False，在或(||)左側，另判斷右側的 printf(“%d\n”,i)[True／值為 1／**輸出 5**]，因此整體為 True，回傳到第二部分=>!p(5,5) [False]

四、第二的三部分整體為 False，因為在或(||)的左側，亦判斷右側的 printf(“%d\n”,i)[True／值為 1／**輸出 4**]，整體為 Ture 回傳到第一部份 =>!p(4,5)[false]

五、第一的三部分整體為 False，因為在或(||)的左側，亦判斷右側的 printf(“%d\n”,i)[True／值為 1／**輸出 3**]，整體為 True，因 p 函式回傳型態為 int，因此最終在 printf(“%d\n”,p(i,N))**輸出為 1**(True)

六、上述依序順序的輸出值分別為 3,4,5,4,3,1，並且與實際輸出結果吻合

七、因此我們從該函式了解到兩件事：

第一為當 printf 也為條件判斷式的一部份時，他**有自己的回傳值**(實際傳入的變數數量)，**同時也會輸出**(依據 printf 的內容輸出)

第二，電腦在判斷**&&(and)**時，是**依序由左往右**一個一個判斷，若遇到 True 則繼續判斷下一部分，**False 則直接停止判斷下一部份**(因 F&&(F||T)必定為 F，也就是 F 無論和 T 或 F 做 and 運算始終為 F，但特殊的是電腦直接不判斷 F 後面的究竟是 T 還是 F，而遇到**||(or)時，則左右兩側都會判斷**

若&&(and)判斷非依此方式判斷時，在運行該函式時，電腦會一直判斷!p(i+1,N)，進入到函式後又再次判斷!p((i+1)+1,N)的數值，而第二部分則會有輸出值，因此會陷入無限遞迴中，並且不斷輸出 i 值，但實際運行結果並無如此，因此可再次印證電腦 and, or 的判斷方式。