程式設計(一)-HW03

Due to 11/3 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) hw0301.c
2) hw0302.c
3) hw0303.c
4) hw0304.c
5) hw0305.c
6) hw0306.c
7) Makefile
8) 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 hw0301.c~hw0306.c→產生 hw0301~hw0306 檔
指令如下：

```
$ make
$ ./hw0301
$ ./hw0302
…
```

以此類推，即可分別執行 hw0301~hw0306

四、索引

第一題————————————————P.2-4
第二題————————————————**P.4-6**
第三題————————————————P.6-8
第四題————————————————**P.9-10**
第五題————————————————P.10-12
第六題————————————————**P.12-14**

說明

1 Taylor's Theorem (20 pts)

I believe you have learned Taylor's Theorem, right? If no, do not worry. I will describe this theorem here.

Theorem 1. *Taylor's Theorem Let $k \geq 1$ be an integer and let the function $f : \mathbb{R} \to \mathbb{R}$ be k times differentiable at the point $a \in \mathbb{R}$. Then there exists a function $hk : \mathbb{R} \to \mathbb{R}$ such that*

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(k)}(a)}{k!}(x-a)^k + h(x)(x-a)^k,$$

and $\lim_{x\to\infty} h(x) = 0$.

Now I want you to use this theorem to calculate the value of e. The approximation will be

$$e^x = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^k}{k!}$$

The approximation accuracy depends on k. Please write a program to make the user input k and your program calculates e.

◎**題意說明**→請使用泰勒展開式估算自然常數 e 的數字，e 的估算型式如上，讓使用者輸入一非負整數，套進公式中，並輸入依據此 k 值所估算出的 e 值

◎**檢查**

1) k 值必須為非負整數→若不符合則提示使用者"input error!"並重新輸入

◎**輸入格式**

```
$ ./hw0301
k-th order Taylor polynomial for e
Please enter k: 1
2
```

編譯後，執行"$ ./hw0301"
並輸入一個非負整數(k)的值

◎**輸出格式**

1) 輸出依據 k 值計算出的自然常數數值
2) 此數值輸出到小數點後第二十位
3) 輸出第一行為以 double 型態計算的數值
4) 輸出第二行為以 float 型態計算的數值

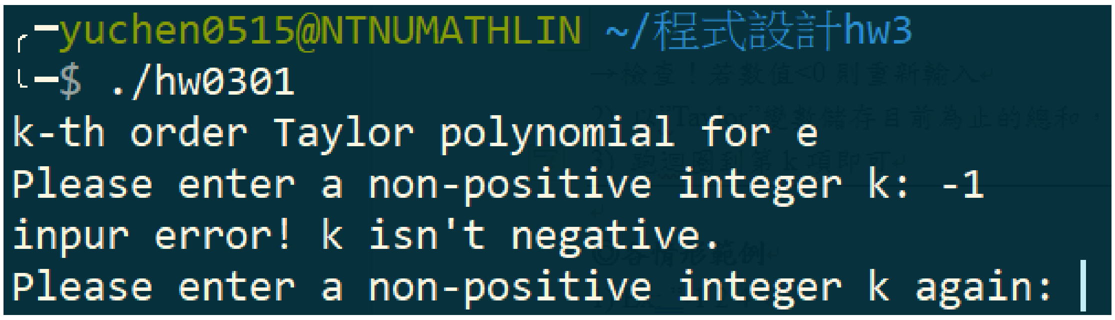
◎**程式設計思路**

1) 以變數 k 儲存使用者輸入的 k

→檢查！若數值<0 則重新輸入
2) 以”Taylor”變數儲存目前為止的總和，以 temp 儲存每次乘以(1/i)後的數值
3) 跑迴圈到第 k 項即可

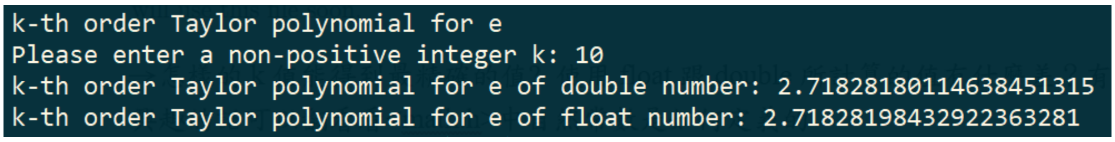
◎各情形範例

1) Ex: ”-1”
Tip: k 值輸入為負，提示使用者重新輸入



```
╭─yuchen0515@NTNUMATHLIN ~/程式設計hw3
╰─$ ./hw0301
k-th order Taylor polynomial for e
Please enter a non-positive integer k: -1
inpur error! k isn't negative.
Please enter a non-positive integer k again: 
```

2) Ex: ”10”
Tip: k 值疊加到(x^10/10!)所估算的情形



```
k-th order Taylor polynomial for e
Please enter a non-positive integer k: 10
k-th order Taylor polynomial for e of double number: 2.71828180114638451315
k-th order Taylor polynomial for e of float number: 2.71828198432922363281
```

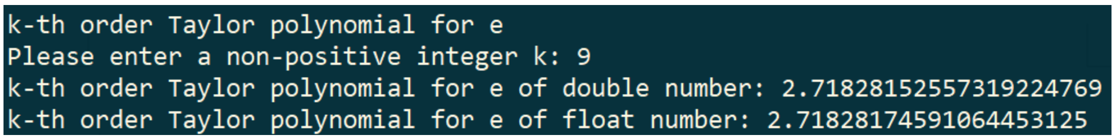
◎題目回應

What is the k value for the best accuracy? Is there any difference between **float** and **double**? Please write down your answer.
BTW, if you are interested in how to use e in standard C, you can read **math.h**. We will use this file soon.

→怎樣的 k 值會得到最精確的值? 使用 float 跟 double 所計算的值有什麼差？有興趣的話可以翻看看<math.h>中自然常數是如何定義的

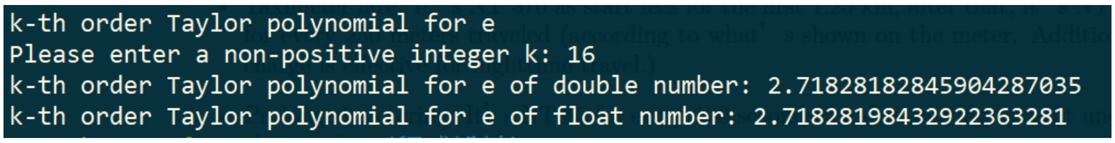
一般自然常數數字大略如下：
$\boldsymbol{e} = \mathbf{2.71828182845904523536}\cdots$

Ans:
(1) 若以 float 計算 k=9 時，是最精確的
當 k=6 時，精確到 2.718
k=7 時，精確到 2.7182
k=9 時，精確到 2.718281，估算為 2.71828174591
k=10 時，精確到 2.7182818 ，估算為 2.718281984329
k=11 時，精確到 2.71828182



```
k-th order Taylor polynomial for e
Please enter a non-positive integer k: 9
k-th order Taylor polynomial for e of double number: 2.71828152557319224769
k-th order Taylor polynomial for e of float number: 2.71828174591064453125
```

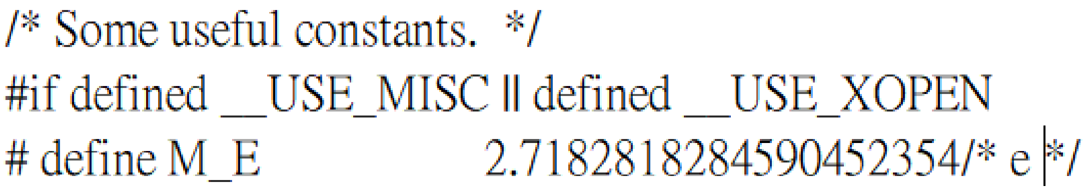
(2) 而若以 double 計算時，則精確到小數點後第 15 位，而當 k>=17 時，數值是較精確的，如圖



```
k-th order Taylor polynomial for e
Please enter a non-positive integer k: 16
k-th order Taylor polynomial for e of double number: 2.71828182845904287035
k-th order Taylor polynomial for e of float number: 2.71828198432922363281
```

(3) 因此 float 和 double，float 有效數字是 7 位，到小數點後 6 位精確，之後因數值過小，超出 float 的精確度因此會出現 k=9 比 k=10 還精確的情形，超出他所能承受的精確度數值就不會是你原本要的數值而會有誤差，進一步影響到值，而 double 有效數字是 16 位，到小數點後 15 位精確

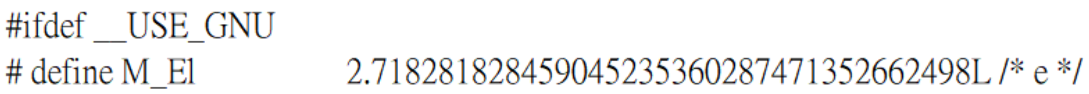
(4) <math.h>中，對自然常數定義如下



```
/* Some useful constants.  */
#if defined __USE_MISC || defined __USE_XOPEN
# define M_E		2.7182818284590452354/* e */
```

為 20 位有效數字的 e，

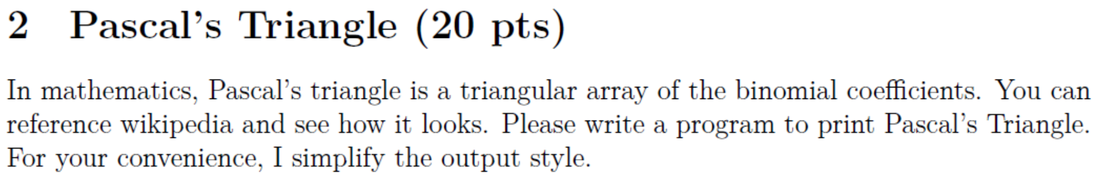
而 long double 定義則如下



```
#ifdef __USE_GNU
# define M_El		2.718281828459045235360287471352662498L /* e */
```

大致而言並不是經由計算，而是直接將計算好的數值定義為 e 值代入

說明



2 Pascal's Triangle (20 pts)

In mathematics, Pascal's triangle is a triangular array of the binomial coefficients. You can reference wikipedia and see how it looks. Please write a program to print Pascal's Triangle. For your convenience, I simplify the output style.

◎題意說明→使用者輸入帕斯卡三角形的高(h)，依照帕斯卡三角形的原則，寫出一個程式印出這種三角形

◎檢查

1) k 值必須在區間[1,10]之內→若不符合則提示使用者"input error!"並重新輸入

◎輸入格式

```
$ ./hw0302
Please enter the height of a Pascal's Triangle (1-10): 4
1
1   1
1   2   1
1   3   3   1
```

編譯後，執行”$ ./hw0302”

並輸入一個介於閉區間[1,10](帕斯卡三角形)的高(integer)

◎輸出格式

1) 依列依行輸出印出帕斯卡三角形

◎程式設計思路

1) 儲存高度，並設定一個一維陣列儲存每行該印出的數值

2) 若使用二維陣列會很簡單好寫，但浪費記憶體又有些複雜

3) 若由左往右更改數值，容易將原本的數值蓋掉，又以新數值計算

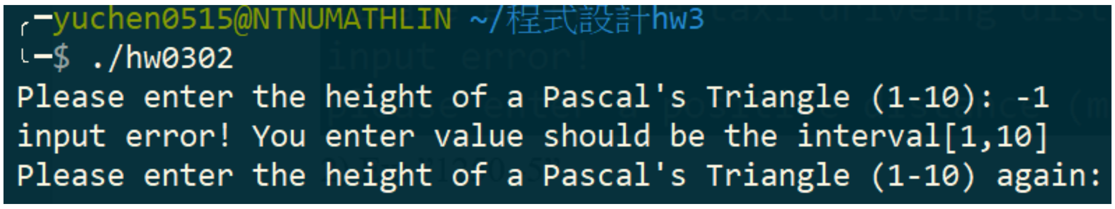
4) 由右往左掃，則不會有此問題，但要注意邊界上的處理

5) 第一行 1，在陣列中會不好處理，pascal 三角形需要運用到[i] + [i-1]會超出邊界，因此先將此數值輸出(使用者輸入必介於[1,10]之間，故第一行必定會輸出

◎各情形範例

1) Ex: ”-1”

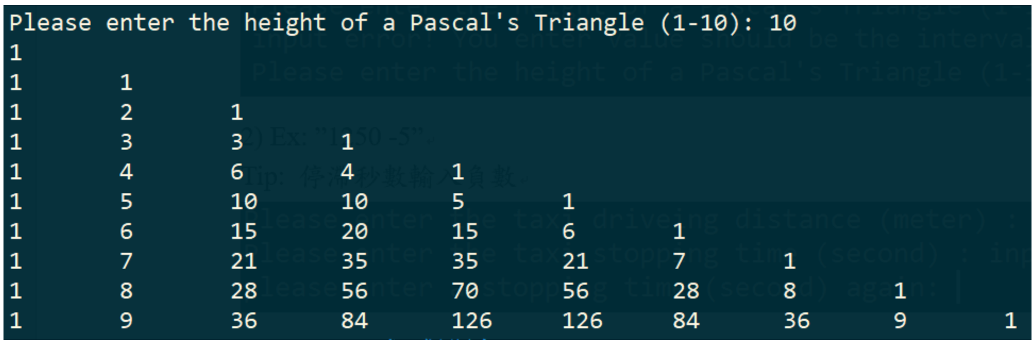
Tip: 高度(integer)輸入[1,10]以外的數值，提示使用者重新輸入



```
┌─yuchen0515@NTNUMATHLIN ~/程式設計hw3
└─$ ./hw0302
Please enter the height of a Pascal's Triangle (1-10): -1
input error! You enter value should be the interval[1,10]
Please enter the height of a Pascal's Triangle (1-10) again:
```

2) Ex: ”10”

Tip: 輸出高度為 10 的 Pascal 三角形



```
Please enter the height of a Pascal's Triangle (1-10): 10
1
1       1
1       2       1
1       3       3       1
1       4       6       4       1
1       5       10      10      5       1
1       6       15      20      15      6       1
1       7       21      35      35      21      7       1
1       8       28      56      70      56      28      8       1
1       9       36      84      126     126     84      36      9       1
```

◎題目回應

How do you handle the alignment issue? Please share your method/

→你怎麼處理校正(排版)問題的呢？請分享你的方法

Ans:

(1) 設計外迴圈、內迴圈，內輸出單行，外輸出多行，在內圈時輸出數值以”**\t**”定位方式排版，而內圈結束時以”**\n**”換行，即可整齊舒適

(2) 主要是使用**陣列**處理，若不使用陣列時，可設置 10 個變數依序處理也可

說明

3 Investment Profit (20 pts)

Please develop a program to calculate your investment profit. Your investment plan is as follows. Every month you put a fixed amount of money on some fund. The fund has an anticipated Annual interest rate. Of course, we call this anticipated rate because the actual rate may be higher or lower. We assume the rate range is +3% or -3%. Your program needs to make the user input **monthly investment amount** and the **anticipated annual rate**. You should list the **annual profits** for 20 years investment, including max, average, min. Note that it is compound interest and monthly interest payment.

◎題意說明→請設計出一個程式，能讓使用者輸入每月存入的金額，**預期年利率**，依”月複利”方式計算，另題目設定**實際年利率**波動最大為”預期年利率+3%”，最小為”預期年利率-3%”去計算，請輸出每年複利和的 max, average, min，其中 max 為以”**預期年利率+3%**”計算，aver 以”**預期年利率**”計算，min 以”**預期年利率-3%**”計算

※經詢問後，應**輸出複利和**，也就是包含本金、利息的金額

◎檢查

1) 檢查每月存入的金額是否介於[1000,50000]內，若否則提示使用者重新輸入

◎輸入格式

```
$ ./hw0303
Please enter your monthly investment amount (1000-50000): 10000
Please enter your anticipated annual rate (%): 1.5
Year 01: xxxx (max) xxxx (average) xxxx (min)
Year 02: xxxx (max) xxxx (average) xxxx (min)
...
```

編譯後，執行”$ ./hw0303”

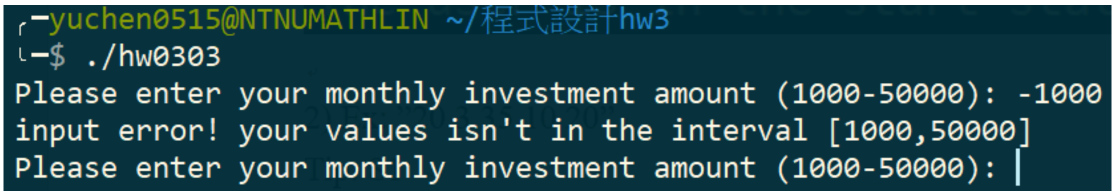
依提示輸入

1) 每月存入的額度(monthly investment amount)
2) 預期年利率(annual rate)

◎輸出格式
1) 依照順序，輸出 “Year xx: xxxx(max)　xxxx(average)　xxxx(min)”

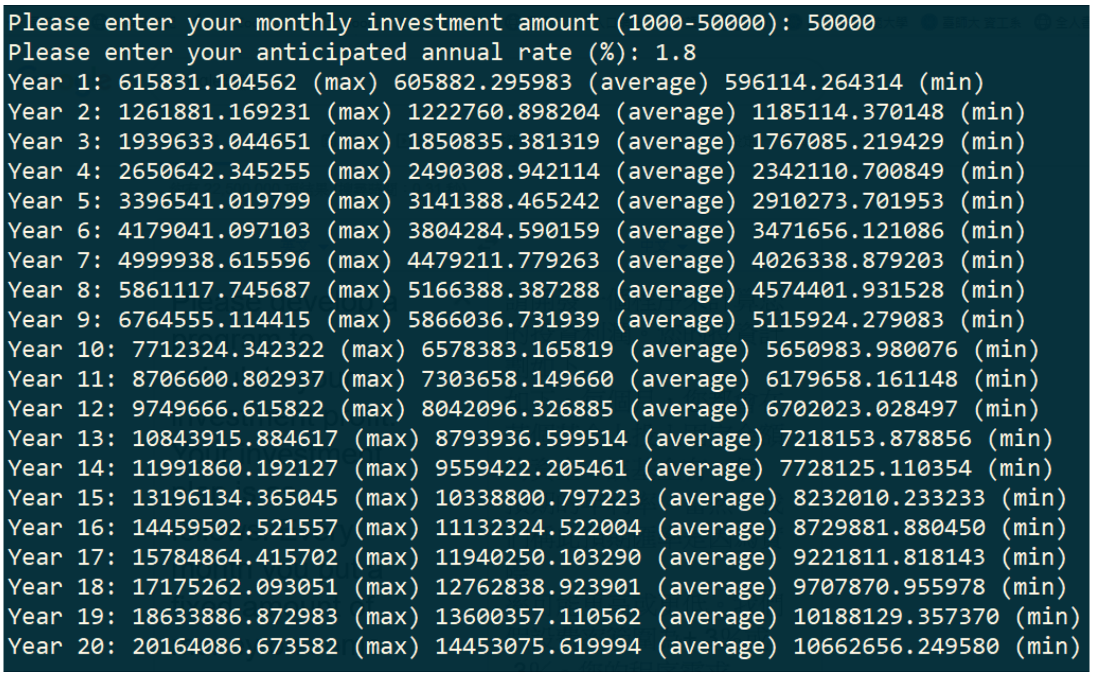
◎程式設計思路
1) 應設置變數儲存使用者輸入的月額度、年利率，並須設置三個變數，依序存放最大可能、預期、最小可能的複利和為 max, aver, min
※另需特別檢查使用者輸入是否在[1000,50000]區間內

2) 計算實際最大利率、實際利率、實際最小利率的變數，並處理掉百分比，設計外迴圈跑 20 次，而內迴圈跑 12 次(一年有十二月)，其中內迴圈能夠計算該年經過十二個月的月複利以後的複利和，再藉此計算出該年最終複利和
※不需扣除本金，我們需要輸出的是**複利和**
※預期年利率**不限多少百分比**(題目沒設限，只有每月金額有限)

◎各情形範例
1) Ex: ”-1000”
Tip: 每月額度超出[1000,50000]的範圍

```
┌─yuchen0515@NTNUMATHLIN ~/程式設計hw3
└─$ ./hw0303
Please enter your monthly investment amount (1000-50000): -1000
input error! your values isn't in the interval [1000,50000]
Please enter your monthly investment amount (1000-50000): 
```

2) Ex: ”50000 1.8”

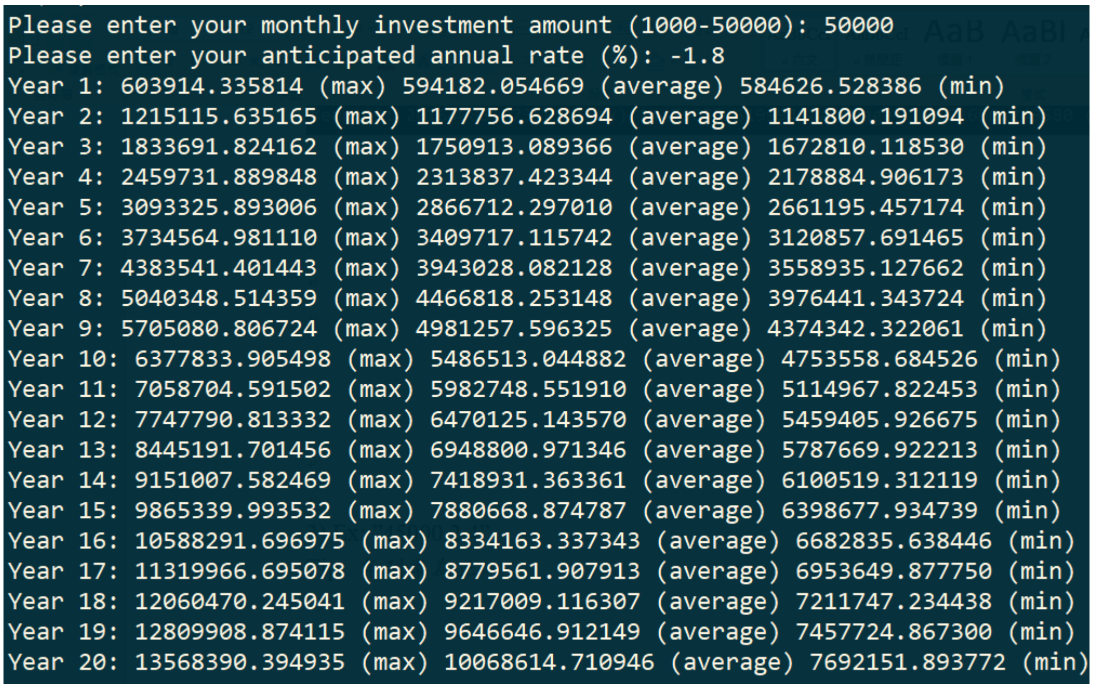
Tip: 年利率為正，實際利率最大為 4.8%，最小為-1.2%，平均為預期利率 1.8%



```
Please enter your monthly investment amount (1000-50000): 50000
Please enter your anticipated annual rate (%): 1.8
Year 1: 615831.104562 (max) 605882.295983 (average) 596114.264314 (min)
Year 2: 1261881.169231 (max) 1222760.898204 (average) 1185114.370148 (min)
Year 3: 1939633.044651 (max) 1850835.381319 (average) 1767085.219429 (min)
Year 4: 2650642.345255 (max) 2490308.942114 (average) 2342110.700849 (min)
Year 5: 3396541.019799 (max) 3141388.465242 (average) 2910273.701953 (min)
Year 6: 4179041.097103 (max) 3804284.590159 (average) 3471656.121086 (min)
Year 7: 4999938.615596 (max) 4479211.779263 (average) 4026338.879203 (min)
Year 8: 5861117.745687 (max) 5166388.387288 (average) 4574401.931528 (min)
Year 9: 6764555.114415 (max) 5866036.731939 (average) 5115924.279083 (min)
Year 10: 7712324.342322 (max) 6578383.165819 (average) 5650983.980076 (min)
Year 11: 8706600.802937 (max) 7303658.149660 (average) 6179658.161148 (min)
Year 12: 9749666.615822 (max) 8042096.326885 (average) 6702023.028497 (min)
Year 13: 10843915.884617 (max) 8793936.599514 (average) 7218153.878856 (min)
Year 14: 11991860.192127 (max) 9559422.205461 (average) 7728125.110354 (min)
Year 15: 13196134.365045 (max) 10338800.797223 (average) 8232010.233233 (min)
Year 16: 14459502.521557 (max) 11132324.522004 (average) 8729881.880450 (min)
Year 17: 15784864.415702 (max) 11940250.103290 (average) 9221811.818143 (min)
Year 18: 17175262.093051 (max) 12762838.923901 (average) 9707870.955978 (min)
Year 19: 18633886.872983 (max) 13600357.110562 (average) 10188129.357370 (min)
Year 20: 20164086.673582 (max) 14453075.619994 (average) 10662656.249580 (min)
```

3) Ex: “50000 -1.8”

Tip: 年利率為負，實際利率最大為 1.2%，最小為-4.8%，平均為預期利率-1.8%



```
Please enter your monthly investment amount (1000-50000): 50000
Please enter your anticipated annual rate (%): -1.8
Year 1: 603914.335814 (max) 594182.054669 (average) 584626.528386 (min)
Year 2: 1215115.635165 (max) 1177756.628694 (average) 1141800.191094 (min)
Year 3: 1833691.824162 (max) 1750913.089366 (average) 1672810.118530 (min)
Year 4: 2459731.889848 (max) 2313837.423344 (average) 2178884.906173 (min)
Year 5: 3093325.893006 (max) 2866712.297010 (average) 2661195.457174 (min)
Year 6: 3734564.981110 (max) 3409717.115742 (average) 3120857.691465 (min)
Year 7: 4383541.401443 (max) 3943028.082128 (average) 3558935.127662 (min)
Year 8: 5040348.514359 (max) 4466818.253148 (average) 3976441.343724 (min)
Year 9: 5705080.806724 (max) 4981257.596325 (average) 4374342.322061 (min)
Year 10: 6377833.905498 (max) 5486513.044882 (average) 4753558.684526 (min)
Year 11: 7058704.591502 (max) 5982748.551910 (average) 5114967.822453 (min)
Year 12: 7747790.813332 (max) 6470125.143570 (average) 5459405.926675 (min)
Year 13: 8445191.701456 (max) 6948800.971346 (average) 5787669.922213 (min)
Year 14: 9151007.582469 (max) 7418931.363361 (average) 6100519.312119 (min)
Year 15: 9865339.993532 (max) 7880668.874787 (average) 6398677.934739 (min)
Year 16: 10588291.696975 (max) 8334163.337343 (average) 6682835.638446 (min)
Year 17: 11319966.695078 (max) 8779561.907913 (average) 6953649.877750 (min)
Year 18: 12060470.245041 (max) 9217009.116307 (average) 7211747.234438 (min)
Year 19: 12809908.874115 (max) 9646646.912149 (average) 7457724.867300 (min)
Year 20: 13568390.394935 (max) 10068614.710946 (average) 7692151.893772 (min)
```

說明

4 Perfect number (20 pts)

What is a perfect number? Perfect number is a positive integer which is equal to the sum of its **proper positive divisors**. A positive proper divisor is a positive divisor of a number excluding itself. For example, 6 is a perfect number because proper divisors of 6 are 1, 2, 3 and $1+2+3=6$.

Please write a program for a user to input n and the program prints all perfect numbers that are smaller than n.

◎題意說明→使用者輸入一個 n 值介於[1,50000000]以內，我們需要設計出一個程式，找出從 1 到 n(不包含)的所有完美數

→完美數定義：定一整數 n，若其所有因數(不包含自己)相加後等於 n 時，則為 perfect number

◎檢查

1) 使用者輸入的 n 值必須介於[1,50000000]以內，若否則提示使用者重新輸入

◎輸入格式

```
$ ./hw0304
Please enter n (1-50000000): 30
6
28
```

編譯後，執行”$ ./hw0304”

依提示輸入一個介於 1~50000000 的整數 n

◎輸出格式

1) 輸出 1~n-1 的所有完美數

※注意：詢問助教該題時間限制後，回覆為本題**無時間限制**

◎程式設計思路

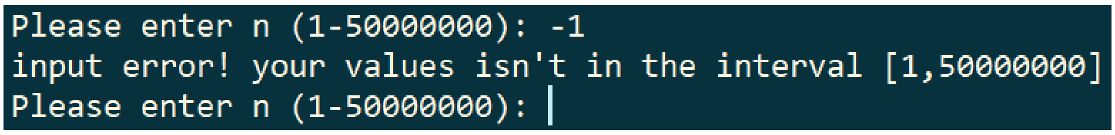
1) 儲存 n 值，並設置 sum_divisor 處理因數總和的問題
2) 外迴圈由 1 跑到 n-1，內迴圈則由 1 到(int)sqrt(i)+1 檢查因數，若是則疊加
3) 到(int)sqrt(i)+1 有幾個考量： 若 i 等於 9，則檢查到 3
   若 i 等於 8，則原本 sqrt(i)應是 2.XXX，取 int 為 2，再+1=3，
   此時 3*3 已經大於 8 了，故按照這樣的方式可以檢查到所有因數
4) EX:令 i=10，則因式有 1*10 = 10／2*5=10／5*2=10／10*1=10
   我們將只需要跑到 3，就能檢查到 1*10, 2*5，取出 1 和 2 的同時，再取出 10,5，即可不用算到後面去

5) 若迴圈中因數總和早已超出原本的數則跳出迴圈

◎各情形範例

1) Ex: ”-1”

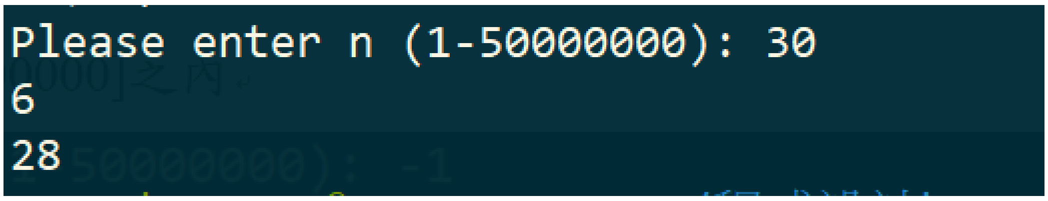
Tip:　n 值不在[1,50000000]之內



```
Please enter n (1-50000000): -1
input error! your values isn't in the interval [1,50000000]
Please enter n (1-50000000): 
```

2) Ex: ”30”

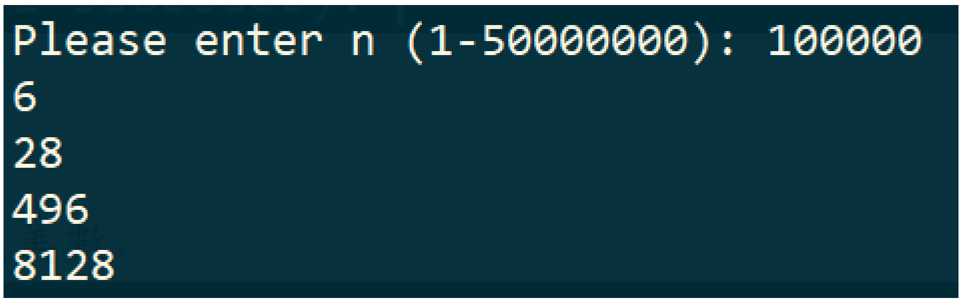
Tip: 取出 1-30 間的完美數



```
Please enter n (1-50000000): 30
6
28
```

3) Ex: ”100000”

Tip: 取出 1-100000 間的完美數



```
Please enter n (1-50000000): 100000
6
28
496
8128
```

說明

5　Add Digits (20 pts)

Given a non-negative integer, repeatedly add all its digits until the result has only one digit. Please print the process. For your convenience, I slightly change the print order.

◎題意說明→使用者輸入一個非負的整數 (小於 2^31 -1)，需要寫一個程式將她每一位數相加，並且再以相加後的數字重複數次動作，直到僅剩一位數為止，輸出”Final: (該數值)”，其中過程的計算必須輸出

◎檢查

1) 使用者輸入是否為非負整數，若否則提示使用者重新輸入

◎輸入格式

```
$ ./hw0305
Please enter a number: 38
8 + 3 = 11
1 + 1 = 2
Final: 2
```

編譯後，執行”$ ./hw0305”

依提示輸入一個非負整數 n

◎輸出格式

1) 將各位數疊加的過程輸出，並不斷疊加，直到位數僅剩一位，再輸出最終結果

◎程式設計思路

1) 以 num 變數儲存使用者輸入，而以 temp 變數放入 num，由此數作為檢查的條件，另以 sum 儲存每次位數疊加的結果，在最後將此數賦予 temp

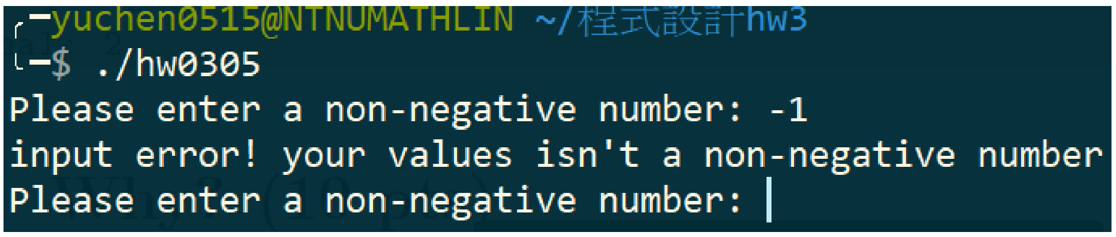
2) 於是外圈設置迴圈條件(temp>=10)，內圈則在跑一個 while(temp>0)的迴圈，在其內則判斷 temp>=10 以及 10>temp>=0，以便”+”號,”=”號的輸出

3) 跳出迴圈後，此時 temp 留存的數值即為最終結果，將此輸出即可

◎各情形範例

1) Ex: ”-1”

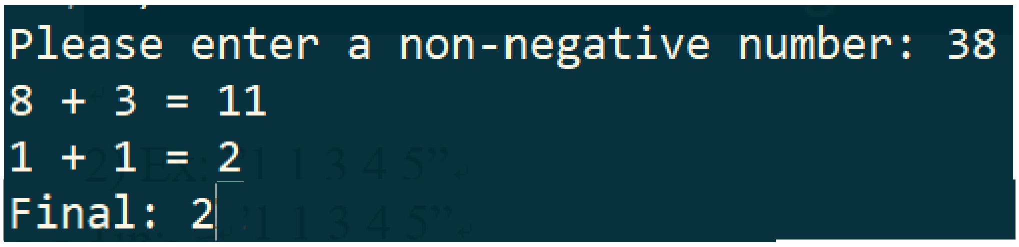
Tip: 輸入負數



```
┌─yuchen0515@NTNUMATHLIN ~/程式設計hw3
└─$ ./hw0305
Please enter a non-negative number: -1
input error! your values isn't a non-negative number
Please enter a non-negative number: 
```

2) Ex: “38”

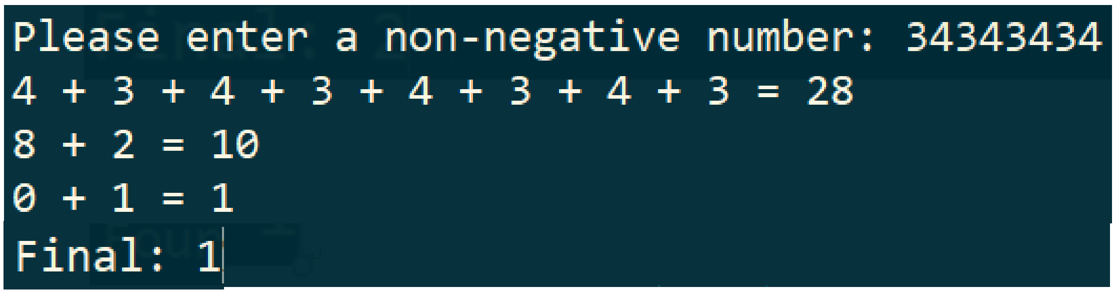
Tip: 僅有一個”+”、一個”=”的情形



```
Please enter a non-negative number: 38
8 + 3 = 11
1 + 1 = 2
Final: 2
```

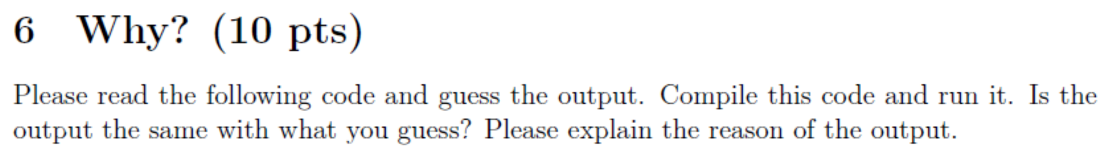
3) Ex: “34343434”

Tip: 第一次 digit 有多個加號的情形



```
Please enter a non-negative number: 34343434
4 + 3 + 4 + 3 + 4 + 3 + 4 + 3 = 28
8 + 2 = 10
0 + 1 = 1
Final: 1
```

說明



6 Why? (10 pts)

Please read the following code and guess the output. Compile this code and run it. Is the output the same with what you guess? Please explain the reason of the output.

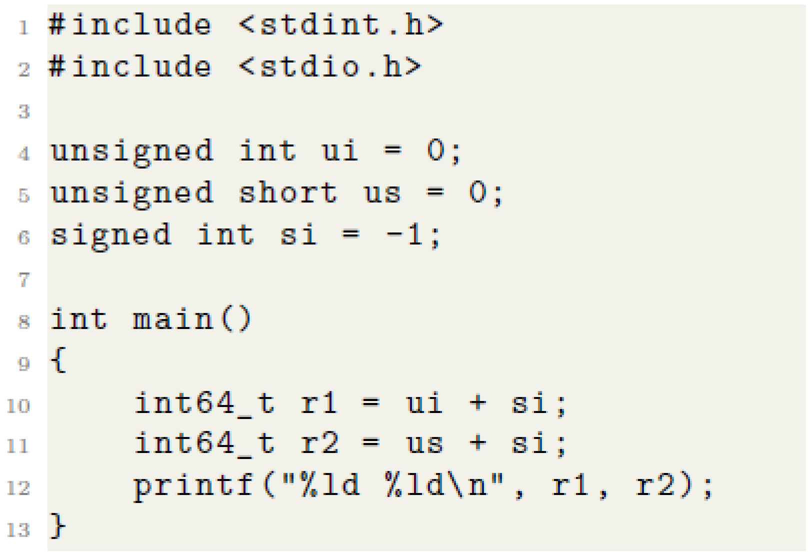
◎題意說明→

(1)閱讀以下程式碼，並猜測他的結果

(2)實際編譯並執行這隻程式碼

(3)這結果是否跟你的猜測相同

(4)請解釋為何出現這樣數值的原因



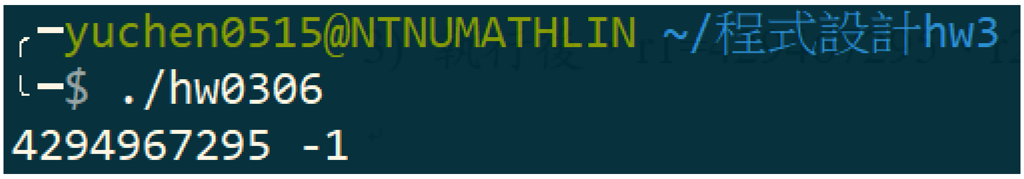
```c
#include <stdint.h>
#include <stdio.h>

unsigned int ui = 0;
unsigned short us = 0;
signed int si = -1;

int main()
{
    int64_t r1 = ui + si;
    int64_t r2 = us + si;
    printf("%ld %ld\n", r1, r2);
}
```

◎回答→

1)　由於 ui 值的範圍從 0 一直到 2^32，而 si 值則從-2^31~2^31-1，兩者範圍大小是一致的，猜測 unsigned int 和 signed int，雖範圍一樣但相加時應以 unsigned 型態計算，ui+si = -1，但因為是 unsigned int 型態此時往下溢位，則跑到最大值 -1，因此**猜測數值應為 2^32-1**

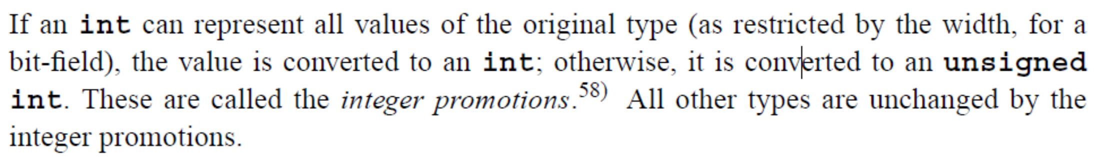
2) us 的值為 unsigned short int，其中範圍從 0 到 2^16-1，si 則為 signed int，範圍則由-2^31~2^31-1，兩者因 si 範圍較大，猜測運算時應會併入範圍較大的 signed int，故 us+ si = (signed int) -1，再和變數 r2(int64_t)合併，變為**(int64_t) -1**

3) 執行後，r1=429467295、r2=-1，與**猜測完全相符**



```
╭─yuchen0515@NTNUMATHLIN ~/程式設計hw3
╰─$ ./hw0306
4294967295 -1
```

4)



If an **int** can represent all values of the original type (as restricted by the width, for a bit-field), the value is converted to an **int**; otherwise, it is converted to an **unsigned int**. These are called the *integer promotions*.[58] All other types are unchanged by the integer promotions.

根據 C11 文件所述，<u>當 int 囊括了原始型態的所有範圍，則將此數值轉為 int 計算，倘若無法涵蓋原始型態的範圍，則轉為 unsigned int 計算</u>。

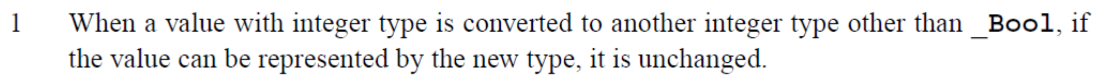
→當我們將 ui(unsigned int)+si(signed int)時，

(1) signed int 範圍從-2^31 到 2^31-1

(2) signed int 範圍從 0 到 2^32-1

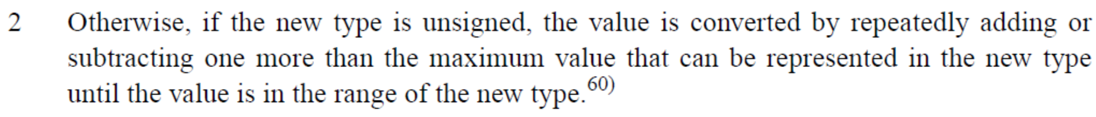
→int 無法涵蓋 2^31~2^32-1 的範圍

→計算時以 ui(unsigned int) + si(unsigned int)處理



1 When a value with integer type is converted to another integer type other than **_Bool**, if the value can be represented by the new type, it is unchanged.

上圖說明道「若由 A 的整數型態轉變到 B 的整數型態，<u>如果值能被 B 的整數型態表示時，其值不會更動</u>」



2 Otherwise, if the new type is unsigned, the value is converted by repeatedly adding or subtracting one more than the maximum value that can be represented in the new type until the value is in the range of the new type.[60]

而如果新型態是 unsigned，數值不在範圍中，則重複加或減，直到該值在 unsigned 的範圍中(處理溢位)

故 ui(unsigned int) + si(signed int) =>

ui(unsigned int) + si(unsinged int) => -1(unsigned int)→

範圍的最大值減 1 => 2^32-1

另外一個輸出則為 us(unsigned short) + si(signed int)
→unsigned short 在[0,2^16-1]範圍內
→signed int 在[-2^31,2^31-1]範圍中
→int 可完全涵蓋 unsigned short 的值→us(unsigned short) + si(signed int)
→us(signed int) + si(signed int) = -1(signed int)=>-1(int64_t)