b02303042 經濟四 鄭雅文

第一題

(a) output:

|  |
| --- |
| 1.32 99  WOW! |

因為r為float，且static\_cast<float>已將a轉為float，所以一個浮點數和一個int做除法運算結果仍為浮點數，所以r=1.32(type為float)。

r為float，b為int，兩者做乘法運算仍為浮點數，又1.32\*75剛好為整數99，所以r\*b輸出99。

r \* b != a為true：因為硬體對浮點數的儲存會有些微誤差，所以可能在比較兩數時，儲存兩數的binary bit會不一樣，所以機器return 兩數不相同。

(b)

程式碼改為

|  |
| --- |
| #include <iostream >  #include <iomanip>  using namespace std;  int main()  {  int a = 99;  int b = 75;  float r = static\_cast <float >(a) / b;  cout << r << " "<<setprecision(10) << r \* b << "\n";  if(r \* b != a)  cout << "WOW!\n";  return 0;  } |

會輸出

|  |
| --- |
| 1.32 99.00000763  WOW! |

證實r \* b會有浮點數儲存上的問題，不是剛好等於99。

(c)

輸出

|  |
| --- |
| 1 3 0.333333 |

程式num由1跑到99，deno從1跑到99，但由num=1，deno=1跑到99，再換num=2…跑到num=99。

因為輸出num=1 deno=3，所以已知跑過num=1 deno=1、num=1 deno=2，每次檢查if(num != deno \* r)其中r=static\_cast <float >(num) / deno;，跑過的

num=1 deno=1：因為r=1，所以沒有浮點數儲存的問題。

num=1 deno=2：因為r=0.5可用2-bit 單精度浮點數準確儲存，所以0.5為精確的數字。

num=1 deno=3：因為r=0.33333不能被準確的儲存成單精度浮點數，所以num != deno \* r成立，輸出(num deno r)

hasBadRatio為一flag，表示現在是否跳出迴圈的狀態，若一直為false則不跳出迴圈，因為在此設兩個break，在int deno=1的迴圈中若hasBadRatio為false，則跳出第一個迴圈；再檢查int num=1的迴圈，若hasBadRatio為false，則跳出第二個迴圈。為了避免hasBadRatio離開第一個迴圈就結束定義，而第二個迴圈又要重新判斷，所以將hasBadRatio設為兩個for loop的共用變數，讓兩個for loop都可使用。

第二題

(a) 因為int能儲存最大的數為2 31 –1= 247483647，但n!的成長速度太快，所以很快就會超過int所能儲存的最大的數，就會發生overflow的問題。

(b)

|  |
| --- |
| #include <iostream >  using namespace std;  int main()  {  int n = 0, m = 0;  cin >> n >> m;  if(n > m)  {  int num = 1;  for(int i = n ; i > n-m ; i-- )  { num \*= i;  }  int de1 = 1;  for(int i = 1 ; i <= m ; i++ )  de1 \*= i;  cout << num / de1 ;  }  return 0;  } |

用直觀的計算方法可化為，所以可化為，因為沒有把分子n!全部乘開，所以大幅減少分子的數值，可擴大計算範圍。

(若要再擴大計算範圍，當時，可再把m assign為n-m，才不會使分子又乘過多數字。)