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;
}
```

$\binom{n}{m}$用直觀的計算方法可化為$\frac{n\times(n-1)\times\ldots\times(n-m+1)}{1\times2\times\ldots\times m}$，所以$\binom{13}{3}$可化為$\frac{13\times12\times11}{1\times2\times3}$，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m $\geq \left\lfloor\frac{n}{2}\right\rfloor$時，可再把 m assign 為 n-m，才不會使分子又乘過多數字。)