A04 – Level Up

Editorial by พี่อยากนอนวันละ 8 ชม.

(Section กล่าวนำเฉย ๆ ข้ามได้ครับน้อง)

ถ้าจะให้พูดถึงความยากของข้อนี้ตอบได้ตรง ๆ ว่าไม่มีอะไรเป็นพิเศษครับ แต่อยากให้น้อง ๆ ได้ระมัดระวังเรื่องเครื่องหมายวงเล็บเป็นพิเศษ ถ้าเราจำกันได้ในวันที่พี่เฉลยข้อนี้พี่ก็ยังทำผิดซะเองด้วย ฮ่าฮ่าฮ่า ฉะนั้นถ้าน้องทำข้อนี้แล้วได้ผลลัพธ์แปลก ๆ ก็ไม่ต้องตกใจไปนะครับ ลองตั้งสติแล้วตรวจให้ดี ๆ เป็นทางที่ดีที่สุด

(เริ่มของจริงหลังจากนี้...)

เริ่มจากเรื่องที่เตือนตอน Section แรกคือเรื่องของการใช้วงเล็บแล้ว อีกส่วนที่สำคัญไม่แพ้กันเลยนั่นคือการคิดเลขจาก Data Type ที่ต่างชนิดกัน ถ้าน้อง ๆ ที่สงสัยว่ามันต่างกันยังไงละก็สามารถเปิดดู 2 คลิปนี้ประกอบดูด้วยกันได้ครับ [คลิกข้อความนี้ได้เลย]

จากสูตรสมการข้างต้นดังต่อไปนี้

$$Power = \lceil (C^2 + \log_e(M+1)) \times (e^{(\frac{L}{C+1})}) \rceil$$

กำหนดให้ C, M และ L คือ Input ที่รับค่าเป็นจำนวนเต็ม $\{C, M, L\} \in Z^+$

ข้อแรก ที่พี่อยากขอแนะนำนั่นเป็นอย่างยิ่งในการทำข้อนี้คือลอง**เริ่มทำทีละนิพจน์**ก่อนครับ โดยที่พี่ขอแบ่งสมการนี้ออกเป็นทั้งหมด 3 นิพจน์ใหญ่ ๆ ดังนี้ โดยที่แต่ละนิพจน์พี่จะให้มันถูกสร้างขึ้นเป็นตัวแปรใหม่เพื่อนำไปคิด

นิพจน์ที่ 1: $(C^2 + \log_e(M+1)) = A$

นิพจน์ที่ 2: $\frac{L}{C+1} = B$

นิพจน์ที่ 3: $A \times e^B$

จะสังเกตได้ว่าถ้าเราแบ่งนิพจน์ออกมาทำทีละนิด ๆ จะทำให้เราสามารถเขียนสมการลงใน Code ได้ง่าย ตรวจสอบได้ดีขึ้นหากเราทำผิด

ข้อสอง ถ้าเรานำไปคิดเลขตรง ๆ สิ่งที่อาจจะเกิดขึ้นคือทำไมได้ค่าของตัวเลขที่ผิดปกติไปนะ **(ยังไม่สนใจการปัดเลขทศนิยม)** นั่นเป็นเพราะว่าวิธีการคิดเลขในแต่ละ Data Type มันต่างกันนั่นเอง !!! ลองย้อนไปดูไฟล์วิดีโอที่พี่แปะไว้ให้ในช่วงต้น ๆ นะครับ และแน่นอนว่าส่วนที่หลายคนจะมีปัญหาคือ **นิพจน์ที่ 2 กับ นิพจน์ที่ 3** ครับ สาเหตุเพราะว่าหากเราให้ตัวแปร C, M และ L เป็นจำนวนเต็มวิธีการคิดเลขของมันก็จะกลายเป็นจำนวนเต็มเช่นกันนั่นเอง และอีกอันที่มักจะพลาดกัน (รวมถึงพี่เอง) ก็คือวงเล็บนั่นเองครับ เรามาลองดูตารางถัดไปนะครับว่าลำดับแต่ละอันคิดเมื่อใด ดังต่อไปนี้ครับ

1	() [] -> . ::	Grouping, scope, array/member access
2	! ~ - + * & sizeof *type cast* ++x --x	(most) unary operations, sizeof and type casts
3	* / %	Multiplication, division, modulo
4	+ -	Addition and subtraction
5	<< >>	Bitwise shift left and right
6	< <= > >=	Comparisons: less-than, ...
7	== !=	Comparisons: equal and not equal
8	&	Bitwise AND
9	^	Bitwise exclusive OR
10	|	Bitwise inclusive (normal) OR
11	&&	Logical AND
12	||	Logical OR
13	?:	Conditional expression (ternary operator)
14	= += -= *= /= %= &= |= ^= <<= >>=	Assignment operators
15	,	Comma operator

สังเกตได้ว่าเครื่องหมายวงเล็บ (Parentheses) มีลำดับที่สูงกว่า / (หาร) และ + (บวก) นั่นเองครับถ้าเราเขียนโค้ดให้มันว่า

int B = L/C+1;

สิ่งที่จะเกิดขึ้นคือจะนำ L/C ก่อนเสมอแล้วจึงค่อยนำค่าหลังการเสร็จไปเพิ่มอีก 1 นั่นคือสิ่งที่ผิดหลักวิธีการคิดทางคณิตศาสตร์ของเศษส่วนในรูปแบบของ PEMDAS อย่างยิ่งครับวิธีการที่ถูกต้องคือทำให้ B = L/(C+1) เพื่อทำให้คิดเลขในทั้งเศษและส่วนก่อนจะหารกันในท้ายที่สุดครับ

ข้อสาม เรื่องการคำนวณของนิพจน์ที่ 3 หากเราประกาศให้ตัวแปรเป็นจำนวนเต็มการคำนวณค่าจึงจะเป็นจำนวนเต็มครับ ฉะนั้นในระหว่างการคิดคำนวณค่าโดยเฉพาะตรงที่เป็นเกี่ยวกับการคูณหรือการหารที่มักจะมีผลโดยตรงครับ

ท้ายที่สุดถ้าน้อง ๆ แหกกฎวิธีการประกาศตัวแปรที่เป็น int ตามที่กำหนดแต่ใช้ double หรือ float ก็สามารถใช้งานได้เพราะนั่นคือวิธีที่คิดเลขที่ถูกต้องที่สุดครับ แต่พี่อยากให้น้อง ๆ ได้ลองพบปัญหาของการทำ Type-casting และการใช้วงเล็บ ข้อนี้จึงได้ถือกำเนิดขึ้นมาเพื่อให้น้อง ๆ ได้ลองฝึกทำกันนั่นเองครับ สู้ ๆ นะครับน้อง ^ ^

ป.ล.อย่าลืมกันนะว่ามีวงเล็บปัดจำนวนทศนิยมคอยดักอยู่ อิอิ