เฉลยข้อสอบปลายภาควิชา 517111 การเขียนโปรแกรมคอมพิวเตอร์ 1

ภาคการศึกษาต้น ปีการศึกษา 2555 ผู้สอน อ.ดร.ภิญโญ แท้ประสาทสิทธิ์ ม.ศิลปากร

1. **ปัญหา** มาท่องสูตรคูณกันเถอะ (3 คะแนน)

   $73 \times 29 = ?$ ตอบ **2117**

   $65 \times 38 = ?$ ตอบ **2470**

   $46 \times 193 = ?$ ตอบ **8878**

2. **ปัญหา** พื้นฐานการดำเนินการทางตัวเลขในภาษาซี (3 คะแนน)

   double d = 34.56789;
   double e = 65.43211;

   printf("%.0lf", d); ตอบ **35**

   printf("%.1lf", d); ตอบ **34.6**

   printf("%.0lf", d + e); ตอบ **100**

3. A[2] = 4 + 1 = **5**
   A[4] = 16 + 9 + 4 + 1 = **30**
   s = A[0] + A[1] + A[2] + A[3] = 0 + 1 + 5 + 14 = **20**

4. ข้อนี้คนทำผิดมากเพราะเวลาที่คิดผลลัพธ์แต่ละบรรทัดไปเอาค่า i ที่ตามหลังมันมา แต่ค่า i ที่ต้องใช้คือค่า i ที่มาก่อนมันต่างหาก การคำนวณที่ถูกต้องถ้าจะอธิบายให้ง่ายก็จะต้องจัดโค้ดใหม่ให้สับสนน้อยลงเป็น

```
int A[] = {1, 2, 3, 4};
int B[] = {5, 7, 3, 2};
int i = 0; printf("%d\n", A[i]+B[3-i]);      ตอบ 1 + 2 = 3
i = 1;     printf("%d\n", A[i]+B[3-i]);      ตอบ 2 + 3 = 5
i = 2;     printf("%d\n", A[i]+B[3-i]);      ตอบ 3 + 7 = 10
i = 3;     printf("%d\n", A[i]+B[3-i]);      ตอบ 4 + 5 = 9
```

5. ข้อนี้จุดที่หลายคนสับสนคือการไล่ลูปแบบค่าย้อนหลัง แต่ถึงจะไล่ไปหน้าหรือย้อนหลังค่า row กับ col มันก็จะถูกส่งไปเก็บในอาเรย์สองมิติในที่ที่เหมาะสม และถ้าใครตีความหมายของลูปสองชั้นได้ก็จะพบว่าค่าที่เก็บไว้ในอาเรย์แต่ละ

ช่อง แท้จริงก็คือผลบวกของเลขตำแหน่งแถวและคอลัมน์นั่นเอง

```
row = 3;
printf("%d %d\n", A[row][1], A[row][2]);     ตอบ 4 5
row = 2;
printf("%d %d\n", A[row][1], A[row][2]);     ตอบ 3 4
row = 1;
printf("%d %d\n", A[row][1], A[row][2]);     ตอบ 2 3
row = 0;
printf("%d %d\n", A[row][1], A[row][2]);     ตอบ 1 2
```

6. ข้อนี้เราต้องไล่ค่าอย่างระวัดระวังและตั้งสติให้ดีว่าพารามิเตอร์แต่ละตัวที่ส่งไปมีค่าเท่าใด และค่าที่ส่งกลับมาเปลี่ยนตัวแปรตัวใด ความยุ่งยากตรงนี้ทำให้สับสนได้ง่ายมาก คะแนนในข้อนี้จึงค่อนข้างสูง

```
printf("%d\n", f(1, 2, 3));   ตอบ 11
x = f(x, y, x);
y = f(x, y, x);
z = f(x, y, z);
printf("%d\n", x);            ตอบ 1
printf("%d\n", y);            ตอบ 5
printf("%d\n", z);            ตอบ 15
```

7. ข้อนี้อาจจะดูหนักหนาในตอนแรก แต่ที่จริงมันต้องการวัดว่าเรามองปัญหาออกหรือเปล่าและนับจำนวนได้ดีเพียงใด และสำหรับอาเรย์ B เราดูออกหรือไม่ว่ามีช่องใดที่ค่าไม่เปลี่ยนเลย เช่น ช่อง [9][9] ของอาเรย์ B จะเหมือนเดิมตลอดโปรแกรม แต่ของ A จะถูกเปลี่ยนค่า

บรรทัดนี้ต้องการถามว่าผู้เข้าสอบทราบหรือไม่ว่าลูปวนไปกี่รอบ

```
printf("%d\n", sum1+sum2);    ตอบ 110
```

บรรทัดต่อมาถามเราว่าเราดูออกหรือไม่ว่าในช่องอาเรย์ที่เกี่ยวข้องกับการหาคำตอบแท้จริงมีแค่ไม่กี่ช่อง หากเราไล่ค่าย้อนหลังก็จะได้คำตอบอย่างรวดเร็ว เนื่องจากมีค่าในช่องอาเรย์แค่ไม่กี่ช่องที่เราต้องการทราบ คือ

A[2][3] → A[1][1] → A[0][0] นั่นคือ ถ้าต้องการทราบค่าของ A[2][3] ก็ต้องทราบค่า A[1][1] และถ้าต้องการทราบ

ค่า A[1][1] ก็ต้องทราบค่า A[0][0] ซึ่งค่า A[0][0] นั้นโจทย์ให้มาตั้งแต่แรก (แต่ก็ต้องคำนวณซ้ำอีกรอบ) ดังนั้นคำตอบคิดได้ตามลำดับดังนี้

`A[0][0] = 1` (ได้มาจากโจทย์กำหนด)

`A[0][0] = A[0/2] + A[0/2] + 1 = 1 + 1 = 2` (เพราะลูปทำการคำนวณค่าใน A ใหม่ทุกช่อง ช่องละหนึ่งครั้ง ดังนั้นต้องมีการเปลี่ยนแปลที่ช่อง A[0][0] ด้วยแน่ ๆ)

`A[1][1] = A[0][0] + 1 = 2 + 1 = 3`

`A[2][3] = A[1][1] + 1 = 3 + 1 = 4`

แต่ถ้าใครจะออกแรงคำนวณค่ามาหมดเลยมันก็เป็นไปได้เหมือนกัน แต่จะใช้เวลามากกว่ากันอย่างเห็นได้ชัดและเสี่ยงต่อการคำนวณที่ผิดพลาด การไล่ค่าย้อนหลังแบบเร็วที่กล่าวมาก่อนหน้านี้จะใช้บ่อยในวิชา data structure และ algorithm ซึ่งเป็นการวัดเชาว์ปัญญาแบบหนึ่งของเรา สำหรับการสอบในวิชาเขียนโปรแกรมนี้ ถ้าเรามองยังไม่ออกก็ไม่มีปัญหา เพราะเวลาที่ให้นั้นเพียงพอต่อผู้เข้าสอบแม้ว่าจะทำแบบตรง ๆ ด้วยการใส่ค่าเข้าไปในตารางอาเรย์ทั้งหมดก็ตาม (ถ้าใครลองทำแบบตรง ๆ จะพบว่าค่ามันซ้ำ ๆ กันคำนวณไม่ยากเท่าไหร่)

`printf("%d\n", A[2][3]);` ตอบ **4**

พอมาถึงการคำนวณ A[8][9] หลายคนก็คงจะตกใจว่า 'นี่กะให้ใช้แรงเข้ากระทำหาคำตอบหรือเนี่ย' แต่วิธีทำที่รวดเร็วนั้นจะคำนวณไม่กี่รอบคือ A[8][9] → A[4][4] → A[2][2] → A[1][1] → A[0][0] ซึ่งเราก็รู้ค่าของ A[0][0] และ A[1][1] เรียบร้อยมาก่อนหน้าแล้ว ดังนั้นคำตอบของเราก็คือ A[2][2] = A[1][1] + 1 = 4 และ A[4][4] = A[2][2] + 1 = 5 และ A[8][9] = A[4][4] + 1 = 6 ดังนั้น

`printf("%d\n", A[8][9]);` ตอบ **6**

อันต่อมาที่จริงผมคิดจะถามถึง A[9][9] ซึ่งคำตอบจะเป็นแบบเดียวกับ A[8][9] แต่เกิดพิมพ์ผิดก็เลยมีการถามถึง

B[9][9] สองครั้ง ซึ่งก็ถือว่าน่าจะง่ายขึ้น เพราะจากลูป เราแก้ค่า B จากการคำนวณ B[row/2][col/2] ซึ่งไม่มีทางไปถึง B[9][9] นั่นคือค่า B[9][9] เคยเป็นอย่างไรก่อนเข้าลูป หลังจบลูปก็จะเป็นค่าเดิมแน่นอน ดังนั้น

```
printf("%d\n", B[9][9]);        ตอบ -2
```

อันดับสุดท้ายเป็นการหาค่า B[2][3] ดูเหมือนจะไม่มีอะไร แต่ปัญหามีอยู่ว่ามีมากกว่าหนึ่งครั้งที่ช่องอาเรย์ B[2][3] ถูกเปลี่ยนค่า นั่นคือ row = 4, col =6; row = 4, col = 7; row = 5, col = 6 และ row = 5, col = 7 การจะหาค่าที่ถูกต้อง ต้องดูจากครั้งสุดท้าย ซึ่งก็คือ row = 5, col = 7 ดังนั้น B[2][3] = A[5][7] + 1;

การคำนวณก็ทำแบบเดิมคือ A[5][7] → A[2][3] → A[1][1] → A[0][0]

A[2][3] = 3 + 1 = 4 และ A[5][7] = 4 + 1 = 5 ดังนั้น A[5][7] = 5 และ B[2][3] = 6

```
printf("%d\n", B[2][3]);        ตอบ 6
```

เรื่องน่าสนใจของข้อนี้มีอยู่ว่า ถ้าเราไปหาค่า B[2][3] จาก row = 4, col = 6 ก็ให้ผลลัพธ์ที่เท่ากัน แต่เราอย่าเข้าใจผิดไปว่ามันจะเป็นแบบนี้เสมอ การที่มันได้ค่าเท่ากันในข้อนี้เป็นเรื่องที่เจาะจงกับปัญหาข้อนี้ กรณีทั่ว ๆ ไปค่ามันจะต่างกัน

ส่วนคำถามย่อยสุดท้ายซ้ำกันอย่างน่าเศร้า

```
printf("%d\n", B[9][9]);        ตอบ -2
```

8. ข้อนี้ต้องการวัดว่าเราอ่านโค้ดเกี่ยวกับสตรัคออกหรือไม่ เราเข้าใจพฤติกรรมว่าค่าถูกใส่เข้าไปอย่างไรหรือไม่ ซึ่งค่าที่ใส่เข้าไปก็จะเป็นค่าซ้ำ ๆ กันหมด สำหรับ lab_scores ก็จะเป็น 4 ส่วนค่า midterm_score กับ final_score จะเป็น 30 เท่ากันทั้งคู่สำหรับ A ทุกช่องข้อมูล

ส่วนลูปที่สองเป็นการรวมคะแนนทุกอย่างเข้าด้วยกัน ซึ่งก็จะเป็น 4 x 5 + 30 + 30 = 80 สำหรับ A ช่องหนึ่ง ๆ และถ้าเอา A ทุกช่องมารวมกันก็จะเป็น 80 x 5 = 400

```
printf("%d\n", A[3].sum_score);     ตอบ 80
```

```
printf("%d", what);
```
ตอบ **400**

9. ข้อนี้วัดว่าเราเห็นสภาพแวดล้อมของโค้ดแล้วเรามองออกหรือไม่ว่าตัวแปรแต่ละตัวที่กำหนดมามันเอาไว้ทำอะไร และสร้างข้อกำหนดในส่วนที่เหลืออย่างไร โดยเฉพาะคำสั่ง return answer; ซึ่งสร้างข้อกำหนดเรื่องการตั้งชื่อตัวแปรในตอนต้น ว่าจะต้องเป็น int answer = 2000000000; (ค่าตัวเลขจะมากกว่านี้ก็ได้ แต่ถ้าเกินกว่า 2^{31} จะผิด)

```
int min(int* arData, int N) {
    int i;
    int answer = 2000000000;    // or INT_MAX;
    for(i = N-1; i >= 0; --i) {
        if(arData[i] < answer) {
            answer = arData[i];
        }
    }
    return answer;
}
```

10. ข้อนี้ต้องการทดสอบว่าเราสามารถตั้งเงื่อนไข if ที่ซับซ้อนขึ้นมาจากเดิมได้หรือเปล่า โดยเฉพาะวิธีการเลือกเอาแต่จำนวนที่เป็นค่าบวก (การสลับลำดับเงื่อนไขใน if ไม่มีผลต่อคะแนน)

```
int min_positive(int* arData, int N) {
    int i;
    int answer = 2000000000;
    for(i = 0; i < N; ++i) {
        if(arData[i] < answer && arData[i] > 0) {
            answer = arData[i];
        }
    }
    return answer;
}
```

11. ข้อนี้ต้องการวัดว่านักศึกษาเข้าใจวิธีการใช้อาเรย์เพียงพอหรือไม่ ทั้งในแง่การเก็บข้อมูลและดึงเอาค่าในอาเรย์สองอันมาใช้งานร่วมกัน ซึ่งการนำค่าอาเรย์สองอันขึ้นไปมาใช้งานร่วมกันเป็นกระบวนการที่พบบ่อยทั้งในงานฐานข้อมูลและการคำนวณทางวิทยาศาสตร์หลายอย่าง

```
int A[10], B[10];
int i;
for(i = 0; i < 10; ++i) {
    scanf("%d", &A[i]);
}
for(i = 0; i < 10; ++i) {
    scanf("%d", &B[i]);
}

for(i = 0; i < 5; ++i) {
    printf("(%d, %d) ", A[i], B[i]);
}
printf("\n");
for(i = 5; i < 10; ++i) {
    printf("(%d, %d) ", A[i], B[i]);
}
```

12. ข้อนี้ต้องการทดสอบความยืดหยุ่นในกระบวนการคิดของผู้เข้าสอบว่าจะสามารถจัดกระบวนการคิดในอีกรูปแบบหนึ่งได้หรือไม่ เพราะมือใหม่จำนวนมากจะทำแบบข้อที่ผ่านมาเพราะรูปแบบการคิดของการจัดการ A กับ B ในข้อที่แล้วมันจะคล้ายกัน คิดจัดการ A ได้ B ก็จะตามมาทันที แต่ข้อนี้จะต้องคิดได้ทั้งสองแบบ ส่วนการกลับค่าตอนไล่ลูปเป็นการทดสอบว่าผู้เข้าสอบรู้หรือไม่ว่าจะต้องปรับการควบคุมอาเรย์อย่างไร ซึ่งแท้จริงมันไม่ได้ส่งผลแค่ตรงต้นขั้วลูป แต่มันส่งผลตามมาที่การหยิบค่าใน A มาใช้ด้วย เพราะเราจะเขียนว่า A[i] ไม่ได้อีกต่อไป แต่ต้องเป็น A[10-i] แทน

(รู้สึกว่าผมจะทำช่องเส้นประและทึบสำหรับใส่คำตอบและกำหนดคะแนนสลับลำดับกัน เพราะ A[10-i] ถือว่าทำผิดได้ง่ายมาก และคะแนนวควรจะสูงกว่าการกำหนดรูปแบบการพิมพ์ซึ่งถามในข้อที่ 11 ไปแล้ว)

ประโยชน์ของวิธีคิดแบบข้อ 12 ในทางปฏิบัติก็คือว่าเราประหยัดหน่วยความจำในการเก็บข้อมูลไปประมาณครึ่งหนึ่ง ซึ่งความประหยัดนี้ยังเป็นสิ่งที่จำเป็นสำหรับการเขียนโปรแกรมสำหรับฮาร์ดแวร์ที่มีหน่วยความจำน้อย เช่น โทรศัพท์มือถือ หรือ ระบบฝังตัวบางชนิด

```
int A[10];  int i;
for(i = 0; i < 10; ++i) {
    scanf("%d", &A[i]);
}

int x;
for(i = 10; i > 0; --i) {
    scanf("%d", &x);
    printf("(%d, %d) ", A[10-i], x);
    if(i == 6) {
        printf("\n");
    }
}
```

13. ข้อนี้ต้องการวัดความสามารถในการใช้งานฟังก์ชันทางคณิตศาสตร์พื้นฐานในภาษาซี วิธีเขียนก็จะมีอยู่สองแบบหลัก ๆ โดยแบบแรกนี้เป็นแบบที่ง่ายและนิยมกว่า (บรรทัดที่เกินมาอย่าง printf มีไว้เพื่อใช้ในการทดสอบโปรแกรม แต่ในข้อสอบถูกตัดออกไป บางทีเจอของที่เกินออกมาบ้างก็ไม่ต้องประหลาดใจ)

```
#include <stdio.h>
#include <math.h>
int main() {
    double side1, side2, side3;
    scanf("%lf %lf", &side1, &side2);
    side3 = sqrt(side1*side1 + side2*side2);
    printf("%lf", side3);
```

```
    return 0;
}
```

ส่วนแบบที่สองเป็นแบบที่ยากขึ้นมา เพราะใช้คำสั่ง pow ซึ่งในทางปฏิบัติเราไม่มีความจำเป็นที่จะต้องทำอะไรให้ยุ่งยากเช่นนี้ แต่ในเมื่อเป็นคำตอบที่ถูกก็ไม่ว่ากัน ส่วนใครที่มาด้วยวิธีประหลาดผสมการคูณกับ pow ทางผมก็ให้คะแนนเช่นกัน แค่ผมเข้าใจว่าไม่มีใครตอบแบบนั้นมา

```
#include <stdio.h>
#include <math.h>
int main() {
    double side1, side2, side3;
    scanf("%lf %lf", &side1, &side2);
    side3 = sqrt(pow(side1, 2) + pow(side2, 2));
    printf("%lf", side3);
    return 0;
}
```

14. ข้อนี้ต้องการวัดความสามารถในการจัดการสตริงและอักขระ ทั้งในเรื่องของที่เก็บข้อมูล คำสั่งพื้นฐานในการจัดการข้อมูลเข้า การหาความยาวสตริง และความเข้าใจเรื่องรหัสแอสกี

```
#include <stdio.h>
#include <string.h>
int main() {
    char msg[1025]; // จะใส่ค่าที่มากกว่า 1025 ก็ถือว่าถูกต้อง
    gets(msg);
    int length = strlen(msg);
    int i;
    for(i = 0; i < length; ++i) {
        if(msg[i] >= 'a' && msg[i] <= 'y') { // ใช้ msg[i] < 'z' ก็ได้
                                             // ใช้รหัสแอสคีมาลุยก็ยังได้ (ถ้าจำเลขถูก)
            msg[i] = msg[i] + 1; // ค่าอักขระทำตัวเป็นตัวเลขธรรมดาได้เสมอในภาษาซี
```

```
        } else if(msg[i] == 'z') {
            msg[i] = 'a';
        } else if(msg[i] >= 'A' &&
                  msg[i] <= 'Y') { // ใช้ msg[i] < 'Z' หรือรหัสแอสคีก็ได้
            msg[i] = msg[i] + 1;
        } else if(msg[i] == 'Z') {
            msg[i] = 'A';
        }
    }
    printf("%s", msg);
    return 0;
}
```