รายงาน

Array 3D

จัดทำโดย

นายธนรัก ชุ่มสวัสดิ์

รหัสศึกษา 68543210018-6

อาจารย์ผู้สอน

นายปิยพล ยืนยงสถาวร

รายงานฉบับนี้เป็นส่วนหนึ่งของวิชา Data Structures and Algorithms

สาขาวิชาวิศวกรรมซอฟต์แวร์ มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา

ภาคเรียนที่ 2 ปีการศึกษา 2568

Code Program Stack

```c
#include <stdio.h>   // use printf()
#include <termios.h> // for custom getch() on macOS
#include <unistd.h>  // for STDIN_FILENO
#include <stdio.h>
#include <stdlib.h>

#define l 1
#define u 5

#define l1 1
#define u1 3

#define l2 1
#define u2 4

#define l3 1
#define u3 5

int *BA1, *BA2, *BA3, *p;
int i, j, k;

/* ---------- 1D ---------- */
void Create1DArray()
{
    int element = (u - l + 1);
    int c = sizeof(*BA1);
    int total_mem = element * c;
    BA1 = (int *)malloc(total_mem);
    if (!BA1)
    {
        perror("malloc BA1");
        exit(EXIT_FAILURE);
    }
}

void A1(int i, int x)
{
    p = BA1 + (i - l);
    *p = x;
```

```
}

int ReadA1(int i)
{
    p = BA1 + (i - l);
    return *p;
}

/* ---------- 2D ---------- */
void Create2DArray()
{
    int element = (u1 - l1 + 1) * (u2 - l2 + 1);
    int c = sizeof(*BA2);
    int total_mem = element * c;
    BA2 = (int *)malloc(total_mem);
    if (!BA2)
    {
        perror("malloc BA2");
        exit(EXIT_FAILURE);
    }
}

void A2(int i, int j, int x)
{
    p = BA2 + ((i - l1) * (u2 - l2 + 1) + (j - l2));
    *p = x;
}

int ReadA2(int i, int j)
{
    p = BA2 + ((i - l1) * (u2 - l2 + 1) + (j - l2));
    return *p;
}

/* ---------- 3D ---------- */
/* Create same memory buffer for 3D (linearized) */
void Create3DArray()
{
    int element = (u1 - l1 + 1) * (u2 - l2 + 1) * (u3 - l3 + 1);
```

```
    int c = sizeof(*BA3);
    int total_mem = element * c;
    BA3 = (int *)malloc(total_mem);
    if (!BA3)
    {
        perror("malloc BA3");
        exit(EXIT_FAILURE);
    }
}

/* Method 1 (เดิม) - row-major แบบ i เป็น major (i เปลี่ยนช้าที่สุด) */
void A3_rowMajor(int i, int j, int k, int x)
{
    int n2 = (u2 - l2 + 1);
    int n3 = (u3 - l3 + 1);
    p = BA3 + ((i - l1) * n2 * n3 + (j - l2) * n3 + (k - l3));
    *p = x;
}

int ReadA3_rowMajor(int i, int j, int k)
{
    int n2 = (u2 - l2 + 1);
    int n3 = (u3 - l3 + 1);
    p = BA3 + ((i - l1) * n2 * n3 + (j - l2) * n3 + (k - l3));
    return *p;
}

/* Method 2 (ใหม่) - column-major แบบ i เป็น fastest (i เปลี่ยนเร็วสุด)
   mapping index = (k-l3) * (n1*n2) + (j-l2) * n1 + (i-l1)
   โดย n1 = u1-l1+1, n2 = u2-l2+1, n3 = u3-l3+1
*/
void A3_colMajor(int i, int j, int k, int x)
{
    int n1 = (u1 - l1 + 1);
    int n2 = (u2 - l2 + 1);
    /* index = (k-l3)*n1*n2 + (j-l2)*n1 + (i-l1) */
    p = BA3 + ((k - l3) * n1 * n2 + (j - l2) * n1 + (i - l1));
    *p = x;
}
```

```
int ReadA3_colMajor(int i, int j, int k)
{
    int n1 = (u1 - l1 + 1);
    int n2 = (u2 - l2 + 1);
    p = BA3 + ((k - l3) * n1 * n2 + (j - l2) * n1 + (i - l1));
    return *p;
}

int main()
{
    printf("1-3 DIMENSION ARRAY FUNCTION (with 2 methods for 3D)...\n");
    printf("============================================================\n");

    Create1DArray();
    Create2DArray();
    Create3DArray();

    /* 1D test */
    i = 2;
    A1(i, 9);
    printf("\nA1(%d) = %d\n", i, ReadA1(i));

    /* 2D test */
    i = 2;
    j = 3;
    A2(i, j, 99);
    printf("A2(%d,%d) = %d\n", i, j, ReadA2(i, j));

    /* 3D test - method 1 (เดิม row-major) */
    i = 3;
    j = 4;
    k = 5;
    A3_rowMajor(i, j, k, 999);
    printf("A3_rowMajor(%d,%d,%d) = %d\n", i, j, k, ReadA3_rowMajor(i, j, k));

    /* 3D test - method 2 (ใหม่ col-major) */
    /* note: เราเขียนตำแหน่งอื่นเพื่อไม่ให้ทับค่าเดิมของตัวอย่างข้างบน
       ตัวอย่างนี้แสดงการเขียนและอ่านโดยใช้วิธีใหม่ */
    i = 1;
```

```
    j = 1;
    k = 1;
    A3_colMajor(i, j, k, 111);
    printf("A3_colMajor(%d,%d,%d) = %d\n", i, j, k, ReadA3_colMajor(i, j, k));

    /* แสดงว่า ถ้าเขียนด้วยวิธีหนึ่ง แล้วอ่านด้วยอีกวิธีก็จะได้ค่าต่างกัน (เพราะ mapping ต่างกัน) */
    printf("\n(หมายเหตุ: BA3 เป็น buffer เดียวกัน — mapping สองแบบต่างกันจะเข้าถึงตำแหน่งต่างกัน)\n");

    printf("\nPress ENTER to exit...");
    getchar();

    free(BA1);
    free(BA2);
    free(BA3);

    return 0;
}
```

สรุปสิ่งที่เพิ่มเติม

```
void A3_colMajor(int i, int j, int k, int x)
{
    int n1 = (u1 - l1 + 1);
    int n2 = (u2 - l2 + 1);
    /* index = (k-l3)*n1*n2 + (j-l2)*n1 + (i-l1) */
    p = BA3 + ((k - l3) * n1 * n2 + (j - l2) * n1 + (i - l1));
    *p = x;
}

int ReadA3_colMajor(int i, int j, int k)
{
    int n1 = (u1 - l1 + 1);
    int n2 = (u2 - l2 + 1);
    p = BA3 + ((k - l3) * n1 * n2 + (j - l2) * n1 + (i - l1));
    return *p;
}
```

เพิ่ม A3_colMajor , ReadA3_colMajor ซึ่งเก็บ/อ่านโดยใช้ลำดับดัชนีแบบ column-major

อธิบายการทำงานของโปรแกรม

ส่วนประกาศตัวแปรและค่าเริ่มต้น

ในส่วนนี้โปรแกรมเริ่มจากการ include ไลบรารี stdio.h และ stdlib.h เพื่อใช้ฟังก์ชันแสดงผลและจองหน่วยความจำ ต่อมามีการกำหนดค่าคงที่ด้วย #define เพื่อระบุขอบเขตล่าง–บนของอาเรย์แต่ละมิติ ซึ่งช่วยให้ฟังก์ชันต่าง ๆ คำนวณตำแหน่งจริงในหน่วยความจำได้อย่างถูกต้อง หลังจากนั้นประกาศ pointer BA1, BA2, BA3 เพื่อเก็บตำแหน่งเริ่มต้นของพื้นที่อาเรย์ 1D, 2D, และ 3D ที่จะถูกสร้างขึ้นด้วย malloc() โดย pointer p ใช้ช่วยคำนวณตำแหน่ง offset ภายในอาเรย์เมื่ออ่านหรือเขียนค่า ส่วนตัวแปร i, j, k เป็นตัวแปรดัชนีที่ใช้ส่งค่าระหว่าง main() และฟังก์ชันจัดการอาเรย์ทั้งหมด ทำให้โปรแกรมสามารถจัดการข้อมูลหลายมิติได้อย่างเป็นระบบ.

```
#include <stdio.h>
#include <stdlib.h>

#define l 1
#define u 5

#define l1 1
#define u1 3

#define l2 1
#define u2 4

#define l3 1
#define u3 5

int *BA1, *BA2, *BA3, *p;
int i, j, k;
```

ฟังก์ชัน Create1DArray()

ฟังก์ชันนี้มีหน้าที่สร้างอาเรย์หนึ่งมิติแบบไดนามิก โดยคำนวณจำนวนสมาชิกที่ต้องการจากขอบเขตดัชนี l ถึง u จากนั้นนำจำนวนสมาชิกมาคูณกับขนาดของข้อมูลชนิด int เพื่อหาเนื้อที่หน่วยความจำทั้งหมดที่ต้องใช้ แล้วจึงเรียก malloc() เพื่อจองพื้นที่ใน heap และเก็บตำแหน่งเริ่มต้นลงในตัวแปร global BA1 ซึ่งจะถูกใช้โดยฟังก์ชันเขียนและอ่านข้อมูล (A1 และ ReadA1) ต่อไป ดังนั้นจุดเชื่อมต่อของฟังก์ชันนี้คือการสร้างหน่วยความจำให้พร้อมสำหรับฟังก์ชันถัดไป และเมื่อฟังก์ชันทำงานเสร็จ โปรแกรมจะกลับไปที่ main() เพื่อเรียกฟังก์ชันจัดการข้อมูลตัวถัดไปโดยใช้ BA1 ที่ถูกสร้างแล้ว

```
void Create1DArray()
{
   int element = (u - l + 1);
   int c = sizeof(*BA1);
   int total_mem = element * c;
   BA1 = (int *)malloc(total_mem);
   if (!BA1)
   {
      perror("malloc BA1");
      exit(EXIT_FAILURE);
   }
}
```

ฟังก์ชัน A1(int i, int x)

ฟังก์ชัน A1 ทำหน้าที่เขียนค่าลงในอาเรย์หนึ่งมิติ โดยเริ่มจากคำนวณตำแหน่งจริงของข้อมูลในหน่วยความจำด้วยการนำดัชนี i ไปลบด้วยขอบเขตเริ่มต้น l เพื่อคำนวณ offset ภายในบัฟเฟอร์ BA1 จากนั้นกำหนด pointer p ให้ชี้ไปยังตำแหน่งนั้นและเขียนค่า x ลงไป การทำงานนี้ไม่มีการคืนค่า แต่เกิดผลกระทบโดยตรงต่อเนื้อหาใน BA1 ซึ่งจะถูกฟังก์ชัน ReadA1 ใช้อ่านต่อไป จึงถือว่านี่คือจุดเชื่อมต่อที่ข้อมูลถูก

ปรับเปลี่ยนเพื่อให้ฟังก์ชันอื่นสามารถเข้าถึงค่าที่ update แล้ว หลังเสร็จฟังก์ชันการควบคุมโปรแกรมจะกลับไปยัง main() เพื่อดำเนินการเรียกฟังก์ชันอื่น เช่น ReadA1 หรือการพิมพ์ผลลัพธ์

```
void A1(int i, int x)
{
   p = BA1 + (i - l);
   *p = x;
}
```

ฟังก์ชัน ReadA1(int i)

ฟังก์ชัน ReadA1 ใช้สำหรับอ่านค่าจากอาเรย์หนึ่งมิติ โดยคำนวณตำแหน่งจริงของข้อมูลจากดัชนี i แบบเดียวกับ A1 และกำหนด pointer p ให้ชี้ไปยังตำแหน่งภายใน BA1 ก่อนจะคืนค่าที่อ่านได้กลับไปยังผู้เรียก ฟังก์ชันนี้จึงเป็นจุดเชื่อมต่อเชิงข้อมูลที่สำคัญ เพราะมันส่งค่าที่เก็บในอาเรย์ไปให้โค้ดส่วนอื่น เช่นส่วนแสดงผลใน main() และหลังจากคืนค่านี้ โปรแกรมจะกลับไปเดินต่อใน main() เพื่อพิมพ์ผลหรือดำเนินการอื่นต่อไป โดย ReadA1 ไม่เปลี่ยนแปลงข้อมูลใด ๆ แต่เป็นตัวกลางเชื่อมระหว่างข้อมูลภายในหน่วยความจำและฟังก์ชันระดับสูงที่ต้องการใช้งานข้อมูลนั้น

```
int ReadA1(int i)
{
   p = BA1 + (i - l);
   return *p;
}
```

ฟังก์ชัน Create2DArray()

ฟังก์ชันนี้จองหน่วยความจำสำหรับอาเรย์สองมิติ ซึ่งในความจริงถูกเก็บเป็นอาเรย์หนึ่งมิติในหน่วยความจำ โดยเริ่มจากคำนวณจำนวนสมาชิกทั้งหมดด้วยการคูณจำนวนแถว (u1 - l1 + 1) กับจำนวนคอลัมน์ (u2 - l2 + 1) จากนั้นคูณกับขนาดข้อมูลชนิด int และเรียก malloc() เพื่อจัดสรรพื้นที่ แล้วเก็บ pointer ที่ได้ลงใน global BA2 ซึ่งเป็นจุดเชื่อมต่อสำหรับฟังก์ชัน A2 และ ReadA2 ที่จะใช้เพื่อเขียนและอ่านข้อมูลต่อไป เมื่อฟังก์ชันเสร็จสิ้น การควบคุมโปรแกรมจะย้อนกลับไปยัง main() เพื่อให้เรียกฟังก์ชันจัดการข้อมูลสองมิติต่อ

```
void Create2DArray()
```

```
{
    int element = (u1 - l1 + 1) * (u2 - l2 + 1);
    int c = sizeof(*BA2);
    int total_mem = element * c;
    BA2 = (int *)malloc(total_mem);
    if (!BA2)
    {
        perror("malloc BA2");
        exit(EXIT_FAILURE);
    }
}
```

ฟังก์ชัน A2(int i, int j, int x)

ฟังก์ชัน A2 เขียนค่า x ลงในอาเรย์สองมิติ โดยใช้การคำนวณตำแหน่งแบบ row-major ซึ่งแปลงดัชนี (i,j) ให้เป็นตำแหน่งหนึ่งมิติในบัฟเฟอร์ BA2 จากนั้น pointer p จะถูกกำหนดให้ไปยังตำแหน่งนั้นเพื่อนำค่า x ไปเก็บ ฟังก์ชันนี้ไม่คืนค่าใด ๆ แต่ทำให้ข้อมูลภายใน BA2 ถูกปรับปรุง ซึ่งเป็นข้อมูลที่ฟังก์ชัน ReadA2 จะต้องอ่านในภายหลัง ดังนั้นจุดเชื่อมต่อสำคัญคือการอัปเดตข้อมูลที่ส่งผลโดยตรงต่อผลลัพธ์ของการอ่าน และหลังจากการเขียนเสร็จ โปรแกรมควบคุมจะกลับไปที่ main() เพื่อให้ดำเนินการขั้นตอนถัดไป เช่นอ่านค่าหรือพิมพ์ผลลัพธ์

```
void A2(int i, int j, int x)
{
    p = BA2 + ((i - l1) * (u2 - l2 + 1) + (j - l2));
    *p = x;
}
```

ฟังก์ชัน ReadA2(int i, int j)

ฟังก์ชันนี้อ่านข้อมูลจากอาเรย์สองมิติ โดยคำนวณตำแหน่งในลักษณะเดียวกับ A2 แล้วคืนค่าจากตำแหน่งนั้นกลับไปยังผู้เรียก การทำงานนี้เป็นจุดเชื่อมต่อที่สำคัญเพราะมันส่งข้อมูลที่ถูกเก็บใน BA2 ไปยังโค้ดระดับบน เช่น main() เพื่อใช้ในการแสดงผลหรือดำเนินงานอื่น ฟังก์ชันนี้ไม่ได้ปรับข้อมูลใด ๆ ใน

หน่วยความจำ แต่เป็นตัวกลางเชื่อมให้ข้อมูลภายในอาเรย์สามารถถูกส่งออกไปยังส่วนควบคุมโปรแกรม เมื่อฟังก์ชันเสร็จโปรแกรมจะกลับไปสู่ main() เพื่อทำงานตามลำดับ เช่นพิมพ์ค่าที่อ่านได้

```
int ReadA2(int i, int j)
{
   p = BA2 + ((i - l1) * (u2 - l2 + 1) + (j - l2));
   return *p;
}
```

ฟังก์ชัน Create3DArray()

ฟังก์ชันนี้ใช้จองพื้นที่หน่วยความจำสำหรับอาเรย์สามมิติ โดยคำนวณจำนวนสมาชิกทั้งหมดในสามมิติแล้วนำไปคูณกับขนาด int ก่อนเรียก malloc() เพื่อสร้างพื้นที่จริงใน heap พื้นที่ที่ได้ถูกเก็บลงใน pointer global BA3 ซึ่งเป็นจุดเชื่อมต่อหลักสำหรับฟังก์ชันจัดการข้อมูลสามมิติทั้งหมด (A3 และ ReadA3) เนื่องจากอาเรย์สามมิติจริง ๆ ถูกเก็บต่อเนื่องแบบอาเรย์หนึ่งมิติ การสร้างพื้นที่นี้จึงจำเป็นต้องเกิดก่อนทุกครั้งที่มีการเขียนหรืออ่านข้อมูลสามมิติ และเมื่อฟังก์ชันเสร็จการจองพื้นที่แล้ว โปรแกรมจะย้อนกลับไปที่ main() เพื่อเรียกฟังก์ชันที่เขียนค่าทดสอบในอาเรย์สามมิติถัดไป

```
void Create3DArray()
{
   int element = (u1 - l1 + 1) * (u2 - l2 + 1) * (u3 - l3 + 1);
   int c = sizeof(*BA3);
   int total_mem = element * c;
   BA3 = (int *)malloc(total_mem);
   if (!BA3)
   {
      perror("malloc BA3");
```

```
        exit(EXIT_FAILURE);
    }
}
```

ฟังก์ชัน A3_rowMajor(int i, int j, int k, int x)

ฟังก์ชันนี้ทำหน้าที่เขียนค่าใน BA3 โดยใช้การแม็ปรูปแบบ row-major (ตามโค้ดเดิมแต่แยกชัดเป็นชื่อฟังก์ชัน) เมื่อคำนวณ index แล้วมันเขียนค่า x ลงใน BA3[idx] โดยไม่มีการคืนค่า ทำให้จุดเชื่อมต่อคือการอัปเดตเนื้อหาใน BA3 ที่ ReadA3_rowMajor หรือโค้ดอื่นสามารถอ่านต่อได้; หลังการเขียนการควบคุมจะกลับไปยัง caller (เช่น main()), ซึ่งมักจะเรียก ReadA3_rowMajor เพื่อตรวจหรือพิมพ์ค่าที่เพิ่งเขียน

```
void A3_rowMajor(int i, int j, int k, int x)
{
    int n2 = (u2 - l2 + 1);
    int n3 = (u3 - l3 + 1);
    p = BA3 + ((i - l1) * n2 * n3 + (j - l2) * n3 + (k - l3));
    *p = x;
}
```

ฟังก์ชัน ReadA3_rowMajor(int i, int j, int k)

ฟังก์ชันนี้อ่านค่าจาก BA3 โดยใช้การแม็ปแบบ row-major เดียวกับ A3_rowMajor แล้วคืนค่าที่อ่านกลับไปยัง caller การคืนค่านี้เป็นจุดเชื่อมต่อหลักเพราะตัวเรียก (เช่น main()) จะรับค่านั้นไปพิมพ์หรือประมวลผลต่อ ดังนั้น ReadA3_rowMajor ทำหน้าที่เป็นสื่อกลางที่ดึงข้อมูลจากบัฟเฟอร์และส่งต่อเป็นผลลัพธ์เชิงข้อมูลกลับไปให้ส่วนควบคุมของโปรแกรม

```
int ReadA3_rowMajor(int i, int j, int k)
{
    int n2 = (u2 - l2 + 1);
    int n3 = (u3 - l3 + 1);
    p = BA3 + ((i - l1) * n2 * n3 + (j - l2) * n3 + (k - l3));
    return *p;
}
```

ฟังก์ชัน A3_colMajor(int i, int j, int k, int x)

ฟังก์ชันนี้เป็นอีกวิธีแม็ป 3D ⟶ 1D แบบ column-major (ซึ่งในตัวอย่างผมเลือกให้ i เป็น fastest) โดยคำนวณ index ใหม่และเขียนค่า x ลงใน BA3 ตาม index นั้น จุดเชื่อมต่อคือการอัปเดตบัฟเฟอร์ BA3 ในตำแหน่งที่แตกต่างจาก row-major ดังนั้นการเขียนด้วยวิธีนี้จะถูกอ่านอย่างถูกต้องเฉพาะเมื่อใช้ ReadA3_colMajor อ่านต่อ ถ้าใช้การอ่านคนละวิธีจะได้ค่าผิดเพี้ยน แสดงให้เห็นว่าจุดเชื่อมต่อไม่เพียงเรื่องข้อมูล แต่ยังเกี่ยวข้องกับการตกลงกันในรูปแบบแม็ปของผู้เรียกและผู้ถูกเรียก

```
void A3_colMajor(int i, int j, int k, int x)
{
   int n1 = (u1 - l1 + 1);
   int n2 = (u2 - l2 + 1);
   /* index = (k-l3)*n1*n2 + (j-l2)*n1 + (i-l1) */
   p = BA3 + ((k - l3) * n1 * n2 + (j - l2) * n1 + (i - l1));
   *p = x;
}
```

ฟังก์ชัน ReadA3_colMajor(int i, int j, int k)

เมื่อถูกเรียกฟังก์ชันนี้จะคำนวณ index ตามสูตร column-major เพื่อชี้ไปยังตำแหน่งใน BA3 แล้วคืนค่าที่ตำแหน่งนั้นให้ caller บทบาทเชื่อมต่อของฟังก์ชันนี้คือเป็นคู่ของ A3_colMajor มันรับช่วง index จาก caller, ดึงข้อมูลจากบัฟเฟอร์ตามข้อตกลงแม็ป แล้วส่งค่ากลับไปสู่ส่วนควบคุมเพื่อแสดงหรือประมวลผลต่อ ดังนั้นทั้งสองฟังก์ชันต้องตกลง "สัญญาแม็ป" เดียวกันเพื่อให้การสื่อสารข้อมูลถูกต้อง

```
int ReadA3_colMajor(int i, int j, int k)
{
   int n1 = (u1 - l1 + 1);
   int n2 = (u2 - l2 + 1);
   p = BA3 + ((k - l3) * n1 * n2 + (j - l2) * n1 + (i - l1));
   return *p;
```

```
}
```

ฟังก์ชัน main()

ฟังก์ชัน main() เป็นจุดเริ่มต้นและตัวควบคุมลำดับการทำงานของโปรแกรม โดยเรียก Create1DArray(), Create2DArray(), Create3DArray() เพื่อจองหน่วยความจำก่อน แล้วสาธิตการใช้งานโดยเรียก A1/A2/A3 เพื่อเขียนค่าและ ReadA1/ReadA2/ReadA3 เพื่ออ่านค่าที่เขียนมาและพิมพ์ผลลงหน้าจอ ฟังก์ชันนี้เป็นตัวรับค่าที่ Read* คืนกลับมาและแสดงผลต่อให้ผู้ใช้เห็น นอกจากนี้ยังรอคีย์อินพุตด้วย getchar() เพื่อให้ผู้ใช้เห็นผลก่อนปิดหน้าต่าง และสุดท้ายเรียก free() ปลดหน่วยความจำของ BA1/BA2/BA3 ก่อนออกจากโปรแกรม ทำให้ main() ทำหน้าที่เป็น orchestrator ที่เชื่อมต่อการจัดสรร, การเขียน/อ่านข้อมูล และการทำความสะอาดทรัพยากรทั้งหมด

```
int main()
{
    printf("1-3 DIMENSION ARRAY FUNCTION (with 2 methods for 3D)...\n");
    printf("============================================================\n");

    Create1DArray();
    Create2DArray();
    Create3DArray();

    /* 1D test */
    i = 2;
    A1(i, 9);
    printf("\nA1(%d) = %d\n", i, ReadA1(i));

    /* 2D test */
    i = 2;
    j = 3;
    A2(i, j, 99);
    printf("A2(%d,%d) = %d\n", i, j, ReadA2(i, j));

    /* 3D test - method 1 (เดิม row-major) */
    i = 3;
    j = 4;
    k = 5;
    A3_rowMajor(i, j, k, 999);
```

```
    printf("A3_rowMajor(%d,%d,%d) = %d\n", i, j, k, ReadA3_rowMajor(i, j, k));

    /* 3D test - method 2 (ใหม่ col-major) */
    /* note: เราเขียนตำแหน่งอื่นเพื่อไม่ให้ทับค่าเดิมของตัวอย่างข้างบน
       ตัวอย่างนี้แสดงการเขียนและอ่านโดยใช้วิธีใหม่ */
    i = 1;
    j = 1;
    k = 1;
    A3_colMajor(i, j, k, 111);
    printf("A3_colMajor(%d,%d,%d) = %d\n", i, j, k, ReadA3_colMajor(i, j, k));

    /* แสดงว่า ถ้าเขียนด้วยวิธีหนึ่ง แล้วอ่านด้วยอีกวิธีก็จะได้ค่าต่างกัน (เพราะ mapping ต่างกัน) */
    printf("\n(หมายเหตุ: BA3 เป็น buffer เดียวกัน — mapping สองแบบต่างกันจะเข้าถึงตำแหน่งต่างกัน)\n");

    printf("\nPress ENTER to exit...");
    getchar();

    free(BA1);
    free(BA2);
    free(BA3);

    return 0;
}
```

ผลลัพธ์ของโปรแกรม

```
1-3 DIMENSION ARRAY FUNCTION (with 2 methods for 3D)...
============================================================

A1(2) = 9
A2(2,3) = 99
A3_rowMajor(3,4,5) = 999
A3_colMajor(1,1,1) = 111

(หมายเหตุ: BA3 เป็น buffer เดียวกัน — mapping สองแบบต่างกันจะเข้าถึงต่าแหน่งต่างกัน)
```