CP 111 Computer Programming (2/2554 Section B01-B02)

Handout 8

<u>หัวข้อ:</u>

Pointers and Arrays (Chapter 16 textbook)

- ทบทวน Pointers (ดู Handout 7 ด้วย)
- ความสัมพันธ์ระหว่าง arrays กับ pointers (ดู Handout 7 ด้วย)
- Problem solving with arrays: Insertion sort
- Multi-Dimensional Array (อาร์เรย์หลายมิติ)

<u>ทบทวน Pointers</u>

1. Basic concept of pointers: Pointer declaration, Address Operator, Indirection Operator

โจทย์ตัวอย่าง: จงเขียน statement ลงในโปรแกรมตามข้อความที่กำหนดไว้ในคำสั่งแต่ละข้อย่อย

```
int main()
{
int x;
```

(i) Pointer declaration: ทำการประกาศตัวแปร ptr1 เป็นแบบ pointer to int (เอาไว้สำหรับชี้ไปยังหรือเก็บแอดเดรสของตัวแปรแบบ int) และประกาศตัวแปร ptr2 เป็นแบบ pointer to pointer to int เอาไว้ชี้ไปยังตัวแปรแบบ pointer to int

(ii) Address operator: ทำการกำหนดให้ ptr1 เก็บค่าแอดเดรสของตัวแปร x หรือชี้ไปที่ตัวแปร x (ประกาศข้างบน) และกำหนดให้ ptr2 ชี้ไปที่ ptr1

(iii) Indirection operator: ทำการกำหนดค่าให้ตัวแปร x มีค่าเท่ากับ 0 โดยให้อ้างถึง x ผ่านตัวแปร ptr1 และทำการบวกเพิ่มค่าของ x ด้วย 1 โดยให้อ้างถึง x ผ่านตัวแปร ptr2

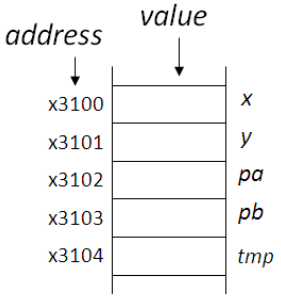
```
  return 0;
}
```

2. Pointer and function argument (pass by reference): เราสามารถใช้ตัวแปร pointer ในการส่งแอดเดรสของตัวแปรในฟังก์ชันผู้เรียกหรือ calling function (ในโปรแกรมข้างล่างคือฟังก์ชัน main) มาเป็นอาร์กิวเมนต์ของฟังก์ชันที่ถูกเรียกหรือCalled function (ฟังก์ชัน Sort2) ซึ่งจะทำให้ใน called function สามารถแก้ไขค่าของตัวแปรใน calling function ได้

โจทย์ตัวอย่าง: จงทำโปรแกรมข้างล่างให้สมบูรณ์เพื่อเขียนและเรียกใช้ฟังก์ชัน Sort2 ซึ่งจะทำการสลับค่าเพื่อทำให้ค่าของตัวแปรที่ถูกชี้โดยอาร์กิวเมนต์ pa น้อยกว่า ค่าของตัวแปรที่ถูกชี้ pb (*pa < *pb)

```
void Sort2(int *,int *);

int main()
{
int x,y;
       x=10; y=4;

       ...................................
       printf("%d %d\n",x,y);
       return 0;
}
```

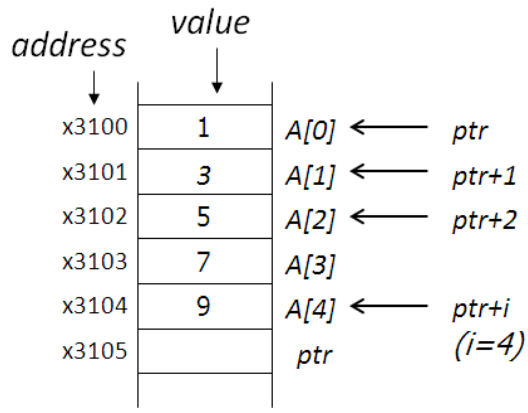
```
void Sort2(int *pa,int *pb)
{

}
```



<u>ความสัมพันธ์ระหว่าง arrays กับ pointers</u>

1. Key #1: ชื่อของตัวแปรแบบอาร์เรย์จะมีค่าเท่ากับแอดเดรสของ element แรกของอาร์เรย์

```
int A[5]={1,3,5,7,9};
int *ptr;
            ptr=A; จะเทียบเท่ากับ ptr=&A[0];
```



จากตัวอย่างข้างบน ptr จะชี้ไปที่ element แรก (array index=0) ของ A

2. Key #2: ถ้าตัวแปรแบบ pointer สมมติว่าคือ ptr ในตัวอย่างข้างบน ชี้ไปที่ element หนึ่งๆ ในอาร์เรย์แล้ว

`ptr+1` จะชี้ไปที่ <u>element ถัดจาก</u> element ที่ ptr ชี้อยู่

`ptr+2` จะชี้ไปที่ <u>2 element ถัดจาก</u> element ที่ ptr ชี้อยู่

`ptr+i` จะชี้ไปที่ <u>i element ถัดจาก</u> element ที่ ptr ชี้อยู่

ดังนั้น `*(ptr+1)`, `*(ptr+2)` และ `*(ptr+i)` จะหมายถึงค่าของอาร์เรย์ใน element ที่ `ptr+1`, `ptr+2` และ `ptr+i` กำลังชี้ไปอยู่

ในโค้ดข้างล่าง จะแสดง 1 3 5

```
int A[5]={1,3,5,7,9};
int *ptr;
int x,y,z;
     ptr=A;
     x=*ptr;      // x <- A[0]
     y=*(ptr+1); // y <- A[1]
     z=*(ptr+2);
     printf("%d %d %d\n",x,y,z);
```

เราอาจจะกำหนดให้ pointer ไปชี้ที่ element อื่นๆ ที่ ไม่ใช่ element แรกก็ได้ แต่ความหมาย ptr+1, ptr+2, ..ที่อธิบายข้างบนยังเหมือนเดิม

```
int A[5]={1,3,5,7,9};
int *ptr;
int x,y,z;
     ptr=&A[2];
     x=*ptr;
     y=*(ptr+1);
     z=*(ptr+2);
     printf("%d %d %d\n",x,y,z);
```

ในโค้ดข้างบน จะแสดง .....

Note: การอ้างถึง element ในอาร์เรย์ โดยใช้ตัวแปร pointer เช่น *(ptr+1) หรือ *(ptr+i) เรียกว่า <u>pointer-offset notation</u> โดยที่การอ้างถึงอาร์เรย์ที่ใช้ชื่ออาร์เรย์แล้วตามด้วย index เช่น A[0] หรือ A[i] เรียกว่า <u>array-index notation</u>

3. Key #3 "Different notations, same meaning"

เราสามารถ<u>ใช้ pointer-offset notation กับตัวแปร array</u> (เช่นถ้า `A` คืออาร์เรย์ แล้วใช้ `*(A+i)`) และในทางกลับกัน<u>ใช้ array-index notation กับตัวแปร pointer</u> (เช่นถ้า `ptr` คือตัวแปร pointer แล้วใช้ `ptr[i]`) ได้โดยทำงานได้เหมือนกันทุกประการ ตัวอย่างเช่นโค้ดข้างล่างที่เขียนโดยใช้ notation ทั้ง 4 แบบจะทำการหาค่าผลรวมของทั้ง 5 element ของอาร์เรย์ A เหมือนกัน

Array-index notation	Pointer-offset notation with Array	Pointer-offset notation	Array-index notation with Pointer

`int A[5]={1,3,5,7,9};`
`int i,sum;`
 	
`sum=0;`
`for(i=0;i<5;i++)`
`sum=sum+_____;`
	`int A[5]={1,3,5,7,9};`
	`int i,sum;`
	
	`sum=0;`
	`for(i=0;i<5;i++)`
	`sum=sum+_____;`
		`int A[5]={1,3,5,7,9};`
		`int i,sum;`
		`int *ptr;`
		
		`ptr=_____;`
		`sum=0;`
		`for(i=0;i<5;i++)`
		`sum=sum+_____;`
			`int A[5]={1,3,5,7,9};`
			`int i,sum;`
			`int *ptr;`
			
			`ptr=_____;`
			`sum=0;`
			`for(i=0;i<5;i++)`
			`sum=sum+_____;`

<u>หมายเหตุ</u> ข้อแตกต่างระหว่างการใช้ชื่ออาร์เรย์ A ในรูปแบบ pointer notation กับการใช้ตัวแปร pointer เช่น ptr คือ ptr เป็นตัวแปรที่เอาไว้เก็บแอดเดรส แต่ A จะหมายถึง<u>ค่าคงที่</u>ที่มีค่าเท่ากับแอดเดรสของ element แรกของอาร์เรย์ A (ซึ่งเรียกว่า pointer constant) ดังนั้นการเขียน `ptr=A;` จึงถูกต้องตามกฎของภาษา C นั่นคือเราสามารถเซ็ตค่าตัวแปร ptr ได้ แต่การเขียน `A=ptr;` จะไม่ถูกต้องตามกฎ เนื่องจากเราไม่สามารถกำหนดค่าให้กับค่าคงที่ได้

4. Key #4 pointer arithmetic: เราสามารถเปลี่ยนค่าของตัวแปรแบบ pointer โดยการบวกหรือลบด้วยค่าแบบ int เช่น ptr=ptr+1 หรือ ptr ดังตัวอย่างโปรแกรมข้างล่าง

```
int A[5]={1,3,5,7,9};
int *ptr;
int i,x,y,z;
            ptr=A;     // ทำการเซ็ตให้ ptr ชี้ไปที่ element แรกของ A
            x=*ptr;  // x มีค่าเท่ากับ element ที่ ptr กำลังชี้ไปอยู่ (A[0])

            ptr=ptr+1;  // หรือ ptr++;  เพื่อทำให้ ptr
                        // ชี้ไปที่ 1 element ถัดจาก element ที่ ptr ชี้อยู่
            y=*ptr;      // คำถาม: ptr ตอนนี้จะชี้ไปที่ A[.....]

            i=2;
            ptr=ptr+i;  // ทำให้ ptr ชี้ไปที่ i element ถัดจาก element ที่ ptr ชี้อยู่ (i มีค่าเท่ากับ 2)
            z=*ptr;     // คำถาม: ptr ตอนนี้จะชี้ไปที่ A[.....]
```

คำถาม: จากโค้ดข้างบน x, y, z มีค่าเท่ากับเท่าไร ……..

<u>หมายเหตุ</u>

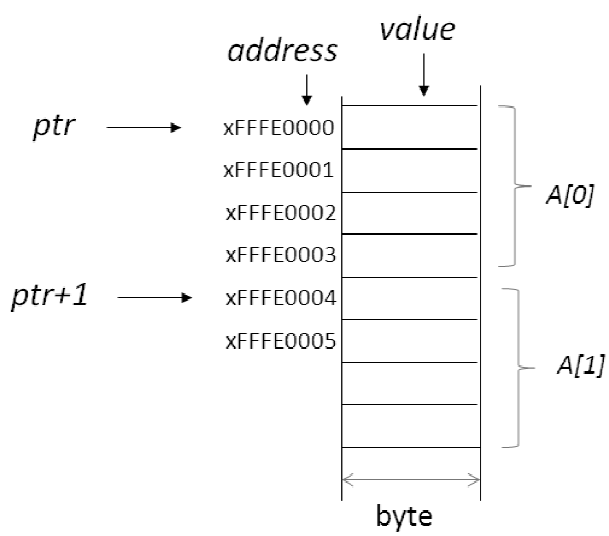
(1) `ptr=ptr+1` ไม่ได้หมายถึงการเพิ่มค่า ptr ด้วย 1 แต่หมายถึงปรับให้ ptr ไปชี้ที่ element ถัดไป โดยบน CPU Intel และ Windows 32 bits พื้นที่ใช้ในหน่วยความจำสำหรับตัวแปร int 1 ตัวคือ 4 bytes ดังในตัวอย่างรูปข้างล่าง ดังนั้นถ้าสมมติว่าปัจจุบัน ptr ชี้ไปที่ A[0] ซึ่งอยู่ที่แอดเดรส xFFFE0000 ดังนั้น ptr=ptr+1; จะทำให้ ptr ไปชี้ที่ A[1] ที่แอดเดรส xFFFE0004 (ไม่ใช่ xFFFE0000+1=xFFFE0001)

(2) sizeof operator ที่ใช้ในรูปแบบ sizeof(object)
ใช้สำหรับระบุจำนวน byte ที่ใช้เก็บ object

```
int x,y;
     x=sizeof(int);
      // คืนค่าจำนวน byte ที่ใช้เก็บ int
     y=sizeof(x);
     // คืนค่าจำนวน byte ที่ใช้เก็บ x (ตัวแปรแบบ int)
```

บน Intel/Win32 x และ y จะมีค่าเท่ากับ 4

(3) ดังนั้น ptr=ptr+1;
เทียบเท่ากับการปรับให้ ptr มีค่าเท่ากับ ptr+sizeof(int)



5. ประโยชน์การใช้ pointer กับ Array

(5.1) การใช้ pointer ในการอ้างถึง element ต่างๆ ใน Array จะทำให้โปรแกรมมีประสิทธิภาพมากขึ้น (เร็วขึ้น) จะเห็นผลอย่างชัดเจนในกรณีที่การประมวลผลที่ซับซ้อนกับอาร์เรย์ขนาดใหญ่ แต่ต้องมีเทคนิคการเขียนโปรแกรมที่ดีด้วย

(5.2) สร้างและใช้งาน Dynamic array (อาร์เรย์ที่ขนาดเปลี่ยนแปลง ณ run-time) ซึ่งจะเรียนในบทถัดไป

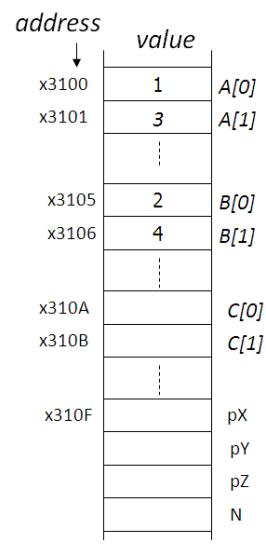
(5.3) ใช้เขียนโปรแกรมเพื่อส่งอาร์เรย์มาเป็นอาร์กิวเมนต์ของฟังก์ชัน (passing arrays as arguments)

Key 1: ใน calling function (ผู้เรียก) ส่งชื่ออาร์เรย์ (แอดเดรสของ element แรก) มาเป็น argument

Key 2: ใน called function (ผู้ถูกเรียก) ใช้ตัวแปรแบบ pointer มารับ argument

<u>โจทย์ตัวอย่าง</u>: จงทำโปรแกรมข้างล่างให้สมบูรณ์เพื่อเขียนและเรียกใช้ฟังก์ชัน Add_Array ซึ่งจะทำการบวกค่าใน element ที่ตรงกัน ของอาร์เรย์ A และ B แล้วเก็บไว้ที่อาร์เรย์ C

```
void Add_Array(int *,int *, int *, int);
int main()
{
int A[5]={1,3,5,7,9},
int B[5]={2,4,6,8,10};
int C[5];

       Add_Array(.....);
}
void Add_Array(int *pX,int *pY, int *pZ, int N)
{
int i;
       for(.....)

          .....
}
```



6. โจทย์ตัวอย่าง (ดัดแปลงจาก Exam II Spring 2006 ECE190 UIUC) จากโปรแกรมข้างล่าง จงตอบคำถามในข้อย่อยต่อไปนี้

```
int fn1(int *B);
void fn2(int x);
void fn3(int *x);

int main()
{
int A[20];
int i;
int x = 5;
       for (i = 0; i < 20; i++)
              A[i] = i;

       x = fn1(A);   /* Line 1 */
       x = A[2];     /* Line 2 */
       x = 34;
       fn2(x);
       i = x;         /* Line 3 */
       fn3(&A[10]);
       i = A[10];    /* Line 4 */
       return 0;
}
```

```
int fn1(int *B)
{
int x;
       x = *(B+2);
       *(B+2) = 15;
       return x;
}

void fn2(int x)
{
int y;
       y = x;
       x = 33;
}

void fn3(int *x)
{
       *x = 25;
}
```

(i) ค่าของ x เมื่อทำงานถึง Line 1 คือ…….. และ ค่าของ x เมื่อทำงานถึง Line 2 คือ……..

(ii) ค่าของ i เมื่อทำงานถึง Line 3 คือ…….. และ ค่าของ i เมื่อทำงานถึง Line 4 คือ……..

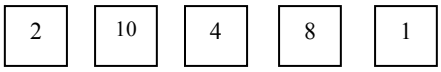
Problem solving with arrays: Insertion sort

1. Problem statement: กำหนดให้อินพุตอาร์เรย์ A ขนาด n (A[0], A[1], …., A[n-1]) ให้ทำการเรียง element ในอาร์เรย์ โดยที่หลังจากการเรียงแล้วเราจะได้อาร์เรย์ที่ A[0] ≤ A[1] ≤ A[2] ≤... A[n-2] ≤ A[n-1]

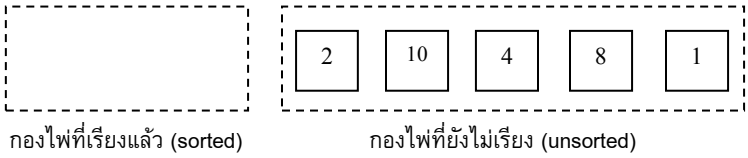
ตัวอย่างเช่น ถ้า n=5 และเริ่มต้น A={A[0], A[1]..., A[4]} = {2, 10, 4, 8, 1} หลังจากผ่านการเรียงจะได้ A={1, 2, 4, 8, 10}

2. อัลกอริทึมการเรียงแบบ Insertion sort

- ลองสมมติว่าข้อมูลในอาร์เรย์คือไพ่ที่อยู่ในมือ

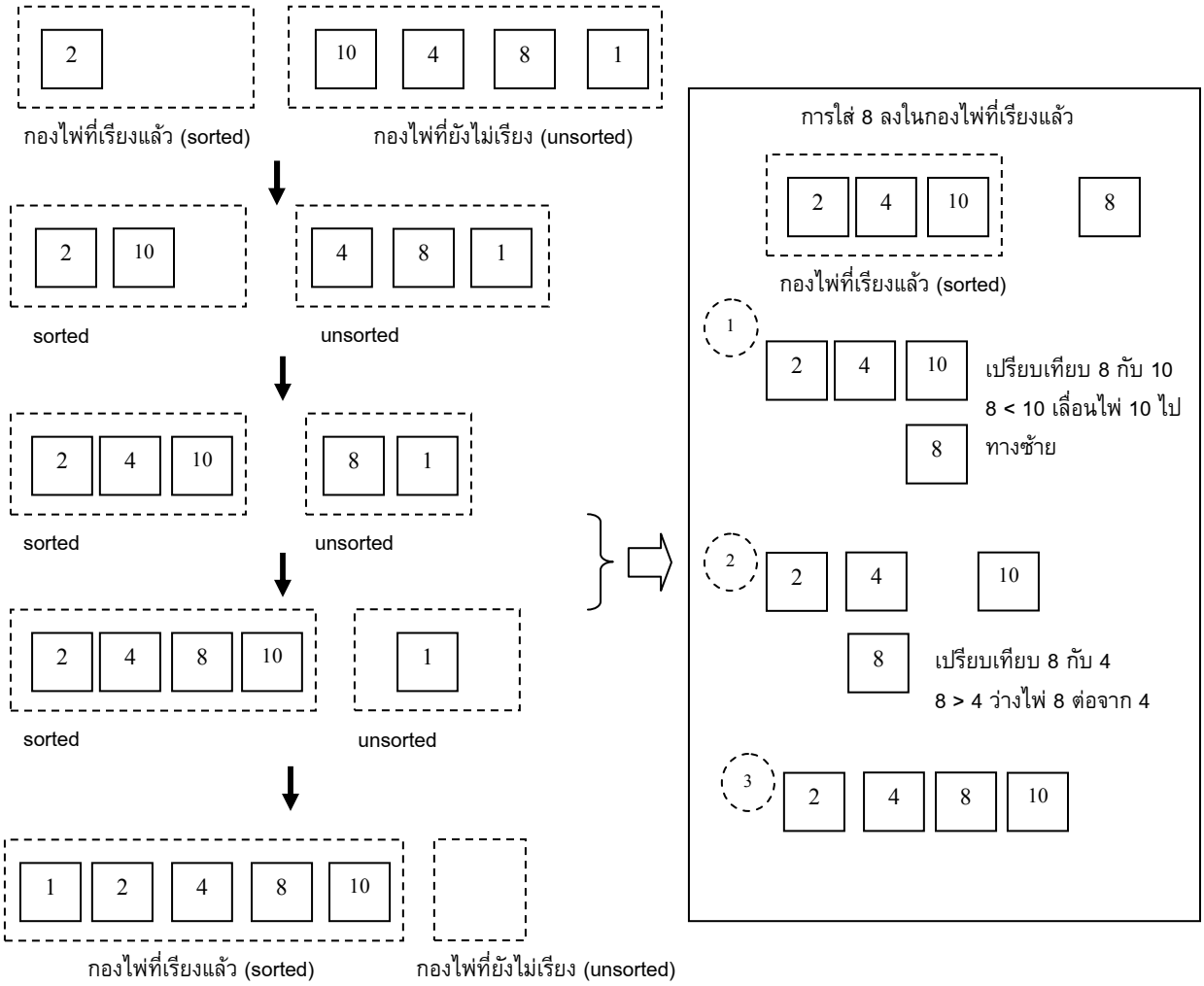


- แบ่งไพ่ออกเป็น 2 กอง คือ กองที่เรียงแล้ว (sorted) กับกองที่ยังไม่เรียง (unsorted) โดยเริ่มต้น กองที่เรียงแล้ว (sorted) คือ เซ็ตว่าง = ϕ = ไม่มีไพ่อยู่ในกองนี้ ส่วนกองที่ยังไม่ได้เรียงเท่ากับไพ่ทั้งหมด



- ทำ action ข้างล่างไปเรื่อยๆ จนกว่า ไม่มีไพ่อยู่ในกองที่ยังไม่ได้เรียง (unsorted)

"นำไพ่ใบแรกของกองที่ยังไม่ได้เรียง (unsorted) มาใส่ในกองที่เรียง (sorted) โดยใส่ในลำดับที่ควรจะเป็น"



3. Coding: จงทำโปรแกรมข้างล่างโดยการเขียนโค้ดในฟังก์ชัน InsertionSort เพื่อทำการเรียงข้อมูล (element) ในอาร์เรย์ที่ระบุโดยอาร์กิวเมนต์ A ขนาด n ให้เรียงจากน้อยไปมาก โดยใช้อัลกอริทึมการเรียงแบบ insertion sort

```
void InsertionSort(int *,int);

int main()
{
int data[5]={2,10,4,8,1};
       InsertionSort(data,5);
       return 0;
}

void InsertionSort(int *A,int n)
{
int
       // ==> ลูปนี้ j จะมีค่าเป็น 1, 2,.. n-1
       for( j=1 ; j<n ; j++ )
       {
            // ==> ในแต่ละรอบ j จะได้ A[0]...A[j-1]เรียงจากน้อยไปมากแล้ว
            // ==>นั้นคือ  A[0]...A[j-1] = กองไพ่ sorted
            // ==>        A[j]...A[n-1]  = กองไพ่ unsorted
            // ==> โดยที่ A[j]คือไพ่ใบแรกของกองไพ่ unsorted
            // ==>ให้นำ A[j] ไปใส่ใน A[0]...A[j-1] ในลำดับที่ควรจะเป็น

            key=A[j]; // เซ็ตค่า key ให้เท่ากับ A[j] (ไพ่ใบแรกของกองไพ่ unsorted)
            // ==> ลูปนี้ i จะมีค่าเป็น j-1, j-2,.. , 1,0
            i=j-1;
            while(i >= ............ && A[..........]>.............)
            {
                // ==>ให้ขยับ A[i] ไปทางขวา จนกว่าจะเจอ A[i]<key
                ..............................................................
               i=.....................
             }
            // ==> หลุดจากลูปข้างบนเมื่อเจอ A[i]ที่ <key ให้นำ key  ไปต่อท้าย A[i]
            ..............................................................
       }
}
```

<u>Multi-Dimensional Array (อาร์เรย์หลายมิติ)</u>

1. ภาษา C อนุญาตให้มีการใช้ตัวแปรแบบอาร์เรย์หลายมิติ
2. กรณี 1-D array (อาร์เรย์ที่ได้เรียนไปก่อนหน้านี้): การประกาศตัวแปร: `DataType name[dim];` เช่น `int A[5];`
3. กรณี 2-D array: การประกาศตัวแปร: `DataType name[#rows][#cols];` เช่น

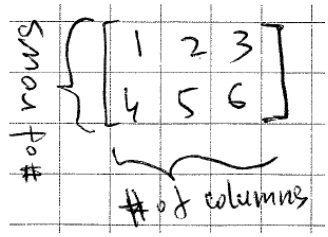
```
int B[2][3];  หรือ int B[2][3]={{1,2,3},{4,5,6}};
```

ซึ่งหมายถึงประกาศตัวแปรแบบอาร์เรย์ของ int ขนาด 2 แถว 3 คอลัมน์ (กรณีหลังทำการกำหนดค่าเริ่มต้นด้วย)

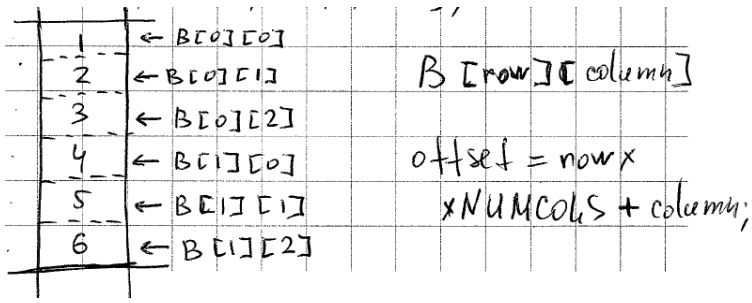
4. กรณี 2-D array: การอ้างถึงแต่ละ element ของอาร์เรย์: `name[row_index][column_index]`

โดยที่ row_index มีค่า 0,1,.. #row-1 และ column_index มีค่า 0,1,.. #col-1

```
{
int B[2][3]={{1,2,3},{4,5,6}};
int a,b,c,d,e,f;

        a=B[0][0]; b=B[0][1]; c=B[0][2];
        d=B[1][0]; e=B[1][1]; f=B[1][2];
        printf("%d %d %d\n",a,b,c);
        printf("%d %d %d\n",d,e,f);
        ...
}
```



5. นอกจากนี้ในภาษา C ข้อมูลในอาร์เรย์จะถูกจะถูกจัดเรียงต่อกันที่ละแถว (row-major order) *



6. โจทย์ตัวอย่าง จงทำโปรแกรมข้างล่างให้สมบูรณ์ เพื่อทำการบวกเมทริกซ์ที่ระบุโดยอาร์เรย์ A กับเมทริกซ์ที่ระบุโดยอาร์เรย์ B โดยนำมาผลลัพธ์เก็บเมทริกซ์ที่ระบุโดยอาร์เรย์ C

```
int main()
{
  int A[3][3]={{0,2,4},
               {6,8,10},
               {12,14,16}};
  int B[3][3]={{1,3,5},
               {7,9,11},
               {13,15,17}};
  int C[3][3];
  int i,j;





}
```

7. กรณีอาร์เรย์มากกว่า 2 มิติ แนวคิดจะคล้ายกัน

การประกาศตัวแปร: `DataType name[dim1][dim2]...[dimN];` เช่น `int C[10][10][10];`

การอ้างถึงแต่ละ element ของอาร์เรย์: `name[dim1_index][dim2_index]... [dimN_index]` เช่น

```
C[0][0][0]=C[1][1][1]+C[9][9][9];
```

* รูปภาพนำมาจาก Slide วิชา ECE190 UIUC by V.Kindratenko (https://wiki.engr.illinois.edu/display/ece190/Lectures)