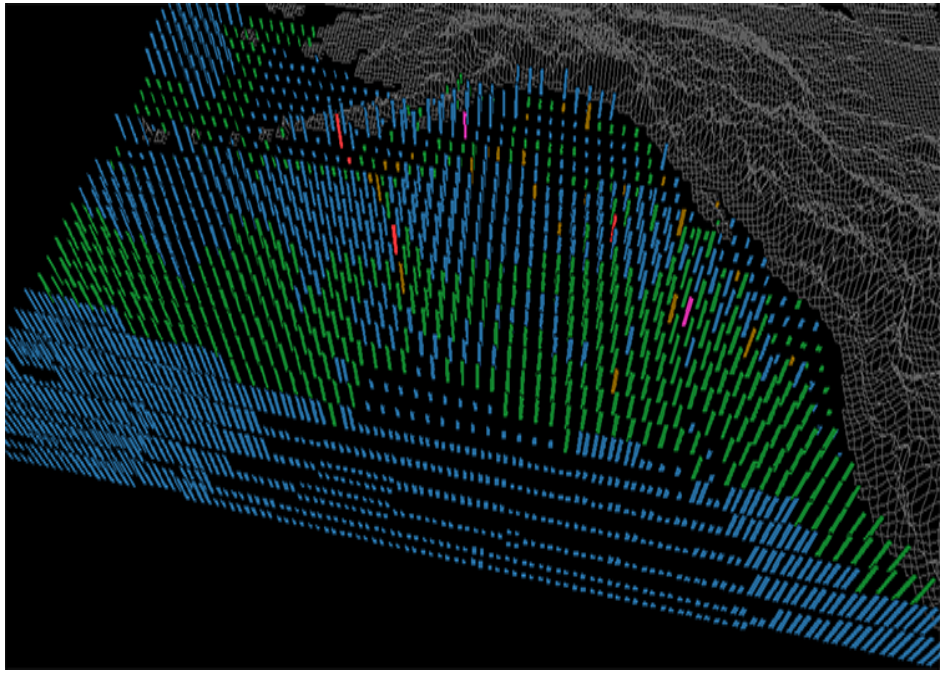
Computer Programming I:
การเขียนโปรแกรมคอมพิวเตอร์ I



ตัวชี้ (Pointer)

อ.ดร.ปัญญนัท อ้นพงษ์

ภาควิชาคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศิลปากร

aonpong_p@su.ac.th

Outline

- **ตัวชี้ ในภาษาซี**
- ลักษณะการใช้งานของตัวชี้
- ตัวดำเนินการ & บนตัวแปรทั่วไป
- การประกาศตัวชี้ และการเรียกใช้งาน
- การประยุกต์ใช้ตัวชี้
  - เปลี่ยนค่าพารามิเตอร์ที่ส่งไป โดยให้ผลกับผู้เรียกด้วย
  - อาเรย์กับตัวชี้
  - การส่งคำตอบจากฟังก์ชันมากกว่าหนึ่งตัว
  - ตัวชี้กับอาเรย์พลวัต (Dynamic array)

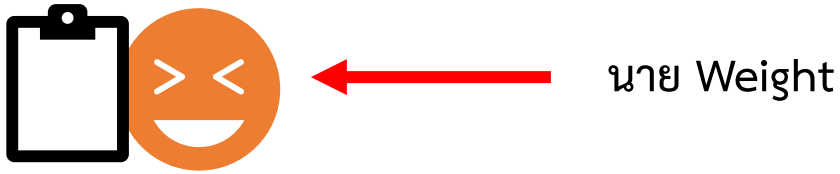
ตัวชี้ ในภาษาซี

- ตัวชี้ หรือ pointer เป็นตัวแปรที่จัดเป็นชนิดข้อมูลขั้นสูงในภาษาซี
- ถ้าเทียบกับ
  - int ที่เก็บตัวเลขจำนวนเต็ม
  - float ที่เก็บตัวเลขทศนิยม

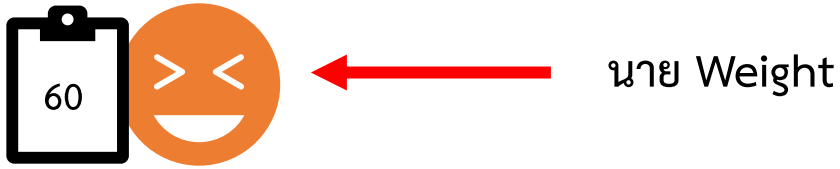
  ตัวแปร pointer จะเก็บ**ที่อยู่ของข้อมูล**

ตัวชี้ ในภาษาซี

- เมื่อเราประกาศตัวแปรขึ้นมา 1 ตัว สมมติว่าประกาศ int weight; เราจะได้พนักงานมาคนนึง (พนักงานคนนี้ทำอะไรไม่ได้ ได้แค่ถือป้ายที่มีตัวเลขจำนวนเต็มบรรจุอยู่เท่านั้น)

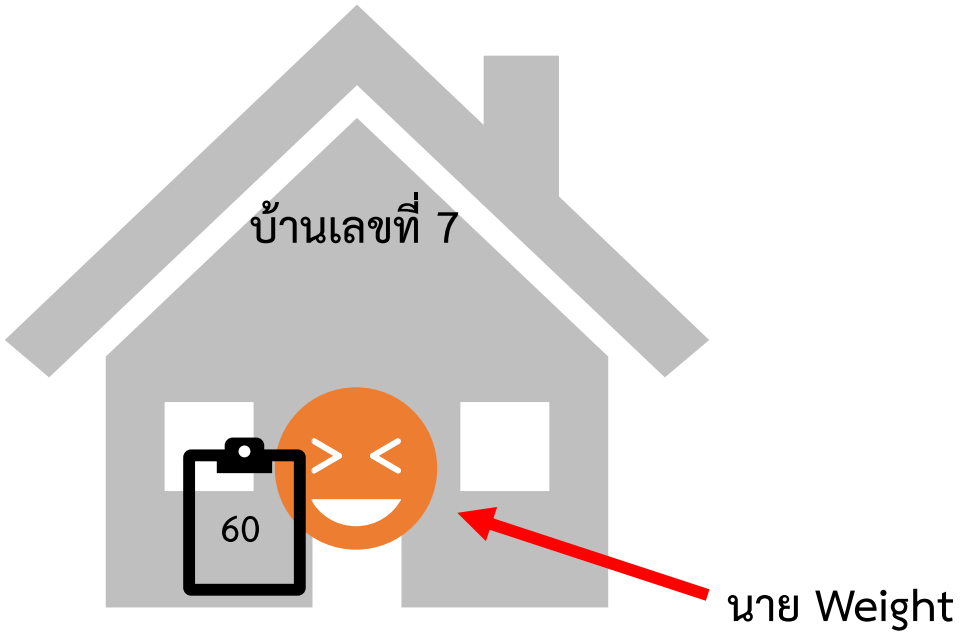


- ถ้าเรากำหนดว่า weight = 60; นายคนนี้ก็จะถือป้าย 60



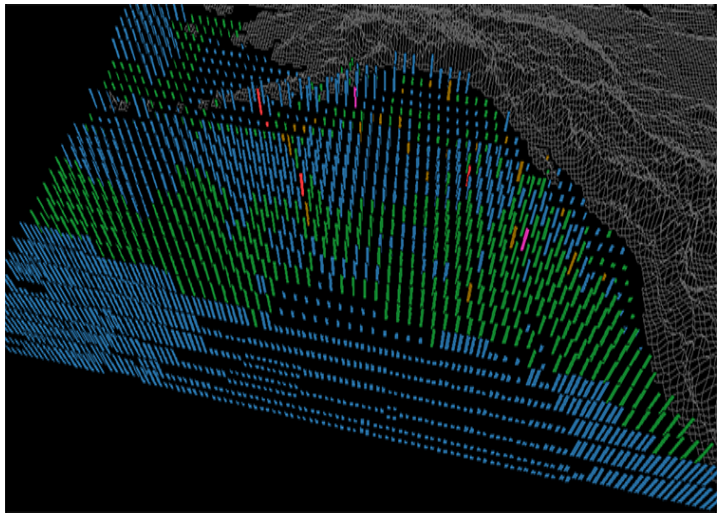
ตัวชี้ ในภาษาซี

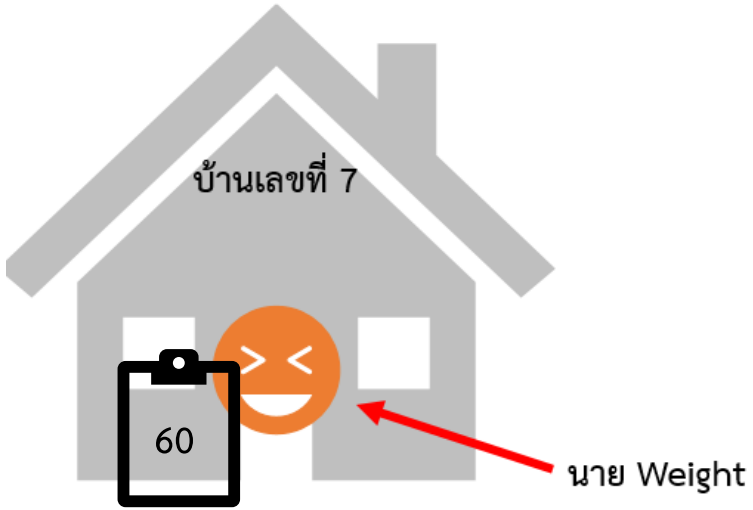
- สมมติว่ามีบ้านหลังหนึ่ง โดยนาย Weight อาศัยอยู่ในบ้านหลังนี้
- นาย weight ถือป้าย 60
- และที่อยู่ของบ้านหลังนี้คือ บ้านเลขที่ 7

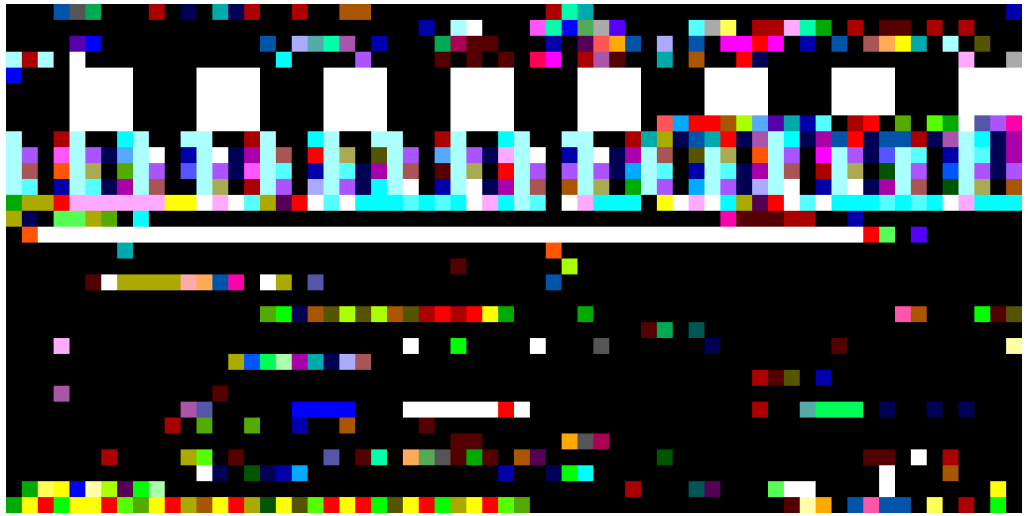


ตัวชี้ ในภาษาซี

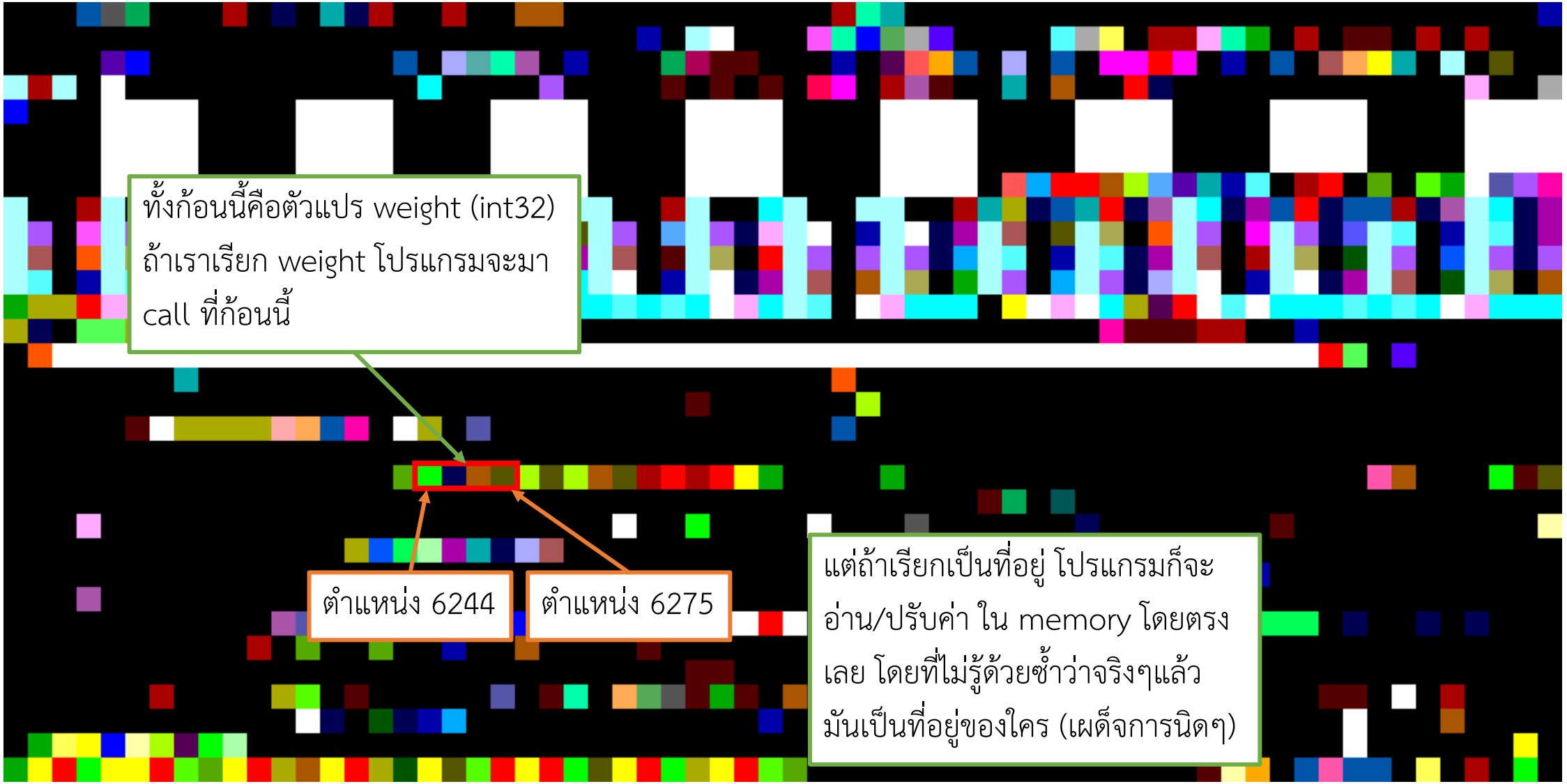
- ถ้านำสิ่งนี้ไปเทียบกับระบบตัวแปร
  - นาย weight คือ ชื่อตัวแปร ที่เก็บค่าน้ำหนัก 60
    - int weight = 60;
  - และบ้านเลขที่ ก็คือที่อยู่ของตัวแปรนี้ในหน่วยความจำ







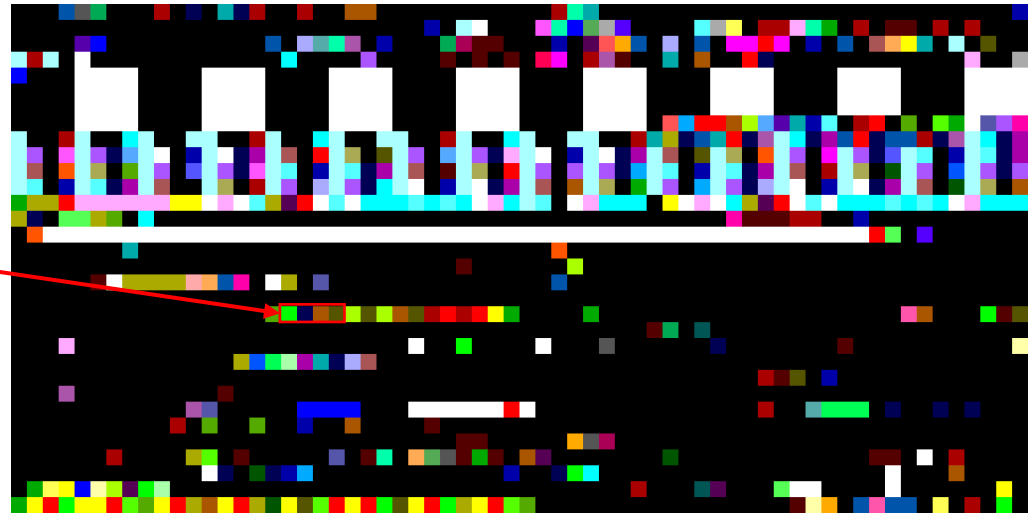
ตัวชี้ ในภาษาซี


ทั้งก้อนนี้คือตัวแปร weight (int32)
ถ้าเราเรียก weight โปรแกรมจะมา call ที่ก้อนนี้
ตำแหน่ง 6244
ตำแหน่ง 6275
แต่ถ้าเรียกเป็นที่อยู่ โปรแกรมก็จะอ่าน/ปรับค่า ใน memory โดยตรงเลย โดยที่ไม่รู้ด้วยซ้ำว่าจริงๆแล้วมันเป็นที่อยู่ของใคร (เผด็จการนิดๆ)

ตัวชี้ ในภาษาซี

- การรู้ที่อยู่ จะทำให้เราอ่านและปรับค่าตัวแปรในตำแหน่งนั้นได้
  - การใช้ที่อยู่ของตัวแปรทำให้เราจัดการกับตัวแปรได้หลากหลายขึ้นมาก
  - ถ้าใช้อย่างถูกต้องและเหมาะสมมันจะทำประโยชน์ในโปรแกรมได้มาก

เจ้าคอมไพล์เลอร์ เปลี่ยนค่าที่
ตำแหน่ง 6244-6275 ซิ



Outline

- ตัวชี้ ในภาษาซี
- **ลักษณะการใช้งานของตัวชี้**
- ตัวดำเนินการ & บนตัวแปรทั่วไป
- การประกาศตัวชี้ และการเรียกใช้งาน
- การประยุกต์ใช้ตัวชี้
  - เปลี่ยนค่าพารามิเตอร์ที่ส่งไป โดยให้ผลกับผู้เรียกด้วย
  - อาเรย์กับตัวชี้
  - การส่งคำตอบจากฟังก์ชันมากกว่าหนึ่งตัว
  - ตัวชี้กับอาเรย์พลวัต (Dynamic array)

ลักษณะการใช้งานของตัวชี้

- จากเรื่องฟังก์ชัน จะเห็นได้ว่าการรับส่งพารามิเตอร์ไปยังฟังก์ชันมีข้อจำกัดอยู่มาก
  - จะส่งค่าไป ก็ปรับแก้ค่าที่ฟังก์ชันตัวที่เรียกใช้ไม่ได้ เพราะพารามิเตอร์ที่ส่งไปเป็นแค่ตัวสำเนา
    - ให้นึกถึง scanf ที่เราต้องส่งที่อยู่ตัวแปรเข้าไปเปลี่ยนค่าในนั้น (คือสาเหตุที่ต้องใช้ &)
  - มีปัญหาเวลาจะคืนค่า คือคืนได้แค่ทีละค่า (return)
    - ถ้าเราเข้าถึงที่อยู่ของตัวแปรได้ จะแก้กี่ตัวก็ได้ เพราะเราเข้าไปจัดการ memory โดยตรง
- ใช้จัดการอาเรย์ก็ได้
  - จริงๆแล้ว อาเรย์ก็คือ pointer ชนิดหนึ่ง โดยจะชี้ไปที่ตำแหน่งแรกของอาเรย์
    - สามารถจัดการข้อมูลที่สร้างขึ้นแบบ Dynamic ได้ (Dynamic Allocation)

Outline

- ตัวชี้ ในภาษาซี
- ลักษณะการใช้งานของตัวชี้
- **ตัวดำเนินการ & บนตัวแปรทั่วไป**
- การประกาศตัวชี้ และการเรียกใช้งาน
- การประยุกต์ใช้ตัวชี้
  - เปลี่ยนค่าพารามิเตอร์ที่ส่งไป โดยให้ผลกับผู้เรียกด้วย
  - อาเรย์กับตัวชี้
  - การส่งคำตอบจากฟังก์ชันมากกว่าหนึ่งตัว

ตัวดำเนินการ & บนตัวแปรทั่วไป

- ณ จุดนี้มีแนวคิดสองอย่างที่แตกต่างกันแต่สัมพันธ์กัน คือ
  - ที่อยู่ของตัวแปร (address) ซึ่งเปรียบเหมือนเลขที่บ้านของตัวแปร
  - ตัวแปรที่ใช้บันทึกเลขที่บ้าน (ตัวชี้/pointer)
- ที่อยู่ของตัวแปรเป็นสิ่งที่มีมาตลอด และเราสามารถเรียกดูได้ผ่านการใช้เครื่องหมาย & โดยนำไปวางไว้ข้างหน้าตัวแปรอะไรก็ได้
  - เช่น &x หมายถึง ที่อยู่ของตัวแปร x
  - & เป็นสิ่งที่ใช้ได้กับตัวแปรทุกชนิด แต่ห้ามใช้กับค่าคงที่ทั่วไป เพราะค่าคงที่ทั่วไปไม่ใช่ตัวแปร
  - ถ้าเรารู้ว่าจะเล่นกับที่อยู่ของตัวแปรบ่อยๆ แทนที่จะพิมพ์ & ทุกครั้ง เราสามารถใช้ pointer มาเก็บค่าที่อยู่ไว้ได้

Outline

- ตัวชี้ ในภาษาซี
- ลักษณะการใช้งานของตัวชี้
- ตัวดำเนินการ & บนตัวแปรทั่วไป
- **การประกาศตัวชี้ และการเรียกใช้งาน**
- การประยุกต์ใช้ตัวชี้
  - เปลี่ยนค่าพารามิเตอร์ที่ส่งไป โดยให้ผลกับผู้เรียกด้วย
  - อาเรย์กับตัวชี้
  - การส่งคำตอบจากฟังก์ชันมากกว่าหนึ่งตัว
  - ตัวชี้กับอาเรย์พลวัต (Dynamic array)

การประกาศตัวชี้ และการเรียกใช้งาน

- ให้แยกจากกันออกเป็น 2 ส่วนอย่างเด็ดขาด เพราะการประกาศและการเรียกใช้งานมีสิ่งที่ชวนสับสนอยู่เล็กน้อย
- ตั้งสติให้ดี แล้วเริ่มที่การประกาศตัวชี้ก่อน

การประกาศตัวชี้ และการเรียกใช้งาน

- การประกาศตัวชี้จะใช้ * ตามหลังชนิดข้อมูลของตัวแปรที่มันจะชี้ เช่น
  - int* เป็นตัวชี้ไปยังข้อมูลชนิดจำนวนเต็ม
  - char* เป็นตัวชี้ไปยังข้อมูลชนิดอักขระ
  - double* เป็นตัวชี้ไปยังข้อมูลชนิดทศนิยมความเที่ยงทวีคูณ
- คำว่า int*, char*, และ double* ทำนองนี้เป็นชนิดข้อมูลในตัวของมันเอง
  - อย่างที่บอกไปว่าตัวชี้แท้จริงเป็นชนิดข้อมูล (data type) ขั้นสูงชนิดหนึ่ง
- การประกาศตัวชี้ก็คือการประกาศตัวแปร ดังนั้นจึงมีชนิดข้อมูล ตามด้วยชื่อ
  - เช่น int* px; char* pc; double* pd;
  - แม้ไม่ใช่เรื่องบังคับ แต่คนนิยมตั้งชื่อตัวชี้ให้มีตัวอักษร p หรือคำว่า ptr นำหน้า

การประกาศตัวชี้ และการเรียกใช้งาน

- ถ้าเราต้องการประกาศ pointer หลายตัวพร้อมกัน เราสามารถใช้เครื่องหมาย , (คอมมา) มาขั้นเพื่อประกาศตัวแปรอื่นที่ชี้ไปยังข้อมูลชนิดเดียวกันได้ แต่
  - เครื่องหมาย * จะต้องย้ายไปอยู่หน้าตัวแปรทุกตัวแทน เช่น

    int *pa, *pb, *pc, *pd;
  - การเว้นว่างระหว่างเครื่องหมาย * และชื่อตัวแปรในขั้นตอนการประกาศ ไม่ทำให้เกิดปัญหา แต่อย่าหาทำ มันทำให้เราสับสนเอง

    int * pa, * pb, * pc, * pd;
  - สังเกตว่าชนิดของตัวแปรที่เราจะชี้ไปก็ยังคงต้องมีอยู่

การประกาศตัวชี้ และการเรียกใช้งาน

- เราประกาศ pointer ได้แล้ว ตอนนี้เราจะย้ายไปยังการเรียกใช้งาน pointer

การประกาศตัวชี้ และการเรียกใช้งาน

การเรียกใช้งาน pointer มีอยู่สองลักษณะ คือ

1. ใช้บันทึกค่าที่อยู่ของตัวแปรที่เราสนใจ
2. ใช้อ่านค่าหรือเปลี่ยนค่าตัวแปรที่เราสนใจ

การประกาศตัวชี้ และการเรียกใช้งาน

การใช้งานในแต่ละลักษณะจะมีวิธีเขียนที่ไม่เหมือนกัน ดังนี้

- ตอนบันทึกค่าที่อยู่ของตัวแปร
  - เราจะใช้ชื่อของตัวชี้ตรง ๆ เช่น

```
int* ptr;
int x = 5;
ptr = &x;
```

  - ตอนอ่านหรือเขียนค่าตัวแปรที่สนใจเราใส่ * ไว้หน้าชื่อตัวชี้

```
printf("%d", *ptr);
*ptr = 7;
printf("%d", x);
```

การประกาศตัวชี้ และการเรียกใช้งาน

Trick: หน้านี้ไม่เคยเห็นอ้างอิงในตำราเล่มไหน แต่อาจจะมีประโยชน์เวลาสับสน

- ถ้าเจอเครื่องหมาย & หน้าตัวแปร (อะไรก็ตาม) ให้พูดในใจว่า “ที่อยู่ของ..” เช่น
  - &x จะอ่านว่า ที่อยู่ของ x
  - &weight จะอ่านว่า ที่อยู่ของ weight
- ถ้าเจอเครื่องหมาย * หน้าตัวแปร (อยู่หน้า pointer) ให้พูดในใจว่า “ค่าที่อยู่ในตำแหน่ง..” เช่น
  - *px จะอ่านว่า ค่าที่อยู่ใน “ตำแหน่ง px”
  - *ptotal จะอ่านว่า ค่าที่อยู่ใน “ตำแหน่ง ptotal”

  //ตรงนี้แหละที่จะสับสนกับขั้นตอนการประกาศ เพราะขั้นตอนการประกาศไม่ต้องอ่านอะไร แค่ประกาศเฉย ๆ ประกาศเสร็จแล้วก็ไม่ต้องไปงงกับเครื่องหมาย * อีก เพราะเป็นคนละส่วนกัน

การประกาศตัวชี้ และการเรียกใช้งาน

- ลองใช้ Trick หน้าที่แล้วดู เข้าใจง่ายขึ้นมั้ย

```
int* ptr;
int x = 5;
ptr = &x;
```

```
printf("%d", *ptr);
*ptr = 7;
printf("%d", x);
```

- แต่ก็ยังคงต้องฝึกดูฝึกท่อง ไม่งั้นก็สับสนได้อยู่ดี
  - ไม่คาดหวังว่าจะได้ใช้ในการสอบแล็บ แต่ Lecture และชีวิตจริงคือหนีไม่พ้นแน่ ๆ
- ให้เลือกวิธีที่เหมาะกับตัวเอง

การประกาศตัวชี้ และการเรียกใช้งาน

ตัวอย่างโปรแกรม

```c
#include<stdio.h>
void main() {
    int* ptr;
    int x = 5;
    ptr = &x;
    printf("%d\n", *ptr);

    *ptr = 7;

    printf("%d\n", x);
}
```

```
5
7
```

การประกาศตัวชี้ และการเรียกใช้งาน

ตัวอย่างโปรแกรม

```c
#include<stdio.h>
void main() {
    int* ptr;
    int x = 5;
    ptr = &x;
    printf("%d\n", *ptr);

    *ptr = 7;

    printf("%d\n", x);
}
```

เพราะ ptr เก็บที่อยู่ของ x ไว้ เมื่อเราใช้ *ptr มันจะดึงค่าของ x มาแสดง ตรงนี้จึงได้ผลเป็นเลข 5

ถ้าเราเขียน *ptr ไว้ที่ทางด้านซ้ายของเครื่องหมายเท่ากับ ตัวแปรที่มันชี้อยู่จะถูกเปลี่ยนค่าตามไปด้วย

ผลจาก *ptr = 7; ทำให้ค่า x เปลี่ยนเป็น 7 ตรงนี้ โปรแกรมจะพิมพ์เลข 7 ออกมา

Outline

- ตัวชี้ ในภาษาซี
- ลักษณะการใช้งานของตัวชี้
- ตัวดำเนินการ & บนตัวแปรทั่วไป
- การประกาศตัวชี้ และการเรียกใช้งาน
- **การประยุกต์ใช้ตัวชี้**
  - **เปลี่ยนค่าพารามิเตอร์ที่ส่งไป โดยให้ผลกับผู้เรียกด้วย**
  - อาเรย์กับตัวชี้
  - การส่งคำตอบจากฟังก์ชันมากกว่าหนึ่งตัว
  - ตัวชี้กับอาเรย์พลวัต (Dynamic array)

การประยุกต์ใช้ตัวชี้ : ปรับค่าต้นฉบับ

- จากหน้าที่แล้ว เพราะค่าตัวแปร x ถูกเปลี่ยนจากการใช้ *ptr = 7;
- แสดงว่าการใช้ตัวชี้หรือที่อยู่ของข้อมูลสามารถโยงไปถึงตัวแปรที่แท้จริงได้
  - เพื่อสร้างความคุ้นเคย ลองมาใช้กับคำสั่ง scanf ที่เราใช้กันมาตลอดก่อน

การประยุกต์ใช้ตัวชี้ : ปรับค่าต้นฉบับ

```c
#include<stdio.h>
void main() {
    int x = 5;
    int* ptr = &x;
    scanf("%d", &x);
    printf("%d\n", x);
    scanf("%d", ptr);
    printf("%d\n", x);
}
```

เราส่งที่อยู่ของ x ผ่าน ptr แบบนี้ก็ได้

เพราะ ptr = &x

(สังเกตว่าเป็น scanf ที่ไม่มี &)

การประยุกต์ใช้ตัวชี้ : ปรับค่าต้นฉบับ

คราวนี้กลับไปเรื่องฟังก์ชันแบบดั้งเดิม

- จะพบว่า แม้ว่าเราจะกำหนดและเปลี่ยนค่า x ในฟังก์ชันยังไง เราก็จะเปลี่ยนได้แค่สำเนาของมันเท่านั้น ค่า x ใน main ไม่ได้เปลี่ยนด้วย

```c
#include<stdio.h>

int add_mult(int x, int y) {
    x = x + y;
    x = x * y;
    return x;
}
void main() {
    int x, y, result;
    x = 5; y = 2;
    result = add_mult(x, y);
    printf("%d\n", x);
    printf("%d\n", result);
}
```

```
5
14
```

การประยุกต์ใช้ตัวชี้ : ปรับค่าต้นฉบับ

พิจารณาความแตกต่างระหว่าง scanf และ add_mult

- scanf ต่างกับฟังก์ชันที่แสดงมาเมื่อสักครู่ตรงชนิดของพารามิเตอร์
  - พารามิเตอร์ที่ส่งไปให้ scanf ถูกเปลี่ยนค่าจริง ๆ แต่พารามิเตอร์ที่ส่งไปให้ add_mult ไม่ถูกเปลี่ยนค่า

- แสดงว่าในกรณีที่เราต้องการเปลี่ยนค่าพารามิเตอร์ที่เป็นตัวแปรของฟังก์ชันที่เรียกใช้ฟังก์ชัน เราสามารถใช้ที่อยู่ของข้อมูลหรือตัวชี้เป็นพารามิเตอร์ แทนที่จะเป็นตัวแปรตรงๆ
  - ลองคิดว่าถ้าเราให้สำเนาเอกสารสำคัญกับใครไป ถ้าเขาจะแก้ ก็จะแก้ได้แค่สำเนา แก้ตัวจริงไม่ได้
  - แต่ถ้าเราให้สำเนาทะเบียนบ้านที่เก็บสำเนานั้นเอาไว้ แม้ว่าจะเป็นสำเนา แต่ก็ทำให้เขารู้ที่อยู่ต้นฉบับ ทีนี้ถ้าเขาจะแก้ เขาก็มาแก้ที่บ้านเราได้เลย เอกสารตัวจริงก็จะถูกเปลี่ยน

การประยุกต์ใช้ตัวชี้ : ปรับค่าต้นฉบับ

พารามิเตอร์ที่รับที่อยู่ของตัวแปรหรือตัวชี้ได้ต้องมีชนิดข้อมูลเป็นแบบตัวชี้

- สมมติว่าเราต้องการแก้ให้ x ใน add_mult เป็นแบบตัวชี้เราก็ต้องใช้เครื่องหมาย * ตามหลังชนิดข้อมูลของ x เช่น

```c
#include<stdio.h>

int add_mult(int x, int y) {
    x = x + y;
    x = x * y;
    return x;
}
void main() {
    int x, y, result;
    x = 5; y = 2;
    result = add_mult(x, y);
    printf("%d\n", x);
    printf("%d\n", result);
}
```

แบบเดิม

```
5
14
```

แบบ ptr

```c
#include<stdio.h>

int add_mult_ptr(int* x, int y) {
    *x = *x + y;
    *x = *x * y;
    return *x;
}
void main() {
    int x, y, result;
    x = 5; y = 2;
    result = add_mult_ptr(&x, y);
    printf("%d\n", x);
    printf("%d\n", result);
}
```

```
14
14
```

การประยุกต์ใช้ตัวชี้ : ปรับค่าต้นฉบับ

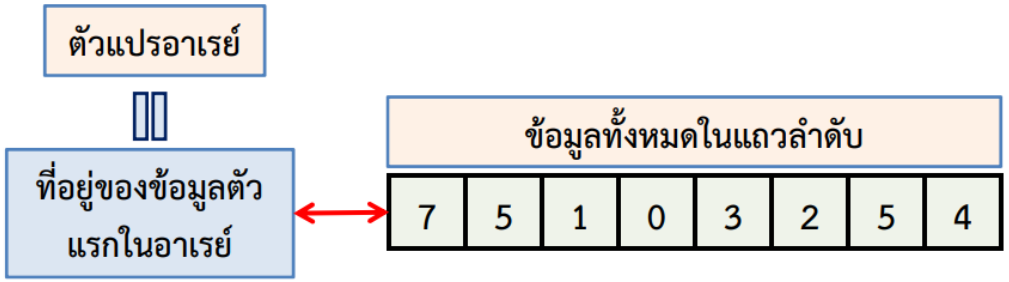
- ตัวชี้ใช้เก็บที่อยู่ของข้อมูล ส่วนที่อยู่ของข้อมูลเป็นสิ่งที่เกิดขึ้นมาคู่กับตัวแปร
- พารามิเตอร์ที่เราประกาศไว้ตรงหัวฟังก์ชัน เช่น int add_mult(int x, int y) และ int add_mult_ptr(int* x, int y) อยู่ในรูปตัวแปรตลอด เพราะเราไม่รู้ว่าคนเรียกจะใส่ค่าอะไรเข้ามา
- ตัวแปรพารามิเตอร์ทั่วไป (ตัวแปร y ในที่นี้) อาจจะรับค่าคงที่มาโดยตรง เช่น เลข 3 หรือตัวแปรจำนวนเต็มจาก main มาก็ได้
- ตัวแปรพารามิเตอร์แบบตัวชี้ (ตัวแปร int* x) อาจจะรับค่าคงที่ คือที่อยู่ของตัวแปรมาโดยตรง เช่น &x หรือจะรับตัวแปรชนิดตัวชี้จาก main มาก็ได้
- นั่นคือ ถ้าหากว่าเรามี int* ptr = &x; ใน main เราเรียกฟังก์ชันโดยส่ง ptr ไปก็ได้ เช่น add_mult_ptr (ptr, y);

Outline

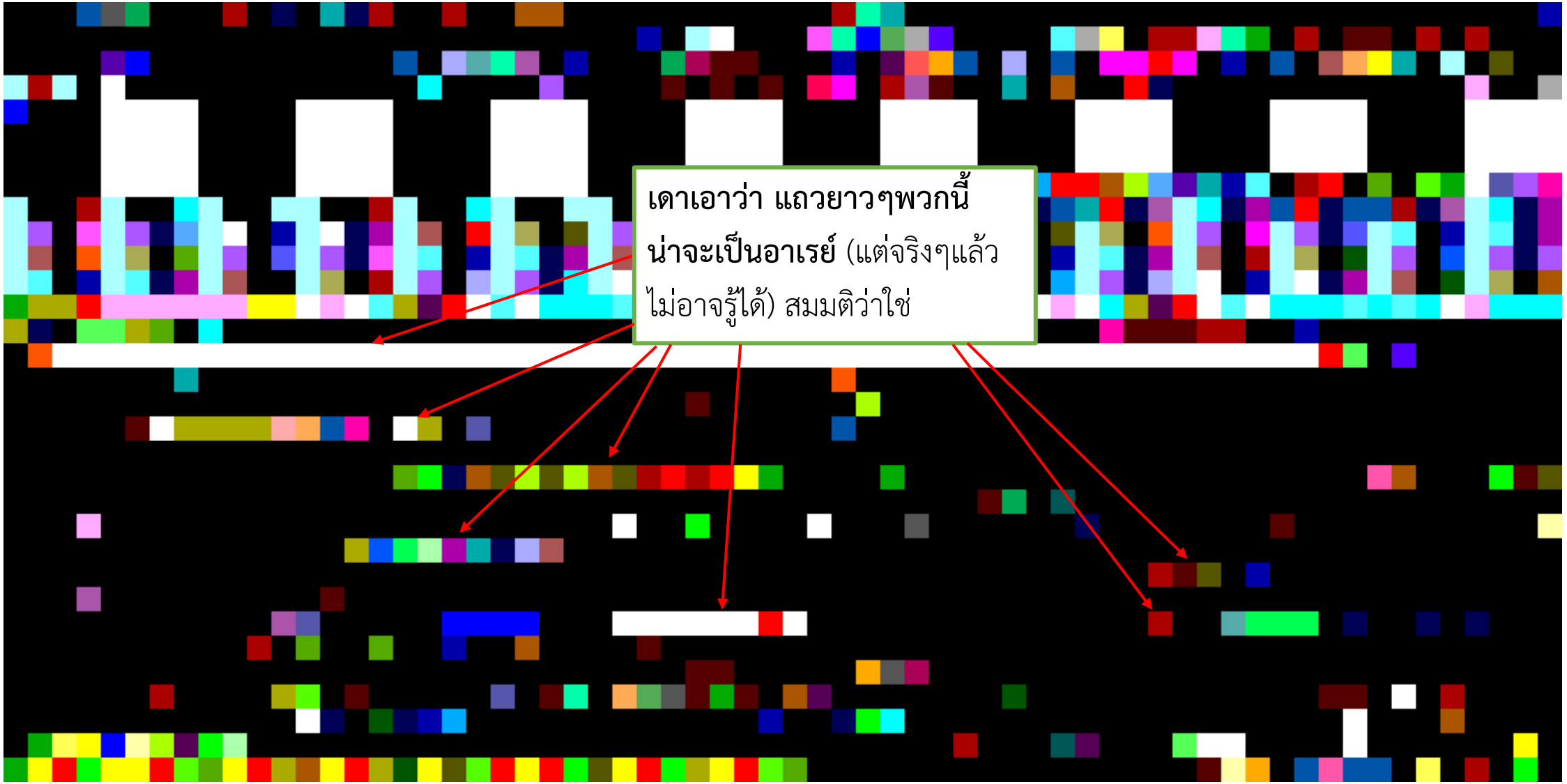
- ตัวชี้ ในภาษาซี
- ลักษณะการใช้งานของตัวชี้
- ตัวดำเนินการ & บนตัวแปรทั่วไป
- การประกาศตัวชี้ และการเรียกใช้งาน
- **การประยุกต์ใช้ตัวชี้**
  - เปลี่ยนค่าพารามิเตอร์ที่ส่งไป โดยให้ผลกับผู้เรียกด้วย
  - **อาเรย์กับตัวชี้**
  - การส่งคำตอบจากฟังก์ชันมากกว่าหนึ่งตัว
  - ตัวชี้กับอาเรย์พลวัต (Dynamic array)

การประยุกต์ใช้ตัวชี้ : อาเรย์กับตัวชี้

- แถวลำดับแท้จริงเป็นตัวชี้ในรูปแบบหนึ่ง การรู้จักวิธีส่งผ่านแถวลำดับไปเป็นพารามิเตอร์ของฟังก์ชันจะช่วยเราได้มาก
- เรากำลังจะพูดถึงสามสิ่งต่อไปนี้ :
  - ตัวแปรแถวลำดับ, ที่อยู่ของข้อมูล, และ ข้อมูลในแถวลำดับ
- ค่าตัวแปรแถวลำดับที่จริงเก็บที่อยู่ของข้อมูลตัวแรกในแถวลำดับไว้
- ตัวแปรแถวลำดับทำให้รูปแบบการอ้างถึงข้อมูลดูเข้าใจง่ายขึ้น



ตัวชี้ ในภาษาซี


เดาเอาว่า แถวยาวๆพวกนี้
น่าจะเป็นอาเรย์ (แต่จริงๆแล้ว
ไม่อาจรู้ได้) สมมติว่าใช่

ตัวชี้ ในภาษาซี

การประยุกต์ใช้ตัวชี้ : อาเรย์กับตัวชี้

- จำไว้ว่าการอ่านเขียนข้อมูลที่ตัวชี้อ้างถึงอยู่ต้องใช้เครื่องหมาย * นำหน้า
  - บ้านเลขที่มักเป็นเลขติดต่อกันไป ที่อยู่ของข้อมูลในแถวลำดับก็เป็นแบบนั้น

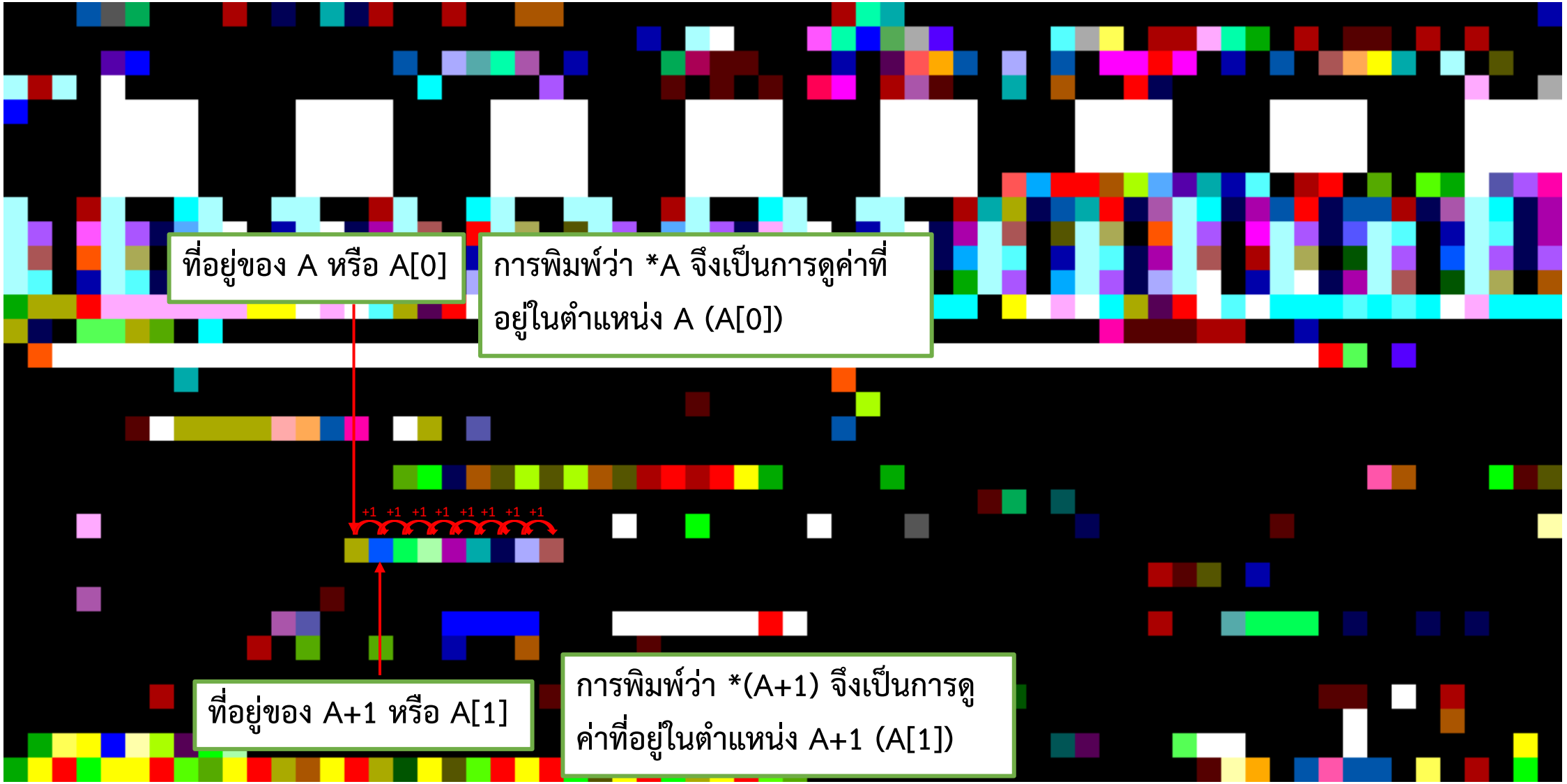
```c
#include<stdio.h>

void main() {
    int A[3] = {6, 7, 8};
    printf("array = %d %d %d\n",
    A[0], A[1], A[2]);
    printf("array = %d %d %d\n", *(A+0), *(A+1), *(A+2));
}
```

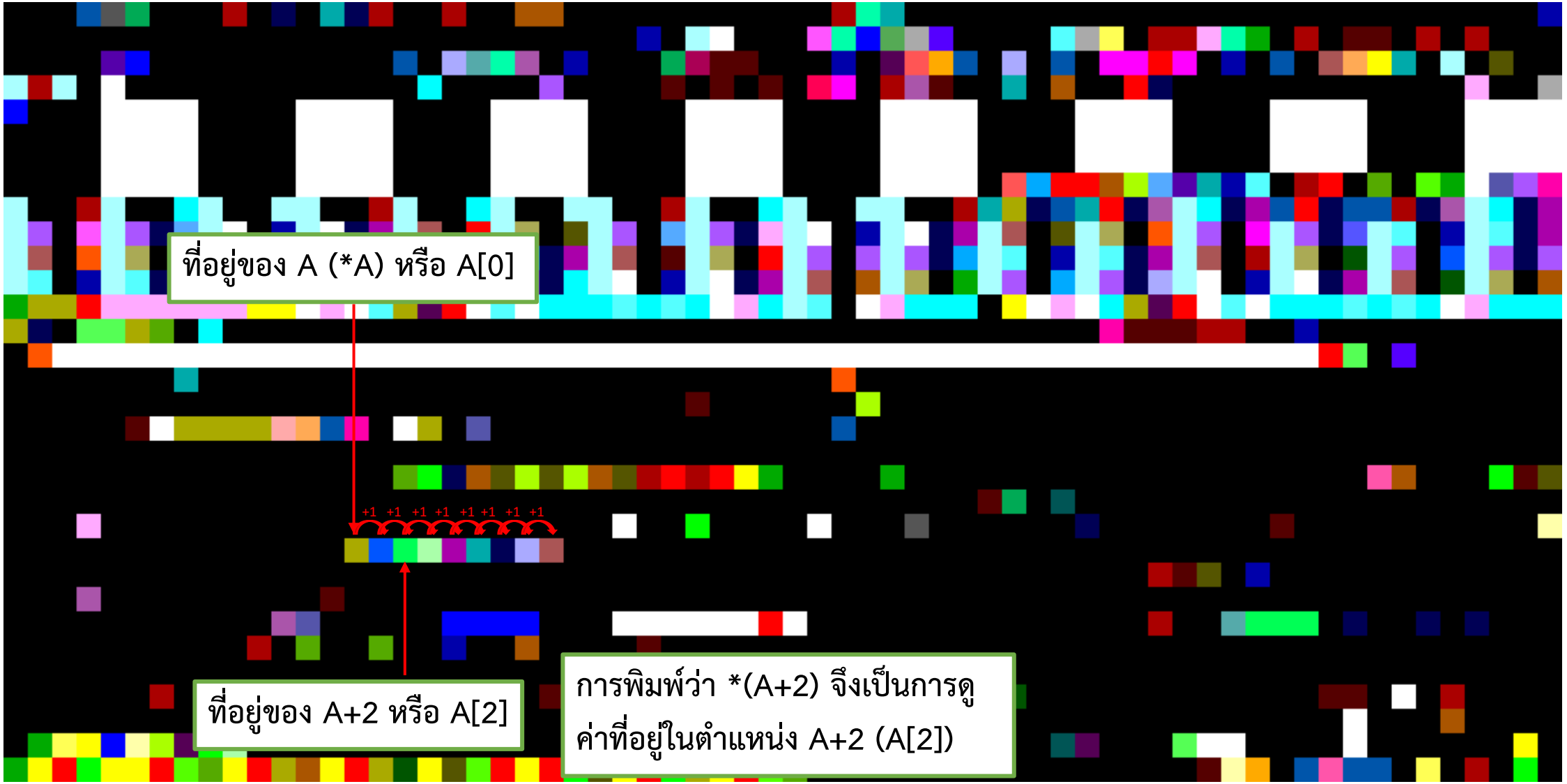
```
array = 6 7 8
array = 6 7 8
```

  - printf ทั้งสองให้ผลลัพธ์เหมือนกันทุกประการ เพราะแท้จริงการเขียนว่า
  - A[0], A[1], A[2], ..., A[i] ก็คือ *(A+0), *(A+1), *(A+2), ..., *(A+i)

ตัวชี้ ในภาษาซี


ที่อยู่ของ A หรือ A[0]
การพิมพ์ว่า *A จึงเป็นการดูค่าที่อยู่ในตำแหน่ง A (A[0])
+1 +1 +1 +1 +1 +1 +1 +1
ที่อยู่ของ A+1 หรือ A[1]
การพิมพ์ว่า *(A+1) จึงเป็นการดูค่าที่อยู่ในตำแหน่ง A+1 (A[1])

ตัวชี้ ในภาษาซี


ที่อยู่ของ A (*A) หรือ A[0]
+1 +1 +1 +1 +1 +1 +1 +1
ที่อยู่ของ A+2 หรือ A[2]
การพิมพ์ว่า *(A+2) จึงเป็นการดู
ค่าที่อยู่ในตำแหน่ง A+2 (A[2])

การประยุกต์ใช้ตัวชี้ : อาเรย์กับตัวชี้

- รูปแบบการใช้อาเรย์ (ผ่าน pointer) กับฟังก์ชัน

สังเกตการส่งพารามิเตอร์ตรงนี้

และการใช้งานอาเรย์ในฟังก์ชัน

```c
#include<stdio.h>

double average(int A[], int n) {
    double sum = 0;
    int i;
    for(i = 0; i < n; ++i) {
        sum += A[i];
    }
    return sum / n;
}
void main() {
    int A[4] = {1, 2, 3, 4};
    double avg = average(A, 4);
    printf("%lf\n", avg);
}
```

การประยุกต์ใช้ตัวชี้ : อาเรย์กับตัวชี้

- รูปแบบการใช้อาเรย์ (ผ่าน pointer) กับฟังก์ชัน
  - วิธีนี้เปลี่ยนพารามิเตอร์จากอาเรย์เป็น pointer โดยชัดแจ้ง

สังเกตการส่งพารามิเตอร์ตรงนี้ ทรงไม่เหมือน Array แล้ว แต่กลายเป็นทรง pointer แทน

แต่เวลาใช้งาน ก็ยังคงเป็นรูปแบบอาเรย์ธรรมดาๆ ในฟังก์ชัน แต่จะใช้ทรงของ pointer ก็ได้ นักศึกษาลองแก้เป็น ptr ได้มั้ย

```c
#include<stdio.h>

double average(int* A, int n) {
    double sum = 0;
    int i;
    for(i = 0; i < n; ++i) {
        sum += A[i];
    }
    return sum / n;
}
void main() {
    int A[4] = {1, 2, 3, 4};
    double avg = average(A, 4);
    printf("%lf\n", avg);
}
```

Outline

- ตัวชี้ ในภาษาซี
- ลักษณะการใช้งานของตัวชี้
- ตัวดำเนินการ & บนตัวแปรทั่วไป
- การประกาศตัวชี้ และการเรียกใช้งาน
- **การประยุกต์ใช้ตัวชี้**
  - เปลี่ยนค่าพารามิเตอร์ที่ส่งไป โดยให้ผลกับผู้เรียกด้วย
  - อาเรย์กับตัวชี้
  - **การส่งคำตอบจากฟังก์ชันมากกว่าหนึ่งตัว**
  - ตัวชี้กับอาเรย์พลวัต (Dynamic array)

การส่งคำตอบจากฟังก์ชันมากกว่าหนึ่งตัว

- โดยปรกติผลลัพธ์ของฟังก์ชันจะถูกส่งกลับไปหาผู้เรียกผ่านคำสั่ง return
  - แต่คำสั่ง return มีข้อจำกัดคือสามารถคืนค่าได้เพียงค่าเดียวเท่านั้น
  - จึงมีปัญหากับฟังก์ชันที่มีผลลัพธ์มากกว่าหนึ่งค่า
- พิจารณากระบวนการรับผลลัพธ์จากฟังก์ชัน

```
int result = add_mult(x, y);
```

  - เห็นได้ว่าเรามักมีตัวแปรมาเก็บผลลัพธ์เอาไว้
  - ตัวแปรเก็บผลลัพธ์แบบนี้มีได้แค่ตัวเดียว
  - สังเกตว่าพารามิเตอร์มีได้มากกว่าหนึ่งตัว
  - ถ้าเราส่งตัวแปรสำหรับเก็บคำตอบไปกับพารามิเตอร์ คำตอบจากฟังก์ชันก็จะมีได้มากกว่าหนึ่งตัว

การส่งคำตอบจากฟังก์ชันมากกว่าหนึ่งตัว

ตัวอย่างโจทย์ 1 จงเขียนฟังก์ชันที่หาตัวเลขค่ามากที่สุดและน้อยที่สุดในอาเรย์ขนาด n ช่องข้อมูล

วิเคราะห์

1. ค่าน้อยที่สุดและมากที่สุดนี้คือผลลัพธ์ซึ่งมีสองค่า
2. เราควรส่งตัวแปรเก็บผลลัพธ์ไปด้วยสองตัวคือ min และ max (ส่งไปให้ฟังก์ชันเปลี่ยนค่ามันให้กลายเป็นคำตอบ)
3. ข้อมูลเข้าของฟังก์ชันคือ อาเรย์ A และจำนวนช่องข้อมูล n
4. แสดงว่าพารามิเตอร์ของฟังก์ชันจะมีทั้งหมดสี่ตัว A, n, min และ max

การส่งคำตอบจากฟังก์ชันมากกว่าหนึ่งตัว


```c
#include<stdio.h>
#include <limits.h>
void min_max(int* A, int N, int* min, int* max)
{
    *max = INT_MIN;
    *min = INT_MAX;
    int i;
    for(i = 0; i < N; ++i) {
        if(A[i] > *max) {
            *max = A[i];
        }
        if(A[i] < *min) {
            *min = A[i];
        }
    }
}
```

การส่งคำตอบจากฟังก์ชันมากกว่าหนึ่งตัว

การเรียกใช้ฟังก์ชัน min_max

```
void main() {
    const int N = 5;
    int Data[5] = {10, 20, 5, 8, 7};
    int min, max;
    min_max(Data, 5, &min, &max);
    printf("min and max = %d and %d.\n", min, max);
}
```

Outline

- ตัวชี้ ในภาษาซี
- ลักษณะการใช้งานของตัวชี้
- ตัวดำเนินการ & บนตัวแปรทั่วไป
- การประกาศตัวชี้ และการเรียกใช้งาน
- **การประยุกต์ใช้ตัวชี้**
  - เปลี่ยนค่าพารามิเตอร์ที่ส่งไป โดยให้ผลกับผู้เรียกด้วย
  - อาเรย์กับตัวชี้
  - การส่งคำตอบจากฟังก์ชันมากกว่าหนึ่งตัว
  - **ตัวชี้กับอาเรย์พลวัต (Dynamic array)**

ตัวชี้กับอาเรย์พลวัต (Dynamic array)

- โดยปรกติแล้วเราสร้างอาเรย์โดยกำหนดขนาดไว้ล่วงหน้า (ใช้ขนาดสูงสุดที่ต้องการใช้มาเป็นตัวกำหนดขนาด สังเกตว่าโจทย์ที่เคยเจอจะบังคับให้เราสร้างอาเรย์ที่ไม่ใหญ่มาก)
- แต่ในกรณีที่เราไม่ทราบขนาดล่วงหน้า เช่น จำนวนข้อมูลต้องเปลี่ยนไปตามข้อมูลที่ผู้ใช้มี เราต้องสร้างอาเรย์ขึ้นมาด้วยขนาดที่กำหนดในภายหลัง
- การสร้างอาเรย์จากขนาดที่ไม่แน่นอนทำได้ด้วยการจัดสรรหน่วยความจำแบบพลวัต (dynamic memory allocation)
  - อาเรย์ที่ได้จากกระบวนการนี้เรียกว่าอาเรย์พลวัต (dynamic array)
- เราใช้คำสั่ง malloc (Memory ALLOCation) สร้างอาเรย์พลวัตขึ้นมา
  - เช่น int* A = (int*) malloc(1000 * sizeof(int));
  - แบบนี้จะเป็นการสร้างอาเรย์พลวัตเก็บจำนวนเต็มจำนวน 1000 ตัว

ตัวชี้กับอาเรย์พลวัต (Dynamic array)

1. เราปฏิบัติกับอาเรย์พลวัตในรูปของตัวชี้
2. คำสั่ง malloc จัดสรรพื้นที่เก็บข้อมูลเป็นจำนวนไบต์ให้เราตามต้องการ
    1. เนื่องจากพื้นที่สำหรับเก็บจำนวนเต็มหนึ่งตัวคือ 4 ไบต์ ซึ่งหาได้อัตโนมัติจากคำสั่ง sizeof(int)
    2. จำนวนเต็มพันจำนวนจึงต้องใช้พื้นที่ 1000 * sizeof(int)
3. คำสั่ง malloc ไม่ระบุชนิดข้อมูลที่อาเรย์จะเก็บ เราจึงต้องระบุไปโดยตรงว่าชนิดข้อมูลที่ต้องการเป็นแบบไหน จึงมีการทำ casting ระบุชนิดข้อมูลว่าเป็น (int*) คือตัวชี้ไปข้อมูลแบบจำนวนเต็มซึ่งก็คืออาเรย์เก็บจำนวนเต็มนั่นเอง
4. * เมื่อได้อาเรย์พลวัตมาแล้ว เราสามารถใช้ตัวชี้ดังกล่าวเหมือนอาเรย์ทั่วไปได้เลย

เรื่องที่ควรใส่ใจ

1. ตัวชี้เป็นแนวคิดที่คนจำนวนมากสับสน เพราะแยกไม่ออกระหว่างตัวแปรทั่วไปกับตัวแปรที่เป็นตัวชี้
2. รูปแบบการเขียนของตัวชี้หรือที่อยู่ของข้อมูลค่อนข้างจะซับซ้อน
   - ต้องใส่ใจว่าควรจะใช้ * และ & หรือไม่ และใช้เมื่อใด
3. ถ้าทำเรื่อง pointer ผิด โปรแกรมเรามักจะแครช เช่นการ scanf แล้วลืม &
   - ความผิดพลาดทั่วไปจะให้แค่ผลลัพธ์ที่ผิด แต่ความผิดพลาดเกี่ยวกับตัวชี้อาจทำให้โปรแกรมหยุดทำงานไปเลย
4. เราใช้ตัวชี้กับฟังก์ชันบ่อย ๆ เพื่อทำให้การส่งข้อมูลเป็นไปโดยสะดวกมากขึ้น โดยเฉพาะเมื่อคำตอบที่ต้องการจากฟังก์ชันมีมากกว่าหนึ่งตัว

ตัวอย่างโจทย์เสริมทักษะ

ตัวอย่าง (1) โปรแกรมนี้ให้ผลลัพธ์อย่างไร

```
void change(int* px, int* py) {
    *px = 5;
    px = py;
}
void main() {
    int x = 0;
    int y = 1;
    int* px = &x;
    int* py = &y;
    change(px, py);
    *px = 2;
    printf("(x, y) = (%d, %d)\n", x, y);
}
```

ตัวอย่างโจทย์เสริมทักษะ

ตัวอย่าง (1) โปรแกรมนี้ให้ผลลัพธ์อย่างไร (พร้อมคำใบ้)

```
void change(int* px, int* py) {
    *px = 5;
    px = py;
}
void main() {
    int x = 0;
    int y = 1;
    int* px = &x;
    int* py = &y;
    change(px, py);
    *px = 2;
    printf("(x, y) = (%d, %d)\n", x, y);
}
```

เรามีคำสั่งเปลี่ยนค่า px ตรงนี้ให้เท่ากับ py

ลองคิดดูว่า จบฟังก์ชันไปแล้ว px ของ main จะชี้ไปที่ใคร

ภายใน change มีการเปลี่ยนค่าทั้ง *px และ px โดยมีการเขียนว่า *px = 5; และ px = py; แต่ว่าคำสั่งไหนบ้างที่มีผลหลังจากที่ฟังก์ชันจบการทำงานแล้ว

คำสั่งนี้เปลี่ยนค่า x หรือเปลี่ยนค่า y

ตัวอย่างโจทย์เสริมทักษะ

ตัวอย่าง (1) โปรแกรมนี้ให้ผลลัพธ์อย่างไร (เฉลยและคำอธิบาย)

```
void change(int* px, int* py) {
    *px = 5;
    px = py;
}
void main() {
    int x = 0;
    int y = 1;
    int* px = &x;
    int* py = &y;
    change(px, py);
    *px = 2;
    printf("(x, y) = (%d, %d)\n", x, y);
}
```

(1) จริงๆแล้วตรงนี้เป็นการแก้สำเนาของที่อยู่ เหมือนการแก้สำเนาทะเบียนบ้าน แน่นอนว่าทะเบียนบ้านตัวจริงไม่เปลี่ยน

(2) แต่การเปลี่ยนค่าที่อยู่ในตำแหน่งที่สำเนาทะเบียนบ้านชี้ตอนแรก อันนี้ส่งผลแน่นอน (ตอนนี้ค่า x ตัวจริงจะถูกเปลี่ยนเป็น 5)

px ข้างนอกก็ยังเป็นตัวเดิม ที่แก้ไปในฟังก์ชันนั้นเป็นเพียงสำเนา (ตอนนี้ค่า x ตัวจริงจะถูกเปลี่ยนเป็น 2)

ส่วนค่าของ y ไม่เคยเปลี่ยนเลย จึงเป็น 1 เท่าเดิม

```
(x, y) = (2, 1)
```

ตัวอย่างโจทย์เสริมทักษะ

ตัวอย่าง (2) โปรแกรมนี้ให้ผลลัพธ์อย่างไร

```
void change(int* px, int* py) {
    px = py;
    *px = 5;
}
void main() {
    int x = 0;
    int y = 1;
    int* px = &x;
    int* py = &y;
    change(px, py);
    *px = 2;
    printf("(x, y) = (%d, %d)\n", x, y);
}
```

ตัวอย่างโจทย์เสริมทักษะ

ตัวอย่าง (2) โปรแกรมนี้ให้ผลลัพธ์อย่างไร (พร้อมคำใบ้)

```
void change(int* px, int* py) {
    px = py;
    *px = 5;
}
void main() {
    int x = 0;
    int y = 1;
    int* px = &x;
    int* py = &y;
    change(px, py);
    *px = 2;
    printf("(x, y) = (%d, %d)\n", x, y);
}
```

(1) เหมือนข้อก่อนหน้านี้ แต่สลับคำสั่งใน change เท่านั้น

(2) ข้อนี้ต้องดูให้ดี เพราะค่าตำแหน่ง px และ py สลับกันตั้งแต่แรก

(3) แล้ว px ตรงนี้เปลี่ยนที่อยู่ไปด้วยหรือไม่ เป็นค่าของ x หรือ y ที่จะต้องเปลี่ยนเป็น 2

ตัวอย่างโจทย์เสริมทักษะ

ตัวอย่าง (2) โปรแกรมนี้ให้ผลลัพธ์อย่างไร (เฉลยพร้อมคำอธิบาย)

```
void change(int* px, int* py) {
    px = py;
    *px = 5;
}
void main() {
    int x = 0;
    int y = 1;
    int* px = &x;
    int* py = &y;
    change(px, py);
    *px = 2;
    printf("(x, y) = (%d, %d)\n
}
```

(1) เมื่อรับค่าที่อยู่ของ x และ y มาแล้ว ให้ px เปลี่ยนสำเนาทะเบียนบ้านที่ตัวเองถือเป็นของ py

(2) แก้ไขค่าที่อยู่ในตำแหน่งนั้นเป็น 5 ดังนั้น ค่า y เปลี่ยน

(3) แต่แก้ในฟังก์ชันก็เป็นเพียงการแก้สำเนา px ต้นฉบับข้างนอกยังคงเก็บที่อยู่ (ทะเบียนบ้าน) ของ x เหมือนเดิม ดังนั้นตรงนี้ ค่า x จะเปลี่ยนเป็น 2

```
(x, y) = (2, 5)
```

สรุป

- ตัวชี้ ในภาษาซี *(บอกตำแหน่งของตัวแปรบน memory คล้ายเลขที่บ้าน สามารถดูและแก้ไขค่าบนนั้นได้โดยตรง)*
- ลักษณะการใช้งานของตัวชี้ *(ใช้พัฒนาขีดความสามารถของฟังก์ชัน อาเรย์ และงานอื่นๆได้)*
- ตัวดำเนินการ & บนตัวแปรทั่วไป *(บอกที่อยู่ของตัวแปรตัวนั้น trick: อ่านว่า “ที่อยู่ของ”)*
- การประกาศตัวชี้ และการเรียกใช้งาน *(เมื่อจะประกาศ พิมพ์หน้าตัวแปรหรือหลัง datatype ด้วย *, ตอนเรียกใช้ อาจใช้ Trick ว่า ถ้าพบเครื่องหมาย * ให้อ่านว่า “ที่อยู่ของ” และ & ให้อ่านว่า “ค่าที่อยู่ในตำแหน่ง”)*
- การประยุกต์ใช้ตัวชี้
  - เปลี่ยนค่าพารามิเตอร์ที่ส่งไป โดยให้ผลกับผู้เรียกด้วย *(ส่งเป็นเลขที่บ้าน ต่อให้เป็นสำเนา ถ้าเราไปถึงเลขที่บ้านนั้นเราก็แก้ต้นฉบับได้)*
  - อาเรย์กับตัวชี้ *(ตัวชี้ชี้ที่อาเรย์ตัวแรก ส่วนการเข้าถึงตัวถัดไป อาจใช้แบบนี้ *(A+1) ก็ได้)*
  - การส่งคำตอบจากฟังก์ชันมากกว่าหนึ่งตัว *(ส่งที่อยู่ของตัวแปรไปให้ฟังก์ชันเปลี่ยนค่าในตัวแปรนั้นโดยตรง ทีนี้จะเปลี่ยนค่าตัวแปรกี่ตัวก็สามารถทำได้)*
  - ตัวชี้กับอาเรย์พลวัต (Dynamic array) *(ประกาศอาเรย์ที่ไม่กำหนดขนาดตายตัว ใช้ malloc กับ sizeof คู่กัน และอย่าลืม cast ข้อมูลเป็นชนิดที่ต้องการด้วย เช่น (int*) เป็นต้น)*