



# การเขียนโปรแกรมคอมพิวเตอร์ 1

## Computer Programming I

### ตัวชี้ (Pointer)

ภิญโญ แท้ประสาทสิทธิ์

Emails : pinyotae+111 at gmail dot com, pinyo at su.ac.th

Web : <http://www.cs.su.ac.th/~pinyotae/compro1/>

Facebook Group : [ComputerProgramming@CPSU](https://www.facebook.com/ComputerProgramming@CPSU)

ภาควิชาคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศิลปากร

สัปดาห์ที่ 11



### หัวข้อเนื้อหา

- ตัวชี้คืออะไร
- เราใช้ตัวชี้ทำอะไร
- ตัวดำเนินการ & ที่อยู่หน้าตัวแปร
- การประกาศและตัวอย่างการใช้ตัวชี้
- การประยุกต์ใช้ตัวชี้
  - เปลี่ยนค่าพารามิเตอร์ที่ส่งไป โดยให้ผลที่ฟังก์ชันผู้เรียกด้วย
  - อาเรย์
  - การส่งคำตอบจากฟังก์ชันเมื่อมีคำตอบมากกว่าหนึ่ง

12 มกราคม 2556

ภิญโญ แท้ประสาทสิทธิ์ มหาวิทยาลัยศิลปากร

2

### ตัวชี้คืออะไร



- เป็นชนิดข้อมูลประเภทหนึ่งและจัดเป็นชนิดข้อมูลขั้นสูง (เรานิยมเรียกตัวชี้แบบทับศัพท์ว่า pointer)
- ตัวชี้ไม่ได้เก็บข้อมูลโดยตรง แต่เก็บที่อยู่ของข้อมูลแทน
- โดยปรกติแล้วตัวแปรทุกตัวอยู่ในหน่วยความจำเครื่อง (RAM) และมีตำแหน่งที่อยู่กำกับไว้คล้ายเลขที่บ้าน



- แท้จริงแล้วที่อยู่ของตัวแปร x กับค่าของ x เป็นของคู่กันตลอด
- แต่ก่อนเราสนใจแต่ค่าของ x ไม่ได้สนใจที่อยู่ของ x
- การรู้ที่อยู่ของ x จะทำให้เราอ่านหรือเปลี่ยนค่าของ x ได้
  - การใช้ที่อยู่ของตัวแปรทำให้เราจัดการกับตัวแปรได้หลากหลายขึ้นมาก
  - ถ้าใช้อย่างถูกต้องและเหมาะสมมันจะทำประโยชน์ในโปรแกรมได้มาก

12 มกราคม 2556

ภิญโญ แท้ประสาทสิทธิ์ มหาวิทยาลัยศิลปากร

3

### เราใช้ตัวชี้หรือที่อยู่ของข้อมูลทำอะไร



- ใช้ก้าวข้ามขีดจำกัดของการรับส่งพารามิเตอร์ของฟังก์ชัน
  - พารามิเตอร์ที่ส่งไปให้ฟังก์ชัน โดยปรกติแล้วจะเป็นตัวแปรทั่วไป ไม่ใช่ตัวชี้
  - เมื่อฟังก์ชันแก้ค่าพารามิเตอร์ในตัวแปรทั่วไป ค่าที่เปลี่ยนไปนั้นจะมีผลอยู่เฉพาะในฟังก์ชัน (เพราะมันเป็นแค่สำเนาของค่าที่ส่งไป)
  - แล้วถ้าเราอยากให้ฟังก์ชันแก้ค่าตัวแปรต้นฉบับด้วยล่ะ จะทำอย่างไร?
  - ที่จริงแล้วเราใช้ที่อยู่ของข้อมูลกับ scanf มาตลอด
- เราใช้ตัวชี้จัดการอาเรย์
  - ➔ แท้จริงแล้วอาเรย์เป็นตัวชี้แบบหนึ่ง คือมันชี้ไปที่ข้อมูลตัวแรกของอาเรย์
- เราใช้ตัวชี้จัดการข้อมูลที่สร้างขึ้นแบบพลวัต (dynamic allocation)

12 มกราคม 2556

ภิญโญ แท้ประสาทสิทธิ์ มหาวิทยาลัยศิลปากร

4

## ตัวดำเนินการ & ที่อยู่หน้าตัวแปร



- ณ จุดนี้มีแนวคิดสองอย่างที่แตกต่างกันแต่สัมพันธ์กัน คือ
  - ที่อยู่ของตัวแปร (address) ซึ่งเปรียบเหมือนเลขที่บ้านของตัวแปร
  - ตัวแปรที่ใช้บันทึกเลขที่บ้าน (ตัวชี้/pointer)
- ที่อยู่ของตัวแปรเป็นสิ่งที่มีความสำคัญ และเราสามารถเรียกดูได้ผ่านการใช้เครื่องหมาย &
  - เช่น &x หมายถึงที่อยู่ของตัวแปร x
  - & เป็นสิ่งที่ใช้ได้กับตัวแปรทุกชนิด แต่ห้ามใช้กับค่าคงที่ทั่วไป เพราะค่าคงที่ทั่วไปไม่ใช่ตัวแปร
- แทนที่เราจะต้องคอยเขียนว่า &x อยู่ตลอด เราสามารถใช้ตัวชี้มาเก็บค่าที่อยู่ไว้ได้ เช่น `int* ptr = &x;` เป็นการเก็บที่อยู่ของตัวแปร x ไว้ที่ ptr

## การประกาศตัวชี้



- การประกาศตัวชี้จะใช้ \* ตามหลังชนิดข้อมูลของตัวแปรที่มันจะชี้ เช่น
  - `int*` เป็นตัวชี้ไปยังข้อมูลชนิดจำนวนเต็ม
  - `char*` เป็นตัวชี้ไปยังข้อมูลชนิดอักขระ
  - `double*` เป็นตัวชี้ไปยังข้อมูลชนิดทศนิยมความเที่ยงทวิคูณ
- คำว่า `int*`, `char*`, และ `double*` ทำนองนี้เป็นชนิดข้อมูลในตัวของมันเอง
  - ย้ำว่าตัวชี้แท้จริงเป็นชนิดข้อมูล (data type) ชั้นสูงชนิดหนึ่ง
- การประกาศตัวชี้ก็คือการประกาศตัวแปร ดังนั้นจึงมีชนิดข้อมูล ตามด้วยชื่อ
  - เช่น `int* px;`    `char* pc;`    `double* pd;`
  - แม้ไม่ใช่เรื่องบังคับ แต่คนนิยมตั้งชื่อตัวชี้ให้มีตัวอักษร p หรือคำว่า ptr นำหน้า

## การใช้ตัวชี้



- มีอยู่สองลักษณะ คือ
  1. ใช้บันทึกค่าที่อยู่ของตัวแปรที่เราสนใจ
  2. ใช้อ่านค่าหรือเปลี่ยนค่าตัวแปรที่เราสนใจ
- การใช้งานในแต่ละลักษณะจะมีวิธีเขียนที่ไม่เหมือนกัน ดังนี้
  - ตอนบันทึกค่าที่อยู่ของตัวแปรที่เราสนใจ เราจะใช้ชื่อของตัวชี้ตรง ๆ เช่น

```
int* ptr;
int x = 5;
ptr = &x;
```
  - ตอนอ่านหรือเขียนค่าตัวแปรที่สนใจเราใส่ \* ไว้หน้าชื่อตัวชี้

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

## ดูตัวอย่างโปรแกรม



```
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 ตัวแปรที่มันชี้จะถูกลบค่าตามไปด้วย

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

## การประยุกต์ใช้กับฟังก์ชัน



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

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

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

อันนี้เป็นการใช้ scanf ในรูปแบบที่เราคุ้นเคย เราใช้ &x ที่อยู่ของตัวแปร x สังเกตได้ว่า scanf เปลี่ยนค่า x ได้เพราะมันแก้ค่า x ผ่านที่อยู่ของ x

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

เราส่งที่อยู่ของ x ผ่าน ptr แบบนี้ก็ได้อีก เพราะ ptr = &x

## ทบทวนเรื่องการส่งค่าและเปลี่ยนค่าพารามิเตอร์



```
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);
}
```

ได้เลข 5 เหมือนตอนต้น

ได้เลข 14

## ย้อนกลับมาดูที่ scanf



- scanf ต่างกับฟังก์ชันที่แสดงมาเมื่อสักครู่ตรงชนิดของพารามิเตอร์
- พารามิเตอร์ที่ส่งไปให้ scanf ถูกเปลี่ยนค่าจริง ๆ แต่พารามิเตอร์ที่ส่งไปให้ add\_mult ไม่ถูกเปลี่ยนค่า
- แสดงว่าในกรณีที่เราต้องการเปลี่ยนค่าพารามิเตอร์ที่เป็นตัวแปรของคนเรียกใช้ฟังก์ชัน เราต้องใช้ที่อยู่ของข้อมูลหรือตัวชี้เป็นพารามิเตอร์
- แล้วเราจะประกาศฟังก์ชันที่ใช้ตัวชี้หรือที่อยู่ของข้อมูลได้อย่างไร ?

## พารามิเตอร์ที่เป็นที่อยู่ของข้อมูลหรือตัวชี้



- พารามิเตอร์ที่รับที่อยู่ของตัวแปรหรือตัวชี้ได้ต้องมีชนิดข้อมูลเป็นแบบตัวชี้
- สมมติว่าเราต้องการแก้ไข x ใน add\_mult เป็นแบบตัวชี้เราก็ต้องใช้เครื่องหมาย \* ตามหลังชนิดข้อมูลของ x เช่น

แบบเดิม

```
int add_mult(int x, int y) {
    x = x + y;
    x = x * y;
    return x;
}
```

แบบตัวชี้

```
int add_mult_ptr(int* x, int y) {
    *x = *x + y;
    *x = *x * y;
    return *x;
}
```

## เปรียบเทียบผลลัพธ์



```
void main() {
    int x, y, result;
    x = 5; y = 2;

    result = add_mult(x, y);
    printf("%d\n", x);
    printf("%d\n\n", result);

    result = add_mult_ptr(&x, y);
    printf("%d\n", x);
    printf("%d\n\n", result);
}
```

แบบเดิมไม่เปลี่ยนค่า x ดังนั้นตรงนี้แสดงค่าออกมาเป็น 5 และ 14 ตามลำดับ

แบบนี้ค่า x ที่ส่งไปจะได้รับการแก้ไขภายใน add\_mult\_ptr และมีผลกับ x ใน main ด้วย

พิมพ์ 14  
14

12 มกราคม 2556

ภิญโญ แท้ประสาทสิทธิ์ มหาวิทยาลัยศิลปากร

13

## ขยายความเรื่องตัวชี้กับที่อยู่ของข้อมูล



- ตัวชี้ใช้เก็บที่อยู่ของข้อมูล ส่วนที่อยู่ของข้อมูลเป็นสิ่งที่เกิดขึ้นมากับตัวแปร
- พารามิเตอร์ที่เราประกาศไว้ตรงหัวฟังก์ชัน เช่น  
`int add_mult(int x, int y)` และ `int add_mult_ptr(int* x, int y)` อยู่ในรูปตัวแปรตลอด เพราะเราไม่รู้ว่าจะเรียกจะใส่ค่าอะไรเข้ามา
- ตัวแปรพารามิเตอร์ทั่วไป (ตัวแปร y ในที่นี้) อาจจะได้รับค่าคงที่มาจากโดยตรง เช่น เลข 3 หรือ ตัวแปรจำนวนเต็มจาก main มากก็ได้
- ตัวแปรพารามิเตอร์แบบตัวชี้ (ตัวแปร `int* x`) อาจจะได้รับค่าคงที่ ที่อยู่ของตัวแปรมาโดยตรง เช่น `&x` หรือจะรับตัวแปรชนิดตัวชี้จาก main มากก็ได้
  - นั่นคือ ถ้าหากว่าเรามี `int* ptr = &x;` ใน main เราเรียกฟังก์ชันโดยส่ง ptr ไปก็ได้ เช่น `add_mult_ptr(ptr, y);`

12 มกราคม 2556

ภิญโญ แท้ประสาทสิทธิ์ มหาวิทยาลัยศิลปากร

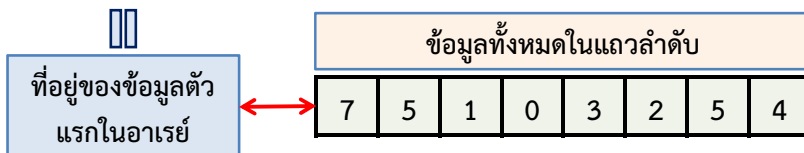
14

## ตัวชี้กับแถวลำดับ



- แถวลำดับแท้จริงเป็นตัวชี้ในรูปแบบหนึ่ง การรู้จักวิธีส่งผ่านแถวลำดับไปเป็นพารามิเตอร์ของฟังก์ชันจะช่วยให้เราทำได้มาก
  - ➔ เราต้องเข้าใจพื้นฐานของตัวชี้ ที่ถูกต้องเสียก่อน
- ของสามอย่าง : ตัวแปรแถวลำดับ, ที่อยู่ของข้อมูล, และ ข้อมูลในแถวลำดับ
  - ค่าตัวแปรแถวลำดับที่จริงเก็บที่อยู่ของข้อมูลตัวแรกในแถวลำดับไว้
  - ตัวแปรแถวลำดับทำให้รูปแบบการอ้างถึงข้อมูลดูเข้าใจง่ายขึ้น

ตัวแปรอาเรย์



12 มกราคม 2556

ภิญโญ แท้ประสาทสิทธิ์ มหาวิทยาลัยศิลปากร

15

## ทดลองใช้ตัวแปรแถวลำดับในฐานะตัวชี้



- จำไว้ว่าการอ่านเขียนข้อมูลที่ตัวชี้อ้างถึงอยู่ต้องใช้เครื่องหมาย \* นำหน้า
- บ้านเลขที่มักเป็นเลขติดต่อกันไป ที่อยู่ของข้อมูลในแถวลำดับก็เป็นแบบนั้น

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

เราใส่ตำแหน่งอาเรย์ตามลำดับ 0, 1, 2

```
printf("array = %d %d %d\n",
       *(A+0), *(A+1), *(A+2));
```

แท้จริงมันก็คือการไล่ไปตามที่อยู่ของข้อมูลในช่องถัดไปนั่นเอง

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

12 มกราคม 2556

ภิญโญ แท้ประสาทสิทธิ์ มหาวิทยาลัยศิลปากร

16

## ส่งอาเรย์ไปเป็นพารามิเตอร์ของฟังก์ชัน (1)



- มาดูลูปแบบที่ถูกต้องและใช้กันบ่อยสำหรับอาเรย์หนึ่งมิติ

```
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);
}
```

12 มกราคม 2556

ภิญโญ แท้ประสาทสิทธิ์ มหาวิทยาลัยศิลปากร

17

## ส่งอาเรย์ไปเป็นพารามิเตอร์ของฟังก์ชัน (2)



- จะเปลี่ยนพารามิเตอร์ให้กลายเป็นตัวชี้โดยชัดแจ้งก็ได้เช่นกัน

```
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);
}
```

ถึงจะเขียนไว้ข้างบนว่าเป็นตัวชี้ แต่เราก็ใช้มันในรูปแบบอาเรย์ได้เหมือนกัน

12 มกราคม 2556

ภิญโญ แท้ประสาทสิทธิ์ มหาวิทยาลัยศิลปากร

18

## ฟังก์ชันที่ต้องการให้ผลลัพธ์มากกว่าหนึ่งค่า



- โดยปรกติผลลัพธ์ของฟังก์ชันจะถูกส่งกลับไปหาผู้เรียกผ่านคำสั่ง return
- แต่คำสั่ง return คืนค่าได้เพียงค่าเดียวเท่านั้น  
→ มีปัญหาเกี่ยวกับฟังก์ชันที่มีผลลัพธ์มากกว่าหนึ่งค่า

- พิจารณากระบวนการรับผลลัพธ์จากฟังก์ชัน

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

- เห็นได้ว่าเรามักมีตัวแปรมาเก็บผลลัพธ์เอาไว้
- ตัวแปรเก็บผลลัพธ์แบบนี้มีได้แค่ตัวเดียว
- สังเกตว่าพารามิเตอร์มีได้มากกว่าหนึ่งตัว
- ถ้าเราส่งตัวแปรสำหรับเก็บคำตอบไปกับพารามิเตอร์  
→ คำตอบจากฟังก์ชันก็จะมีได้มากกว่าหนึ่งตัว

12 มกราคม 2556

ภิญโญ แท้ประสาทสิทธิ์ มหาวิทยาลัยศิลปากร

19

## การประยุกต์ใช้ตัวชี้เก็บผลลัพธ์จากฟังก์ชัน



เราสามารถระบุค่าพารามิเตอร์ที่เป็นตัวชี้ได้ ซึ่งค่าที่ระบุไว้ในฟังก์ชันจะส่งผลกลับมาสู่ตัวแปรในฟังก์ชันที่เป็นผู้เรียกใช้ได้ (โดยมากผู้เรียกใช้คือ main)

ตัวอย่าง จงเขียนฟังก์ชันที่หาตัวเลขค่ามากที่สุดและน้อยที่สุดในอาเรย์ขนาด n ช่องข้อมูล

### วิเคราะห์

- ค่าน้อยที่สุดและมากที่สุดนี้คือผลลัพธ์ซึ่งมีสองค่า
- เราควรส่งตัวแปรเก็บผลลัพธ์ไปด้วยสองตัวคือ min และ max
- ข้อมูลเข้าของฟังก์ชันคือ อาเรย์ A และจำนวนช่องข้อมูล n
- แสดงว่าพารามิเตอร์ของฟังก์ชันจะมีทั้งหมดสี่ตัว A, n, min และ max

12 มกราคม 2556

ภิญโญ แท้ประสาทสิทธิ์ มหาวิทยาลัยศิลปากร

20

## ฟังก์ชัน min\_max



```
#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];
        }
    }
}
```

12 มกราคม 2556

ภิญโญ แท้ประสาทสิทธิ์ มหาวิทยาลัยศิลปากร

21

## ทดสอบการใช้ min\_max



```
void min_max(int* A, int N, int* min, int* max)
{
    ...
}

int 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);

    return 0;
}
```

12 มกราคม 2556

ภิญโญ แท้ประสาทสิทธิ์ มหาวิทยาลัยศิลปากร

22

## ตัวชี้กับอาร์เรย์พลวัต (dynamic array)



- โดยปรกติแล้วเราสร้างอาร์เรย์โดยกำหนดขนาดไว้ล่วงหน้า (ใช้ขนาดสูงสุดที่ต้องการใช้มาเป็นตัวกำหนดขนาด)
- แต่ในกรณีที่เราไม่ทราบขนาดล่วงหน้า เพราะมันเปลี่ยนไปตามข้อมูลที่ผู้ใช้มี เราต้องสร้างอาร์เรย์ขึ้นมาด้วยขนาดที่กำหนดในภายหลัง
- การสร้างอาร์เรย์จากขนาดที่ไม่แน่นอนทำได้ด้วยการจัดสรรหน่วยความจำแบบพลวัต (dynamic memory allocation)
- อาร์เรย์ที่ได้จากกระบวนการนี้เรียกว่าอาร์เรย์พลวัต (dynamic array)
- เราใช้คำสั่ง malloc (Memory ALLOCation) สร้างอาร์เรย์พลวัตขึ้นมา เช่น `int* A = (int*) malloc(1000 * sizeof(int));` เป็นการสร้างอาร์เรย์พลวัตเก็บจำนวนเต็มจำนวน 1000 ตัว

12 มกราคม 2556

ภิญโญ แท้ประสาทสิทธิ์ มหาวิทยาลัยศิลปากร

23

## สังเกตรูปแบบการเขียนโค้ดสำหรับอาร์เรย์พลวัต



1. เราปฏิบัติกับอาร์เรย์พลวัตในรูปของตัวชี้
  2. คำสั่ง malloc จัดสรรพื้นที่เก็บข้อมูลเป็นจำนวนไบต์ให้เราตามต้องการ
    - เนื่องจากพื้นที่สำหรับเก็บจำนวนเต็มหนึ่งตัวคือ 4 ไบต์ ซึ่งหาได้อัตโนมัติจากคำสั่ง `sizeof( int )`
    - จำนวนเต็มพันจำนวนจึงต้องใช้พื้นที่ `1000 * sizeof( int )`
  3. คำสั่ง malloc ไม่ระบุชนิดข้อมูลที่อาร์เรย์จะเก็บ เราจึงต้องระบุไปโดยตรงว่าชนิดข้อมูลที่ต้องการเป็นแบบไหน จึงมีการทำ casting ระบุชนิดข้อมูลว่าเป็น `(int*)` คือตัวชี้ไปข้อมูลแบบจำนวนเต็มซึ่งก็คืออาร์เรย์เก็บจำนวนเต็มนั่นเอง
- \* เมื่อได้อาร์เรย์พลวัตมาแล้ว เราสามารถใช้ตัวชี้ดังกล่าวเหมือนอาร์เรย์ทั่วไปได้เลย

12 มกราคม 2556

ภิญโญ แท้ประสาทสิทธิ์ มหาวิทยาลัยศิลปากร

24

## เรื่องควรใส่ใจเกี่ยวกับตัวชี้



- ตัวชี้เป็นแนวคิดที่คนจำนวนมากสับสน เพราะแยกไม่ออกระหว่างตัวแปรทั่วไปกับตัวแปรที่เป็นตัวชี้
- รูปแบบการเขียนของตัวชี้หรือที่อยู่ของข้อมูลค่อนข้างจะซับซ้อน  
→ ต้องใส่ใจว่าจะจะใช้ \* และ & หรือไม่ และใช้เมื่อใด
- ถ้าทำเรื่องนี้ผิด โปรแกรมเรามักจะแฉกช ความผิดพลาดทั่วไปจะให้แก่ผลลัพธ์ที่ผิด แต่ความผิดพลาดเกี่ยวกับตัวชี้ อาจทำให้โปรแกรมหยุดทำงานไปเลย
- เราใช้ตัวชี้กับฟังก์ชันบ่อย ๆ เพื่อให้การส่งข้อมูลเป็นไปโดยสะดวกมากขึ้น โดยเฉพาะเมื่อคำตอบที่ต้องการจากฟังก์ชันมีมากกว่าหนึ่งตัว

## แบบฝึกหัด 1: โปรแกรมจะพิมพ์อะไรออกมา



```
void change(int* px, int* py) {  
    *px = 5;  
    px = py;  
}  
  
int 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  
คำถามมีอยู่ว่า จบฟังก์ชันไปแล้ว px ของ main จะชี้ไปที่ใคร

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

ตกลงคำสั่งนี้เปลี่ยนค่า x หรือเปลี่ยนค่า y กันแน่

## แบบฝึกหัด 2: โปรแกรมนี้จะพิมพ์อะไรออกมา



```
void change(int* px, int* py) {  
    px = py;  
    *px = 5;  
}  
  
int 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);  
}
```