การเขียนโปรแกรมคอมพิวเตอร์ 1

Computer Programming I

สตรัค (Struct)

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

ภาควิชาคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศิลปากร

สัปดาห์ที่ 13

หัวข้อเนื้อหา

- สตรัค (struct) และ ประโยชน์ของสตรัค
- ก้าวแรกกับการประกาศสตรัค
- การประกาศสตรัคด้วย typedef
- การเข้าถึงข้อมูลในสตรัค
- การรับและแสดงผลข้อมูลในสตรัค
- สตรัคในสตรัค
- อาเรย์ของสตรัค และ อาเรย์ในสตรัค

สตรัค (struct) และ ประโยชน์ของสตรัค

- สตรัคเป็นโครงสร้างข้อมูลที่รวบรวมข้อมูลหลายอย่างไว้ด้วยกัน
  - ข้อมูลแต่ละอย่างนั้นจะเป็นชนิดเดียวกันหรือต่างกันก็ได้
  - อาเรย์มีข้อมูลหลายอย่างได้แต่ชนิดข้อมูลเป็นแบบเดียวกันหมด
- **สตรัคเป็นการสร้างชนิดข้อมูลตัวใหม่ขึ้นมา**
- ประโยชน์ของสตรัคคือการรวบรวมข้อมูลหลายตัวที่มีความสัมพันธ์กันในฐานะเป็นองค์ประกอบย่อยของข้อมูลมาอยู่ด้วยกัน
  - เช่น ชื่อ นามสกุล ที่อยู่ อายุ หมายเลขโทรศัพท์ เงือนเดือน ของพนักงาน ควรถูกเก็บไว้ด้วยกันเพราะต่างก็เป็นองค์ประกอบย่อยของข้อมูลพนักงาน
  - ชื่อจังหวัด รายการชื่ออำเภอในจังหวัด จำนวนประชากร และ งบประมาณ ควรถูกเก็บไว้ด้วยกันเพราะต่างก็เป็นองค์ประกอบย่อยของข้อมูลจังหวัด

ก้าวแรกกับการประกาศสตรัค

1. เราใช้คีย์เวิร์ด (key word) ว่า struct นำหน้า ตามด้วยชื่อสตรัคที่เราอยากให้เป็น เช่น

```
struct employee {
    …
};
```

2. ระบุรายการข้อมูลที่ต้องการลงในสตรัค

```
struct employee {
  char name[16];
  char surname[31];
  char address[151];
  float salary;
};
```

ผลของการประกาศสตรัค

- เราได้ชนิดข้อมูลใหม่ขึ้นมา จากตัวอย่างที่แล้วคือ struct employee;
- เราสามารถใช้ struct employee ไปประกาศเป็นชนิดข้อมูลของตัวแปร ได้ราวกับว่ามันเป็นชนิดข้อมูลทั่วไป เช่น

```
struct employee emp;
```

- จากตัวอย่าง emp เป็นชื่อตัวแปรที่มีชนิดข้อมูลคือ struct employee
- สังเกตด้วยว่าเราต้องใช้คำว่า struct คู่กับ employee เสมอ
  - เขียนแบบนี้แล้วโค้ดยาวเยิ่นเย้อมาก
  - เราย่อชื่อชนิดข้อมูลเป็นตัวใหม่ได้ด้วยคีย์เวิร์ด typedef

การประกาศสตรัคด้วย typedef

- เป็นการประกาศชื่อเรียกอีกอย่างให้กับชนิดข้อมูล
  - ชนิดข้อมูลอันหนึ่งสามารถมีชื่อเรียกได้หลายชื่อ
  - ในที่นี้เราจะทำให้ struct employee มีชื่อเรียกสั้น ๆ ว่า EMPLOYEE

```
typedef struct {
  char name[16];
  char surname[31];
  char address[151];
  float salary;
} EMPLOYEE;
```

- ต่อมาเราสามารถใช้คำว่า EMPLOYEE ในการประกาศตัวแปรได้เลย

```
EMPLOYEE emp;
```

การใช้ typedef กับการประกาศ struct อีกแบบ

- วิธีประกาศแบบเดิมทำให้ struct มีชื่อว่า EMPLOYEE แบบเดียว เราจะไม่สามารถใช้ชื่อชนิดข้อมูลว่า struct employee ได้
- ถ้าอยากให้มีหลายชื่อ เราสามารถประกาศ typedef ไว้ด้านท้าย เช่น

```
struct employee {
   char name[16];
   char surname[31];
   char address[151];
   float salary;
} typedef EMPLOYEE;
```

- วิธีนี้จะทำให้ได้ชื่อชนิดข้อมูลเดียวกันมาสองชื่อคือ struct employee และ EMPLOYEE มาเฉย ๆ

การเข้าถึงข้อมูลในตัวแปรสตรัค

- เราใช้เครื่องหมายจุด (dot) ในการเข้าถึงข้อมูลในตัวแปรสตรัค
- **เน้น** ต้องอ้างถึงตัวแปร ไม่ใช่ชนิดข้อมูล
  - แบบนี้ได้

```
EMPLOYEE emp;
emp.salary = 18000;
```

  - แบบนี้ไม่ได้ เพราะไปอ้างผ่านชนิดข้อมูล

```
EMPLOYEE emp;
EMPLOYEE.salary = 18000;
```

- เรื่องนี้เข้าใจได้ง่าย เพราะ EMPLOYEE อาจมีได้หลายคน เราจึงต้องระบุข้อมูลใน EMPLOYEE แยกย่อยเจาะจงเป็นรายบุคคลไป

ตัวอย่างการเข้าถึงข้อมูลในสตรัค

- เราปฏิบัติกับข้อมูลแต่ละตัวในสตรัคที่เราเข้าถึงเหมือนตัวแปรโดดทั่วไป
  - สังเกตได้จากการกำหนดค่าเงินเดือนพนักงาน

```
EMPLOYEE emp;
emp.salary = 18000;
```

  - ในตอนที่เราจะแสดงผล เราก็สามารถทำได้ตามปรกติ

```
printf("%f", emp.salary);
```

- พวกที่เป็นแบบข้อความเราก็กำหนดสตริงเข้าไปได้เลย
  - แต่ต้องระวังว่าความยาวของสตริงจะไม่เกินที่เก็บข้อมูลในอาเรย์ที่เตรียมไว้

```
emp.name = "Pinyo";    แบบนี้ได้
emp.name = "Sanamchandra Palace" แบบนี้ยาวเกินไป
```

การรับข้อมูลเข้าไปเก็บไว้ในสตรัค

- เวลาที่เราใช้ scanf เราจะใช้มันกับตัวแปรแต่ละตัว
  - เราไม่สามารถ scanf กับสตรัคทั้งก้อนรวดเดียวได้
  - ต้องทำทีละตัว และต้องระบุชนิดข้อมูลใน scanf ให้ถูกต้องด้วย

```
scanf("%s", emp.name);
scanf("%s", emp.surname);
scanf("%f", &emp.salary);
```

- กฎเกณฑ์การรับข้อมูลเข้าด้วย scanf เหมือนเดิมทุกประการ คือเรายังต้องส่งที่อยู่ของตัวแปรไปให้
  - ยังต้องใช้ & นำหน้าชื่อตัวแปรทั่วไป เช่น &emp.salary
  - แต่ไม่ต้องใช้ & นำหน้าตัวแปรสตริง เช่น emp.name
  - แยกให้ออกว่าสตริงกับช่องข้อมูลช่องหนึ่งในอาเรย์เป็นของคนละอย่างกัน

การพิมพ์ข้อมูลภายในสตรัค

- โดยทั่วไปเราไม่สามารถพิมพ์ข้อมูลทั้งหมดในสตรัคออกมารวดเดียวได้
  - เราต้องพิมพ์ออกมาทีละตัวคล้ายกับตอนทำ scanf
  - ต้องระบุชนิดข้อมูลให้ตรงกันตามระเบียบ

```
printf("%s %s\n", emp.name, emp.surname);
printf("%s\n", emp.address);
printf("%.2f", emp.salary);
```

ตัวอย่างโปรแกรม

```
struct employee {
    char name[16];
    char surname[31];
    char address[151];
    float salary;
} typedef EMPLOYEE;

void main() {
  EMPLOYEE emp;
  scanf("%s", emp.name);
  scanf("%s", emp.surname);
  scanf("%s", emp.address);
  scanf("%f", &emp.salary);

  printf("%s %s\n", emp.name, emp.surname);
  printf("%s\n", emp.address);
  printf("%.2f", emp.salary);
}
```

ตัวอย่างเพิ่มเติม

- ตัวอย่างนี้แสดงสตรัคที่มีข้อมูลที่หลากหลายมากขึ้น
- เป็นสตรัคที่เก็บข้อมูลนักศึกษา ประกอบด้วย
  - รหัสประจำตัวนักศึกษา
  - ชื่อ-นามสกุลรวมอยู่ด้วยกัน
  - ชั้นปี
  - ระดับการเรียนเฉลี่ย

```
struct StudentRecord {
    int ID;
    char name[256];
    int year;
    float GPA;
} typedef STUDENT_RECORD;
```

ตัวอย่างเพิ่มเติม (ต่อ)

```
void main() {
  STUDENT_RECORD student;
  printf("Enter student ID: ");
  scanf("%d", &student.ID);
  printf("Enter student name: ");
  scanf("%s", &student.name[0]);
  printf("Enter student year: ");
  scanf("%d", &student.year);
  printf("Enter student GPA: ");
  scanf("%f", &student.GPA);

  printf("\n\nStudent Record:\n");
  printf("ID: %d\n", student.ID);
  printf("Name: %s\n", student.name);
  printf("Year: %d\n", student.year);
  printf("GPA: %.2f\n", student.GPA);
}
```

หัวข้อเนื้อหา

- สตรัค (struct) และ ประโยชน์ของสตรัค
- ก้าวแรกกับการประกาศสตรัค
- การประกาศสตรัคด้วย typedef
- การเข้าถึงข้อมูลในสตรัค
- การรับและแสดงผลข้อมูลในสตรัค
- **สตรัคในสตรัค**
- อาเรย์ของสตรัค และ อาเรย์ในสตรัค

สตรัคในสตรัค

- สตรัคแท้จริงเป็นชนิดข้อมูลที่เราสร้างขึ้นมาเอง
- เราสามารถนำชนิดข้อมูลใดก็ได้ไปเป็นสมาชิกในสตรัค รวมทั้งสตรัคชนิดอื่นด้วย
- ยกตัวอย่างเช่นหากเรามีสตรัคข้อมูลนักศึกษา
  - เราสามารถสร้างสตรัคของ*กลุ่มนักศึกษา* ซึ่งประกอบด้วยข้อมูลนักศึกษาในกลุ่ม และข้อมูลอื่น ๆ ได้ เช่นจำนวนกิจกรรมที่นักศึกษาในกลุ่มแต่ละคนเข้าร่วม
  - สมมติว่ากลุ่มหนึ่งมีนักศึกษา 4 คน เราจะได้สตรัคกลุ่มนักศึกษาเป็น

```
struct group {
    STUDENT_RECORD s1, s2, s3, s4;
    int act1, act2, act3, act4;
} typedef GROUP;
```

ตัวอย่าง

กำหนดกลุ่มนักศึกษา GROUP g มาให้ จงเขียนโปรแกรมที่รับข้อมูลนักศึกษา
มาเก็บไว้ในกลุ่ม g

วิธีทำ

เราเก็บข้อมูลเข้าในสตรัคพร้อมกันทุกตัวด้วย scanf ไม่ได้
จึงต้องเก็บข้อมูลของนักศึกษาทีละคน ซึ่งก็ต้องเก็บ ชื่อ รหัส ... ทีละค่า

วิธีเก็บข้อมูลมีอยู่หลายแบบ ขอยกตัวอย่างที่เข้าใจได้ง่าย

1. แบบไม่มีอาเรย์ข้อมูลนักศึกษาขึ้นมาก่อน
   และต้องการเก็บข้อมูลเข้าไปในกลุ่มโดยตรง
2. แบบสร้างอาเรย์ข้อมูลนักศึกษาขึ้นมาก่อน จากนั้นเก็บข้อมูลเข้าอาเรย์
   แล้วจึงคัดลอกข้อมูลนักศึกษาจากอาเรย์ไปเก็บไว้ในกลุ่ม

วิธีแรก : เก็บข้อมูลเข้ากลุ่มโดยตรง

```
void main() {
  GROUP g;

  scanf("%d", &g.s1.ID);
  scanf("%s %s", g.s1.name, g.s1.surname);
  scanf("%d", &g.s1.year);
  scanf("%f", &g.s1.GPA);
  scanf("%d", &g.act1);

  scanf("%d", &g.s2.ID);
  scanf("%s %s", g.s2.name, g.s2.surname);
  scanf("%d", &g.s2.year);
  scanf("%f", &g.s2.GPA);
  scanf("%d", &g.act2);}

  .....
```

สังเกตการใช้จุดซ้อนกันสองชั้น (สตรัคที่อยู่ในสตรัค)

act1 ไม่ใช่สตรัค จึงไม่ซ้อนจุด

ทำลักษณะเดิมกับ s3, s4, act1 และ act2

สังเกตโค้ดที่ผ่านมา

- เราต้องเขียนโค้ดแบบเดิมซ้ำกันหลายรอบ จาก s1, s2 ไปต่อแบบเดิมที่ s3 และ s4 ด้วย ทำอย่างไรจึงจะไม่ต้องเขียนโค้ดที่ซ้ำแบบนี้ได้ ?
- วิธีหนึ่งที่เป็นไปได้ก็คือให้เก็บข้อมูลไว้ในอาเรย์ก่อน
  - จัดการอาเรย์ด้วยลูป → ใช้โค้ดกับนักเรียนหลาย ๆ คนด้วยลูปเดียว
  - เปลี่ยนดัชนีของอาเรย์เท่ากับเปลี่ยนไปเก็บข้อมูลนักศึกษาคนอื่น
  - เนื่องจากจำนวนกิจกรรมที่นักศึกษาแต่ละคนเข้าร่วมไม่อยู่สตรัค → ต้องสร้างอาเรย์ของข้อมูลนี้แยกออกมาเพิ่มเติม
- เพื่อที่จะใช้เทคนิคนี้ได้ เราต้องทำความเข้าใจเรื่องอาเรย์ของสตรัคเพิ่มเติม

อาเรย์ของสตรัค

- สตรัคเป็นชนิดข้อมูลที่เราสร้างเอง
- อาเรย์เป็นแถวลำดับที่เก็บข้อมูลชนิดเดียวกันไว้ด้วยกันได้
- เราสามารถที่จะเก็บข้อมูลนักศึกษาซึ่งเป็นสตรัคไว้ในอาเรย์ได้
  - ในสตรัคมีข้อมูลหลากหลายชนิดก็จริง
  - แต่เมื่อมองที่ชนิดข้อมูลสตรัคแบบองค์รวม ถือเป็นชนิดข้อมูลอันหนึ่งอันเดียว
  - เมื่อถือเป็นข้อมูลชนิดเดียวก็ใช้กับอาเรย์ได้ เช่น

```
STUDENT_RECORD S[200];
```

เก็บข้อมูลหลักนักศึกษา

```
int A[200];
```

เก็บจำนวนกิจกรรมของนักศึกษา

- สังเกตว่าการประกาศอาเรย์ของสตรัคก็ดูคล้ายกับชนิดข้อมูลทั่วไป

ตัวอย่างเดิม, วิธีที่สอง : เก็บข้อมูลเข้าไปในอาเรย์ก่อน

```
STUDENT_RECORD S[4];
int A[4];

int i;
for(i = 0; i < 4; ++i) {
  scanf("%d", &S[i].ID);
  scanf("%s %s", S[i].name, S[i].surname);
  scanf("%d", &S[i].year);
  scanf("%f", &S[i].GPA);
  scanf("%d", &A[i]);
}

GROUP g;
g.s1 = S[0];     g.act1 = A[0];
g.s2 = S[1];     g.act2 = A[1];
g.s3 = S[2];     g.act3 = A[2];
g.s4 = S[3];     g.act4 = A[3];
```

เอาข้อมูลไปพักไว้ในอาเรย์ก่อน

copy ข้อมูลไปเก็บไว้ในกลุ่มนักศึกษา

ข้อสังเกตจากตัวอย่างที่แล้ว

- เมื่อเราใช้ลูปแล้วเปลี่ยนตัวเลขในวงเล็บเหลี่ยมไปเรื่อย ๆ จะมีนักศึกษากี่คนก็ใช้แค่ลูปเดียว เหมาะกับกรณีที่มีนักศึกษามาก
- เวลาที่เราสั่ง copy ข้อมูลจากช่องหนึ่งของอาเรย์ไปเก็บไว้ในกลุ่มนักศึกษา เราไม่ต้องคอย copy ข้อมูลย่อยทีละตัว
  - ไม่ต้องคอยเขียนว่า g.s1.ID = S[0].ID; g.s1.name = S[0].name;
  - เราสั่งทีเดียวได้เลยว่า g.s1 = S[0] ข้อมูลทั้งก้อนจะถูกคัดลอกมาหมด
  - เป็นการดำเนินการที่สะดวกมาก
- อย่างไรก็ตาม เรายังต้องเขียนบางอย่างซ้ำหลายรอบ เช่น

  g.s1 = S[0]; g.act1 = A[0];

  g.s2 = S[1]; g.act2 = A[1]; ... **แต่เราไม่อยากทำซ้ำแบบนี้**

อาเรย์ในสตรัค

- ในตัวอย่างกลุ่มนักศึกษาอันที่แล้วนั้น เรายังต้องเขียนบางอย่างซ้ำซาก
  - เพราะชื่อตัวแปรที่เก็บนักศึกษามันต่างกัน เราจึงต้องคอยแก้ทีละตัว
  - ถ้ามีนักศึกษาในกลุ่มแค่สี่คน คงไม่เป็นไรมาก แต่ถ้ามีเป็นสิบล่ะ
  - ในเมื่อนักศึกษาแต่ละคนมีชนิดข้อมูลเหมือนกัน เราใช้อาเรย์มาเก็บรวมได้

ตัวอย่าง

กำหนดให้แต่ละกลุ่มมีนักศึกษาอยู่สิบคน จงเปลี่ยนสตรัคกลุ่มนักศึกษาเดิมให้เก็บข้อมูลนักศึกษารวมถึงจำนวนกิจกรรมโดยใช้อาเรย์

```
struct group {
    STUDENT_RECORD Member[10];
    int AMember[10];
} typedef GROUP;
```

ตัวอย่างอาเรย์ในสตรัค

กำหนดให้นักศึกษาในชั้นเรียนมี 200 คน อาจารย์ต้องการแบ่งกลุ่มนักศึกษาออกเป็นกลุ่มละ 10 เรียงตามข้อมูลที่ใส่เข้ามา (10 คนแรกอยู่กลุ่ม 1 อีก 10 คนต่อมาอยู่กลุ่ม 2 และเป็นเช่นนี้ไปจนครบ 20 กลุ่ม)

วิธีทำ

1. เราควรใช้อาเรย์มาเก็บข้อมูลกลุ่ม เพราะมีซ้ำถึง 20 กลุ่ม
2. ใช้สตรัคที่แสดงเป็นตัวอย่างให้ดูก่อนหน้าได้
3. เราอาจจะเก็บข้อมูลนักศึกษาลงในอาเรย์ของนักศึกษาก่อนหรือไม่ก็ได้ ในตัวอย่างนี้จะใช้วิธีเก็บเข้าในอาเรย์นักศึกษาก่อน เพื่อให้คล้ายกับวิธีการเดิมที่เคยทำมาก่อนหน้า

วิธีแบบใส่ข้อมูลไปพักไว้ในอาเรย์ยาว ๆ ก่อน

```
STUDENT_RECORD S[200];
int A[200];
GROUP g[20];
int i, j;

for(i = 0; i < 200; ++i) {
  scanf("%d", &S[i].ID);
  scanf("%s %s", S[i].name, S[i].surname);
  scanf("%d", &S[i].year);
  scanf("%f", &S[i].GPA);
  scanf("%d", &A[i]);
}

for(i = 0; i < 20; ++i) {
  for(j = 0; j < 10; ++j) {
    g[i].Member[j] = S[j + 10*i];
    g[i].AMember[j] = A[j + 10*i];
  }
}
```

ตรงนี้เหมือนเดิม

แบบนี้จะยากตรงที่เราต้องคำนวณว่าจะเอาอาเรย์ช่องไหนมาใช้

ถ้าไม่อยากคำนวณว่าจะเอาอาเรย์ช่องไหนมาใช้ล่ะ?

ข้อนี้มีทางแก้ที่ใช้ได้ก็คือ สร้างตัวนับมาเพิ่มแล้วให้นับไปเรื่อย ๆ

(ใช้ได้กับข้อนี้ แต่จะใช้กับข้ออื่นได้หรือไม่ เราต้องดูบริบทของปัญหาก่อน)

```
……
int count = 0;
for(i = 0; i < 20; ++i) {
  for(j = 0; j < 10; ++j) {
    g[i].Member[j] = S[count];
    g[i].AMember[j] = A[count];
    ++count;
  }
}
```

เพราะการนับเพิ่มตรงนี้จะทำให้เกิดการเลื่อนไปรับดูข้อมูลนักศึกษาถัดไปเรื่อย ๆ อย่างถูกต้อง

วิธีแบบใส่ข้อมูลเข้าไปในกลุ่มตรง ๆ

อาจจะถือได้ว่าเป็นวิธีที่ดีที่สุดในกรณี เพราะนอกจากโค้ดจะสั้นลง ยังเข้าใจง่าย และ ไม่ต้องมีอาเรย์มาเก็บข้อมูลซ้ำซ้อนด้วย

```
GROUP g[20];
int i, j;

for(i = 0; i < 20; ++i) {
  for(j = 0; j < 10; ++j) {
    scanf("%d", &g[i].Member[j].ID);
    scanf("%s %s", g[i].Member[j].name,
                   g[i].Member[j].surname);
    scanf("%d", &g[i].Member[j].year);
    scanf("%f", &g[i].Member[j].GPA);
    scanf("%d", &g[i].AMember[j]);
  }
}
```

ชื่อซ้ำสำคัญฉไน

ชื่อตัวแปรที่เป็นสมาชิกอยู่ในสตรัค ซ้ำกับชื่อตัวแปรภายนอกสตรัคได้หรือไม่ ?

- ซ้ำได้ เพราะชื่อมีขอบเขต (scope) ที่ไม่ซ้ำซ้อนกัน
- แยกแยะออกได้ ของที่อยู่ในสตรัคอ้างผ่านจุด ของอยู่นอกสตรัคอ้างตรง ๆ
  เช่น มี int ID; และ STUDENT_RECORD a;
  ถ้าอ้างถึง ID โดด ๆ มันก็หมายถึง int ID; ที่อยู่นอกสตรัค
  แต่ถ้าใช้ a.ID; มันก็จะหมายถึง ID ที่อยู่ในสตรัค
- ตัวแปรใครตัวแปรมัน ต่อให้มีตัวแปรสตรัค 2 ตัวก็ไม่เป็นปัญหา
  เช่น มี STUDENT_RECORD a, b; เราก็แยกด้วย a.ID และ b.ID
- ต่อให้ชนิดข้อมูลในสตรัคเป็นอาเรย์ เป็นข้อความ ตัวชี้หรืออะไรก็ซ้ำได้
- การใช้จุด การใช้ชื่อตัวแปร แยกความแตกต่างพวกนี้ได้เสมอ

ตัวอย่างการตั้งชื่อในสตรัคซ้ำกับชื่อภายนอก (1)

- สมมติว่าเราเปลี่ยนชื่ออาเรย์ในสตรัค GROUP ไปเป็น ...

```
struct group {
     STUDENT_RECORD S[10];
     int A[10];
} typedef GROUP;
```

- สมมติว่าเราย้อนกลับไปใช้อาเรย์พักข้อมูล และตั้งชื่ออาเรย์เป็น S และ A

```
int main() {
   STUDENT_RECORD S[200];
   int A[200];
   GROUP g[20];
   int i, j;
   ……
}
```

ตัวอย่างการตั้งชื่อในสตรัคซ้ำกับชื่อภายนอก (2)

- โค้ดของเรายังทำงานได้ตามปรกติ ขอแค่เปลี่ยนชื่อให้สอดคล้องกันก็พอ

```
for(i = 0; i < 200; ++i) {
  scanf("%d", &S[i].ID);
  scanf("%s %s", S[i].name, S[i].surname);
  scanf("%d", &S[i].year);
  scanf("%f", &S[i].GPA);
  scanf("%d", &A[i]);
}
```

ตอนต้นยังเหมือนเดิมทุกอย่าง
ซึ่ง S กับ A เป็นตัวแปรภายนอกสตรัค

S กับ A ตรงด้านซ้ายขวาเป็นคนละตัวกันเราแยกความแตกต่างได้ผ่าน . ซึ่งเป็นตัวกำหนดบริบทว่าตัวแปรเป็นของใคร

```
for(i = 0; i < 20; ++i) {
  for(j = 0; j < 10; ++j) {
    g[i].S[j] = S[j + 20*i];
    g[i].A[j] = A[j + 20*i];
  }
}
```

การกำหนดค่าเริ่มต้นให้กับอาเรย์และสตรัคในโค้ด

- เราสามารถกำหนดค่าเริ่มต้นให้อาเรย์ได้ ผ่านการใช้เครื่องหมาย { } เช่น
  - int A[5] = {9, 7, 10, 0, 2};
  - float F[4] = {2.35, 1.78, -1.2, 0.5};
- วิธีข้างบนนี้จะทำให้ตัวเลขไปปรากฏในอาเรย์เรียงตามลำดับจากช่องที่ 0 ไปช่องที่ 1, 2, ...
- เราสามารถกำหนดค่าเริ่มต้นให้กับสตรัคได้เหมือนกันผ่านเครื่องหมาย { }

  เราใส่ข้อมูลเข้าไปทีละตัวตามลำดับการปรากฏตอนประกาศสตรัค

เช่น STUDENT_RECORD s = {07540123, "Pinyo", "Taeprasartsit",

1, 3.21};

กำหนดค่าเริ่มต้น อาเรย์ของสตรัค

อาจจะถือได้ว่าเป็นขั้นสูงสุดของการดำเนินการแบบนี้ก็ได้ :P

1. เราใช้ { } กับอาเรย์ด้านนอกตามปรกติ
2. ส่วนข้อมูลของสตรัคแต่ละตัวจะมี { } ของมันเอง

ตัวอย่าง

```
STUDENT_RECORD SR[3] =
    { {07540123, "Pinyo", "Tae", 1, 3.21},
      {07540456, "Opas", "Wong", 2, 3.45},
      {07540789, "Ann", "Center", 3, 3.99} };
```

แบบฝึกหัด

แบบฝึกหัดท้ายบท 12 ข้อ 1 – 4 หน้า 284 – 285

เพื่อที่จะไม่ต้องพิมพ์ข้อมูลเข้าซ้ำซากในแบบฝึกหัดนี้ เราสามารถใส่มันลงไปในโค้ดได้เลย ดูตัวอย่างได้ในหน้าที่แล้วหรือในเฉลย