SCIENCE
SILPAKORN UNIVERSITY

ค่ายโอลิมปิก สอวน. 2

การเขียนโปรแกรมสำหรับการแข่งขัน
Competitive Programming

อ.ดร.ภิญโญ แท้ประสาทสิทธิ์

ภาควิชาคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศิลปากร

pinyotae at gmail dot com

หัวข้อเนื้อหา

SCIENCE
SILPAKORN UNIVERSITY

- การเรียงข้อมูลในภาษา C++ (ทบทวนอีกที)
- การใช้ Priority Queue
- การจำผลลัพธ์ที่ผ่านมาไว้ใช้ในรอบถัดไป (หลายส่วนคล้าย Dynamic Programming)
- ~~การใช้ Tree Map~~
- Range Sum
- Range Max
- Recursive function is your friend
  - Counting Method
  - Depth First Search

การเรียงข้อมูลในภาษา C++

SCIENCE
SILPAKORN UNIVERSITY

- เราอาจจะเคยเรียนทฤษฎีการเรียงข้อมูลมาพอสมควร
- เราอาจจะรู้ว่า insertion sort, bubble sort, selection sort คืออะไร
- เรารู้ว่าของพวกนี้มันเขียนเองได้ง่าย ๆ แต่มันช้ามาก
  → ถ้าใช้กับข้อมูลใหญ่ ๆ ถึง 10,000 ตัวมักจะคำนวณไม่ค่อยทัน
- ถ้าเราจะใช้ Quick sort เราก็ไม่ควรเขียนเอง เพราะมันมีให้ใช้
  - ทั้งในบริบทของภาษา C คือฟังก์ชัน qsort
  - ในภาษา C++ มี quick sort ใน container เช่น vector และ (tree) map
- ถ้าหากเราไม่รู้จักใช้ advanced data structure โอกาสที่จะประสบความสำเร็จในโลกของการเขียนโปรแกรมจะมีน้อยมาก
  - เพราะทำงานไม่ทันคนอื่น ๆ ที่รู้จักใช้เครื่องมือด้านการเขียนโปรแกรม

การเรียงข้อมูลใน C++ STL container

- หลักการก็คือว่าเราจะต้องกับ C++ Library ว่าข้อมูล 2 ตัวที่ต้องการเปรียบเทียบนั้น คำว่าน้อยกว่าหรือมากกว่านิยามจากอะไร
  - เช่น ถ้าเรามี class Student {public: int ID; double GPA;};
  - ต่อมาเรามี student1 และ student2
  - C++ Library ต้องการให้เราบอกว่า คำว่าน้อยกว่าดูจากอะไร
- ทำไมจำเป็นต้องบอก
  - อย่างที่เห็น มันเป็นไปได้ว่าเราอาจจะเรียงข้อมูลนักเรียนจาก ID หรือ GPA
  - C++ ไม่สามารถรู้ได้ว่าเรามีเจตนาอย่างไร เราจึงต้องระบุเอง
- แต่ถ้าหากเราจะเรียงข้อมูลพื้นฐานเช่น int, double, float
  - ข้อมูลพื้นฐานเหล่านี้มีวิธีเรียงจากน้อยมากอยู่แล้ว ไม่ต้องบอก C++ ก็ได้
  - โดยมากการจัดเรียงจะทำจากน้อยไปมาก

งั้นลอง sort กับพวก int ก่อนละกัน

- สมมติว่าเริ่มมาเรามีข้อมูลในเวคเตอร์ดังนี้

```
int data[] = {7, 2, 4, 5, 3, 1, 6};
vector<int> vec(data, data + 7);
```

- เราต้องการให้มันเรียงจากน้อยไปมาก ➔ 1, 2, 3, 4, 5, 6, 7
- เราเพียงบอกบริเวณที่จะเรียงข้อมูลใน container ก็พอ
  - ซึ่งในที่นี้ก็คือเรียงข้อมูลจากจุดเริ่มไปจนหมด
  - นั่นคือเรียงจาก vec.begin() ไปจนถึง (แต่ไม่รวม) vec.end()

```
std::sort(vec.begin(), vec.end());
```

- ส่วนเฮดเดอร์ไฟล์ที่ต้องใช้ในที่นี้คือ vector และ algorithm

```
#include <vector>
#include <algorithm>
```

โค้ดแบบเต็มสำหรับการเรียง int

SCIENCE
SILPAKORN UNIVERSITY

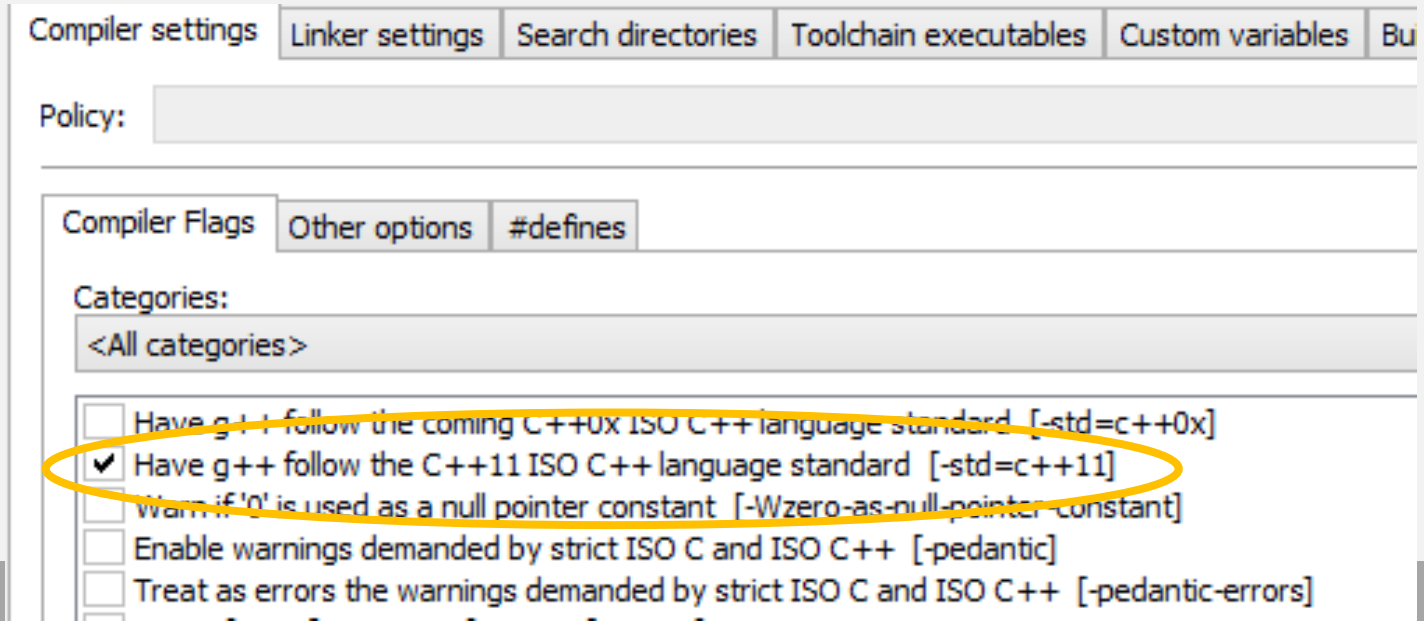
```
#include <vector>
#include <algorithm>

using namespace std;
int main() {
  int data[] = {7, 2, 4, 5, 3, 1, 6};
  vector<int> vec(data, data + 7);
  std::sort(vec.begin(), vec.end());

  for(auto val : vec)
    printf("%d ", val);
  return 0;
}
```

แบบนี้ทำได้เฉพาะใน C++11

C++11 คืออะไรอ่ะครับ

SCIENCE
SILPAKORN UNIVERSITY

- มันคือมาตรฐานภาษา C++ ที่ถือว่าใหม่
  - เมื่อเทียบกับตัว default ใน Code::Block ที่เป็น C++98
  - ในชุดคอมไพเลอร์อื่น ๆ เช่น Visual C++ อาจให้ C++11 เป็น default
- แล้วจะเรียกใช้มันยังไงดี?
  - ไปที่ Settings->Compiler..
  - เลือกออปชัน Have g++ follow the C++11 ... [-std=c++11]



คราวนี้มาเรียงออปเจ็คกันบ้าง

- เรียงแค่ตัวเลขตรง ๆ มักจะไม่ได้ทำประโยชน์อะไรมากนัก
  - ยกตัวอย่างเช่น ถ้าหากเรามี vector<int> vecID;
    และมี vector<double> vecGPA; แยกเป็นสองอัน
  - เวลาที่เราจะเรียงนักศึกษาตามรหัสประจำตัว เราจะพบว่าข้อมูลใน vecGPA ไม่ได้เปลี่ยนไปตามลำดับของ vecID แต่แยกกันอยู่เป็นอิสระจากกัน
- ถ้าเราอยากให้ข้อมูลสองตัวเรียงไปด้วยกัน
  → เราต้องจัดข้อมูลให้อยู่ในคลาสเดียวกันก่อน
- แต่จะทำแบบนี้ได้ก็ต้องบอกกับ C++ Library ว่าจะเรียงมากกว่าน้อยกว่าจะอะไร ทำให้งานงอกขึ้นเล็กน้อย แต่โดยรวมก็ถือว่าง่ายมาก

ทดลองเรียงข้อมูลนักเรียนจาก ID

SCIENCE
SILPAKORN UNIVERSITY

- สมมติว่าเรามีคลาสของนักเรียนตามนี้

```
class Student {
public:
    int ID;
    double GPA;
    Student(int ID, double GPA) {
        this->ID = ID;
        this->GPA = GPA;
    };
};
```

ถ้าลืมใส่ this ความหมายจะผิดไป

- และข้อมูลเริ่มต้นคือ

```
Student data[] = {Student(7, 3.5), Student(2, 3.6),
  Student(4, 3.2), Student(5, 4.0), Student(3, 3.8),
  Student(1, 3.9), Student(6, 2.8)};
vector<Student> vec(data, data + 7);
```

ใช้ std::sort แบบเดิมกับออปเจ็คไม่ได้

- ถ้าเราคิดจะใช้โค้ดแบบเดิมว่า

```
std::sort(vec.begin(), vec.end());
```

- เราอาจจะได้ข้อความบอกความผิดพลาดที่ยาวมากจนเราไม่รู้เรื่อง
  - ที่เป็นเช่นนั้นเพราะคอมไพเลอร์อาจจะพยายามสรุปให้ได้ว่าเราต้องการอะไร
  - แต่ในความพยายามดังกล่าวเส้นทางก็ยาวมาก ทำให้ได้ความเป็นไปได้ และสาเหตุที่เกี่ยวข้องกับความผิดพลาดเต็มไปหมด
  - ถ้าเราเห็นข้อความแสดงความผิดพลาดยาว ๆ มันมักเกี่ยวกับ template
  - เพราะ template เปิดโอกาสของความเป็นไปได้ที่หลากหลายและซับซ้อนมาก
- ทางแก้ปัญหานี้ก็คือว่า เราจะต้องระบุฟังก์ชันสำหรับเปรียบเทียบข้อมูลติดไปกับ std::sort ด้วย โดยใส่เป็นพารามิเตอร์ตัวที่สาม

นิยามฟังก์ชันสำหรับเรียงออปเจ็ค

- มีอยู่หลายวิธี แต่ในที่นี้ขอแนะนำวิธีที่จำง่าย ใช้ง่าย และยืดหยุ่น
- เราจะสร้างฟังก์ชันที่คืน bool ซึ่งรับออปเจ็คสองตัวที่จะเปรียบเทียบมาเป็นพารามิเตอร์
  - ฟังก์ชันนี้คืนค่า true เมื่อออปเจ็คตัวแรกมาก่อนตัวที่สอง
  - ออปเจ็คทั้งสองตัวต้องเป็นออปเจ็คค่าคงที่ **คือมี const นำหน้า**
- เช่น ถ้าเราต้องการเรียงนักเรียนจาก ID น้อยไปมาก เราจะเขียนฟังก์ชันว่า

```
bool sortByID(const Student& s1, const Student& s2) {
    return (s1.ID < s2.ID);
}
```

- เนื่องจากเราจะให้ s1 อยู่หน้า s2 เมื่อ ID ของ s1 มีค่าน้อยกว่า เราจึงคืนค่าจากการเปรียบเทียบ **`s1.ID < s2.ID`** ไปเป็นผลลัพธ์

เรื่องควรเข้าใจเกี่ยวกับฟังก์ชันเปรียบเทียบ

- **การใช้ const เป็นสิ่งบังคับ** ต้องมี
- การใช้ reference คือตัว & ไม่บังคับ แต่ควรใช้
  - เพราะโดยทั่วไป มันทำให้โปรแกรมเร็วขึ้น
    เนื่องจากไม่เสียพลังในการ copy ข้อมูลในออปเจ็คมาแบบยกก้อน
  - ถ้ามีข้อมูลในออปเจ็คมาก ควรใช้เป็นอย่างยิ่ง
  - แต่ถึงมีน้อยจะใช้ก็ไม่เสียหายอะไร ดังนั้นใช้ & ไปเหอะ
- ค่าความจริงที่คืนไปนั้น มีใจความว่า ถ้าออปเจ็คแรกมาก่อน ให้คืน true
- ตัวฟังก์ชันเปรียบเทียบจะยาวหลายร้อยบรรทัดก็ได้ จะเรียกฟังก์ชันอื่นต่อ ๆ กันไปก็ได้ตามปรกติ แต่โดยมากเราเห็นคนเขียนเป็นตัวอย่างให้มีบรรทัดเดียว

ทดลองเรียกใช้ฟังก์ชัน sortByID

SCIENCE
SILPAKORN UNIVERSITY

- ตรงไปตรงมามาก

```
std::sort(vec.begin(), vec.end(), sortByID);

for(auto& student : vec)
  printf("%d %.1f\n", student.ID, student.GPA);
```

- แบบฝึกหัด
  - จงทดลองเปลี่ยนเป็นเรียงตาม GPA จากมากไปน้อย (ไม่ต้องส่ง)
  - ทำข้อ Runner ในเกรดเดอร์

Priority Queue in C++

- เป็นโครงสร้างข้อมูลที่วิธีใช้ดูแปลก ๆ ในตอนประกาศ
  - เพราะมันเป็น template ที่ต้องการพารามิเตอร์ถึงสามตัว
  - ทำให้สับสนและหลายคนไม่อยากใช้ โดยเฉพาะตัวฟังก์ชันเปรียบเทียบค่า
  - แต่หลังจากประกาศไปแล้ว วิธีใช้งานจะตรงตามแนวคิดที่เรียนมาทุกอย่าง
  - ทำให้รู้สึกว่ามันเข้าใจง่ายและน่าใช้ขึ้นมาพอสมควร
- วิธีที่ง่ายที่สุดในการทำความเข้าใจคือดูตัวอย่างการใช้งาน
  - สมมติว่าเราจะใช้ priority queue เพื่อทำ Prim's algorithm

```
priority_queue<Edge, vector<Edge>, EdgeComparator> pq;
```