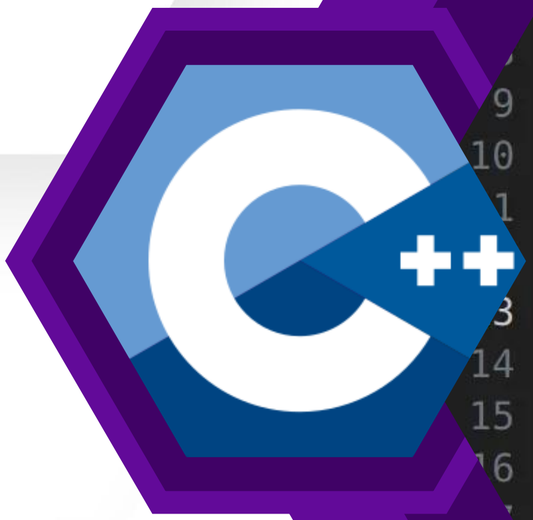
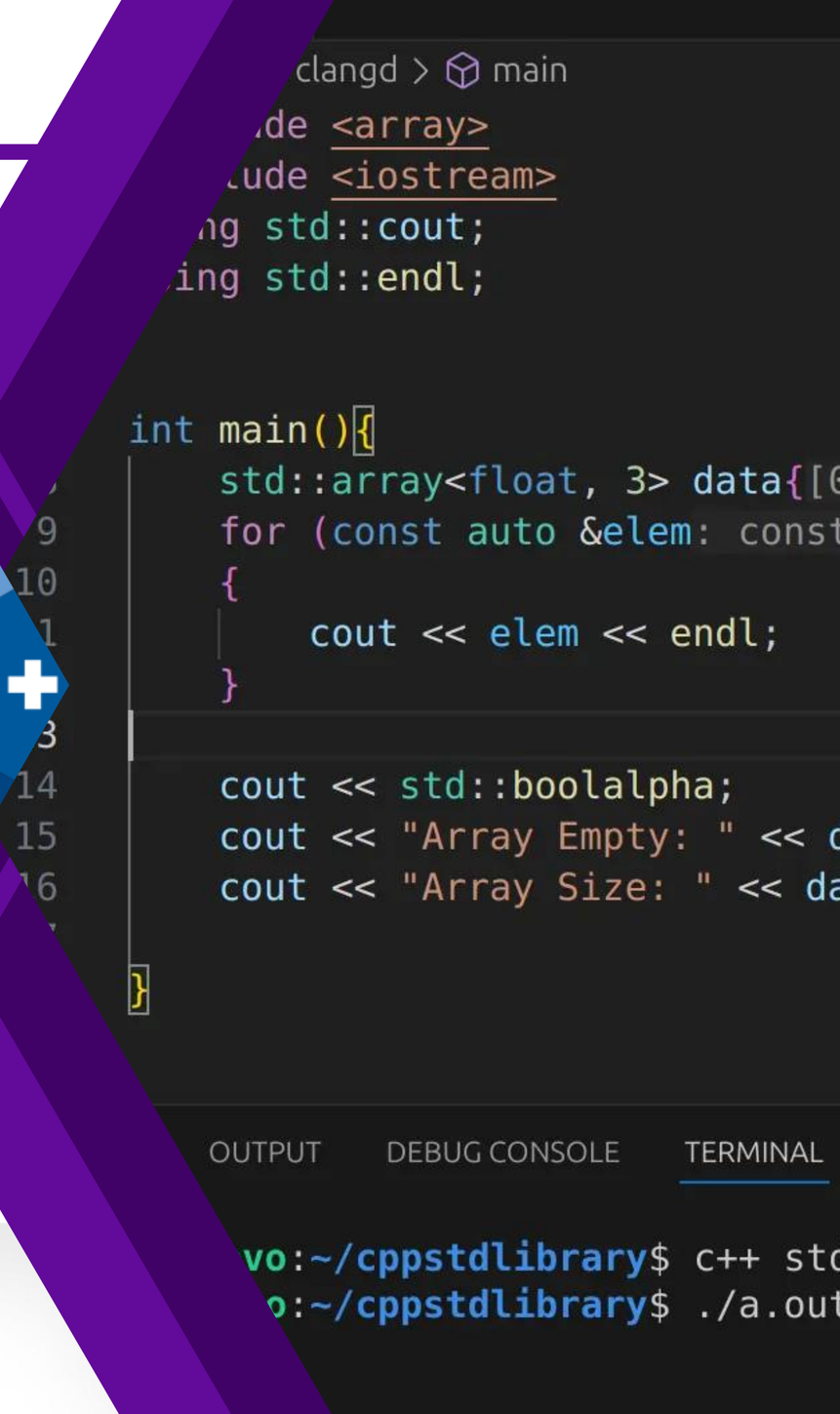
STL

Standard Template Library





STL

STL คืออะไร

ย่อมาจาก **Standard Template Library** เป็นชุดของ **Template Class** และ **Function** ที่ถูกพัฒนาขึ้นมาเพื่อให้การจัดการข้อมูลและอัลกอริธึมเป็นเรื่องง่ายและมีประสิทธิภาพ

STL

ส่วนประกอบหลัก: ประกอบด้วย 4 ส่วนหลัก

Containers (ภาชนะ): โครงสร้างข้อมูลสำหรับเก็บข้อมูล (เช่น vector, list, map, set)

Algorithms (อัลกอริธึม): ฟังก์ชันสำหรับประมวลผลข้อมูลใน Containers (เช่น sort, find, copy)

Iterators (ตัววนซ้ำ): ทำหน้าที่เหมือนพอยน์เตอร์ทั่วไป ใช้ชี้ไปยังตำแหน่งข้อมูลใน Containers เพื่อเชื่อมต่อระหว่าง Containers และ Algorithms

Functors (Function Objects): ออบเจ็กต์ที่ทำงานเหมือนฟังก์ชัน

Vector

std::vector

เป็น **Dynamic Array** (อาร์เรย์แบบไดนามิก) ที่สามารถเพิ่มหรือลดขนาดได้เองอัตโนมัติขณะรันโปรแกรม และเก็บข้อมูลในหน่วยความจำที่ต่อเนื่องกัน (Contiguous Memory)

Vector

การประกาศและเริ่มต้น (Declaration & Initialization)

รูปแบบการประกาศ	ตัวอย่าง	คำอธิบาย
แบบว่างเปล่า	std::vector<int> vec1;	สร้าง vector ชนิด int ที่ไม่มีสมาชิกเริ่มต้น
กำหนดขนาด	std::vector<double> vec2(10);	สร้าง vector ชนิด double ขนาด 10 ตัว ที่มีค่าเริ่มต้นเป็น 0.0
กำหนดขนาดและค่าเริ่มต้น	std::vector<std::string> vec3(5, "Hello");	สร้าง vector ชนิด string ขนาด 5 ตัว โดยทุกตัวมีค่าเป็น "Hello"
ใช้ List Initializer (C++11)	std::vector<int> vec4 = {10, 20, 30};	สร้างและกำหนดค่าเริ่มต้นให้สมาชิกโดยตรง

Vector

ฟังก์ชันหลักในการใช้งาน (Core Member Functions)

ฟังก์ชัน	การใช้งาน	ตัวอย่าง	คำอธิบาย
push_back()	การเพิ่มข้อมูล	vec1.push_back(40);	เพิ่มข้อมูลไปท้ายสุดของ vector (เร็ว)
pop_back()	การลบข้อมูล	vec1.pop_back();	ลบข้อมูลตัวสุดท้ายออก (เร็ว)
size()	ขนาดปัจจุบัน	vec1.size()	คืนค่าจำนวนสมาชิกปัจจุบันใน vector
capacity()	ความจุหน่วยความจำ	vec1.capacity()	คืนค่าจำนวนสมาชิกที่ vector สามารถเก็บได้ก่อนที่จะต้องทำการจัดสรรหน่วยความจำใหม่

Vector

ฟังก์ชันหลักในการใช้งาน (Core Member Functions)

ฟังก์ชัน	การใช้งาน	ตัวอย่าง	คำอธิบาย
capacity()	ความจุหน่วยความจำ	vec1.capacity()	คืนค่าจำนวนสมาชิกที่ vector สามารถเก็บได้ก่อนที่จะต้องทำการจัดสรรหน่วยความจำใหม่
empty()	ตรวจสอบว่าว่างหรือไม่	vec1.empty()	คืนค่า true ถ้า vector ไม่มีสมาชิกเลย
at()	การเข้าถึงข้อมูลตามดัชนี	int x = vec1.at(i);	เข้าถึงสมาชิกที่ดัชนี i (จะตรวจสอบขอบเขต หากเกินจะเกิด Exception)
[]	การเข้าถึงข้อมูลตามดัชนี	int y = vec1[i];	เข้าถึงสมาชิกที่ดัชนี i (ไม่ตรวจสอบขอบเขต, เร็วกว่า at())
front()	สมาชิกตัวแรก	vec1.front()	คืนค่าอ้างอิงถึงสมาชิกตัวแรก

Vector

ฟังก์ชันหลักในการใช้งาน (Core Member Functions)

ฟังก์ชัน	การใช้งาน	ตัวอย่าง	คำอธิบาย
back()	สมาชิกตัวสุดท้าย	vec1.back()	คืนค่าอ้างอิงถึงสมาชิกตัวสุดท้าย
insert()	แทรกข้อมูล	vec1.insert(it, 50);	แทรกค่า 50 ณ ตำแหน่งที่ iterator it ชี้อยู่ (ซ้าย)
erase()	ลบข้อมูล	vec1.erase(it);	ลบสมาชิก ณ ตำแหน่งที่ iterator it ชี้อยู่ (ซ้าย)

Vector

การวนซ้ำ (Iteration)

Classic For Loop: ใช้วิธีแบบอาร์เรย์ทั่วไป

C++

```
for (size_t i = 0; i < vec.size(); ++i) {
    std::cout << vec[i] << " ";
}
```

Range-based For Loop (ใช้งานง่ายและสะดวกกว่า)

C++

```
for (const auto& elem : vec) {
    std::cout << elem << " ";
}
```

Vector

การวนซ้ำ (Iteration)

Iterators: ใช้ begin() และ end()

C++

```
for (auto it = vec.begin(); it != vec.end(); ++it) {
    std::cout << *it << " ";
}
```

Vector

ตัวอย่างการใช้อัลกอริธึมร่วมกับ vector

#include <algorithm>:

ต้องการ include header นี้เพื่อใช้อัลกอริธึม STL

std::sort: การจัดเรียงข้อมูล

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

Vector

ตัวอย่างการใช้อัลกอริธึมร่วมกับ vector

std::find: การค้นหาข้อมูล

```
auto it = std::find(vec.begin(), vec.end(), 20);
    if (it != vec.end()) {    // พบข้อมูล}
```

Vector

ตัวอย่างการใช้อัลกอริธึมร่วมกับ vector

std::reverse: การกลับลำดับข้อมูล

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

ตัวอย่างที่ 1: การประกาศ, เพิ่ม, เข้าถึง, และลบข้อมูล

```
#include <iostream>
#include <vector>
#include <string>
int main() {
// 1. การประกาศ vector (ชนิด string)
std::vector<std::string> names;

// 2. การเพิ่มข้อมูล: push_back()
names.push_back("Alice");
names.push_back("Bob");
names.push_back("Charlie");
names.push_back("David");

std::cout << "--- 1. ข้อมูลใน Vector ปัจจุบัน ---" << std::endl;
// 3. การเข้าถึง/วนซ้ำข้อมูล (Range-based for loop - C++11)
for (const std::string& name : names) {
    std::cout << name << " "; // Output: Alice Bob Charlie David
 }   std::cout << "\nขนาดปัจจุบัน: " << names.size() << std::endl; // Output: 4
```

ตัวอย่างที่ 1: การประกาศ, เพิ่ม, เข้าถึง, และลบข้อมูล

```
// 4. การเข้าถึงข้อมูลตามดัชนี
std::cout << "สมาชิกตัวแรก: " << names[0] << std::endl; // Output: Alice
std::cout << "สมาชิกตัวสุดท้าย: " << names.back() << std::endl; // Output: David


// 5. การลบข้อมูลตัวสุดท้าย: pop_back()     names.pop_back(); // ลบ David ออกไป
std::cout << "\n--- 2. หลังใช้ pop_back() ---" << std::endl;
std::cout << "ขนาดใหม่: " << names.size() << std::endl; // Output: 3
std::cout << "สมาชิกที่เหลือ: ";
for (const std::string& name : names) {
    std::cout << name << " "; // Output: Alice Bob Charlie
}    std::cout << std::endl;
 return 0;
}
```

Vector

ตัวอย่างการใช้อัลกอริธึมร่วมกับ vector

การใช้งาน std::sort (อัลกอริธึม)

การใช้อัลกอริธึม **std::sort** ร่วมกับ std::vector

โดยใช้ **Iterators** (ตัววนซ้ำ) เป็นตัวกำหนดช่วง

ตัวอย่างที่ 2: การจัดเรียงข้อมูล (Sorting)

Vector

ตัวอย่างการใช้อัลกอริธึมร่วมกับ vector

การใช้งาน std::sort (อัลกอริธึม)

```
#include <iostream>
#include <vector>#include <algorithm> // ต้อง include header นี้สำหรับ Algorithms
// Function สำหรับแสดงผล Vector
void print_vector(const std::vector<int>& vec) {
        for (int num : vec) {
            std::cout << num << " ";
       }    std::cout << std::endl;
}
int main() {
       std::vector<int> numbers = {50, 10, 40, 20, 30};
       std::cout << "ข้อมูลเริ่มต้น: ";
        print_vector(numbers); // Output: 50 10 40 20 30
     // 1. การจัดเรียงจากน้อยไปมาก (Ascending Order)
    // std::sort จะรับ Iterator สองตัว: .begin() และ .end()
    std::sort(numbers.begin(), numbers.end());
```

Vector

ตัวอย่างการใช้อัลกอริธึมร่วมกับ vector

การใช้งาน std::sort (อัลกอริธึม)

```
std::cout << "จัดเรียง (น้อยไปมาก): ";
print_vector(numbers); // Output: 10 20 30 40 50


// 2. การจัดเรียงจากมากไปน้อย (Descending Order)
// ใช้ร่วมกับ 'std::greater<int>()'
        std::sort(numbers.begin(), numbers.end(), std::greater<int>());
         std::cout << "จัดเรียง (มากไปน้อย): ";
        print_vector(numbers); // Output: 50 40 30 20 10
return 0;
}
```

Vector

ตัวอย่างการใช้อัลกอริธึมร่วมกับ vector

การค้นหาข้อมูล (Searching)

```
#include <iostream>
#include <vector>
#include <algorithm>
   int main() {
    std::vector<int> data = {1, 5, 8, 3, 9, 2};
    Int target = 8;
    int non_existent = 100;
  // 1. ค้นหาค่า '8'
 // std::find จะคืนค่าเป็น Iterator
auto it_found = std::find(data.begin(), data.end(), target);
```

Vector

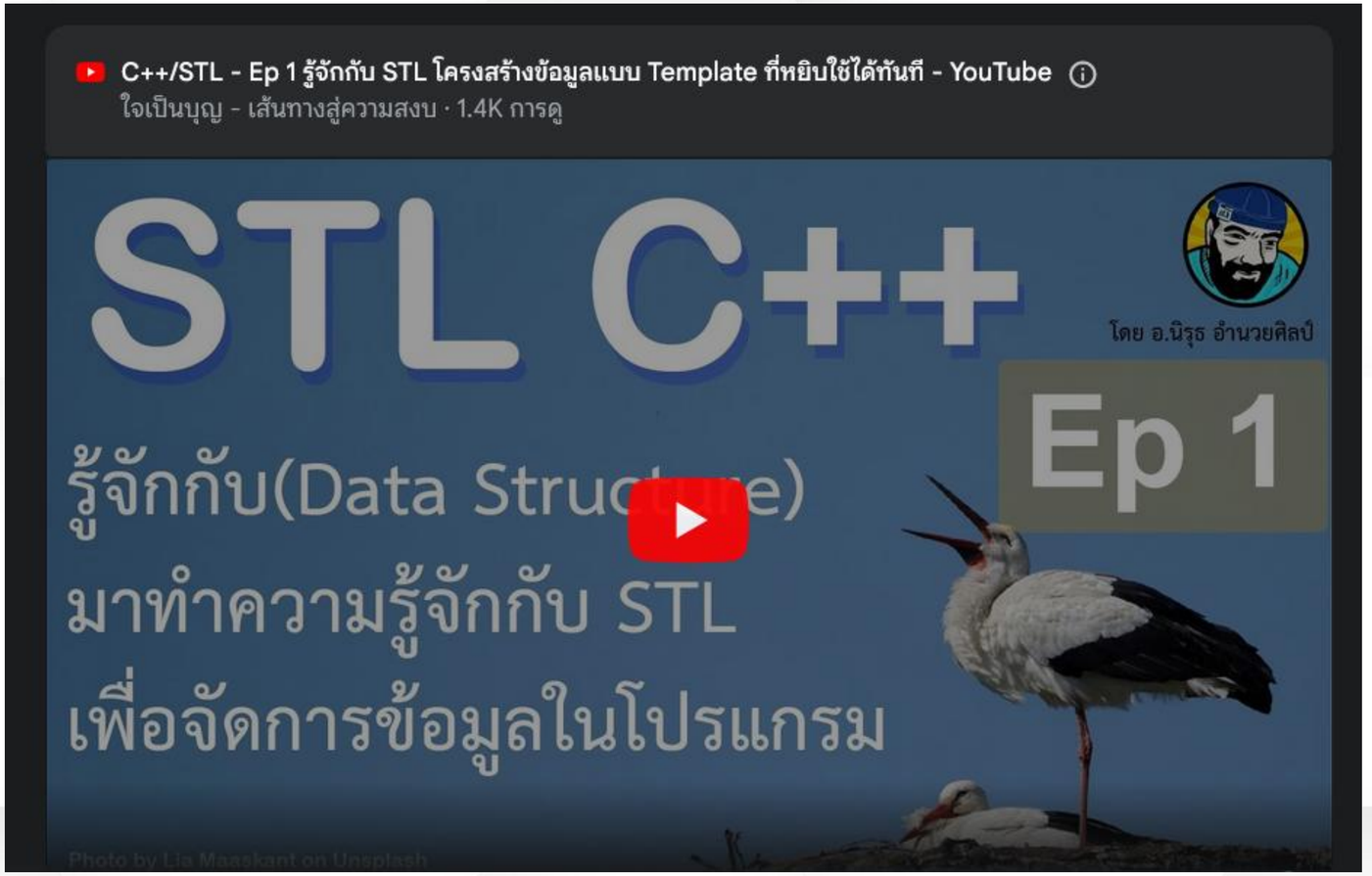
ตัวอย่างการใช้อัลกอริธึมร่วมกับ vector

การค้นหาข้อมูล (Searching)

```
// 2. ตรวจสอบผลลัพธ์
 // ถ้าพบ จะไม่เท่ากับ data.end()
if (it_found != data.end()) {
        std::cout << "พบค่า " << target << " ที่ดัชนี: "
                  << std::distance(data.begin(), it_found) << std::endl;
// std::distance ใช้คำนวณระยะห่างระหว่าง iterator (คือ index)
} else {
    std::cout << "ไม่พบค่า " << target << std::endl;
}
// 3. ค้นหาค่า '100'
auto it_not_found = std::find(data.begin(), data.end(), non_existent);
 if (it_not_found == data.end()) { // ถ้าไม่พบ จะเท่ากับ data.end()
          std::cout << "ไม่พบค่า " << non_existent << std::endl;
}         return 0;}
```

Vector

ตัวอย่างเพิ่มเติม



https://www.youtube.com/watch?v=IeAzsBgrxNk