การใช้งาน set และ map เบื้องต้น

สิรักข์ แก้วจำนงค์

set และ unordered_set

- Set เป็นโครงสร้างข้อมูลชนิดหนึ่งที่เก็บข้อมูลแบบไม่ซ้ำกัน โดย Set ใช้ Tree ในการเก็บข้อมูล ซึ่งทำให้การค้นหาและการเพิ่มข้อมูลมีประสิทธิภาพสูงกว่า Array และ Vector

```
#include <set>
using namespace std;

set<ชนิดข้อมูล>ชื่อตัวแปร;
```

- unordered_set เป็น container class ใน C++ ที่เก็บข้อมูลในลักษณะของเซต (set) ที่ไม่มีการจัดเรียง (unordered) โดยใช้เทคนิคของการเก็บข้อมูลแบบ Hash table ซึ่งสามารถใช้ในการค้นหาข้อมูลได้อย่างมีประสิทธิภาพ

```
unordered_set<int> my_set;
```

- ข้อมูลใน set และ unordered_set จะต้องไม่ซ้ำกัน

ตัวอย่างการใช้ set และผลลัพธ์

```cpp
using namespace std;
int main() {
    // เก็บข้อมูลชนิด int
    set<int> s1;
    s1.insert(30);
    s1.insert(20);
    s1.insert(50);
    s1.insert(40);
    // แสดงผลลัพธ์
    cout << "set of int elements: ";
    for(auto it = s1.begin(); it != s1.end(); it++) {
        cout << *it << " ";
    }
    cout << endl;
    // เก็บข้อมูลชนิด string
    set<string> s2;
    s2.insert("Somchai");
    s2.insert("Prayuth");
    s2.insert("Akradej");
    // แสดงผลลัพธ์
    cout << "set of string elements: ";
    for(auto it = s2.begin(); it != s2.end(); it++) {
        cout << *it << " ";
    }
    cout << endl;
    return 0;
}
```

```
set of int elements: 20 30 40 50
set of string elements: Akradej Prayuth Somchai
```

ฟังก์ชั่นของ set

- insert() ใช้สำหรับเพิ่มข้อมูลเข้าไปใน Set

```
set<int> mySet;
mySet.insert(10); // ใส่ข้อมูลเลข 10 เข้าไปใน set
```

- erase() ใช้สำหรับลบข้อมูลออกจาก Set

```
set<int> mySet;
mySet.erase(10); // ลบข้อมูลเลข 10 ออกจาก set
```

- clear() ใช้สำหรับลบข้อมูลทั้งหมดใน Set

```
set<int> mySet;
mySet.clear(); // ลบข้อมูลทั้งหมดใน set
```

ฟังก์ชั่นของ set

- find() ใช้สำหรับค้นหาข้อมูลใน Set และคืนค่า iterator ของข้อมูลนั้น (หากพบ) หรือคืนค่า iterator สุดท้ายของ Set (หากไม่พบ)

```
set<int> mySet;
set<int>::iterator it;
it = mySet.find(10); // ค้นหาข้อมูลเลข 10 ใน set
if (it != mySet.end()) { // ถ้าพบ
    cout << "Found " << *it << endl;
} else { // ถ้าไม่พบ
    cout << "Not found" << endl;
}
```

- begin() / end() ใช้สำหรับคืนค่า iterator ที่ชี้ไปที่จุดเริ่มต้น (begin) และ element สุดท้าย (end)

ฟังก์ชั่นของ set

- size() ใช้สำหรับคืนค่าจำนวนข้อมูลใน Set

```
set<int> mySet;
mySet.insert(10);
mySet.insert(20);
mySet.insert(30);
cout << "Size: " << mySet.size() << endl; // แสดงจำนวนข้อมูลใน set
```

- empty() ใช้สำหรับตรวจสอบว่า Set ว่างหรือไม่ โดยคืนค่า true หาก Set ว่างและคืนค่า false หาก Set มีข้อมูลอยู่

```
set<int> mySet;
if (mySet.empty()) { // ตรวจสอบว่า set ว่างหรือไม่
    cout << "Set is empty" << endl;
} else {
    cout << "Set is not empty" << endl;
}
```

การใช้ set เก็บ object

- Set สามารถเก็บ object ได้ โดย object จะถูกเก็บเป็นค่าตามลำดับของประเภทของข้อมูลที่เรากำหนดไว้ในการสร้าง Set โดยปกติแล้ว Set จะใช้การเปรียบเทียบนิยามขึ้นมาเพื่อเปรียบเทียบสมาชิกภายใน Set ว่าเท่ากันหรือไม่ ดังนั้นหากต้องการค้นหา object ใน Set ที่เก็บ object ต้องระบุเงื่อนไขในการเปรียบเทียบตามลำดับของประเภทของข้อมูลนั้น ๆ
- ตัวอย่างเช่น หากต้องการเก็บ object ของคลาส Person ใน Set แล้วต้องการค้นหา Person โดยใช้เงื่อนไขชื่อ (name) และอายุ (age) จะต้องนิยามฟังก์ชัน operator< เพื่อใช้ในการเปรียบเทียบ object ของ Person ใน Set ได้ดังนี้

การนิยามฟังก์ชัน operator<

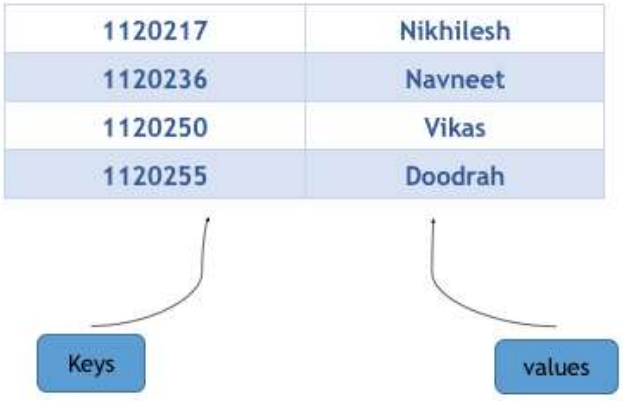
```
#include <iostream>
#include <set>
#include <string>
using namespace std;
class Person {
public:
    string name;
    int age;

    // constructor
    Person(string n, int a) {
        name = n;
        age = a;
    }
    // operator overloading
    bool operator<(const Person& other) const {
        if (name != other.name) {
            return name < other.name;
        } else {
            return age < other.age;
        }
    }
};
```

การค้นหา object

```
int main() {
    set<Person> persons;
    // insert objects of Person class into set
    persons.insert(Person("Prayuth", 30));
    persons.insert(Person("Somchai", 25));
    persons.insert(Person("Akreadej", 35));
    // search for a person by name and age
    Person p("Prayuth", 30);
    set<Person>::iterator it = persons.find(p);
    if (it != persons.end()) {
        cout << "Found " << it->name << " aged " << it->age << endl;
    } else {
        cout << "Person not found" << endl;
    }
    return 0;
}
```

map คืออะไร

- ในการเก็บค่าต่าง ๆ ในอะเรย์ จะเก็บข้อมูลหรือ value ต่อเนื่องกันไป โดยมีเลขที่ลำดับของอะเรย์เป็นตัวอ้างอิงลำดับการเก็บ
- ข้อเสียของอะเรย์คือถ้าไม่รู้ว่าข้อมูลที่ต้องการเก็บไว้ในอะเรย์ลำดับที่เท่าใด จะต้องเสียเวลาในการค้นหาข้อมูลที่ต้องการ O(n/2)
- Map คือ container ชนิดหนึ่งที่ใช้ในการเก็บข้อมูล ที่จะเก็บในลักษณะเป็นคู่ของ key-value โดยการอ้างอิงถึง value จะอ้างอิงโดย key



ภาพจาก https://www.studytonight.com/cpp/stl/stl-container-map

STL map

- STL map class เป็นการกำหนดคีย์กับข้อมูล โดยคีย์เป็นข้อมูลที่ไม่ซ้ำกันเลย เช่น เลขบัตรประชาชนกับชื่อสกุล เป็นต้น
- STL map class สามารถใช้ชนิดข้อมูลใดๆ ก็ได้เป็นคีย์หรือเป็นข้อมูล
- 1 to 1 mapping (หนึ่งคีย์สัมพันธ์กับข้อมูลหนึ่งข้อมูล)
- การใช้ STL map class จะต้อง #include <map>

Key-value pair

- Key ใน map จะเป็นอะไรก็ได้ เช่น primitive data type ต่าง ๆ เช่น int, double, float, char, string หรือแม้แต่ object
  - key ทุก key ใน map เดียวกัน ต้องเป็น data type เดียวกัน
  - Key แต่ละ key จะมีได้ค่าเดียว ไม่สามารถใช้ซ้ำกันได้
- Value ใน map ก็เช่นกัน จะเป็นอะไรก็ได้ เช่น primitive data type ต่าง ๆ เช่น int, double, float, char, string หรือแม้แต่ object
  - Value ทุก value ใน map เดียวกัน ต้องเป็น data type เดียวกัน
  - value สามารถเหมือนกันได้ เช่น value ของ key ต่างกัน อาจมีค่าเดียวกัน

map and unordered map

C++ มี container map ให้ใช้สองแบบ

- map เป็นการเก็บข้อมูลที่มีการเรียงลำดับ key จากน้อยไปหามาก (ใช้ tree ในการ implement
  - Search time จะอยู่ที่ O log(n)
  - นิยมใช้ในกรณีที่ต้องเรียงลำดับข้อมูล
- unordered map เป็นการเก็บข้อมูลที่ไม่มีการเรียงลำดับ key แต่ใช้ hash table
  - Search time เป็น O(1) หรือ
  - นิยมใช้ในกรณีที่ไม่สนใจลำดับของข้อมูล แต่เน้นความเร็วในการค้นหา

การสร้าง map

- การใช้ STL map class จะต้อง #include <map>
- ในการสร้าง map จะใช้คำสั่ง

```
std::map<string,int> mymap;
```

  - จากตัวอย่างเป็นการสร้าง map ชื่อ mymap โดย key จะเป็น string ส่วน value เป็น integer

- การสร้าง unordered map จะใช้

```
std::unordered_map <int, string> mymap;
```

  - จากตัวอย่างเป็นการสร้าง unordered map ชื่อ mymap โดย key จะเป็น integer ส่วน value เป็น string

map vs unordered_map

การใช้งาน map หรือ unordered_map จะขึ้นอยู่กับลักษณะของปัญหาและลักษณะของข้อมูล

- หากต้องการค้นหาข้อมูลด้วย key แบบเจาะจงและต้องการเข้าถึงข้อมูลโดยมีลำดับการจัดเก็บข้อมูลอย่างชัดเจน ให้ใช้ map
- หากต้องการเข้าถึงข้อมูลด้วย key แบบเจาะจง แต่ไม่จำเป็นต้องมีลำดับการจัดเก็บข้อมูลอย่างชัดเจน หรือต้องการใช้งานข้อมูลโดยสุ่ม ให้ใช้ unordered_map
- ประสิทธิภาพของการทำงาน map กับ unordered_map นั้นขึ้นอยู่กับลักษณะของข้อมูล และจำนวนข้อมูลที่มีอยู่ใน container
  - หากจำนวนข้อมูลมาก การใช้ unordered_map จะมีประสิทธิภาพที่ดีกว่า เนื่องจากการเข้าถึงข้อมูลจะใช้เวลาคงที่ ไม่ขึ้นอยู่กับจำนวนข้อมูลที่มีอยู่ใน container
  - การเข้าถึงข้อมูลใน map จะใช้เวลาโดยเพิ่มขึ้นตามจำนวนข้อมูลที่มีอยู่ใน container
  - ดังนั้นในกรณีที่ต้องการค้นหาข้อมูลโดยใช้ key แต่ไม่ต้องการใช้งานข้อมูลโดยสุ่ม และมีจำนวนข้อมูลมาก ให้ใช้ map
  - หากต้องการใช้งานข้อมูลโดยสุ่มหรือมีจำนวนข้อมูลมาก ให้ใช้ unordered_map แทน

ฟังก์ชันพื้นฐานของ map ที่ควรรู้

- size() จะ return จำนวน element ใน map
- empty() จะ return 1 ถ้า map นั้นไม่มี element ใด ๆ
- begin() จะ return iterator ไปยัง element แรกใน map นิยมใช้ในการเข้าไปดึงข้อมูลทุก element ใน map
- end() จะ return iterator ไปยังตำแหน่งที่ถัดจาก element สุดท้ายใน map (เป็นตำแหน่งในทางทฤษฎี เพื่อให้ระบุ element สุดท้ายได้)
- insert({key, value}) ใช้เพิ่ม element
- erase(key) ใช้ลบ element จาก key

การเพิ่มข้อมูลลงใน map

- การเพิ่มข้อมูลใน map ทำได้หลายแบบ เช่น

```
std::map<string,int> mymap;
mymap[“Somchai”] = 500;
// เป็นการเพิ่ม key “Somchai” ที่มี value 500

mymap.insert({“kratib”, 123456});
// เป็นการเพิ่ม key “kratib” ที่มี value 123456

cin >> mymap[“juan”];
// เป็นการสั่งให้รอรับค่า value จากแป้นพิมพ์ เพื่อไปใส่ใน element ที่ใช้คีย์
“juan”
```

การนำค่าใน map ไปใช้

- การนำ value มาใช้ สามารถใช้ชื่อ map ตามด้วยเครื่องหมาย [key] ได้เลย เช่น

```
std::map<string,int> mymap;
mymap[“Somchai”] = 500;
int newValue = maymap[“Somchai”];
// จากตัวอย่างเป็นการนำค่า 500 ซึ่งเป็น value ของ key “Somchai” ไปใส่ในตัวแปรที่ชื่อว่า newValue;
```

การลบ element

- ถ้าต้องการลบ element ใด ๆ ให้ใช้ฟังก์ชัน erase()
- ในฟังก์ชันให้ใช้ key เป็น parameter

```
std::map<string,int> mymap;
mymap[“Somchai”] = 500;
// เป็นการเพิ่ม key “Somchai” ที่มี value 500
mymap.erase(“Somchai”);
// เป็นการลบ element ที่ใช้ key “Somchai”
```

การค้นหาว่ามี key ที่ต้องการหรือไม่

- ในการค้นหาว่าใน map มี key ที่ต้องการหรือไม่ สามารถใช้วิธีดังตัวอย่าง

```
string x;
cin >> x;
mymap.erase("juan");
if(mymap.find(x) == mymap.end()){
//  if(mymap.count(x)<= 0){ //  ใช้แบบนี้ได้เช่นกัน
    cout << "not found\n";
}else{
    cout << "key " << x << " value " << mymap[x] << endl;
}
```

การแสดงค่าทุก element

- ในการแสดงค่าทุก element ใน map จะต้องใช้ iterator เพื่อช่วยในการเข้าถึง

  map<int, int>::iterator it;

  - จากตัวอย่างเป็นการสร้าง iterator object ชื่อ it ที่อ้างถึง map ชนิด <int,int>
  - ต้องกำหนดชนิดข้อมูลของ map ให้ตรงกับ map ที่จะใช้ iterator
- ใน c++ 11 ขึ้นไป สามารถใช้ keywork auto แทนการสร้าง iterator ตามรูปแบบด้านบนได้
- การใช้ keyword auto คอมไพเลอร์จะหาชนิดข้อมูลที่เหมาะสมให้เอง ทำให้ไม่ต้องระบุชนิดข้อมูล

การแสดงค่าทุก element

- เมื่อสร้าง iterator แล้ว จะใช้งานได้ดังตัวอย่าง

```
map<int, int>::iterator it;
for(it = mymap.begin(); it != mymap.end(); it++){
        cout << it->first << ":" << it->second << endl;
    }
```

- จากตัวอย่าง it->first จะเป็นค่า key
- It->second จะเป็น value ของ key นั้น
- ถ้าใช้ keywork auto จะไม่ต้องสร้าง iterator ก่อน สามารถใช้:

```
for(auto it = mymap.begin(); it != mymap.end(); it++){
        cout << it->first << ":" << it->second << endl;
    }
```

Range-based for loops

- การใช้งาน Range-based for loops เป็นวิธีหนึ่งใน C++ ที่ช่วยให้สามารถวนลูปผ่านตัวแปรที่เป็น range ได้ง่ายขึ้น เช่น อาร์เรย์, คอนเทนเนอร์ (เช่น vector หรือ map) หรือสตริง โดยรูปแบบ Syntax ของ Range-based for loop มีดังนี้:

```
for (auto& element : range) {
    // do something with element
}
```

- ใน range-based for loop ตัวแปร "range" สามารถเป็นประเภท range ใดก็ได้ที่มี iterator และตัวแปร "element" จะถูกกำหนดค่าอัตโนมัติเป็นแต่ละสมาชิกใน range เมื่อลูปวนลูปผ่าน
- การใช้ "auto&" ในส่วนหัวของลูปจะสร้างอ้างอิงไปยังสมาชิกปัจจุบัน ทำให้สามารถแก้ไขสมาชิกต้นฉบับใน range ได้หากจำเป็น แต่ถ้าใช้ "auto" จะเป็นการ copy ค่ามา ทำให้ไม่สามารถแก้ไขเปลี่ยนแปลง element ต้นทางได้ใน loop

Range-based for loops

- ตัวอย่างการใช้ **range-based for**

```cpp
#include <iostream>
#include <vector>

int main() {
    std::vector<int> v = {1, 2, 3, 4, 5};
    for (auto& x : v) {
        std::cout << x << " ";
    }
    return 0;
}
```

range-based for loops

- ในโค้ดนี้ for loop วนลูปผ่านทุกๆ element ใน vector v และกำหนดค่า element นั้นให้กับตัวแปร x ซึ่งเป็นการอ้างอิงไปยังตัวแปรต้นฉบับใน vector จากนั้น for loop จะพิมพ์แต่ละ element ลงบน console

- การใช้ range-based for loops สามารถทำให้โค้ดสั้นและอ่านง่ายกว่า traditional for loops เมื่อทำการวนลูปผ่าน ranges โดยเฉพาะเมื่อมีการใช้งาน complex types เช่น containers และ iterators

range-based for loops with map

```
#include <map>

using namespace std;

int main() {
    map<int, string> m = {{1, "one"}, {2, "two"}, {3, "three"}};
    for (auto& [key, value] : m) {
        cout << "Key: " << key << ", Value: " << value << endl;
    }

    return 0;
}
```

range-based for loops with map

```cpp
#include <iostream>
#include <iostream>
#include <map>

using namespace std;

int main() {
    map<int, string> m = {{1, "one"}, {2, "two"}, {3, "three"}};
    for (auto& at : m) {
        cout << "Key: " << at.first << ", Value: " << at.second << endl;
    }

    return 0;
}
```

ตัวอย่างการใช้ Map

```cpp
#include <iostream>
#include <map>
using namespace std;

int main() {
    // สร้าง map ของชนิดข้อมูล string เป็น key และ int เป็น value
    map<string, int> marks;
    // เพิ่มข้อมูลลงใน map
    marks["John"] = 85;
    marks["Mary"] = 92;
    marks["Peter"] = 78;
    // ใช้ฟังก์ชัน find() เพื่อค้นหาข้อมูลใน map
    auto it = marks.find("John");
    if (it != marks.end()) {
        cout << "John's score is " << it->second << endl;
    } else {
        cout << "John's score not found" << endl;
    }
    // วนลูปแสดงข้อมูลทั้งหมดใน map
    for (auto& p : marks) {
        cout << p.first << " got " << p.second << " marks." << endl;
    }
    return 0;
}
```

```
John's score is 85
John got 85 marks.
Mary got 92 marks.
Peter got 78 marks.

Process returned 0 (0x0)   execution time : 0.227 s
Press any key to continue.
```

อธิบายตัวอย่าง

- บรรทัดที่ 13 auto it = marks.find("John"); เป็นการค้นหาค่า John ใน container marks โดยใช้ฟังก์ชัน find() ซึ่งจะคืนค่าเป็น iterator ที่ชี้ไปยังสมาชิกใน marks ที่มีค่าเท่ากับ John หากไม่พบสมาชิกที่มีค่าเท่ากับ John ฟังก์ชัน find() จะคืนค่า iterator ที่ชี้ไปยังตำแหน่งหลังสุดของ marks ซึ่งสามารถตรวจสอบได้โดยใช้

  if (it != marks.end())

- บรรทัดที่ 14 if (it != marks.end()) เป็นการตรวจสอบว่า iterator it ที่ใช้วนลูปผ่านสมาชิกของ container marks นั้น ยังไม่ถึงตำแหน่งสุดท้ายของ container หรือไม่ ถ้า iterator it มีค่าเท่ากับ marks.end() แสดงว่า iterator นั้นชี้ไปยังตำแหน่งหลังสุดของ marks แล้ว ซึ่งไม่ใช่ตำแหน่งของสมาชิกใน marks ดังนั้นถ้า iterator it มีค่าไม่เท่ากับ marks.end() จะแสดงว่ามีสมาชิกใน marks อยู่และสามารถเข้าถึงได้

auto& p : marks

- auto& p : marks เป็น Syntax ของ Range-based for loop ในภาษา C++ ซึ่งใช้ในการวนลูปผ่านสมาชิกของ container marks โดย auto& p จะเป็นตัวแปรที่ใช้เก็บค่าของสมาชิกแต่ละตัวใน marks ตามลำดับ โดยประเภทของตัวแปร p จะถูกตัดสินใจโดยคอมไพเลอร์โดยอัตโนมัติจากประเภทของสมาชิกใน marks
- เมื่อลูปเริ่มทำงาน p จะถูกกำหนดค่าเป็นสมาชิกตัวแรกใน marks และบรรทัดต่อไปในลูป p จะถูกอัปเดตให้เป็นสมาชิกต่อไปใน marks จนกระทั่งวนลูปเสร็จสิ้น
- auto& แสดงถึงการประกาศ p ว่าเป็นตัวแปร reference ซึ่งจะชี้ไปยังตัวแปรจริงของสมาชิกใน marks แทนที่จะคัดลอกค่าของสมาชิกนั้นๆ ซึ่งมีประโยชน์ในกรณีที่ต้องการแก้ไขค่าของสมาชิกใน marks ผ่านตัวแปร p โดยไม่ต้องใช้การเข้าถึงผ่าน index หรือ iterator โดยตรง

การวนดูค่าทุก element ใน map วิธีอื่น

- ใช้ for loop ด้วย index

```
for (int i = 0; i < marks.size(); ++i) {
    std::cout << "Subject: " << marks[i].first << ", Marks: " << marks[i].second << std::endl;
}
```

- ใช้ while loop กับ iterator

```
auto it = marks.begin();
while (it != marks.end()) {
    std::cout << "Subject: " << it->first << ", Marks: " << it->second << std::endl;
    ++it;
}
```

- ใช้ foreach loop ด้วย std::for_each และ lambda function

```
std::for_each(marks.begin(), marks.end(), [](std::pair<std::string, int>& p) {
    std::cout << "Subject: " << p.first << ", Marks: " << p.second << std::endl;
});
```

ฝึกใช้ map

- ทดลองทำโจทย์ Count Frequency

การใช้งาน map กับ object

- เราสามารถใช้ container map กับชนิดข้อมูลที่เป็น object ได้เช่นกัน
- object สามารถเป็นได้ทั้ง key หรือ value
- ตัวอย่างการใช้งานกับ object เช่น กำหนด class Student ขึ้นมา โดยใน Student อาจมี
  - รหัสนักศึกษา (id)
  - ชื่อ (name)
  - คะแนน(score)
- เราอาจออกแบบโปรแกรม โดยใช้ id เป็น key และใช้ object Student เป็น value

การใช้งาน map กับ object

- สร้าง class Student และ function

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

class Student {
private:
    string student_name;
    int scores;
public:
    Student(string name, int score) : student_name(name), scores(score) {}

    string getName(){
        return student_name;
    }
    int getScore(){
        return scores;
    }
};
```

การสร้าง map ของ object

- สร้าง object ของ class Student และเพิ่มลงใน map

```
int main() {
    map<int, Student> students;

    Student std1("Alice", 90);
    Student std2("Bob", 80);
    Student std3("Charlie", 70);
    // add students using insert
    students.insert({1,std1});
    students.insert({2,std2});
    students.insert({3,std3});
```

การสร้าง map ของ object

- สร้าง object ของ class Student และเพิ่มลงใน map โดยไม่ต้องสร้าง object ก่อนก็ได้เช่นกัน โดยใช้ function make_pair ดังตัวอย่าง

```
students.insert(make_pair(1, Student("Alice", 90)));
students.insert(make_pair(2, Student("Bob", 80)));
students.insert(make_pair(3, Student("Charlie", 70)));
```

- นอกจากใช้ function insert ยังสามารถใช้ emplace ได้เช่นกัน

```
students.emplace(1, Student("Alice", 90));
students.emplace(2, Student("Bob", 80));
students.emplace(3, Student("Charlie", 95));
```

- การใช้ emplace จะมีการสร้าง object ตอนที่เรียกใช้งานเลย แต่การ emplace จะไม่มีการเพิ่ม element ถ้าใน map มี key นั้นก่อน แต่ถ้า insert จะเป็นการนำ object ใหม่ไปแทนที่ object เดิมถ้ามี key เดียวกัน

การใช้งาน map กับ object

- ถ้าต้องการเรียกใช้ function ใน object สามารถทำได้ดังตัวอย่าง

```
string x;
cin >> x;  // รับค่า key
Student std = mymap[x];
cout << std.getName() << endl;
 // สร้าง object ใหม่ขึ้นมาก่อน และเรียก function จาก object ใหม่
cout << students[x].getName() << endl;
 //เรียกใช้ object โดยตรงจาก  map
```

การใช้ iterator กับ map ของ object

- การใช้ **iterator** ไม่แตกต่างกับการใช้งานกับ **primitive data type**
- ถ้าต้องการเรียกใช้ **function** ของ **class** สามารถเรียกใช้ได้ดังในตัวอย่าง

```
    // iterate over students
    for (auto& [key, value] : students) {
        cout << "ID: " << key << ", Name: " << value.getName()
        << ", Score: " << value.getScore() << endl;
    }

    for(auto it = students.begin(); it != students.end(); it++){
        cout << "ID: " << it->first << ", Name: " << it->second.getName()
        << ", Score: " << it->second.getScore() << endl;
    }


    return 0;
}
```

ฝึกใช้ map

- ทดลองทำโจทย์ข้อ Computer