실습 077 - std::map with Person class

제출 마감일: 2023-05-07 23:59
업로드 가능한 파일 수: 2
제출 방식: 개인

목적

이 실습은 사용자가 정의한 객체를 std::map 의 Key 로 사용하는 연습을 합니다.

설명

std::map 은 고유 키(unique key)를 가지는 key-value pairs 를 저장하는 정렬된 컨테이너입니다.

std::map 의 key 로 우리가 정의한 클래스를 사용해 보겠습니다.

그러기 위해서는 먼저 std::map 에 관해 몇 가지 사실을 알고 있어야 합니다.

1. key 는 Compare 함수 오브젝트 (std::less<Key>) 로 비교하여 정렬됩니다. 즉, T 타입의 operator < 를 호출합니다.
2. key 의 고유성(uniqueness)는 equivalence relation 을 이용하여 결정합니다. 즉, 동등하다는 것은 !comp(a, b) && !comp(b, a) 표현식으로 결정합니다. (Compare requirements 충족)
3. std::map 은 일반적으로 red-black 트리로 구현됩니다. (검색, 제거, 입력이 logarithmic complexity 를 가짐)

참고: https://en.cppreference.com/w/cpp/container/map

먼저, 우리가 정의한 클래스가 Person 이라고 하면, Person 클래스에 operator < () 멤버 함수를 정의해야 Person 타입을 std::map 의 키로 사용할 수 있습니다.

bool operator < (const Person& rhs) const;

다음으로, uniqueness 와 equivalence relation 이라는 말이 어려운데, 코드로 이해해 보겠습니다.

```
int a = 1;
int b = 1;

if (a < b) return "a가 더 작음 (less)";
else if (b < a) return "b가 더 작음 (less)";
else return "a 와 b 는 동등함 (equivalence)"; // uniqueness in that C++ Standard Library uses the Compare requirements
```

즉, std::map 뿐만 아니라 C++ 의 표준 라이브러리에서 **Compare** requirements 를 사용하는 모든 곳에서 유일함 (uniqueness) 은 위 비교에서 else 인 경우를 의미한다고 이해가 됩니다.

키 값으로 사용할 객체의 유일성을 어떻게 구현할 것인지 고민이 필요해 보입니다.

마지막으로, map 자료구조는 보통 values 의 해시 (hash) 값을 키로 사용하며, 자료구조의 이름이 HashMap 인 언어도 있습니다. 그러나, C++ 의 std::map 은 red-black tree 로 내부가 구현되어 있어서 그런지 좀 특이한(?) 특성을 가지고 있습니다.

- 키 값이 중복인 데이터는 입력되지 않습니다.
- std::map 에 입력된 데이터의 키 값은 수정할 수 없습니다. 즉, 키 값이 const 가 됩니다.

그래서, 동일한 키 값을 덮어 쓰거나, 키 값을 수정하기 위해서는 삭제 후 입력해야 합니다.

문제

std::map 의 키로 사용할 수 있는 Person 클래스를 구현하여 프로그램이 정상 수행되도록 하시오.

<참고>

Person.h ----------

```
class Person {
public:
  Person(std::string name, size_t age) : name{name}, age{age} {
    // set ID with uniqueness
  }

  friend std::ostream& operator << (std::ostream& out, const Person& p) {
    // implementation
  }

  bool operator < (const Person& rhs) const {
    // implementation
  }

  class ComparatorByAge {
  public:
    bool operator()(const Person& lhs, const Person& rhs) const{
      // implementation
    }
  };

  class ComparatorByName {
  public:
    // implementation
```

```
    };

private:
    long long ID;
    std::string name;
    size_t age;
};



// main.cpp

#include "Person.h"

auto make_person(){
    std::string name; size_t age; size_t salary;
    std::cin >> name; std::cin >> age; std::cin >> salary;
    return std::pair(Person{name, age}, salary);
}

int main() {
    std::map<Person, size_t>  payroll;
    int N;
    std::cin >> N;
    for (int i=0; i < N; i++)
        payroll.insert(make_person());

    std::cout << "Sort By ID" << std::endl;
    for (const auto& [person, salary] : payroll)
        std::cout << person << " " << salary << std::endl;

    std::vector<std::pair<Person, size_t>> v_payroll(std::begin(payroll), std::end(payroll));

    std::cout << "Sort By Age" << std::endl;
    std::sort(std::begin(v_payroll), std::end(v_payroll),
            [](const auto& lhs, const auto& rhs) {
                auto Comparator = Person::ComparatorByAge();
                return Comparator(lhs.first, rhs.first);
            });
    for(const auto& [person, salary] : v_payroll)
        std::cout << person<< " " << salary << std::endl;

    std::cout << "Sort By Name" << std::endl;
    std::sort(std::begin(v_payroll), std::end(v_payroll),
            [](const auto& lhs, const auto& rhs) {
                auto Comparator = Person::ComparatorByName();
                return Comparator(lhs.first, rhs.first);
            });
    for(const auto& [person, salary] : v_payroll)
        std::cout << person<< " " << salary << std::endl;

    std::cout << "Sort By Salary" << std::endl;
    std::sort(std::begin(v_payroll), std::end(v_payroll),
            [](const auto& lhs, const auto& rhs){
                return lhs.second < rhs.second;
            });
    for(const auto& [person, salary] : v_payroll)
        std::cout << person<< " " << salary << std::endl;

    return 0;
}
```

입력

3
Lee 20 1000
Kim 10 3000
Ahn 30 2000

출력

Sort By ID

1 Lee 20 1000

2 Kim 10 3000

3 Ahn 30 2000

Sort By Age

2 Kim 10 3000

1 Lee 20 1000

3 Ahn 30 2000

Sort By Name

3 Ahn 30 2000

2 Kim 10 3000

1 Lee 20 1000

Sort By Salary

1 Lee 20 1000

3 Ahn 30 2000

2 Kim 10 3000

제출파일

Person.h
77.csv