Creative Software Programming Practice (week-9-1)

Every assignment will be announced on **Thursday** and should be sumitted by next **Tuesday**.

We have practice classes on Wednesdays and Thursdays.
The contents of the practice class are different from the assignments and aim to be completed on the same day.

Assignments will be publish on Thursday and must be submitted by the next Tuesday.

In this week **Handed out will be Oct 29, 2020, Due Nov 3, 2020**

Topics

1. Inheritance
2. Practices

1. Inheritance

Inheritance establishes an is-a relationship between a parent and a child. The is-a relationship is typically stated as as a specialization relationship, i.e., child is-a parent.

```
// inheritance.cc
#include <iostream>
#include <string>
#include <vector>

class Person {
public:
    int age;
    std::string name;

    Person(int age, std::string name)
    : age(age), name(name) {

    }

    void hello() {
        std::cout << "Hello, my name is " << this->name << std::endl;
    }
};

class Student
: public Person {
public:
    int student_id;
    std::string university;

    Student(int age, std::string name, int student_id, std::string university)
    : Person(age, name), student_id(student_id), university(university) {
    }
```

```
    // overriding
    void hello() {
        std::cout << "Hello, my name is " << this->name << std::endl;
        std::cout << "My id is " << this->student_id << std::endl;
    }
};

class Professor
: public Person {
public:
    std::string office;

    Professor(int age, std::string name, std::string office)
    : Person(age, name), office(office) {

    }

    // overriding
    void hello() {
        std::cout << "Hello, my name is " << this->name << std::endl;
        std::cout << "My office is " << this->office << std::endl;
    }
};

int main() {
    std::vector<Person> people;
    std::vector<Student> students;
    std::vector<Professor> professors;
    Student jiun{25, "jiun", 2019170229, "hyu"};
    Student jisu{24, "jisu", 2015170229, "cnu"};
    Student soomin{23, "soomin", 201417024, "hyu"};

    students.push_back(jiun);
    students.push_back(jisu);
    students.push_back(soomin);
    // we can handle Student as Person (because Student inherits Person)
    // But in this case, instance is considered like Person.
    people.push_back(jiun);
    people.push_back(jisu);
    people.push_back(soomin);

    Professor jongwoo{45, "jongwoo", "itbt505"};
    professors.push_back(jongwoo);
    people.push_back(jongwoo);

    for (int i = 0; i < people.size(); i++) {
        Person person = people[i];
        person.hello();
    }

    for (auto person : people) {
        person.hello();
    }

    for (auto student : students) {
        student.hello();
    }
```

```
    for (auto professor : professors) {
        professor.hello();
    }
}
```

2. Pratices

Implement using inheritance appropriately according to the specification below:

Person class has a name (`name`, *std::string*) and a current asset (`assets`, int).

Vehicle class has vector variables that contain maximum speed (`max_speed`, *float*), current speed (`cur_speed`, *float*), maximum number of passengers (`max_passengers`, *int*), and passengers (`passengers`, *vector < Person >*).
Methods include `start`, `stop`, `get_in`, `get_off` and `size`. `start` makes the current speed the maximum speed, and `stop` makes the current speed zero. `size` return count of current passengers. `get_in` and `get_off` have different implementations depending on the class they inherit from.

Car class inherits from the **Vehicle** class. It also has an owner variable (`owner`, Person). In the `get_in` function, you can ride if the person you want to ride is the owner or if the owner already rides. `get_off` is when the owner gets off, all passengers should get off.

Bus class inherits from the **Vehicle** class. It also has an charge variable (`charge`, int). In bus class, when `start` is called, the current speed becomes half of the maximum speed. You can get on board with `get_in` function and get off with `get_off` function. However, there is also overloading of `get_in` and `get_off`, which can be boarded or get off as a group. These functions take a `std::vector<Person&>` as an argument. All passengers on the bus will have their `assets` reduced by bus's `charge`.

Taxi class inherits the vehicle class and has a charge variable (`charge`, int). In the taxi class, multiple people can get on and off like the bus class, but additional boarding is not allowed when there are passengers. In addition, taxi class passengers also have to pay a charge, but pay when getting off.

Of course, passengers in all vehicles cannot exceed the maximum number of passengers (`max_passengers`). When boarding as a group, the **Bus** can accommodate part of the group up to the maximum number of passengers, but **Taxi* will not take all of them if even part of the group cannot.

NOTICE

It's pretty hard to implement all the specs. In Thursday's offline lab, I'll be going to demonstrate the implementation of this task. If you have completed this assignment by Wednesday and uploaded it to GitLab, you do not need to attend the offline classes on Thursday.

```
// vehicle.cc
#include <iostream>
#include <vector>
#include <algorithm>

class Person {
public:
    std::string name;
    int assets;
```

```
    Person(std::string name, int assets)
    : name(name), assets(assets) {
    }
};

bool operator==(const Person& lhs, const Person& rhs) {
    return &lhs == &rhs;
}


class Vehicle {
public:
    float max_speed, cur_speed;
    int max_passengers;
    std::vector<Person *> passengers;

    Vehicle(float max_speed, int max_passengers)
    : max_speed(max_speed), cur_speed(0), max_passengers(max_passengers) {
    }

    void start() {
        cur_speed = max_speed;
    }

    void stop() {
        cur_speed = 0.f;
    }

    int size() {
        return passengers.size();
    }

    virtual void get_in(Person& person) { }
    virtual void get_off(Person& person) { }
};

class Car : public Vehicle {
public:
    Person& owner;

    Car(float max_speed, int max_passengers, Person& owner)
    : Vehicle(max_speed, max_passengers), owner(owner) {
    }

    void get_in(Person& person) {
    }

    void get_off(Person& person) {
    }
};

class Bus : public Vehicle {
    int charge;

    Bus(float max_speed, int max_passengers, int charge)
    : Vehicle(max_speed, max_passengers), charge(charge) {
    }
```

```
    void get_in(Person& person) {
    }
    void get_in(std::vector<Person *> people) {
    }
    void get_off(Person& person) {
    }
    void get_off(std::vector<Person *> people) {
    }
};

class Taxi : public Vehicle {
    int charge;

    Taxi(float max_speed, int max_passengers, int charge)
        : Vehicle(max_speed, max_passengers), charge(charge) {
    }

    void get_in(Person& person) {
    }
    void get_in(std::vector<Person *> people) {
    }
    void get_off(Person& person) {
    }
    void get_off(std::vector<Person *> people) {
    }
};

int main() {
    Person jiun{"jiun", 300};

    Car jiun_car{30, 3, jiun};

    Person jisu1{"jisu1", 100};
    Person jisu2{"jisu2", 100};
    Person jisu3{"jisu3", 100};
    Person jisu4{"jisu4", 100};

    jiun_car.get_in(jiun);
    jiun_car.get_in(jisu1);
    jiun_car.get_in(jisu2);

    for (auto v : jiun_car.passengers) {
        std::cout << v->name << std::endl;
    }
}
```