ООО “МайТона”
Эл.почта: job@mytona.com

Дорогой кандидат!

Направляем Вам тестовое задание на вакансию **«Программист C++».**

Срок выполнения задания:

- Не более 2 недель с момента получения задания.

Формат выполнения задания:

- Выполненное задание необходимо залить на Google drive и отправить ссылку на задание на e-mail: job@mytona.com, в теме письма указать “Выполненное задание кандидата на позицию “Программист C++” _____ (фамилия, имя)”.

Задание на должность Программиста

Исправить логику и оптимизировать код. Допускается дополнение и изменение кода.

Программа решает очередность проезда машин на перекрестке:

- Если есть помеха справа то машина должна пропустить машину справа.
- Если на перекрестке стоят машины со всех 4х сторон, то первой проезжает машина, у которой минимальная координата по X.
- Машины не должны наезжать друг на друга.

```
struct sPos {
    sPos() { x = 0; y = 0; }
    sPos(int aX, int aY) { x = aX; y = aY; }
    int x;
    int y;
};

struct sSize
{
    sSize() { width = 0; height = 0; }
    sSize(int aW, int aH) { width = aW; height = aW; }
    int width;
    int height;
};

struct sRect
{
    sRect() {};
    sRect(int x, int y, int w, int h) { pos.x = x; pos.y = y; size.width = w; size.height = h; }
    sPos pos;
    sSize size;
    bool intersects(const sRect& other) {
        return !((other.pos.x + other.size.width <= pos.x) ||
            (other.pos.y + other.size.height <= pos.y) ||
            (other.pos.x >= pos.x + size.width) ||
            (other.pos.y >= pos.y + size.height));
    }
};

enum class eDirection {
    UP,
    LEFT,
    RIGHT,
    DOWN
};
```

```
struct sCar {
    sRect rect;
    eDirection dir;
    int speed;
    void move() {
        switch (dir) {
        case eDirection::up:
            rect.pos.y += speed;
        case eDirection::DOWN:
            rect.pos.y -= speed;
        case eDirection::RIGHT:
            rect.pos.x += speed;
        case eDirection::LEFT:
            rect.pos.x -= speed;
        }
    }

sRect getFuturePos() {
        switch (dir) {
        case eDirection::up:
            return sRect(rect.pos.x, rect.pos.y + speed, rect.size.width, rect.size.height);
        case eDirection::DOWN:
            return sRect(rect.pos.x, rect.pos.y - speed, rect.size.width, rect.size.height);
        case eDirection::RIGHT:
            return sRect(rect.pos.x + speed, rect.pos.y, rect.size.width, rect.size.width);
        case eDirection::LEFT:
            return sRect(rect.pos.x + speed, rect.pos.y, rect.size.width, rect.size.height);
        }
    }
```

```
bool needPassOtherCar(sCar* otherCar) {
        bool result;
        switch (dir) {
        case eDirection::up:
            auto otherdir = otherCar->dir;
            if (otherdir == eDirection::LEFT)
                result = true;
            break;
        case eDirection::DOWN:
            auto otherdir = otherCar->dir;
            if (otherdir == eDirection::RIGHT)
                result = true;
            break;
        case eDirection::RIGHT:
            auto otherdir = otherCar->dir;
            if (otherdir == eDirection::UP)
                result = true;
            break;
        case eDirection::LEFT:
            auto otherdir = otherCar->dir;
            if (otherdir == eDirection::LEFT)
                result = false;
            else
                result = false;
            break;
        }
        return result;
    }

virtual int getFuel() = 0;
    virtual void refill(int count) = 0;
};

struct sGasEngine : sCar {
    int getFuel() { return fuel; }
    void refill(int count) { fuel += count; }
    void move() { fuel--; sCar::move(); }
    int fuel;
};
```

```
struct sElectroCar : sCar {
    int getFuel() { return charge; }
    void refill(int count) { charge += count; }
    void move() { charge--; sCar::move(); }
    int charge;
};

struct sHybrid : sGasEngine, sElectroCar {
    void refill(int count) { charge += count / 2; fuel += count / 2; }
    int getFuel() { return charge + fuel; }
    void move() {
        if (rand() % 2 == 0)
            charge--;
        else
            fuel--;
        sCar::move();
    }
};

std::vector<sCar*> asdasd;
const int initialCarsCount = 10;
#define SCREEN_WIDTH 1024
#define SCREEN_HEIGHT 768
void spawnCar() {
    if (rand() % 4 == 1)
        spawnCarFromRight();
    else if (rand() % 4 == 2)
        spawnCarFromTop();
    else if (rand() % 4 == 3)
        spawnCarFromBot();
    else if (rand() % 4 == 4)
        SpawnCarFromLeft();
}
```

```
void spawnCarFromTop() {
    sCar* car;
    int carType = rand();
    if (carType % 3 == 0) {
        car = new sGasEngine();
    }
    else if (carType % 3 == 1) {
        car = new sElectroCar();
    }
    else if (carType % 3 == 2) {
        car = new sHybrid();
    }
    car->rect = sRect(SCREEN_WIDTH / 2, 0, 100, 100);
    car->speed = 1;
    car->dir = eDirection::DOWN;
}

void spawnCarFromBot() {
    sCar* car;
    int carType = rand();
    if (carType % 3 == 0) {
        car = new sGasEngine();
    }
    else if (carType % 3 == 1) {
        car = new sElectroCar();
    }
    else if (carType % 3 == 2) {
        car = new sHybrid();
    }
    car->rect = sRect(SCREEN_WIDTH / 2, SCREEN_HEIGHT, 100, 100);
    car->speed = 1;
}
```

```
void SpawnCarFromLeft() {
    sCar* car;
    int carType = rand();
    if (carType % 3 == 0) {
        car = new sGasEngine();
    }
    else if (carType % 3 == 1) {
        car = new sElectroCar();
    }
    else if (carType % 3 == 2) {
        car = new sHybrid();
    }
    car->rect = sRect(0, SCREEN_HEIGHT / 2, 100, 100);
    car->speed = 1;
}

void spawnCarFromRight() {
    sCar* car;
    int carType = rand();
    if (carType % 3 == 0) {
        car = new sGasEngine();
    }
    else if (carType % 3 == 1) {
        car = new sElectroCar();
    }
    else if (carType % 3 == 2) {
        car = new sHybrid();
    }
    car->rect = sRect(0, SCREEN_HEIGHT / 2, 100, 100);
    car->speed = 1;
}
```

```
bool main_loop() {
    for (auto car : asdasd) {
        for (auto car22 : asdasd) {
            if (car->getFuturePos().intersects(car22->getFuturePos())) {
                if (car->needPassOtherCar(car22))
                    car->move();
            }
            else {
                car22->move();
            }
        }
        if (car->rect.pos.x <= 0 || car->rect.pos.x >= SCREEN_WIDTH || car->rect.pos.y <= 0 ||
car->rect.pos.y >= SCREEN_HEIGHT)
            spawnCar();
    }
    return main_loop();
}

int main(int argc, char** argv) {
    for (auto i = 0; i < initialCarsCount; ++i) {
        spawnCar();
    }
    main_loop();
    return 0;
}
```

Все тестовые задания должны быть выполнены Вами самостоятельно и являться результатом исключительно Вашей интеллектуальной деятельности, созданы без нарушения прав третьих лиц, незаконных заимствований и плагиата. Направляя результат выполненных тестовых заданий, Вы подтверждаете, что понимаете и принимаете тот факт, что ООО "Майтона" могло получить или получать от третьих лиц, либо разработать или разрабатывать самостоятельно материалы, схожие с направленными Вами.