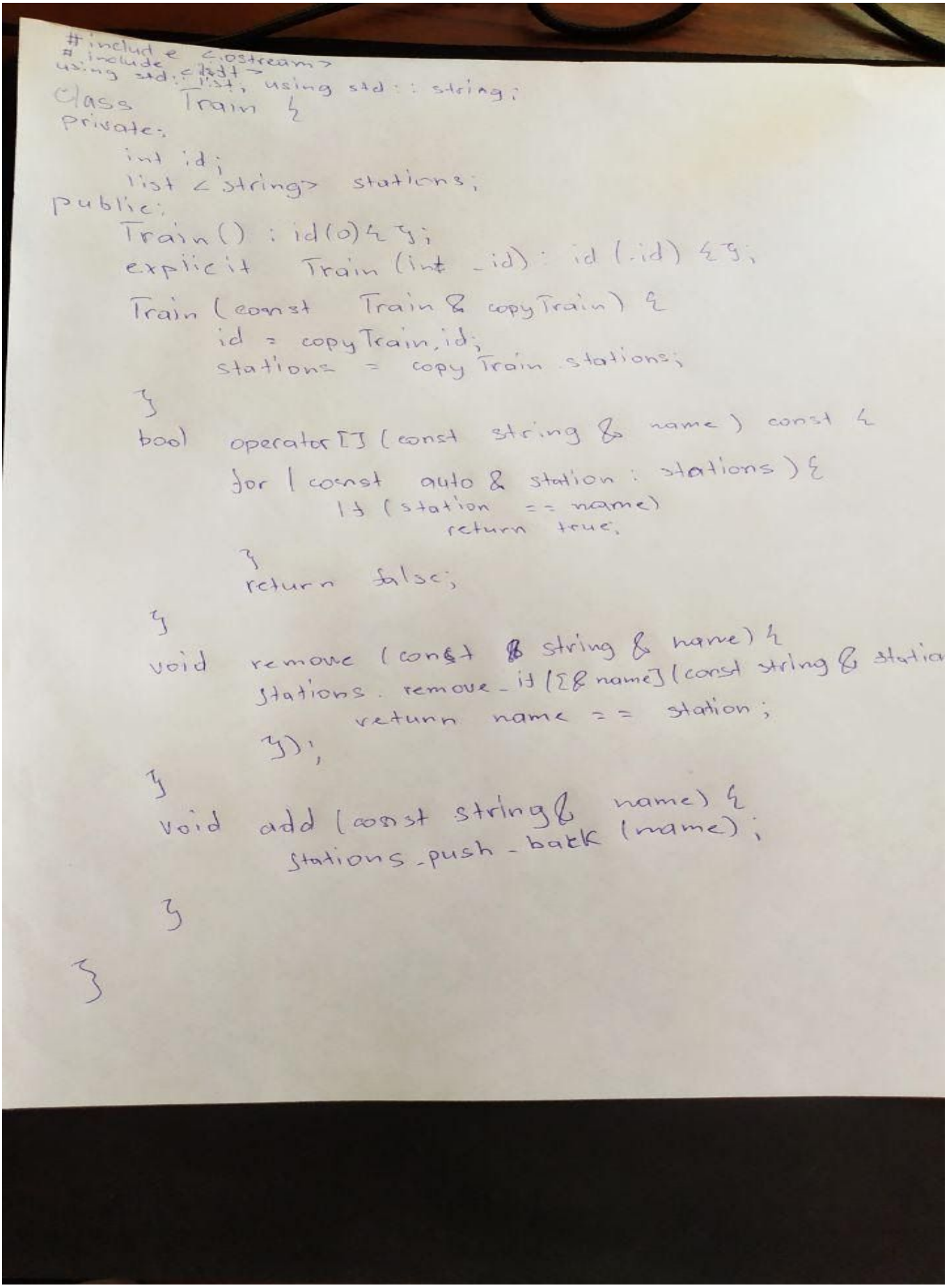
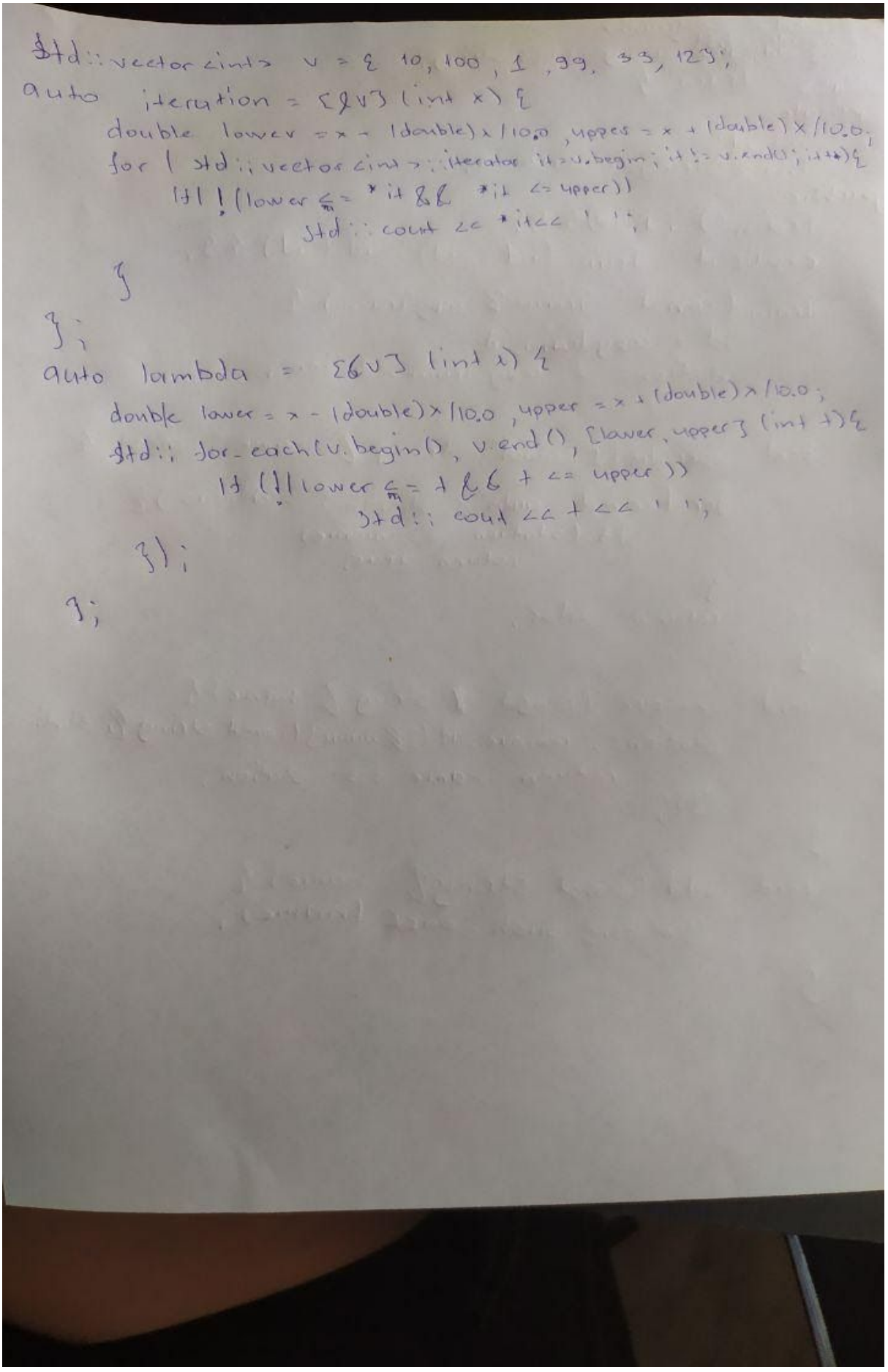
Задание 1. Class



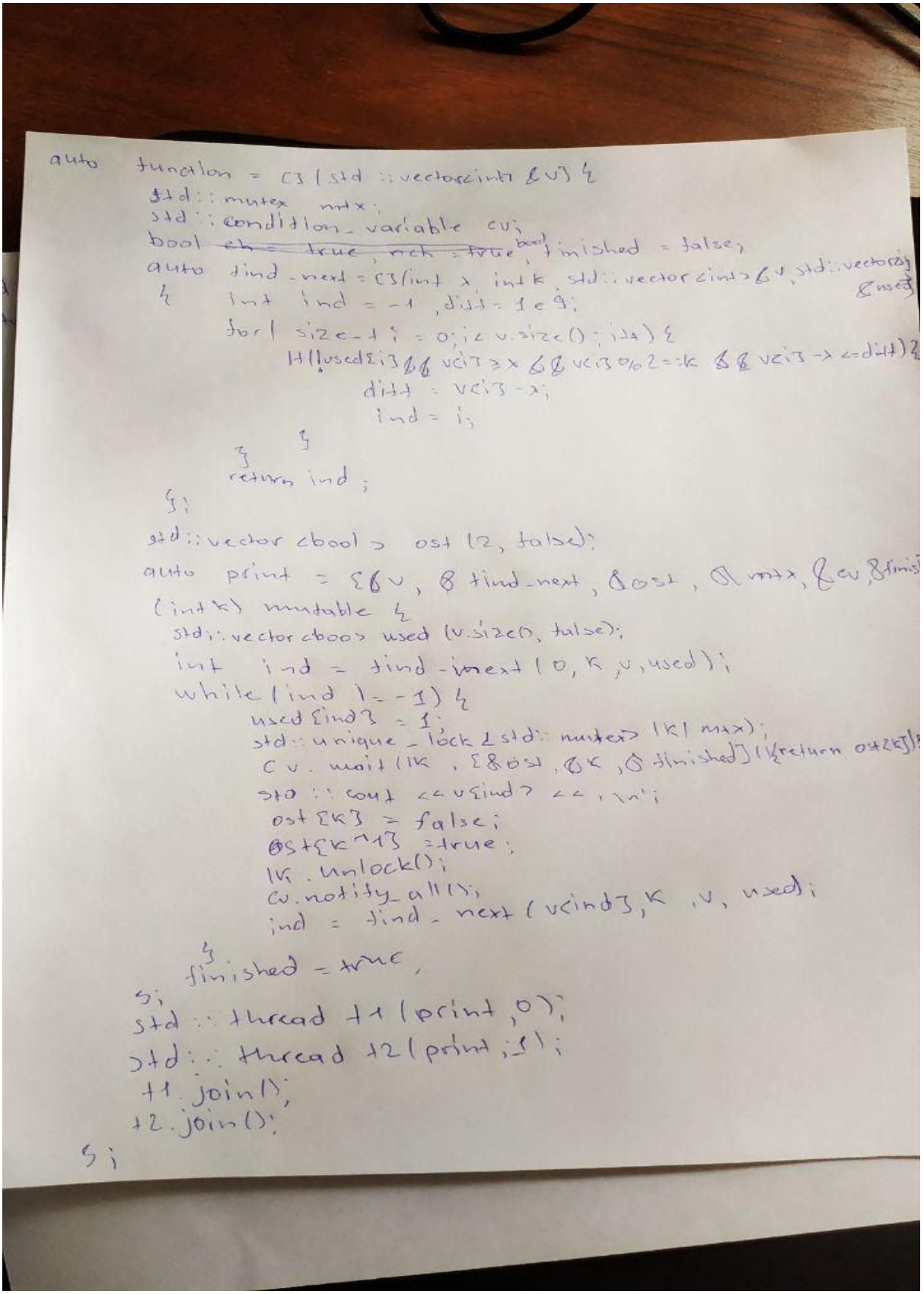
```
#include <iostream>
#include <list>
using std::list; using std::string;
class Train {
private:
    int id;
    list <string> stations;
public:
    Train() : id(0){};
    explicit Train(int _id) : id(_id) {};
    Train(const Train& copyTrain) {
        id = copyTrain.id;
        stations = copyTrain.stations;
    }
    bool operator[](const string& name) const {
        for(const auto& station : stations){
            if(station == name)
                return true;
        }
        return false;
    }
    void remove(const string& name){
        stations.remove_if([&name](const string& statio
            return name == station;
        });
    }
    void add(const string& name){
        stations.push_back(name);
    }
};
```

Задание 2 с вектором



```
std::vector<int> v = { 10, 100, 1, 99, 33, 123};
auto iteration = [&v] (int x) {
    double lower = x - (double)x/10.0, upper = x + (double)x/10.0;
    for (std::vector<int>::iterator it=v.begin; it != v.end(); it++){
        if (!(lower <= *it && *it <= upper))
            std::cout << *it << " ";
    }
};
auto lambda = [&v] (int x) {
    double lower = x - (double)x/10.0, upper = x + (double)x/10.0;
    std::for_each(v.begin(), v.end(), [lower, upper] (int t){
        if (!(lower <= t && t <= upper))
            std::cout << t << " ";
    });
};
```

Задание 3 Потоки



```
auto function = [](std::vector<int> &v) {
    std::mutex mtx;
    std::condition_variable cv;
    bool ch = true, mch = true, bool finished = false;
    auto find_next = [](int x, int k, std::vector<int> &v, std::vector<bool>
    {   int ind = -1, dist = 1e9;
        for(size_t i = 0; i < v.size(); i++) {
            if(!used[i] && v[i] >= x && v[i] % 2 == k && v[i] - x <= dist) {
                dist = v[i] - x;
                ind = i;
            }
        }
        return ind;
    };
    std::vector<bool> ost(2, false);
    auto print = [&v, &find_next, &ost, &mtx, &cv, &finis
    (int k) mutable {
        std::vector<bool> used(v.size(), false);
        int ind = find_next(0, k, v, used);
        while(ind != -1) {
            used[ind] = 1;
            std::unique_lock<std::mutex> lk(mtx);
            cv.wait(lk, [&ost, &k, &finished]{return ost[k]
            std::cout << v[ind] << ' ';
            ost[k] = false;
            ost[k^1] = true;
            lk.unlock();
            cv.notify_all();
            ind = find_next(v[ind], k, v, used);
        }
        finished = true;
    };
    std::thread t1(print, 0);
    std::thread t2(print, 1);
    t1.join();
    t2.join();
};
```