Solutions, C++ Programming Examination

2014–08–29

1. Private members in `Map`, then definition of member functions:

```
        struct City {
            double x;
            double y;
        };
        std::vector<City> cities;
        std::vector<std::vector<double>> dist;
        void compute_dist();

    Map::Map(size_t n) {
        default_random_engine e(time(0));
        uniform_real_distribution<double> rand(0, 1);
        for (size_t i = 0; i != n; ++i) {
            cities.push_back({rand(e), rand(e)});
        }
        compute_dist();
    }

    Map::Map(const string& file_name) {
        ifstream in(file_name);
        double x;
        double y;
        double min_x = 1E10, min_y = 1E10;
        double max_x = -1E10, max_y = -1E10;
        while (in >> x >> y) {
            cities.push_back({x, y});
            min_x = min(x, min_x);
            min_y = min(y, min_y);
            max_x = max(x, max_x);
            max_y = max(y, max_y);
        }
        double scale = 1 / max(max_x - min_x, max_y - min_y);
        for (City& c : cities) {
            c.x = (c.x - min_x) * scale;
            c.y = (c.y - min_y) * scale;
        }
        compute_dist();
    }

    size_t Map::size() const {
        return cities.size();
    }

    pair<double, double> Map::get_coords(size_t i) const {
        return {cities[i].x, cities[i].y};
    }

    double Map::get_dist(size_t i, size_t j) const {
        return dist[i][j];
    }
```

```
void Map::compute_dist() {
    dist.resize(cities.size());
    for (size_t i = 0; i != dist.size(); ++i) {
        dist[i].resize(cities.size());
    }
    for (size_t i = 1; i != cities.size(); ++i) {
        for (size_t j = 0; j != i; ++j) {
            double dx = cities[j].x - cities[i].x;
            double dy = cities[j].y - cities[i].y;
            dist[i][j] = dist[j][i] = sqrt(dx * dx + dy * dy);
        }
    }
}
```

2.
```
class Tour {
public:
    Tour(const Map& m);
    void create_random_tour();
    void draw(const Window& w) const;
    double get_length() const;
private:
    const Map& map;
    std::vector<size_t> tour;
};

Tour::Tour(const Map& m) : map(m), tour(m.size()) {}

void Tour::create_random_tour() {
    iota(tour.begin(), tour.end(), 0);
    default_random_engine e(time(0));
    shuffle(tour.begin(), tour.end(), e);
}

void Tour::draw(const Window& w) const {
    double scale = min(w.get_width(), w.get_height());
    pair<double, double> c1 = map.get_coords(0);
    for (size_t i = 1; i != tour.size(); ++i) {
        pair<double, double> c2 = map.get_coords(i);
        w.line(c1.first * scale, c1.second * scale, c2.first * scale, c2.second * scale);
        c1 = c2;
    }
    c2 = map.get_coords(0);
    w.line(c1.first * scale, c1.second * scale, c2.first * scale, c2.second * scale);
}

double Tour::get_length() const {
    double length = 0;
    for (size_t i = 0; i != tour.size() - 1; ++i) {
        length += map.get_dist(tour[i], tour[i + 1]);
    }
    length += map.get_dist(tour[tour.size() - 1], tour[0]);
    return length;
}
```

3.
```
void Tour::create_nearest_neighbor_tour() {
    vector<bool> visited(map.size());
    size_t current = 0;
    tour[0] = current;
    visited[current] = true;
    for (size_t i = 1; i != map.size(); ++i) {
        double min_dist = 1E10;
        size_t min_index = 0;
        for (size_t j = 0; j != map.size(); ++j) {
            if (!visited[j] && map.get_dist(current, j) < min_dist) {
                min_dist = map.get_dist(current, j);
                min_index = j;
            }
        }
        current = min_index;
        tour[i] = current;
        visited[current] = true;
    }
}
```

4.
```
void Tour::create_optimal_tour() {
    iota(tour.begin(), tour.end(), 0);
    double min_length = get_length();
    vector<size_t> best_tour = tour;
    while (next_permutation(tour.begin() + 1, tour.end())) {
        double l = get_length();
        if (l < min_length) {
            min_length = l;
            best_tour = tour;
        }
    }
    tour = best_tour;
}
```

5.
```
void Tour::improve_random() {
    default_random_engine e(time(0));
    uniform_int_distribution<unsigned> rand(0, tour.size() - 1);
    double min_length = get_length();
    for (size_t i = 0; i != 100000; ++i) {
        size_t n1 = rand(e);
        size_t n2 = rand(e);
        swap(tour[n1], tour[n2]);
        double l = get_length();
        if (l < min_length) {
            min_length = l;
        } else {
            swap(tour[n1], tour[n2]);
        }
    }
}
```

6.
```
void Tour::improve_2opt() {
    bool improved = true;
    while (improved) {
        improved = false;
        for (size_t n1 = 0; n1 < tour.size() - 2 && !improved; ++n1) {
            for (size_t n2 = n1 + 2; n2 < tour.size() && !improved; ++n2) {
                double delta = get_delta(n1, n2);
                if (delta < 0) {
                    reverse(tour.begin() + n1 + 1, tour.begin() + n2 + 1);
                    improved = true;
                }
            }
        }
    }
}

double Tour::get_delta(size_t i, size_t j) {
    int j1 = (j + 1) % tour.size();
    double d1 = map.get_dist(tour[i], tour[i + 1]);
    double d2 = map.get_dist(tour[j], tour[j1]);
    double d3 = map.get_dist(tour[i], tour[j]);
    double d4 = map.get_dist(tour[j1], tour[i + 1]);
    return (d3 + d4) - (d1 + d2);
}
```