HW_6 REPORT

محدرضابابایی مصلح

دانشگاه صنعتی امیرکبیر

در این گزارش به پاسخ دادن به چهار سوال مستقل از هم پرداختیم هر فایل را بهترتیب در ادامه شرح میدهیم اما پیش ار ان لازم به ذکر است در فایل های پیش رو در ابتدای توابعی که دارای حلقه هستند عبارت static قرار گرفته تا مانع از تعریف دوباره انها و redefinition error شود و در ابتدای تعریف توابعی که بدون حلقه هستند عبارت inline قرار گرفته تا باعث تعریف ان ها در لحظه استفاده شده و از همان ارور قبلی جلوگیری شود.

:q1

```
#ifndef Q1_H
#define Q1_H
#include<cmath>


namespace q1
{
    template<typename T, typename Func>
    inline double derivative(T point, Func func) //by using derivation definition
    {
        double h{ 0.001 }; // accuracy
        double derivation{ (func(point + h) - func(point - h)) / (2.0 * h) };

        return derivation;
    }

    template<typename T , typename Func>
    static double gradient_descent(T initial_value, double step_size , Func
func=Func{})
    {

        double learn_rate{ step_size } , tolerance{ 0.00001 }; //accuracy
parameters
        T tmp{ initial_value };
        double diff{ learn_rate * derivative(tmp, func) };
        while (std::abs(diff) >= tolerance)
        {
            diff = learn_rate * derivative(tmp, func);
            tmp = tmp - diff;
        }
```

```
        return tmp;
    }
}


#endif //Q1_H
```

اولین تابع این فایل تابع مشتق گیر است که به صورت template نوشته شده است به طور کلی این سوال نکته خاصی ندارد و تنها نکته ان قسمت زیر است:

```
    static double gradient_descent(T initial_value, double step_size , Func
func=Func{})
```

در این قسمت func متغیری است که در اصل function pointer است و در داخل تعریف تابع برابر با Func{} قرار داده شده است که به این دلیل است که در حالت پیش قرض objectدرست کند برای موارد که به ان functor پاس داده میشود.

:q2

```
#ifndef Q2_H
#define Q2_H

#include<iostream>
#include<string>
#include<fstream>
#include<sstream>
#include<regex>
#include<vector>


namespace q2
{

    struct Patient
```

```
    {
        Patient(std::string _name, size_t _age, size_t  _smokes, size_t _area_q,
size_t _alkhol)
            : name{ _name }
            , age{ _age }
            , smokes{ _smokes }
            , area_q {_area_q }
            , alkhol{ _alkhol }
        {
        };

        std::string name;
        size_t age;
        size_t smokes;
        size_t area_q;
        size_t alkhol;
    };

    static std::vector<Patient> read_file(std::string filename)
    {
        std::ifstream file(filename);
        std::stringstream buffer;
        buffer << file.rdbuf();
        std::string txt = buffer.str();

        std::regex pattern(R"((\w+)\ ?,(\w+)\ ?,(\d+)\,(\d+)\,(\d+)\,(\d+))");
        std::smatch match;

        std::vector<Patient> Patients{};

        while(std::regex_search(txt, match, pattern))
        {
            std::string first_name{ match[1] }, last_name{ match[2] };
            std::string full_name{ first_name + " " + last_name };
            size_t age{ static_cast<size_t>(std::stoi(match[3])) };
            size_t smokes{ static_cast<size_t>(std::stoi(match[4])) };
            size_t area_q{ static_cast<size_t>(std::stoi(match[5])) };
            size_t alkhol{ static_cast<size_t>(std::stoi(match[6])) };
            Patients.push_back(Patient{ full_name, age, smokes, area_q, alkhol
});
            txt = match.suffix().str();
        }

        return Patients;
```

```
    }

    //also can be defined as lambda of functor
    inline bool comparison(Patient& a, Patient& b)
    {
        return 3*(a.age) + 5*(a.smokes) + 2*(a.area_q) + 4*(a.alkhol) > 3*(b.age)
+ 5*(b.smokes) + 2*(b.area_q) + 4*(b.alkhol);
    }

    inline void sort(std::vector<Patient>& patients)
    {
        std::sort(patients.begin(), patients.end(), comparison);
    }
}

#endif //Q2_H
```

فایل دوم هم تنها دارای چند نکته است نکته اول ان بخش regex ان است که مانند الگوی بالا طراحی شده است تا اطلاعات مورد نظر را از داخل فایل در اختیار قرار گرفته استخراج کند.

نکته بعدی در تابع sort است که از توابع stl در خود دارد که تابع sort است که به عنوان دو المان اول خود iterator های ابتدا و انتها را میگیرد و به عنوان المان سوم خود یک تابع دریافت میکند که مقایسه را برای صورت بر اساس ان انجام میدهد.

:q3

```
#ifndef Q3_H
#define Q3_H

#include<iostream>
#include<queue>
#include<vector>

namespace q3
{
    struct Flight
    {
        Flight(std::string _Flight_number, size_t _duration, size_t _connections,
size_t _connection_times, size_t _price)
```

```
        : flight_number{ _Flight_number }
        , duration{ _duration }
        , connections{ _connections }
        , connection_times {_connection_times }
        , price{ _price }
        {
        }
        std::string flight_number;
        size_t duration;
        size_t connections;
        size_t connection_times;
        size_t price;
    };

    //function for turning time into minutes
    inline size_t time_cal(std::string time)
    {
        if(time.empty())
        {
            return 0;
        }

        std::regex pattern(R"((\d+)\h(\d+)?\m?)");
        std::smatch match;
        std::regex_search(time, match, pattern);
        size_t total_time{ static_cast<size_t>(std::stoi(match[1])) * 60 };
        std::string emp_check { match[2] };
        if(emp_check.empty()) //check if it contains minutes
        {
            return total_time;
        }
        else
        {
            return total_time + static_cast<size_t>(std::stoi(match[2]));

        }
    }

    //function for being use ass pq argument
    inline auto comparison{ [](Flight a, Flight b){return (a.duration +
a.connection_times + 3 * a.price) > (b.duration + b.connection_times + 3 *
b.price);} };

    inline std::priority_queue<Flight, std::vector<Flight>, decltype(comparison)>
Flight_info{comparison};
```

```
    static auto gather_flights(std::string filename)
    {
        std::ifstream file(filename);
        std::stringstream buffer;
        buffer << file.rdbuf();
        std::string txt = buffer.str();

        std::regex pattern(R"(\d+\- \w+\:(\w+)\ - \w+\:(\d+\h\d*\m*)\ -
\w+\:(\d+)\ - \w+\:(\d+\h\d*\m*)\,?(\d*\h*\d*\m*)\,?(\d*\h*\d*\m*) -
\w+\:(\d+))");
        std::smatch match;

        while(std::regex_search(txt, match, pattern))
        {

            std::string flight_number{ match[1] };
            size_t duration { time_cal(match[2]) };
            size_t connections{ static_cast<size_t>(std::stoi(match[3])) };
            size_t connection_times{ time_cal(match[4]) + time_cal(match[5]) +
time_cal(match[6]) };
            size_t price{ static_cast<size_t>(std::stoi(match[7])) };

            Flight_info.push(Flight{flight_number, duration, connections,
connection_times, price});
            txt = match.suffix().str();
        }

        return Flight_info;
    }

}

#endif //Q3_H
```

نکته این قایل استفاده از priority_queue است که به عنوان المان سوم خود یک تابع دریافت میکند که اعضایی که به ان داده میشود را بر ان اساس sort میکند

نکته بعدی وجود تابع time_cal است که در string که در ابتدا توسط regex استخراج شده بود را دوباره تحلیل کرده و زمان مورد نظر را از ان استخراج میکند.

نکته بعدی استفاده از تابع stoi برای تبدیل stringبه int است

:q4

```
#ifndef Q4_H
#define Q4_H
#include<numeric>

namespace q4
{
    struct Vector2D
    {
        Vector2D(double _x = 0, double _y = 0)
        :x{ _x }
        ,y{ _y }
        {
        }
        double x{};
        double y{};
    };

    struct Sensor
    {
        Sensor(Vector2D _pos, double _accuracy = 0)
        :pos{ _pos }
        ,accuracy{ _accuracy }
        {
        }
        Vector2D pos;
        double accuracy;
    };

    //functions for accumulate
    inline double avg_cal_x(double avg, Sensor& s)
    {
        return avg + s.pos.x * s.accuracy;
    }

    inline double avg_cal_y(double avg, Sensor& s)
    {
        return avg + s.pos.y * s.accuracy;
    }

    inline Vector2D kalman_filter(std::vector<Sensor> sensors)
```

```
    {
        double accuracy_sum{ std::accumulate(sensors.begin(), sensors.end(), 0.0
, [](double ans, Sensor& s){return ans + s.accuracy;}) };
        double x_pos{ std::accumulate(sensors.begin(), sensors.end(), 0.0,
avg_cal_x) / accuracy_sum };
        double y_pos{ std::accumulate(sensors.begin(), sensors.end(), 0.0,
avg_cal_y) / accuracy_sum };

        Vector2D final_pos{ x_pos, y_pos };

        return final_pos;
    }
}

#endif //Q4_H
```

تنها نکته ان استفاده از تابع stl به اسم accumulate است که دو iterator ابتدا و انتها را میگیرد به عنوان المان سوم مقدار اولیه را میگیرد و به عنوان المان چهارم یک تابع برای انجام عملیات روی المان های ارایه مورد نظر میکند.

پایان

Github link: https://github.com/ghostoftime111/hw6.git