

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 object درست 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 string به 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