Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет Кафедра «Информационные технологии и автоматизированные системы» направление подготовки: 09.03.04 - «Программная инженерия»

**Лабораторная работа №13**

**"Стандартные обобщенные алгоритмы библиотеки STL"**

Выполнил студент гр. РИС-24-3б Носков Егор Михайлович

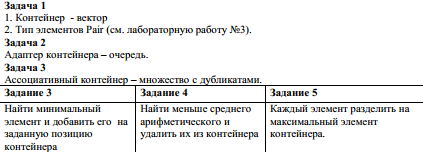
Проверил: Доцент кафедры ИТАС Ольга Андреевна Полякова

|  |  |
| --- | --- |
|  | |
| (оценка) (подпись) | |
|  |
| (дата) |

Г. Пермь, 2024

**Постановка задачи**

**Вариант 1:**

****

**Программа на языке C++**

Pair.h:

#pragma once

#include <iostream>

using namespace std;

class Pair

{

private:

int first;

double second;

public:

Pair() : first(0), second(0.0) {}

Pair(int first, double second) : first(first), second(second) {};

Pair(const Pair& p) : first(p.first), second(p.second) {};

~Pair() {}

int get\_first() const { return first; }

int get\_second() const { return second; }

void set\_first(int f) { first = f; }

void set\_second(int s) { second = s; }

bool operator<(const Pair& pair) const;

bool operator>(const Pair& pair) const;

Pair operator+(const Pair& pair) const;

Pair operator/(size\_t size) const;

Pair operator/(const Pair& pair) const;

friend ostream& operator<<(ostream& out, const Pair& pair);

friend istream& operator>>(istream& in, Pair& pair);

};

Pair.cpp:

#include "Pair.h"

bool Pair::operator<(const Pair& pair) const

{

return (static\_cast<double>(first + second) < static\_cast<double>(pair.first + pair.second));

}

bool Pair::operator>(const Pair& pair) const

{

return (static\_cast<double>(first + second) > static\_cast<double>(pair.first + pair.second));

}

Pair Pair::operator+(const Pair& pair) const

{

return Pair(first + pair.first, second + pair.second);

}

Pair Pair::operator/(size\_t size) const

{

return Pair(first / size, second / size);

}

Pair Pair::operator/(const Pair& pair) const

{

return Pair(first / pair.first, second / pair.second);

}

ostream& operator<<(ostream& out, const Pair& pair)

{

out << '(' << pair.first << " : " << pair.second << ')';

return out;

}

istream& operator>>(istream& in, Pair& pair)

{

cout << "Enter pair first: "; in >> pair.first;

cout << "Enter pair second: "; in >> pair.second;

return in;

}

Lab13OOP\_main:

#include <iostream>

#include <vector>

#include <queue>

#include <map>

#include <algorithm>

#include "Pair.h"

using namespace std;

typedef vector<Pair> pvec;

pvec make\_pvec(int n)

{

pvec pv;

Pair tmp;

for (int i = 0; i < n; ++i)

{

cin >> tmp;

pv.push\_back(tmp);

}

return pv;

}

void print\_pvec(const pvec& pv)

{

if (pv.empty())

{

cout << "Vector is empty !\n";

return;

}

cout << "====================\n";

for (const auto& it : pv)

{

cout << it << '\n';

}

cout << "====================\n";

}

void add\_min\_at\_pos(pvec& pv, int pos)

{

if (pv.empty()) return;

Pair min\_el = pv[0];

for (const auto& it : pv)

{

if (it < min\_el) min\_el = it;

}

auto it = pv.begin() + pos;

pv.insert(it, min\_el);

}

void remove\_below\_average(pvec& pv)

{

if (pv.empty()) return;

Pair avg(0, .0);

for (const auto& it : pv)

{

avg = avg + it;

}

avg = avg / pv.size();

pv.erase(

remove\_if(pv.begin(), pv.end(), [&avg](const Pair& pair) {return pair < avg; }),

pv.end()

);

}

void divide\_by\_max(pvec& pv)

{

if (pv.empty()) return;

Pair max\_el = pv[0];

for (const auto& it : pv)

{

if (it > max\_el) max\_el = it;

}

for\_each(pv.begin(), pv.end(), [&max\_el](Pair& it)

{ it = it / max\_el; });

}

typedef queue<Pair> pque;

pque make\_queue(int n)

{

pque pq;

Pair tmp;

for (int i = 0; i < n; ++i)

{

cin >> tmp;

pq.push(tmp);

}

return pq;

}

void print\_pvec(const pque& pq)

{

if (pq.empty())

{

cout << "Queue is empty !\n";

return;

}

pque tmp = pq;

cout << "====================\n";

while (!tmp.empty())

{

cout << tmp.front() << '\n';

tmp.pop();

}

cout << "====================\n";

}

void add\_min\_at\_pos(pque& pq, int pos)

{

if (pq.empty()) return;

Pair min\_el = pq.front();

pque tmp = pq;

while (!tmp.empty())

{

if (tmp.front() < min\_el) min\_el = tmp.front();

tmp.pop();

}

for (int i = 0; i < pos; ++i)

{

tmp.push(pq.front());

pq.pop();

}

tmp.push(min\_el);

while (!pq.empty())

{

tmp.push(pq.front());

pq.pop();

}

pq = move(tmp);

}

void remove\_below\_average(pque& pq)

{

if (pq.empty()) return;

Pair avg(0, 0.0);

pque tmp = pq;

while (!tmp.empty())

{

avg = avg + tmp.front();

tmp.pop();

}

avg = avg / pq.size();

while (!pq.empty())

{

if (pq.front() > avg) tmp.push(pq.front());

pq.pop();

}

pq = move(tmp);

}

void divide\_by\_max(pque& pq)

{

if (pq.empty()) return;

Pair max\_el = pq.front();

pque tmp = pq;

while (!tmp.empty())

{

if (tmp.front() > max\_el) max\_el = tmp.front();

tmp.pop();

}

while (!pq.empty())

{

tmp.push(pq.front() / max\_el);

pq.pop();

}

pq = move(tmp);

}

typedef multimap<int, Pair> PairMultimap;

PairMultimap make\_pair\_multimap(int n)

{

PairMultimap pm;

Pair el;

for (int i = 0; i < n; ++i)

{

cout << "Enter pair: " << endl;

cin >> el;

pm.insert(make\_pair(i, el));

}

return pm;

}

void print\_pair\_multimap(const PairMultimap& pm)

{

cout << "==============================\n";

for (auto it = pm.begin(); it != pm.end(); ++it)

{

cout << '(' << it->first << ':' << it->second << ")\n";

}

cout << "==============================\n";

}

void add\_min\_at\_position(PairMultimap& pm, const int pos)

{

if (pos < 0 || pos > pm.size() || pm.empty()) return;

Pair min\_el = (pm.begin())->second;

for (const auto& it : pm)

{

if (it.second < min\_el) min\_el = it.second;

}

auto it = pm.begin();

advance(it, pos);

pm.insert(it, make\_pair(pos, min\_el));

}

void remove\_below\_average(PairMultimap& pm)

{

if (pm.empty()) return;

Pair sum\_pair;

for (const auto& it : pm)

{

sum\_pair.set\_first(sum\_pair.get\_first() + it.second.get\_first());

sum\_pair.set\_second(sum\_pair.get\_second() + it.second.get\_second());

}

sum\_pair.set\_first(sum\_pair.get\_first() / pm.size());

sum\_pair.set\_second(sum\_pair.get\_second() / pm.size());

for (auto it = pm.begin(); it != pm.end(); )

{

if ((it->second) < sum\_pair) it = pm.erase(it);

else ++it;

}

}

void divide\_by\_max(PairMultimap& pm)

{

if (pm.empty()) return;

Pair max\_pair = pm.begin()->second;

for (const auto& it : pm)

{

if (it.second > max\_pair) max\_pair = it.second;

}

for (auto& it : pm)

{

it.second = it.second / max\_pair;

}

}

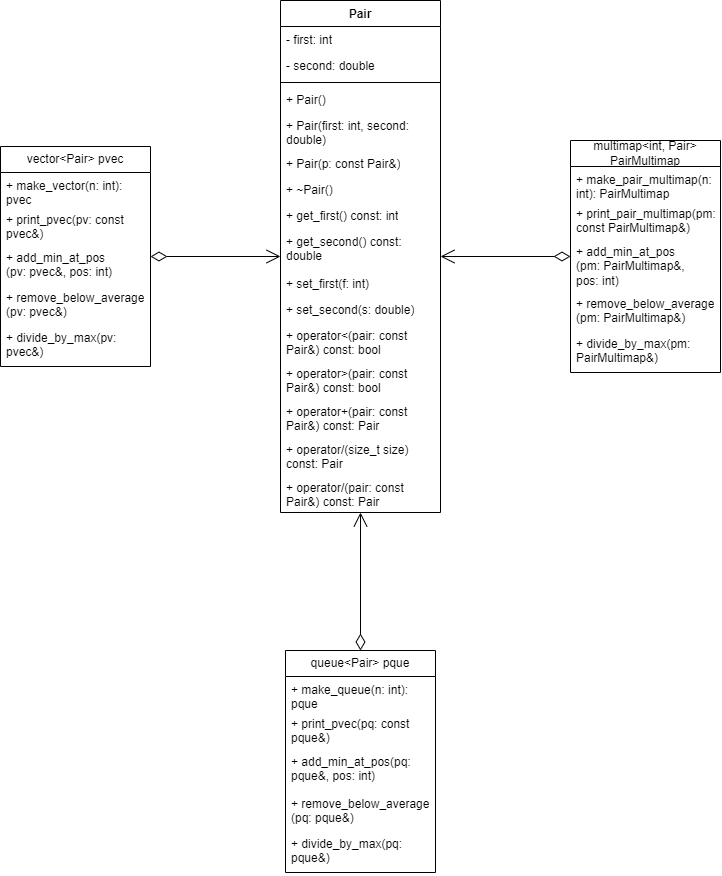
int main()

{

return 0;

}

**UML-диаграмма классов**



**Ссылка на репозиторий с отчетом Github**

****