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 №12**

**"Ассоциативные контейнеры библиотеки STL"**

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

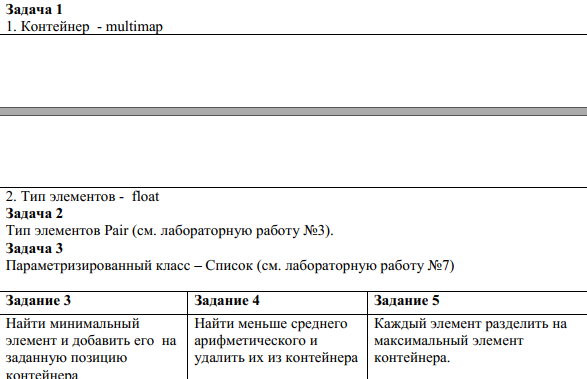
Проверил: Доцент кафедры ИТАС Ольга Андреевна Полякова

|  |  |
| --- | --- |
|  | |
| (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;

int second;

public:

Pair() { first = 0; second = 0; }

Pair(int f, int s) { first = f; second = s; }

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

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; }

Pair operator+(const Pair&);

Pair operator-(const Pair&);

Pair operator/(const Pair&);

Pair operator/(const int);

bool operator<(const Pair&) const;

bool operator>(const Pair&) const;

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

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

};

Pair.cpp:

#include "Pair.h"

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

{

int f = first + p.first;

int s = second + p.second;

f += s / 100;

s = s % 100;

return Pair(f, s);

}

Pair Pair::operator-(const Pair& p)

{

int f = first - p.first;

int s = second - p.second;

return Pair(f, s);

}

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

{

int total1 = first \* 100 + second;

int total2 = p.first \* 100 + p.second;

return Pair(0, total1 / total2);

}

Pair Pair::operator/(const int n)

{

int total = first \* 100 + second;

total /= n;

return Pair(total / 100, total % 100);

}

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

{

int total1 = first \* 100 + second;

int total2 = p.first \* 100 + p.second;

return total1 < total2;

}

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

{

int total1 = first \* 100 + second;

int total2 = p.first \* 100 + p.second;

return total1 > total2;

}

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

{

cout << "First: "; in >> p.first;

cout << "Second: "; in >> p.second;

return in;

}

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

{

out << '(' << p.first << "," << p.second << ')';

return out;

}

Container.h:

#pragma once

#include <list>

template<class T>

class Container

{

private:

list<T> lst;

public:

Container() {}

Container(int n)

{

T a;

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

{

cin >> a;

lst.push\_back(a);

}

}

void Print()

{

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

{

cout << \*it << " ";

}

cout << endl;

}

T find\_min()

{

return \*min\_element(lst.begin(), lst.end());

}

void add\_at\_position(int pos, T val)

{

auto it = lst.begin();

advance(it, pos);

lst.insert(it, val);

}

void remove\_below\_average()

{

T sum = accumulate(lst.begin(), lst.end(), T());

T avg = sum / lst.size();

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

{

if (\*it < avg) it = lst.erase(it);

else ++it;

}

}

void divide\_by\_max()

{

T max\_val = \*max\_element(lst.begin(), lst.end());

for (auto& elem : lst)

{

elem = elem / max\_val;

}

}

};Lab11\_task2.cpp:

#include <iostream>

#include <deque>

#include "Money.h"

using namespace std;

typedef deque<Money>Deq;

Deq make\_deque(int n)

{

Deq d;

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

{

long r = rand() % 1000;

int k = rand() % 100;

Money m(r, k);

d.push\_back(m);

}

return d;

}

void print\_deque(Deq d)

{

for (int i = 0; i < d.size(); i++)

{

cout << d[i] << " ";

}

cout << endl;

}

Deq push\_back\_max(Deq d)

{

Money max = d[0];

for (int i = 0; i < d.size(); i++)

{

if (d[i] > max) max = d[i];

}

d.push\_back(max);

return d;

}

Deq remove\_elem(Deq d, Money value)

{

for (deque<Money>::iterator di = d.begin(); di != d.end();)

{

if (\*di == value)

di = d.erase(di);

else

di++;

}

return(d);

}

Deq add\_mean\_all(Deq d)

{

Money sum;

for (int i = 0; i < d.size(); i++)

{

sum = sum + d[i];

}

cout << sum << "\n";

Money mean = sum / d.size();

cout << mean << "\n";

for (int i = 0; i < d.size(); i++)

{

d[i] = d[i] + mean;

}

return d;

}

int main()

{

try

{

deque<Money> d;

int n;

cout << "Enter n: "; cin >> n;

d = make\_deque(n);

cout << "deque: "; print\_deque(d);

d = push\_back\_max(d);

cout << "push\_back\_max: "; print\_deque(d);

Money value;

cout << "Enter the value to remove: \n"; cin >> value;

d = remove\_elem(d, value);

cout << "remove\_elem: "; print\_deque(d);

d = add\_mean\_all(d);

cout << "add\_mean\_all: "; print\_deque(d);

}

catch (int)

{

cout << "Error!";

}

}

Lab12OOP\_main.cpp:

#include <iostream>

#include <map>

#include <algorithm>

#include "Pair.h"

using namespace std;

typedef multimap<int, float> FloatMultimap;

FloatMultimap make\_float\_multimap(int n)

{

FloatMultimap fm;

float el;

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

{

cout << "Enter multimap element: ";

cin >> el;

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

}

return fm;

}

void print\_float\_multimap(const FloatMultimap& fm)

{

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

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

{

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

}

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

}

void add\_min\_at\_position(FloatMultimap& fm, const int pos)

{

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

float min\_el = (fm.begin())->second;

for (const auto& it : fm)

{

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

}

auto it = fm.begin();

advance(it, pos);

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

}

void remove\_below\_average(FloatMultimap& fm)

{

float sum = 0;

for (const auto& it : fm) sum += it.second;

float avg = sum / fm.size();

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

{

if (it->second < avg) it = fm.erase(it);

else ++it;

}

}

void divide\_by\_max(FloatMultimap& fm)

{

if (fm.empty()) return;

float max\_el = (fm.begin())->second;

for (const auto& it : fm)

{

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

}

for (auto& it : fm) it.second /= max\_el;

}

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;

}

}

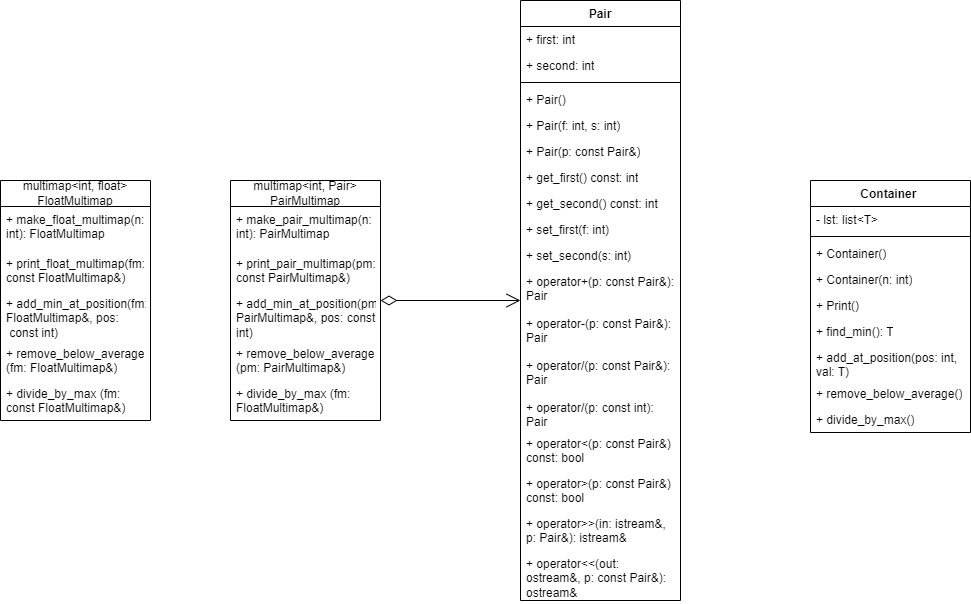
int main()

{

return 0;

}

**UML-диаграмма классов**



**Контрольные вопросы**

#### **1. Что представляет собой ассоциативный контейнер?**

Ассоциативный контейнер — это структура данных, хранящая элементы в виде пар «ключ-значение» (например, map) или только ключей (например, set), обеспечивающая быстрый поиск по ключу. Элементы автоматически сортируются по ключу.

**2. Ассоциативные контейнеры STL:**

* map — упорядоченный набор уникальных пар «ключ-значение».
* multimap — аналогичен map, но позволяет дубликаты ключей.
* set — упорядоченный набор уникальных ключей.
* multiset — аналогичен set, но позволяет дубликаты ключей.
* unordered\_map, unordered\_set — неупорядоченные версии (хеш-таблицы).

**3. Доступ к элементам:**

* По ключу: map[key] или map.at(key).
* С помощью итераторов: begin(), end().
* Методы find(), lower\_bound().

**4. Примеры методов:**

map<int, string> m;

m.insert(make\_pair(1, "one")); *// Вставка*

m.erase(1); *// Удаление*

auto it = m.find(1); *// Поиск*

m.size(); *// Размер*

m.empty(); *// Проверка на пустоту*

**5. Создание**map**:**

#include <map>

map<int, string> m1; *// Пустой map*

map<string, float> m2 = {{"pi", 3.14}, {"e", 2.71}}; *// Инициализация*

**6. Упорядочивание в**map**:**

* По умолчанию: по возрастанию ключа (используется less<Key>).
* Для обратного порядка:

map<int, string, greater<int>> m;

**7. Операции для**map**:**

* Вставка: insert(), emplace(), operator[].
* Удаление: erase(), clear().
* Поиск: find(), count(), lower\_bound().
* Доступ: operator[], at().

**8. Добавление через**make\_pair**:**

void addToMap(map<int, string>& m, int key, string value) {

m.insert(make\_pair(key, value));

}

**9. Добавление через**operator[]**:**

void addToMap(map<int, string>& m, int key, string value) {

m[key] = value;

}

**10. Печать через итератор:**

void printMap(const map<int, string>& m) {

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

cout << it->first << ": " << it->second << endl;

}

}

**11. Печать через**operator[]**(не рекомендуется, так как требует наличия ключей):**

void printMap(const map<int, string>& m) {

for (const auto& pair : m) {

cout << pair.first << ": " << pair.second << endl;

}

}

**12. Отличие**map**от**multimap**:**

* map: ключи уникальны.
* multimap: допускает дубликаты ключей.

**13. Контейнер**set**:**

Упорядоченный набор уникальных элементов (ключей), где значение совпадает с ключом.

**14. Отличие**map**от**set**:**

* map хранит пары «ключ-значение».
* set хранит только ключи.

**15. Создание**set**:**

#include <set>

set<int> s1 = {1, 2, 3}; *// Инициализация*

set<string> s2; *// Пустой set*

**16. Упорядочивание в**set**:**

* По умолчанию: по возрастанию (less<Key>).
* Для обратного порядка:

set<int, greater<int>> s;

**17. Операции для**set**:**

* Вставка: insert(), emplace().
* Удаление: erase(), clear().
* Поиск: find(), count().

**18. Добавление в**set**:**

void addToSet(set<int>& s, int value) {

s.insert(value);

}

**19. Печать**set**:**

void printSet(const set<int>& s) {

for (const auto& elem : s) {

cout << elem << " ";

}

cout << endl;

}

**20. Отличие**set**от**multiset**:**

* set: элементы уникальны.
* multiset: допускает дубликаты.

**Ссылка на репозиторий с отчетом Github**

****