Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

# Лабораторная работа №13

# «Стандартные обобщенные алгоритмы библиотеки STL»

# Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Юхновец В.Г.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**Постановка задачи**

**Задача 1.**

1. Создать последовательный контейнер.

2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.

3. Заменить элементы в соответствии с заданием (использовать алгоритмы replace\_if(), replace\_copy(), replace\_copy\_if(), fill()).

4. Удалить элементы в соответствии с заданием (использовать алгоритмы remove(),remove\_if(), remove\_copy\_if(),remove\_copy())

5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).

6. Найти в контейнере заданный элемент (использовать алгоритмы find(), find\_if(), count(), count\_if()).

7. Выполнить задание варианта для полученного контейнера (использовать алгоритм for\_each()).

8. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

**Задача 2.**

1. Создать адаптер контейнера.

2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.

3. Заменить элементы в соответствии с заданием (использовать алгоритмы replace\_if(), replace\_copy(), replace\_copy\_if(), fill()).

4. Удалить элементы в соответствии с заданием (использовать алгоритмы remove(),remove\_if(), remove\_copy\_if(),remove\_copy())

5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).

6. Найти в контейнере элемент с заданным ключевым полем (использовать алгоритмы find(), find\_if(), count(), count\_if()).

7. Выполнить задание варианта для полученного контейнера (использовать алгоритм for\_each()).

8. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

**Задача 3.**

1. Создать ассоциативный контейнер.

2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.

3. Заменить элементы в соответствии с заданием (использовать алгоритмы replace\_if(), replace\_copy(), replace\_copy\_if(), fill()).

4. Удалить элементы в соответствии с заданием (использовать алгоритмы remove(),remove\_if(), remove\_copy\_if(),remove\_copy())

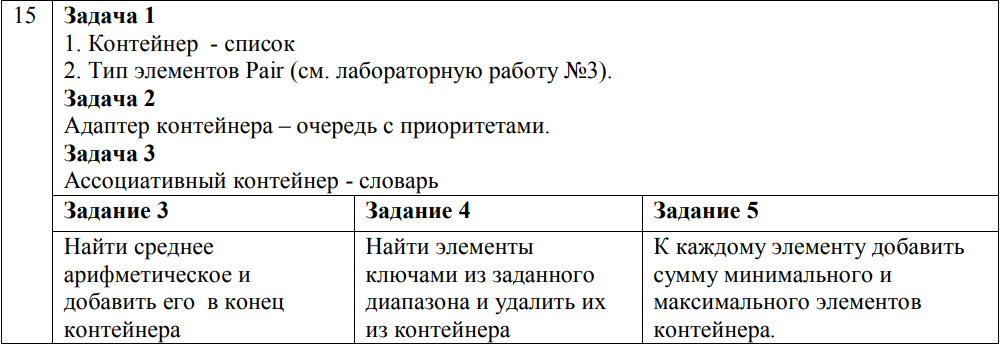
5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).

6. Найти в контейнере элемент с заданным ключевым полем (использовать алгоритмы find(), find\_if(), count(), count\_if()).

7. Выполнить задание варианта для полученного контейнера (использовать алгоритм for\_each()) .

8. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

**Вариант 15**



**Диаграмма класса**

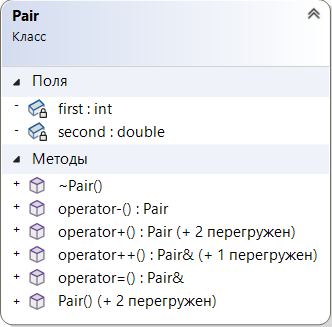


Рисунок 1 – диаграмма класса Pair

Код программы

Sourse.cpp

#include <iostream>

using namespace std;

#include "Pair.h"

#include "Ex1.h"

#include "Ex2.h"

#include "Ex3.h"

int main()

{

foo1();

foo2();

foo3();

return 0;

}

Pair.h

#pragma once

#include <iostream>

#include <iomanip>

using namespace std;

class Pair

{

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

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

private:

int first;

double second;

public:

Pair();

Pair(int first, double second);

Pair(const Pair& ob);

int get\_first() { return first; }

double get\_second() { return second; }

void set\_first(int first) { this->first = first; }

void set\_second(double second) { this->second = second; }

Pair operator-(const Pair& pair) const;

Pair operator+(const Pair& pair) const;

Pair operator+(const int data) const;

Pair operator+(const double data) const;

Pair& operator++();

Pair operator++(int);

Pair& operator=(const Pair& pair);

Pair operator/(double n);

bool operator ==(const Pair& pair) const;

bool operator<(const Pair& pair) const;

bool operator>(const Pair& pair) const;

~Pair();

};

Pair.cpp

#include "Pair.h"

Pair::Pair()

{

first = 0;

second = 0;

}

Pair::Pair(int first, double second)

{

this->first = first;

this->second = second;

}

Pair::Pair(const Pair& ob)

{

this->first = ob.first;

this->second = ob.second;

}

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

{

Pair temp = \*this;

temp.first -= pair.first;

temp.second -= pair.second;

return temp;

}

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

{

Pair temp = \*this;

temp.first += pair.first;

temp.second += pair.second;

return temp;

}

Pair Pair::operator+(int data) const

{

Pair temp = \*this;

temp.first += data;

return temp;

}

Pair Pair::operator+(double data) const

{

Pair temp = \*this;

temp.second += data;

return temp;

}

Pair& Pair::operator++()

{

++first;

++second;

return \*this;

}

Pair Pair::operator++(int)

{

Pair temp = \*this;

++first;

++second;

return temp;

}

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

{

if (this == &pair)

return \*this;

first = pair.first;

second = pair.second;

return \*this;

}

Pair Pair::operator/(double n)

{

Pair temp = \*this;

temp.first /= n;

temp.second /= n;

return temp;

}

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

{

if (this->first == pair.first && this->second == pair.second)

return true;

return false;

}

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

{

if ((this->first + this->second) < (pair.first + pair.second))

return true;

return false;

}

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

{

if ((this->first + this->second) > (pair.first + pair.second))

return true;

return false;

}

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

{

out << "(" << ob.first << ":" << ob.second << ")";

return out;

}

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

{

cout << "first(int): "; in >> ob.first;

cout << "second(double): "; in >> ob.second;

return in;

}

Pair::~Pair()

{

}

Ex1.h

#pragma once

#include <list>

#include <algorithm>

#include <ctime>

void fill\_list(list<Pair>& lst)

{

int n;

cout << "Input size: "; cin >> n; cout << endl;

for (size\_t i = 0; i < n; i++)

{

Pair p((rand() % 300 - 50) / 10, (double)(rand() % 300 - 50) / 10);

lst.push\_back(p);

}

}

void show\_list(const list<Pair>& lst)

{

if (!lst.empty())

{

cout << "List: ";

for\_each(lst.begin(), lst.end(), [](const Pair& p) { cout << p << " "; });

cout << endl;

}

}

void add\_average(list<Pair>& lst)

{

if (!lst.empty())

{

Pair temp;

for\_each(lst.begin(), lst.end(), [&temp](const Pair& p) {temp = temp + p; });

lst.push\_back(temp / lst.size());

}

}

void find\_el(list<Pair>& lst)

{

if (!lst.empty())

{

Pair temp;

cout << "\n\t[Input Pair]\n"; cin >> temp;

if (find(lst.begin(), lst.end(), temp) != lst.end())

cout << "\nelement: " << temp << " - was found\n";

else

cout << "\nelement: " << temp << " - not found\n";

}

}

void del(list<Pair>& lst)

{

if (!lst.empty())

{

Pair start, end;

cout << "\n\t[Start position]\n"; cin >> start;

cout << "\t[End position]\n"; cin >> end;

auto it = remove\_if(lst.begin(), lst.end(), [&start, &end](const Pair& p) { return (start < p && p < end); });

lst.erase(it, lst.end());

}

}

void add\_min\_max(list<Pair>& lst)

{

if (!lst.empty())

{

Pair itMin = \*min\_element(lst.begin(), lst.end());

Pair itMax = \*max\_element(lst.begin(), lst.end());

for\_each(lst.begin(), lst.end(), [&itMin, &itMax](Pair& p) { p = p + itMin + itMax; });

}

}

void foo1()

{

system("cls");

cout << "\t\t\tEXERCISE 1\n\n";

srand(time(0));

list<Pair> lst;

fill\_list(lst);

cout << "---------------------- [initial list] ----------------------\n";

show\_list(lst);

cout << "\n---------------------- [find\_el from list] ----------------------\n";

find\_el(lst);

add\_average(lst);

cout << "\n--------------------- [list + average] ---------------------\n";

show\_list(lst);

del(lst);

cout << "\n------------------- [list after deletion] ------------------\n";

show\_list(lst);

add\_min\_max(lst);

cout << "\n------------------- [list + (min + max)] -------------------\n";

show\_list(lst);

lst.sort([](const Pair& p, const Pair& p2) { return p > p2; });

cout << "\n------------------- [descending list] -------------------\n";

show\_list(lst);

lst.sort();

cout << "\n------------------- [ascending list] -------------------\n";

show\_list(lst);

cout << endl;

system("pause");

}

Ex2.h

#pragma once

#include <iostream>

#include <queue>

using namespace std;

typedef priority\_queue<Pair> p\_q;

void fill\_list(p\_q& l)

{

int n = 0;

Pair p;

cout << "Input size: "; cin >> n;

for (size\_t i = 0; i < n; i++)

{

p.set\_first((int)(rand() % 500 - 50) / 10);

p.set\_second((double)(rand() % 500 - 50) / 10);

l.emplace(p);

}

cout << endl;

}

void show\_list(p\_q l)

{

if (l.size() != 0)

{

cout << "List: ";

while (!l.empty())

{

cout << l.top() << " ";

l.pop();

}

cout << endl;

}

}

void average(p\_q& l)

{

p\_q temp;

Pair p;

if (l.size() != 0)

{

int av\_first = 0;

double av\_second = 0;

while (!l.empty())

{

p = l.top();

temp.push(p);

av\_first += p.get\_first();

av\_second += p.get\_second();

l.pop();

}

av\_first /= temp.size();

av\_second /= temp.size();

p.set\_first(av\_first);

p.set\_second(av\_second);

l = temp;

l.push(p);

}

}

void remove(p\_q& l)

{

if (l.size() != 0)

{

Pair start, end;

cout << "\n\t[Start position]\n"; cin >> start;

cout << "\t[End position]\n"; cin >> end;

p\_q temp;

Pair p;

while (!l.empty())

{

p = l.top();

if (!((start.get\_first() <= p.get\_first() && start.get\_second() <= p.get\_second()) && (p.get\_first() <= end.get\_first() && p.get\_second() <= end.get\_second())))

temp.push(p);

l.pop();

}

l = temp;

}

}

void add\_min\_max(p\_q& l)

{

if (l.size() != 0)

{

Pair p\_min, p\_max;

Pair p = l.top();

p\_max.set\_first(p.get\_first());

p\_max.set\_second(p.get\_second());

p\_min.set\_first(p.get\_first());

p\_min.set\_second(p.get\_second());

p\_q temp;

while (!l.empty())

{

p = l.top();

temp.push(p);

if ((p\_max.get\_first() + p\_max.get\_second()) < (p.get\_first() + p.get\_second()))

{

p\_max.set\_first(p.get\_first());

p\_max.set\_second(p.get\_second());

}

else if ((p\_min.get\_first() + p\_min.get\_second()) > (p.get\_first() + p.get\_second()))

{

p\_min.set\_first(p.get\_first());

p\_min.set\_second(p.get\_second());

}

l.pop();

}

while (!temp.empty())

{

p = temp.top();

p = p + p\_max.get\_first() + p\_min.get\_first();

p = p + p\_max.get\_second() + p\_min.get\_second();

l.push(p);

temp.pop();

}

}

}

void foo2()

{

system("cls");

cout << "\t\t\tEXERCISE 2\n\n";

srand(time(0));

priority\_queue<Pair> queue;

fill\_list(queue);

cout << "---------------------- [initial priority\_queue] ----------------------\n";

show\_list(queue);

average(queue);

cout << "\n--------------------- [priority\_queue + average] ---------------------\n";

show\_list(queue);

remove(queue);

cout << "\n------------------- [priority\_queue after deletion] ------------------\n";

show\_list(queue);

add\_min\_max(queue);

cout << "\n------------------- [priority\_queue + (min + max)] -------------------\n";

show\_list(queue);

cout << endl;

system("pause");

}

Ex3.h

#pragma once

#include <map>

#include <algorithm>

#include <ctime>

typedef map<int, Pair> m\_p;

istream& operator>>(istream& in, pair<int, Pair>& p)

{

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

cout << "\t[input Pair]\n"; in >> p.second;

return in;

}

ostream& operator<<(ostream& out, const pair<int, Pair>& p)

{

cout << p.first << " " << p.second;

return out;

}

void fill\_map(m\_p& lst)

{

int n;

cout << "Input size: "; cin >> n; cout << endl;

for (size\_t i = 0; i < n; i++)

{

Pair p((rand() % 300 - 50) / 10, (double)(rand() % 300 - 50) / 10);

int m = rand() % (n \* 4);

auto check = lst.emplace(m, p);

if (check.second == false)

--i;

}

}

void show\_map(m\_p& lst, const string& message = "")

{

cout << message;

for\_each(lst.begin(), lst.end(), [](const pair<int, Pair>& p) { cout << p.first << " " << p.second << endl; });

}

int add\_average(m\_p& lst)

{

pair<int, Pair> temp;

for\_each(lst.begin(), lst.end(), [&temp](const pair<int, Pair>& p) { temp.first += p.first; temp.second = temp.second + p.second; });

auto check = lst.emplace(temp.first / lst.size(), temp.second / lst.size());

if (check.second == false)

return -1;

}

void find\_el(m\_p& lst)

{

cout << endl;

pair<int, Pair> p;

cin >> p;

if (find\_if(lst.begin(), lst.end(), [&p](const pair<int, Pair>& p2) { return (p.first == p2.first && p.second == p2.second);}) != lst.end())

cout << "\n-> element: " << p << " - was found!\n";

else

cout << "\n-> element: " << p << " - not found!\n";

}

void foo3()

{

system("cls");

cout << "\t\t\tEXERCISE 3\n\n";

srand(time(0));

map<int, Pair> lst;

string message;

fill\_map(lst);

message = "---------------------- [initial map] ----------------------\n\n";

show\_map(lst, message);

message = "\n--------------------- [map + average] ---------------------\n\n";

show\_map(lst, message);

if (add\_average(lst) == -1)

cout << "\n-> average can't be add for key!\n";

message = "\n---------------------- [find\_el from list] ----------------------\n\n";

show\_map(lst, message);

find\_el(lst);

cout << endl;

system("pause");

}

**Результаты работы программы**

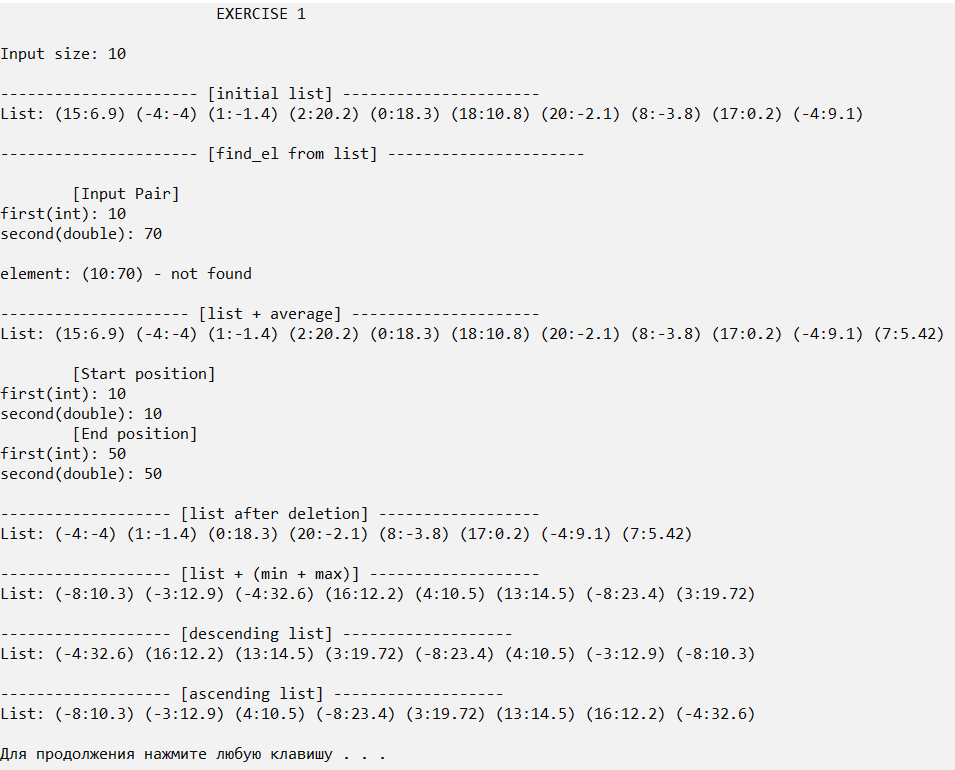


Рисунок 2 – задача 1

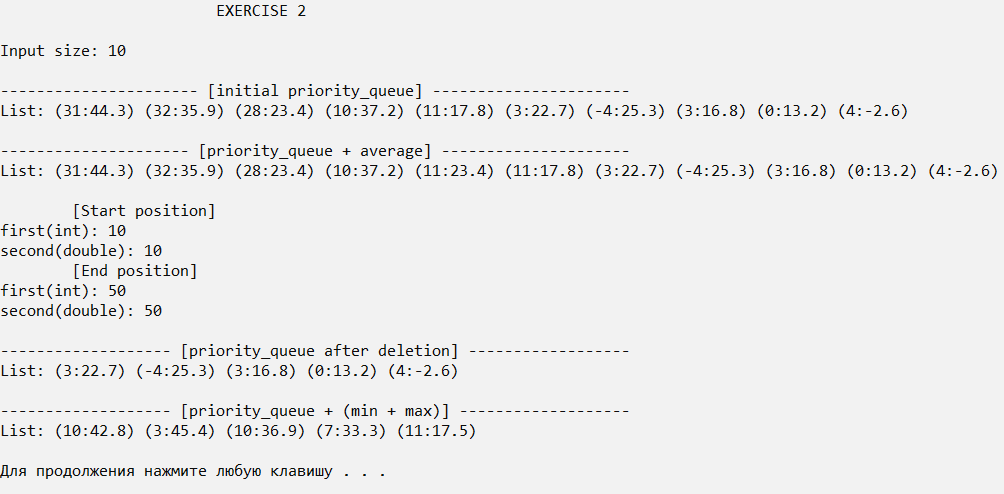


Рисунок 3 – задача 2

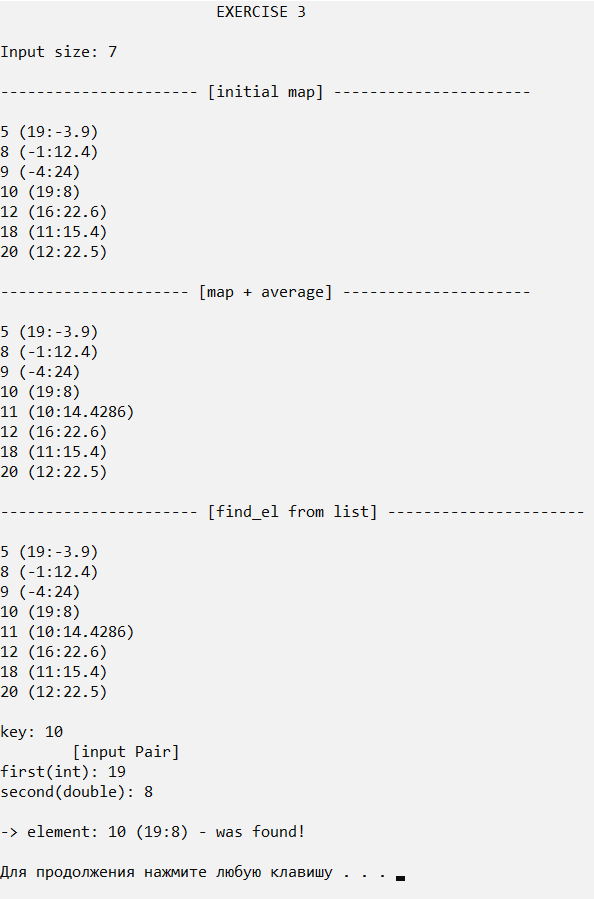


Рисунок 4 – задача 3