Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе № 15.13**

Дисциплина: «Информатика»

Тема : Объектно-ориентированное программирование. Стандартные обобщенные алгоритмы библиотеки STL

Вариант 12

Выполнила:

Студентка группы РИС-22-1б

Черкасова А.А.

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

**Пермь 2023**

**Цель работы**

1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Использование последовательных контейнеров библиотеки STL в ОО программе.

**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.

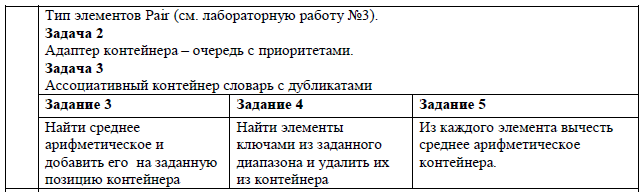
Задача 2.

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.

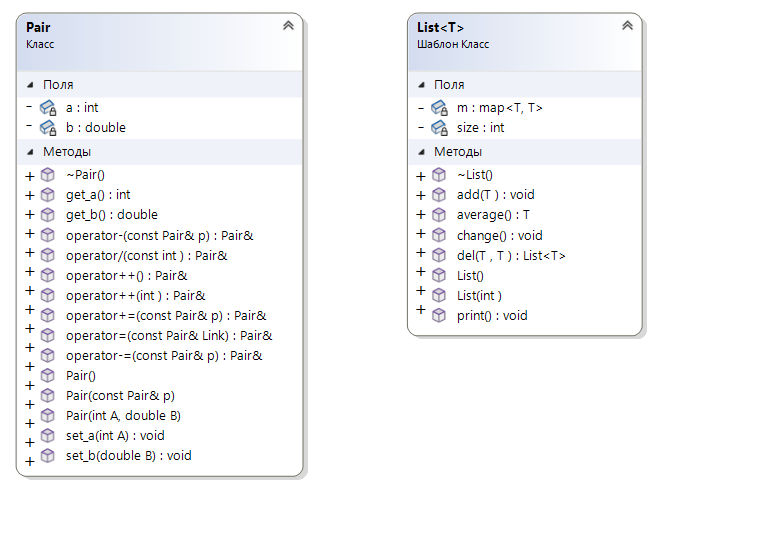




**Анализ задачи**

1. Для решения задачи необходимо:
   1. Реализовать определение функций для работы со списоком типа Pair: создание списка, вычисление среднего арифметического, вычитание среднего арифметического, добавление среднего арифметического на заданную позицию, вывод списка.
   2. Реализовать определение функций для работы с адаптером контейнера priority queue: создание очереди, вычисление среднего арифметического, вычитание среднего арифметического, добавление среднего арифметического на заданную позицию, вывод очереди.
   3. Реализовать определение функций для работы с адаптером контейнера multimap: создание словаря, вычисление среднего арифметического, вычитание среднего арифметического, добавление среднего арифметического на заданную позицию, вывод словаря.
   4. Реализовать применение этих функций в главной функции.
2. Поставленные задачи решены следующими действиями:
   1. В задании 1 реализованы функции для работы с списоком типа Pair: создание списка, вычисление среднего арифметического, вычитание среднего арифметического, добавление среднего арифметического на заданную позицию, вывод списка.
   2. В задании 2 реализованы функции для работы с адаптером контейнера priority queue: создание очереди, вычисление среднего арифметического, вычитание среднего арифметического, добавление среднего арифметического на заданную позицию, вывод очереди.
   3. В задании 3 реализованы функции для работы с адаптером контейнера multimap: создание словаря, вычисление среднего арифметического, вычитание минимального элемента словаря, добавление среднего арифметического в начало словаря, вывод словаря.

**UML – диаграмма**

****

**Реализация задачи на языке С++**

**Заголовочный файл Pair.h**

#pragma once

#include <iostream>

using namespace std;

class Pair

{

private:

int a;

double b;

public:

Pair();

Pair(int A, double B);

Pair(const Pair& p);

~Pair();

int get\_a();

void set\_a(int A);

double get\_b();

void set\_b(double B);

Pair& operator ++(); // Оператор префиксного инкремента

Pair& operator ++(int); // Оператор постфиксного инкремента

Pair& operator = (const Pair& Link); // Оператор присваивания

friend bool operator < (const Pair& p, const Pair& q); // Оператор сравнения

friend bool operator > (const Pair& p, const Pair& q); // Оператор сравнения

friend bool operator ==(const Pair& p, const Pair& q);

friend istream& operator >> (istream& in, Pair& Pointer); // Оператор ввода

friend ostream& operator << (ostream& out, const Pair& Pointer); // Оператор вывода

Pair& operator /(const int);

Pair& operator +=(const Pair& p);

Pair& operator -(const Pair& p);

friend bool operator <= (const Pair& p, const Pair& q); // Оператор сравнения

friend bool operator >= (const Pair& p, const Pair& q);

Pair& operator -=(const Pair& p);

};

**Файл с описанием методов класса Pair.cpp**

#include "Pair.h"

Pair::Pair()

{

a = 0;

b = 0.0;

}

Pair::Pair(int A, double B)

{

a = A;

b = B;

}

Pair::Pair(const Pair& p)

{

a = p.a;

b = p.b;

}

Pair::~Pair()

{

}

int Pair::get\_a()

{

return a;

}

void Pair::set\_a(int A)

{

a = A;

}

double Pair::get\_b()

{

return b;

}

void Pair::set\_b(double B)

{

b = B;

}

Pair& Pair::operator ++() // Оператор префиксного инкремента

{

++a;

return \*this;

}

Pair& Pair::operator ++(int) // Оператор постфиксного инкремента

{

b++;

return \*this;

}

Pair& Pair::operator = (const Pair& Link) // Оператор присваивания

{

if (&Link != this)

{

a = Link.a;

b = Link.b;

}

return \*this;

}

bool operator < (const Pair& p, const Pair& q) // Оператор сравнения

{

if (p.a < q.a)

{

return true;

}

else

{

if (p.a > q.a)

{

return false;

}

else

{

if (p.b < q.b)

{

return true;

}

else

{

return false;

}

}

}

}

bool operator > (const Pair& p, const Pair& q) // Оператор сравнения

{

if (p.a > q.a)

{

return true;

}

else

{

if (p.a < q.a)

{

return false;

}

else

{

if (p.b > q.b)

{

return true;

}

else

{

return false;

}

}

}

}

bool operator <= (const Pair& p, const Pair& q) // Оператор сравнения

{

if (p.a <= q.a)

{

return true;

}

else

{

if (p.a >= q.a)

{

return false;

}

else

{

if (p.b <= q.b)

{

return true;

}

else

{

return false;

}

}

}

}

bool operator >= (const Pair& p, const Pair& q) // Оператор сравнения

{

if (p.a >= q.a)

{

return true;

}

else

{

if (p.a <= q.a)

{

return false;

}

else

{

if (p.b >= q.b)

{

return true;

}

else

{

return false;

}

}

}

}

bool operator == (const Pair& p, const Pair& q) // Оператор сравнения

{

return (p.a == q.a and p.b == q.b);

}

istream& operator >> (istream& in, Pair& Pointer) // Оператор ввода

{

cout << "input a: ";

in >> Pointer.a;

cout << "input b: ";

in >> Pointer.b;

return in;

}

ostream& operator << (ostream& out, const Pair& Pointer) // Оператор вывода

{

cout << "a:b ";

return (out << Pointer.a << " : " << Pointer.b << "\n");

}

Pair& Pair::operator /(const int x)

{

a = a / x;

b = b / x;

return \*this;

}

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

{

this->a += p.a;

this->b += p.b;

return \*this;

}

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

this->a -= p.a;

this->b -= p.b;

return \*this;

}

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

{

this->a -= p.a;

this->b -= p.b;

return \*this;

}

**(1)**

**Файл с главной программой main.cpp**

#include <iostream>

#include <list>

#include <algorithm>

#include <functional>

#include "Pair.h"

using namespace std;

typedef list<Pair> lst;

lst make\_list(int n)

{

lst l;

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

{

Pair a;

cin >> a;

l.push\_back(a);

}

return l;

}

void print\_list(lst& l)

{

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

cout << \*it;

cout << endl;

}

Pair average(lst& l)

{

Pair sum;

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

sum += \*it;

int size = l.size();

return sum / size;

}

void find(const lst& l,const Pair &t)

{

if (find(l.begin(), l.end(), t) != l.end())

cout << "\nelement found\n\n";

else

cout << "\nelement missing\n\n";

}

lst add(lst& l, int pos)

{

Pair elem = average(l);

auto it = l.begin();

advance(it, pos - 1); // -1, т.к учитывается begin

l.insert(it, elem);

return l;

}

void main()

{

srand(time(NULL));

try

{

list <Pair> tmp;

int n;

cout << "enter n: ";

cin >> n;

tmp = make\_list(n);

cout << "\nmake: \n";

print\_list(tmp);

Pair sr = average(tmp);

cout << "\naverage: \n" << sr << endl;

cout << "\nchange: \n";

for\_each(begin(tmp), end(tmp), [&sr](Pair& p) { p -= sr; });

print\_list(tmp);

cout << "\nsort asc: \n";

tmp.sort();

print\_list(tmp);

cout << "\nsort desc: \n";

tmp.sort([](Pair a, Pair b) {return a > b;});

print\_list(tmp);

Pair l, r;

cout << "\nenter range: \n";

cin >> l >> r;

auto it = remove\_if(tmp.begin(), tmp.end(), [&l, &r](const Pair& p) { return (l <= p and p <= r); });

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

cout << "\ndel: \n";

print\_list(tmp);

Pair t;

cout << "\nenter pair: \n";

cin >> t;

find(tmp, t);

int pos;

cout << "\nenter pos: ";

cin >> pos;

tmp = add(tmp, pos);

cout << "\nadd: \n";

print\_list(tmp);

}

catch (int)

{

cout << "\nError!\n";

}

}

**(2)**

**Файл с главной программой main.cpp**

#include <iostream>

#include "Pair.h"

#include <queue>

using namespace std;

// практически все функции остаются неизменными, если работать непосредственно с адаптером контейнера - очередь с приоритетом

// можно из очереди перенести данные в другие контейнеры, поработать с ними и вернуть в очередь [как по мне бессмысленно]

//очередь с приоритетом уже является отсортированной по убыванию

typedef priority\_queue <Pair> pq;

pq make(int n)

{

pq tmp;

Pair el;

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

{

cin >> el;

tmp.push(el);

}

return tmp;

}

void print(pq t)

{

while (!t.empty())

{

cout << t.top();

t.pop();

}

cout << endl;

}

Pair average(pq t)

{

Pair sum;

int size = t.size();

while (!t.empty())

{

sum += t.top();

t.pop();

}

return sum / size;

}

void add(pq& t) //в данном случае, не получиться добавить в указанную позицию. Добавление по приоритету

{

Pair el = average(t);

t.push(el);

}

void del(pq& t, Pair l, Pair r)

{

pq tmp;

while (!t.empty())

{

Pair el = t.top();

if (el < l or el > r) tmp.push(el);

t.pop();

}

swap(tmp, t);

}

void change(pq& t)

{

Pair el = average(t);

pq tmp;

while (!t.empty())

{

Pair elem = t.top();

Pair em = elem - el;

tmp.push(em);

t.pop();

}

swap(tmp, t);

}

void sort\_asc(pq t)

{

priority\_queue < Pair, vector<Pair>, greater<Pair>> que;

while (!t.empty())

{

Pair el = t.top();

que.push(el);

t.pop();

}

while (!que.empty()) // вывод

{

cout << que.top();

que.pop();

}

cout << endl;

}

void find\_el(pq t, Pair a)

{

vector<Pair> que;

while (!t.empty())

{

Pair el = t.top();

que.push\_back(el);

t.pop();

}

auto it = find\_if(que.begin(), que.end(), [&](const Pair& i) {

return i == a;});

if (it != que.end())

cout << "\nelement found\n\n";

else

cout << "\nelement missing\n\n";

}

int main()

{

try

{

int n;

cout << "enter n: ";

cin >> n;

pq tmp;

tmp = make(n);

cout << "\nmake: \n";

print(tmp);

Pair sr = average(tmp);

cout << "\naverage: \n" << sr << endl;

cout << "\nchange: \n";

change(tmp);

print(tmp);

Pair l, r;

cout << "\nenter range: \n";

cin >> l >> r;

del(tmp, l, r);

cout << "\ndel: \n";

print(tmp);

cout << "\nsort asc: \n";

sort\_asc(tmp);

cout << "\nsort desc: \n";

print(tmp);

add(tmp);

cout << "\nadd: " << endl;

print(tmp);

Pair t;

cout << "\nenter pair: \n";

cin >> t;

find\_el(tmp, t);

}

catch (int)

{

cout << "\nError!\n";

}

}

**(3)**

**Файл с главной программой main.cpp**

#include <iostream>

#include <map>

#include <algorithm>

#include <functional>

#include "Pair.h"

using namespace std;

typedef multimap<Pair, Pair> mmp;

typedef pair<Pair, Pair> pr;

mmp make\_list(int n)

{

mmp l;

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

{

Pair key, val;

cout << "\nEnter key: \n";

cin >> key;

cout << "\nEnter value: \n";

cin >> val;

l.emplace(key, val);

}

return l;

}

void print(mmp& l)

{

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

{

cout << "\nkey - value\n";

cout << it->first;

cout << it->second;

}

cout << endl;

}

pr average(mmp& l)

{

Pair sum\_key;

Pair sum\_val;

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

{

sum\_key += it->first;

sum\_val += it->second;

}

int size = l.size();

pr p(sum\_key / size, sum\_val / size);

return p;

}

void change(const pr &sr, mmp& l)

{

mmp tmp;

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

{

tmp.emplace(it->first - sr.first, it->second - sr.second);

}

swap(l, tmp);

}

void sort\_desc(mmp t)

{

multimap<Pair, Pair, greater<Pair>> p;

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

p.emplace(it->first, it->second);

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

{

cout << "\nkey - value\n";

cout << it->first;

cout << it->second;

}

cout << endl;

}

void find(const mmp& l, const pr& t)

{

if (find\_if(l.begin(), l.end(), [&t](const pr& p) { return (t.first == p.first and t.second == p.second); }) != l.end())

cout << "\nelement found\n\n";

else

cout << "\nelement missing\n\n";

}

void add(mmp& l, const pr& sr) // вставка невозможна по заданной позиции, т.к. контейнер отсортирован

{

l.emplace(sr.first, sr.second);

}

void del(mmp& t, const Pair& l, const Pair& r)

{

auto it = remove\_if(t.begin(), t.end(), [&l, &r](const pr& t) { return (l <= t.first and t.first <= r); });

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

}

int main()

{

try

{

mmp tmp;

int n;

cout << "enter n: ";

cin >> n;

tmp = make\_list(n);

cout << "\nmake: \n";

print(tmp);

pr sr = average(tmp);

cout << "\naverage: \n";

cout << sr.first;

cout << sr.second;

cout << "\nchange: \n";

//for\_each(tmp.begin(), tmp.end(), [&sr](pr &t) { t.first -= sr.first; t.second -= sr.second; }); // не работает!

change(sr, tmp);

print(tmp);

cout << "\nsort asc: \n";

print(tmp);

cout << "\nsort desc: \n";

sort\_desc(tmp);

pr t;

cout << "\nenter pair: \n";

cout << "\nEnter key: \n";

cin >> t.first;

cout << "\nEnter value: \n";

cin >> t.second;

find(tmp, t);

Pair l, r;

cout << "\nenter range: \n";

cin >> l >> r;

del(tmp, l, r);

cout << "\ndel: \n";

print(tmp);

}

catch (int)

{

cout << "\nError!\n";

}

}