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 № 15.12**

Дисциплина: «Информатика»

Тема : Объектно-ориентированное программирование. Ассоциативные контейнеры 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.

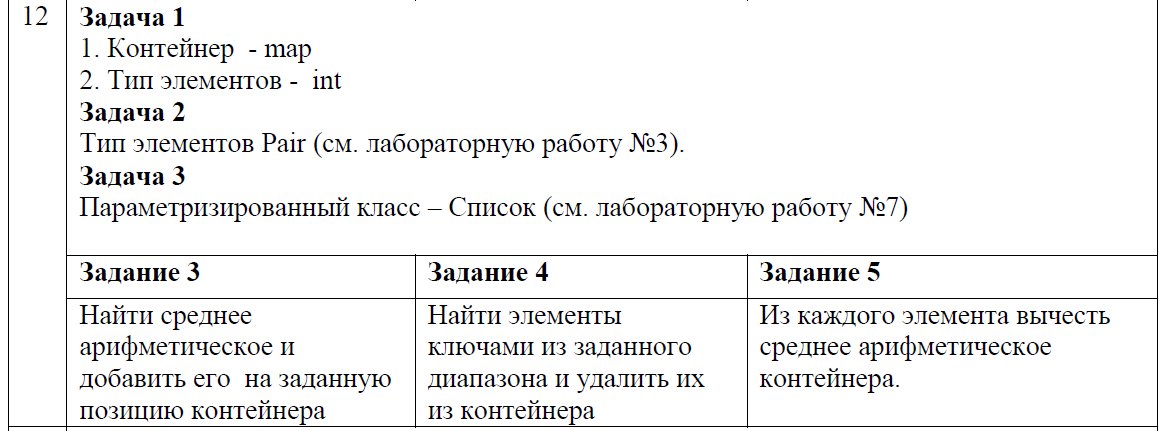
1. Создать ассоциативный контейнер.
2. Заполнить его элементами стандартного типа (тип указан в варианте).
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде глобальных функций.

Задача 2.

1. Создать ассоциативный контейнер.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде глобальных функций.

Задача 3

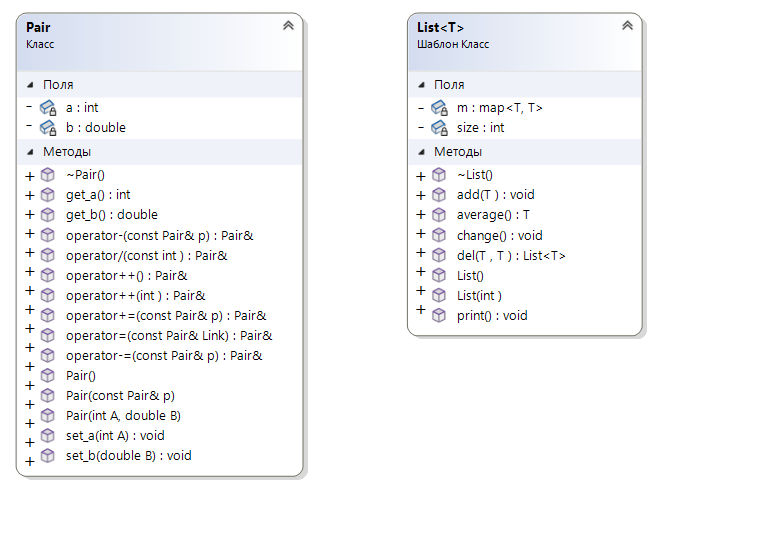
1. Создать параметризированный класс, используя в качестве контейнера ассоциативный контейнер.
2. Заполнить его элементами.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде методов параметризированного класса.

****

**Анализ задачи**

1. Для решения задачи необходимо:
   1. Реализовать определение функций для работы с словарь с дубликатами типа int: создание множества, вычисление среднего арифметического, вычитание среднего арифметического, добавление среднего арифметического в словарь, вывод словаря.
   2. Реализовать определение функций для работы с словарь с дубликатами типа Pair: создание множества, вычисление среднего арифметического, вычитание среднего арифметического, добавление среднего арифметического в словарь, вывод словаря..
   3. Реализовать определение параметризированного класса Container и методов данного класса.
   4. Реализовать применение этих функций в главной функции.
2. Поставленные задачи решены следующими действиями:
   1. В задании 1 реализованы функции для работы с словарь с дубликатами типа Pair: создание множества, вычисление среднего арифметического, вычитание среднего арифметического, добавление среднего арифметического в словарь, вывод словаря.
   2. В задании реализованы функции для работы с словарь с дубликатами типа Pair: создание множества, вычисление среднего арифметического, вычитание среднего арифметического, добавление среднего арифметического в словарь, вывод словаря.
   3. В задании находится реализация параметризированного списка и методов: создание множества, вычисление среднего арифметического, вычитание среднего арифметического, добавление среднего арифметического в словарь, 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 <map>

using namespace std;

typedef map <int, int> Tmap;

Tmap make(int n)

{

Tmap tmp;

int el;

for (int i = 1; i <= n; i++)

{

cout << "enter elem: ";

cin >> el;

tmp.emplace(i, el);

}

return tmp;

}

void print(Tmap& t)

{

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

cout << it->first << " \_ " << it->second << "\n";

cout << endl;

}

int average(Tmap& t)

{

int sum = 0;

int size = 0;

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

{

sum += it->second;

size++;

}

return sum / size;

}

void add(Tmap& t, int pos)

{

int elem = average(t);

if (t.count(pos))

{

cout << "adding with this key is not possible" << endl; // ключ должен быть уникальный

return;

}

else

{

t.insert(make\_pair(pos, elem));

}

}

Tmap del(Tmap& t, int l, int r)

{

Tmap tmp = t;

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

{

if (it->first >= l and it->first <= r)

tmp.erase(it->first);

}

return tmp;

}

void change(Tmap& t)

{

int el = average(t);

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

it->second -= el;

}

int main()

{

int n;

Tmap ex;

cout << "enter n: ";

cin >> n;

ex = make(n);

print(ex);

int pos;

cout << "enter pos: ";

cin >> pos;

add(ex, pos);

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

print(ex);

change(ex);

cout << "change: " << endl;

print(ex);

int l, r;

cout << "enter range: ";

cin >> l >> r;

ex = del(ex, l, r);

cout << "del: " << endl;

print(ex);

}

**(2)**

**Файл с главной программой main.cpp**

#include <iostream>

#include <map>

#include "Pair.h"

using namespace std;

typedef map <Pair, Pair> Tmap;

Tmap make(int n)

{

Tmap tmp;

Pair el, k;

for (int i = 1; i <= n; i++)

{

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

cin >> k;

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

cin >> el;

tmp.emplace(k, el);

}

return tmp;

}

void print(Tmap& t)

{

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

cout << "\nkey - value\n" << it->first << it->second << "\n";

cout << endl;

}

Pair average(Tmap& t)

{

Pair sum;

int size = 0;

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

{

sum += it->second;

size++;

}

return sum / size;

}

void add(Tmap& t, Pair pos)

{

Pair elem = average(t);

if (t.count(pos))

{

cout << "adding with this key is not possible" << endl; // ключ должен быть уникальный

return;

}

else

{

t.insert(make\_pair(pos, elem));

}

}

Tmap del(Tmap& t, Pair l, Pair r)

{

Tmap tmp = t;

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

{

if (it->first >= l and it->first <= r)

tmp.erase(it->first);

}

return tmp;

}

void change(Tmap& t)

{

Pair el = average(t);

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

it->second -= el;

}

int main()

{

int n;

Tmap ex;

cout << "enter n: ";

cin >> n;

ex = make(n);

print(ex);

Pair pos;

cout << "enter pos: \n";

cin >> pos;

add(ex, pos);

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

print(ex);

change(ex);

cout << "change: " << endl;

print(ex);

Pair l, r;

cout << "enter range: \n";

cout << "left: \n";

cin >> l;

cout << "right: \n";

cin >> r;

ex = del(ex, l, r);

cout << "\ndel: " << endl;

print(ex);

}

**(3)**

**Заголовочный файл List.h**

#pragma once

#include <iostream>

#include <list>

#include <map>

using namespace std;

template <class T>

class List

{

private:

map <T, T> m;

int size;

public:

List();

List(int); // Конструктор с параметрами: выделение под s элементов и заполнение их значением k

~List(); // Деструктор

T average();

void add(T);

List <T> del(T, T);

void change();

void print();

};

template <class T>

List<T>::List() { size = 0; }

template <class T>

List<T>::List(int n)

{

T el, k;

for (int i = 1; i <= n; i++)

{

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

cin >> k;

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

cin >> el;

m.emplace(k, el);

}

size = m.size();

}

template <class T>

List<T>::~List() {}

template <class T>

void List<T>::print()

{

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

cout << "\nkey - value\n" << it->first << it->second << "\n";

cout << endl;

}

template <class T>

T List<T>::average()

{

T sum;

size = 0;

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

{

sum += it->second;

size++;

}

return sum / size;

}

template <class T>

void List<T>::add(T pos)

{

T elem = average();

if (m.count(pos))

{

cout << "adding with this key is not possible" << endl; // ключ должен быть уникальный

return;

}

else

{

m.insert(make\_pair(pos, elem));

}

}

template <class T>

List<T> List<T>::del(T l, T r)

{

map <T, T> tmp = m;

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

{

if (it->first >= l and it->first <= r)

tmp.erase(it->first);

}

m = tmp;

return \*this;

}

template <class T>

void List<T>::change()

{

T el = average();

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

it->second -= el;

}

**Файл с главной программой main.cpp**

#include <iostream>

#include <ctime>

#include "Pair.h"

#include "List.h"

using namespace std;

int main()

{

try

{

int n;

cout << "enter n: ";

cin >> n;

List <Pair> tmp(n);

List <Pair> tmp2;

cout << "\nmake: " << endl;

tmp.print();

Pair pos;

cout << "enter pos: \n";

cin >> pos;

tmp.add(pos);

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

tmp.print();

tmp.change();

cout << "\nchange: " << endl;

tmp.print();

Pair l, r;

cout << "left: \n";

cin >> l;

cout << "right: \n";

cin >> r;

tmp = tmp.del(l, r);

cout << "\ndel: " << endl;

tmp.print();

}

catch (int)

{

cout << "\nError!\n";

}

}

**Ответы на вопросы**

1. Что представляет собой ассоциативный контейнер?

Ассоциативный массив содержит пары значений. Зная одно значение,

называемое ключом (key), мы можем получить доступ к другому, называемому отображаемым значением (mapped value).

Ассоциативный массив можно представить как массив, для которого индекс не обязательно должен иметь целочисленный тип.

2. Перечислить ассоциативные контейнеры библиотеки STL.

map - ассоциативный массив, по ключу в контейнере хранится одно значение

multimap - ассоциативный массив с повторяющимися ключами

set - массив уникальных ключей без значений

multiset - массив с повторяющимися ключами без значений.

3. Каким образом можно получить доступ к элементам ассоциативного контейнера?

V& operator[](const K&) возвращает ссылку на элемент V, соответствующий значению K.

4. Привести примеры методов, используемых в ассоциативных контейнерах.

bool empty() const, size\_type size() const, size\_type max\_size(), insert(), erase(), clear(), swap(), key\_comp(), value\_comp(), find(), count(), lower\_bound(), upper\_bound().

5. Каким образом можно создать контейнер map? Привести примеры.

#include <iostream>

#include <map>

using namespace std;

int main() {

map<int, float> m;//словарь\

int n;//количество элементов

cout << "n: ";

cin >> n;

float a;

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

cout <<" ?";

cin >> a; //создаем пару и добавляем ее в словарь

m.insert(make\_pair(i, a)); }

return 0;

}

6. Каким образом упорядочены элементы в контейнере map по умолчанию? Как изменить порядок на обратный?

Ассоциативный контейнер map требует, чтобы для типов ключа существовала операция “<”, то есть, элементы хранятся в порядке возрастания. Он хранит свои элементы отсортированными по ключу так, что перебор происходит по порядку.

Спецификация шаблона для класса map:

template <class Key, class T, class Comp = less <Key>, class Allocator = allocator <pair> >

class Comp = less <Key> - параметр, определяющий критерий

упорядочения, по умолчанию less (по возрастанию ключа).

Чтобы изменить критерий упорядочивания, нужно изменить параметр Comp.

7. Какие операции определены для контейнера map?

Определена операция присваивания: map& operator=(const map&);

Определены следующие операции: ==, <, <=, !=, >, >= и операция индексации ([]).

8. Написать функцию для добавления элементов в контейнер map с помощью функции make\_pair().

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

cout <<" ?";

cin >> a;

m.insert(make\_pair(i, a));

}

9. Написать функцию для добавления элементов в контейнер map с помощью функции операции прямого доступа [].

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

cout <<" ?";

cin >> a;

m[i] = a;

}

10. Написать функцию для печати контейнера map с помощью итератора.

for (map<int, float>::iterator i = m.begin(); i != m.end(); ++i) {

cout << (\*i).first << " " << (\*i).second << endl;

}

11. Написать функцию для печати контейнера map с помощью функции операции прямого доступа [].

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

cout << m[i] << endl;

}

12. Чем отличаются контейнеры map и multimap?

Словари с дубликатами (multimap) допускают хранение элементов с одинаковыми ключами. Поэтому для них не определена операция доступа по индексу. Элементы с одинаковыми ключами хранятся в словаре в порядке их занесения. При удалении по ключу функция erase возвращает количество удаленных элементов. В остальном они аналогичны обычным словарям.

13. Что представляет собой контейнер set?

set — это контейнер, который автоматически сортирует добавляемые элементы в порядке возрастания. Но при добавлении одинаковых значений, set будет хранить только один его экземпляр. По другому его еще называют словарьм.

14. Чем отличаются контейнеры map и set?

Тип set позволяет хранить уникальные объекты различных типов, эффективно добавлять, удалять объекты и выполнять поиск. Тип map позволяет хранить пары ключ-значение, причем ключи должны быть уникальными.

15. Каким образом можно создать контейнер set? Привести примеры.

set<int> set1; // создается пустое словарь

int а[5] = { 1. 2. 3. 4, 5};

set<int> set2(a, а + 5);// инициализация копированием

set<int> set3(set2); // инициализация другим словарьм

16. Каким образом упорядочены элементы в контейнере set по умолчанию? Как изменить порядок на обратный?

Словарь, как и словарь, требует, чтобы для типа T существовала операция “меньше” (<). Оно хранит свои элементы отсортированными, так что перебор происходит по порядку.

В множестве хранятся объекты, упорядоченные по некоторому ключу, являющемуся атрибутом самого объекта. Например, словарь может хранить объекты класса Person, упорядоченные в алфавитном порядке по значению ключевого поля name. Если в множестве хранятся значения одного из встроенных типов, например int, то ключом является сам элемент.

set<int, greater<int>> set - изменение критерия упорядочения путем спецификации параметра компаратора

17. Какие операции определены для контейнера set?

Для вставки элементов в словарь можно использовать метод insert(), для

удаления — метод erase(). Также к множествам применимы общие для всех контейнеров методы. Во всех ассоциативных контейнерах есть метод count(), возвращающий количество объектов с заданным ключом.

18. Написать функцию для добавления элементов в контейнер set.

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

cout <<" ? ";

cin >> a;

s.insert(a);

}

19. Написать функцию для печати контейнера set.

for(auto it = s.begin(); it != s.end(); ++i) cout<<\*i<<” ”;

20. Чем отличаются контейнеры set и multiset?

В множествах с дубликатами ключи могут повторяться. Элементы с одинаковыми ключами хранятся в множестве в порядке их занесения. Функция find() возвращает значение первого найденного элемента или end(), если ни одного элемента с заданным ключом не