Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по лабораторной работе №12

Тема: «Ассоциативные контейнеры библиотеки STL.»

Выполнили:

Студенты группы РИС-22-2б

Третьяков Н.А.

Проверил доц. Кафедры ИТАС:

Полякова Ольга Андреевна

Пермь 2023

# 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.

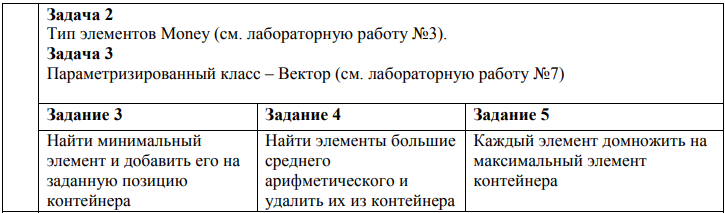
Задача 2.

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.





**Контрольные вопросы**

1. Что представляет собой ассоциативный контейнер?

Ассоциативный контейнер - это контейнер, который содержит упорядоченный набор элементов, представленных в форме пар ключ-значение. Элементы располагаются в контейнере в соответствии с заданным отношением порядка, определенным в компараторе. Ключи уникальны в контейнере, т.е. в контейнере не может быть двух элементов с одинаковым ключом. Основным преимуществом ассоциативных контейнеров является возможность быстрого поиска элемента по ключу благодаря использованию бинарного дерева поиска (например, красно-черного дерева) для хранения элементов. Примерами ассоциативных контейнеров в STL являются map и set.

1. Перечислить ассоциативные контейнеры в библиотеке STL.

* set - контейнер, хранящий уникальные элементы в отсортированном порядке. Все элементы в set уникальны, то есть не могут дублироваться.
* map - контейнер, хранящий пары ключ-значение, отсортированные по ключу. Ключи в map должны быть уникальными.
* multiset - контейнер, хранящий уникальные элементы в отсортированном порядке, но допускающий дублирование элементов.
* multimap - контейнер, хранящий пары ключ-значение, отсортированные по

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

Для доступа к элементам ассоциативного контейнера в STL используются итераторы. В отличие от последовательных контейнеров, в ассоциативных контейнерах элементы хранятся не в порядке их добавления, а в отсортированном порядке на основе ключа. Поэтому для доступа к элементам по индексу, как в последовательных контейнерах, в ассоциативных контейнерах используют итераторы. С помощью итераторов можно получить доступ к ключу и соответствующему ему значению элемента контейнера.

1. Привести примеры методов, используемых в ассоциативных контейнерах.

* insert(): добавляет элемент в контейнер.
* erase(): удаляет элемент из контейнера.
* find(): ищет элемент по заданному ключу.
* count(): возвращает количество элементов с заданным ключом в контейнере.
* size(): возвращает количество элементов в контейнере.
* empty(): возвращает значение true, если контейнер пуст, и false в противном случае.
* begin(): возвращает итератор, указывающий на первый элемент в контейнере.
* end(): возвращает итератор, указывающий на элемент следующий за последним \* элементом контейнера.
* lower\_bound(): возвращает итератор на первый элемент в контейнере, не меньший \* заданного ключа.
* upper\_bound(): возвращает итератор на первый элемент в контейнере, больший заданного ключа.
* equal\_range(): возвращает диапазон элементов в контейнере, соответствующих заданному ключу.

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

* С помощью конструктора по умолчанию:

map<string, int> myMap;

В данном случае создается пустой контейнер map с ключами типа string и значениями типа int.

* С помощью списка инициализации:

map<string, int> myMap = {{"apple", 1}, {"banana", 2}, {"cherry", 3}};

В данном случае создается контейнер map с начальными значениями ключей и значений, которые передаются в список инициализации.

* С помощью пары итераторов:

map<std::string, int> myMap(anotherMap.begin(), anotherMap.end());

В данном случае создается контейнер map, который инициализируется парами ключ-значение из другого контейнера, заданного итераторами begin() и end().

* С помощью списка пар ключ-значение:

map<std::string, int> myMap = {make\_pair("apple", 1), make\_pair("banana", 2),make\_pair("cherry", 3)};

В данном случае создается контейнер map, который инициализируется парами ключ-значение из списка пар, созданных с помощью функции make\_pair().

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

Элементы отсортированы в порядке возрастания ключей.

Чтобы изменить порядок, можно определить пользовательскую функцию сравнения, которая будет сравнивать ключи в обратном порядке. Например:

bool compare(int a, int b) {

return a > b;

}

Затем, мы можем создать map следующим образом:

std::map<int, std::string, decltype(compare)\*> myMap(compare);

1. Какие операции определены для контейнера map?

Контейнер map поддерживает операции добавления и удаления элементов, поиска и доступа к элементам по ключу, а также проверки наличия элементов в контейнере. Кроме того, контейнер map поддерживает итераторы для обхода содержимого.

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

template <class T1, class T2>

void addElements(std::map<T1, T2>& m, int n) {

T1 temp1 = 0;

T2 temp2;

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

{

cin >> temp2;

m.insert(make\_pair(temp1++, temp2));

}

}

1. Написать функцию для добавления элементов в контейнер map с помощь функции операци прямого доступпа [].

template <class T1, class T2>

void addElements(std::map<T1, T2>& m, T1 key, T2 value) {

m[key] = value;

}

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

template <class T1, class T2>

void printMap(map<T1, T2>& m) {

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

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

}

}

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

template <class T1, class T2>

void printMap(map<T1, T2>& m) {

for (const auto& p : m)

cout << p.first << " : " << m[p.first] << endl;

}

1. Чем отличаются контейнеры map и multimap?

Отличие между map и multimap заключается в том, что map хранит только уникальные ключи и соответствующие значения, тогда как multimap может хранить несколько значений для одного и того же ключа. Другими словами, map — это контейнер с уникальными ключами, а multimap - контейнер с неуникальными ключами.

1. Что представляет собой контейнер set?

Контейнер set - упорядоченное множество уникальных элементов. Он реализован в виде бинарного дерева поиска и обеспечивает быстрый доступ, вставку и удаление элементов в отсортированном порядке.

1. Чем отличаются контейнеры map и set?

Контейнер map предназначен для хранения пары "ключ-значение", где каждый ключ уникален, а контейнер set используется для хранения уникальных элементов, без пары "ключ-значение".

Таким образом, map используется для хранения и доступа к значению по ключу, а set используется для хранения элементов в отсортированном порядке и быстрого поиска элементов по значению.

1. Каким ообразом можно создать контейнер set? Привести примеры.

* Создание пустого контейнера с помощью конструктора по умолчанию:

set<int> mySet;

* Создание с заданными начальными значениями с помощью списка инициализации:

set<int> mySet = {1, 2, 3, 4};

* Создание с помощью диапазона значений другого контейнера:

vector<int> myVec = {1, 2, 3, 4};

set<int> mySet(myVec.begin(), myVec.end());

* Создание пустого контейнера с заданным компаратором:

struct Compare {

bool operator()(int a, int b) const {

return a > b;

}

};

set<int, Compare> mySet(Compare());

* Создание с заданными начальными значениями и компаратором:

struct Compare {

bool operator()(int a, int b) const {

return a > b;

}

};

set<int, Compare> mySet = {1, 2, 3, 4};

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

Элементы в контейнере set упорядочены по возрастанию. Чтобы изменить порядок на убывание, можно задать компаратор при создании контейнера, который будет сравнивать элементы в обратном порядке. Например:

#include <functional> // для std::greater

int main() {

set<int,greater<int>> s {5, 2, 7, 1, 8};

// элементы будут упорядочены в порядке убывания

return 0;

}

Здесь std::greater<int> - это функциональный объект, который сравнивает элементы в порядке убывания. Он передается вторым параметром шаблона контейнера set.

1. Какие методы определены для контейнера set?

* insert() - добавляет элемент в контейнер
* erase() - удаляет элемент из контейнера по значению или по итератору
* find() - ищет элемент в контейнере и возвращает итератор на найденный элемент, либо итератор на конец контейнера, если элемент не найден
* size() - возвращает количество элементов в контейнере
* empty() - возвращает true, если контейнер пуст, иначе – false
* clear() - удаляет все элементы из контейнера

1. Написать функцию для добавления элементов в контейнер set.

template <class T1>

void addElements(set<T1>& st, int n) {

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

st.insert(T1(rand()));

}

1. Написать функцию для печати контейнера set.

template <class T1>

void printSet(set<T1>& st) {

for (const auto& i : st)

cout << i << endl;

}

1. Чем отличается контейнер set и multiset?

Отличие между ними заключается в том, что set может хранить только уникальные элементы, а multiset может хранить несколько одинаковых элементов.

**Код Программы**

**Задача 1**

#include <iostream>

#include <map>

using namespace std;

typedef multimap<int, float> Mul;

typedef multimap<int, float> ::iterator it;

Mul Mmap(int n)

{

Mul m;

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

{

m.insert(make\_pair(i, rand() % 100));

}

return m;

}

void Print(Mul m)

{

it iter;

for (iter = m.begin(); iter != m.end(); iter++)

{

cout << iter->second << endl;

}

cout << "--------------------------------------" << endl;

}

Mul MapM(Mul m)

{

it iter;

float mn = 100;

for (iter = m.begin(); iter != m.end(); iter++)

{

if (mn > iter->second)

mn = iter->second;

}

cout << "Enter the number to put the item on" << endl;

int n;

cin >> n;

m.emplace(n, mn);

return m;

}

Mul MapS(Mul m)

{

it iter;

Mul m1;

float s = 0, ms;

int n, n1 = 0;

for (iter = m.begin(); iter != m.end(); iter++)

{

s += iter->second;

n = iter->first + 1;

}

s = s / n;

for (iter = m.begin(); iter != m.end(); iter++)

{

if (s > iter->second)

{

ms = iter->second;

m1.insert(make\_pair(n1, ms));

n1++;

}

}

return m1;

}

Mul MapMax(Mul m)

{

it iter;

float mx = 0;

for (iter = m.begin(); iter != m.end(); iter++)

{

if (mx < iter->second)

mx = iter->second;

}

for (iter = m.begin(); iter != m.end(); iter++)

{

iter->second = iter->second \* mx;

}

return m;

}

int main()

{

Mul m;

cout << "Enter the size of the container" << endl;

int n;

cin >> n;

m = Mmap(n);

Print(m);

m = MapM(m);

Print(m);

m = MapS(m);

Print(m);

m = MapMax(m);

Print(m);

}

**Задача 2**

**Money.h**

#pragma once

#include <iostream>

using namespace std;

class Money

{

long rubles;

int kopecks;

public:

Money();

Money(long r, int k);

Money(const Money&);

long GetMoneyR();

int GetMoneyK();

void SetMoney(long r, int k);

Money& operator = (const Money&);

Money& operator \* (const Money&);

Money& operator --();

Money& operator --(int);

friend std::ostream& operator << (std::ostream& out, const Money&);

friend std::istream& operator >> (std::istream& in, Money&);

bool operator == (const Money&);

bool operator != (const Money&);

bool operator < (const Money&);

~Money() {};

};

**Money.cpp**

#include "Money.h"

#include <iostream>

using namespace std;

Money::Money()

{

rubles = 0;

kopecks = 0;

}

Money::Money(long r, int k)

{

rubles = r;

if (k < 100)

{

kopecks = k;

}

else

{

rubles = rubles + k / 100;

kopecks = k % 100;

}

}

Money::Money(const Money& m)

{

rubles = m.rubles;

kopecks = m.kopecks;

}

long Money::GetMoneyR()

{

return rubles;

}

int Money::GetMoneyK()

{

return kopecks;

}

void Money::SetMoney(long r, int k)

{

rubles = r;

kopecks = k;

}

Money& Money::operator = (const Money& m)

{

rubles = m.rubles;

kopecks = m.kopecks;

return \*this;

}

Money& Money::operator \* (const Money& m)

{

int M;

M = (rubles \* 100 + kopecks) \* (m.rubles \* 100 + m.kopecks);

rubles = M / 100;

kopecks = M % 100;

return \*this;

}

Money& Money::operator -- ()

{

kopecks = kopecks - 1;

return \*this;

}

Money& Money::operator--(int)

{

kopecks = kopecks - 1;

return \*this;

}

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

{

return(out << p.rubles << ", " << p.kopecks);

}

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

{

in >> p.rubles;

in >> p.kopecks;

return in;

}

bool Money::operator!=(const Money& m)

{

if (this->rubles != m.rubles)

{

return 1;

}

else

{

if (this->kopecks != m.kopecks)

{

return 1;

}

else

{

return 0;

}

}

}

bool Money::operator==(const Money& m)

{

if (this->rubles == m.rubles)

{

if (this->kopecks == m.kopecks)

{

return 1;

}

else

{

return 0;

}

}

else

{

return 0;

}

}

bool Money:: operator <(const Money& m)

{

if (this->rubles < m.rubles)

{

return 1;

}

else

{

if (this->rubles == m.rubles)

{

if (this->kopecks < m.kopecks)

return 1;

else

return 0;

}

else

return 0;

}

}

**Main.cpp**

#include <iostream>

#include <map>

#include "Money.h"

using namespace std;

typedef multimap<int, Money> Mult;

typedef multimap<int, Money> :: iterator it;

Mult Mmap(int n)

{

Mult m;

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

{

Money M(rand() % 100, rand() % 100);

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

}

return m;

}

void Print(Mult m)

{

it iter;

for (iter = m.begin(); iter != m.end(); iter++)

{

Money M;

M = iter->second;

cout << M << endl;

}

cout << "-------------------------------------------------" << endl;

}

Mult MapM(Mult m)

{

Money mn(100, 99);

it iter;

for (iter = m.begin(); iter != m.end(); iter++)

{

if (iter->second < mn)

mn = iter->second;

}

cout << "Enter the number to put the item on" << endl;

int n;

cin >> n;

m.emplace(n, mn);

return m;

}

Mult MapS(Mult m)

{

it iter;

Mult m1;

int s = 0;

int n = 0, n1 = 0;

for (iter = m.begin(); iter != m.end(); iter++)

{

s = s + iter->second.GetMoneyR() \* 100 + iter->second.GetMoneyK();

n = iter->first + 1;

}

s = s / n;

Money M(s / 100, s & 100);

for (iter = m.begin(); iter != m.end(); iter++)

{

if (iter->second < M)

{

m1.insert(make\_pair(n1, iter->second));

n1++;

}

}

return m1;

}

Mult MapMax(Mult m)

{

int M;

it iter;

Money mx(0, 0);

for (iter = m.begin(); iter != m.end(); iter++)

{

if (mx < iter->second)

mx = iter->second;

}

for (iter = m.begin(); iter != m.end(); iter++)

{

iter->second = iter->second \* mx;

}

return m;

}

int main()

{

Mult m;

cout << "Enter the size of the container" << endl;

int n;

cin >> n;

m = Mmap(n);

Print(m);

m = MapM(m);

Print(m);

m = MapS(m);

Print(m);

m = MapMax(m);

Print(m);

}

**Задача 3**

#include <iostream>

#include <map>

using namespace std;

template <class T>

class Plenty

{

multimap<int, T> plenty;

typename multimap<int, T>::iterator it;

int len;

public:

Plenty();

Plenty(int);

void Print();

Plenty& MapMn();

Plenty& MapS();

Plenty& MapMax();

~Plenty() {};

};

template <class T>

Plenty<T>::Plenty()

{

len = 0;

}

template <class T>

Plenty<T>::Plenty(int n)

{

len = n;

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

{

plenty.insert(make\_pair(i, rand() % 100));

}

}

template<class T>

void Plenty<T>::Print()

{

for (it = plenty.begin(); it != plenty.end(); it++)

{

cout << it->second << endl;

}

cout << "----------------------------------------------" << endl;

}

template<class T>

Plenty<T>& Plenty<T>::MapMn()

{

T mn = 100;

for (it = plenty.begin(); it != plenty.end(); it++)

{

if (mn > it->second)

mn = it->second;

}

cout << "Enter the number to put the item on" << endl;

int n;

cin >> n;

plenty.emplace(n, mn);

len++;

return \*this;

}

template <class T>

Plenty<T>& Plenty<T>::MapS()

{

multimap<int, T> multi;

T s = 0;

int n, n1 = 0;

for (it = plenty.begin(); it != plenty.end();it++)

{

s += it->second;

n = it->first + 1;

}

s = s / n;

for (it = plenty.begin(); it != plenty.end(); it++)

{

if (it->second < s)

{

multi.insert(make\_pair(n1, it->second));

n1++;

}

}

this->plenty = multi;

return \*this;

}

template<class T>

Plenty<T>& Plenty<T>::MapMax()

{

T mx = 0;

for (it = plenty.begin(); it != plenty.end(); it++)

{

if (mx < it->second)

mx = it->second;

}

for (it = plenty.begin(); it != plenty.end(); it++)

it->second = it->second \* mx;

return \*this;

}

int main()

{

cout << "Enter the size of the container" << endl;

int n;

cin >> n;

Plenty<int> P(n);

P.Print();

P.MapMn();

P.Print();

P.MapS();

P.Print();

P.MapMax();

P.Print();

}