МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра информационные технологии и автоматизированные системы

**Дисциплина Информатика**

**Библиотека STL**

Выполнил студент ИВТ-22-2б:

Токмаков Герман Максимович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил доцент 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.

ВАРИАНТ 15:

|  |  |  |
| --- | --- | --- |
| **Задача 1**   1. Контейнер - multiset 2. Тип элементов - double   **Задача 2**  Тип элементов Pair (см. лабораторную работу №3).  **Задача 3**  Параметризированный класс – Список (см. лабораторную работу №7) | | |
| **Задание 3** | **Задание 4** | **Задание 5** |
| Найти среднее арифметическое и добавить его в конец  контейнера | Найти элементы ключами из заданного диапазона и удалить их  из контейнера | К каждому элементу добавить сумму минимального и максимального элементов  контейнера. |

**Программный код**

main 12.1:

#include <iostream>

#include <set>

using namespace std;

multiset<double> pushArithmeticMean(const multiset<double>& f\_mset);

multiset<double> eraseRange(const multiset<double>& f\_mset, const double lowerBound, const double upperBound);

bool isInRange(const double value, const double lowerBound, const double upperBound);

multiset<double> increaseOnMinMax(const multiset<double>& f\_mset);

template <typename T>

void print(const multiset<T>& p\_list);

int main() {

cout << "task 1:" << endl;

multiset<double> a = { 1, 2, 3 };

multiset<double> b = pushArithmeticMean(a);

print(a);

print(b);

cout << "task 2:" << endl;

multiset<double> a1 = { 1, 2, 3, 4 };

multiset<double> b1 = eraseRange(a1, 2, 3);

print(a1);

print(b1);

cout << "task 3:" << endl;

multiset<double> a2 = { 1, 2, 3, 4 };

multiset<double> b2 = increaseOnMinMax(a2);

print(a2);

print(b2);

return 0;

}

multiset<double> pushArithmeticMean(const multiset<double>& f\_mset) {

multiset<double> r\_mset = f\_mset;

double sum = 0;

for (auto it = f\_mset.begin(); it != f\_mset.end(); it++) {

sum += \*it;

}

double arithmeticMean = sum / f\_mset.size();

r\_mset.insert(arithmeticMean);

return r\_mset;

}

multiset<double> eraseRange(const multiset<double>& f\_mset, const double lowerBound, const double upperBound) {

multiset<double> r\_mset = f\_mset;

for (auto it = r\_mset.begin(); it != r\_mset.end(); it++) {

if (isInRange(\*it, lowerBound, upperBound)) {

it = r\_mset.erase(it);

it--;

}

}

return r\_mset;

}

bool isInRange(const double value, const double lowerBound, const double upperBound) {

return (value >= lowerBound && value <= upperBound);

}

multiset<double> increaseOnMinMax(const multiset<double>& f\_mset) {

multiset<double> temp\_mset = f\_mset;

multiset<double> r\_mset;

double maxListValue = DBL\_MIN;

double minListValue = DBL\_MAX;

for (auto it = temp\_mset.begin(); it != temp\_mset.end(); it++) {

if (\*it > maxListValue) maxListValue = \*it;

if (\*it < minListValue) minListValue = \*it;

}

for (auto it = temp\_mset.begin(); it != temp\_mset.end(); it++) {

double value = \*it + maxListValue + minListValue;

r\_mset.insert(value);

}

return r\_mset;

}

template <typename T>

void print(const multiset<T>& p\_list) {

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

cout << \*it << " ";

}

cout << endl;

}

main 12.2:

#include <set>

#include <iostream>

#include "Pair.h"

using namespace std;

void pushPair(multiset<Pair>& f\_mset);

multiset<Pair> pushArithmeticMean(const multiset<Pair>& f\_mset);

multiset<Pair> eraseRange(const multiset<Pair>& f\_mset, const double lowerBound, const double upperBound);

bool isInRange(const double value, const double lowerBound, const double upperBound);

multiset<Pair> increaseOnMinMax(const multiset<Pair>& f\_mset);

template <typename T>

void print(const multiset<T>& p\_mset);

int main() {

cout << "task 1:" << endl;

multiset<Pair> a;

pushPair(a);

multiset<Pair> b = pushArithmeticMean(a);

print(a);

print(b);

cout << "task 2:" << endl;

multiset<Pair> a1;

pushPair(a1);

multiset<Pair> b1 = eraseRange(a1, 2, 8);

print(a1);

print(b1);

cout << "task 3:" << endl;

multiset<Pair> a2;

pushPair(a2);

multiset<Pair> b2 = increaseOnMinMax(a2);

print(a2);

print(b2);

return 0;

}

void pushPair(multiset<Pair>& f\_mset) {

f\_mset.insert(Pair(1, 5.5));

f\_mset.insert(Pair(2, 7.8));

f\_mset.insert(Pair(3, 4.3));

}

multiset<Pair> pushArithmeticMean(const multiset<Pair>& f\_mset) {

multiset<Pair> r\_mset = f\_mset;

int sumForInt = 0;

double sumForDouble = 0;

Pair tmp;

for (auto it = f\_mset.begin(); it != f\_mset.end(); it++) {

tmp = \*it;

sumForInt += tmp.getFirst();

sumForDouble += tmp.getSecond();

}

int arithmeticMeanForInt = sumForInt / static\_cast<int>(f\_mset.size());

double arithmeticMeanForDouble = sumForDouble / f\_mset.size();

r\_mset.insert(Pair(arithmeticMeanForInt, arithmeticMeanForDouble));

return r\_mset;

}

multiset<Pair> eraseRange(const multiset<Pair>& f\_mset, const double lowerBound, const double upperBound) {

multiset<Pair> r\_mset = f\_mset;

Pair tempPair;

for (auto it = r\_mset.begin(); it != r\_mset.end();) {

tempPair = \*it;

double sumFS = tempPair.getFirst() + tempPair.getSecond();

if (isInRange(sumFS, lowerBound, upperBound)) {

it = r\_mset.erase(it);

}

else it++;

}

return r\_mset;

}

bool isInRange(const double value, const double lowerBound, const double upperBound) {

return (value >= lowerBound && value <= upperBound);

}

multiset<Pair> increaseOnMinMax(const multiset<Pair>& f\_mset) {

multiset<Pair> r\_mset = f\_mset;

multiset<Pair> temp\_mset;

Pair tempPair;

int maxListValueInt = INT\_MIN;

int minListValueInt = INT\_MAX;

double maxListValueDouble = DBL\_MIN;

double minListValueDouble = DBL\_MAX;

for (auto it = r\_mset.begin(); it != r\_mset.end(); it++) {

tempPair = \*it;

if (tempPair.getFirst() < minListValueInt) minListValueInt = tempPair.getFirst();

if (tempPair.getFirst() > maxListValueInt) maxListValueInt = tempPair.getFirst();

if (tempPair.getSecond() < minListValueDouble) minListValueDouble = tempPair.getSecond();

if (tempPair.getSecond() > maxListValueDouble) maxListValueDouble = tempPair.getSecond();

}

for (auto it = r\_mset.begin(); it != r\_mset.end(); it++) {

tempPair = \*it;

tempPair = tempPair + minListValueInt + maxListValueInt + minListValueDouble + maxListValueDouble;

temp\_mset.insert(tempPair);

}

return temp\_mset;

}

template <typename T>

void print(const multiset<T>& p\_mset) {

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

cout << \*it << " ";

}

cout << endl;

}

main 12.3:

#include <iostream>

#include <set>

#include <Windows.h>

#include "Pair.h"

using namespace std;

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

multiset <Pair> mst;

multiset <Pair> tmp;

mst.insert(Pair(1, 5.5));

mst.insert(Pair(2, 7.8));

mst.insert(Pair(3, 4.3));

Pair sr(0, 0);

cout << "task 1:" << endl;

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

{

sr = sr + \*it;

cout << \*it << endl;

}

//1

sr.setFirst(sr.getFirst() / mst.size());

sr.setSecond(sr.getSecond() / mst.size());

mst.insert(sr);

//2

Pair max = \*mst.begin();

Pair min = \*mst.begin();

tmp = mst;

mst.clear();

int c = 0;

int i = 2, j = 8;

Pair res;

cout << "task 2:" << endl;

for (auto it = tmp.begin(); it != tmp.end(); it++, c++)

{

if ((c < i) || (c > j) || (res.getFirst() != (\*it).getFirst() || res.getSecond() != (\*it).getSecond()))

{

if (max < \*it) max = \*it;

if (!(min < \*it)) min = \*it;

mst.insert(\*it);

cout << \*it << endl;

}

}

cout << "task 3:" << endl;

tmp = mst;

mst.clear();

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

{

mst.insert(max + min + \*it);

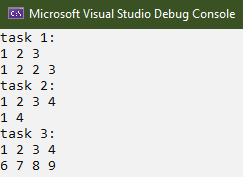
cout << max + min + \*it << endl;

}

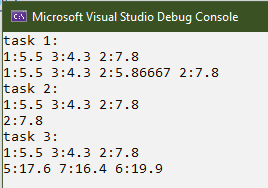
}

**Вывод программы**

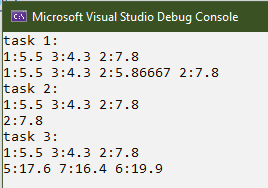
main 12.1:



main 12.2:



main 12.3:



**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.