Министерство образования и науки Российской Федерации

Федеральное государственное образовательное автономное учреждение высшего образования

"Пермский национальный исследовательский политехнический университет"

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №12

Дисциплина: Основы алгоритмизации и программирования

Тема: Ассоциативные контейнеры библиотеки STL.

|  |  |
| --- | --- |
| Выполнил работу | |
| Студент группы РИС-22-1б | |
| Бреднев М.П. | |
|  | |
| Проверил работу | |
| Доцент кафедры ИТАС | |
| Полякова О.А. | |
|  | |

Пермь – 2023

**Анализ предметной области**

**Постановка задачи**

Задача 1:

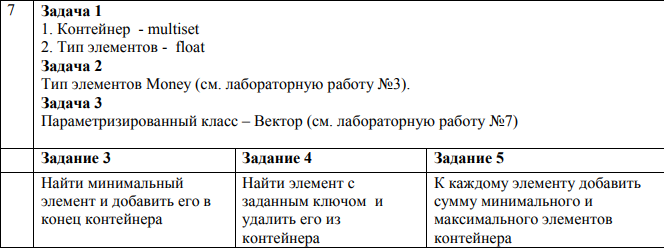
1. Создать ассоциативный контейнер.
2. Заполнить его элементами стандартного типа (тип указан в варианте).
3. Добавить элементы в соответствии с заданием.
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде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

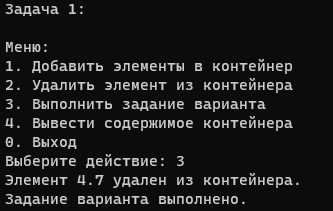
1. Создать параметризированный класс, используя в качестве контейнера ассоциативный контейнер.
2. Заполнить его элементами.
3. Добавить элементы в соответствии с заданием.
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий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



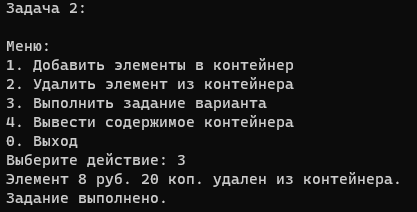
**Анализ задачи**

В данной задаче необходимо использовать контейнер multiset, который хранит данные в упорядоченном порядке, так как представляет из себя бинарное дерево. Поэтому выполнение задания 1 полностью невозможно, так как полученное число добавиться в соответствии с логикой бинарного дерева. Остальные задачи выполняются корректно с помощью встроенных методов контейнеров.

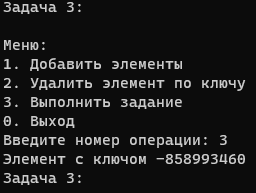
**Тестирование программы**

****

*Рис. 1 – Задача 1.*

**

*Рис. 2 – Задача 2.*

**

*Рис. 3 – Задача 3.*

**Заключение**

Была разработана программа, которая взаимодействует с ассоциативными контейнером STL библиотеки и использует его для хранения, взаимодействия и упорядочивания данных.

# Ответы на 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;

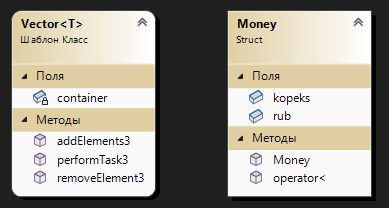
}

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.

**Приложения**

UML-диаграмма:



Приложение Б – код программы

Main.cpp:

#include <iostream>

#include <set>

#include <vector>

#include "Money1.h"

#include "CustomVector.h"

using namespace std;

// Функция для добавления элементов в контейнер

void addElements1(std::multiset<float>& container) {

container.insert(3.5);

container.insert(1.2);

container.insert(4.7);

container.insert(2.9);

}

// Функция для удаления элемента из контейнера по заданному значению

void removeElement1(std::multiset<float>& container, float value) {

auto it = container.find(value);

if (it != container.end()) {

container.erase(it);

std::cout << "Элемент " << value << " удален из контейнера.\n";

}

else {

std::cout << "Элемент " << value << " не найден в контейнере.\n";

}

}

// Функция для выполнения задания варианта

void performTask1(std::multiset<float>& container) {

if (container.empty()) {

std::cout << "Контейнер пуст. Добавьте элементы перед выполнением задания.\n";

return;

}

// Находим минимальный элемент

float minElement = \*container.begin();

// Добавляем минимальный элемент в конец контейнера

container.insert(minElement);

// Находим элемент с заданным ключом и удаляем его

float key = 4.7;

removeElement1(container, key);

// Находим максимальный элемент

float maxElement = \*container.rbegin();

// Изменяем каждый элемент контейнера, добавляя сумму минимального и максимального элементов

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

float& value = const\_cast<float&>(\*it);

value += minElement + maxElement;

}

}

// Функция для вывода содержимого контейнера

void printContainer1(const std::multiset<float>& container) {

std::cout << "Содержимое контейнера:\n";

for (const auto& value : container) {

std::cout << value << ' ';

}

std::cout << '\n';

}

void menu() {

setlocale(0, "");

std::multiset<float> container;

int choice;

do {

std::cout << "Задача 1: \n\n";

std::cout << "Меню:\n";

std::cout << "1. Добавить элементы в контейнер\n";

std::cout << "2. Удалить элемент из контейнера\n";

std::cout << "3. Выполнить задание варианта\n";

std::cout << "4. Вывести содержимое контейнера\n";

std::cout << "0. Выход\n";

std::cout << "Выберите действие: ";

std::cin >> choice;

switch (choice) {

case 1:

addElements1(container);

std::cout << "Элементы успешно добавлены в контейнер.\n";

break;

case 2: {

float value;

std::cout << "Введите значение элемента для удаления: ";

std::cin >> value;

removeElement1(container, value);

break;

}

case 3:

performTask1(container);

std::cout << "Задание варианта выполнено.\n";

break;

case 4:

printContainer1(container);

break;

case 0:

std::cout << "Выход из программы.\n";

break;

default:

std::cout << "Некорректный выбор. Попробуйте снова.\n";

break;

}

std::cout << '\n';

} while (choice != 0);

}

// Функция для добавления элементов в контейнер

void addElements2(std::multiset<Money>& container) {

container.insert(Money(10, 50));

container.insert(Money(5, 30));

container.insert(Money(8, 20));

container.insert(Money(7, 10));

}

// Функция для удаления элемента из контейнера по заданному ключу

void removeElement2(std::multiset<Money>& container, const Money& key) {

auto it = container.find(key);

if (it != container.end()) {

container.erase(it);

std::cout << "Элемент " << key << " удален из контейнера.\n";

}

else {

std::cout << "Элемент " << key << " не найден в контейнере.\n";

}

}

// Функция для выполнения задания варианта

void performTask2(std::multiset<Money>& container) {

if (container.empty()) {

std::cout << "Контейнер пуст. Добавьте элементы перед выполнением задания.\n";

return;

}

// Находим минимальный элемент

const Money& minElement = \*container.begin();

// Добавляем минимальный элемент в конец контейнера

container.insert(minElement);

// Находим элемент с заданным ключом и удаляем его

Money key(8, 20);

removeElement2(container, key);

// Находим максимальный элемент

const Money& maxElement = \*container.rbegin();

// Создаем временный контейнер для сохранения измененных элементов

std::multiset<Money> tempContainer;

// Добавляем сумму минимального и максимального элементов к каждому элементу контейнера

for (const auto& money : container) {

Money updatedMoney = money;

updatedMoney.rub += minElement.rub + maxElement.rub;

updatedMoney.kopeks += minElement.kopeks + maxElement.kopeks;

// Проверяем и корректируем значение копеек, если оно превышает 99

if (updatedMoney.kopeks > 99) {

updatedMoney.rub += updatedMoney.kopeks / 100;

updatedMoney.kopeks %= 100;

}

// Добавляем измененный элемент во временный контейнер

tempContainer.insert(updatedMoney);

}

// Заменяем содержимое исходного контейнера на содержимое временного контейнера

container = tempContainer;

}

// Функция для вывода содержимого контейнера

void printContainer2(const std::multiset<Money>& container) {

std::cout << "Содержимое контейнера:\n";

for (const auto& money : container) {

std::cout << money << '\n';

}

}

void menu2() {

std::multiset<Money> container;

int choice;

do {

cout << "Задача 2: \n\n";

std::cout << "Меню:\n";

std::cout << "1. Добавить элементы в контейнер\n";

std::cout << "2. Удалить элемент из контейнера\n";

std::cout << "3. Выполнить задание варианта\n";

std::cout << "4. Вывести содержимое контейнера\n";

std::cout << "0. Выход\n";

std::cout << "Выберите действие: ";

std::cin >> choice;

switch (choice) {

case 1:

addElements2(container);

std::cout << "Элементы добавлены в контейнер.\n";

break;

case 2: {

int rub, kopeks;

std::cout << "Введите ключ для удаления (рубли и копейки): ";

std::cin >> rub >> kopeks;

removeElement2(container, Money(rub, kopeks));

break;

}

case 3:

performTask2(container);

std::cout << "Задание выполнено.\n";

break;

case 4:

printContainer2(container);

break;

case 0:

std::cout << "Выход из программы.\n";

break;

default:

std::cout << "Некорректный выбор. Попробуйте снова.\n";

break;

}

std::cout << '\n';

} while (choice != 0);

}

void menu3() {

Vector<int> vec;

// Заполнение контейнера элементами

std::vector<int> elements = { 5, 2, 8, 1, 9 };

vec.addElements3(elements);

int choice;

do {

cout << "Задача 3: \n\n";

// Вывод меню и обработка выбора пользователя

std::cout << "Меню:\n";

std::cout << "1. Добавить элементы\n";

std::cout << "2. Удалить элемент по ключу\n";

std::cout << "3. Выполнить задание\n";

std::cout << "0. Выход\n";

std::cout << "Введите номер операции: ";

std::cin >> choice;

switch (choice) {

case 1: {

vector<int> newElements;

// Ввод новых элементов

// ...

vec.addElements3(newElements);

break;

}

case 2: {

int key;

// Ввод ключа элемента для удаления

// ...

vec.removeElement3(key);

break;

}

case 3: {

vec.performTask3();

break;

}

case 0: {

std::cout << "Программа завершена.\n";

break;

}

default: {

std::cout << "Некорректный выбор. Попробуйте снова.\n";

break;

}

}

} while (choice != 0);

}

int main() {

setlocale(0, "");

menu();

menu2();

menu3();

return 0;

}

CustomVector.h:

#pragma once

#include <iostream>

#include <set>

#include <vector>

template <typename T>

class Vector {

private:

std::multiset<T> container;

public:

void addElements3(const std::vector<T>& elements) {

for (const auto& element : elements) {

container.insert(element);

}

}

void removeElement3(const T& key) {

auto it = container.find(key);

if (it != container.end()) {

container.erase(it);

std::cout << "Элемент " << key << " удален из контейнера.\n";

}

else {

std::cout << "Элемент " << key << " не найден в контейнере.\n";

}

}

void performTask3() {

if (container.empty()) {

std::cout << "Контейнер пуст. Добавьте элементы перед выполнением задания.\n";

return;

}

// Находим минимальный элемент

const T& minElement = \*container.begin();

// Добавляем минимальный элемент в конец контейнера

container.insert(minElement);

// Находим элемент с заданным ключом

T keyElement; // Здесь нужно указать соответствующий тип для ключа элемента

auto keyIt = container.find(keyElement);

if (keyIt != container.end()) {

// Удаляем элемент с заданным ключом

container.erase(keyIt);

std::cout << "Элемент с ключом " << keyElement << " удален из контейнера.\n";

}

else {

std::cout << "Элемент с ключом " << keyElement << " не найден в контейнере.\n";

}

// Добавляем сумму минимального и максимального элементов к каждому элементу контейнера

const T& maxElement = \*container.rbegin();

for (auto element : container) {

element += minElement + maxElement;

}

}

};

Money.h:

#pragma once

#include <iostream>

#include <set>

#include <vector>

struct Money {

int rub;

int kopeks;

Money(int rub = 0, int kopeks = 0) : rub(rub), kopeks(kopeks) {}

// Перегрузка оператора < для сравнения элементов Money

bool operator<(const Money& other) const {

if (rub != other.rub) {

return rub < other.rub;

}

return kopeks < other.kopeks;

}

// Перегрузка оператора << для вывода элементов Money

friend std::ostream& operator<<(std::ostream& os, const Money& money) {

os << money.rub << " руб. " << money.kopeks << " коп.";

return os;

}};