Министерство образования и науки Российской Федерации

Федеральное государственное образовательное автономное учреждение высшего образования

"Пермский национальный исследовательский политехнический университет"

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №13

Дисциплина: Основы алгоритмизации и программирования

Тема: Стандартные обобщенные алгоритмы 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:

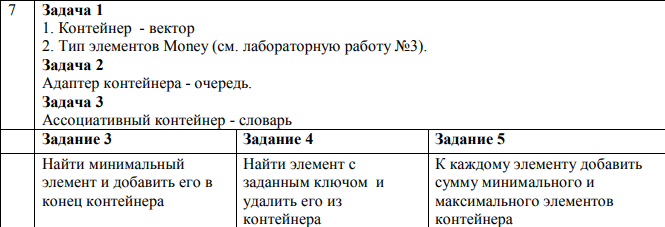
1. Создать последовательный контейнер.
2. Заполнить его элементами стандартного типа (тип указан в варианте).
3. Заменить элементы в соответствии с заданием (использовать алгоритмы replace\_if(), replace\_copy(), replace copy\_if(), fill()).
4. Удалить элементы в соответствии с заданием (использовать 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. Выполнение всех заданий использовать стандартные алгоритмы библиотеки STL.

Задача 2:

1. Создать адаптер контейнер.
2. Заполнить его элементами стандартного типа (тип указан в варианте).
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. Выполнение всех заданий использовать стандартные алгоритмы библиотеки STL.

Задача 3:

1. Создать ассоциативный контейнер.
2. Заполнить его элементами стандартного типа (тип указан в варианте).
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. Выполнение всех заданий использовать стандартные алгоритмы библиотеки STL.

****

**Анализ задачи**

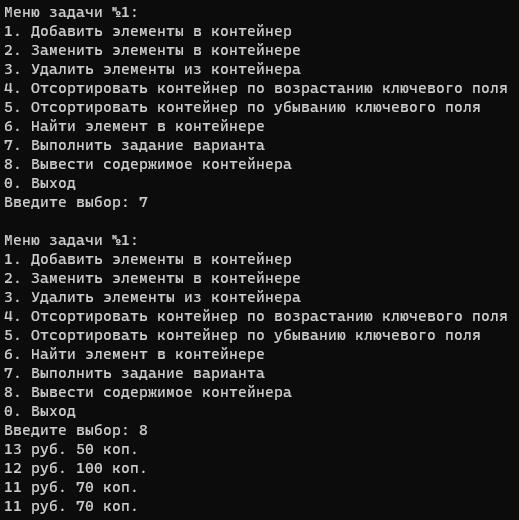
Задача 1: для того, чтобы реализовать функцию добавления элемента со средним значений можно воспользоваться алгоритмом for\_each с лямбда выражением, которое будет добавлять к временному элементу данные. А после поделить данный элемент на длину списка. Для задания 2 необходимо использовать функцию remove\_if с лямбда выражением (предикатом), который сверяет данные списка с заданным элементом. Для выполнения задания 3 необходимо использовать функции min\_element и max\_element, для того чтобы найти минимальный и максимальный элемент в списке.

Задача 2: у очереди с приоритетами нет итераторов, а также с ней не могут взаимодействовать функции из библиотеки algorithm, потому что они основаны на том, что они работают с итераторами. Поэтому задачи, которые указаны в варианте номер 15 невозможно выполнить методами, которые предложены. Даже какими-либо методами приоритетной очереди это тоже не

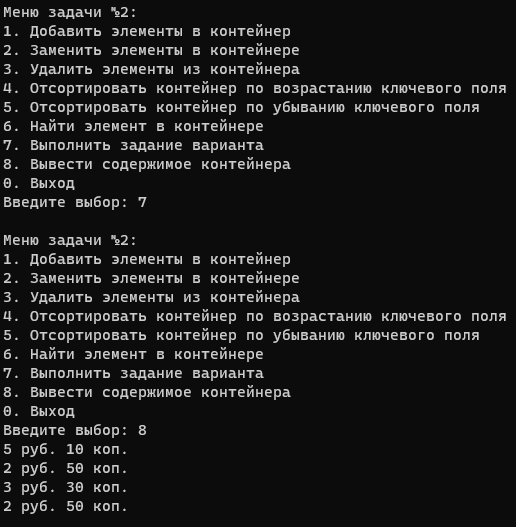
выполняются.

Задача 3: Итераторы контейнера map в большинстве функций библиотеки algorithm не поддерживаются. Поэтому в данной задаче использовано 2 функции: min\_element и max\_element, для того чтобы найти минимальный и максимальный элемент в списке соответственно. Это обусловлено тем, что данные в этом контейнере хранятся парно.

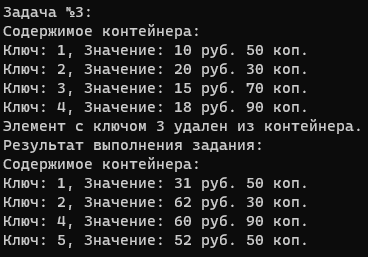
**Тестирование программы**

****

*Рис. 1 – Задача 1.*

**

*Рис. 2 – Задача 2.*

**

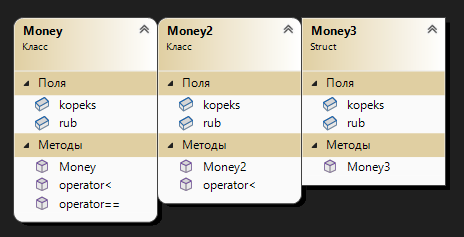
*Рис. 3 – Задача 3.*

**Заключение**

Была разработана программа, которая взаимодействует с контейнерами STL библиотеки и использует его для хранения, взаимодействия и упорядочивания данных. С данными контейнерами взаимодействуют функции из библиотеки STL algorithm.

**Приложения**

UML-диаграмма:



Приложение Б – код программы

Main.cpp:

#include <iostream>

#include <vector>

#include <algorithm>

#include <queue>

#include <map>

#include "Money1.h"

#include "Money2.h"

#include "Money3.h"

using namespace std;

// Определение класса Money

void addElements1(std::vector<Money>& container) {

container.push\_back(Money(5, 10));

container.push\_back(Money(2, 50));

container.push\_back(Money(8, 75));

container.push\_back(Money(3, 30));

}

// Функция для замены элементов в контейнере

void replaceElements1(std::vector<Money>& container) {

Money oldValue(2, 50);

Money newValue(4, 60);

std::replace(container.begin(), container.end(), oldValue, newValue);

}

// Функция для удаления элементов из контейнера

void removeElements1(std::vector<Money>& container) {

Money valueToRemove(8, 75);

container.erase(std::remove(container.begin(), container.end(), valueToRemove), container.end());

}

// Функция для сортировки контейнера по возрастанию ключевого поля

void sortAscending1(std::vector<Money>& container) {

std::sort(container.begin(), container.end());

}

// Функция для сортировки контейнера по убыванию ключевого поля

void sortDescending1(std::vector<Money>& container) {

std::sort(container.rbegin(), container.rend());

}

// Функция для поиска элемента в контейнере

void findElement1(const std::vector<Money>& container) {

Money target(3, 30);

auto it = std::find\_if(container.begin(), container.end(), [&target](const Money& m) {

return m == target;

});

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

std::cout << "Элемент найден.\n";

}

else {

std::cout << "Элемент не найден.\n";

}

}

// Функция для выполнения задания варианта

void performTask1(std::vector<Money>& container) {

if (container.empty()) {

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

return;

}

// Находим минимальный элемент

Money minElement = \*std::min\_element(container.begin(), container.end());

// Добавляем минимальный элемент в конец контейнера

container.push\_back(minElement);

// Находим элемент с заданным ключом и удаляем его

Money key(8, 75);

container.erase(std::remove\_if(container.begin(), container.end(), [&key](const Money& m) {

return m == key;

}), container.end());

// Находим максимальный элемент

Money maxElement = \*std::max\_element(container.begin(), container.end());

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

std::for\_each(container.begin(), container.end(), [&minElement, &maxElement](Money& m) {

m.rub += minElement.rub + maxElement.rub;

m.kopeks += minElement.kopeks + maxElement.kopeks;

});

}

// Функция для вывода содержимого контейнера

void printContainer1(const std::vector<Money>& container) {

for (const auto& money : container) {

std::cout << money.rub << " руб. " << money.kopeks << " коп.\n";

}

}

void menu1() {

std::vector<Money> container;

int choice;

do {

std::cout << "Меню задачи №1:\n";

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

std::cout << "2. Заменить элементы в контейнере\n";

std::cout << "3. Удалить элементы из контейнера\n";

std::cout << "4. Отсортировать контейнер по возрастанию ключевого поля\n";

std::cout << "5. Отсортировать контейнер по убыванию ключевого поля\n";

std::cout << "6. Найти элемент в контейнере\n";

std::cout << "7. Выполнить задание варианта\n";

std::cout << "8. Вывести содержимое контейнера\n";

std::cout << "0. Выход\n";

std::cout << "Введите выбор: ";

std::cin >> choice;

switch (choice) {

case 1:

addElements1(container);

break;

case 2:

replaceElements1(container);

break;

case 3:

removeElements1(container);

break;

case 4:

sortAscending1(container);

break;

case 5:

sortDescending1(container);

break;

case 6:

findElement1(container);

break;

case 7:

performTask1(container);

break;

case 8:

printContainer1(container);

break;

case 0:

std::cout << "Выход.\n";

break;

default:

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

break;

}

std::cout << std::endl;

} while (choice != 0);

}

// Определение класса Money

// Функция для добавления элементов в контейнер

void addElements2(queue<Money2>& container) {

container.push(Money2(5, 10));

container.push(Money2(2, 50));

container.push(Money2(8, 75));

container.push(Money2(3, 30));

}

// Функция для замены элементов в контейнере

void replaceElements2(queue<Money2>& container) {

Money2 oldValue(2, 50);

Money2 newValue(4, 60);

queue<Money2> temp;

while (!container.empty()) {

Money2& money = container.front();

if (money == oldValue) {

temp.push(newValue);

}

else {

temp.push(money);

}

container.pop();

}

container = move(temp);

}

// Функция для удаления элементов из контейнера

void removeElements2(queue<Money2>& container) {

Money2 valueToRemove(8, 75);

queue<Money2> temp;

while (!container.empty()) {

Money2& money = container.front();

if (money != valueToRemove) {

temp.push(money);

}

container.pop();

}

container = move(temp);

}

// Функция для сортировки контейнера по возрастанию ключевого поля

void sortAscending2(queue<Money2>& container) {

vector<Money2> temp;

while (!container.empty()) {

temp.push\_back(container.front());

container.pop();

}

sort(temp.begin(), temp.end());

for (const auto& money : temp) {

container.push(money);

}

}

// Функция для сортировки контейнера по убыванию ключевого поля

void sortDescending2(queue<Money2>& container) {

vector<Money2> temp;

while (!container.empty()) {

temp.push\_back(container.front());

container.pop();

}

sort(temp.rbegin(), temp.rend());

for (const auto& money : temp) {

container.push(money);

}

}

// Функция для поиска элемента в контейнере

void findElement2(const queue<Money2>& container) {

Money2 target(3, 30);

queue<Money2> temp = container;

bool found = false;

while (!temp.empty()) {

if (temp.front() == target) {

found = true;

break;

}

temp.pop();

}

if (found) {

cout << "Элемент найден.\n";

}

else {

cout << "Элемент не найден.\n";

}

}

// Функция для выполнения задания варианта

void performTask2(queue<Money2>& container) {

if (container.empty()) {

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

return;

}

// Находим минимальный элемент

Money2 minElement = container.front();

queue<Money2> temp = container;

while (!temp.empty()) {

if (temp.front() < minElement) {

minElement = temp.front();

}

temp.pop();

}

// Добавляем минимальный элемент в конец контейнера

container.push(minElement);

// Находим элемент с заданным ключом и удаляем его

Money2 key(8, 75);

temp = container;

queue<Money2> updatedContainer;

while (!temp.empty()) {

if (temp.front() != key) {

updatedContainer.push(temp.front());

}

temp.pop();

}

container = move(updatedContainer);

// Находим максимальный элемент

Money2 maxElement = container.front();

temp = container;

while (!temp.empty()) {

if (temp.front() > maxElement) {

maxElement = temp.front();

}

temp.pop();

}

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

temp = container;

while (!temp.empty()) {

temp.front().rub += minElement.rub + maxElement.rub;

temp.front().kopeks += minElement.kopeks + maxElement.kopeks;

temp.pop();

}

}

// Функция для вывода содержимого контейнера

void printContainer2(const queue<Money2>& container) {

queue<Money2> temp = container;

while (!temp.empty()) {

cout << temp.front().rub << " руб. " << temp.front().kopeks << " коп.\n";

temp.pop();

}

}

void menu2() {

queue<Money2> container;

int choice;

do {

cout << "Меню задачи №2:\n";

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

cout << "2. Заменить элементы в контейнере\n";

cout << "3. Удалить элементы из контейнера\n";

cout << "4. Отсортировать контейнер по возрастанию ключевого поля\n";

cout << "5. Отсортировать контейнер по убыванию ключевого поля\n";

cout << "6. Найти элемент в контейнере\n";

cout << "7. Выполнить задание варианта\n";

cout << "8. Вывести содержимое контейнера\n";

cout << "0. Выход\n";

cout << "Введите выбор: ";

cin >> choice;

switch (choice) {

case 1:

addElements2(container);

break;

case 2:

replaceElements2(container);

break;

case 3:

removeElements2(container);

break;

case 4:

sortAscending2(container);

break;

case 5:

sortDescending2(container);

break;

case 6:

findElement2(container);

break;

case 7:

performTask2(container);

break;

case 8:

printContainer2(container);

break;

case 0:

cout << "Программа завершена.\n";

break;

default:

cout << "Некорректный выбор. Попробуйте еще раз.\n";

break;

}

cout << "\n";

} while (choice != 0);

}

// Функция для добавления элементов в контейнер

void addElements3(map<int, Money3>& container) {

container.insert({ 1, Money3(10, 50) });

container.insert({ 2, Money3(20, 30) });

container.insert({ 3, Money3(15, 70) });

container.insert({ 4, Money3(18, 90) });

}

// Функция для удаления элемента из контейнера по заданному ключу

void removeElement3(map<int, Money3>& container, int key) {

auto it = container.find(key);

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

container.erase(it);

cout << "Элемент с ключом " << key << " удален из контейнера.\n";

}

else {

cout << "Элемент с ключом " << key << " не найден в контейнере.\n";

}

}

// Функция для выполнения задания варианта

void performTask3(map<int, Money3>& container) {

if (container.empty()) {

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

return;

}

// Находим минимальный элемент

const Money3& minElement = container.begin()->second;

// Добавляем минимальный элемент в конец контейнера

container.insert({ 5, minElement });

// Находим элемент с заданным ключом и удаляем его

int key = 3;

removeElement3(container, key);

// Находим максимальный элемент

const Money3& maxElement = container.rbegin()->second;

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

for (auto& entry : container) {

Money3& money = entry.second;

money.rub += minElement.rub + maxElement.rub;

money.kopeks += minElement.kopeks + maxElement.kopeks;

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

if (money.kopeks > 99) {

money.rub += money.kopeks / 100;

money.kopeks %= 100;

}

}

}

// Функция для вывода содержимого контейнера

void printContainer3(const map<int, Money3>& container) {

cout << "Содержимое контейнера:\n";

for (const auto& entry : container) {

cout << "Ключ: " << entry.first << ", Значение: " << entry.second.rub << " руб. "

<< entry.second.kopeks << " коп.\n";

}

}

void menu3() {

cout << "Задача №3: \n";

map<int, Money3> container;

addElements3(container);

printContainer3(container);

performTask3(container);

cout << "Результат выполнения задания:\n";

printContainer3(container);

}

int main() {

setlocale(0, "");

menu1();

menu2();

menu3();

}

Money1.h:

#pragma once

#include <iostream>

#include <vector>

#include <algorithm>

#include <queue>

#include <map>

class Money {

public:

int rub;

int kopeks;

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

bool operator<(const Money& other) const {

if (rub == other.rub)

return kopeks < other.kopeks;

return rub < other.rub;

}

bool operator==(const Money& other) const {

return rub == other.rub && kopeks == other.kopeks;

}

};

Money2.h:

#pragma once

#include <iostream>

#include <vector>

#include <algorithm>

#include <queue>

#include <map>

class Money2 {

public:

int rub;

int kopeks;

Money2(int rub = 0, int kopeks = 0) : rub(rub), kopeks(kopeks) {}

bool operator<(const Money2& other) const {

if (rub == other.rub)

return kopeks < other.kopeks;

return rub < other.rub;

}

};

bool operator==(const Money2& lhs, const Money2& rhs) {

return lhs.rub == rhs.rub && lhs.kopeks == rhs.kopeks;

}

bool operator!=(const Money2& lhs, const Money2& rhs) {

return !(lhs == rhs);

}

bool operator>(const Money2& lhs, const Money2& rhs) {

if (lhs.rub == rhs.rub)

return lhs.kopeks > rhs.kopeks;

return lhs.rub > rhs.rub;

}

Money3.h:

#pragma once

#include <iostream>

#include <vector>

#include <algorithm>

#include <queue>

#include <map>

struct Money3 {

int rub;

int kopeks;

Money3(int rub = 0, int kopeks = 0) : rub(rub), kopeks(kopeks) {}

};

bool operator<(const Money3& lhs, const Money3& rhs) {

if (lhs.rub < rhs.rub) {

return true;

}

else if (lhs.rub == rhs.rub) {

return lhs.kopeks < rhs.kopeks;

}

return false;

}

bool operator==(const Money3& lhs, const Money3& rhs) {

return lhs.rub == rhs.rub && lhs.kopeks == rhs.kopeks;

}