Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы» направление подготовки: 09.03.04 – «Программная инженерия»

# Лабораторная работа " Стандартные обобщенные алгоритмы библиотеки STL."

Выполнил студент гр. РИС-24-3б

Жиряков Леонид Антонович

Проверил:

Доцент кафедры ИТАС   
Ольга Андреевна Полякова

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (оценка) (подпись)

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

(дата)

г. Пермь, 2024

Вариант 8.

Постановка задачи:

Общая:

Задача 1.

I. Создать последовательный контейнер.

2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.

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. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

Задача 2.

1. Создать адаптер контейнера.

2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.

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 элемент с заданным ключевым полем (использовать алгоритмы find(), find\_if(), count(), count\_if()).

7. Выполнить задание варианта для полученного контейнера (использовать алгоритм for each()).

8. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

Задача 3

I. Создать ассоциативный контейнер.

2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.

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())

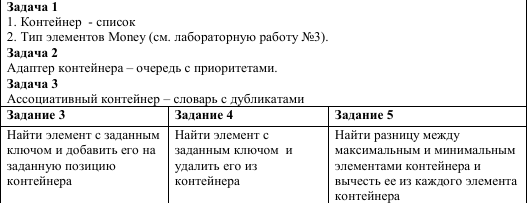
5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort () ).

6. Найти в контейнере элемент с заданным ключевым полем (использовать алгоритмы find(), find\_if(), count(), count\_if()).

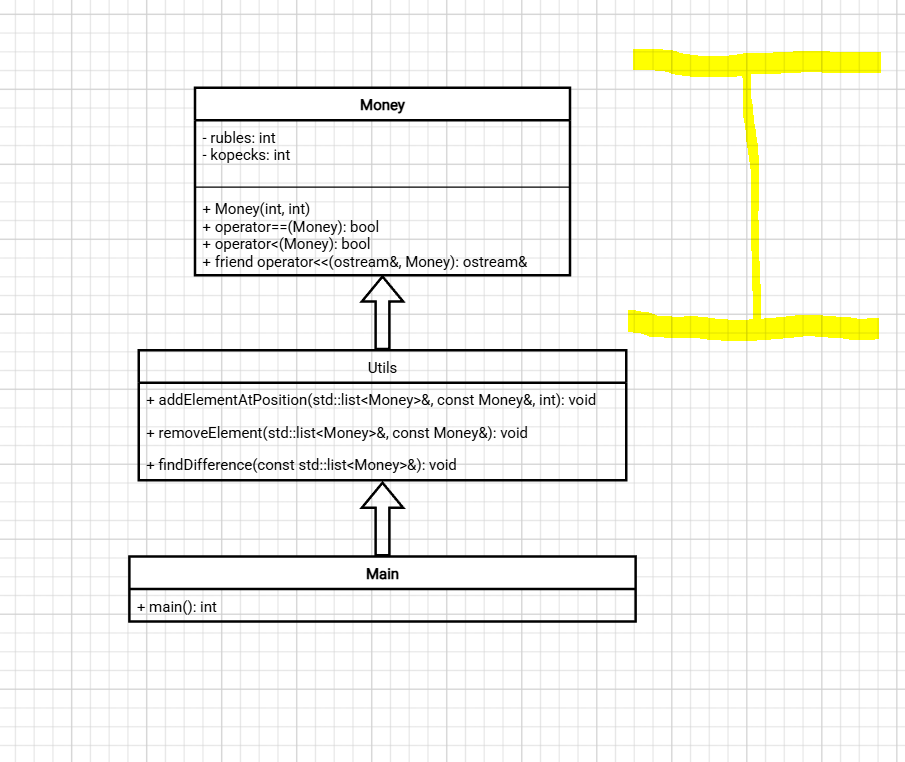
7. Выполнить задание варианта для полученного контейнера (использовать алгоритм for each()).

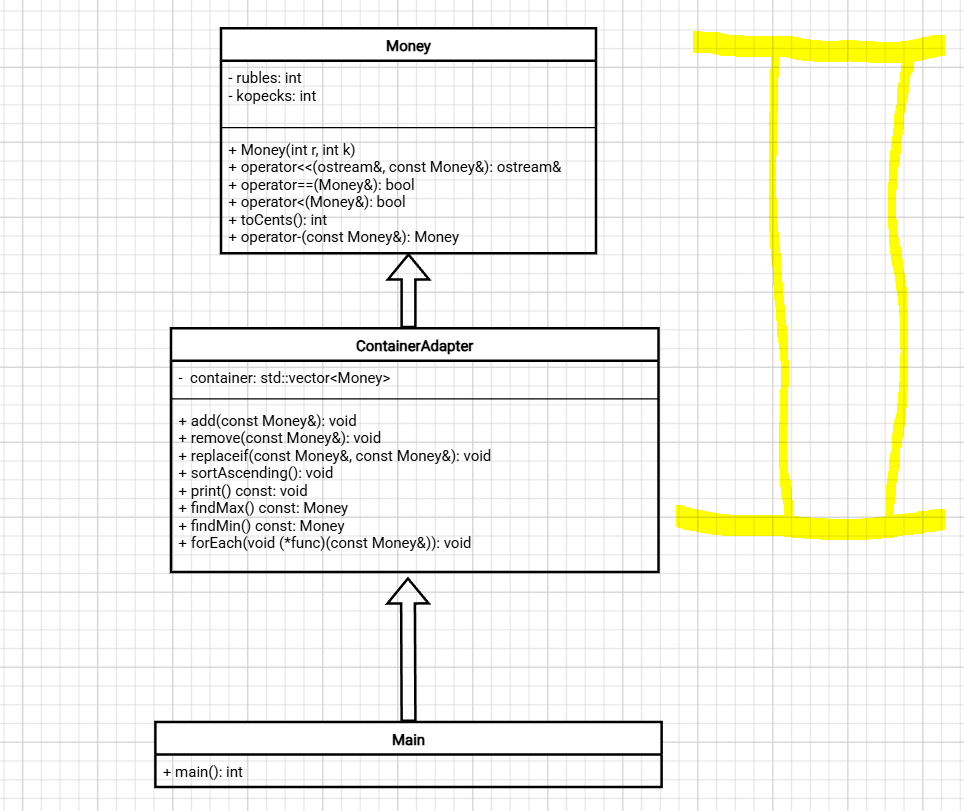
8. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

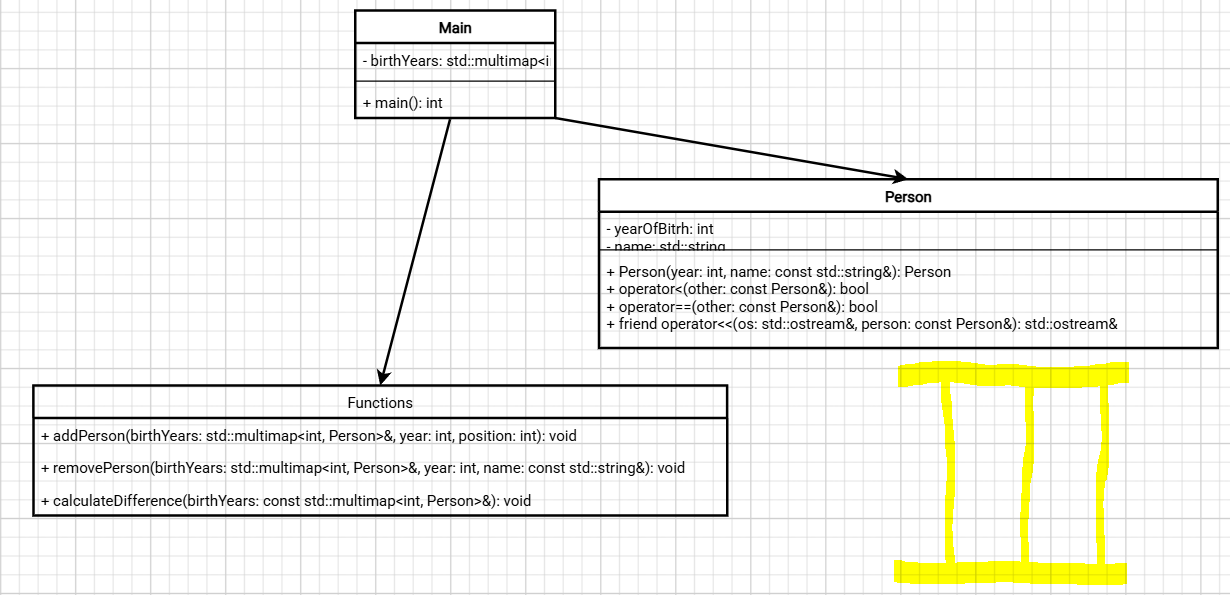
Персональная:



UML – диаграмма:







Программная реализация

**Заголовочный файлы**

**I  
money.h:**#pragma once

#include <iostream>

#include <iomanip>

class Money

{

public:

int rubles;

int kopecks;

Money(int r = 0, int k = 0) : rubles(r), kopecks(k)

{

if (kopecks >= 100)

{

rubles += kopecks / 100;

kopecks = kopecks % 100;

}

}

bool operator==(const Money& other) const

{

return rubles == other.rubles && kopecks == other.kopecks;

}

bool operator<(const Money& other) const

{

return rubles < other.rubles || (rubles == other.rubles && kopecks < other.kopecks);

}

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

{

os << money.rubles << " руб. " << std::setw(2) << std::setfill('0') << money.kopecks << " коп.";

return os;

}

}; **utils.h:**#pragma once

#include <list>

#include <algorithm>

#include <iostream>

#include "money.h"

void addElementAtPosition(std::list<Money>& moneyList, const Money& element, int position)

{

auto it = moneyList.begin();

std::advance(it, position);

moneyList.insert(it, element);

}

void removeElement(std::list<Money>& moneyList, const Money& element)

{

moneyList.remove(element);

}

void findDifference(const std::list<Money>& moneyList)

{

auto minElement = std::min\_element(moneyList.begin(), moneyList.end());

auto maxElement = std::max\_element(moneyList.begin(), moneyList.end());

int difference = (maxElement->rubles - minElement->rubles) \* 100 + (maxElement->kopecks - minElement->kopecks);

std::cout << "Разница между максимальным и минимальным элементами: " << difference << " коп." << std::endl;

for (const auto& money : moneyList)

{

Money newMoney(money.rubles, money.kopecks - difference);

std::cout << "Элемент после вычитания: " << newMoney << std::endl;

}

} **II  
ContainerAdapter.h:**#pragma once

#include <vector>

#include <algorithm>

#include "Money.h"

class ContainerAdapter

{

private:

std::vector<Money> container;

public:

void add(const Money& m)

{

container.push\_back(m);

}

void remove(const Money& m)

{

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

}

void replaceIf(const Money& oldValue, const Money& newValue)

{

std::replace\_if(container.begin(), container.end(),

[&oldValue](const Money& m) { return m == oldValue; }, newValue);

}

void sortAscending()

{

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

}

void sortDescending()

{

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

}

void print() const

{

for (const auto& money : container)

{

std::cout << money << " ";

}

std::cout << std::endl;

}

Money findMax() const

{

return \*std::max\_element(container.begin(), container.end());

}

Money findMin() const

{

return \*std::min\_element(container.begin(), container.end());

}

void forEach(void (\*func)(const Money&)) const

{

std::for\_each(container.begin(), container.end(), func);

}

}; **Money.h:**# pragma once

#include <iostream>

class Money

{

private:

int rubles; // рубли

int kopecks; // копейки

public:

Money(int r = 0, int k = 0) : rubles(r), kopecks(k)

{

if (kopecks >= 100)

{

rubles += kopecks / 100;

kopecks %= 100;

}

}

// Перегрузка операторов

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

{

os << m.rubles << "." << (m.kopecks < 10 ? "0" : "") << m.kopecks;

return os;

}

bool operator==(const Money& other) const

{

return rubles == other.rubles && kopecks == other.kopecks;

}

bool operator<(const Money& other) const

{

return rubles \* 100 + kopecks < other.rubles \* 100 + other.kopecks;

}

int toCents() const

{

return rubles \* 100 + kopecks;

}

Money operator-(const Money& other) const

{

int totalKopecks = (rubles \* 100 + kopecks) - (other.rubles \* 100 + other.kopecks);

return Money(totalKopecks / 100, totalKopecks % 100);

}

}; **III**

**Person.h:**#pragma once

#include <iostream>

#include <string>

class Person

{

public:

int yearOfBirth;

std::string name;

Person(int year, const std::string& name) : yearOfBirth(year), name(name) {}

bool operator<(const Person& other) const

{

return yearOfBirth < other.yearOfBirth;

}

bool operator==(const Person& other) const

{

return yearOfBirth == other.yearOfBirth && name == other.name;

}

friend std::ostream& operator<<(std::ostream& os, const Person& person)

{

os << person.name << " (" << person.yearOfBirth << ")";

return os;

}

}; **Functions.h:**#pragma once

#include <iostream>

#include <map>

#include <algorithm>

void addPerson(std::multimap<int, Person>& birthYears, int year, int position)

{

auto it = birthYears.find(year);

if (it != birthYears.end())

{

// Проверка, чтобы позиция не превышала количество найденных элементов

int count = std::distance(birthYears.lower\_bound(year), birthYears.upper\_bound(year));

if (position >= count) {

birthYears.emplace(year, Person(year, it->second.name));

}

else {

std::advance(it, position);

birthYears.emplace(it->first, Person(year, it->second.name));

}

}

}

void removePerson(std::multimap<int, Person>& birthYears, int year, const std::string& name)

{

auto range = birthYears.equal\_range(year);

for (auto it = range.first; it != range.second; ++it)

{

if (it->second.name == name)

{

birthYears.erase(it);

return; // Удаляем только первого совпавшего человека

}

}

}

void calculateDifference(const std::multimap<int, Person>& birthYears)

{

auto minMax = std::minmax\_element(birthYears.begin(), birthYears.end(),

[](const auto& a, const auto& b)

{

return a.second.yearOfBirth < b.second.yearOfBirth;

});

int minYear = minMax.first->second.yearOfBirth;

int maxYear = minMax.second->second.yearOfBirth;

int difference = maxYear - minYear;

std::cout << "Разница между максимальным и минимальным годом: " << difference << std::endl;

}

**Функция main**

**I**#include <iostream>

#include <list>

#include <algorithm>

#include "money.h"

#include "utils.h"

int main()

{

setlocale(LC\_ALL, "Russian");

std::list<Money> moneyList =

{

Money(10, 50), Money(5, 25), Money(7, 75), Money(10, 0), Money(5, 100)

};

// Задание 3: Найти элемент и добавить его на заданную позицию

Money key(10, 50);

int position = 2; // Вставляем на позицию 2

addElementAtPosition(moneyList, key, position);

// Задание 4: Удалить элемент

removeElement(moneyList, key);

// Задание 5: Найти разницу между макс. и мин. элементами

findDifference(moneyList);

// Сортировка по возрастанию

moneyList.sort();

// Вывод списка

std::cout << "Список денег: " << std::endl;

for (const auto& money : moneyList)

{

std::cout << money << std::endl;

}

return 0;

}

**II**#include <iostream>

#include "Money.h"

#include "ContainerAdapter.h"

void printMoney(const Money& m)

{

std::cout << m << " ";

}

int main()

{

setlocale(LC\_ALL, "Russian");

ContainerAdapter adapter;

adapter.add(Money(10, 50));

adapter.add(Money(5, 75));

adapter.add(Money(7, 25));

adapter.add(Money(10, 50));

std::cout << "Элементы контейнера: ";

adapter.print();

// Задача 3: Найти элемент и добавить его на заданную позицию

Money searchValue(10, 50);

adapter.add(searchValue);

adapter.print();

// Задача 4: Найти элемент и удалить его

adapter.remove(searchValue);

std::cout << "После удаления " << searchValue << ": ";

adapter.print();

// Задача 5: Найти разницу между максимальным и минимальным элементами

Money maxMoney = adapter.findMax();

Money minMoney = adapter.findMin();

Money difference = maxMoney - minMoney;

std::cout << "Разница между максимальным и минимальным значением: " << difference << std::endl;

std::cout << "Все элементы после применения функции: ";

adapter.forEach(printMoney);

return 0;

} **III**

#include <iostream>

#include <map>

#include <vector>

#include <algorithm>

#include "Person.h"

#include "Functions.h"

int main()

{

setlocale(LC\_ALL, "Russinan");

system("chcp 1251");

system("cls");

// Контейнер-словарь с дубликатами

std::multimap<int, Person> birthYears;

// Заполнение контейнера

birthYears.emplace(1984, Person(1987, "Иван"));

birthYears.emplace(1990, Person(1990, "Максим"));

birthYears.emplace(1987, Person(1987, "Черешня"));

// Задание 3: Найти элемент с заданным ключом и добавить его на заданную позицию

addPerson(birthYears, 1990, 2); // Добавляем на позицию 1

// Задание 4: Найти элемент с заданным ключом и удалить его из контейнера

removePerson(birthYears, 1984, "Иван"); // Удаляем только Иван

// Задание 5: Найти разницу между максимальным и минимальным элементами контейнера

calculateDifference(birthYears);

// Сортировка по возрастанию и убыванию

std::vector<std::pair<int, Person>> sortedBirthYears(birthYears.begin(), birthYears.end());

// Убывание

std::sort(sortedBirthYears.begin(), sortedBirthYears.end(), std::greater<>());

// Вывод результатов в порядке убывания

std::cout << "Отсортировано по убыванию:\n";

for (const auto& p : sortedBirthYears)

{

std::cout << p.second << std::endl;

}

// Возрастание

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

// Вывод результатов в порядке возрастания

std::cout << "Отсортировано по возрастанию:\n";

for (const auto& p : sortedBirthYears)

{

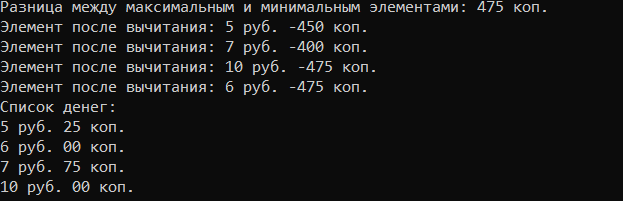
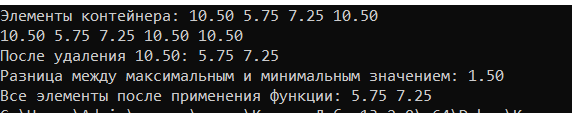
std::cout << p.second << std::endl;

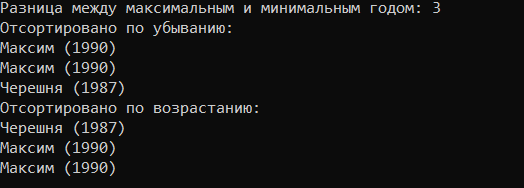
}

return 0;

}

Результат работы:

**I** **II** **III**



Ссылка:

**https://github.com/LeonidZhir/-**