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 №13

# По теме **«**Стандартные обобщенные алгоритмы библиотеки STL**»**

Вариант №10

Выполнял:

студент группы РИС-24-1б

Морозова Н.С.

Проверял:

Доцент кафедры ИТАС

Полякова О.А.

Постановка задач:

## Задача 1

1. 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()).

1. Удалить элементы в соответствии с заданием (использовать алгоритмы

remove(),remove\_if(), remove\_copy\_if(),remove\_copy())

1. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).
2. Найти в контейнере заданный элемент (использовать алгоритмы find(), find\_if(), count(), count\_if()).
3. Выполнить задание варианта для полученного контейнера (использовать алгоритм for\_each()) .
4. 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()).

1. Удалить элементы в соответствии с заданием (использовать алгоритмы

remove(),remove\_if(), remove\_copy\_if(),remove\_copy())

1. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).
2. Найти в контейнере элемент с заданным ключевым полем (использовать

алгоритмы find(), find\_if(), count(), count\_if()).

1. Выполнить задание варианта для полученного контейнера (использовать алгоритм for\_each()) .
2. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

## Задача 3

1. 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()).

1. Удалить элементы в соответствии с заданием (использовать алгоритмы

remove(),remove\_if(), remove\_copy\_if(),remove\_copy())

1. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).
2. Найти в контейнере элемент с заданным ключевым полем (использовать

алгоритмы find(), find\_if(), count(), count\_if()).

1. Выполнить задание варианта для полученного контейнера (использовать алгоритм for\_each()) .
2. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

Задание:

Задача 1

1. Контейнер - вектор
2. Тип элементов Money.

Задача 2

Адаптер контейнера – очередь с приоритетами.

Задача 3

Ассоциативный контейнер – множество с дубликатами

|  |  |  |
| --- | --- | --- |
| **Задание 3** | **Задание 4** | **Задание 7** |
| Найти минимальный элемент и добавить его на  заданную позицию контейнера | Найти элементы большие среднего арифметического и удалить их из контейнера | Каждый элемент домножить на максимальный элемент контейнера |

Анализ задачи:

1. В первой задаче (файл Лаба\_13.1, тут я не забыла, гы) у класса Money перегружены такие функции как: операции сравнения, ввода/вывода, прибавление (для accumulate) и умножение.
2. Во второй задаче (файл Лаба\_13.2) реализован адаптер котейнера - очередь с приоритетами (priority\_queue). У неё описаны компаратор для сортировки по возрастанию (т.к. очередь у нас с приоритетами), функции её заполнения, печати и копирования очереди в вектор для работы с алгоритмами STL.
3. В третьей задаче (файл Лаба\_13.3) реализован ассоциативный контейнер - множество с дубликатами. У него описаны функции его заполнения и печати. Так же стоит отметить, что в таком множестве значения сортируются автоматически и прямого доступа по индексу у него нет.

Решение

Код. Задача 1. Файл Лаба\_13.1

#include <iostream>

#include <vector>

#include <algorithm>

#include <numeric>

#include <functional>

#include <Windows.h>

#include <iomanip>

using namespace std;

// Класс Money

class Money {

private:

long rubles;

int kopecks;

public:

Money(long r = 0, int k = 0) : rubles(r), kopecks(k) { normalize(); }

void normalize() {

rubles += kopecks / 100;

kopecks %= 100;

if (kopecks < 0) {

rubles--;

kopecks += 100;

}

}

// Перегрузка операторов

bool operator>(const Money& other) const {

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

}

bool operator<(const Money& other) const {

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

}

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

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

}

Money operator\*(const Money& other) const {

long totalKopecks = (rubles \* 100 + kopecks) \* (other.rubles \* 100 + other.kopecks) / 100;

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

}

Money operator+(const Money& other) const {

return Money(rubles + other.rubles, kopecks + other.kopecks);

}

Money operator/(int divisor) const {

if (divisor == 0) {

cerr << "Ошибка: деление на ноль!\n";

return Money(0, 0);

}

long totalKopecks = rubles \* 100 + kopecks;

totalKopecks /= divisor;

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

}

// Перегрузка оператора вывода

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

os << m.rubles << "," << setw(2) << setfill('0') << m.kopecks << " руб.";

return os;

}

// Перегрузка оператора ввода

friend istream& operator>>(istream& in, Money& m)

{

do {

cout << "рубли: ";

in >> m.rubles;

} while (m.rubles < 0);

do {

cout << "копейки: ";

in >> m.kopecks;

} while (m.kopecks < 0);

if (m.kopecks >= 100) {

m.rubles += m.kopecks / 100;

m.kopecks = m.kopecks % 100;

}

return in;

}

};

// Функция для создания вектора

vector<Money> make\_money(int n) {

std::vector<Money> v;

if (n <= 0) throw invalid\_argument("Размер вектора должен быть положительным!");

srand(time(0));

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

long rubles = rand() % 100;

int kopecks = rand() % 100;

v.push\_back(Money(rubles, kopecks));

}

return v;

}

// Функция для печати вектора

void print(const std::vector<Money>& v) {

for (const auto& item : v)

std::cout << item << endl;

std::cout << endl;

}

// Функция для вычисления среднего арифметического

Money calculate\_average(const vector<Money>& v) {

if (v.empty()) {

cerr << "Ошибка: вектор пустой!\n";

return Money(0, 0);

}

Money sum = accumulate(v.begin(), v.end(), Money(0, 0));

return sum / static\_cast<int>(v.size());

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int n, pos;

std::cout << "Введите количество элементов: ";

std::cin >> n;

// 1. Создание и заполнение вектора

vector<Money> v = make\_money(n);

std::cout << "Исходный вектор:" << endl;

print(v);

// 2. Найти минимальный элемент и добавить его на заданную позицию

auto min\_it = std::min\_element(v.begin(), v.end());

if (min\_it != v.end()) {

std::cout << "Минимальный элемент: " << \*min\_it << std::endl;

std::cout << "Введите позицию для вставки (1-" << v.size() << "): ";

std::cin >> pos;

if (pos >= 0 && pos <= v.size()) {

v.insert(v.begin() + pos - 1, \*min\_it);

std::cout << "Вектор после вставки минимального элемента:" << endl;

print(v);

}

}

// 3. Найти элементы большие среднего арифметического и удалить их

Money avg = calculate\_average(v);

std::cout << "Среднее арифметическое: " << avg << std::endl;

v.erase(std::remove\_if(v.begin(), v.end(),

[&avg](const Money& m) { return m > avg; }), v.end());

std::cout << "Вектор после удаления элементов больших среднего:" << endl;

print(v);

// 4. Сортировка по возрастанию и убыванию

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

std::cout << "Вектор после сортировки по возрастанию:\n";

print(v);

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

std::cout << "Вектор после сортировки по убыванию:\n";

print(v);

// 5. Поиск элемента

Money search\_val;

std::cout << "\nВведите значение для поиска: " << endl;

std::cin >> search\_val;

auto found = std::find(v.begin(), v.end(), search\_val);

if (found != v.end()) cout << "Элемент найден на позиции: " << (found - v.begin()) + 1 << endl;

else cout << "Элемент не найден\n";

// Подсчет элементов с заданным значением

int count = std::count(v.begin(), v.end(), search\_val);

std::cout << "Количество элементов с таким значением: " << count << endl;

// 6. Каждый элемент домножить на максимальный элемент

auto max\_it = std::max\_element(v.begin(), v.end());

if (max\_it != v.end()) {

std::cout << "\nМаксимальный элемент: " << \*max\_it << endl;

std::for\_each(v.begin(), v.end(),

[&max\_it](Money& m) { m = m \* (\*max\_it); });

std::cout << "Вектор после умножения на максимальный элемент:" << endl;

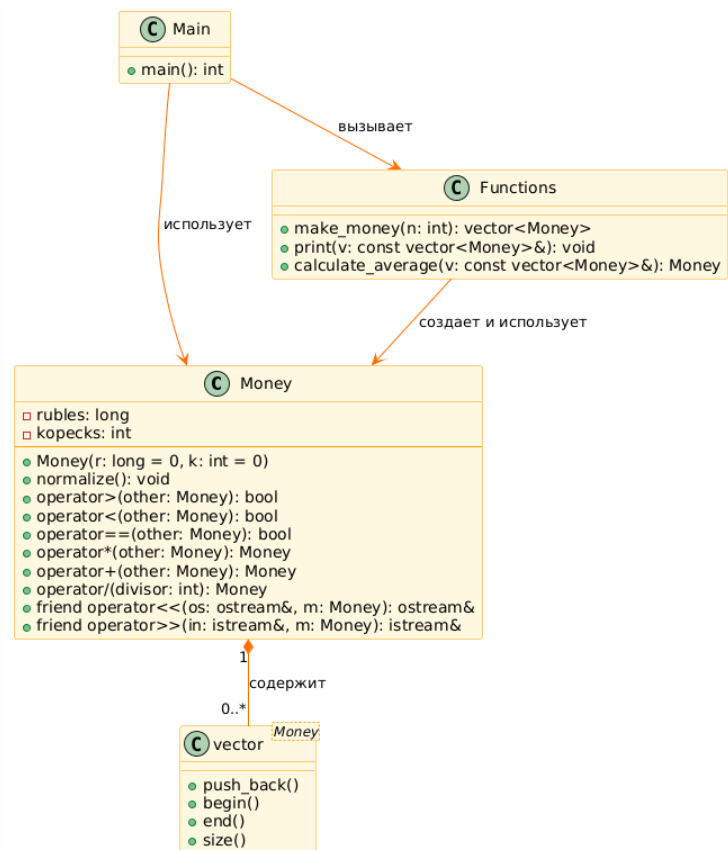
print(v);

}

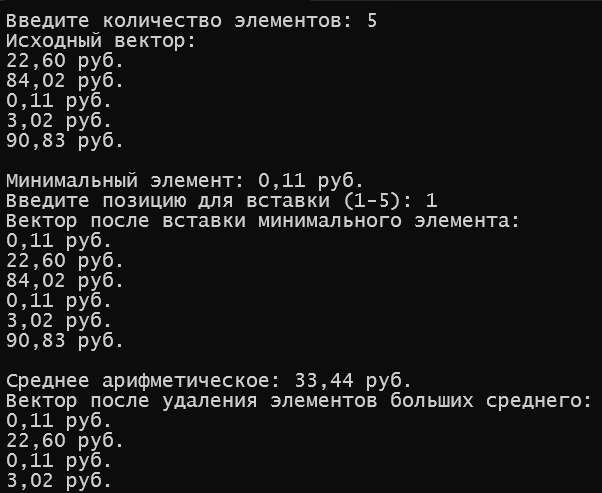
return 0;

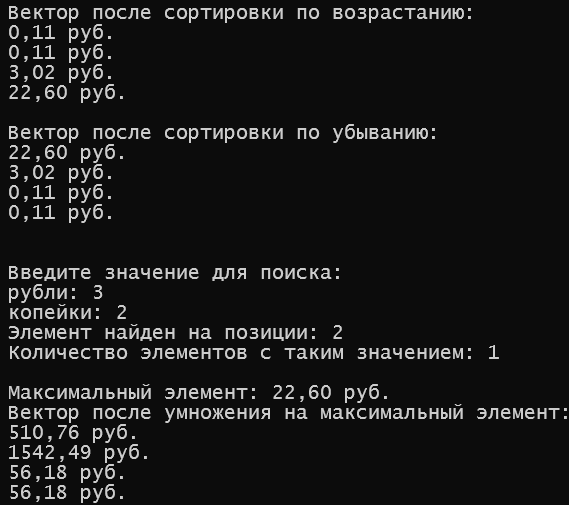
}

UML-диаграмма



Результат работы программы:





Код. Задача 2. Файл Лаба\_13.2

#include <iostream>

#include <queue>

#include <vector>

#include <algorithm>

#include <numeric>

#include <functional>

#include <Windows.h>

#include <iomanip>

#include <ctime>

#include <stdexcept>

using namespace std;

class Money

{

private:

long rubles;

int kopecks;

public:

Money(long r = 0, int k = 0) : rubles(r), kopecks(k) { normalize(); }

void normalize() {

rubles += kopecks / 100;

kopecks %= 100;

if (kopecks < 0) {

rubles--;

kopecks += 100;

}

}

bool operator>(const Money& other) const {

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

}

bool operator<(const Money& other) const {

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

}

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

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

}

Money operator\*(const Money& other) const {

long totalKopecks = (rubles \* 100 + kopecks) \* (other.rubles \* 100 + other.kopecks) / 100;

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

}

Money operator+(const Money& other) const {

return Money(rubles + other.rubles, kopecks + other.kopecks);

}

Money operator/(int divisor) const {

if (divisor == 0) throw runtime\_error("Деление на ноль!");

long totalKopecks = rubles \* 100 + kopecks;

totalKopecks /= divisor;

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

}

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

os << m.rubles << "," << setw(2) << setfill('0') << m.kopecks << " руб.";

return os;

}

friend istream& operator>>(istream& in, Money& m) {

do {

cout << "рубли: ";

in >> m.rubles;

} while (m.rubles < 0);

do {

cout << "копейки: ";

in >> m.kopecks;

} while (m.kopecks < 0);

m.normalize();

return in;

}

};

// Компаратор для сортировки по возрастанию (для priority\_queue)

struct MoneyAscComparator {

bool operator()(const Money& a, const Money& b) const {

return a > b; }

};

// Функция для заполнения очереди с приоритетами

priority\_queue<Money, vector<Money>, MoneyAscComparator> fill\_priority\_queue(int n) {

priority\_queue<Money, vector<Money>, MoneyAscComparator> pq;

if (n <= 0) throw invalid\_argument("Размер очереди должен быть положительным!");

srand(time(0));

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

long rubles = rand() % 100;

int kopecks = rand() % 100;

pq.push(Money(rubles, kopecks));

}

return pq;

}

// Функция для печати очереди с приоритетами (с разрушающим чтением)

void print\_qu(priority\_queue<Money, vector<Money>, MoneyAscComparator> pq) {

cout << "Содержимое очереди с приоритетами (от наибольшего к наименьшему):\n";

while (!pq.empty()) {

cout << pq.top() << endl;

pq.pop();

}

cout << endl;

}

// Функция для копирования очереди в вектор

vector<Money> priority(priority\_queue<Money, vector<Money>, MoneyAscComparator> pq) {

vector<Money> v;

while (!pq.empty()) {

v.push\_back(pq.top());

pq.pop();

}

return v;

}

// Функция для копирования вектора в очередь с приоритетами

priority\_queue<Money, vector<Money>, MoneyAscComparator> vector\_to\_priority\_queue(const vector<Money>& v) {

priority\_queue<Money, vector<Money>, MoneyAscComparator> pq;

for (const auto& item : v)

pq.push(item);

return pq;

}

// Функция для вычисления среднего арифметического

Money average(const vector<Money>& v) {

if (v.empty()) throw runtime\_error("Вектор пустой!");

Money sum = accumulate(v.begin(), v.end(), Money(0, 0));

return sum / static\_cast<int>(v.size());

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

try {

int n, pos;

cout << "Введите количество элементов: ";

cin >> n;

// 1. Создание и заполнение очереди с приоритетами

auto pq = fill\_priority\_queue(n);

cout << "Исходная очередь с приоритетами:\n";

print\_qu(pq);

// Для работы с алгоритмами STL преобразуем очередь в вектор

vector<Money> v = priority(pq);

// 2. Найти минимальный элемент и добавить его на заданную позицию

auto min\_it = min\_element(v.begin(), v.end());

if (min\_it != v.end()) {

cout << "Минимальный элемент: " << \*min\_it << endl;

cout << "Введите позицию для вставки (1-" << v.size() << "): ";

cin >> pos;

if (pos >= 0 && pos <= v.size()) {

v.insert(v.begin() + pos - 1, \*min\_it);

cout << "Вектор после вставки минимального элемента:\n";

for (const auto& item : v) {

cout << item << endl;

}

cout << endl;

}

}

// 3. Найти элементы большие среднего арифметического и удалить их

Money avg = average(v);

cout << "Среднее арифметическое: " << avg << endl;

v.erase(remove\_if(v.begin(), v.end(),

[&avg](const Money& m) { return m > avg; }), v.end());

cout << "Вектор после удаления элементов больших среднего:\n";

for (const auto& item : v) {

cout << item << endl;

}

cout << endl;

// 4. Сортировка по возрастанию и убыванию

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

cout << "Вектор после сортировки по возрастанию:\n";

for (const auto& item : v) {

cout << item << endl;

}

cout << endl;

sort(v.begin(), v.end(), greater<Money>());

cout << "Вектор после сортировки по убыванию:\n";

for (const auto& item : v) {

cout << item << endl;

}

cout << endl;

// 5. Поиск элемента

Money search\_val;

cout << "Введите значение для поиска: " << endl;

cin >> search\_val;

auto found = find(v.begin(), v.end(), search\_val);

if (found != v.end())

cout << "Элемент найден на позиции: " << (found - v.begin()) + 1 << endl;

else

cout << "Элемент не найден\n";

// Подсчет элементов с заданным значением

int c = count(v.begin(), v.end(), search\_val);

cout << "Количество элементов с таким значением: " << c << endl;

// 6. Каждый элемент домножить на максимальный элемент

auto max\_it = max\_element(v.begin(), v.end());

if (max\_it != v.end()) {

cout << "\nМаксимальный элемент: " << \*max\_it << endl;

for\_each(v.begin(), v.end(),

[&max\_it](Money& m) { m = m \* (\*max\_it); });

cout << "Вектор после умножения на максимальный элемент:\n";

for (const auto& item : v) {

cout << item << endl;

}

cout << endl;

}

}

catch (const exception& e) {

cerr << "Ошибка: " << e.what() << endl;

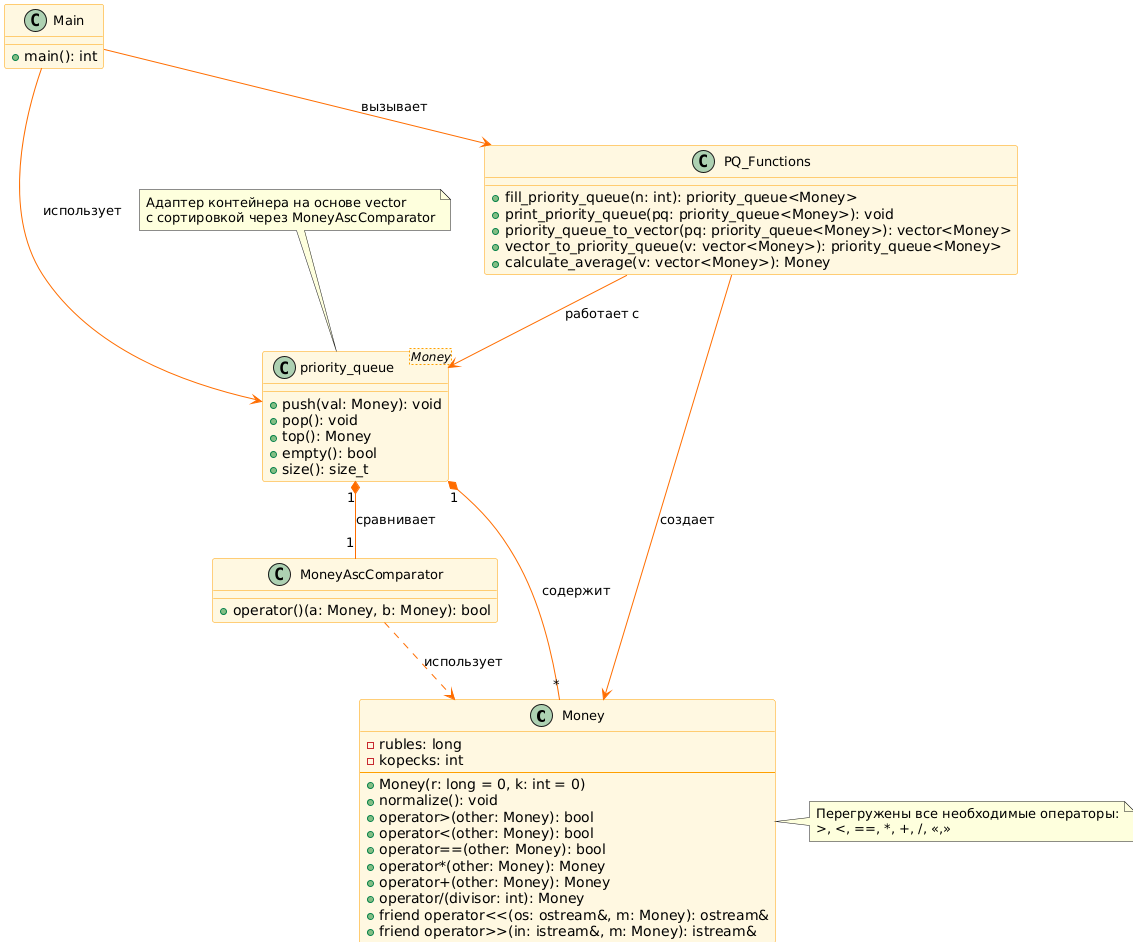
return 1;

}

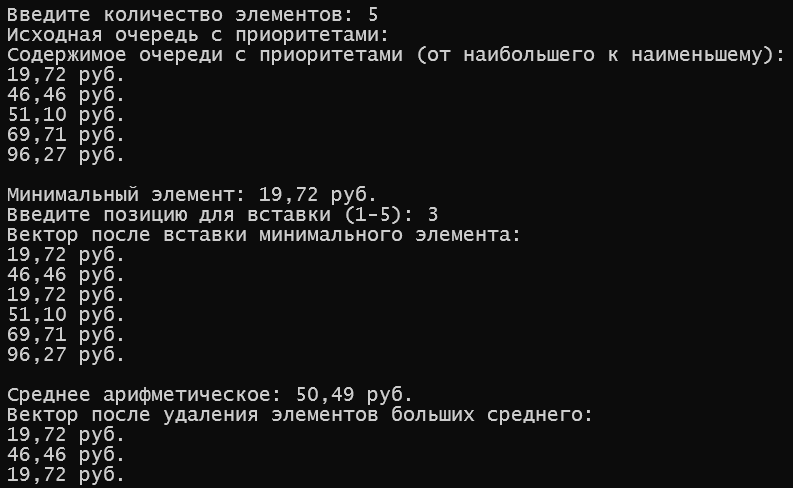
return 0;

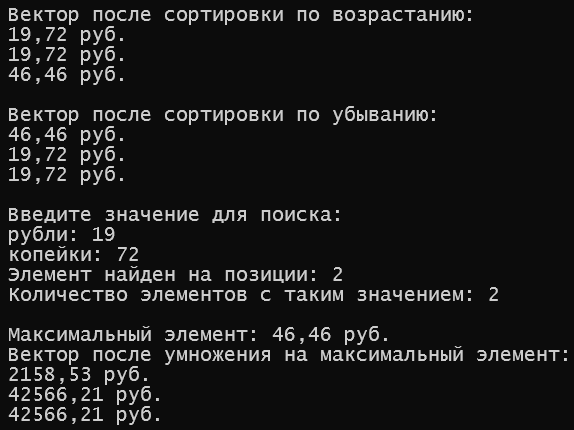
}

UML-диаграмма



Результат работы программы:





Код. Задача 3. Файл Лаба\_13.3

#include <iostream>

#include <set>

#include <vector>

#include <algorithm>

#include <numeric>

#include <functional>

#include <Windows.h>

#include <iomanip>

#include <ctime>

#include <stdexcept>

using namespace std;

class Money {

private:

long rubles;

int kopecks;

public:

Money(long r = 0, int k = 0) : rubles(r), kopecks(k) { normalize(); }

void normalize() {

rubles += kopecks / 100;

kopecks %= 100;

if (kopecks < 0) {

rubles--;

kopecks += 100;

}

}

bool operator>(const Money& other) const {

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

}

bool operator<(const Money& other) const {

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

}

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

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

}

Money operator\*(const Money& other) const {

long totalKopecks = (rubles \* 100 + kopecks) \* (other.rubles \* 100 + other.kopecks) / 100;

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

}

Money operator+(const Money& other) const {

return Money(rubles + other.rubles, kopecks + other.kopecks);

}

Money operator/(int divisor) const {

if (divisor == 0) throw runtime\_error("Деление на ноль!");

long totalKopecks = rubles \* 100 + kopecks;

totalKopecks /= divisor;

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

}

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

os << m.rubles << "," << setw(2) << setfill('0') << m.kopecks << " руб.";

return os;

}

friend istream& operator>>(istream& in, Money& m) {

do {

cout << "рубли: ";

in >> m.rubles;

} while (m.rubles < 0);

do {

cout << "копейки: ";

in >> m.kopecks;

} while (m.kopecks < 0);

m.normalize();

return in;

}

};

// Функция для заполнения multiset

multiset<Money> fill\_multiset(int n) {

multiset<Money> ms;

if (n <= 0) throw invalid\_argument("Размер multiset должен быть положительным!");

srand(time(0));

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

long rubles = rand() % 100;

int kopecks = rand() % 100;

ms.insert(Money(rubles, kopecks));

}

return ms;

}

void print(const multiset<Money>& ms) {

for (const auto& item : ms)

cout << item << endl;

cout << endl;

}

// Функция для вычисления среднего арифметического

Money average(const multiset<Money>& ms) {

if (ms.empty()) throw runtime\_error("Множество пустое!");

Money sum = accumulate(ms.begin(), ms.end(), Money(0, 0));

return sum / static\_cast<int>(ms.size());

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

try {

int n, pos;

cout << "Введите количество элементов: ";

cin >> n;

// 1. Создание и заполнение multiset

multiset<Money> ms = fill\_multiset(n);

cout << "Исходное множество (сортирует по возрастанию):\n";

print(ms);

// 2. Найти минимальный элемент и добавить его на заданную позицию

if (!ms.empty()) {

Money min\_val = \*ms.begin(); // В multiset элементы отсортированы

cout << "Минимальный элемент: " << min\_val << endl;

// Для вставки на позицию преобразуем в вектор

vector<Money> v(ms.begin(), ms.end());

cout << "Введите позицию для вставки (1-" << v.size() << "): ";

cin >> pos;

if (pos >= 0 && pos <= v.size()) {

v.insert(v.begin() + pos - 1, min\_val);

ms = multiset<Money>(v.begin(), v.end());

cout << "Множество после вставки минимального элемента:\n";

cout << "(это множество с приоритетами, поэтому вставилось как есть)" << endl;

print(ms);

}

}

// 3. Найти элементы большие среднего арифметического и удалить их

Money avg = average(ms);

cout << "Среднее арифметическое: " << avg << endl;

// Удаление элементов через временный вектор

vector<Money> v(ms.begin(), ms.end());

v.erase(remove\_if(v.begin(), v.end(),

[&avg](const Money& m) { return m > avg; }), v.end());

ms = multiset<Money>(v.begin(), v.end());

cout << "Множество после удаления элементов больших среднего:\n";

print(ms);

// 4. Сортировка по убыванию (создаем новый multiset с компаратором)

multiset<Money, greater<Money>> ms\_desc(ms.begin(), ms.end());

cout << "Множество после сортировки по убыванию:\n";

for (const auto& item : ms\_desc) {

cout << item << endl;

}

// 5. Поиск элемента

Money search\_val;

cout << "\nВведите значение для поиска: " << endl;

cin >> search\_val;

auto found = ms.find(search\_val);

if (found != ms.end()) {

cout << "Элемент найден, количество таких элементов: " << ms.count(search\_val) << endl;

cout << "(у множества с приоритетами нет прямого доступа по индексу)" << endl;

}

else cout << "Элемент не найден\n";

// 6. Каждый элемент домножить на максимальный элемент

if (!ms.empty()) {

Money max\_val = \*ms.rbegin(); // Максимальный элемент в конце

cout << "\nМаксимальный элемент: " << max\_val << endl;

vector<Money> v\_mult(ms.begin(), ms.end());

for\_each(v\_mult.begin(), v\_mult.end(),

[&max\_val](Money& m) { m = m \* max\_val; });

ms = multiset<Money>(v\_mult.begin(), v\_mult.end());

cout << "Множество после умножения на максимальный элемент:\n";

print(ms);

}

}

catch (const exception& e) {

cerr << "Ошибка: " << e.what() << endl;

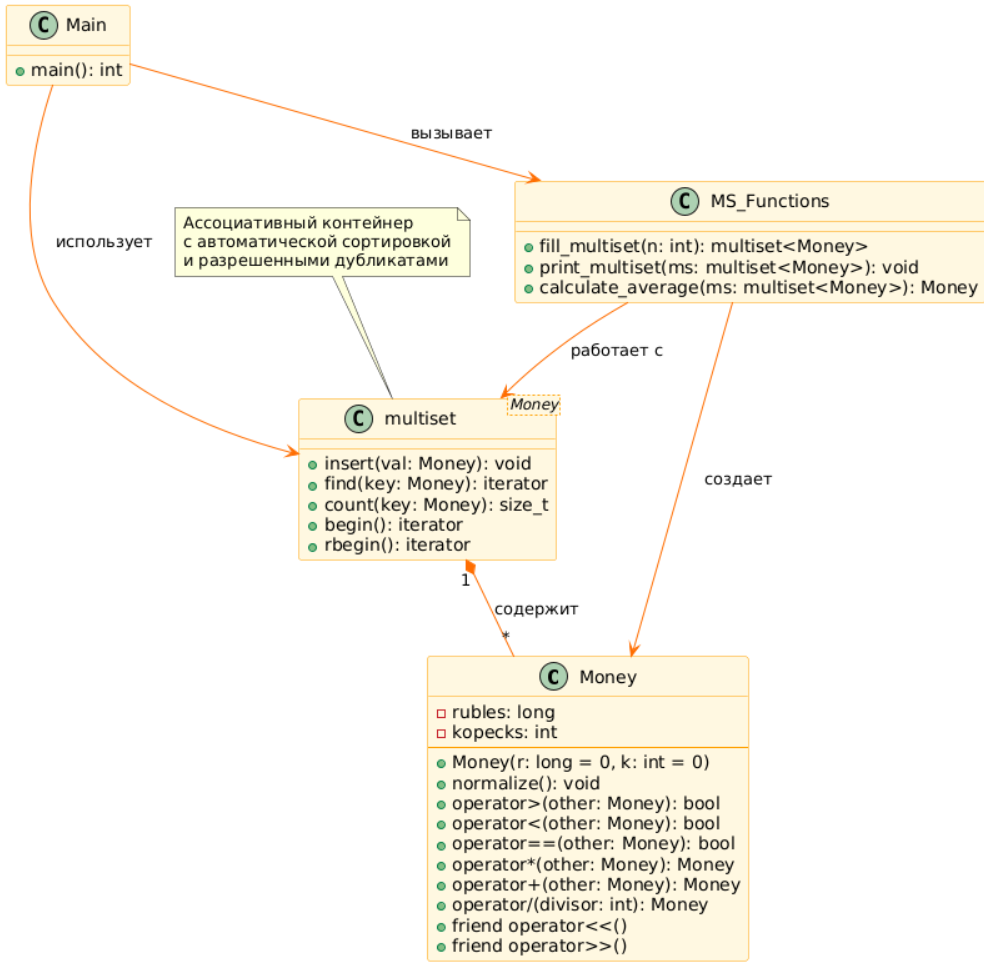
return 1;

}

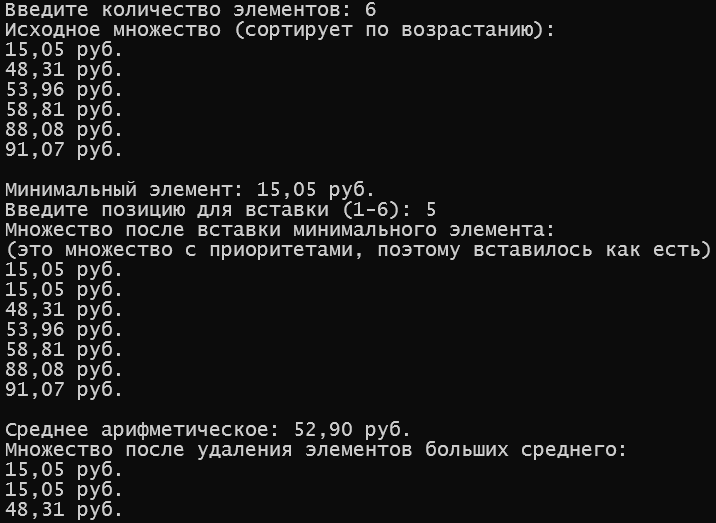
return 0;

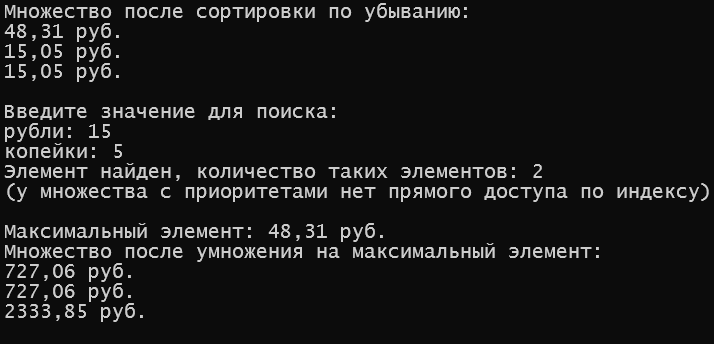
}

UML-диаграмма



Результат работы программы:





Контрольные вопросы: (их нет в методичке)