Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

ПНИПУ

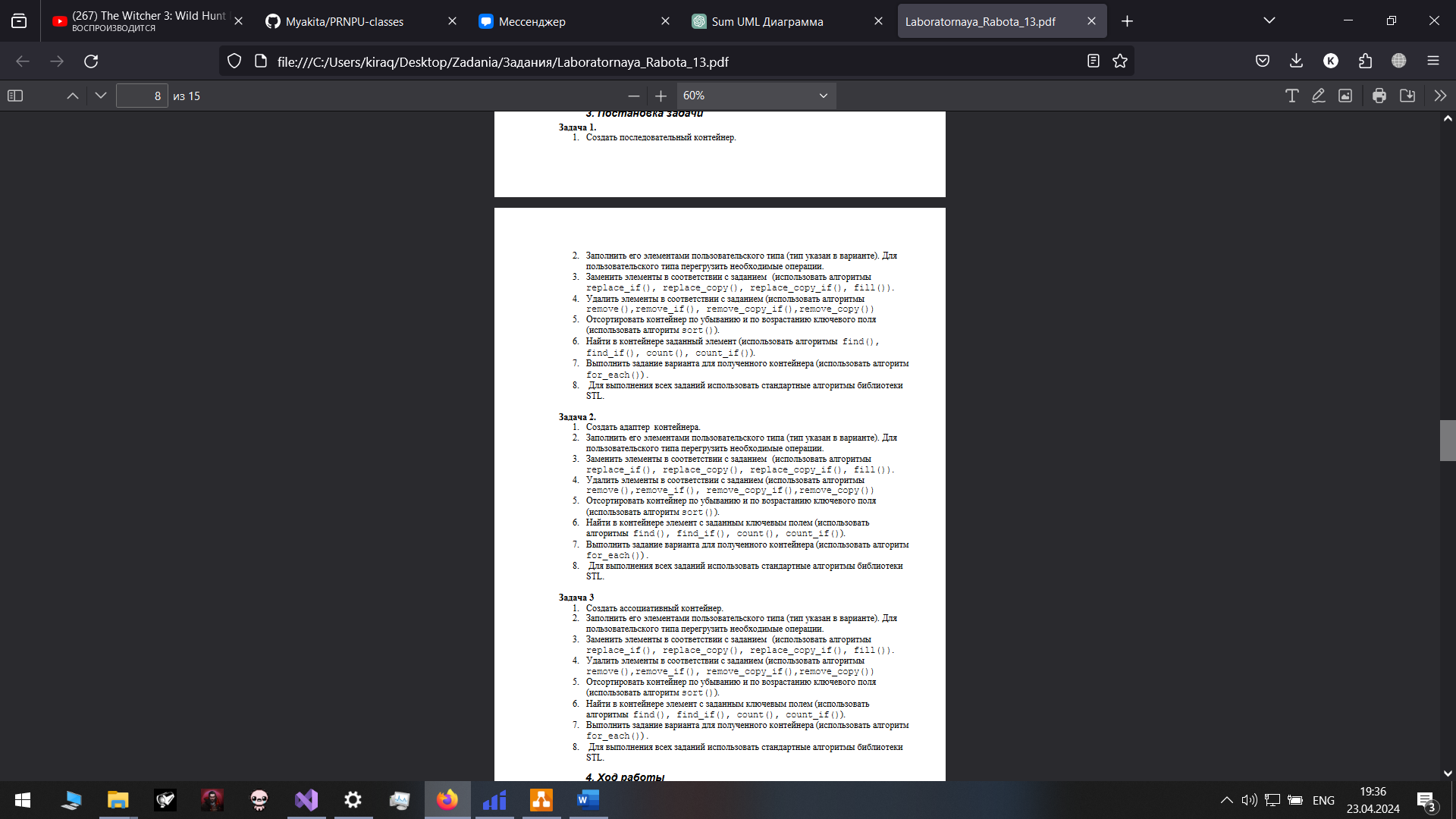
Лабораторная работа  
«Библиотеки STL»

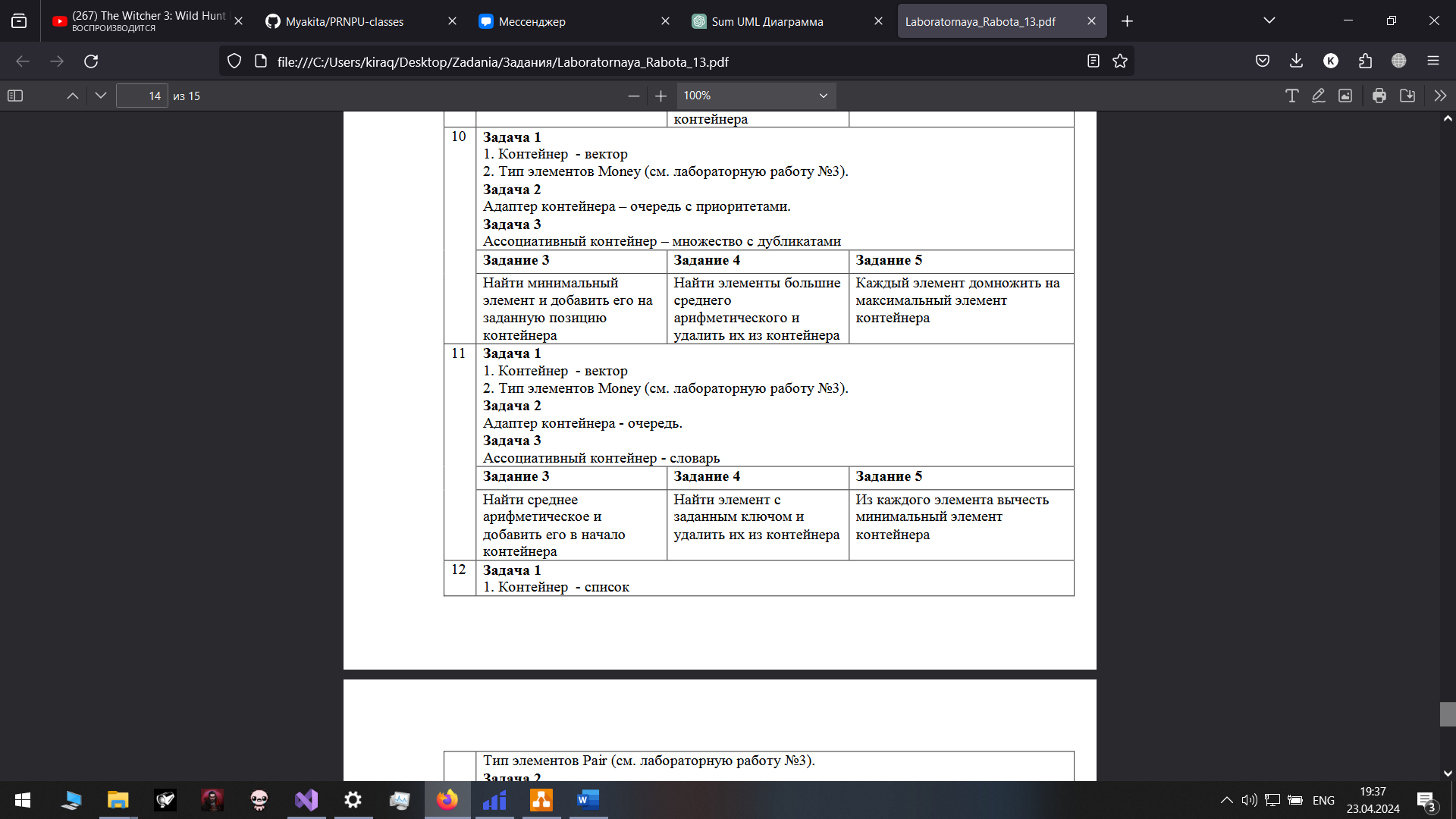
Выполнил:   
студент группы РИС-23-1б   
Мокрушин Никита Дмитриевич

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

Пермь 2024 г.

Постановка задачи:





Анализ задачи:

**Алгоритм задачи по пунктам:**

1. **Инициализация и ввод данных:**
   * Создание вектора m с заданным пользователем количеством элементов.
   * Генерация случайных значений для элементов вектора.
2. **Вычисление среднего арифметического:**
   * Вычисление суммы всех элементов вектора (в рублях и копейках).
   * Расчет среднего арифметического.
   * Создание нового объекта Money с полученным средним значением.
   * Добавление нового объекта в начало вектора.
3. **Удаление элементов по ключу:**
   * Ввод ключа для удаления.
   * Перебор всех элементов вектора.
   * Если текущий элемент равен ключу, удаляем его из вектора.
4. **Вычитание минимума из всех элементов:**
   * Нахождение минимального элемента в векторе.
   * Вычитание минимального элемента из всех элементов вектора.
5. **Генерация и обработка очереди:**
   * Создание очереди qm с заданным пользователем количеством элементов.
   * Генерация случайных значений для элементов очереди.
   * Добавление среднего арифметического в начало очереди.
   * Удаление элементов по ключу.
   * Вычитание минимального элемента из всех элементов очереди.
6. **Генерация и обработка словаря:**
   * Создание словаря moneyMap с заданным пользователем количеством элементов.
   * Генерация случайных значений для элементов словаря.
   * Добавление среднего арифметического в начало словаря.
   * Удаление элементов по ключу.
   * Вычитание минимального элемента из всех элементов словаря.

Код:

#include <iostream>

#include <vector>

#include <queue>

#include <list>

#include <map>

#include <stack>

using namespace std;

class Money {

private:

long rubles;

int kopecks;

public:

Money() : rubles(0), kopecks(0) {}

Money(long r, int k) : rubles(r), kopecks(k) {}

Money(const Money& other) : rubles(other.rubles), kopecks(other.kopecks) {}

~Money() {}

long getRubles() const { return rubles; }

int getKopecks() const { return kopecks; }

void setRubles(long r) { rubles = r; }

void setKopecks(int k) { kopecks = k; }

Money& operator=(const Money& other) {

if (this != &other) {

rubles = other.rubles;

kopecks = other.kopecks;

}

return \*this;

}

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

os << money.rubles << "," << money.kopecks;

return os;

}

friend istream& operator>>(istream& is, Money& money) {

is >> money.rubles >> money.kopecks;

return is;

}

bool operator<(const Money& other) const {

if (rubles < other.rubles)

return true;

else if (rubles == other.rubles && kopecks < other.kopecks)

return true;

return false;

}

bool operator>(const Money& other) const {

if (rubles > other.rubles)

return true;

else if (rubles == other.rubles && kopecks > other.kopecks)

return true;

return false;

}

bool operator==(const Money& other)

{

if ((rubles == other.rubles) && (kopecks == other.kopecks))

{

return true;

}

else

{

return false;

}

}

Money operator++(int) {

Money temp(\*this);

kopecks++;

if (kopecks >= 100) {

rubles++;

kopecks -= 100;

}

return temp;

}

Money& operator++() {

kopecks++;

if (kopecks >= 100) {

rubles++;

kopecks -= 100;

}

return \*this;

}

Money operator-(const Money& other) const {

Money result;

long totalKopecks = rubles \* 100 + kopecks;

long otherTotalKopecks = other.rubles \* 100 + other.kopecks;

if (totalKopecks < otherTotalKopecks) {

return Money();

}

long difference = totalKopecks - otherTotalKopecks;

result.rubles = difference / 100;

result.kopecks = difference % 100;

return result;

}

};

ostream& operator<<(ostream& os, vector<Money>& m)

{

for\_each(m.begin(), m.end(), [](Money& m1) {cout << m1 << endl; });

return os;

}

ostream& operator<<(ostream& os, const queue<Money, list<Money>>& q) {

queue<Money, list<Money>> tempQ = q;

while (!tempQ.empty()) {

Money user = tempQ.front();

cout << user << endl;

tempQ.pop();

}

return os;

}

void randomizeQ(Money& m)

{

long x = rand() % 1000;

int y = rand() % 100;

m = Money(x, y);

}

void generateQ(queue<Money, list<Money>>& q, int qsize)

{

Money qwe(0, 0);

for (int i = 0; i < qsize; ++i)

{

randomizeQ(qwe);

q.push(qwe);

}

}

Money min(vector<Money> m,int vsize)

{

Money min = m[0];

for (int i = 1; i < vsize; i++)

{

if (m[i] < min)

{

min = m[i];

}

}

return min;

}

void generateMap(map<int, Money>& moneyMap, int size) {

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

long rubles = rand() % 1000;

int kopecks = rand() % 100;

moneyMap[i] = Money(rubles, kopecks);

}

}

ostream& operator<<(ostream& os, const map<int, Money>& moneyMap) {

for (const auto& pair : moneyMap) {

os << pair.first << ": " << pair.second << endl;

}

return os;

}

Money findAverage(const queue<Money, list<Money>>& q) {

if (q.empty()) {

return Money(0, 0);

}

queue<Money, list<Money>> tempQ = q;

int size = tempQ.size();

long sumKopecks = 0;

while (!tempQ.empty()) {

Money front = tempQ.front();

sumKopecks += front.getRubles() \* 100 + front.getKopecks();

tempQ.pop();

}

long totalKopecks = sumKopecks / size;

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

return average;

}

void addAverageToBeginning(queue<Money, list<Money>>& q) {

if (q.empty()) {

return;

}

Money average = findAverage(q);

int size = q.size();

q.push(average);

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

q.push(q.front());

q.pop();

}

}

void removeByKey(queue<Money, list<Money>>& q, const Money& key) {

int size = q.size();

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

Money front = q.front();

q.pop();

if (!(front == key)) {

q.push(front);

}

}

}

void subtractMin(queue<Money, list<Money>>& q) {

if (q.empty()) {

return;

}

Money min = q.front();

int size = q.size();

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

Money current = q.front();

q.pop();

if (current < min) {

min = current;

}

q.push(current);

}

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

Money current = q.front();

q.pop();

q.push(current - min);

}

}

Money findAverageMap(const map<int, Money>& moneyMap) {

if (moneyMap.empty()) {

return Money(0, 0);

}

long sumKopecks = 0;

for (const auto& pair : moneyMap) {

sumKopecks += pair.second.getRubles() \* 100 + pair.second.getKopecks();

}

int size = moneyMap.size();

long totalKopecks = sumKopecks / size;

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

return average;

}

void addAverageToBeginning(map<int, Money>& moneyMap) {

if (moneyMap.empty()) {

return;

}

Money average = findAverageMap(moneyMap);

map<int, Money> tempMap;

tempMap[0] = average;

int index = 1;

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

tempMap[index++] = it->second;

}

moneyMap = tempMap;

}

void removeByKey(map<int, Money>& moneyMap, const Money& key) {

auto it = moneyMap.begin();

while (it != moneyMap.end()) {

if (it->second == key) {

it = moneyMap.erase(it);

}

else {

++it;

}

}

}

void subtractMin(map<int, Money>& moneyMap) {

if (moneyMap.empty()) {

return;

}

Money min = moneyMap.begin()->second;

for (const auto& pair : moneyMap) {

if (pair.second < min) {

min = pair.second;

}

}

for (auto& pair : moneyMap) {

pair.second = pair.second - min;

}

}

int main() {

setlocale(0, "");

system("chcp 1251>nul");

srand(time(NULL));

cout << "Введите кол-во элементов в векторе: ";

int count;

cin >> count;

vector<Money> m(count);

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

m[i].setKopecks(rand() % 100);

m[i].setRubles(rand() % 1000);

}

cout << "------Вектор------ " << endl;

cout << m;

cout << "----Конец вектора----" << endl;

long sumRubles = 0;

int sumKopecks = 0;

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

sumRubles += m[i].getRubles();

sumKopecks += m[i].getKopecks();

}

sumRubles += sumKopecks / 100;

sumKopecks %= 100;

int n = count;

int sumR = sumRubles / n;

m.insert(m.begin(), Money(sumR, sumKopecks));

cout << "Вектор после добавления среднего арифметического: " << endl;

cout << "------Вектор------ " << endl;

cout << m;

cout << "----Конец вектора----" << endl;

Money key;

cout << "Введите ключ для удаления: ";

cin >> key;

vector<Money>::iterator it = m.begin();

while (it != m.end()) {

if (\*it == key) {

it = m.erase(it);

}

else {

++it;

}

}

cout << "Вектор после удаления ключа: " << endl;

cout << "------Вектор------ " << endl;

cout << m;

cout << "----Конец вектора----" << endl;

Money minim = min(m, m.size());

for (int i = 0; i < m.size(); ++i) {

m[i] = m[i] - minim;

}

cout << "Вектор после вычитания минимума: " << endl;

cout << "------Вектор------ " << endl;

cout << m;

cout << "----Конец вектора----" << endl;

cout << "Введите кол-во элементов в очереди: ";

int count1;

cin >> count1;

queue<Money, list<Money>> qm;

generateQ(qm, count1);

cout << "Очередь: " << endl;

cout << qm;

addAverageToBeginning(qm);

cout << "Очередь после вставки среднего арифметического: " << endl;

cout << qm;

Money key1;

cout << "Введите ключ для удаления: ";

cin >> key1;

removeByKey(qm, key1);

cout << "Очередь после удаления элементов по ключу: " << endl;

cout << qm;

subtractMin(qm);

cout << "Очередь после вычитания минимального элемента: " << endl;

cout << qm;

cout << "Введите кол-во элементов в словаре: ";

int count2;

cin >> count2;

map<int, Money> moneyMap;

generateMap(moneyMap, count2);

cout << "Словарь: " << endl;

cout << moneyMap;

addAverageToBeginning(moneyMap);

cout << "Словарь после добавления среднего арифметического: " << endl;

cout << moneyMap;

Money key2;

cout << "Введите ключ для удаления: ";

cin >> key2;

removeByKey(moneyMap, key2);

cout << "Словарь после удаления элементов по ключу: " << endl;

cout << moneyMap;

subtractMin(moneyMap);

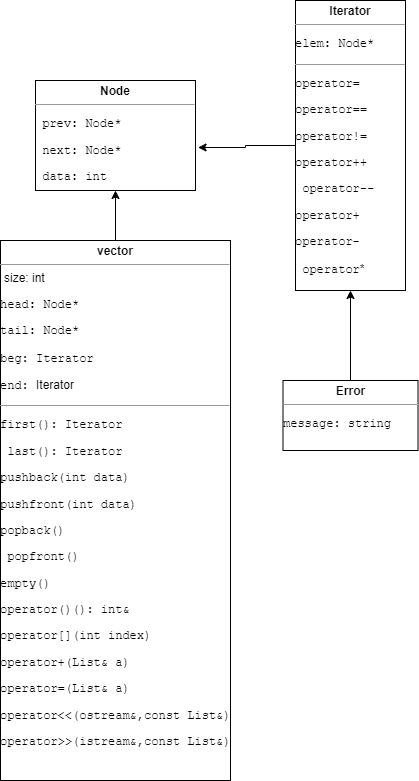
cout << "Словарь после вычитания минимального элемента: " << endl;

cout << moneyMap;

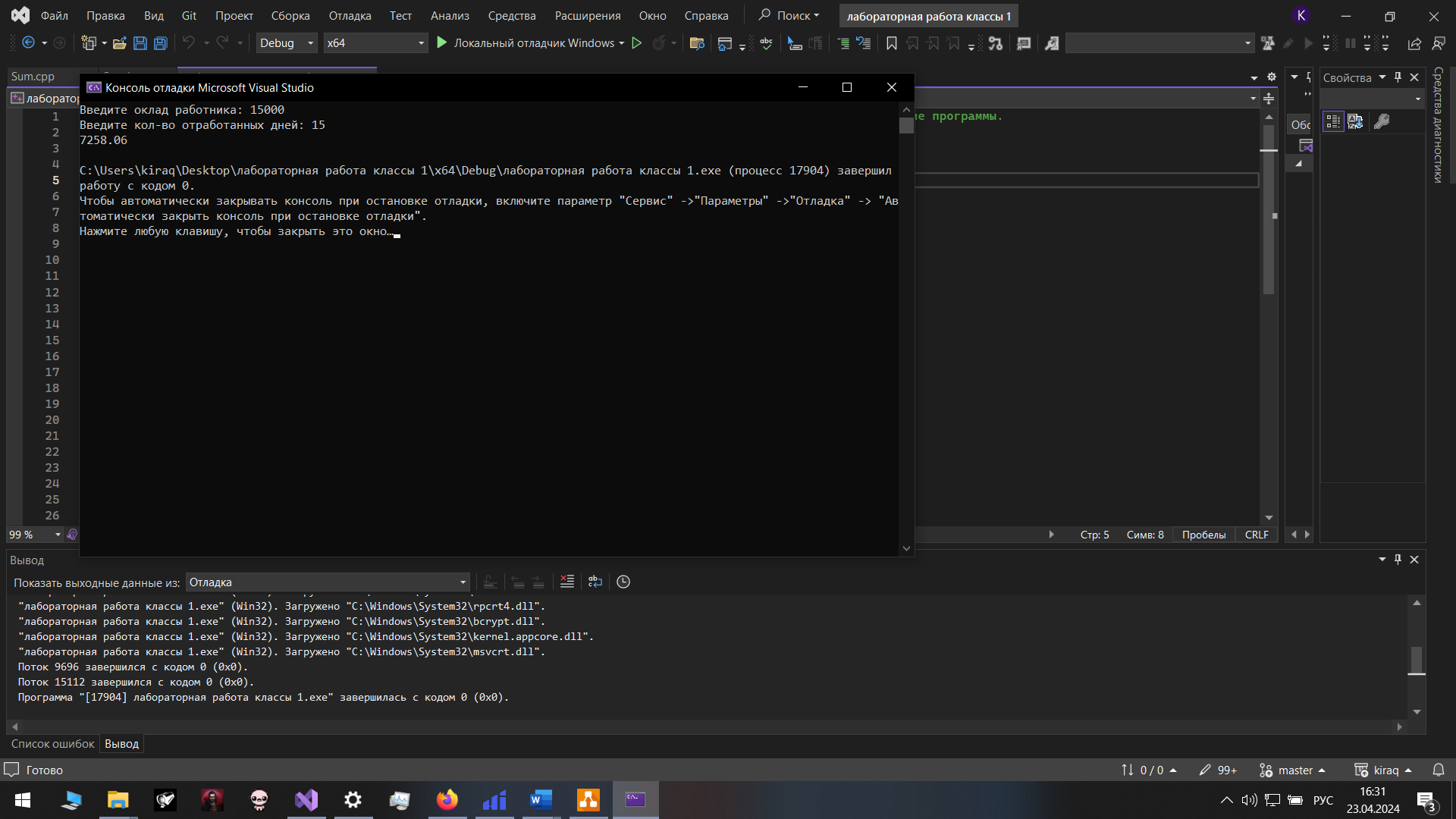
return 0;

}

UML Диаграммы:



Вывод:



программа работает корректно и выдаёт требуемый результат.