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 №11

# По теме **«**Последовательные контейнеры 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 стандартного типа.
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 последовательный контейнер.
2. Заполнить его элементами пользовательского типа. Для него 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 последовательный 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.

Задача 4

1. Создать адаптер контейнера.
2. Заполнить его элементами пользовательского типа. Для него 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 5

1. Создать параметризированный класс, используя в качестве контейнера адаптер контейнера.
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.

Задание:

**Задача 1**

1. Контейнер - вектор
2. Тип элементов - float

**Задача 2**

Тип элементов Money (см. лабораторную работу №3).

**Задача 3**

Параметризированный класс – Вектор (см. лабораторную работу №7)

**Задача 4**

Адаптер контейнера – очередь с приоритетами.

**Задача 5**

Параметризированный класс – Вектор Адаптер контейнера - очередь с приоритетами.

|  |  |  |
| --- | --- | --- |
| **Задание 3** | **Задание 4** | **Задание 5** |
| 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. Все задания разбиты по файлам, к каждому есть пример работы программы.
2. Стоит уточнить, что сам priority\_queue и его адаптер не поддерживают прямое удаление/вставку по позициям, поэтому задание 3 немного изменено, а для модификации элементов используется временный вектор.
3. Во многих задачах были перегружены такие операции, как вывод, сравнение и умножение.

Решение

Код. Задача 1. Файл Лаба\_11.cpp

#include <iostream>

#include <vector>

#include <algorithm>

#include <numeric>

#include <Windows.h>

using namespace std;

typedef vector<float> FloatVec;

FloatVec make\_vector(int n) {

FloatVec v;

srand(time(0));

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

v.push\_back(rand() % 100 / 10.0f);

}

return v;

}

void print\_vector(const FloatVec& v) {

for (float x : v) cout << x << " ";

cout << endl;

}

// Задание 3: Вставка минимума на позицию

// Функция для нахождения минимального элемента

float find\_min(const FloatVec& v) {

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

return \*min\_element(v.begin(), v.end());

}

// Функция для добавления элемента на позицию

void insert\_at(FloatVec& v, float el, int pos) {

if (pos < 0 || pos > v.size()) throw out\_of\_range("Неверная позиция!");

v.insert(v.begin() + pos - 1, el);

}

// Задание 4: Удаление элементов > среднего

void remove\_above\_avg(FloatVec& v) {

if (v.empty()) return;

float avg = accumulate(v.begin(), v.end(), 0.0f) / v.size();

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

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

[avg](float x) { return x > avg; }),

v.end());

}

// Задание 5: Умножение на максимум

void multiply\_by\_max(FloatVec& v) {

if (v.empty()) return;

float max\_val = \*max\_element(v.begin(), v.end());

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

for (float& x : v) x \*= max\_val;

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

// 1. Создание и заполнение вектора

int n;

cout << "Введите размер вектора: ";

cin >> n;

FloatVec v = make\_vector(n);

cout << "Исходный вектор: ";

print\_vector(v)

// 3. Нахождение минимального элемента

float min\_val = find\_min(v);

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

// Вставка минимума на заданную позицию

int pos;

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

cin >> pos;

insert\_at(v, min\_val, pos);

cout << "После вставки минимума: "; print\_vector(v);

// Задание 4

remove\_above\_avg(v);

cout << "После удаления > среднего: "; print\_vector(v);

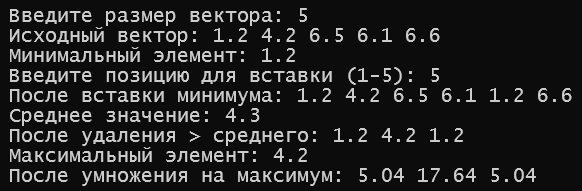
// Задание 5

multiply\_by\_max(v);

cout << "После умножения на максимум: "; print\_vector(v);

}

Результат работы программы:



Код. Задача 2. Файл Лаба\_11.2.cpp

#include <iostream>

#include <vector>

#include <algorithm>

#include <numeric>

#include <Windows.h>

#include <iomanip>

#include <string>

using namespace std;

// Пользовательский тип Money

class Money {

private:

int rubles;

int kopecks;

void normalize() {

rubles += kopecks / 100;

kopecks %= 100;

}

public:

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

normalize();

}

// Для сравнения

bool operator<(const Money& other) const {

return (rubles < other.rubles) ||

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

}

// Для умножения

Money operator\*(float multiplier) const {

float total = rubles + kopecks / 100.0f;

total \*= multiplier;

return Money(static\_cast<int>(total),

static\_cast<int>(roundf((total - static\_cast<int>(total)) \* 100)));

}

// Для вывода

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

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

return os;

}

// Для сложения (нужно для accumulate)

Money operator+(const Money& other) const {

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

}

// Для вычисления среднего

float toFloat() const {

return rubles + kopecks / 100.0f;

}

};

typedef vector<Money> MoneyVec;

// Создание вектора

MoneyVec make\_money\_vector(int n) {

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

MoneyVec v;

srand(time(0));

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

v.push\_back(Money(rand() % 100, rand() % 100));

}

return v;

}

// Печать вектора

void print\_money\_vector(const MoneyVec& v) {

for (const auto& m : v) cout << m << " ";

cout << endl;

}

// Задание 3: Вставка минимума на позицию

Money find\_min(const MoneyVec& v) {

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

return \*min\_element(v.begin(), v.end());

}

void insert\_at(MoneyVec& v, const Money& el, int pos) {

if (pos < 0 || pos > v.size()) {

throw out\_of\_range("Позиция должна быть от 0 до " + to\_string(v.size()));

}

v.insert(v.begin() + pos - 1, el);

}

// Задание 4: Удаление элементов > среднего

void remove\_above\_avg(MoneyVec& v) {

if (v.empty()) return;

float avg = accumulate(v.begin(), v.end(), 0.0f,

[](float sum, const Money& m) { return sum + m.toFloat(); }) / v.size();

cout << "Среднее значение: " << fixed << setprecision(2) << avg << " руб." << endl;

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

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

v.end());

}

// Задание 5: Умножение на максимум

void multiply\_by\_max(MoneyVec& v) {

if (v.empty()) return;

Money max\_val = \*max\_element(v.begin(), v.end());

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

for (auto& m : v) m = m \* max\_val.toFloat();

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

try {

// 1. Создание вектора

int n;

cout << "Введите размер вектора: ";

cin >> n;

MoneyVec v = make\_money\_vector(n);

cout << "Исходный вектор: ";

print\_money\_vector(v);

// Задание 3

Money min\_val = find\_min(v);

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

int pos;

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

cin >> pos;

insert\_at(v, min\_val, pos);

cout << "После вставки минимума: ";

print\_money\_vector(v);

// Задание 4

remove\_above\_avg(v);

cout << "После удаления > среднего: ";

print\_money\_vector(v);

// Задание 5

multiply\_by\_max(v);

cout << "После умножения на максимум: ";

print\_money\_vector(v);

}

catch (const exception& e) {

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

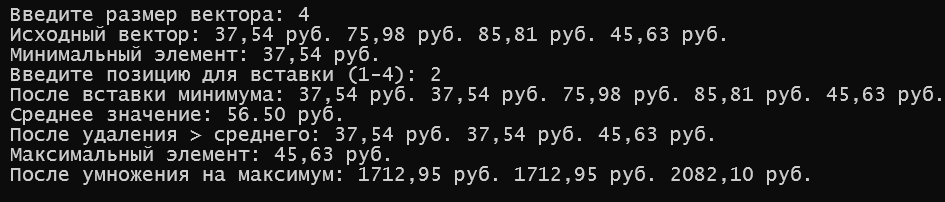
return 1;

}

return 0;

}

Результат работы программы:



Код. Задача 3. Файл Лаба\_11.3.cpp

#include <iostream>

#include <vector>

#include <algorithm>

#include <numeric>

#include <Windows.h>

#include <iomanip>

using namespace std;

// Пользовательский тип Money (как в задаче 2)

class Money {

private:

int rubles;

int kopecks;

void normalize() {

rubles += kopecks / 100;

kopecks %= 100;

}

public:

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

normalize();

}

bool operator<(const Money& other) const {

return (rubles < other.rubles) ||

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

}

Money operator\*(float multiplier) const {

float total = rubles + kopecks / 100.0f;

total \*= multiplier;

return Money(static\_cast<int>(total),

static\_cast<int>((total - static\_cast<int>(total)) \* 100));

}

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

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

return os;

}

Money operator+(const Money& other) const {

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

}

float toFloat() const {

return rubles + kopecks / 100.0f;

}

};

// Параметризированный класс MyVector

template <typename T>

class MyVector {

private:

vector<T> data;

public:

// Добавление элемента

void add(const T& item) {

data.push\_back(item);

}

// Удаление элемента по индексу

void remove(size\_t index) {

if (index >= data.size())

throw out\_of\_range("Индекс выходит за границы вектора!");

data.erase(data.begin() + index);

}

// Печать вектора

void print() const {

for (const auto& item : data)

cout << item << " ";

cout << endl;

}

// Задание 3: Вставка минимума на позицию

T find\_min() {

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

return \*min\_element(data.begin(), data.end());

}

void insertMinAt(size\_t pos) {

if (pos > data.size()) throw out\_of\_range("Неверная позиция!");

T min\_val = \*min\_element(data.begin(), data.end());

data.insert(data.begin() + pos, min\_val);

}

// Задание 4: Удаление элементов > среднего

void remov() {

if (data.empty()) return;

float avg = accumulate(data.begin(), data.end(), 0.0f,

[](float sum, const T& item) { return sum + item.toFloat(); }) / data.size();

cout << "Среднее значение: " << fixed << setprecision(2) << avg << endl;

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

[avg](const T& item) { return item.toFloat() > avg; }),

data.end());

}

// Задание 5: Умножение на максимум

void multiply() {

if (data.empty()) return;

T max\_val = \*max\_element(data.begin(), data.end());

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

for (auto& item : data)

item = item \* max\_val.toFloat();

}

};

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

try {

// 1. Создание контейнера

MyVector<Money> vec;

// 2. Заполнение элементами

int n, r, k;

cout << "Введите размер вектора: ";

cin >> n;

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

r = rand() % 100;

k = rand() % 100;

vec.add(Money(r, k));

}

cout << "Исходный вектор: ";

vec.print();

// Задание 3: Вставка минимума

cout << "Минимальный элемент: " << vec.find\_min() << endl;

int pos;

cout << "Введите позицию для вставки: ";

cin >> pos;

vec.insertMinAt(pos - 1);

cout << "После вставки минимума: ";

vec.print();

// Задание 4: Удаление > среднего

vec.remov();

cout << "После удаления > среднего: ";

vec.print();

// Задание 5: Умножение на максимум

vec.multiply();

cout << "После умножения на максимум: ";

vec.print();

}

catch (const exception& e) {

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

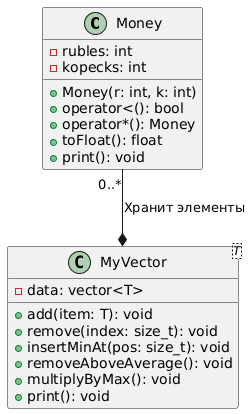
return 1;

}

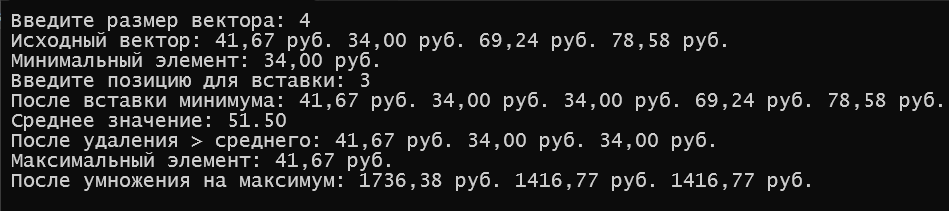
return 0;

}

Общая UML-диаграмма для 1-3 заданий



Результат работы программы:



Код. Задача 4. Файл Лаба\_11.4.cpp

#include <iostream>

#include <queue>

#include <vector>

#include <algorithm>

#include <numeric>

#include <Windows.h>

#include <iomanip>

using namespace std;

// Класс Money (аналогично предыдущим задачам)

class Money

{

private:

int rubles;

int kopecks;

void normalize() {

rubles += kopecks / 100;

kopecks %= 100;

}

public:

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

normalize();

}

bool operator<(const Money& other) const {

return (rubles < other.rubles) ||

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

}

Money operator\*(float multiplier) const {

float total = rubles + kopecks / 100.0f;

total \*= multiplier;

return Money(static\_cast<int>(total),

static\_cast<int>((total - static\_cast<int>(total)) \* 100));

}

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

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

return os;

}

float toFloat() const {

return rubles + kopecks / 100.0f;

}

};

// Функция для создания priority\_queue

priority\_queue<Money> make\_pqueue(int n) {

priority\_queue<Money> pq;

srand(time(0));

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

pq.push(Money(rand() % 100, rand() % 100));

}

return pq;

}

// Функция для печати priority\_queue (с разрушением копии)

void print\_pqueue(priority\_queue<Money> pq) {

while (!pq.empty()) {

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

pq.pop();

}

}

// Задание 3: Вставка минимума (для priority\_queue не имеет смысла, пропускаем)

// Задание 4: Удаление элементов > среднего

void remove\_above\_avg(priority\_queue<Money>& pq) {

if (pq.empty()) return;

// Вычисление среднего

vector<Money> temp;

float sum = 0;

int count = 0;

while (!pq.empty()) {

Money m = pq.top();

sum += m.toFloat();

count++;

temp.push\_back(m);

pq.pop();

}

float avg = sum / count;

cout << "Среднее значение: " << fixed << setprecision(2) << avg << " руб." << endl;

// Возвращаем только элементы <= avg

for (const Money& m : temp) {

if (m.toFloat() <= avg)

pq.push(m);

}

}

// Задание 5: Умножение на максимум

void multiply\_by\_max(priority\_queue<Money>& pq) {

if (pq.empty()) return;

vector<Money> temp;

Money max\_val = pq.top();

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

// Извлекаем, умножаем и сохраняем

while (!pq.empty()) {

Money m = pq.top();

temp.push\_back(m \* max\_val.toFloat());

pq.pop();

}

// Возвращаем обратно

for (const Money& m : temp) {

pq.push(m);

}

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

try {

// 1. Создание очереди

int n;

cout << "Введите количество элементов: ";

cin >> n;

priority\_queue<Money> pq = make\_pqueue(n);

// 2. Печать исходной очереди

cout << "Исходная очередь с приоритетами:" << endl;

print\_pqueue(pq);

// 3. Добавление элементов

// Для priority\_queue нет прямого добавления по позиции

pq.push(Money(150, 50));

pq.push(Money(75, 25));

cout << "После добавления:" << endl;

print\_pqueue(pq);

// Задание 4: Удаление > среднего

remove\_above\_avg(pq);

cout << "После удаления > среднего:" << endl;

print\_pqueue(pq);

// Задание 5: Умножение на максимум

multiply\_by\_max(pq);

cout << "После умножения на максимум:" << endl;

print\_pqueue(pq);

}

catch (const exception& e) {

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

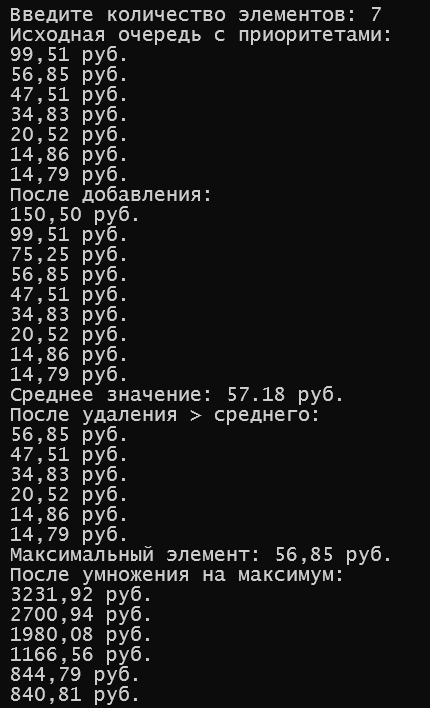
return 1;

}

return 0;

}

Результат работы программы:



Код. Задача 5. Файл Лаба\_11.5.cpp

#include <iostream>

#include <queue>

#include <vector>

#include <algorithm>

#include <numeric>

#include <Windows.h>

#include <iomanip>

using namespace std;

// Класс Money (как в предыдущих задачах)

class Money

{

private:

int rubles;

int kopecks;

void normalize() {

rubles += kopecks / 100;

kopecks %= 100;

}

public:

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

normalize();

}

// Для priority\_queue нужно operator<

bool operator<(const Money& other) const {

return (rubles < other.rubles) ||

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

}

// Для умножения (задание 5)

Money operator\*(float multiplier) const {

float total = rubles + kopecks / 100.0f;

total \*= multiplier;

return Money(static\_cast<int>(total),

static\_cast<int>((total - static\_cast<int>(total)) \* 100));

}

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

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

return os;

}

float toFloat() const {

return rubles + kopecks / 100.0f;

}

};

// Параметризированный класс PriorityContainer

template <typename T>

class PriorityContainer {

private:

priority\_queue<T> data;

public:

// Добавление элемента

void add(const T& item) {

data.push(item);

}

// Печать (разрушающая)

void print() {

priority\_queue<T> temp = data;

while (!temp.empty()) {

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

temp.pop();

}

}

// Задание 3: Вставка минимума (не поддерживается для priority\_queue)

void insertMinAt(int pos) {

cout << "Для priority\_queue вставка на позицию недоступна!" << endl;

}

// Задание 4: Удаление элементов > среднего

void remAv() {

if (data.empty()) return;

// Вычисляем среднее

vector<T> temp;

float sum = 0;

int count = 0;

while (!data.empty()) {

T item = data.top();

sum += item.toFloat();

count++;

temp.push\_back(item);

data.pop();

}

float avg = sum / count;

cout << "Среднее значение: " << fixed << setprecision(2) << avg << endl;

// Возвращаем только элементы <= avg

for (const T& item : temp) {

if (item.toFloat() <= avg) {

data.push(item);

}

}

}

// Задание 5: Умножение на максимум

void multiM() {

if (data.empty()) return;

vector<T> temp;

T max\_val = data.top();

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

// Извлекаем, умножаем и сохраняем

while (!data.empty()) {

T item = data.top();

temp.push\_back(item \* max\_val.toFloat());

data.pop();

}

// Возвращаем обратно

for (const T& item : temp) {

data.push(item);

}

}

};

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

try {

PriorityContainer<Money> container;

// 1. Заполнение контейнера

int n, r, k;

cout << "Введите размер вектора: ";

cin >> n;

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

r = rand() % 100;

k = rand() % 100;

container.add(Money(r, k));

}

cout << "Исходный контейнер:" << endl;

container.print();

// 2. Добавление элементов (по заданию)

container.add(Money(150, 0));

container.add(Money(80, 30));

cout << "После добавления:" << endl;

container.print();

// 3. Добавление элементов (пропускаем, т.к. в priority\_queue нет добавления/удаления по позиции)

// Задание 4: Удаление > среднего

container.remAv();

cout << "После удаления > среднего:" << endl;

container.print();

// Задание 5: Умножение на максимум

container.multiM();

cout << "После умножения на максимум:" << endl;

container.print();

}

catch (const exception& e) {

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

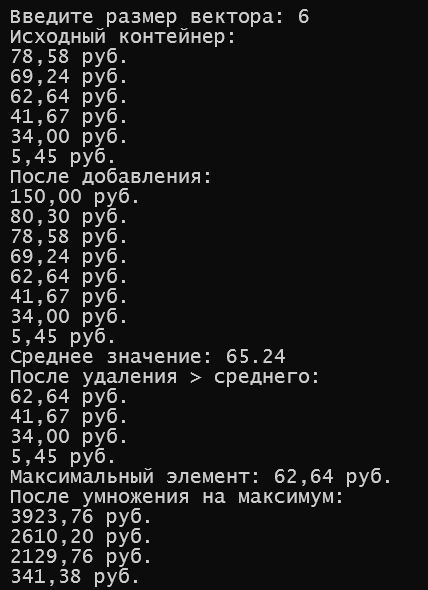
return 1;

}

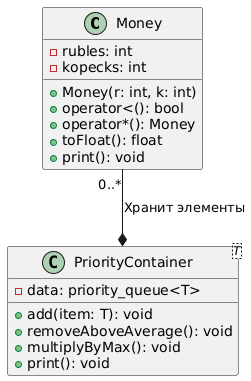
return 0;

}

Результат работы программы:



Общая UML-диаграмма для 4-5 заданий



Конртольные впросы:

* 1. Из каких частей состоит библиотека STL?

STL (Standard Template Library) составляющие:

Контейнеры (vector, list, deque, set, map и др.)

Итераторы (объекты для обхода контейнеров)

Алгоритмы (sort, find, transform и др.)

Функциональные объекты (функторы и лямбда-выражения)

Адаптеры (stack, queue, priority\_queue)

Аллокаторы (управление памятью)

* 1. Какие типы контейнеров существуют в STL?

Последовательные (vector, list, deque, array, forward\_list).

Ассоциативные (set, map, multiset, multimap).

Неупорядоченные ассоциативные (unordered\_set, unordered\_map, unordered\_multiset, unordered\_multimap).

Адаптеры контейнеров (stack, queue, priority\_queue).

* 1. Что нужно сделать для использования контейнера STL в своей программе?

Подключить соответствующий заголовочный файл и использовать пространство имён std:

#include <vector> // для вектора

#include <list> // для списка

#include <deque> // для двусторонней очереди

using namespace std; // или явно указывать std::vector

* 1. Что представляет собой итератор?

Итератор — это объект, похожий на указатель, который предоставляет доступ к элементам контейнера и позволяет перебирать их.

* 1. Какие операции можно выполнять над итераторами?

Инкремент (++it, it++).

Декремент (--it, it--, только для двунаправленных).

Разыменование (\*it, it->field).

Сравнение (it1 == it2, it1 != it2).

Арифметика (it + n, it - n, только для произвольного доступа).

* 1. Каким образом можно организовать цикл для перебора контейнера с использованием итератора?

vector<int> v = {1, 2, 3};

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

cout << \*it << " ";

}

// или

for (int x : v) {

cout << x << " ";

}

* 1. Какие типы итераторов существуют?

Input — только чтение (istream\_iterator).

Output — только запись (ostream\_iterator).

Forward — однонаправленный (forward\_list).

Bidirectional — двунаправленный (list, set, map).

Random Access — произвольный доступ (vector, deque, array).

* 1. Перечислить операции и методы общие для всех контейнеров.

|  |  |
| --- | --- |
| **Операция или метод** | **Пояснение** |
| Операции равенства (==) и неравенства (!=) | Возвращают значение true или false |
| Операция присваивания (=) | Копирует один контейнер в другой |
| clear | Удаляет все элементы |
| insert | Добавляет один элемент или диапазон элементов |
| erase | Удаляет один элемент или диапазон элементов |
| size\_type size() const | Возвращает число элементов |
| size\_type max\_size() const | Возвращает максимально допустимый размер контейнера |
| bool empty0 const | Возвращает true, если контейнер пуст |
| iterator begin() | Возвращают итератор на начало контейнера (итерации будут производиться в прямом направлении) |
| iterator end() | Возвращают итератор на конец контейнера (итерации в прямом направлении будут закончены) |
| reverse\_iterator begin() | Возвращают обратный итератор на конец контейнера (итерации будут производиться в обратном направлении) |
| reverse\_iterator end() | Возвращают обратный итератор на начало контейнера (итерации в обратном направлении будут закончены) |

* 1. Какие операции являются эффективными для контейнера vector? Почему?

Доступ по индексу (v[i], v.at(i)) — O(1): vector хранит элементы в непрерывном блоке памяти, как массив, поэтому доступ по индексу выполняется мгновенно.

Добавление/удаление в конце (push\_back(), pop\_back()) — O(1): если в vector есть свободное место, добавление в конец не требует перевыделения памяти.

При заполнении памяти vector увеличивает свой размер (обычно в 2 раза), что приводит к O(n) в худшем случае, но в среднем остаётся O(1).

При вставке/удалении в начало или середину (insert(), erase()) — O(n) требуется сдвиг всех последующих элементов (неэффективно).

* 1. Какие операции являются эффективными для контейнера list? Почему?

Вставка/удаление в любом месте (insert(), erase()) — O(1): list реализован как двусвязный список, поэтому вставка и удаление по итератору выполняются за константное время (нужно лишь переставить указатели соседних узлов).

Добавление/удаление в начале и конце (push\_front(), pop\_front(), push\_back(), pop\_back()) — O(1): доступ к началу и концу списка происходит мгновенно, так как list хранит указатели на head и tail.

При доступе по индексу (list[i]) — O(n) нужно перебирать элементы от начала до нужной позиции. А при поиске (find(), count()) — O(n) требуется линейный обход (неэффективно).

* 1. Какие операции являются эффективными для контейнера deque? Почему?

Доступ по индексу (d[i]) — O(1): deque использует сегментированный массив (несколько блоков памяти), но вычисление позиции остаётся быстрым.

Добавление/удаление в начале и конце (push\_front(), pop\_front(), push\_back(), pop\_back()) — O(1): deque может быстро расширяться в обе стороны за счёт добавления новых блоков.

При вставке/удалении в середину (insert(), erase()) — O(n): требуется сдвиг элементов, как в vector (неэффективно).

* 1. Перечислить методы, которые поддерживает последовательный контейнер vector.

vector<T> v;

// Доступ:

v[i]; // Быстрый доступ (O(1)), но без проверки границ

v.at(i); // С проверкой границ (бросает исключение)

v.front(); // Первый элемент

v.back(); // Последний элемент

v.data(); // Указатель на массив (для совместимости с C)

// Изменение размера:

v.resize(n); // Изменяет размер (если n > size, добавляет элементы по умолчанию)

v.reserve(n); // Резервирует память (уменьшает количество переаллокаций)

// Добавление/удаление:

v.push\_back(x); // добавляет в конец (O(1))

v.pop\_back(); // Удаляет последний (O(1))

v.insert(it, x); // Вставляет перед итератором (O(n))

v.erase(it); // Удаляет элемент (O(n))

v.clear(); // Очищает вектор (O(n))

* 1. Перечислить методы, которые поддерживает последовательный контейнер list.

list<T> l;

// Добавление/удаление:

l.push\_front(x); // В начало (O(1))

l.pop\_front(); // Удаляет первый (O(1))

l.push\_back(x); // В конец (O(1))

l.pop\_back(); // Удаляет последний (O(1))

l.insert(it, x); // Вставляет перед итератором (O(1))

l.erase(it); // Удаляет элемент (O(1))

// Специфичные для списка:

l.remove(x); // Удаляет все элементы == x (O(n))

l.remove\_if(pred); // Удаляет по условию (O(n))

l.sort(); // Сортировка (O(n log n))

l.merge(other); // Слияние двух отсортированных списков (O(n))

l.unique(); // Удаляет дубликаты (O(n))

l.splice(it, other\_list); // Переносит элементы из другого списка (O(1))

* 1. Перечислить методы, которые поддерживает последовательный контейнер deque.

deque<T> d;

// Доступ:

d[i]; // Доступ по индексу (O(1))

d.at(i); // С проверкой границ

d.front(); // Первый элемент

d.back(); // Последний элемент

// Добавление/удаление:

d.push\_front(x); // В начало (O(1))

d.pop\_front(); // Удаляет первый (O(1))

d.push\_back(x); // В конец (O(1))

d.pop\_back(); // Удаляет последний (O(1))

d.insert(it, x); // Вставляет перед итератором (O(n))

d.erase(it); // Удаляет элемент (O(n))

d.clear(); // Очищает очередь (O(n))

* 1. Задан контейнер vector. Как удалить из него элементы со 2 по 5?

vector<int> v = {10, 20, 30, 40, 50, 60, 70};

auto it1 = v.begin() + 1; // Второй элемент (индекс 1)

auto it2 = v.begin() + 5; // Шестой элемент (индекс 5)

v.erase(it1, it2); // Удаляет элементы [it1, it2)

После удаления: v = {10, 60, 70}.

* 1. Задан контейнер vector. Как удалить из него последний элемент?

v.pop\_back(); // Удаляет 70 (из предыдущего)

* 1. Задан контейнер list. Как удалить из него элементы со 2 по 5?

Так как list не поддерживает произвольный доступ, перебираем итераторами:

list<int> l = {10, 20, 30, 40, 50, 60, 70};

auto it1 = next(l.begin(), 1); // Второй элемент (индекс 1)

auto it2 = next(l.begin(), 5); // Шестой элемент (индекс 5)

l.erase(it1, it2); // Удаляет [it1, it2)

Результат: l = {10, 60, 70}.

* 1. Задан контейнер list. Как удалить из него последний элемент?

l.pop\_back(); // Удаляет 70

* 1. Задан контейнер deque. Как удалить из него элементы со 2 по 5?

Аналогично vector, используем erase():

deque<int> d = {10, 20, 30, 40, 50, 60, 70};

auto it1 = d.begin() + 1; // Второй элемент (индекс 1)

auto it2 = d.begin() + 5; // Шестой элемент (индекс 5)

d.erase(it1, it2); // Удаляет [it1, it2)

Результат: l = {10, 60, 70}.

* 1. Задан контейнер deque. Как удалить из него последний элемент?

d.pop\_back(); // Удаляет 70

* 1. Написать функцию для печати последовательного контейнера с использованием итератора.

#include <iostream>

#include <vector>

template<typename Container>

void printContainer(const Container& c) {

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

std::cout << \*it << " ";

}

std::cout << std::endl;

}

// Использование:

vector<int> v = {1, 2, 3};

printContainer(v); // Вывод: 1 2 3

* 1. Что представляют собой адаптеры контейнеров?

Адаптеры — это обёртки над стандартными контейнерами, предоставляющие ограниченный интерфейс:

stack — LIFO (последним пришёл — первым ушёл).

queue — FIFO (первым пришёл — первым ушёл).

priority\_queue — элементы с высшим приоритетом извлекаются первыми.

Они используют deque (по умолчанию), list или vector в качестве базового контейнера.

* 1. Чем отличаются друг от друга объявления stack<int> s и stack<int, list<int> > s?

stack<int> s — использует deque по умолчанию.

stack<int, list<int>> s — использует list в качестве внутреннего контейнера.

Разница в производительности:

deque — быстрый доступ с обеих сторон, но возможна переаллокация.

list — медленнее доступ по индексу, но стабильные вставки/удаления.

* 1. Перечислить методы, которые поддерживает контейнер stack.

stack<int> s;

// Добавление и удаление:

s.push(10); // Добавляет элемент на вершину стека — O(1)

s.pop(); // Удаляет элемент с вершины — O(1)

// Доступ:

s.top(); // Возвращает элемент на вершине — O(1)

s.empty(); // true, если стек пуст — O(1)

s.size(); // Возвращает количество элементов — O(1)

* 1. Перечислить методы, которые поддерживает контейнер queue.

queue<int> q;

// Добавление и удаление:

q.push(10); // Добавляет элемент в конец очереди — O(1)

q.pop(); // Удаляет первый элемент — O(1)

// Доступ:

q.front(); // Возвращает первый элемент (голову очереди) — O(1)

q.back(); // Возвращает последний элемент — O(1)

q.empty(); // true, если очередь пуста — O(1)

q.size(); // Возвращает количество элементов — O(1)

* 1. Чем отличаются друг от друга контейнеры queue и priority\_queue?

| **Характеристика** | queue**(очередь)** | priority\_queue**(очередь с приоритетом)** |
| --- | --- | --- |
| **Порядок элементов** | FIFO (первый вошёл — первый вышел) | Элементы упорядочены по приоритету (по умолчанию max-heap — наибольший первым) |
| **Вставка** | push() — в конец (O(1)) | push() — с переупорядочиванием (O(log n)) |
| **Извлечение** | front() + pop() (O(1)) | top() + pop() (O(log n)) |
| **Используемый контейнер** | По умолчанию deque | По умолчанию vector + std::less для сравнения |
| **Пример** | Очередь задач в порядке поступления | Очередь задач с приоритетом (напр., системные процессы) |

* 1. Задан контейнер stack. Как удалить из него элемент с заданным номером?

Стандартный stack не поддерживает произвольное удаление. Решение:

1. Использовать дополнительный стек для временного хранения.
2. Удалить нужный элемент.

#include <stack>

#include <iostream>

void removeFromStack(std::stack<int>& s, size\_t index) {

std::stack<int> temp;

// Переносим элементы во временный стек до нужного индекса

for (size\_t i = 0; i < index; ++i) {

temp.push(s.top());

s.pop();

}

s.pop(); // Удаляем целевой элемент

// Возвращаем остальные элементы обратно

while (!temp.empty()) {

s.push(temp.top());

temp.pop();

}

}

* 1. Задан контейнер queue. Как удалить из него элемент с заданным номером?

Аналогично stack, стандартная queue не поддерживает произвольное удаление. Решение:

#include <queue>

#include <iostream>

void removeFromQueue(std::queue<int>& q, size\_t index) {

std::queue<int> temp;

// Переносим элементы во временную очередь до нужного индекса

for (size\_t i = 0; i < index; ++i) {

temp.push(q.front());

q.pop();

}

q.pop(); // Удаляем целевой элемент

// Возвращаем остальные элементы обратно

while (!temp.empty()) {

q.push(temp.front());

temp.pop();

}

}

* 1. Написать функцию для печати контейнера stack с использованием итератора.

Так как stack не предоставляет итераторы, можно:

1. Скопировать элементы в вектор и напечатать его.
2. Использовать рекурсию (но это изменит стек).

#include <stack>

#include <vector>

#include <iostream>

void printStack(std::stack<int> s) { // Передаём по значению (копия)

std::vector<int> elements;

while (!s.empty()) {

elements.push\_back(s.top());

s.pop();

}

// Печатаем в обратном порядке (чтобы сохранить порядок стека)

for (auto it = elements.rbegin(); it != elements.rend(); ++it) {

std::cout << \*it << " ";

}

std::cout << std::endl;

}

* 1. Написать функцию для печати контейнера queue с использованием итератора.

Аналогично, queue не имеет итераторов:

#include <queue>

#include <iostream>

void printQueue(std::queue<int> q) { // Передаём по значению (копия)

while (!q.empty()) {

std::cout << q.front() << " ";

q.pop();

}

std::cout << std::endl;

}