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

«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ПНИПУ)

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

Лабораторные работы по

«Основам алгоритмизации и программирования»

вариант № 15

за 2 семестр

Выполнил:

студент группы РИС-21-1бз

Тимолянов Григорий Константинович

Проверила:

Доцент кафедры ИТАС,

к.т.н. Полякова О.А.

2024

###### **Лабораторная работа №13**

**Ассоциативные контейнеры библиотеки STL**

**Цель:** 1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.

2) Использование ассоциативных контейнеров библиотеки STL в ОО программе

###### Постановка задачи

Задача 1.

1. Создать последовательный контейнер.
2. Заполнить его элементами пользовательского типа (тип указан в варианте).
3. Заменить 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.

Контейнер - вектор

Тип элементов – money

Найти минимальный элемент и добавить его в конец контейнера;

Найти элемент с заданным ключом и удалить его из контейнера;

К каждому элементу добавить сумму минимального и максимального элементов контейнера.

###### **Функции для решения задачи 1**

#include<iostream>

#include<vector>

#include<algorithm>

#include"Money.h"

typedef std::vector<Money> TMoney;

Money temp;

struct Greater\_temp

{

bool operator()(Money& m\_temp)

{

return (m\_temp > temp);

}

};

struct Comp\_less

{

bool operator()(const Money& \_Left, const Money& \_Right)

const{

return (\_Left > \_Right);

}

};

struct Equal\_temp

{

bool operator()(Money& m\_temp)

const{

return(m\_temp == temp);

}

};

TMoney make\_vector(const size\_t n)

{

Money money;

TMoney tmoney;

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

{

std::cin >> money;

tmoney.push\_back(money);

}

return tmoney;

}

void print\_vector(const TMoney& tmoney)

{

for (auto &i: tmoney)

{

std::cout << i << ": ";

}

std::cout << std::endl;

}

Money arithmetic\_mean(const TMoney& tmoney)

{

Money temp = tmoney[0];

for (size\_t i = 0; i < tmoney.size(); i++)

temp += tmoney[i];

return temp / tmoney.size();

}

void division(Money& money)

{

money /= temp;

}

void add\_sum\_max\_min(TMoney& s)

{

Money sum\_min\_max = \*std::max\_element(s.begin(), s.end()) + \*std::max\_element(s.begin(), s.end());

size\_t i = 0;

for (size\_t i = 0; i < s.size(); i++)

s[i] += sum\_min\_max;

}

###### Основная программа для решения задачи 1

int main()

{

size\_t n;

std::cout << "N? ";

std::cin >> n;

TMoney tmoney;

tmoney = make\_vector(n);

print\_vector(tmoney);

TMoney::iterator i;

i = std::max\_element(tmoney.begin(), tmoney.end());

std::cout << "Max: " << \*i << std::endl;

Money money\_max = \*i;

temp = arithmetic\_mean(tmoney);

std::cout << "Arithmetic mean: " << temp << std::endl;

std::replace\_if(tmoney.begin(), tmoney.end(), Greater\_temp(), money\_max);

std::cout << "Replacement: " << std::endl;

print\_vector(tmoney);

std::cout << "Descending sort: " << std::endl;

std::sort(tmoney.begin(), tmoney.end(), Comp\_less());

print\_vector(tmoney);

std::cout << "Ascending sort: " << std::endl;

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

print\_vector(tmoney);

std::cout << "Search: " << std::endl;

std::cin >> temp;

i = std::find\_if(tmoney.begin(), tmoney.end(), Equal\_temp());

if (i != tmoney.end())

std::cout << \*i << std::endl;

else

std::cout << "Not such element!" << std::endl;

std::cout << "Deletion: " << std::endl;

i = std::min\_element(tmoney.begin(), tmoney.end());

temp = \*i;

i = std::remove\_if(tmoney.begin(), tmoney.end(), Equal\_temp());

tmoney.erase(i, tmoney.end());

print\_vector(tmoney);

std::cout << "Add a minimal element: " << std::endl;

i = std::min\_element(tmoney.begin(), tmoney.end());

tmoney.push\_back(\*i);

print\_vector(tmoney);

std::cout << "Add the sum of the maximum and minimum: " << std::endl;

add\_sum\_max\_min(tmoney);

print\_vector(tmoney);

return 0;

}

###### Объяснение результатов работы программы.

###### 

Создаем вектор на 3 элемента и заполняем его элементами типа Money;

Производим сортировку по убыванию, затем по возрастанию;

Производим поиск значения для удаления, затем удаляем его;

Вставка минимального элемента в вектор;

Добавление к всем элементам суммы минимального и максимального значений контейнера.

###### Задача 2.

1. Создать последовательный контейнер.
2. Заполнить его элементами пользовательского типа (тип указан в варианте).
3. Заменить 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.

Адаптер контейнера - очередь

###### Классы для решения задачи 2

Money.h

 #pragma once

#include<iostream>

class Money

{

int m\_rubles;

int m\_kopeck;

public:

//---------constructor-----------

Money() :m\_rubles(0), m\_kopeck(0) {};

Money(long, short);

Money(const Money&);

~Money();

//-------Getters and Setters----

long Get\_rubles();

short Get\_kopeck();

void Set\_rubles(long);

void Set\_kopeck(short);

//-------overloaded-functions----

Money& operator=(const Money&);

Money operator+(const Money&);

Money operator+(int);

Money& operator+=(const Money&);

Money operator/(const Money&);

Money& operator/=(const Money&);

Money operator/(int);

bool operator>(const Money&);

bool operator<(const Money&);

bool operator!=(const Money&);

bool operator==(const Money&);

void clear();

//-------friend-functions---------

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

friend std::istream& operator>>(std::istream&, Money&);

};

Money.cpp

#include "Money.h"

Money::Money(const long rubles, const short kopeck)

{

if (rubles < 0 || kopeck < 0)

{

m\_rubles = -abs(m\_rubles);

m\_kopeck = -abs(m\_kopeck);

}

long long money = (long long)m\_rubles \* 100 + m\_kopeck;

m\_rubles = money / 100;

m\_kopeck = money % 100;

}

Money::Money(const Money& money)

{

m\_rubles = money.m\_rubles;

m\_kopeck = money.m\_kopeck;

}

Money::~Money()

{

}

long Money::Get\_rubles()

{

return m\_rubles;

}

short Money::Get\_kopeck()

{

return m\_kopeck;

}

void Money::Set\_rubles(const long rubles)

{

if (m\_kopeck >= 0)

m\_rubles = rubles;

else

m\_rubles = -abs(rubles);

}

void Money::Set\_kopeck(const short kopeck)

{

m\_kopeck = kopeck;

if (m\_rubles < 0)

m\_kopeck = -abs(m\_kopeck);

if (m\_kopeck < 0)

m\_rubles = -abs(m\_rubles);

}

Money& Money::operator=(const Money& money)

{

if (&money == this)

return \*this;

m\_rubles = money.m\_rubles;

m\_kopeck = money.m\_kopeck;

return \*this;

}

Money Money::operator+(const Money& money)

{

long long money\_first = (long long)m\_rubles \* 100 + m\_kopeck;

long long money\_second = (long long)money.m\_rubles \* 100 + money.m\_kopeck;

Money p;

p.m\_rubles = (money\_first + money\_second) / 100;

p.m\_kopeck = (money\_first + money\_second) % 100;

return p;

}

Money Money::operator+(int value)

{

long long money = (long long)m\_rubles \* 100 + m\_kopeck;

money += value;

Money temp;

temp.m\_rubles = (money) / 100;

temp.m\_kopeck = (money) % 100;

return temp;

}

Money& Money::operator+=(const Money& money)

{

long long temp1 = (long long)m\_rubles \* 100 + m\_kopeck;

long long temp2 = (long long)money.m\_rubles \* 100 + money.m\_kopeck;

m\_rubles = (temp1 + temp2) / 100;

m\_kopeck = (temp1 + temp2) % 100;

return \*this;

}

Money Money::operator/(const Money& money)

{

Money division;

if (&money == this)

{

division.m\_rubles = 0;

division.m\_kopeck = 1;

return division;

}

long long money\_first = m\_rubles;

money\_first \*= 100;

money\_first += m\_kopeck;

long long money\_second = money.m\_rubles;

money\_second \*= 100;

money\_second += money.m\_kopeck;

division.m\_rubles = (money\_first / money\_second) / 100;

division.m\_kopeck = (money\_first / money\_second) % 100;

return division;

}

Money& Money::operator/=(const Money& money)

{

if (&money == this)

{

m\_rubles = 0;

m\_kopeck = 1;

return \*this;

}

long long money\_first = (long long)m\_rubles \* 100 + m\_kopeck;

long long money\_second = (long long)money.m\_rubles \* 100 + money.m\_kopeck;

m\_rubles = (money\_first / money\_second) / 100;

m\_kopeck = (money\_first / money\_second) % 100;

return \*this;

}

Money Money::operator/(const int value)

{

int temp1 = m\_rubles \* 100 + m\_kopeck;

Money temp\_money;

temp\_money.m\_rubles = (temp1 / value) / 100;

temp\_money.m\_kopeck = (temp1 / value) % 100;

return temp\_money;

}

bool Money::operator>(const Money& money)

{

return (m\_rubles > money.m\_rubles || m\_rubles == money.m\_rubles && m\_kopeck > money.m\_kopeck);

}

bool Money::operator<(const Money& money)

{

return (m\_rubles < money.m\_rubles || m\_rubles == money.m\_rubles && m\_kopeck < money.m\_kopeck);

}

bool Money::operator!=(const Money& money)

{

return !(m\_rubles == money.m\_rubles && m\_kopeck == money.m\_kopeck);

}

bool Money::operator==(const Money& money)

{

return (m\_rubles == money.m\_rubles && m\_kopeck == money.m\_kopeck);

}

void Money::clear()

{

m\_kopeck = 0;

m\_rubles = 0;

}

std::ostream& operator<<(std::ostream& out, const Money& time)

{

return (out << time.m\_rubles << ":" << time.m\_kopeck);

}

std::istream& operator>>(std::istream& in, Money& money)

{

int kopeck\_sup;

std::cout << "Rubles? "; in >> money.m\_rubles;

do

{

std::cout << "Kopeck? "; in >> kopeck\_sup;

if (abs(kopeck\_sup) > 99)

std::cout << "Error!. Please enter a number from -99 to 99" << std::endl;

} while (abs(kopeck\_sup) > 99);

money.m\_kopeck = kopeck\_sup;

if (money.m\_rubles < 0 || money.m\_kopeck < 0)

{

money.m\_rubles = -abs(money.m\_rubles);

money.m\_kopeck = -abs(money.m\_kopeck);

}

return in;

}

###### Функции для решения задачи 2

#include <iostream>

#include<vector>

#include<queue>

#include"Money.h"

typedef std::queue<Money> Qu;

typedef std::vector<Money> Vec;

Qu make\_queue(int n) //функция для формирования очереди

{

Qu v; //пустой вектор

Money money;

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

{

std::cin >> money;

v.push(money); //добавить money в конец вектора

}

return v;

}

Vec copy\_queue\_to\_vector(Qu s) //копируем стек в вектор

{

Vec v;

while (!s.empty()) //пока очередь не пустая

{

v.push\_back(s.front()); //добавить в вектор элемент из вершины очереди

s.pop();

}

return v; //вернуть вектор как результат функции

}

Qu copy\_vector\_to\_queue(Vec v) //копируем вектор в очередь

{

Qu s;

for (size\_t i = 0; i < v.size(); i++)

{

s.push(v[i]); //добавить в очередь элемент вектора

}

return s; //вернуть очередь как результат функции

}

void print\_queue(Qu s)

{

while (!s.empty())

{

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

s.pop();

}

std::cout << std::endl;

}

Money arithmetic\_mean(Qu s) //вычисление среднего значения

{

Vec v = copy\_queue\_to\_vector(s); //копируем значение для суммы

int n = 1;

Money sum = s.front(); //начальное значение для суммы

s.pop(); //удалить первый элемент из вектора

while (!s.empty()) //пока стек не пустой

{

sum += s.front(); //добавить в сумму элемент из вершины стека

s.pop(); //удалить элемент

n++;

}

s = copy\_vector\_to\_queue(v); //скопировать вектор в очередь

return (sum / n);

}

void Add\_to\_queue(Qu& s, const Money& el, int pos) //добавление элемента в очередь

{

Vec v;

Money t;

int i = s.size(); //размер очереди

if (!pos)

v.push\_back(el); //если номер позиции 0, то добавить его в конец

while (!s.empty()) //пока стек не пустой

{

t = s.front(); //получить элемент из вершины

if (i == pos) //если номер равен номеру позиции, на которую добавлен элемент

v.push\_back(el); //добавить новый элемент из очeреди в вектор

v.push\_back(t); //добавить элемент из очереди в вектор

s.pop(); //удалить элемент из очереди

i--;

}

s = copy\_vector\_to\_queue(v); //скопировать вектор в очередь

}

Money Max(Qu s) // поиск максимального элемента в очереди

{

Money m = s.front(); //переменной m присваивается значение из вершины очереди

Vec v = copy\_queue\_to\_vector(s); //копируем очередь в вектор

while (!s.empty()) //пока стек не пустой

{

if (s.front() > m) //сравнить m и элемент в вершине очереди

m = s.front(); // удалить элемент из очереди

s.pop();

}

s = copy\_vector\_to\_queue(v); //скопировать вектор в очередь

return m;

}

void Delete\_from\_queue(Qu& s)//удалить максимальный элемент из очереди

{

Money m = Max(s); //находим максимальный элемент

Vec v;

Money t;

while (!s.empty()) //пока очередь пустая

{

t = s.front(); //получаем элемент из вершины очереди

if (t != m) //если он не равен максимальному, занести элемент в вектор

v.push\_back(t);

s.pop(); //удалить элемент из очереди

}

s = copy\_vector\_to\_queue(v); //скопировать вектор в очередь

}

Money Min(Qu s)

{

Money m = s.front();

Vec v = copy\_queue\_to\_vector(s);

while (!s.empty()) //пока очередь не пустая

{

if (s.front() < m) //сравнить m и элемент в вершине очереди

m = s.front(); // удалить элемент из стека

s.pop();

}

s = copy\_vector\_to\_queue(v); //скопировать вектор в очередь

return m;

}

void Division(Qu& s)

{

Money m = Min(s);

Vec v;

Money t;

while (!s.empty()) //пока стек не пустой

{

t = s.front(); //получить элемент из вершины

v.push\_back(t / m); //делить t на минимальный элемент и добавить в вектор

s.pop(); //удалить элемент из вершины

}

s = copy\_vector\_to\_queue(v); //скопировать вектор в стек

}

void add\_min\_element(Qu& s)

{

Vec v;

Money min = Min(s);

Money t;

size\_t i = 0;

while (!s.empty()) //пока очередь пустая

{

t = s.front(); //получаем элемент из вершины очереди

v.push\_back(t);

s.pop(); //удалить элемент из стека

i++;

}

v.push\_back(min);

s = copy\_vector\_to\_queue(v); //скопировать вектор в очередь

}

void remove\_element\_by\_index(Qu& s, int pos)

{

Vec v;

Money t;

size\_t i = 0;

while (!s.empty()) //пока очередь пустая

{

t = s.front(); //получаем элемент из вершины очереди

if (pos != i) //если он не равен максимальному, занести элемент в вектор

v.push\_back(t);

s.pop(); //удалить элемент из стека

i++;

}

s = copy\_vector\_to\_queue(v); //скопировать вектор в очередь

}

void add\_sum\_max\_min(Qu& s)

{

Vec v;

Money sum\_min\_max = Max(s) + Min(s);

Money t;

size\_t i = 0;

while (!s.empty()) //пока очередь пустая

{

t = s.front(); //получаем элемент из вершины очереди

v.push\_back(t + sum\_min\_max);

s.pop(); //удалить элемент из стека

i++;

}

s = copy\_vector\_to\_queue(v); //скопировать вектор в очередь

}

###### Основная программа для решения задачи 2

int main()

{

Money t;

Qu q;

int n;

std::cout << "n? ";

std::cin >> n;

q = make\_queue(n);

print\_queue(q);

t = arithmetic\_mean(q);

std::cout << "Arithmetic mean = " << arithmetic\_mean(q) << std::endl;

std::cout << "Add arithmetic mean: " << std::endl;

std::cout << "Position? ";

int pos;

std::cin >> pos;

Add\_to\_queue(q, t, pos);

print\_queue(q);

add\_min\_element(q);

std::cout << "Add a minimal element: " << std::endl;

print\_queue(q);

std::cout << "\nPosition to delete an element? ";

std::cin >> pos;

remove\_element\_by\_index(q, pos); //удалить элемент с этим номером

std::cout << "Remove element by index: " << std::endl;

print\_queue(q);

add\_sum\_max\_min(q);

std::cout << "Add the sum of the maximum and minimum: " << std::endl;

print\_queue(q);

std::cout << "Delete Max = " << std::endl;

Delete\_from\_queue(q);

print\_queue(q);

}

###### Объяснение результатов работы программы.

###### 

Создаем очередь на 3 элемента и заполняем его элементами типа Money;

Находим средне-арифметическое значение и выбираем позиции для вставки;

Находим минимальное значение и вставляем его в конец очереди;

Производим ввод позиции для удаления, затем удаляем данный элемент;

Добавление к всем элементам суммы минимального и максимального значений очереди;

Удаляем максимальное значение очереди.

###### Задача 3

1. Создать последовательный контейнер.
2. Заполнить его элементами пользовательского типа (тип указан в варианте).
3. Заменить 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.

Ассоциативный контейнер – словарь;

###### Функции для решения задачи 3

#include<iostream>

#include<vector>

#include<map>

#include<algorithm>

#include"Money.h"

typedef std::map<int, Money> TMap;

typedef TMap::iterator it;

Money temp;

struct Greater\_temp

{

bool operator()(const std::pair<int, Money>& m\_temp)

{

return (m\_temp.second > temp);

}

};

struct Comp\_less

{

bool operator()(const std::pair<int, Money>& \_Left, const std::pair<int, Money>& \_Right)

const{

return (\_Left.second > \_Right.second);

}

};

struct Equal\_temp

{

bool operator()(std::pair<int, Money> m\_temp)

{

return(m\_temp.second == temp);

}

};

TMap make\_map(const size\_t n)

{

Money money;

TMap tmoney;

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

{

std::cin >> money;

tmoney.insert(std::make\_pair(i, money));

}

return tmoney;

}

void print\_map(TMap m)

{

for (size\_t i = 0; i < m.size(); i++)

std::cout << i << " : " << m[i] << " " << std::endl;

}

Money arithmetic\_mean(TMap v)

{

Money s{};

for (size\_t i = 0; i < v.size(); i++)

s += v[i];

return s / v.size();

}

void division(TMap& v)

{

it i = std::max\_element(v.begin(), v.end());

Money m = i->second;

for (size\_t i = 0; i < v.size(); i++)

v[i] = v[i] / m;

}

void add\_sum\_max\_min(TMap& s)

{

it i = std::max\_element(s.begin(), s.end());

Money sum\_min\_max = i->second;

i = std::max\_element(s.begin(), s.end());

sum\_min\_max += i->second;

for (size\_t i = 0; i < s.size(); i++)

s[i] += sum\_min\_max;

}

###### Основная программа для решения задачи 3

int main()

{

size\_t n;

std::cout << "N? ";

std::cin >> n;

TMap tmoney;

tmoney = make\_map(n);

print\_map(tmoney);

it i;

i = std::max\_element(tmoney.begin(), tmoney.end());

std::cout << "Max: " << i->second << std::endl;

Money money\_max = i->second;

temp = arithmetic\_mean(tmoney);

std::cout << "Arithmetic mean: " << temp << std::endl;

std::cout << "Search: " << std::endl;

std::cin >> temp;

i = std::find\_if(tmoney.begin(), tmoney.end(), Equal\_temp());

if (i != tmoney.end())

std::cout << i->second << std::endl;

else

std::cout << "Not such element!" << std::endl;

std::cout << "Removing the minimum element: " << std::endl;

i = std::min\_element(tmoney.begin(), tmoney.end());

temp = i->second;

i = tmoney.find(i->first);

tmoney.erase(i);

print\_map(tmoney);

std::cout << "Add a minimal element: " << std::endl;

i = std::min\_element(tmoney.begin(), tmoney.end());

tmoney.insert(std::make\_pair(i->first, i->second));

print\_map(tmoney);

std::cout << "Add the sum of the maximum and minimum: " << std::endl;

add\_sum\_max\_min(tmoney);

print\_map(tmoney);

return 0;

}

###### Объяснение результатов работы программы.

###### 

Создаем словарь на 3 элемента и заполняем его элементами типа Money;

Производим поиск значения и удаляем его;

Находим минимальное значение и вставляем его в словарь;

Добавление к всем элементам суммы минимального и максимального значений словаря;