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 №11**

###### **Последовательные контейнеры библиотеки STL.**

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

2) Использование последовательных контейнеров библиотеки STL в ОО  
программе.

###### Постановка задачи

Задача 1.

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

Контейнер - вектор

Тип элементов - float

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<cstdlib>

typedef std::vector<float>Vec; //определить тип для работы с вектором

Vec make\_vector(int n) //функция для формирования вектора

{

Vec v; //пустой вектор

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

{

srand(unsigned(time(0)));

float a = ((float)rand() / RAND\_MAX) + rand() % 100 - 50;

v.push\_back(a); //добавить а в конец вектора

}

return v;

}

void print\_vector(const Vec& v)

{

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

{

if (i)

std::cout << "; \t" << v[i];

else

std::cout << v[i];

}

std::cout << "." << std::endl;

}

float arithmetic\_mean(const Vec& v)

{

float s = 0;

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

{

s += v[i];

}

float n = v.size(); //кол-во элементов в векторе

return s / n;

}

int max(const Vec& v)

{

int m = v[0]; //мин. элемент

int n = 0; //номер мин. элемента

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

{

if (m < v[i])

{

m = v[i]; //макс. элемент

n = i; //номер макс. элемента

}

}

return n;

}

void remove\_element\_by\_index(Vec& v, int pos) //удалить элемент из позиции pos

{

v.erase(v.begin() + pos);

}

int min(const Vec& v) //поиск минимального элемента

{

float m = v[0]; //мин. элемент

int n = 0; //номер мин. элемента

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

{

if (m > v[i])

{

m = v[i]; //мин. элемент

n = i; //номер мин. элемента

}

}

return n;

}

void subtract\_min\_element(Vec& v)

{

float m = v[min(v)] + v[max(v)];

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

{

v[i] += m;

}

###### **}**

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

int main()

{

srand(unsigned(time(0)));

try

{

std::vector<float> v;

std::vector<float>::iterator vi = v.begin();

int n;

std::cout << "N? ";

std::cin >> n;

v = make\_vector(n); //формирование вектора

print\_vector(v); //печать вектора

std::cout << "\nAdd minimal element: ";

v.push\_back(v[min(v)]);

print\_vector(v); //печать вектора

//

int pos;

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

std::cin >> pos;

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

std::cout << "\nRemove element by index: " << std::endl;

print\_vector(v);

//

subtract\_min\_element(v); //каждый элемент разделить на мин. значение вектора

std::cout << "\nAdd the sum of the minimum and maximum elements: " << std::endl;

print\_vector(v);

}

catch (int)

{

std::cout << "Error!" << std::endl;

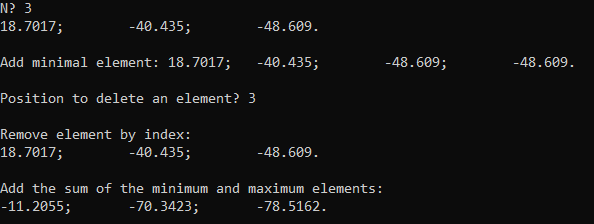
}

return 0;

}

###### 

###### **Объяснение результатов работы программы.**



Вводим в переменную N длину вектора для создания;

Создается вектор длиной N и заполнение его случайными элементами;

Происходит поиск минимального элемента и добавление его в конец вектора;

Выбор элемента для удаления, затем происходит удаление элемента по индексу;

Добавление суммы минимального и максимального элементов каждому элементу вектора.

###### Задача 2.

1. Создать последовательный контейнер.

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

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 2

Money.h

#pragma once

#include<iostream>

class Money

{

long m\_rubles;

short m\_kopeck;

public:

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

Money();

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+=(const Money&);

Money operator/(const Money&);

Money& operator/=(const Money&);

Money operator/(int);

Money& operator-=(const int);

Money& 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()

{

m\_rubles = 0;

m\_kopeck = 0;

}

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);

//m\_rubles = 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+=(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;

}

Money& Money::operator-=(const int value)

{

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

money -= value;

m\_rubles = money / 100;

m\_kopeck = money % 100;

return \*this;

}

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

{

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

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

m\_rubles = (money\_this - money\_other) / 100;

m\_kopeck = (money\_this - money\_other) % 100;

return \*this;

}

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);

}

void Money::clear()

{

m\_kopeck = 0;

m\_rubles = 0;

}

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

{

return (out << money.m\_rubles << ":" << money.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;

}

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

#include<iostream>

#include<vector>

#include<cstdlib>

#include"Money.h"

typedef std::vector<Money>Vec;

Vec make\_vector(const int n)

{

Vec v;

Money money;

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

{

money.Set\_rubles(rand() % 100 - 50);

money.Set\_kopeck(rand() % 100 - 50);

v.push\_back(money);

money.clear();

}

return v;

}

void print\_vector(const Vec& v)

{

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

{

if (i)

std::cout << "; " << v[i];

else

std::cout << v[i];

}

std::cout << "." << std::endl;

}

Money arithmetic\_mean(const Vec& v)

{

Money time;

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

{

time += v[i];

}

int n = v.size();

return (time / n);

}

void add\_vector(Vec& v, Money el, int pos)

{

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

}

int max(const Vec& v)

{

Money time = v[0];

int n = 0;

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

{

if (time < v[i])

{

time = v[i];

n = i;

}

}

return n;

}

void remove\_element\_by\_index(Vec& v, int pos)

{

v.erase(v.begin() + pos);

}

int min(const Vec& v)

{

Money time = v[0];

int n = 0;

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

{

if (time > v[i])

{

time = v[i];

n = i;

}

}

return n;

}

void subtract\_min\_element(Vec& v)

{

Money m = v[min(v)] + v[max(v)];

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

{

v[i] += m;

}

}

int main()

{

try

{

std::vector<Money> v;

std::vector<Money>::iterator vi = v.begin();

int n;

std::cout << "N? ";

std::cin >> n;

v = make\_vector(n);

print\_vector(v);

//

std::cout << "\nAdd minimal element: ";

v.push\_back(v[min(v)]);

print\_vector(v); //печать вектора

int pos;

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

std::cin >> pos;

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

std::cout << "\nRemove element by index: " << std::endl;

print\_vector(v);

//

subtract\_min\_element(v); //каждый элемент разделить на мин. значение вектора

std::cout << "\nAdd the sum of the minimum and maximum elements: " << std::endl;

print\_vector(v);

if (pos > v.size())

throw 1;

}

catch (int)

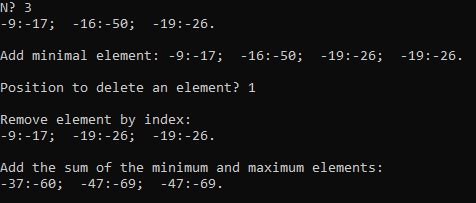
{

std::cout << "Error!" << std::endl;

}

###### **}**

###### **Объяснение результатов работы программы.**



Вводим в переменную N длину вектора для создания;

Создается вектор длиной N и заполняем его элементами типа Money;

Происходит поиск минимального элемента и добавление его в конец вектора;

Выбор элемента для удаления, затем происходит удаление элемента по индексу;

Добавление суммы минимального и максимального элементов каждому элементу вектора.

###### **Задача 3**

1. Создать последовательный контейнер.

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

Параметризированный класс – Вектор

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.

###### **Классы для решения задачи 3**

Vector.h

#pragma once

#include<iostream>

#include<vector>

#include<ctime>

template <class T>

class Vector

{

std::vector <T> v;

public:

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

Vector(int);

~Vector();

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

void Add();

void Print();

size\_t Size();

int min();

int max();

void remove\_element\_by\_index(int);

void add\_min\_max();

void add\_min\_el();

Vector& operator=(const Vector <T>&);

Vector operator+(const T&);

Vector operator\*(const Vector <T>&);

T& operator[](int);

};

template<class T>

inline Vector<T>::Vector(int n)

{

T a{};

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

v.push\_back(a);

}

template<class T>

inline Vector<T>::~Vector()

{

}

template<class T>

inline void Vector<T>::Add()

{

T a{};

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

{

std::cin >> a;

v[i] = a;

}

}

template<class T>

inline void Vector<T>::Print()

{

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

{

if (i)

std::cout << "; " << v[i];

else

std::cout << v[i];

}

std::cout << "." << std::endl;

}

template<class T>

inline size\_t Vector<T>::Size()

{

return v.size();

}

template<class T>

inline int Vector<T>::min()

{

T time = v[0];

int n = 0;

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

{

if (time > v[i])

{

time = v[i];

n = i;

}

}

return n;

}

template<class T>

inline int Vector<T>::max()

{

T time = v[0];

int n = 0;

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

{

if (time < v[i])

{

time = v[i];

n = i;

}

}

return n;

}

template<class T>

inline void Vector<T>::remove\_element\_by\_index(int pos)

{

v.erase(v.begin() + pos);

}

template<class T>

inline void Vector<T>::add\_min\_max()

{

T m = v[min()] + v[max()];

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

{

v[i] += m;

}

}

template<class T>

inline void Vector<T>::add\_min\_el()

{

v.push\_back(v[min()]);

}

template <class T>

Vector<T>& Vector<T>::operator=(const Vector<T>& temp)

{

if (this == &temp)

return \*this;

v.clear();

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

{

v.push\_back(temp.v[i]);

}

return \*this;

}

template <class T>

Vector<T> Vector<T>::operator+(const T& k)

{

Vector<T> temp(v.size());

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

temp.v[i] = v[i] + k;

return temp;

}

template <class T>

Vector<T> Vector<T>::operator\*(const Vector<T>& second)

{

Vector<T> temp(v.size());

size\_t n = 0;

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

{

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

{

if (v[i] == second.v[j])

{

temp.v[n] = v[i];

n++;

break;

}

}

}

if (!n)

{

Vector<T> null(0);

std::cout << "No intersections found" << std::endl;

return null;

}

T value = 0;

for (size\_t i = 0; (i < (n - 1)) && (i < (v.size() - 1)); i++)

{

for (size\_t j = i + 1; (j < n) && (j < v.size()); j++)

{

if (temp.v[i] >= temp.v[j])

{

value = temp.v[i];

temp.v[i] = temp.v[j];

temp.v[j] = value;

}

}

}

Vector<T> temp\_short(temp, n);

size\_t counter = 1;

for (size\_t i = 0; (i < (n - 1)) && n && (i < v.size()); i++)

{

if (temp.v[i] == temp.v[i + 1])

v.erase(i + 1);

}

return temp;

}

template <class T>

T& Vector<T>::operator[](int index)

{

if (index < v.size())

return v[index];

else

std::cout << "Error. Index > Size" << std::endl;

exit(1);

}

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&);

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);

//m\_rubles = 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);

}

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;

}

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

#include<iostream>

#include"Vector.h"

#include"Money.h"

int main()

{

int n;

std::cout << "N? ";

std::cin >> n;

Vector <Money> money(n);

money.Add();

money.Print();

//

std::cout << "\nAdd minimal element: ";

money.add\_min\_el();

money.Print();

int pos;

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

std::cin >> pos;

std::cout << "\nRemove element by index: " << std::endl;

money.remove\_element\_by\_index(pos);

money.Print();

//

std::cout << "\nAdd the sum of the minimum and maximum elements: " << std::endl;

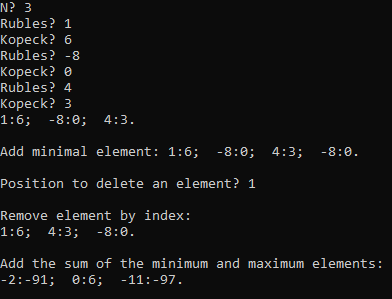
money.add\_min\_max();

money.Print();

}

###### 

###### **Объяснение результатов работы программы.**



Вводим в переменную N длину вектора для создания;

Создается вектор длиной N и заполняем его элементами типа Money;

Происходит поиск минимального элемента и добавление его в конец вектора;

Выбор элемента для удаления, затем происходит удаление элемента по индексу;

Добавление суммы минимального и максимального элементов каждому элементу вектора.

###### **Задача 4**

1. Создать последовательный контейнер.

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

Адаптер контейнера - очередь.

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 4

#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 4

int main()

{

Money t;

Qu q;

int n;

std::cout << "n? ";

std::cin >> n;

q = make\_queue(n);

print\_queue(q);

int pos;

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);

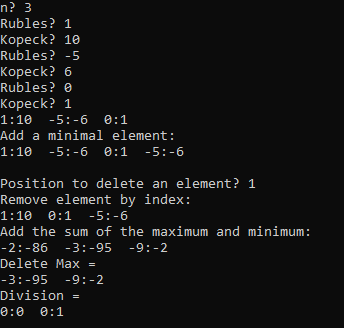
std::cout << "Division = " << std::endl;

Division(q);

print\_queue(q);

}

###### **Результат работы программы.**



Вводим в переменную N длину очереди для создания;

Создается очередь длиной N и заполняем его элементами типа Money;

Происходит поиск минимального элемента и добавление его в конец очереди;

Выбор элемента для удаления, затем происходит удаление элемента по индексу;

Добавление суммы минимального и максимального элементов каждому элементу очереди.

Удаление максимального элемента;

Деление на минимальный элемент всех элементов очереди.

###### **Задача 5**

Создать последовательный контейнер.

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

Параметризированный класс – Вектор

Адаптер контейнера - очередь.

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.

###### **Класс для решения задачи 5**

#pragma once

#include<iostream>

#include<queue>

#include<vector>

#include"Money.h"

template <class T>

class Vector

{

std::queue <T> q;

size\_t len;

public:

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

//Vector() :len(0) {};

Vector() {};

Vector(int);

Vector(const Vector<T>&);

~Vector();

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

void Print();

T arithmetic\_mean();

void Add\_to\_queue(const T&, int);

T Max();

void Delete\_from\_queue();

T Min();

void Division();

void add\_min\_element();

void remove\_element\_by\_index(int);

void add\_sum\_max\_min();

};

template<class T>

std::vector<T> copy\_queue\_to\_vector(std::queue <T> q)

{

std::vector<T> v;

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

{

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

q.pop();

}

return v; //вернуть вектор как результат функции

};

template<class T>

std::queue<T> copy\_vector\_to\_queue(std::vector<T> v) //êîïèðóåì âåêòîð â ñòåê

{

std::queue<T> q;

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

{

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

}

return q; //вернуть очередь как результат функции

};

template<class T>

Vector<T>::Vector(int n)

{

T a;

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

{

std::cin >> a;

q.push(a);

}

len = q.size();

}

template<class T>

inline Vector<T>::Vector(const Vector<T>& Vec)

{

len = Vec.len;

std::vector<T> v = copy\_queue\_to\_vector(Vec.q);

q = copy\_vector\_to\_queue(v);

}

template<class T>

inline Vector<T>::~Vector()

{

}

template<class T>

void Vector<T>::Print()

{

std::vector<T> v = copy\_queue\_to\_vector(q);

while (!q.empty())

{

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

q.pop();

}

std::cout << std::endl;

q = copy\_vector\_to\_queue(v);

}

template<class T>

T Vector<T>::arithmetic\_mean() //вычисление среднего значения

{

std::vector<T> v = copy\_queue\_to\_vector(q); //копируем значение для суммы

int n = 1;

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

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

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

{

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

q.pop(); //удалить элемент

n++;

}

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

return (sum / n);

}

template<class T>

void Vector<T>::Add\_to\_queue(const T& el, int pos) //добавление элемента в очередь

{

std::vector<T> v;

T t;

int i = q.size(); //размер очереди

if (!pos)

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

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

{

t = q.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

q.pop(); //удалить элемент из очереди

i--;

}

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

}

template<class T>

T Vector<T>::Max() // поиск максимального элемента в очереди

{

T m = q.front(); //переменной m присваивается значение из вершины очереди

std::vector<T> v = copy\_queue\_to\_vector(q); //копируем очередь в вектор

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

{

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

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

q.pop();

}

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

return m;

}

template<class T>

void Vector<T>::Delete\_from\_queue()//удалить максимальный элемент из очереди

{

T m = Max(); //находим максимальный элемент

std::vector<T> v;

T t;

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

{

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

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

v.push\_back(t);

q.pop(); //удалить элемент из очереди

}

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

}

template<class T>

T Vector<T>::Min()

{

T m = q.front();

std::vector<T> v = copy\_queue\_to\_vector(q);

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

{

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

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

q.pop();

}

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

return m;

}

template<class T>

void Vector<T>::Division()

{

T m = Min();

std::vector<T> v;

Money t;

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

{

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

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

q.pop(); //удалить элемент из вершины

}

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

}

template<class T>

void Vector<T>::add\_min\_element()

{

std::vector<T> v;

Money min = Min();

Money t;

size\_t i = 0;

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

{

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

v.push\_back(t);

q.pop(); //удалить элемент из стека

i++;

}

v.push\_back(min);

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

}

template<class T>

void Vector<T>::remove\_element\_by\_index(int pos)

{

std::vector<T> v;

T t;

size\_t i = 0;

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

{

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

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

v.push\_back(t);

q.pop(); //удалить элемент из стека

i++;

}

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

}

template<class T>

void Vector<T>::add\_sum\_max\_min()

{

std::vector<T> v;

T sum\_min\_max = Max() + Min();

T t;

size\_t i = 0;

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

{

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

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

q.pop(); //удалить элемент из стека

i++;

}

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

}

###### Основная программа для решения задачи 5

#include <iostream>

#include<vector>

#include"Money.h"

#include"Vector.h"

int main()

{

size\_t n{};

std::cout << "n? ";

std::cin >> n;

Vector<Money> v(n);

v.Print();

int pos;

v.add\_min\_element();

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

v.Print();

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

std::cin >> pos;

v.remove\_element\_by\_index(pos); //удалить элемент с этим номером

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

v.Print();

v.add\_sum\_max\_min();

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

v.Print();

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

v.Delete\_from\_queue();

v.Print();

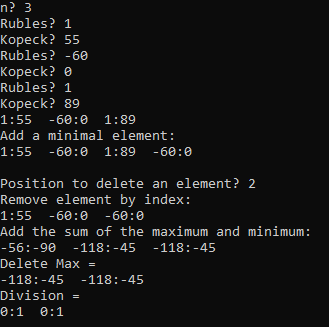
std::cout << "Division = " << std::endl;

v.Division();

v.Print();

}

###### Объяснение результатов работы программы.



Вводим в переменную N длину очереди для создания;

Создается очередь длиной N и заполняем его элементами типа Money;

Происходит поиск минимального элемента и добавление его в конец очереди;

Выбор элемента для удаления, затем происходит удаление элемента по индексу;

Добавление суммы минимального и максимального элементов каждому элементу очереди.

Удаление максимального элемента;

Деление на минимальный элемент всех элементов очереди.

###### **Ответы на контрольные вопросы.**

1. Из каких частей состоит библиотека STL?

Контейнеры, алгоритмы

1. Какие типы контейнеров существуют в STL?

list, vector, queue, map (multimap), set (multiset)

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

Подключить библиотеку.

1. Что представляет собой итератор?

Объект, который способен перебирать элементы контейнерного класса без необходимости пользователю знать реализацию определенного контейнерного класса.

1. Какие операции можно выполнять над итераторами?

Разыменование с целью получение значения объекта.

Присваивание одного итератору другому.

Сравнение итераторов на равенство.

Перемещение по элементам контейнера с помощью инкремента и декремента.

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

С использованием цикла for( i = first; I != last; ++i);

1. Какие типы итераторов существуют?

Входные, выходные, прямые, двунаправленные, произвольного доступа.

1. Перечислить операции и методы общие для всех контейнеров.

Копирование, присваивание, операции сравнения на равенство, инкремент, декремент, разыменование.

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

Вставка или удаление в конце вектора, произвольная выборка с помощью оператора индексации. Если вставлять или удалять в произвольное место вектора элемент, то придётся смещать все элементы справа путём копирования.

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

Вставка и удаление в любом месте контейнера эффективны, так как элементы хранятся в виде двусвязного списка. Для этих операций нужно только лишь поменять адреса указателей у двух соседних элементов.

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

Совмещает в себе эффективность вектора и списка, так как является неким гибридом вектора листа. Для него выделяются маленькие массивы, которые связаны между собой, как двусвязный список.

1. Перечислить методы, которые поддерживает последовательный контейнер vector.

Swap, clear, pushback(emplace\_back), erase, pop\_back, resize

1. Перечислить методы, которые поддерживает последовательный контейнер list.

Push\_back, push\_front, clear, erase, swap, splice, merge, sort

1. Перечислить методы, которые поддерживает последовательный контейнер deque.

Push\_back, push\_front, pop\_back, pop\_front, swap, insert, erase, clear.

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

erase(v.begin() + 2, v.begin() + 6);

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

pop\_back()

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

list<int>::iterator it1 = l.begin();

list<int>::iterator it2 = l.begin();

advance(it1, 2);

advance(it2, 6);

erase(it1, it2);

Так как для list<T>::iterator не перегружен operator+(), то нельзя поступить так же, как и с vector. Поэтому через advance() инкрементируем итератор на заданное кол-во элемнетов. и уже потом через erase удаляем диапазон элементов.

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

pop\_back()

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

erase(b.begin() + 2, b.begin() + 6);

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

pop\_back()

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

list<int> l{ 0, 1, 2, 3, 4, 5, 6, 7 };

list<int>::iterator it;

for (it = l.begin(); it != l.end(); ++it)

{

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

}

1. Что представляют собой адаптеры контейнеров?

Это специальные предопределенные контейнерные классы, которые адаптированы для выполнения конкретных заданий.

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

stack<int> s – объявлен стек s.

stack<int, list<int> > s – объявлен стек, за основу которого взят список.

1. Перечислить методы, которые поддерживает контейнер stack.

Push, pop, top, swap, size, empty.

1. Перечислить методы, которые поддерживает контейнер queue.

Pop, push, front, back, empty, size

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

Реализация priority\_queue возможно либо на основе вектора, либо списка. Извлечение из priority\_queue начинается с максимального элемента и по убыванию.

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

Конвертировать stack в любой последовательный контейнер. Удалить. Выполнить обратную конвертацию в stack.

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

Конвертировать queue в любой последовательный контейнер. Удалить. Выполнить обратную конвертацию в queue.

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

У stack нет итераторов.

void printStack(stack<int> st)

{

while (!st.empty())

{

cout << st.top() << " ";

st.pop();

}

}

Передаём в функцию копию стека и выводим её печать, при этом оригинал остаётся неизменный.

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

У queue нет интераторов

void printQueue(queue<int> q)

{

while (!q.empty())

{

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

q.pop();

}

}

Передаём в функцию копию очереди и выводим её печать, при этом оригинал остаётся неизменный.