Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №18.11**

Дисциплина: «Информатика»

Тема: Последовательные контейнеры библиотеки STL

Вариант 4

Выполнил:

Студент группы РИС-20-1б

Еске Вячеслав Сергеевич

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

**Пермь, 2021**

**Цель задачи**

* 1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
  2. Использование последовательных контейнеров библиотеки STL в ОО программе.

Для достижения поставленной цели необходимо решить следующие задачи:

* Проанализировать теоретические сведения о ООП
* Абстрактные типы данных. Контейнеры
* Реализовать алгоритм поставленной задачи
* Учесть все исключительные ситуации на ввод элементов
* Создать отдельные файлы: .cpp для описания методов класса, .cpp для описания главной функции, .h для описания класса

Постановка задачи

Задача 1

1. Контейнер - двунаправленная очередь

2. Тип элементов - int

Задача 2

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

Задача 3

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

Задача 4

Адаптер контейнера - очередь.

Задача 5

Параметризированный класс – Вектор Адаптер контейнера - очередь.

Задание 3

Найти максимальный элемент и добавить его в конец контейнера

Задание 4

Найти элемент с заданным ключом и удалить его из контейнера

Задание 5

К каждому элементу добавить среднее арифметическое элементов контейнера

Анализ задачи

1. Определить какие операции должны быть выполнены по заданию:

* Создание класса

class Money

class Vector

* Создание группы сумм, заполнение класса
* Создание методов получения элемента по номеру, печати, конструкторов
* Отчистка памяти

1. Для решения задачи используются переменные:

Несколько переменных целочисленного значения для ввода чисел и реализации меню

int s, in, menu = 4;

Указатель для работы с динамической памятью

1. Ввод данных осуществляется посредством функции

cin >> rub;

Вывод данных осуществляется посредством функции

cout << "Искомый элемент = " << a.give(in);

1. Создание метода класса , который печатает сумму:

void Money::Print()

{

cout << "\nРубли - " << rub << ", копейки - " << cop;

1. Создание методов класса для нахождения размера и доступа по индексу

int give(int index);

int givesize();

Создание контейнера

Money::Money(int r, long int c)

{

rub = r;

cop = c;

double co = c;

sum = r + co / 100;

}

1. Создание инструмента обработки исключительных ситуаций. Для этого была использована логика try:

try

{

cout << "Искомый элемент: " << a.give(in);

}

catch (const int in)

{

cout << "Ошибка ввода";

}

}

Код

#include <iostream>

#include "Vector.h"

#include "Time.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

Vector<Time>vec(5);//создать вектор из 5 элементов

vec.Print();//печать вектора

Time max,k;

max = 0;

vec.Add(max);

vec.Print();//печать вектора

cout << "Введите ключ: ";

cin >> k;

vec.Del(k);

vec.Print();//печать вектора

Time s = vec.Srednee();

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

vec.Sum(s);

vec.Print();//печать вектора

}

#pragma once

#include <vector>

#include <iostream>

using namespace std;

//шаблон класса

template<class T> class Vector

{

vector <T> v;//последовательный контейнер для хранения элементов вектора

int len;

public:

Vector(void);//конструктор без параметра

Vector(int n);//конструктор с параметром

void Print();//печать

~Vector(void);//деструктор

void Add(T el);//добавление элемента el на позицию pos

void Del(T k);//удалить элемент из позиции pos

T Srednee();//вычисление среднего арифметического

void Sum(T k);//удалить элемент из позиции pos

};

//конструктор без параметра

template <class T> Vector<T>::Vector()

{

len = 0;

}

//деструктор

template <class T>

Vector<T>::~Vector(void) { }

//конструктор с параметром

template <class T> Vector<T>::Vector(int n)

{

T a;

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

{

cin >> a; v.push\_back(a);

}

len = v.size();

}

//печать

template <class T> void Vector<T>::Print()

{

for (int i = 0; i < v.size(); i++) cout << v[i] << " ";

cout << endl;

}

template<class T> void Vector<T>::Add(T el)

{

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

{

if (v[i] > el) el = v[i];

}

v.push\_back(el);

}

template<class T> void Vector<T>::Del(T k)

{

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

{

if (v[i] == k)

{

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

}

}

}

template<class T> T Vector<T>::Srednee()

{

T s = v[0];

for (int i = 1; i < v.size(); i++) s = s + v[i];

int n = v.size();

return s / n;

}

template<class T> void Vector<T>::Sum(T k)

{

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

{

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

}

}

#include <iostream>

using namespace std;

class Time {

public:

Time(void);

Time(int, int);

Time(const Time&);

Time& operator=(const Time&); //перегруженные операции ввода-вывода

Time& operator=(int);

bool operator >(const Time&);

bool operator ==(const Time&);

Time operator+(const Time&);

Time operator/(const Time&);

Time operator/(const int&);

friend ostream& operator<< (ostream& out, const Time&);

friend istream& operator>> (istream& in, Time&);

Time operator\*(Time k);

public:

virtual ~Time(void) {};

private:

int min, sec;

};

#include "Time.h"

Time::Time(void)

{

min = sec = 0;

}

Time::Time(int M, int S)

{

min = M;

sec = S;

}

Time::Time(const Time& t)

{

min = t.min;

sec = t.sec;

}

Time& Time::operator =(const Time& t)

{

min = t.min;

sec = t.sec;

return\*this;

}

ostream& operator<<(ostream& out, const Time& t)

{

out << t.min << ":" << t.sec;

return out;

}

istream& operator>>(istream& in, Time& t)

{

cout << "\nВведите минуты: "; in >> t.min;

do {

cout << "\nВведите секунды: "; in >> t.sec;

} while ((t.sec < 0) || (t.sec >= 60));

return in;

}

Time Time::operator\*(Time k)

{

int t = min \* k.min \* 60;

int kt = sec \* k.sec;

t += kt;

Time temp(t / 60, t % 60);

return temp;

}

Time& Time::operator=(int t)

{

min = t;

sec = t;

return \*this;

}

bool Time::operator >(const Time& t) {

if (min > t.min)return true;

if (min == t.min && sec > t.sec)

return true;

return false;

}

bool Time::operator ==(const Time& t) {

if (min == t.min && sec == t.sec)

return true;

return false;

}

Time Time::operator+(const Time& t) {

int temp1 = min \* 60 + sec;

int temp2 = t.min \* 60 + t.sec;

Time p;

p.min = (temp1 + temp2) / 60;

p.sec = (temp1 + temp2) % 60;

return p;

}

Time Time::operator/(const Time& t) {

int temp1 = min \* 60 + sec;

int temp2 = t.min \* 60 + t.sec;

Time p;

p.min = (temp1 / temp2) / 60;

p.sec = (temp1 / temp2) % 60;

return p;

}

Time Time::operator/(const int& t) {

int temp1 = min \* 60 + sec;

Time p;

p.min = (temp1 / t) / 60;

p.sec = (temp1 / t) % 60;

return p;

}

#include <iostream>

#include "Time.h"

#include <queue>

#include <vector>

typedef queue<Time>q;

typedef vector<Time>v;

q make\_queue(int n) {

q que;

Time t;

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

{

cin >> t;

que.push(t);

}

return que;

}

v copy\_queue\_to\_vector(q que) {

v vec;

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

{

//добавить в вектор элемент из вершиы стека

vec.push\_back(que.front());

que.pop();

}

return vec; //вернуть вектор как результат функции

}

void copy\_vector\_to\_queue(v& vec,q& que) {

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

{

que.push(vec[i]);

}

}

void print\_queue(q que)

{

while (!que.empty())

{

Time val = que.front();

que.pop();

cout << val << " ";

}

cout << endl;

}

Time Max(q que) {

Time m = que.front();

v vec = copy\_queue\_to\_vector(que);

while (!que.empty()) {

if (que.front() > m) m = que.front();

que.pop();

}

copy\_vector\_to\_queue(vec,que);

return m;

}

void Delete\_from\_queue(q& que, Time k) {

v vec;

Time t;

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

{

t = que.front();//получаем элемент из вершины стека

//если он не равен максимальному, заносим элемент в вектор

if(t!=k) vec.push\_back(t);

que.pop();//удаляем элемент из стека

}

copy\_vector\_to\_queue(vec,que);//копируем вектор в стек

}

Time Srednee(q que)

{

v vec = copy\_queue\_to\_vector(que);//копируем стек в вектор

int n=1;

Time sum=que.front();//начальное значениедля суммы

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

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

{

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

que.pop();//удалить элемент

n++;

}

copy\_vector\_to\_queue(vec,que);//скопировать вектор в стек

return sum/n; //вернуть среднее арифметическое

}

void Sum(q& que)

{

Time m = Srednee(que);

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

v vec; Time t;

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

{

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

vec.push\_back(t+m);

que.pop();//удаляем элемент из вершины

}

copy\_vector\_to\_queue(vec,que);//копируем вектор в стек

}

int main()

{

setlocale(LC\_ALL, "rus");

Time t,k;

q que;

int n;

cout << "n?";

cin >> n;

que = make\_queue(n);//создать стек

print\_queue(que);//печать стека

que.push(Max(que));

print\_queue(que);//печать стека

cout << "Введите ключ для удаления: ";

cin >> k;

Delete\_from\_queue(que, k);

print\_queue(que);//печать стека

Sum(que);

print\_queue(que);//печать стека

}

#pragma once

#include <iostream>

using namespace std;

class Time {

int min, sec;

public:

Time() { min = 0; sec = 0; };

Time(int m, int s) { min = m; sec = s; }

Time(const Time& t) { min = t.min; sec = t.sec; }

~Time() {};

int get\_min() { return min; }

int get\_sec() { return sec; }

void set\_min(int m) { min = m; }

void set\_sec(int s) { sec = s; }

Time& operator=(const Time&);

Time& operator=(int);

Time operator+(const Time&);

Time operator/(const Time&);

Time operator/(const int&);

bool operator >(const Time&);

bool operator <(const Time&);

bool operator ==(const Time&);

bool operator !=(const Time&);

friend istream& operator>>(istream& in, Time& t);

friend ostream& operator<<(ostream& out, const Time& t);

};

#include "Time.h"

#include <iostream>

using namespace std;

//перегрузка операции присваивания

Time& Time::operator=(const Time& t) {

//проверка на самоприсваивание

if (&t == this) return \*this;

min = t.min;

sec = t.sec;

return \*this;

}

Time& Time::operator=(int t)

{

min = t;

sec = t;

return \*this;

}

//перегрузка глобальной функции-операции ввода

istream& operator>>(istream& in, Time& t) {

cout << "Введите минуты - ";

in >> t.min;

do {

cout << "Введите секунды - ";

in >> t.sec;

} while ((t.sec < 0) || (t.sec >= 60));

return in;

}

//перегрузка глобальной функции-операции вывода

ostream& operator<<(ostream& out, const Time& t) {

return (out << t.min << ":" << t.sec);

}

bool Time::operator <(const Time& t) {

if (min < t.min)return true;

if (min == t.min && sec < t.sec)

return true;

return false;

}

bool Time::operator >(const Time& t) {

if (min > t.min)return true;

if (min == t.min && sec > t.sec)

return true;

return false;

}

bool Time::operator ==(const Time& t) {

if (min == t.min && sec == t.sec)

return true;

return false;

}

bool Time::operator !=(const Time& t) {

if (min != t.min || sec != t.sec)

return true;

return false;

}

Time Time::operator+(const Time& t) {

int temp1 = min \* 60 + sec;

int temp2 = t.min \* 60 + t.sec;

Time p;

p.min = (temp1 + temp2) / 60;

p.sec = (temp1 + temp2) % 60;

return p;

}

//перегрузка бинарной операции деления

Time Time::operator/(const Time& t) {

int temp1 = min \* 60 + sec;

int temp2 = t.min \* 60 + t.sec;

Time p;

p.min = (temp1 / temp2) / 60;

p.sec = (temp1 / temp2) % 60;

return p;

}

Time Time::operator/(const int& t) {

int temp1 = min \* 60 + sec;

Time p;

p.min = (temp1 / t) / 60;

p.sec = (temp1 / t) % 60;

return p;

}

#include <iostream>

#include "Time.h"

#include <queue>

#include "Vector.h"

int main()

{

setlocale(LC\_ALL, "rus");

Vector<Time>v(3);

Time k;

v.Print();

v.Add();

v.Print();

cout << "Введите ключ для удаления: ";

cin >> k;

v.Del(k);

v.Print();

v.Sum();

v.Print();

}

#pragma once

#include <iostream>

using namespace std;

class Time {

int min, sec;

public:

Time() { min = 0; sec = 0; };

Time(int m, int s) { min = m; sec = s; }

Time(const Time& t) { min = t.min; sec = t.sec; }

~Time() {};

int get\_min() { return min; }

int get\_sec() { return sec; }

void set\_min(int m) { min = m; }

void set\_sec(int s) { sec = s; }

Time& operator=(const Time&);

Time& operator=(int);

Time operator+(const Time&);

Time operator/(const Time&);

Time operator/(const int&);

bool operator >(const Time&);

bool operator <(const Time&);

bool operator ==(const Time&);

bool operator !=(const Time&);

friend istream& operator>>(istream& in, Time& t);

friend ostream& operator<<(ostream& out, const Time& t);

};

#include "Time.h"

#include <iostream>

using namespace std;

//перегрузка операции присваивания

Time& Time::operator=(const Time& t) {

//проверка на самоприсваивание

if (&t == this) return \*this;

min = t.min;

sec = t.sec;

return \*this;

}

Time& Time::operator=(int t)

{

min = t;

sec = t;

return \*this;

}

//перегрузка глобальной функции-операции ввода

istream& operator>>(istream& in, Time& t) {

cout << "Введите минуты - ";

in >> t.min;

do {

cout << "Введите секунды - ";

in >> t.sec;

} while ((t.sec < 0) || (t.sec >= 60));

return in;

}

//перегрузка глобальной функции-операции вывода

ostream& operator<<(ostream& out, const Time& t) {

return (out << t.min << ":" << t.sec);

}

bool Time::operator <(const Time& t) {

if (min < t.min)return true;

if (min == t.min && sec < t.sec)

return true;

return false;

}

bool Time::operator >(const Time& t) {

if (min > t.min)return true;

if (min == t.min && sec > t.sec)

return true;

return false;

}

bool Time::operator ==(const Time& t) {

if (min == t.min && sec == t.sec)

return true;

return false;

}

bool Time::operator !=(const Time& t) {

if (min != t.min || sec != t.sec)

return true;

return false;

}

Time Time::operator+(const Time& t) {

int temp1 = min \* 60 + sec;

int temp2 = t.min \* 60 + t.sec;

Time p;

p.min = (temp1 + temp2) / 60;

p.sec = (temp1 + temp2) % 60;

return p;

}

//перегрузка бинарной операции деления

Time Time::operator/(const Time& t) {

int temp1 = min \* 60 + sec;

int temp2 = t.min \* 60 + t.sec;

Time p;

p.min = (temp1 / temp2) / 60;

p.sec = (temp1 / temp2) % 60;

return p;

}

Time Time::operator/(const int& t) {

int temp1 = min \* 60 + sec;

Time p;

p.min = (temp1 / t) / 60;

p.sec = (temp1 / t) % 60;

return p;

}

#pragma once

#include <vector>

#include <queue>

#include <iostream>

using namespace std;

//шаблон класса

template<class T> class Vector

{

queue <T> q;//последовательный контейнер для хранения элементов вектора

int len;

public:

Vector(void);//конструктор без параметра

Vector(int n);//конструктор с параметром

Vector(const Vector<T>&);//конструктор копирования

void Print();//печать

~Vector(void);//деструктор

T Max();

void Add();

void Del(T);

T Srednee();

void Sum();

};

//конструктор без параметра

template <class T> Vector<T>::Vector()

{

len = 0;

}

//деструктор

template <class T>

Vector<T>::~Vector(void) { }

//конструктор с параметром

template <class T> Vector<T>::Vector(int n)

{

T a;

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

{

cin >> a;

q.push(a);

}

len = q.size();

}

//конструктор копирования

template <class T> Vector<T>::Vector(const Vector<T> &v)

{

len = v.len;

//копируем значения стека Vec.q в вектор v vector

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

//копируем вектор v в стек s

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

}

//печать

template <class T> void Vector<T>::Print()

{

//копируем стек в вектор

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

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

{

cout << q.front() << "\t";//вывод элемента в вершине стека

q.pop();//удаляем элемент из вершины

}

cout << endl;

//копируем вектор в стек

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

}

template <class T> vector<T> copy\_queue\_to\_vector(queue<T> q) {

vector<T> v;

while (!q.empty())

{

v.push\_back(q.front()); q.pop();

} return v;

}

//копирование вектора в стек

template <class T> queue<T> copy\_vector\_to\_queue(vector<T> v)

{

queue<T> q;

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

{

q.push(v[i]);

}

return q;

}

template <class T> T Vector<T>::Max()

{

T m = q.front();//m присвоить значение из вершины стека

//в вектор скопировать элементы стека

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

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

{

//сравниваем m и элемент в вершине стека

if(q.front()>m) m=q.front();

q.pop();//удаляем элемент из вершины стека

}

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

return m;

}

template <class T> void Vector<T>::Add()

{

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

T m = Max();

v.push\_back(m);

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

}

template <class T> void Vector<T>::Del(T k)

{

vector<T> v;

T t;

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

{

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

//если t не равен максимальному, то добавить его в вектор

if(t!=k) v.push\_back(t);

q.pop();//удалить элемент из стека

}

//копируем вектор в стек

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

}

//среднее арифметическое

template <class T> T Vector<T>::Srednee() {

//копируем стек в вектор

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

int n = 1;//количество элементов в стеке

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

q.pop();//удаляем элемент из вершины стека

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

{

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

q.pop();//удаляем элемент из вершины стека

n++;//увеличиваем количество элементов

}

//копируем вектор в стек

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

return sum/n;

}

template <class T> void Vector<T>::Sum() {

T m = Srednee();

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

vector<T> v;

T t;

while (!q.empty())

{

t = q.front();

v.push\_back(t+m); q.pop();

}

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

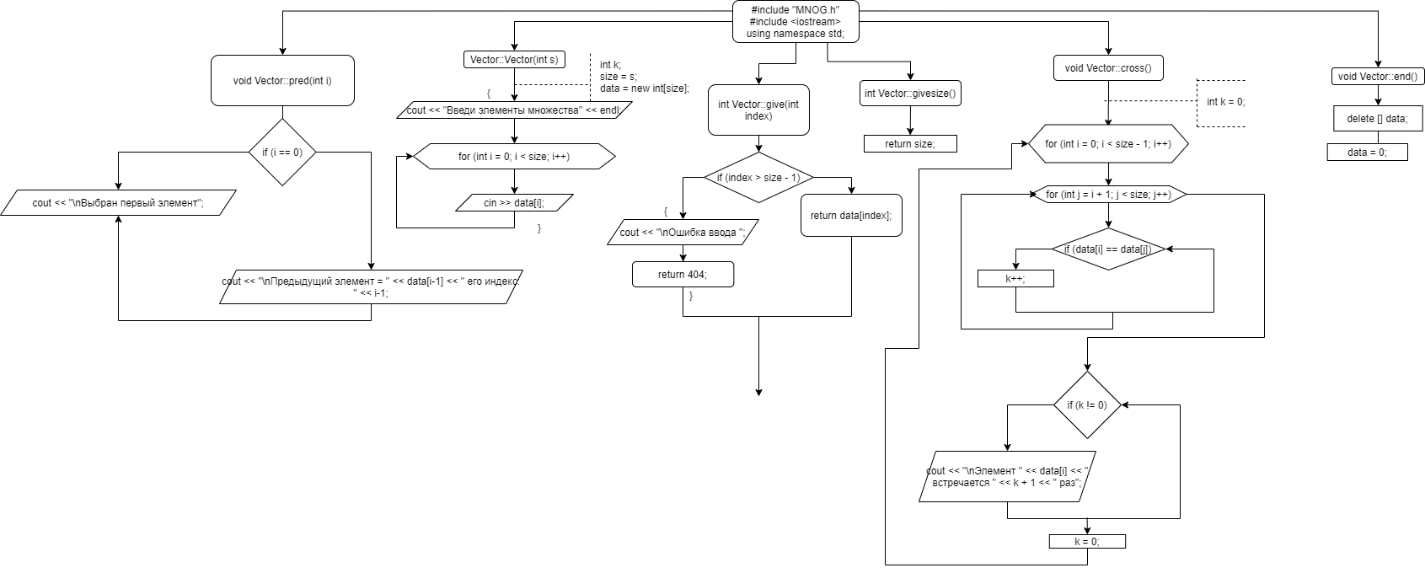
}

Блок-схема:

Main.cpp:



MHETODS.cpp:



CLASS.h:



Работа кода

