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 №18.13**

Дисциплина: «Информатика»

Тема: Стандартные обобщенные алгоритмы 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 Time (см. лабораторную работу №3).

Задача 2

Адаптер контейнера - очередь.

Задача 3

Ассоциативный контейнер – словарь с дубликатами

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 <deque>

#include <algorithm>

#include "Time.h"

using namespace std;

typedef deque<Time> Tdeq;

Time s;

Tdeq make\_deque(int n)

{

Time a;

Tdeq d;

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

{

cin >> a;

d.push\_back(a);

}

return d;

}

void print\_deque(Tdeq d)

{

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

cout << d[i] << endl;

cout << endl;

}

struct Equal\_s {

bool operator()(Time t) {

return t == s;

}

};

Time srednee(Tdeq v) {

Time s = v[0];

//перебор вектора

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

int n = v.size();//количество элементов в векторе

return s/n;

}

void sum(Time& t)

{

t = t + s;

}

int main()

{

setlocale(LC\_ALL, "rus");

Tdeq::iterator i;

int n;

cout << "Введите количество элементов: ";

cin >> n;

Tdeq v;

v = make\_deque(n);

print\_deque(v);

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

s = \*i;

cout << "Максимальный элемент: " << s << endl;

v.push\_back(s);

print\_deque(v);

cout << "Введите ключ для удаления: ";

cin >> s;

i = remove\_if(v.begin(), v.end(), Equal\_s());

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

print\_deque(v);

s = srednee(v);

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

for\_each(v.begin(), v.end(), sum);

print\_deque(v);

}

#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"

//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 <deque>

#include <queue>

using namespace std;

typedef queue<Time>q;

typedef deque<Time>d;

Time s;

q make\_queue(int n)

{

Time a;

q que;

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

{

cin >> a;

que.push(a);

}

return que;

}

d copy\_queue\_to\_deque(q que)

{

d deq;

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

{

//добавить в вектор элемент из вершиы стека

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

que.pop();

}

return deq; //вернуть вектор как результат функции

}

void copy\_deque\_to\_queue(d& deq, q& que)

{

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

{

que.push(deq[i]);

}

}

void print\_queue(q que)

{

while (!que.empty())

{

Time val = que.front();

que.pop();

cout << val << " ";

}

cout << endl;

}

struct Equal\_s {

bool operator()(Time t) {

return t == s;

}

};

Time srednee(d deq) {

Time s = deq[0];

//перебор вектора

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

int n = deq.size();//количество элементов в векторе

return s / n;

}

void sum(Time& t)

{

t = t + s;

}

int main()

{

setlocale(LC\_ALL, "rus");

d::iterator i;

q que;

d deq;

int n;

cout << "Введите количество элементов: ";

cin >> n;

que = make\_queue(n);

print\_queue(que);

deq = copy\_queue\_to\_deque(que);

i = max\_element(deq.begin(), deq.end());

s = \*i;

cout << "Максимальный элемент: " << s << endl;

deq.push\_back(s);

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

{

que.pop();//удалить элемент

}

copy\_deque\_to\_queue(deq, que);

print\_queue(que);

cout << "Введите ключ для удаления: ";

cin >> s;

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

{

deq.pop\_back();//удалить элемент

}

deq = copy\_queue\_to\_deque(que);

i = remove\_if(deq.begin(), deq.end(), Equal\_s());

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

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

{

que.pop();//удалить элемент

}

copy\_deque\_to\_queue(deq, que);

print\_queue(que);

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

{

deq.pop\_back();//удалить элемент

}

deq = copy\_queue\_to\_deque(que);

s = srednee(deq);

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

for\_each(deq.begin(), deq.end(), sum);

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

{

que.pop();//удалить элемент

}

copy\_deque\_to\_queue(deq, que);

print\_queue(que);

}

#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"

//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 <map>

#include <algorithm>

#include "Time.h"

using namespace std;

typedef map<int, Time>TMap;//определяем тип для работы со словарем

typedef TMap::iterator it;//итератор

Time s;

TMap make\_map(int n)

{

TMap m;//пустой словарь

Time a;

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

{

cout << "Введите элемент: "; cin >> a;

//создаем пару и добавляем ее в словарь

m.insert(make\_pair(i, a));

}

return m;//возвращаем словарь как результат работы функции

}

//функция для печати словаря

void print\_map(TMap m)

{

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

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

}

Time srednee(TMap v) {

Time s = v[0];

//перебор вектора

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

int n = v.size();//количество элементов в векторе

return s / n;

}

void sum(TMap& v) {

Time sr = srednee(v);

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

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

}

int Max(TMap v)

{

it i = v.begin();

int nom = 0,//номер максимального

k = 0;//счетчик элементов

Time m = (\*i).second;//значение первого элемента

while (i != v.end())

{

if (m < (\*i).second)

{

m = (\*i).second;

nom = k;

}

i++;//итератор

k++;//счетчик элементов

}

return nom;//номер max

}

struct Equal\_s {

bool operator()(Time t) {

return t == s;

}

};

int main()

{

setlocale(LC\_ALL, "rus");

int n, k;

cout << "Введите количество элементов: ";

cin >> n;//количество элементов

it i;

TMap m = make\_map(n);//создать словарь

int max = Max(m);

print\_map(m);//напечатать словарь

m.insert(make\_pair(n, m[max]));

print\_map(m);//напечатать словарь

cout << "Введите ключ для удаления: ";

cin >> k;

m.erase(k);

print\_map(m);//напечатать словарь

sum(m);

print\_map(m);//напечатать словарь

}

#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"

//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;

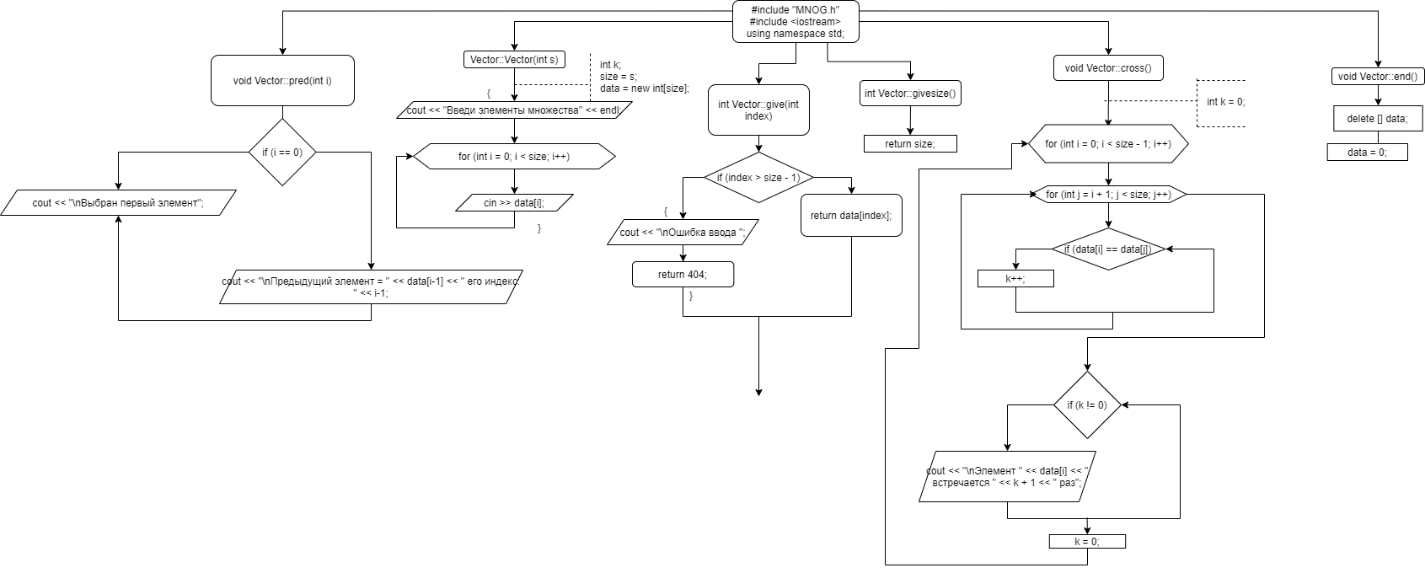
}

Блок-схема:

Main.cpp:



MHETODS.cpp:



CLASS.h:



Работа кода

