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: “ Объектно-ориентированное программирование.

Стандартные обобщённые алгоритмы библиотеки STL.”

Вариант 19

Выполнил:

Студент группы РИС-20-2Б

Пономарёв Артём Викторович

Проверила:

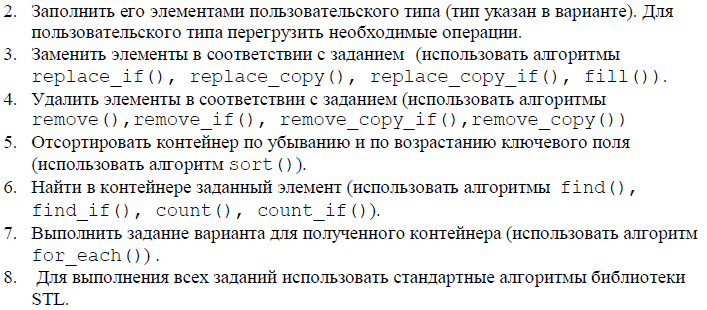
Доцент кафедры ИТАС

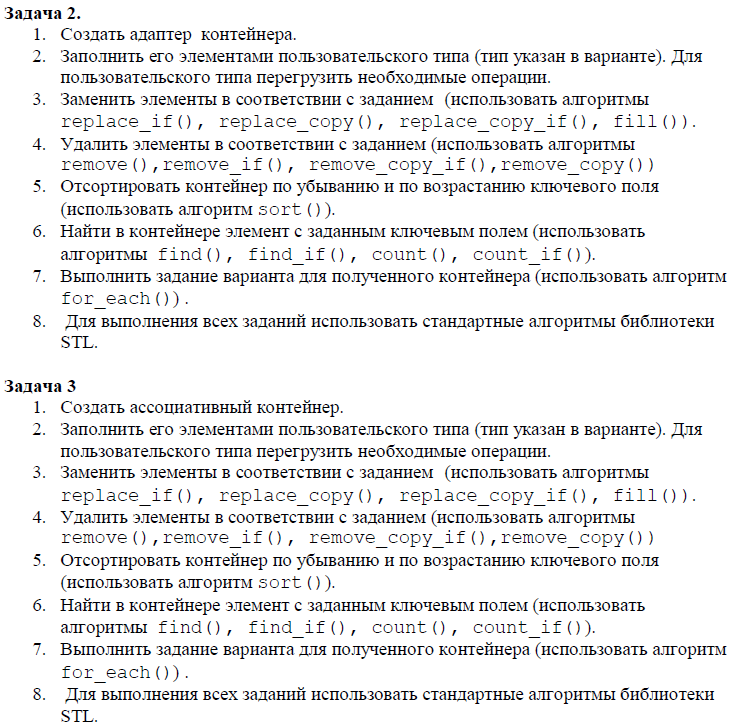
Полякова О.А.

Пермь, 2021

**Постановка задачи**







**Анализ задачи**

**1.** Для решения задач необходимо:

**1.1.** Организовать функцию Problem1(), в которой будет использоваться библиотека deque для создания двунаправленной типа Time, для решения задачи №1.

**1.2.** Организовать функцию Problem2(), в которой будет использоваться библиотека queue для создания очереди типа Time, для решения задачи №2.

**1.3.** Организовать класс Time для хранения в полях для минут и секунд (int: mins, secs) для экземпляров данного класса.

**1.4.** Организовать функцию Problem3(), в которой будет использоваться библиотека map для создания словаря с дубликатами типа Time, для решения задачи №3.

**2.** В ходе работы были использованы следующие типы данных:

**2.1.** В функции main() используется пременная vvod типа int для меню, реализованного через ветвление switch().

char vvod = '6';

while (vvod != '0') {

cout << "\nКакое задание?" << endl

<< "1) Задание 1;" << endl

<< "2) Задание 2;" << endl

<< "3) Задание 3;" << endl

<< "0) Выход из меню" << "\n\n" << ">";

cin >> vvod;

switch (vvod)

{

case '1':

{

Problem1();

}

break;

case '2':

{

Problem2();

}

break;

case '3':

{

Problem3();

}

break;

default:

if (vvod != '0')

cout << "ERROR!";

break;

}

}

**2.2.** Функция Problem1(), в которой будет использоваться библиотека deque для создания двунаправленной очереди – line, типа Time, для решения задачи №1. Переменная size устанавливает размер двунаправленной очереди.

deque<Time> line, temp;

cout << "Введите размерность двунаправленной очереди: ";

int size = SafeInput1(true);

**2.3.** Функция Problem2(), в которой будет использоваться библиотека queue для создания очереди – line, типа Time, для решения задачи №2. Переменная size устанавливает размер очереди.

queue<Time> line,temp;

cout << "Введите размерность очереди: ";

int size = SafeInput1(true);

**2.4.** Функция Problem3(), в которой будет использоваться, в которой будет использоваться библиотека map для создания словаря с дубликатами (multimap) - map, типа int для ключа и типа Time для хранящихся данных, для решения задачи №3. Переменная size устанавливает размер контейнера.

multimap <int, Time> map;

cout << "Введите размерность ассоциативного контейнера multimap: ";

int size = SafeInput1(true);

class Time {

protected:

int mins, secs;

public:

Time();

Time(int minute, int second);

~Time();

int GetMins();

int GetSecs();

int getM() { return mins; }

int getS() { return secs; }

void SetMins(int value);

void SetSecs(int value);

void Show();

int GetValueToComapare() { int result = secs + (mins \* 60); return result; }

friend bool operator >(Time& t1, Time& t2) { return (t1.GetValueToComapare() > t2.GetValueToComapare()); }

friend bool operator <(Time& t1, Time& t2) { return (t1.GetValueToComapare() < t2.GetValueToComapare()); }

Time operator+(const Time& t);

Time& operator=(const Time& other);

friend bool operator==(const Time& t1, const Time& t2);

friend bool operator!=(const Time& t1, const Time& t2);

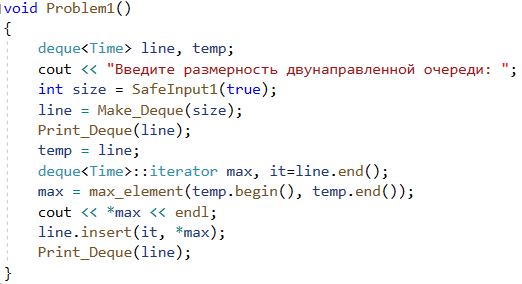
friend ostream& operator<<(ostream& out, const Time& t);

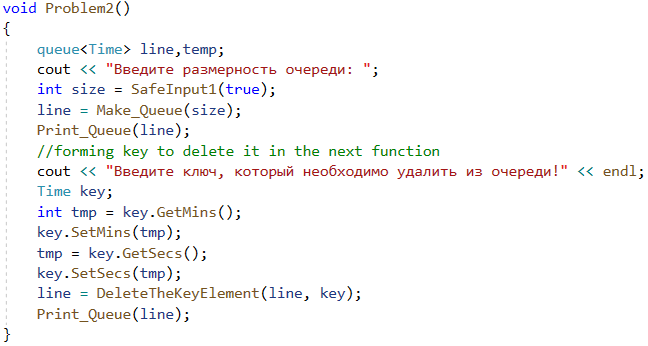
friend istream& operator>>(istream& in, Time& t);

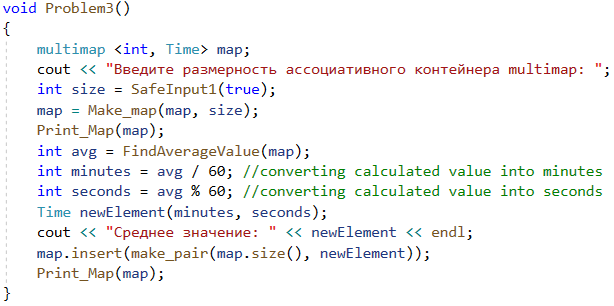
};

**3.** Для решения задачи данные были представлены в следующем виде:

**3.1.** Решения задач осуществляются в функциях Problem1(), Problem2(), Problem3().

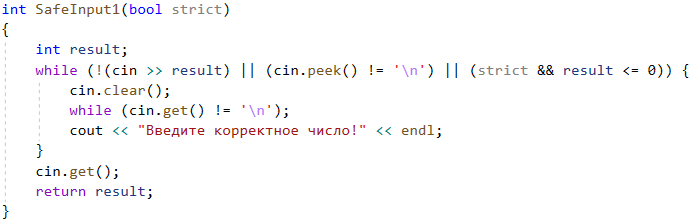




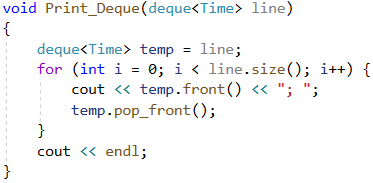


**4.** Для операций ввода и вывода использовались следующие операторы и функции:

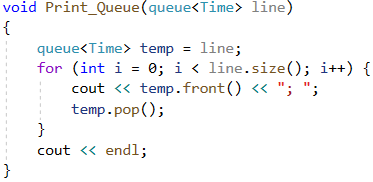
**4.1.** Для ввода целочисленных значений используется функция SafeInput1(), которая проверяет введённое данное. Таким образом будут игнорироваться данные типа char, double, float.



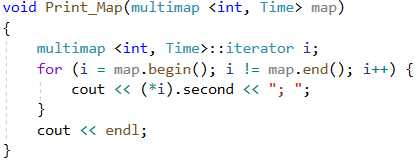
**4.2.** Для печати двунаправленной очереди типа Time используется функция Print\_Deque().



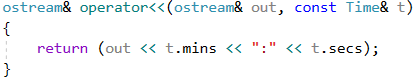
**4.3.** Для печати двунаправленной очереди типа Time используется функция Print\_Queue().



**4.4.** Для печати словаря с дубликатами типа Time используется функция Print\_Map().

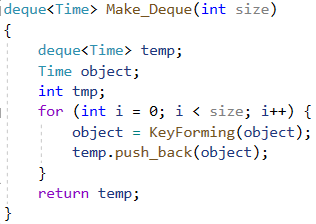


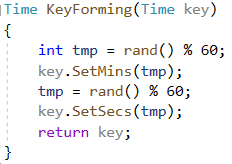
**4.5.** Для печати объектов класса Time используется перегрузка оператора сдвига.



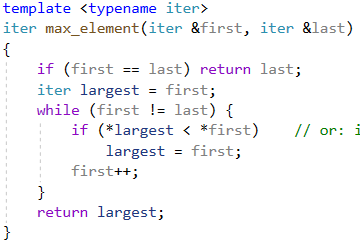
**5.** Поставленные задачи будут решены следующими действиями:

**5.1.** Для формирования двунаправленной очереди типа Time в функции Problem1() используется функция Make\_Deque(). Используется цикл for, который итерируется по всему контейнеру и элементы добавляются через метод push\_back, прописанный в библиотеке deque. Добавляемый элемент генерируется в функции KeyForming().



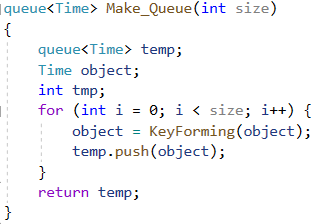


**5.2.** Для поиска максимального элемента используется алгоритм max\_element() из библиотеки algorithm. Однако для корректной работы с классом Time понадобилось перегрузить этот алгоритм:

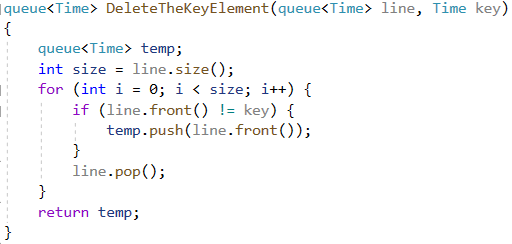


Для вывода этого максимального алгоритма используется итератор для двунаправленной очереди – max, который добавляется через метод insert библиотеки deque.

**5.3.** Для формирования очереди типа Time в функции Problem2() используется функция Make\_Queue(). Используется цикл for, который итерируется по всему контейнеру и элементы добавляются через метод push, прописанный в библиотеке queue. Добавляемый элемент генерируется в функции KeyForming().

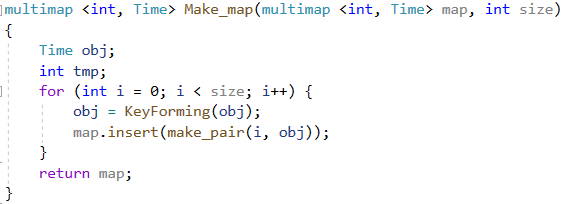


**5.4.** Для удаления введённого пользователем ключа используется функция DeleteTheKeyElement().

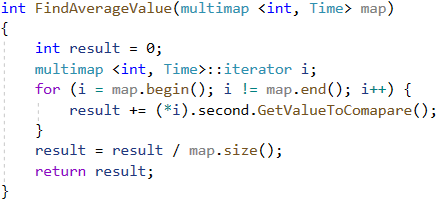


Используется цикл for, который итерируется по всей очереди и в ветвлении сравниваются все элементы очереди. Если текущий элемент очереди не равен ключу, то он переписывается в другую очередь – temp, через метод push, в противном случае он не запишется. После отработки цикла возвращается изменённая очередь temp.

**5.5.** Для формирования контейнера multimap типа Time в функции Problem3() используется функция Make\_Map(). Используется цикл for, который итерируется по всему контейнеру и элементы добавляются через метод insert, в который передаётся функция make\_pair(), прописанная в библиотеке map, прописанный также в библиотеке map. Добавляемый элемент генерируется в функции KeyForming().



**5.6.** Для поиска среднего арифметического значения используется функция FindAverageValue().



В цикле for происходит итерирование через итератор контейнера multimap. В переменную result добавляется значение second (в котором хранятся данные класса Time) и через метод GetValueToCompare() происходит перевод минут и секунд объектов в секунды для вычисления среднего значения. После цикла происходит деление суммы секунд, на количество объектов в контейнере.

Далее значение result возвращается в функцию Problem3(), в которой данное значение конвертируется в минуты и секунды и помещаются в обхект newElement класса Time.

int avg = FindAverageValue(map);

int minutes = avg / 60; //converting calculated value into minutes

int seconds = avg % 60; //converting calculated value into seconds

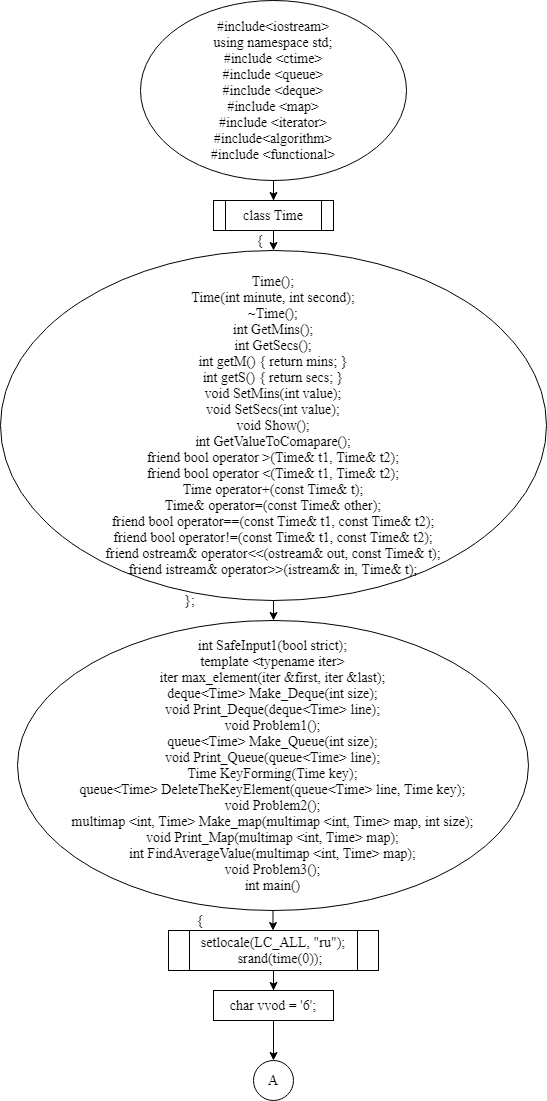
Time newElement(minutes, seconds);

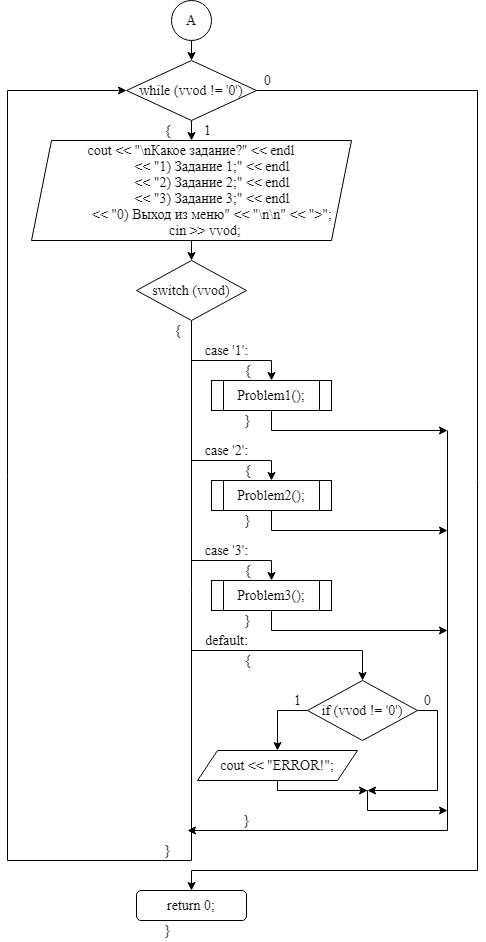
cout << "Среднее значение: " << newElement << endl;

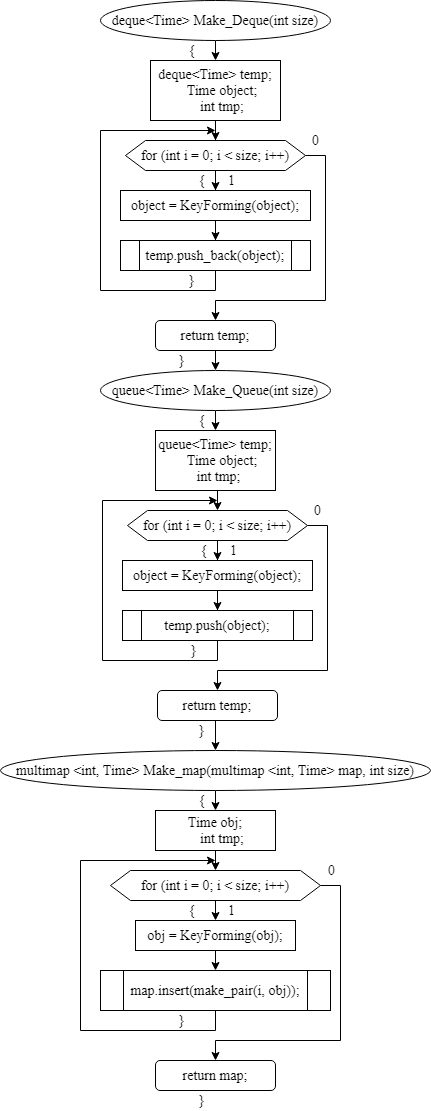
**5.7.** Добавление среднего значения происходит через метод insert(), в качестве параметра которого отправляется функция make\_pair().

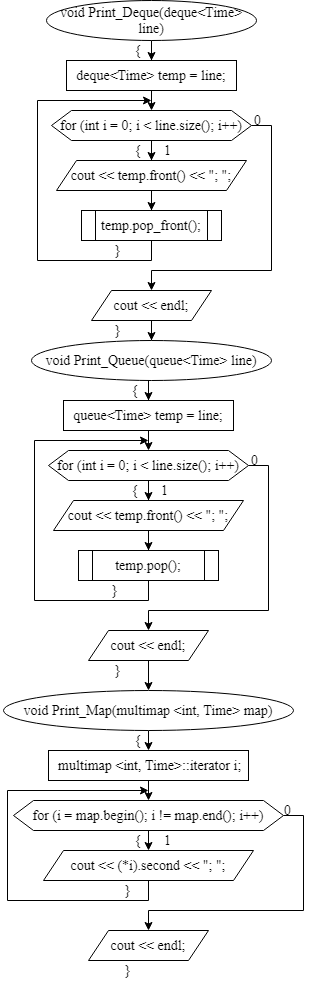
map.insert(make\_pair(map.size(), newElement));

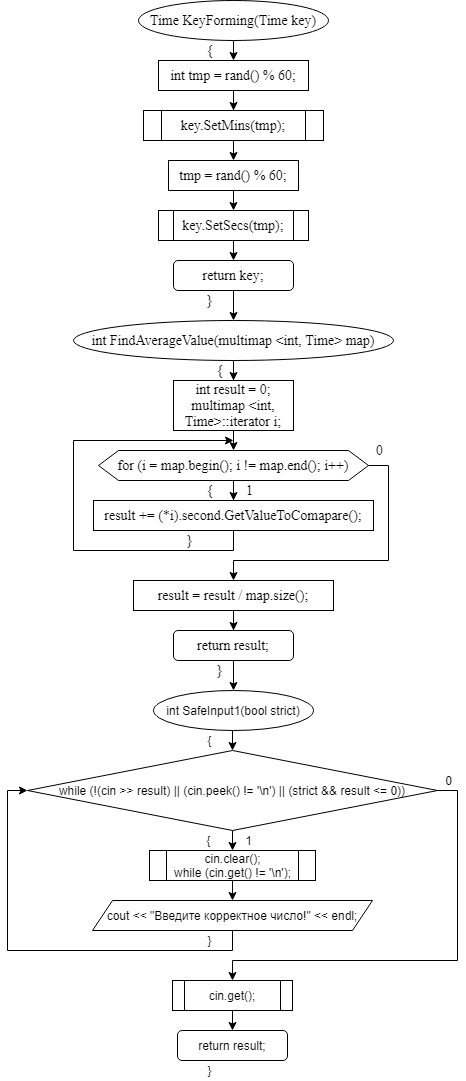
**Блок-схема**

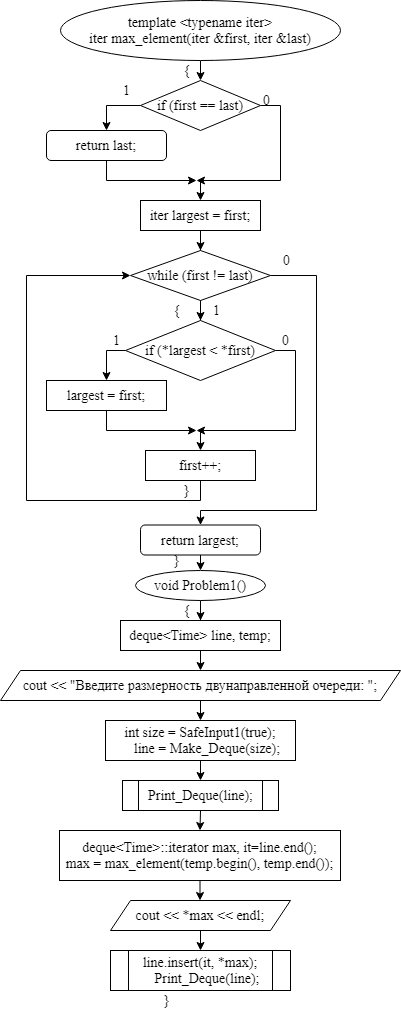


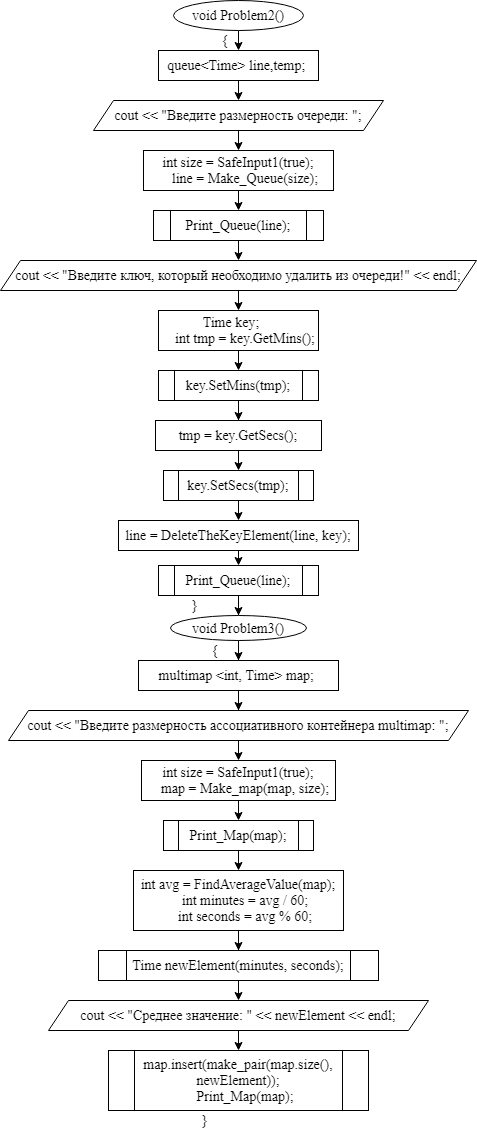












**Код**

#include "Time.h"

#include <ctime>

#include <queue>

#include <deque>

#include <map>

#include <iterator>

#include<algorithm>

#include <functional>

int SafeInput1(bool strict);

//overloading "max\_element" as it doesn't work properly with class Time

template <typename iter>

iter max\_element(iter &first, iter &last)

{

if (first == last) return last;

iter largest = first;

while (first != last) {

if (\*largest < \*first) // or: if (comp(\*largest,\*first)) for version (2)

largest = first;

first++;

}

return largest;

}

deque<Time> Make\_Deque(int size);

void Print\_Deque(deque<Time> line);

void Problem1()

{

deque<Time> line, temp;

cout << "Введите размерность двунаправленной очереди: ";

int size = SafeInput1(true);

line = Make\_Deque(size);

Print\_Deque(line);

temp = line;

deque<Time>::iterator max, it=line.end();

max = max\_element(temp.begin(), temp.end());

cout << \*max << endl;

line.insert(it, \*max);

Print\_Deque(line);

}

queue<Time> Make\_Queue(int size);

void Print\_Queue(queue<Time> line);

Time KeyForming(Time key);

queue<Time> DeleteTheKeyElement(queue<Time> line, Time key);

void Problem2()

{

queue<Time> line,temp;

cout << "Введите размерность очереди: ";

int size = SafeInput1(true);

line = Make\_Queue(size);

Print\_Queue(line);

//forming key to delete it in the next function

cout << "Введите ключ, который необходимо удалить из очереди!" << endl;

Time key;

int tmp = key.GetMins();

key.SetMins(tmp);

tmp = key.GetSecs();

key.SetSecs(tmp);

line = DeleteTheKeyElement(line, key);

Print\_Queue(line);

}

multimap <int, Time> Make\_map(multimap <int, Time> map, int size);

void Print\_Map(multimap <int, Time> map);

int FindAverageValue(multimap <int, Time> map);

void Problem3()

{

multimap <int, Time> map;

cout << "Введите размерность ассоциативного контейнера multimap: ";

int size = SafeInput1(true);

map = Make\_map(map, size);

Print\_Map(map);

int avg = FindAverageValue(map);

int minutes = avg / 60; //converting calculated value into minutes

int seconds = avg % 60; //converting calculated value into seconds

Time newElement(minutes, seconds);

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

map.insert(make\_pair(map.size(), newElement));

Print\_Map(map);

}

int main()

{

setlocale(LC\_ALL, "ru");

srand(time(0));

char vvod = '6';

while (vvod != '0') {

cout << "\nКакое задание?" << endl

<< "1) Задание 1;" << endl

<< "2) Задание 2;" << endl

<< "3) Задание 3;" << endl

<< "0) Выход из меню" << "\n\n" << ">";

cin >> vvod;

switch (vvod) {

case '1': {

Problem1();

}

break;

case '2': {

Problem2();

}

break;

case '3': {

Problem3();

}

break;

default:

if (vvod != '0')

cout << "ERROR!";

break;

}

}

return 0;

}

int SafeInput1(bool strict)

{

int result;

while (!(cin >> result) || (cin.peek() != '\n') || (strict && result <= 0)) {

cin.clear();

while (cin.get() != '\n');

cout << "Введите корректное число!" << endl;

}

cin.get();

return result;

}

deque<Time> Make\_Deque(int size)

{

deque<Time> temp;

Time object;

int tmp;

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

object = KeyForming(object);

temp.push\_back(object);

}

return temp;

}

void Print\_Deque(deque<Time> line)

{

deque<Time> temp = line;

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

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

temp.pop\_front();

}

cout << endl;

}

queue<Time> Make\_Queue(int size)

{

queue<Time> temp;

Time object;

int tmp;

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

object = KeyForming(object);

temp.push(object);

}

return temp;

}

void Print\_Queue(queue<Time> line)

{

queue<Time> temp = line;

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

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

temp.pop();

}

cout << endl;

}

Time KeyForming(Time key)

{

int tmp = rand() % 60;

key.SetMins(tmp);

tmp = rand() % 60;

key.SetSecs(tmp);

return key;

}

queue<Time> DeleteTheKeyElement(queue<Time> line, Time key)

{

queue<Time> temp;

int size = line.size();

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

if (line.front() != key) {

temp.push(line.front());

}

line.pop();

}

return temp;

}

multimap <int, Time> Make\_map(multimap <int, Time> map, int size)

{

Time obj;

int tmp;

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

obj = KeyForming(obj);

map.insert(make\_pair(i, obj));

}

return map;

}

void Print\_Map(multimap <int, Time> map)

{

multimap <int, Time>::iterator i;

for (i = map.begin(); i != map.end(); i++) {

cout << (\*i).second << "; ";

}

cout << endl;

}

int FindAverageValue(multimap <int, Time> map)

{

int result = 0;

multimap <int, Time>::iterator i;

for (i = map.begin(); i != map.end(); i++) {

result += (\*i).second.GetValueToComapare(); //adding each samples converted into seconds

}

result = result / map.size();

return result;

}

#pragma once

#include<iostream>

using namespace std;

class Time {

protected:

int mins, secs;

public:

Time();

Time(int minute, int second);

~Time();

int GetMins();

int GetSecs();

int getM() { return mins; }

int getS() { return secs; }

void SetMins(int value);

void SetSecs(int value);

void Show();

int GetValueToComapare() { int result = secs + (mins \* 60); return result; }

friend bool operator >(Time& t1, Time& t2) { return (t1.GetValueToComapare() > t2.GetValueToComapare()); }

friend bool operator <(Time& t1, Time& t2) { return (t1.GetValueToComapare() < t2.GetValueToComapare()); }

Time operator+(const Time& t);

Time& operator=(const Time& other);

friend bool operator==(const Time& t1, const Time& t2);

friend bool operator!=(const Time& t1, const Time& t2);

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

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

};

#include "Time.h"

int SafeInput(bool strict)

{

int result;

while (!(cin >> result) || (cin.peek() != '\n') || (strict && result <= -1) || (result > 60)) {

cin.clear();

while (cin.get() != '\n');

cout << "Введите корректное число!" << endl;

}

cin.get();

return result;

}

Time::Time() {

mins = 0; secs = 0;

}

Time::Time(int minute, int second)

{

SetMins(minute);

SetSecs(second);

}

Time::~Time() {

}

int Time::GetMins()

{

int minutes;

cout << "Введите минуты >";

minutes = SafeInput(true);

return minutes;

}

int Time::GetSecs()

{

int seconds;

cout << "Введите секунды >";

seconds = SafeInput(true);

return seconds;

}

void Time::SetMins(int value)

{

mins = value;

}

void Time::SetSecs(int value)

{

secs = value;

}

void Time::Show()

{

cout << "\n\n" << mins << ":" << secs;

}

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

{

int temp1 = mins \* 60 + secs;

int temp2 = t.mins \* 60 + t.secs;

Time p;

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

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

return p;

}

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

{

if (this == &other)

return \*this;

mins = other.mins;

secs = other.secs;

return \*this;

}

bool operator==(const Time& t1, const Time& t2)

{

return (t1.mins == t2.mins && t1.secs == t2.secs);

}

bool operator!=(const Time& t1, const Time& t2)

{

return (!(t1.mins == t2.mins && t1.secs == t2.secs));

}

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

{

return (out << t.mins << ":" << t.secs);

}

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

{

cout << "Введите минуты >";

in >> t.mins;

cout << "Введите секунды >";

in >> t.secs;

return in;

}

**Скриншоты**

