**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

# Лабораторная работа № 2

## Тема: Перегрузка операторов в C++

Студент: Короткевич Леонид Витальевич

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Создать класс TimePoint для работы с моментами времени в формате «час:минута:секунда». Обязательными операциями являются: вычисление разницы между двумя моментами времени, сумма моментов времени, сложение момента времени и заданного количества секунд, вычитание из момента времени заданного количества секунд, вычисление во раз сколько один момент времени больше (меньше) другого, сравнение моментов времени, перевод в секунды и обратно, перевод в минуты (с округлением до минуты) и обратно.

Операции сложения и вычитания TimePoint, а так же сравнения (больше, меньше и равно) необходимо реализовать в виде перегрузки операторов.

Необходимо реализовать пользовательский литерал для работы с константами типа TimePoint.

2. Описание программы

| Vector.hpp | |
|---|---|
| Vector(); | конструктор по умолчанию |
| Vector(const int &n); | конструктор с 1 параметром: размер вектора |
| Vector(const int &n, T elem); | конструктор с 2 параметрами: размер вектора, заполнить элементами T |
| ~Vector(); | деструктор по умолчанию |
| void assert(const int &n, T elem); | метод: заполнить вектор n одинаковыми элементами |
| T &operator[](const int &index) const; | конст. метод: обратиться к элементу вектора |
| void push_back(T elem); | метод: добавить в конец вектора элемент |
| unsigned int size() const; | конст. метод: получить размер вектора |
| void erase(const int &idx); | метод: удалить элемент из вектора |
| TimePoint.hpp | |
| TimePoint(); | конструктор по умолчанию |
| TimePoint(const std::string &_time); | конструктор с 1 параметром: чч:мм:сс |
| TimePoint(const int &_val); | конструктор с 1 параметром: секунды |
| TimePoint operator-(const TimePoint &rhs) const; | "вычитание" двух временных точек - разность в чч:мм:сс м/у ними |
| TimePoint operator+(const TimePoint &rhs) const; | "сложение" двух временных точек (далее - ВТ) |

| | |
|---|---|
| double operator/(const TimePoint &rhs) const; | "деление" двух временных точек: во ск-ко раз больше/меньше |
| void add(const int &_ss); | метод: добавить к ВТ секунды |
| void subtract(const int &_ss); | метод: отнять от ВТ секунды |
| bool operator>(TimePoint const &rhs) const; | сравнение временных точек |
| bool operator==(TimePoint const &rhs) const; | |
| bool operator<(TimePoint const &rhs) const; | |
| int toMins() const; | конст. метод: циферблат в минуты |
| int toSecs() const; | конст. метод: циферблат в секунды |
| std::string getTime() const; | конст. метод: получение строки-циферблата |
| void display() const; | конст. метод: вывести строку-циферблат |

## 3. Набор тестов

| test_01.txt | test_02.txt | test_03.txt |
|---|---|---|
| 1 | 1 | 1 |
| 4:30:29 | 12:12:12 | 12:00:00 |
| 1 | 1 | 1 |
| 4:29:01 | 2:03:04 | 12:00:01 |
| 11 | 11 | 7 |
| 3 | 5 | 1 |
| 1 2 | 1 2 | 7 |
| 1 | 5 | 2 |
| 4:30:59 | 2 1 | 8 |
| 3 | 1 | 43201 |
| 2 3 | 2:03:04 | 9 |
| 4 | 11 | 1 |
| 3 1 | 6 | 10 |
| 4 | 2 3 | 720 |
| 2 3 | 6 | 11 |
| 12 | 1 3 | 12 |
| | 6 | |
| | 1 2 | |
| | 12 | |

## 4. Результаты выполнения тестов

| test_01.txt | test_02.txt | test_03.txt |
|---|---|---|
| 1 - добавить<br>2 - удалить<br>3 - сложить<br>4 - вычесть<br>5 - поделить<br>6 - сравнить<br>7 - вывести в секундах<br>8 - из секунд в час:мин:сек и вывести<br>9 - вывести в минутах<br>10 - из минут в час:мин:сек и вывести<br>11 - вывести список<br>12 - выйти<br><br>1: 04:30:29<br>2: 04:29:01<br>04:30:29 + 04:29:01 = 08:59:30<br>04:29:01 + 04:30:59 = 09:00:00<br>04:30:59 - 04:30:29 = 00:00:30<br>04:29:01 - 04:30:59 = 00:01:58 | 1 - добавить<br>2 - удалить<br>3 - сложить<br>4 - вычесть<br>5 - поделить<br>6 - сравнить<br>7 - вывести в секундах<br>8 - из секунд в час:мин:сек и вывести<br>9 - вывести в минутах<br>10 - из минут в час:мин:сек и вывести<br>11 - вывести список<br>12 - выйти<br><br>1: 12:12:12<br>2: 02:03:04<br>12:12:12 / 02:03:04 = 5.94962<br>02:03:04 / 12:12:12 = 0.168078<br>1: 12:12:12<br>2: 02:03:04<br>3: 02:03:04<br>равны<br>первый больше<br>первый больше | 1 - добавить<br>2 - удалить<br>3 - сложить<br>4 - вычесть<br>5 - поделить<br>6 - сравнить<br>7 - вывести в секундах<br>8 - из секунд в час:мин:сек и вывести<br>9 - вывести в минута x<br>10 - из минут в час:мин:сек и вывести<br>11 - вывести список<br>12 - выйти<br><br>43200<br>43201<br>12:00:01<br>720<br>12:00:00<br>1: 12:00:00<br>2: 12:00:01 |

## 5. Листинг программы

### main.cpp

```
#include <iostream>
#include <string>

#include "TimePoint.hpp"
#include "Vector.hpp"

/*
Короткевич Л. В.
github.com/anxieuse/oop_exercise_02
Создать класс TimePoint для работы с моментами времени в формате «час:минута:секунда».
Обязательными операциями являются:
    вычисление разницы между двумя моментами времени,
    сумма моментов времени,
    сложение момента времени и заданного количества секунд,
    вычитание из момента времени заданного количества секунд,
    вычисление во раз сколько один момент времени больше (меньше) другого,
    сравнение моментов времени,
    перевод в секунды и обратно,
    перевод в минуты (с округлением до минуты) и обратно.
Операции сложения и вычитания TimePoint, а так же сравнения (больше, меньше и равно) необходимо реализовать в виде перегрузки операторов.
Необходимо реализовать пользовательский литерал для работы с константами типа TimePoint.
*/
```

```
const TimePoint mrn("06:00:00"), aft("12:00:00"), eve("18:00:00"), nght("00:00:00"), null("00:0:-1");

std::string operator "" _t(const char *s, size_t size)
{
        return
                s == "morning" ? mrn.getTime() :
      s == "afternoon" ? aft.getTime() :
      s == "evening" ? eve.getTime() :
      s == "night" ? nght.getTime() : null.getTime();
}

int main() {
   int idToDel, id, id1, id2, secs, mins;
   std::string time;
   Vector<TimePoint> tps;

   int command_idx;
   bool input = true;
   std::cout << '\n'
         << " 1  - добавить" << '\n'
         << " 2  - удалить" << '\n'
         << " 3  - сложить" << '\n'
         << " 4  - вычесть" << '\n'
         << " 5  - поделить" << '\n'
         << " 6  - сравнить" << '\n'
         << " 7  - вывести в секундах" << '\n'
         << " 8  - из секунд в час:мин:сек и вывести\n"
         << " 9  - вывести в минутах\n"
         << " 10  - из минут в час:мин:сек и вывести\n"
         << " 11 - вывести список\n"
         << " 12 - выйти" << '\n'
         << '\n';
   while (input)
   {
      std::cin >> command_idx;
      switch (command_idx)
      {
         case 1:
         {
            std::cin >> time;
            tps.push_back({time});
            break;
         }
         case 2:
         {
            std::cin >> idToDel;
            --idToDel;
            if (idToDel < tps.size() && idToDel >= 0)
            {
               std::cout << "Временной промежуток ";
               tps[idToDel].display();
               std::cout << " успешно удален из списка\n";
```

```
                tps.erase(idToDel);
            }
            else
            {
                std::cout << "Введен некорректный индекс\n";
            }
            break;
        }
        case 3:
        {
            std::cin >> id1 >> id2;
            --id1, --id2;
                std::cout << tps[id1].getTime() << " + " << tps[id2].getTime() << " = " << (tps[id1] +
tps[id2]).getTime();
            std::cout << '\n';
            break;
        }
        case 4:
        {
            std::cin >> id1 >> id2;
            --id1, --id2;
                 std::cout << tps[id1].getTime() << " - " << tps[id2].getTime() << " = " << (tps[id1] -
tps[id2]).getTime();
            std::cout << '\n';
            break;
        }
        case 5:
        {
            std::cin >> id1 >> id2;
            --id1, --id2;
            std::cout << tps[id1].getTime() << " / " << tps[id2].getTime() << " = " << (tps[id1]/tps[id2]);
            std::cout << '\n';
            break;
        }
        case 6:
        {
            std::cin >> id1 >> id2;
            if (tps[id1 - 1] == tps[id2 - 1])
            {
                std::cout << "равны\n";
            }
            else
            {
                if (tps[id1 - 1] > tps[id2 - 1])
                    std::cout << "первый больше\n";
                else
                    std::cout << "второй больше\n";
            }
            break;
        }
        case 7:
        {
            std::cin >> id;
```

```
            std::cout << tps[id - 1].toSecs();
            std::cout << '\n';
            break;
        }
        case 8:
        {
            std::cin >> secs;
            std::cout << fromSecs(secs);
            std::cout << '\n';
            break;
        }
        case 9:
        {
            std::cin >> id;
            std::cout << tps[id - 1].toMins();
            std::cout << '\n';
            break;
        }
        case 10:
        {
            std::cin >> mins;
            std::cout << fromMins(mins);
            std::cout << '\n';
            break;
        }
        case 11:
        {
            for(int i = 0; i < tps.size(); ++i) {
                std::cout << i + 1 << ": ";
                tps[i].display();
                std::cout << "\n";
            }
            break;
        }
        case 12:
        {
            input = false;
            break;
        }
      }
   }
}
```

## TimePoint.hpp

```
#ifndef TIMEPOINT_HPP
#define TIMEPOINT_HPP

class TimePoint
{
public:
    TimePoint();
    TimePoint(const std::string &_time);
```

```
    TimePoint(const int &_val);
    TimePoint operator-(const TimePoint &rhs) const;
    TimePoint operator+(const TimePoint &rhs) const;
    double operator/(const TimePoint &rhs) const;
    void add(const int &_ss);
    void subtract(const int &_ss);
    bool operator>(TimePoint const &rhs) const;
    bool operator==(TimePoint const &rhs) const;
    bool operator<(TimePoint const &rhs) const;
    int toMins() const;
    int toSecs() const;
    std::string getTime() const;
    void display() const;

private:
    std::string time;
    int hh, mm, ss, val;
};

TimePoint::TimePoint() : hh(0), mm(0), ss(-1), val(-1) {}

TimePoint::TimePoint(const std::string &_time)
{
    sscanf(_time.c_str(), "%d:%d:%d", &hh, &mm, &ss);
    val = hh * 60 * 60 + mm * 60 + ss;
    char tmp[9];
    snprintf(tmp, 10, "%02d:%02d:%02d", hh, mm, ss);
    time = tmp;
}

void renew(std::string &time, int &hh, int &mm, int &ss, int &val, const int &_val) {
    val = _val;
    hh = val / 60 / 60, val -= hh * 60 * 60;
    mm = val / 60, val -= mm * 60;
    ss = val;
    val = _val;
    char tmp[9];
    snprintf(tmp, 10, "%02d:%02d:%02d", hh, mm, ss);
    time = tmp;
}

TimePoint::TimePoint(const int &_val) {
    renew(time, hh, mm, ss, val, _val);
}

TimePoint TimePoint::operator-(const TimePoint &rhs) const {
    return TimePoint(abs(val - rhs.val));
}

TimePoint TimePoint::operator+(const TimePoint &rhs) const {
    return TimePoint(val + rhs.val);
}
```

```
double TimePoint::operator/(const TimePoint &rhs) const {
    return 1.0 * val / rhs.val;
}

void TimePoint::add(const int &_ss) {
    renew(time, hh, mm, ss, val, val + _ss);
}

void TimePoint::subtract(const int &_ss) {
    renew(time, hh, mm, ss, val, val - _ss);
}

bool TimePoint::operator>(TimePoint const &rhs) const {
    return val > rhs.val;
}
bool TimePoint::operator==(TimePoint const &rhs) const {
    return val == rhs.val;
}
bool TimePoint::operator<(TimePoint const &rhs) const {
    return val < rhs.val;
}

int TimePoint::toSecs() const {
    return val;
}
std::string fromSecs(const int &ss) {
    return TimePoint(ss).getTime();
}

int TimePoint::toMins() const {
    return val / 60;
}
std::string fromMins(const int &mm) {
    return TimePoint(mm * 60).getTime();
}

void TimePoint::display() const {
    std::cout << time;
}

std::string TimePoint::getTime() const {
    return time;
}

#endif
```

## Vector.hpp:

```
#ifndef VECTOR_HPP
#define VECTOR_HPP

template <class T>
class Vector
```

```
{
public:
	Vector();
	Vector(const int &n);
	Vector(const int &n, T elem);
	~Vector();

	void assert(const int &n, T elem);
	T &operator[](const int &index) const;
	void push_back(T elem);
	unsigned int size() const;
	void erase(const int &idx);

private:
	unsigned int capacity;
	unsigned int maxsize;
	T *data;
};

template <class T>
void Vector<T>::assert(const int &n, T elem)
{
	for (int i = 0; i < n; i++)
	{
		data[i] = elem;
	}
}

template <class T>
T &Vector<T>::operator[](const int &index) const
{
	return data[index];
}

template <class T>
void Vector<T>::push_back(T elem)
{
	if (data == 0)
	{
		maxsize = 1;
		data = new T[maxsize];
	}
	if (capacity == maxsize)
	{
		maxsize = maxsize * 2;
		T *new_data = new T[maxsize];
		for (int i = 0; i < capacity; i++)
		{
			new_data[i] = data[i];
		}
		delete[] data;
		data = new_data;
	}
```

```
	data[capacity] = elem;
	capacity++;
};

template <class T>
unsigned int Vector<T>::size() const
{
	return capacity;
}


template <class T>
void Vector<T>::erase(const int &idx)
{
   for (size_t i = idx; i < capacity - 1; ++i)
			data[i] = data[i + 1];

	capacity--;
}

template <class T>
Vector<T>::Vector()
{
	capacity = 0;
	maxsize = 0;
	data = 0;
}

template <class T>
Vector<T>::Vector(const int &n)
{
	capacity = n;
	maxsize = n;
	data = new T[capacity];
	assert(n, T());
}

template <class T>
Vector<T>::Vector(const int &n, T elem)
{
	capacity = n;
	maxsize = n;
	data = new T[capacity];
	assert(n, elem);
}

template <class T>
Vector<T>::~Vector()
{
	delete[] data;
}
```

#endif