МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №2

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студент: *Морозов Артем Борисович, группа М8О-208Б-20*

Преподаватель: *Дорохов Евгений Павлович, каф. 806*

Задание:

Разработать программу на языке C++ согласно варианту задания. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод. Реализовать пользовательский литерал для работы с константами объектов созданного класса.

Вариант №14:

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.

Описание программы:

Исходный код разделён на 3 файла:

- `TimePoint.h` – описание основных функций класса TimePoint
- `TimePoint.cpp` – реализация функционала класса TimePoint
- `main.cpp` – основная программа

Дневник отладки:

Во время выполнения данной лабораторной работы небольшие проблемы возникли с перегрузкой операторов, однако были почти сразу же устранены.

Вывод:

Данная лабораторная работа научила меня двум очень важным вещам: **1) перегрузке операторов.** Без перегрузки операторов не обходится ни один большой проект, это очень важное понятие в сфере объектно-ориентированного-программирования, ведь классы бывают совершенно разные, с разными полями. Например, в моем задании нужно складывать два объекта, хранящих в себе 3 поля: часы, минуты и секунды. Перегрузка операторов нам в этом деле очень сильно помогает. **2) пользовательским литералам.** Оказывается, это очень удобная и практичная вещь, о которой я никогда не знал. Прелесть данного средства в том, что мы вычисляем какие-то значения без использования вспомогательных функций, а попросту переопределением специального оператора.

Исходный код:

`TimePoint.h`:

```
#ifndef TIMEPOINT_H
#define TIMEPOINT_H
#include <iostream>
```

```
        class TimePoint {

            public:
                TimePoint();
                TimePoint(int h, int m, int s);
                TimePoint(const TimePoint &other);
                void AddSeconds(int s);
                void RemoveSeconds(int s);
                int IsBigger(const TimePoint &other);
                int ToSeconds();
                int ToMinutes();
                TimePoint operator + (const TimePoint &object);
                TimePoint operator - (const TimePoint &object);
                bool operator == (const TimePoint &object);
                bool operator > (const TimePoint &other);
                bool operator < (const TimePoint &other);
                friend std::istream& operator >> (std::istream& is, TimePoint &object);
                friend std::ostream& operator <<(std::ostream& os, TimePoint &object);
                ~TimePoint();
            private:
                int hours;
                int minutes;
                int seconds;
        };
#endif
```

TimePoint.cpp:

```
#include "TimePoint.h"
TimePoint::TimePoint() {
   hours = 0;
   minutes = 0;
   seconds = 0;
   std:: cout << "The default time-ojbect has been created" << std:: endl;
}

TimePoint::TimePoint(int h, int m, int s) {
   if (h >= 0 && m >= 0 && s >= 0) {
      hours = h;
      minutes = m;
      seconds = s;
   }
   else {
      std:: cout << "Please enter positive numbers!" << std:: endl;
   }
   std:: cout << "The time-object according to your parameters has been created" << std:: endl;
}

TimePoint::TimePoint(const TimePoint& other) {
   hours = other.hours;
   minutes = other.minutes;
   seconds = other.seconds;
   std:: cout << "The copy of your time-object has been created" << std:: endl;
}

void TimePoint::AddSeconds(int s) {
   if (s < 0) {
      std:: cout << "Please enter positive number!" << std:: endl;
   }
   else {
      int x = hours * 3600 + minutes * 60 + seconds + s;
      hours = x / 3600;
      minutes = ((x % 3600) / 60);
      seconds = (x % 3600) - (((x % 3600) / 60) * 60);
      std:: cout << "After adding seconds your time is: "  << hours << ":" << minutes << ":" << seconds << std:: endl;
```

```
    }
}

void TimePoint::RemoveSeconds(int s) {
    if (s < 0) {
        std:: cout << "Please enter positive number!" << std:: endl;
    }
    else {
        int x = hours * 3600 + minutes * 60 + seconds - s;
        hours = x / 3600;
        minutes = ((x % 3600) / 60);
        seconds = (x % 3600) - (((x % 3600) / 60) * 60);
        std:: cout << "After removing seconds your time is: "  << hours << ":" << minutes << ":" << seconds << std:: endl;
    }
}

int TimePoint::IsBigger(const TimePoint &other ) {
    int x = hours * 3600 + minutes * 60 + seconds;
    int y = other.hours * 3600 + other.minutes * 60 + other.seconds;
    if ((hours > other.hours) || (hours == other.hours && minutes > other.minutes) || (hours == other.hours && minutes ==
other.minutes && seconds > other.seconds)) {
        return x / y;
    }
    return y / x;
}

int TimePoint::ToSeconds() {
    return hours * 3600 + minutes * 60 + seconds;
}

int TimePoint::ToMinutes() {
    int z = hours * 3600 + minutes * 60 + seconds;
    int m = z / 60;
    if (z % 60 == 0) {
        return m;
    }
    else {
        if (z % 60 >= 30) {
            return m + 1;
        }
    }
    return m;
}

TimePoint TimePoint::operator + (const TimePoint &object) {
    int x = hours * 3600 + minutes * 60 + seconds;
    int y = object.hours * 3600 + object.minutes * 60 + object.seconds;
    int z = x + y;
    int dhours = z / 3600;
    int dminutes = (z % 3600) / 60;
    int dseconds = (z % 3600) - (dminutes * 60);
    this->hours = dhours;
    this->minutes = dminutes;
    this->seconds = dseconds;
    return *this;
}

TimePoint TimePoint::operator - (const TimePoint &object) {
    int x = hours * 3600 + minutes * 60 + seconds;
    int y = object.hours * 3600 + object.minutes * 60 + object.seconds;
    int dhours, dminutes, dseconds;
    if ((hours > object.hours) || (hours == object.hours && minutes > object.minutes) || (hours == object.hours && minutes ==
object.minutes && seconds > object.seconds)) {
        int z = x - y;
        dhours = z / 3600;
        dminutes = (z % 3600) / 60;
```

```
        dseconds = (z % 3600) - (dminutes * 60);
    }
    else {
        int z = y - x;
        dhours = z / 3600;
        dminutes = (z % 3600) / 60;
        dseconds = (z % 3600) - (dminutes * 60);
    }
    this->hours = dhours;
    this->minutes = dminutes;
    this->seconds = dseconds;
    return *this;
}

bool TimePoint::operator == (const TimePoint &object) {
    if (hours == object.hours && minutes == object.minutes && seconds == object.seconds) {
        return true;
    }
    return false;
}

bool TimePoint::operator > (const TimePoint &other) {
    if ((hours > other.hours) || (hours == other.hours && minutes > other.minutes) || (hours == other.hours && minutes ==
other.minutes && seconds > other.seconds)) {
        return true;
    }
    else {
        return false;
    }
}

bool TimePoint::operator < (const TimePoint &other) {
    if ((hours < other.hours) || (hours == other.hours && minutes < other.minutes) || (hours == other.hours && minutes ==
other.minutes && seconds < other.seconds)) {
        return true;
    }
    else {
        return false;
    }
}

std::istream& operator >> (std::istream& is, TimePoint &object) {
    std:: cout << "Please enter your time-object data: " << std:: endl;
    is >> object.hours >> object.minutes >> object.seconds;
    if ((object.hours < 0 || object.hours > 23) || (object.minutes < 0 || object.minutes > 59) || (object.seconds < 0 || object.seconds >
59)) {
        std:: cout << "Invalind input. Enter again!" << std:: endl;
        is >> object.hours >> object.minutes >> object.seconds;
    }
    return is;
}

std::ostream& operator << (std::ostream& os, TimePoint &object) {
    os << object.hours << ":" << object.minutes << ":" << object.seconds << std:: endl;
    return os;
}

TimePoint::~TimePoint() {
    std:: cout << "FROM DESTRUCTOR: Your time-object has been deleted" << std:: endl;
}
```

main.cpp:

```
#include "TimePoint.h"
```

```
unsigned long long operator "" _tohours(unsigned long long sec) {
    unsigned long long hours_lit = sec / 3600;
    return hours_lit;
}

unsigned long long operator "" _tominutes(unsigned long long sec) {
    unsigned long long minutes_lit = sec / 60;
    return minutes_lit;
}

int main () {
    TimePoint a(15,0,5);
    TimePoint b(15,0,5);
    std:: cout << (b == a) << std:: endl;
    std:: cout << (a > b) << std:: endl;
    std:: cout << (b < a) << std:: endl;
    TimePoint c = a + b;
    std:: cout << c;
    TimePoint d;
    std:: cin >> d;
    TimePoint e = d - b;
    std:: cout << e;
    TimePoint x(23,59,59);
    TimePoint y = x - a;
    TimePoint j, h, u;
    std:: cin >> j >> h >> u;
    std:: cout << j << h << u;
    std:: cout << "The example of using to-hours literal is: " << 3600_tohours << std:: endl;
    std:: cout << "The example of using to-minutes literal is: " << 50505_tominutes << std:: endl;
    return 0;
}
```

Пример работы:

```
#include "TimePoint.h"

//literals
unsigned long long operator "" _tohours(unsigned long long sec) {
    unsigned long long hours_lit = sec / 3600;
    return hours_lit;
}

unsigned long long operator "" _tominutes(unsigned long long sec) {
    unsigned long long minutes_lit = sec / 60;
    return minutes_lit;
}

int main () {
    TimePoint a(15,0,5);
    TimePoint b(15,0,5);
    std:: cout << (b == a) << std:: endl;
    std:: cout << (a > b) << std:: endl;
    std:: cout << (b < a) << std:: endl;
    TimePoint c = a + b;
    std:: cout << c;
    TimePoint d;
    std:: cin >> d;
    TimePoint e = d - b;
    std:: cout << e;
    TimePoint x(23,59,59);
    TimePoint y = x - a;
    TimePoint j, h, u;
    std:: cin >> j >> h >> u;
    std:: cout << j << h << u;
    std:: cout << "The example of using to-hours literal is: " << 3600_tohours << std:: endl;
    std:: cout << "The example of using to-minutes literal is: " << 50505_tominutes << std:: endl;
    return 0;
}
```

```
The time-object according to your parameters has been created
The time-object according to your parameters has been created
1
0
0
The copy of your time-object has been created
30:0:10
The default time-ojbect has been created
Please enter your time-object data:
10 20 45
The copy of your time-object has been created
4:39:20
The time-object according to your parameters has been created
The copy of your time-object has been created
The default time-ojbect has been created
The default time-ojbect has been created
The default time-ojbect has been created
Please enter your time-object data:
10 11 10
Please enter your time-object data:
09 30 45
Please enter your time-object data:
03 30 23
10:11:10
9:30:45
3:30:23
The example of using to-hours literal is: 1
The example of using to-minutes literal is: 841
FROM DESTRUCTOR: Your time-object has been deleted
FROM DESTRUCTOR: Your time-object has been deleted
FROM DESTRUCTOR: Your time-object has been deleted
FROM DESTRUCTOR: Your time-object has been deleted
FROM DESTRUCTOR: Your time-object has been deleted
FROM DESTRUCTOR: Your time-object has been deleted
FROM DESTRUCTOR: Your time-object has been deleted
FROM DESTRUCTOR: Your time-object has been deleted
FROM DESTRUCTOR: Your time-object has been deleted
FROM DESTRUCTOR: Your time-object has been deleted
```