Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: Перегрузка операторов

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Токарев Павел Аркадьевич

Проверил

Доцент кафедры ИТАС

Полякова Ольга Андреевна

Г. Пермь-2023

**Постановка задачи**

1.Определить новый класс.

2.Определить в классе конструкторы: без параметров, с параметрами, копирования и деструктор.

3.Определить в классе селекторы и модификаторы

4.Перегрузить операцию присваивания, операции ввода и вывода, операцию вычитания временных интервалов и операцию их сравнения(!=).

5.В главной функции продемонстрировать методы класса.

**Анализ**

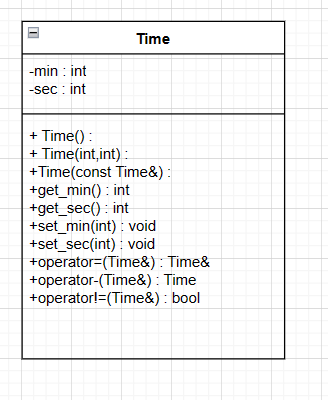
1.Определим класс Time, работающий с временными интервалами. В нём определим 3 вида конструкторов, деструктор, селекторы и модификаторы.

2.Перегрузим операторы ввода и вывод, как внешние дружественные функции.

3.Перегрузим оператор присваивания. Теперь она возвращает ссылку на объект, которому присваивают.

4.В функции main покажем создание объектов и все перегруженные операции.

**Диаграмма UML**

****

**Код программы**

main.cpp

#include"Time.h"

#include <iostream>

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

Time a, b;

cout << "Временной интервал a" << endl;

cin >> a;

cout << "Временной интервал b" << endl;

cin >> b;

if (a != b) {

cout << "Временные интервалы не равны"<<endl;

}

else {

cout << "Временные интервалы равны"<<endl;

}

cout << "Вычтем из первого интервала второй" << endl;

Time c = a - b;

cout << c;

}

time.h

#pragma once

#include <iostream>

using namespace std;

class Time {

int min, sec;

public:

Time() {

min = 0;

sec = 0;

}

Time(int tmp1, int tmp2) {

min = tmp1;

sec = tmp2;

}

Time(const Time& t) {

min = t.min;

sec = t.sec;

}

~Time() {};

int getminute() {

return min;

}

int getsecond() {

return sec;

}

void setminute(int tmp) {

min = tmp;

}

void setsecond(int tmp) {

sec = tmp;

}

Time& operator=(const Time&);

Time operator-(const Time&);

bool operator!=(const Time&);

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

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

};

time.cpp

#include "Time.h"

#include <iostream>

using namespace std;

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

if (&t == this) {

return \*this;

}

else {

min = t.min;

sec = t.sec;

return \*this;

}

}

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

int tmp1 = (min \* 60) + sec;

int tmp2 = (t.min \* 60) + t.sec;

Time newtime;

newtime.min = (tmp1 - tmp2) / 60;

newtime.sec = (tmp1 - tmp2) % 60;

return newtime;

}

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

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

return true;

}

else {

return false;

}

};

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

cout << "минута:";

in >> t.min;

cout << "секунда:";

in >> t.sec;

return in;

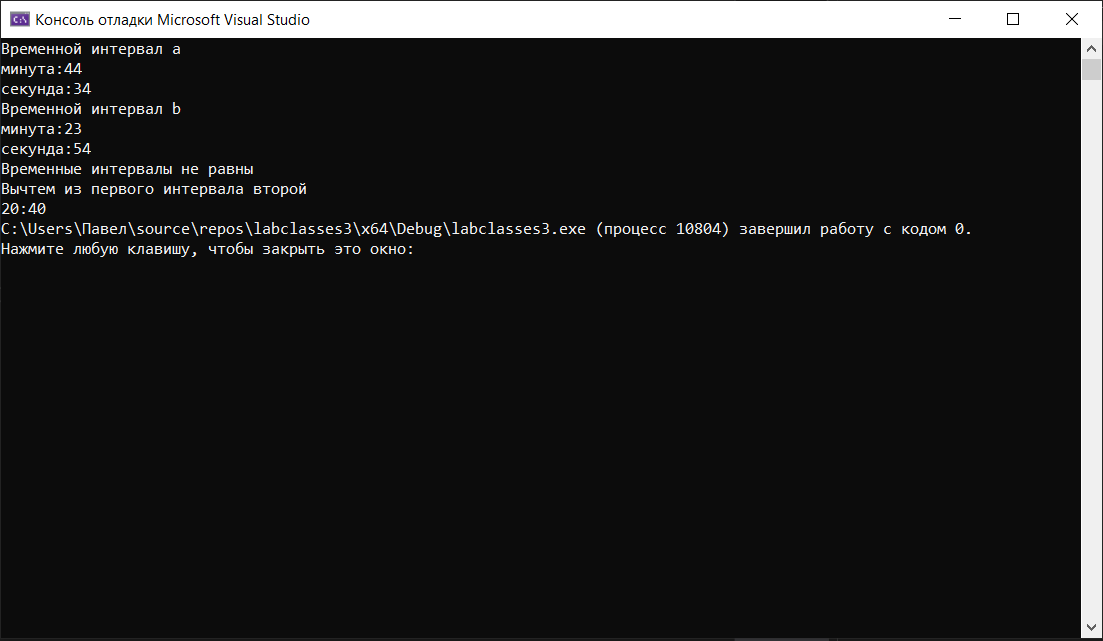
}

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

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

}

**Результаты работы программы**

****

**Анализ результатов**

Как видно из результата работы программы, создание экземпляра класса выполняется корректно. Корректно работают перегруженные операторы вывода и перегруженные бинарные операторы.

**Контрольные вопросы**

1.Дружественные функции применяются, когда необходимо получить доступ к скрытым полям класса извне.

2.Дружестыенные функции определяются внутри класса, к элементам которого необходимо получить доступ. В качестве аргумента ей должна передаваться на ссылку объекта класса. Дружественная функция может быть обычной функцией или методо ранее определённого класса. На неё не распространяются спецификаторы доступа. Одна функция может быть дружественной сразу к нескольким классам.

3.Унарную операцию можно перегрузить как:

-Компонентную функцию класса

-Как внешнюю функцию

4.Унарная функция-операция, определённая внутри класса должна быть представлена как нестатическая функция без параметров.

5.Унарная функция-операция, определённая вне класса должна иметь один операнд (тип класса).

6.Бинарная функция-операция, определённая внутри класса имеет 1 операнд (спарава).

7. Бинарная функция-операция, определённая вне класса имеет 2 операнда (слева и справа от оператора ).

8.Префиксная форма возвращает ссылку на объект класса, а постфиксная форма не возвращает ничего.

9. Операцию присваивания можно перегрузить так:

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

if (&t == this) {

return \*this;

}

else {

min = t.min;

sec = t.sec;

return \*this;

}

}

10.Перегруженная операция присваивания должна возвращать ссылку на левый операнд.

11.Операции ввода и вывода можно перегрузить так:

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

cout << "минута:";

in >> t.min;

cout << "секунда:";

in >> t.sec;

return in;

}

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

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

}

12.Компилятор будет воспринимать вызов функции-оператора как компонентную функцию.

13.Компилятор будет воспринимать вызов функции-оператора как внешнюю дружественную функцию.

14.Компилятор будет воспринимать функцию-оператор как компонентную функцию.

15.Компилятор будет воспринимать функцию-оператор как внешнюю дружественную функцию.