Министерство образования и науки РФ

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

Лабораторная работа № 3

Тема: «Перегрузка операций»

Выполнил: студент группы РИС-22-2б

Мизёв В.А. ф

Проверил: доцент кафедры ИТАС

Полякова О.А.

г. Пермь – 2024

*Постановка задачи:*

1. Определить пользовательский класс.
2. Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.
3. Определить в классе деструктор.
4. Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы)
5. Перегрузить операцию присваивания.
6. Перегрузить операции ввода и вывода объектов с помощью потоков.
7. Перегрузить операции, указанные в варианте.
8. Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.

*Задание вариант 1:*

Создать класс Time для работы с временными интервалами. Интервал должен быть представлен в виде двух полей: минуты типа int и секунды типа int. при выводе минуты отделяются от секунд двоеточием. Реализовать:

* cложение временных интервалов (учесть, что в минуте не может быть более 60 секунд)
* сравнение временных интервалов (==)

*Анализ задачи:*

Основываясь на задании необходимо реализовать один класс “Time”, с полями m\_minutes, m\_seconds, а также методы для получения и установки значений для этих полей. Три конструктора: конструктор по умолчанию Time(), копирования Time(const Time& time) и с параметрами Time(int minutes, int seconds). Реализовать перегрузку операторов для сравнения и сложения.

*Код программы:*

Прикреплён в приложении 1.

*UML-диаграмма класса:*

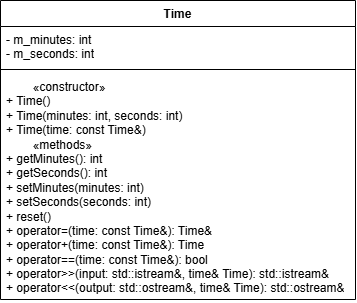


Рисунок 1 - UML-диаграмма класса Time

*Скриншот работы программы:*

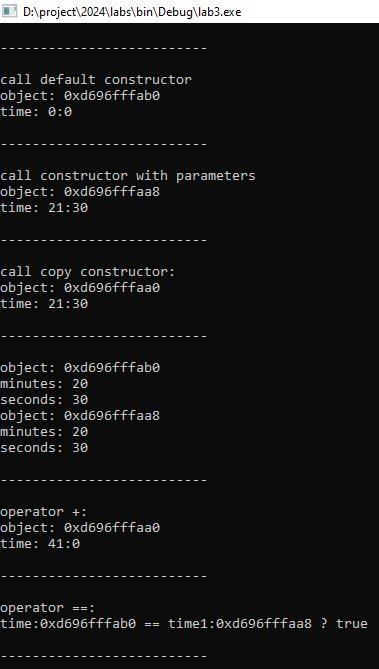


Рисунок 2 – Результат работы программы

*Контрольные вопросы:*

1. Для чего используются дружественные функции и классы?

Дружественные функции применяются для доступа к скрытым полям класса и представляют собой альтернативу методам. Класс, методы которого должны иметь доступ к скрытым полях другого, то весь класс объявляется дружественным.

Пример дружественной функции

class Map;

class Player {

void init(Map& map);

};

class Map {

friend void Player::init(Map& map);

}

Пример объявления дружественного класса

class Map;

class Player {

void init();

}

class Map {

friend class Player;

}

1. Сформулировать правила описания и особенности дружественных функций.

Правила описания:

Дружественные функции объявляются внутри класса, к элементам которого ей нужен доступ, с ключевым словом friend.

Дружественная функция может быть обычной функцией или методом другого ранее определённого класса.

Одна функция может быть дружественной сразу нескольким классами.

Особенности дружественных функций:

Доступ к закрытым членам класса.

Определение вне класса. (внутри дружественных функций можно через переданную ссылку обратиться ко всем его закрытым переменным)

Вызываются дружественные функции, как обычные функции.

1. Каким образом можно перегрузить унарные операции?

Унарную операцию можно перегрузить как компонентную функцию класса и как внешнюю (глобальную) функцию. Унарная функция-операция, определяемая внутри класса, должна быть представлена с помощью нестатического метода без параметров, при это операндом является вызвавший её объект.

Внутри класса

class Person {

string name;

int age;

public

Person(string, int);

….

Person& operator ++() {

++age;

return \*this;

}

}

Вне класса

class Person {

string name;

int age;

public:

Person(string, int);

friend Person& operator ++(Person&);

};

Person& operator ++(Person& p){

++p.age;

return p;

}

1. Сколько операндов должна иметь унарная функция-операция, определяемая внутри класса?

Унарная функция-операция, определяемая внутри класса, не имеет входных параметров. Операндом такой операции является вызвавший её объект.

1. Сколько операндов должна иметь унарная функция-операция, определяемая вне класса?

Функция, определённая вне класса, должна иметь один параметр типа класса.

1. Сколько операндов должна иметь бинарная функция-операция, определяемая внутри класса?

Один операнд.

class Person {

string name;

int age;

public

Person(string, int);

….

Person& operator +(int x) {

age+=x;

return \*this;

}

};

1. Сколько операндов должна иметь бинарная функция-операция, определяемая вне класса?

Два операнда.

class Person {

string name;

int age;

public:

Person(string, int);

friend Person& operator+(Person& p, int x);

}

Person& operator+(Person& p, int x) {

p.age+=x;

return p;

}

1. Чем отличается перегрузка префиксных и постфиксных унарных операций?

Постфиксная перегрузка должна иметь первым параметром тип int, а также возвращать копию старого объекта.

Person& operator++()

Person operator++(int)

1. Каким образом можно перегрузить операцию присваивания?

Чтобы перегрузить операцию присваивания, необходимо определить её в виде функции класса.

Также для сохранения семантики присваивания, функция должна: возвращать ссылку на объект, для которого она вызвана, принимать в качестве параметра ссылку на присваиваемый объект.

class Complex {

Complex& operator=(const Complex& cm);

}

1. Что должна возвращать операция присваивания?

Ссылку на объект, для которого она вызвана.

1. Каким образом можно перегрузить операции ввода-вывода?

Операции ввода-вывода всегда реализуются как внешние дружественные функции т.к. левым операндом этих операций являются потоки.

Внутри класса:

friend istream& operator>>(istream& in, Person& p);

friend ostream& operator<<(ostream& out, const Person& p);

1. В программе описан класс

class Student {

Student& operator++();

};

И определён объект этого класса

Student s;

Выполняется операция: ++s;

Каким образом, компилятор будет воспринимать вызов функции-операции?

Как вызов метода класса: s.operator++();

1. В программе описан класс

class Student {

friend Student& operator++(Student&);

}

И определён объект этого класса

Student s;

Выполняется операция ++s;

Каким образом, компилятор будет воспринимать вызов функции-операции?

Как вызов глобальной функции operator++(s);

1. В программе описан класс

class Student {

bool operator<(Student &P);

}

И определенны объекты этого класса

Student a,b;

Выполняется операция

cout <<a<b;

Каким образом, компилятор будет воспринимать вызов функции-операции?

Приоритет операции << выше, чем у оператора <. Программа не скомпилируется.

Добавив скобки cout << (a<b)

Как вызов метода класса: a.operator<(b);

1. В программе описан класс

class Student {

friend bool operator>(const Person&, Person&)

}

И определены объекты этого класса

Student a,b;

Выполняется операция cout << a>b;

Каким образом, компилятор будет воспринимать вызов функции-операции?

Аналогично как в 14 вопросе, приоритет у оператора << выше, программа не скомпилируется.

**ПРИЛОЖЕНИЕ 1**

lab3.cpp

//-------------------------------------

#include <iostream>

#include "lab3\_time.hpp"

//-------------------------------------

#define IDENT\_PRINT printf("\n--------------------------\n\n")

//-------------------------------------

int main() {

IDENT\_PRINT;

//call default constructor

std::cout << "call default constructor\n";

Time time;

std::cout << time << std::endl;

IDENT\_PRINT;

//call constructor with parameters

std::cout << "call constructor with parameters\n";

Time time1(21, 30);

std::cout << time1 << std::endl;

IDENT\_PRINT;

//call copy constructor

std::cout << "call copy constructor:\n";

Time time2(time1);

std::cout << time2 << std::endl;

IDENT\_PRINT;

std::cin >> time;

std::cin >> time1;

IDENT\_PRINT;

time2 = time + time1;

std::cout << "operator +:" << std::endl;

std::cout << time2 << std::endl;

IDENT\_PRINT;

std::cout << "operator ==:" << std::endl;

std::cout << "time:" << &time << " == time1:" << &time1 << " ? " << ((time == time1) ? "true" : "false")<< std::endl;

IDENT\_PRINT;

return 0;

}

//-------------------------------------

lab3\_time.hpp

//-------------------------------------

#ifndef LAB2\_STUDENT\_HPP\_INCLUDED

#define LAB2\_STUDENT\_HPP\_INCLUDED

//-------------------------------------

#include <iostream>

#include <cstdint>

//-------------------------------------

#define MAX\_SECONDS 60

//-------------------------------------

class Time {

public:

Time();

Time(int minutes, int seconds);

Time(const Time& time);

~Time();

int getMinutes() { return m\_minutes; }

int getSeconds() { return m\_seconds; }

void setMinutes(int minutes) { m\_minutes = minutes; }

void setSeconds(int seconds) { m\_seconds = seconds % MAX\_SECONDS; }

void reset() { m\_minutes = 0; m\_seconds = 0; }

Time& operator=(const Time& time);

Time operator+(const Time& time);

bool operator==(const Time& time) const;

friend std::istream& operator>>(std::istream& input, Time& time);

friend std::ostream& operator<<(std::ostream& output, Time& time);

private:

int m\_minutes,

m\_seconds;

};

//-------------------------------------

#endif // LAB2\_STUDENT\_HPP\_INCLUDED

//-------------------------------------

lab3\_time.cpp

//-------------------------------------

#include "lab3\_time.hpp"

//-------------------------------------

Time::Time() {

m\_minutes = 0;

m\_seconds = 0;

}

//-------------------------------------

Time::Time(int minutes, int seconds) {

this->setMinutes(minutes);

this->setSeconds(seconds);

}

//-------------------------------------

Time::Time(const Time& time) {

m\_minutes = time.m\_minutes;

m\_seconds = time.m\_seconds;

}

//-------------------------------------

Time::~Time() {

//

}

//-------------------------------------

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

if(this == &time)

return \*this;

m\_minutes = time.m\_minutes;

m\_seconds = time.m\_seconds;

return \*this;

}

//-------------------------------------

Time Time::operator+(const Time& time) {

uint32\_t newtime = time.m\_minutes\*MAX\_SECONDS + time.m\_seconds;

newtime += (m\_minutes\*MAX\_SECONDS + m\_seconds);

return Time(newtime/MAX\_SECONDS, newtime%MAX\_SECONDS);

}

//-------------------------------------

bool Time::operator==(const Time& time) const {

return time.m\_minutes == m\_minutes && time.m\_seconds == m\_seconds;

}

//-------------------------------------

std::istream& operator>>(std::istream& input, Time& time) {

#ifdef debug\_v

std::cout << "object: " << &time << std::endl;

#endif //debug\_v

std::cout << "minutes: ";

input >> time.m\_minutes;

std::cout << "seconds: ";

input >> time.m\_seconds;

return input;

}

//-------------------------------------

std::ostream& operator<<(std::ostream& output, Time& time) {

return (output <<

#ifdef debug\_v

"object: " << &time << "\n" <<

#endif // debug\_v

"time: " << time.getMinutes() << ":" << (int)time.getSeconds());

}

//-------------------------------------