Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

ПНИПУ

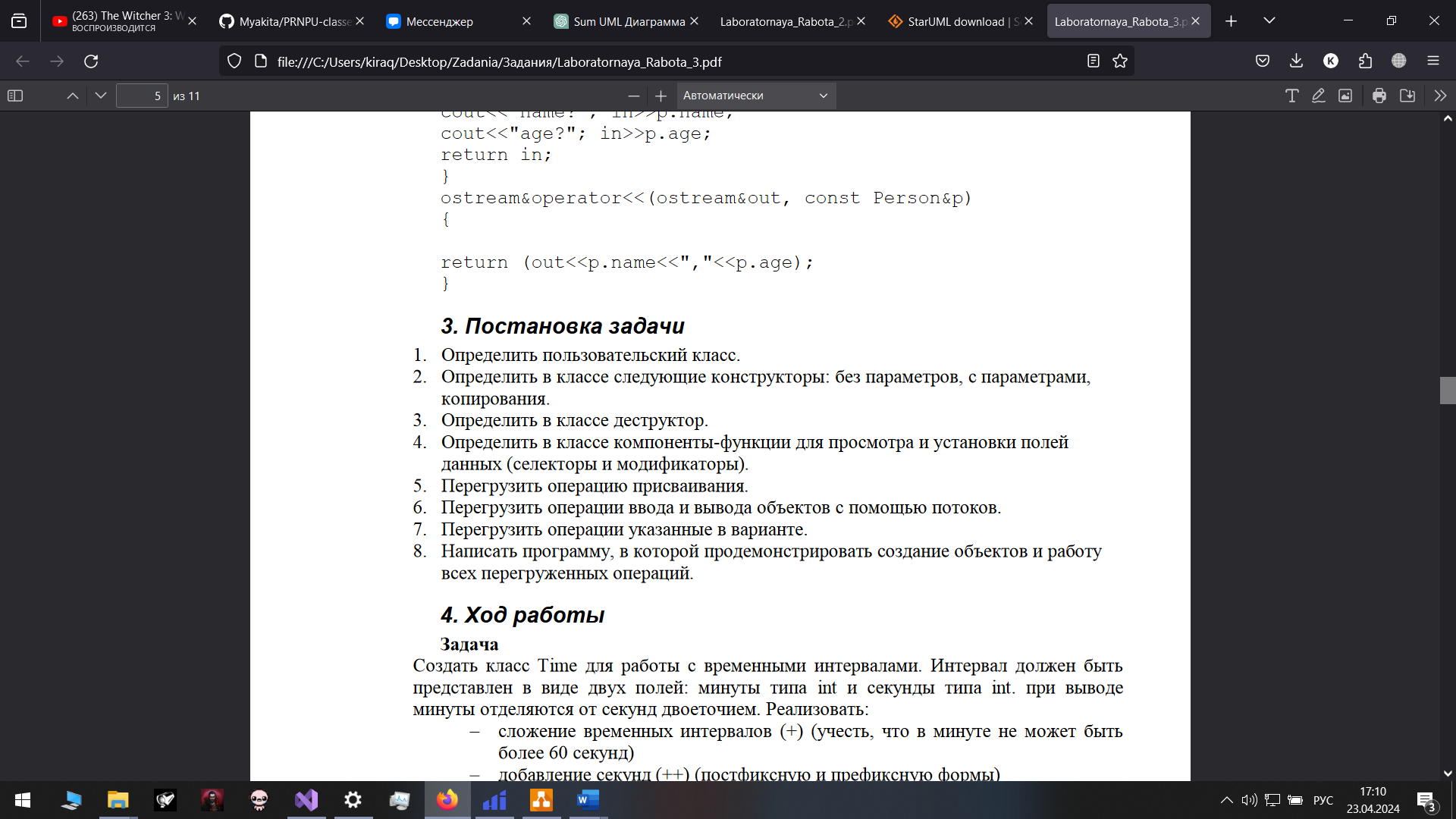
Лабораторная работа  
«Перегрузка Операторов. Друж.функции»

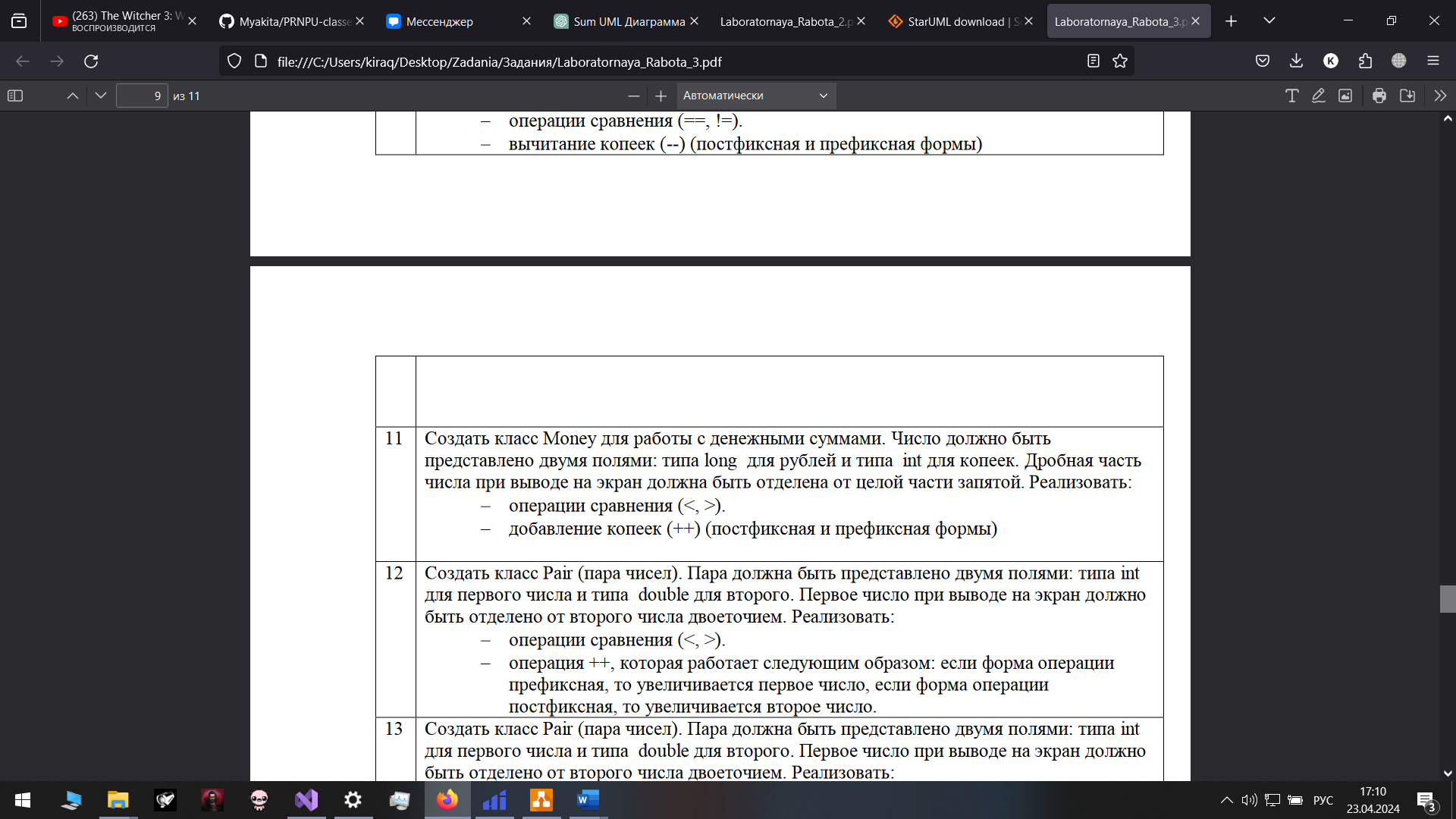
Выполнил:   
студент группы РИС-23-1б   
Мокрушин Никита Дмитриевич

Проверила:   
доцент кафедры ИТАС   
О.А. Полякова

Пермь 2024 г.

Постановка задачи:





Анализ задачи:

1. **Конструкторы**: Класс Money имеет три конструктора: конструктор по умолчанию, конструктор с параметрами и конструктор копирования. Конструкторы инициализируют атрибуты объекта Money.
2. **Деструктор**: В классе Money определен деструктор, который освобождает ресурсы (в данном случае не используется).
3. **Селекторы и модификаторы**: Класс Money содержит методы для получения и установки значений атрибутов rubles и kopecks.
4. **Перегрузка операций**: В классе перегружены операции присваивания (=), ввода (>>), вывода (<<), сравнения (<, >) и инкремента (++).
5. **Main функция**: В функции main() создаются объекты класса Money, выполняются операции сравнения и инкремента, а также выводятся результаты в консоль.

Код:

#include <iostream>

using namespace std;

class Money {

private:

long rubles;

int kopecks;

public:

// Конструкторы

Money() : rubles(0), kopecks(0) {}

Money(long r, int k) : rubles(r), kopecks(k) {}

Money(const Money& other) : rubles(other.rubles), kopecks(other.kopecks) {}

// Деструктор

~Money() {}

// Селекторы и модификаторы

long getRubles() const { return rubles; }

int getKopecks() const { return kopecks; }

void setRubles(long r) { rubles = r; }

void setKopecks(int k) { kopecks = k; }

// Перегрузка операции присваивания

Money& operator=(const Money& other) {

if (this != &other) {

rubles = other.rubles;

kopecks = other.kopecks;

}

return \*this;

}

// Перегрузка операций ввода и вывода

friend ostream& operator<<(ostream& os, const Money& money) {

os << money.rubles << "," << money.kopecks;

return os;

}

friend istream& operator>>(istream& is, Money& money) {

is >> money.rubles >> money.kopecks;

return is;

}

// Перегрузка операции сравнения <

bool operator<(const Money& other) const {

if (rubles < other.rubles)

return true;

else if (rubles == other.rubles && kopecks < other.kopecks)

return true;

return false;

}

// Перегрузка операции сравнения >

bool operator>(const Money& other) const {

if (rubles > other.rubles)

return true;

else if (rubles == other.rubles && kopecks > other.kopecks)

return true;

return false;

}

// Перегрузка операции постфиксного инкремента ++

Money operator++(int) {

Money temp(\*this);

kopecks++;

if (kopecks >= 100) {

rubles++;

kopecks -= 100;

}

return temp;

}

// Перегрузка операции префиксного инкремента ++

Money& operator++() {

kopecks++;

if (kopecks >= 100) {

rubles++;

kopecks -= 100;

}

return \*this;

}

};

int main() {

setlocale(0, "");

system("chcp 1251>nul");

Money m1(10, 50);

Money m2(5, 75);

cout << "m1: " << m1 << endl;

cout << "m2: " << m2 << endl;

if (m1 < m2)

cout << "m1 меньше, чем m2" << endl;

else

cout << "m1 не меньше, чем m2" << endl;

if (m1 > m2)

cout << "m1 больше, чем m2" << endl;

else

cout << "m1 не больше, чем m2" << endl;

cout << "Постфиксный инкремент m1: " << (m1++) << endl;

cout << "Теперь m1: " << m1 << endl;

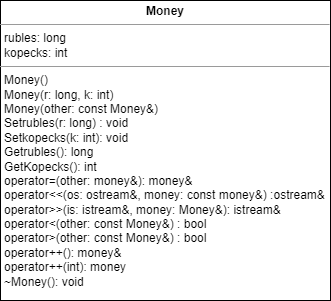
cout << "Префиксный инкремент m2: " << (++m2) << endl;

cout << "Теперь m2: " << m2 << endl;

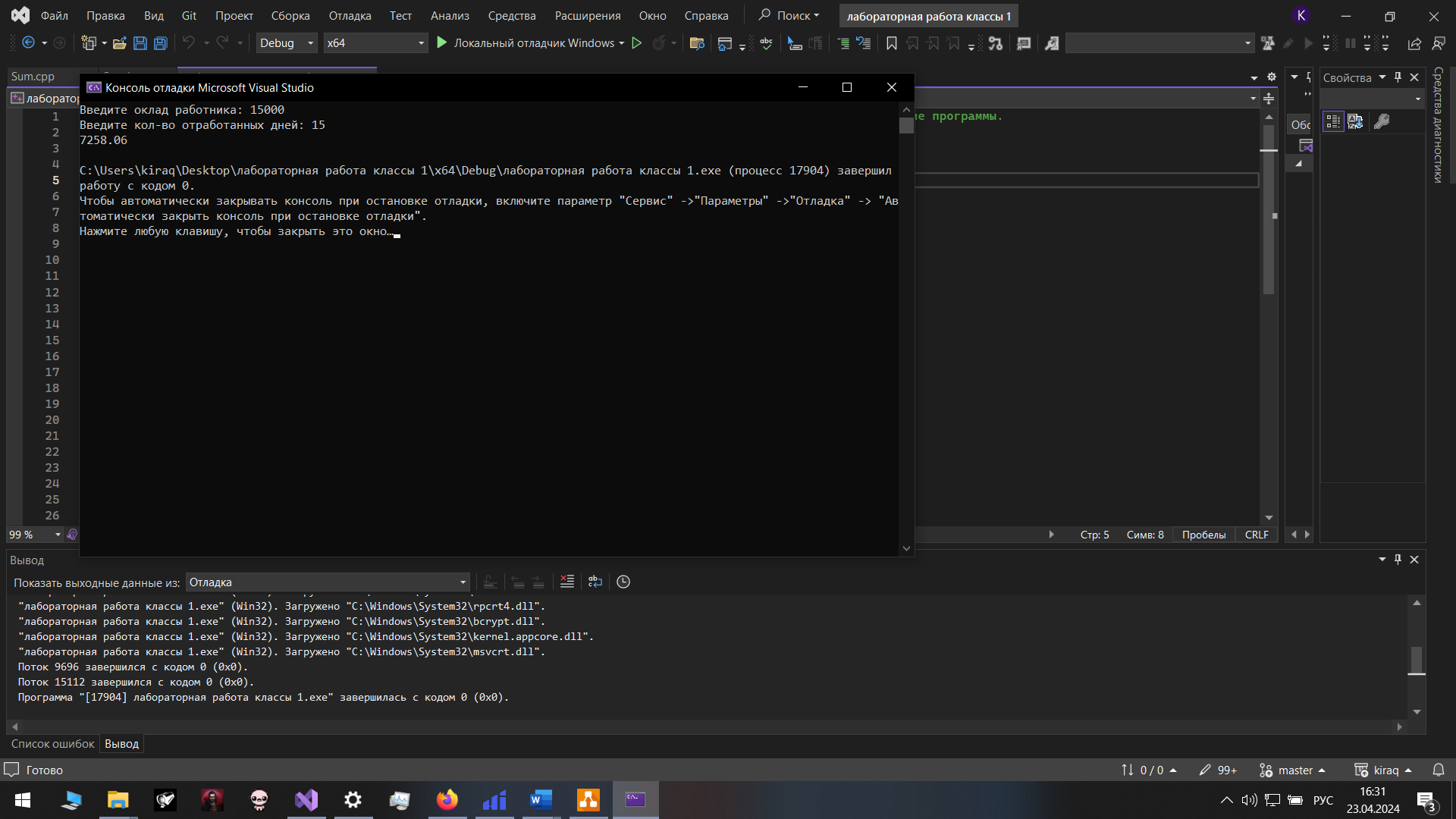
return 0;

}

UML Диаграммы:



Вывод:



программа работает корректно и выдаёт требуемый результат.  
Ответы на вопросы:

1. **Дружественные функции и классы** используются для предоставления доступа к приватным или защищенным членам класса другим функциям или классам, которые не являются членами этого класса.
2. **Правила описания и особенности дружественных функций**:
   * Дружественные функции объявляются внутри класса с помощью ключевого слова friend.
   * Дружественные функции не являются методами класса, но имеют доступ к приватным и защищенным членам класса.
   * Дружественные функции могут быть как обычными функциями, так и другими классами.
3. Унарные операции могут быть перегружены как методы класса с одним операндом или как дружественные функции с одним операндом.
4. Унарная функция-операция, определяемая внутри класса, должна иметь один операнд.
5. Унарная функция-операция, определяемая вне класса, также должна иметь один операнд.
6. Бинарная функция-операция, определяемая внутри класса, должна иметь два операнда, один из которых является объектом класса.
7. Бинарная функция-операция, определяемая вне класса, также должна иметь два операнда, один из которых является объектом класса.
8. Перегрузка префиксных и постфиксных унарных операций отличается тем, что префиксная операция выполняется до увеличения или уменьшения значения операнда, а постфиксная - после.
9. Операцию присваивания можно перегрузить как методом класса, так и как дружественной функцией.
10. Операция присваивания должна возвращать ссылку на объект, для поддержания цепочки присваиваний (например, a = b = c).
11. Операции ввода-вывода можно перегрузить как методами класса, так и как дружественными функциями.
12. При выполнении операции ++s компилятор будет воспринимать вызов метода operator++() объекта s.
13. При выполнении операции ++s компилятор будет воспринимать вызов дружественной функции operator++(Student&).
14. При выполнении операции cout << a < b компилятор будет воспринимать вызов метода operator<() объекта a.
15. При выполнении операции cout << a > b компилятор будет воспринимать вызов дружественной функции operator>(const Person&, Person&).