Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе 4**

Дисциплина: Основы алгоритмизации и программирования

Вариант 10

Выполнил:

Студент группы ИВТ-22-2б

Мифтахов Марат Ринатович

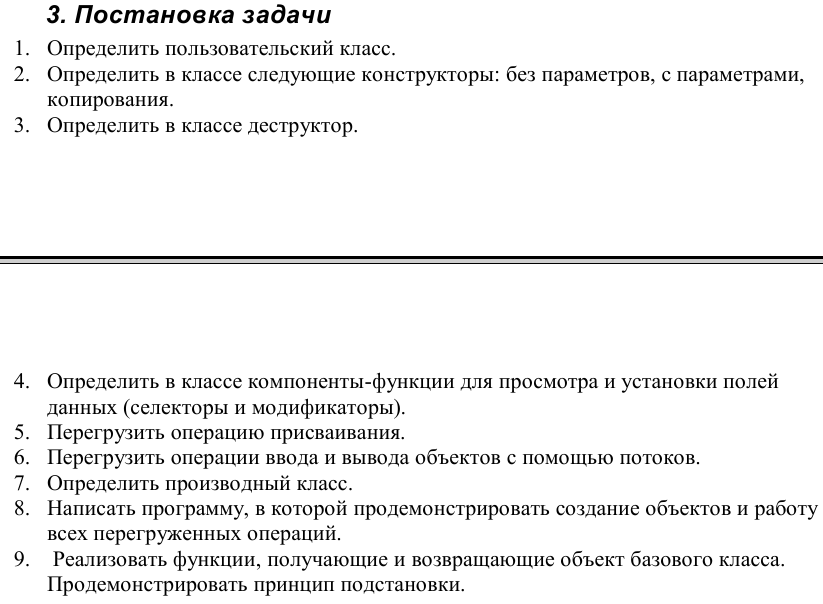
Проверил:

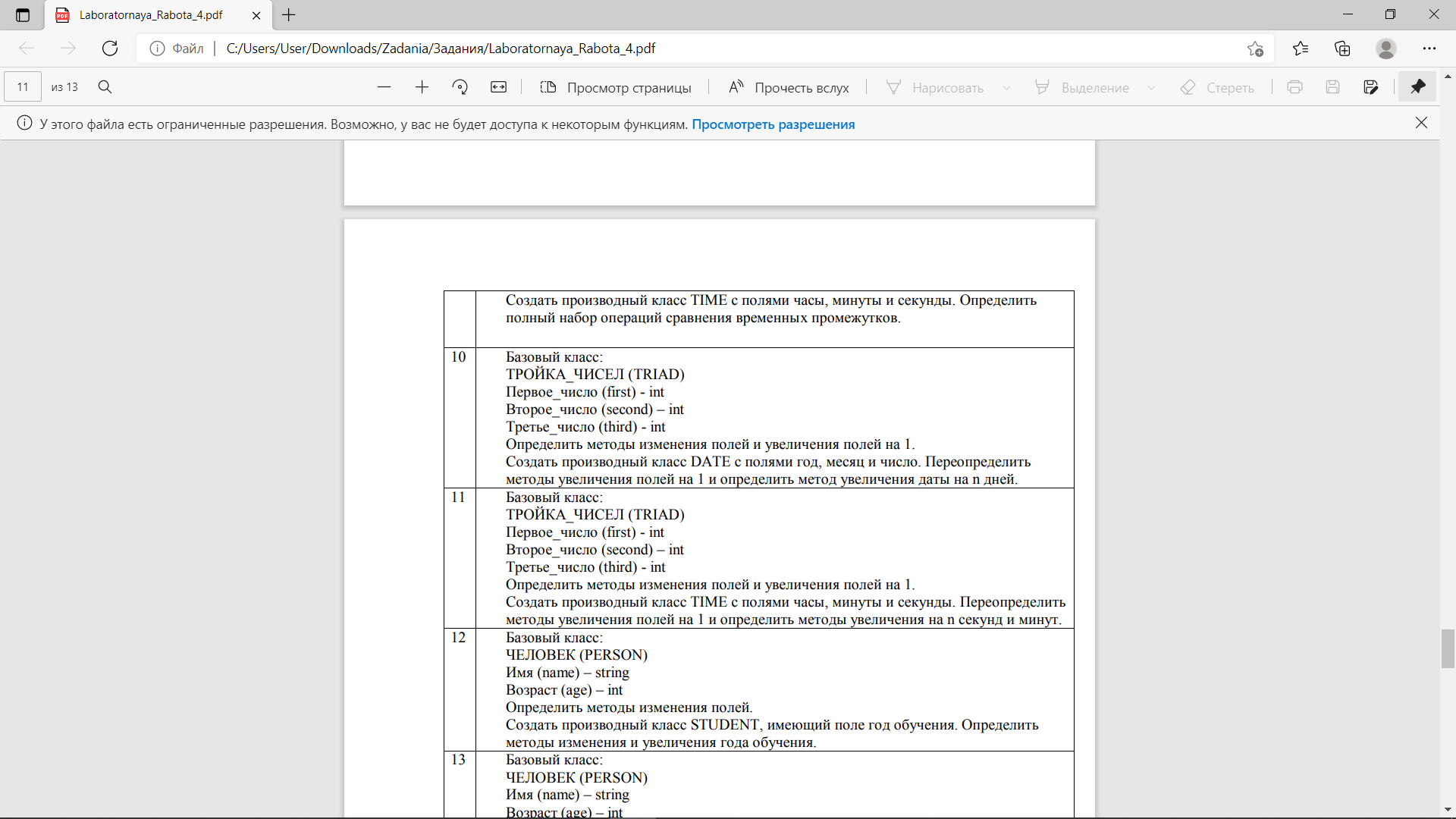
Доцент кафедры ИТАС

Полякова О. А.

**Пермь, 2023**

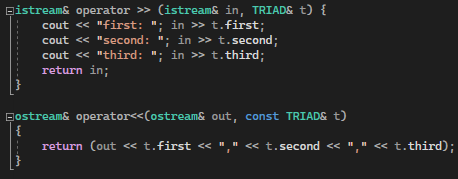
**Постановка задачи**



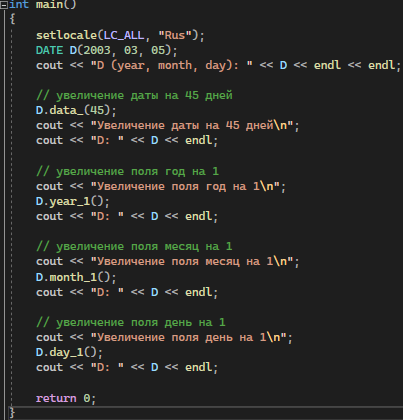


**Анализ задачи**

1. Используемые типы данных
   1. Int
   2. Triad
   3. Date
2. Действия над используемыми данными
   1. Int – арифметические операции.
3. Структуры
   1. Triad – класс для хранения значений чисел.
   2. Date : public Triad – класс для хранения дат. Наследуется для класса Triad.
4. Организация ввода-вывода
   1. Cin
   2. Cout
   3. Перегрузка оператора << и >> .



1. Действия для решения задачи. Демонстрация работы класса в функции main().



**Код программы**

**Main.cpp**

#include "Triad.h"

#include "Date.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "Rus");

DATE D(2003, 03, 05);

cout << "D (year, month, day): " << D << endl << endl;

// увеличение даты на 45 дней

D.data\_(45);

cout << "Увеличение даты на 45 дней\n";

cout << "D: " << D << endl;

// увеличение поля год на 1

cout << "Увеличение поля год на 1\n";

D.year\_1();

cout << "D: " << D << endl;

// увеличение поля месяц на 1

cout << "Увеличение поля месяц на 1\n";

D.month\_1();

cout << "D: " << D << endl;

// увеличение поля день на 1

cout << "Увеличение поля день на 1\n";

D.day\_1();

cout << "D: " << D << endl;

return 0;

}

**Triad.h**

#pragma once

#include <iostream>

using namespace std;

class TRIAD

{

protected:

int first;

int second;

int third;

public:

// конструкторы

TRIAD(int, int, int);

TRIAD();

TRIAD(TRIAD&);

~TRIAD();

// модификаторы

void setFirst(int);

void setSecond(int);

void setThird(int);

// селекторы

int getFirst();

int getSecond();

int getThird();

// перегрузка оператора присваивания

TRIAD& operator= (const TRIAD& p);

// методы увеличения полей на 1

void first\_1();

void second\_1();

void third\_1();

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

friend ostream& operator<<(ostream& out, const TRIAD& tmp);

};

**Triad.cpp**

#include "Triad.h"

// конструкторы

TRIAD::TRIAD(int f, int s, int t)

: first(f), second(s), third(t) {}

TRIAD::TRIAD()

: first(0), second(0), third(0) {}

TRIAD::TRIAD(TRIAD& p)

: first(p.first), second(p.second), third(p.third) {}

TRIAD::~TRIAD() {}

// модификаторы

void TRIAD::setFirst(int f) {

first = f;

}

void TRIAD::setSecond(int s) {

second = s;

}

void TRIAD::setThird(int t) {

third = t;

}

// селекторы

int TRIAD::getFirst() {

return first;

}

int TRIAD::getSecond() {

return second;

}

int TRIAD::getThird() {

return third;

}

// перегрузка оператора присваивания

TRIAD& TRIAD::operator= (const TRIAD& p) {

if (&p == this) return \*this;

first = p.first;

second = p.second;

third = p.third;

return \*this;

}

// методы увеличения полей на 1

void TRIAD::first\_1() {

first++;

}

void TRIAD::second\_1() {

second++;

}

void TRIAD::third\_1() {

third++;

}

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

cout << "first: "; in >> t.first;

cout << "second: "; in >> t.second;

cout << "third: "; in >> t.third;

return in;

}

ostream& operator<<(ostream& out, const TRIAD& t)

{

return (out << t.first << "," << t.second << "," << t.third);

}

**Date.h**

#pragma once

#include "Triad.h"

class DATE : public TRIAD {

int year = first;

int month = second;

int day = third;

public:

DATE() {}

DATE(int year, int month, int day) :TRIAD(year, month, day) {};

~DATE() {}

void data\_(int);

void year\_1();

void month\_1();

void day\_1();

};

**Date.cpp**

#include "Date.h"

void DATE::data\_(int n) {

third = day + n;

int t = 0;

while (third > 31) {

t++;

third -= 31;

}

second += t;

t = 0;

while (second > 12) {

t++;

second -= 12;

}

first += t;

}

void DATE::year\_1() {

first++;

}

void DATE::month\_1() {

second++;

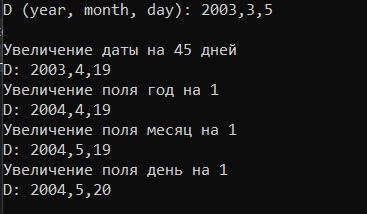
}

void DATE::day\_1() {

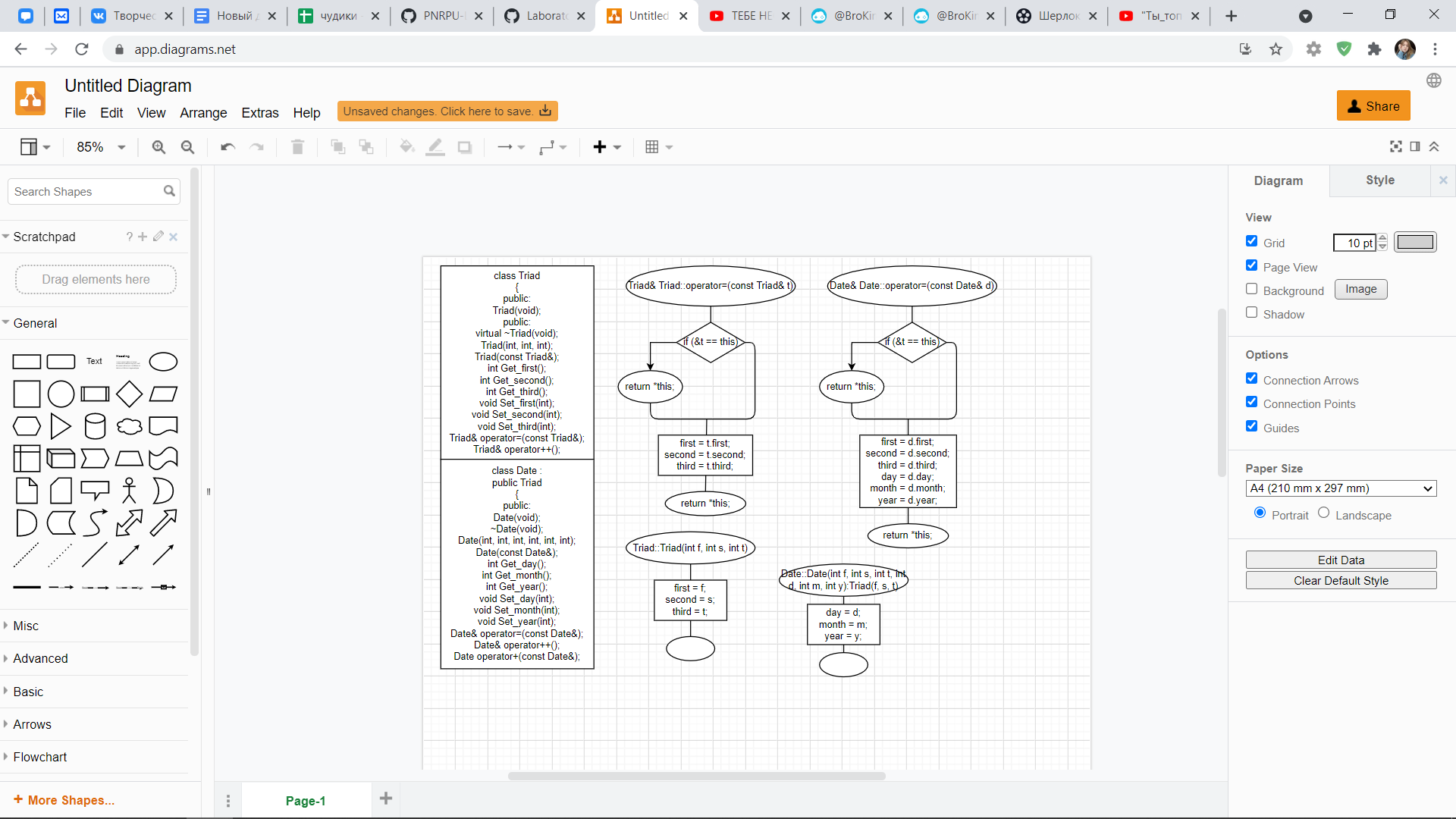
third++;

}

**Скриншоты**



**Блок-схема**



**Ответы на контрольные вопросы**



1. Наследование позволяет реализовать типизированное повторное использование объектов, создание иерархии «тип-подтип» с сохранением подтипом всех свойств своего предка. Можно рассматривать наследование для всех трех составляющих объекта.



2. Когда вы открыто наследуете родительский класс, то унаследованные public-члены остаются public.



3. Когда вы открыто наследуете родительский класс, то унаследованные private-члены остаются недоступными для дочернего класса.



4. К protected-членам родительского класса доступ открыт для членов дочернего класса.



5. 



6. Конструкторы не наследуются.



7. Деструкторы не наследуются.



8. 



9. 

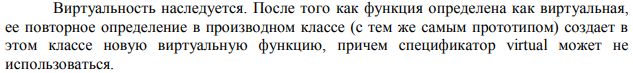


10. 

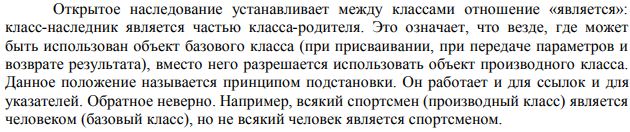


11. 



12. 



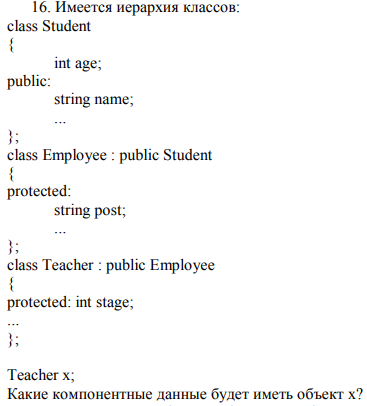
13. 



14. 



15. Пункт 13 и 14 – принцип подстановки.



16. age, name, post, stage.



17. Student(); Employee() : Student(); Teacher() : Employee();