Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа по ООП**

**«№4»**

Выполнил:

студент группы РИС-23-1б

Кривошеин Александр Антонович

Проверила:

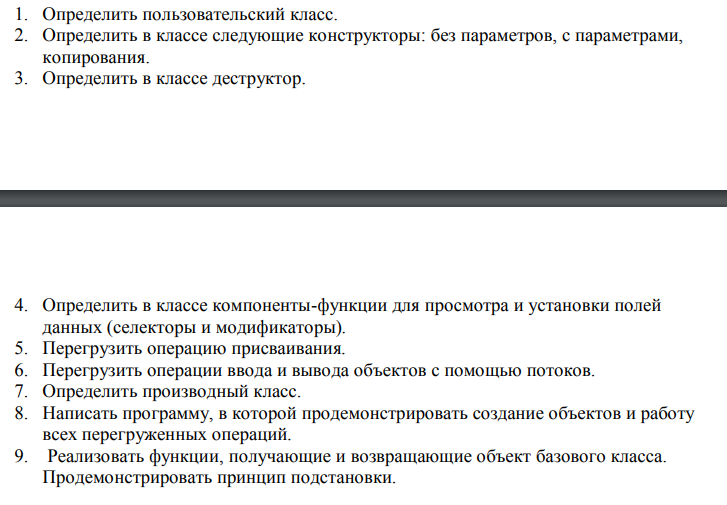
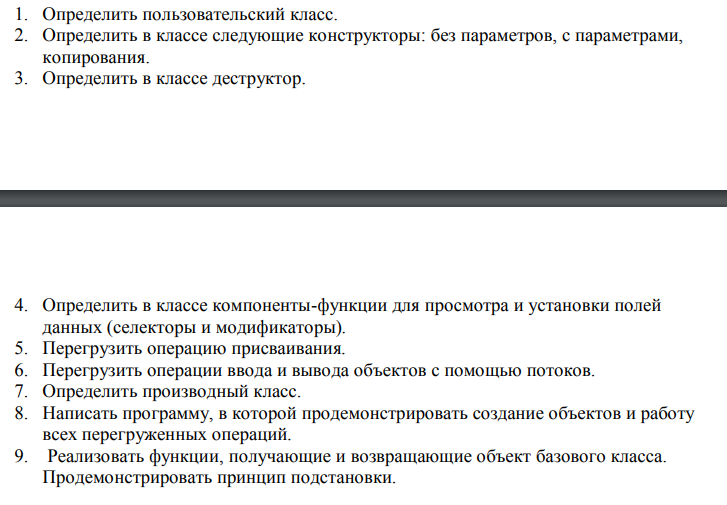
доцент кафедры ИТАС

Полякова Ольга Андреевна

2024 г.

**Разработка алгоритма**

**Постановка задачи:**

** Анализ задачи:**

1. Будем использовать несколько файлов для удобной работы с кодом.
2. Создадим пользовательский класс Person и производный класс Student.
3. Добавим конструктор без параметров, с параметрами, конструктор копирования для обоих классов. Также определим пустые деструкторы.
4. Создадим геттеры и сеттеры в соответствии с поставленной задачей, перегрузим функцию ввода/вывода для обоих классов.
5. Напишем код для проверки реализации всех функций.

**Код программы на C++:**

main.cpp:

#include "Person.h"

using namespace std;

int main()

{

    Person person1("Krivoshein Alexandr Antonovich", 19);

    cout << person1;

    Person person2;

    cout << person2;

    Person person3(person1);

    cout << person3;

    Student student1("Kaminsky Mihail Evgenievich", 19, 0);

    cout << student1;

    student1.setYoS(1);

    cout << student1;

    student1++;

    cout << student1;

    Student student2;

    cout << student2 << endl << endl;

    cin >> student2;

    Student student3;

    student3 = student2;

    cout << student3 << endl;

    return 0;

    cout << student3;

}

Person.h:

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Person

{

protected:

    string name;

    int age;

public:

    Person();

    Person(string, int);

    Person(const Person &);

    ~Person();

    void setName(const string);

    void setAge(const int);

    void operator=(const Person &);

    friend ostream &operator<<(ostream &, Person &);

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

};

class Student : public Person

{

protected:

    int year\_of\_study;

public:

    Student();

    Student(string, int, int);

    void setYoS(int);

    Student &operator++();

    Student &operator++(int);

    friend ostream &operator<<(ostream &, const Student &);

    friend istream &operator>>(istream &, Student &);

};

Student &Student::operator++()

{

    ++this->year\_of\_study;

    return \*this;

}

Student &Student::operator++(int)

{

    this->year\_of\_study++;

    return \*this;

}

Person::Person()

{

    cout << "The constructor without parameters for the object was called " << this << endl;

    this->name = "-";

    this->age = 0;

}

Person::Person(string name, int age)

{

    cout << "The constructor with the parameters for the object was called " << this << endl;

    this->name = name;

    this->age = age;

}

Person::Person(const Person &other)

{

    cout << "The copy constructor for the object was called " << this << endl;

    this->name = other.name;

    this->age = other.age;

}

Person::~Person()

{

    cout << "The destructor for the object was called " << this << endl;

}

void Person::setName(const string name)

{

    this->name = name;

}

void Person::setAge(const int age)

{

    this->age = age;

}

void Person::operator=(const Person &person)

{

    this->name = person.name;

    this->age = person.age;

}

ostream &operator<<(ostream &stream, Person &person)

{

    stream << "name: " << person.name << endl

           << "age: " << person.age << "\n\n";

    return stream;

}

istream &operator>>(istream &stream, Person &person)

{

    string name;

    int age;

    cout << "Enter name: ";

    cin.seekg(cin.eof());

    getline(cin, name);

    cout << "Enter age: ";

    cin >> age;

    person.setName(name);

    person.setAge(age);

    return stream;

}

Student::Student()

{

    this->name = "-";

    this->age = 0;

    this->year\_of\_study = 0;

}

Student::Student(string name, int age, int yos)

{

    this->name = name;

    this->age = age;

    this->year\_of\_study = yos;

}

void Student::setYoS(int yos)

{

    this->year\_of\_study = yos;

}

ostream &operator<<(ostream &stream, const Student &student)

{

    stream << "name: " << student.name << endl

           << "age: " << student.age << endl

           << "Year of study: " << student.year\_of\_study << endl << endl;

    return stream;

}

istream &operator>>(istream &stream, Student &student)

{

    string name;

    int yos, age;

    cout << "Enter name: ";

    cin.seekg(cin.eof());

    getline(stream, student.name);

    cout << "Enter age: ";

    stream >> age;

    student.age = age;

    cout << "Enter year of study: ";

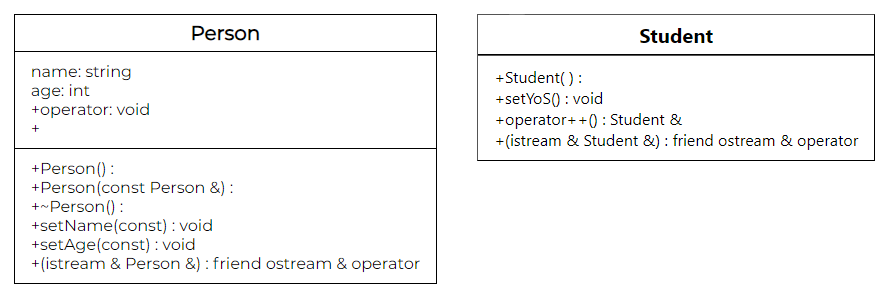
    stream >> yos;

    student.setYoS(yos);

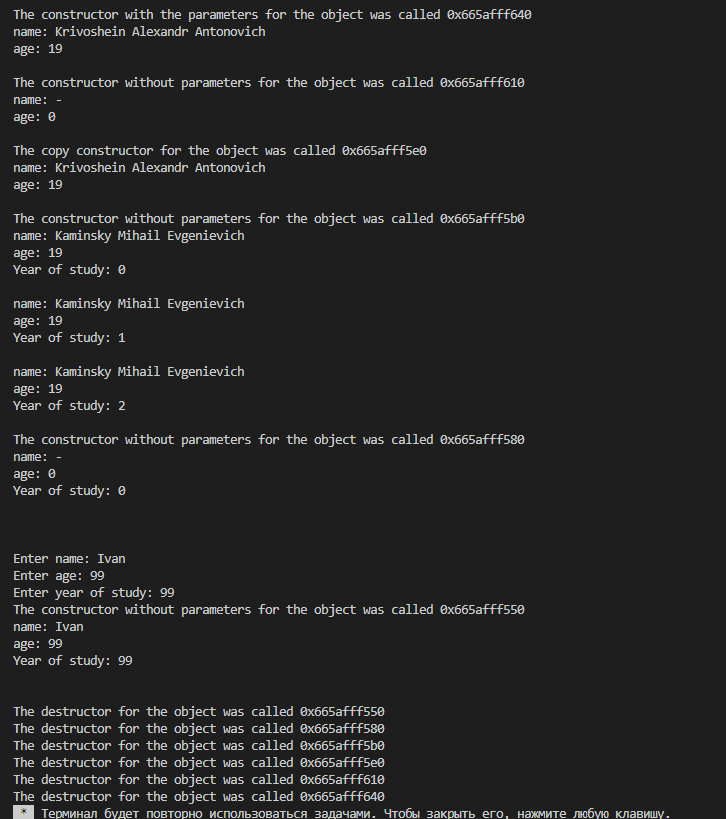
    return stream;

}

Диаграмма класса:



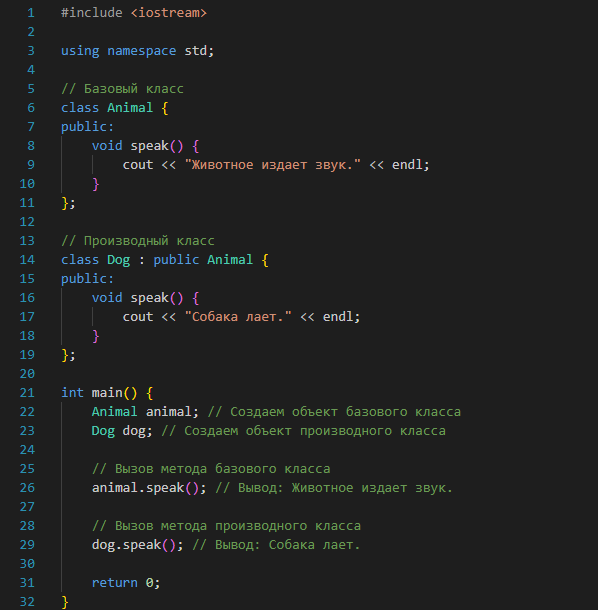
Пример работы программы:



**Контрольные вопросы**

1. **Для чего используется механизм наследования?**

Механизм наследования в C++ используется для создания иерархий классов, позволяющих повторно использовать код и обеспечивающих возможность расширения функциональности классов без изменения их исходного кода. Это позволяет создавать более гибкие и мощные программы.

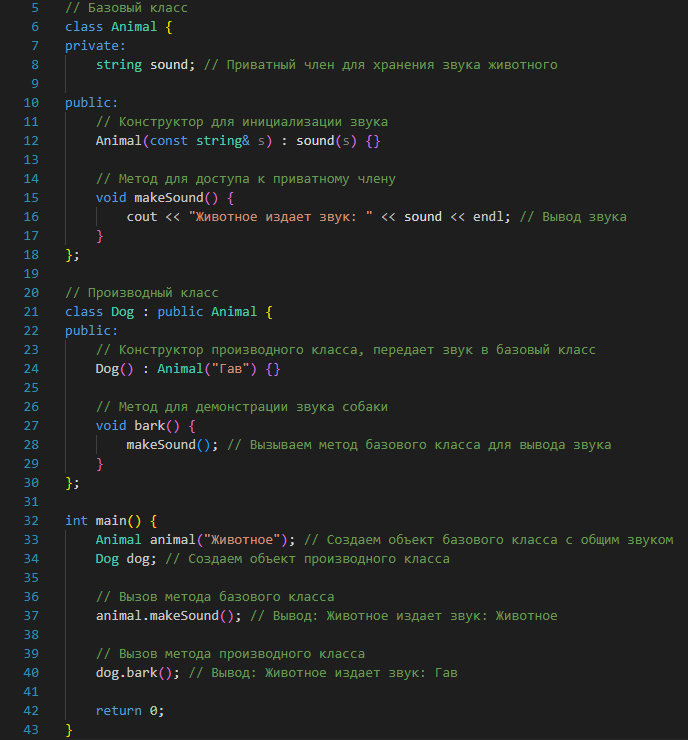


1. **Каким образом наследуются компоненты класса, описанные со спецификатором public?**

Компоненты класса, описанные со спецификатором public, наследуются открыто. Это означает, что они доступны для использования в производном классе так же, как и в базовом классе. Они могут быть переопределены в производном классе, если это необходимо. (пример в вопросе 1)

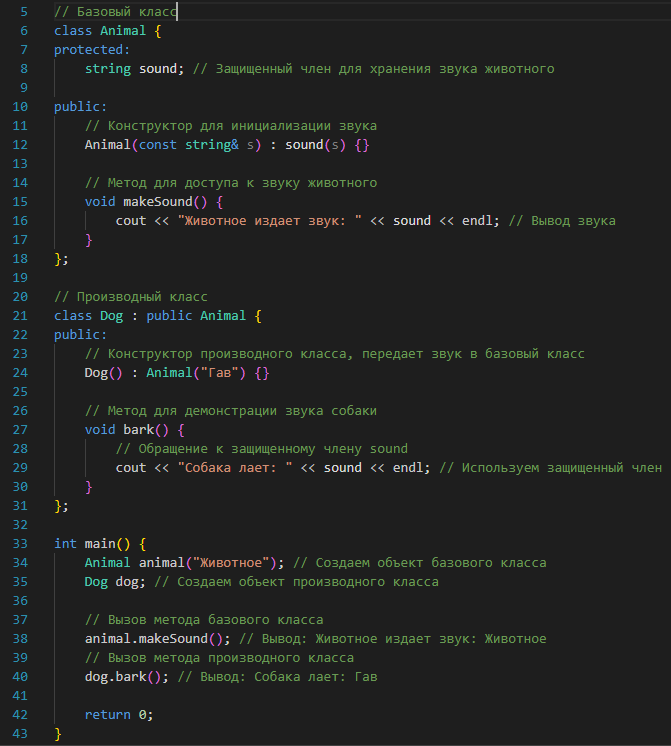
1. **Каким образом наследуются компоненты класса, описанные со спецификатором private?**

Компоненты класса, описанные со спецификатором private, не наследуются. Они являются приватными членами базового класса и недоступны для прямого доступа или изменения в производном классе.



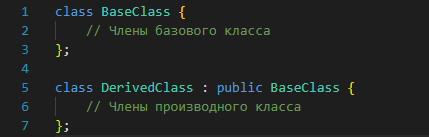
1. **Каким образом наследуются компоненты класса, описанные со спецификатором protected?**

Компоненты класса, описанные со спецификатором protected, наследуются защищенно. Они доступны для использования в производном классе, но не доступны напрямую извне производного класса. Защищенные члены могут быть переопределены в производном классе



1. **Каким образом описывается производный класс?**

Производный класс описывается путем объявления его базовым классом



1. **Наследуются ли конструкторы?**

Конструкторы не наследуются, а создаются заново в каждом производном классе.

1. **Наследуются ли деструкторы?**

Не наследуются, но они вызываются, когда дочерний класс инициализирует свой объект.

1. **В каком порядке конструируются объекты производных**

**классов?**

Объекты класса конструируются снизу-вверх: сначала базовый, потом компоненты-объекты (если они имеются), а потом сам производный класс. Таким образом объект производного класса содержит в качестве подобъекта объект базового класса.

1. **В каком порядке уничтожаются объекты производных**

**классов?**

Деструкторы вызываются в обратном порядке от конструкторов.

1. **Что представляют собой виртуальные функции и механизм позднего связывания?**

Виртуальные функции и механизм позднего связывания позволяют полиморфно вызывать функцию, определенную в базовом классе, через объект производного класса. Это означает, что во время выполнения программы будет выбрана правильная версия функции, соответствующая типу объекта, даже если вызывается функция базового класса.

1. **Могут ли быть виртуальными конструкторы? Деструкторы?**

Конструкторы нет, деструкторы да.

1. **Наследуется ли спецификатор virtual?**

Да.

1. **Какое отношение устанавливает между классами открытое наследование?**

Все члены базового класса становятся доступными в производном

классе. Это позволяет производному классу полностью использовать

функциональность базового класса.

1. **Какое отношение устанавливает между классами закрытое наследование?**

Производный класс не имеет доступа к членам базового класса, описанным как private. Это ограничивает возможности производного класса по использованию функциональности базового класса. Закрытое (также как и защищенное) наследование не создает иерархии типов.

1. **В чем заключается принцип подстановки?**

Принцип подстановки гласит, что, если два выражения имеют одно и то же значение, они могут быть взаимозаменяемыми в любом контексте. В контексте наследования это означает, что, если объект производного класса может быть использован там, где ожидается объект базового класса, благодаря полиморфизму, это не должно привести к ошибкам во время выполнения программы.

1. **Имеется иерархия классов: class Student**

**{**

**int age;**

**public:**

**};**

**string name;**

**class Employee : public Student**

**{**

**protected:**

**};**

**string post;**

**class Teacher public Employee**

**{**

**protected: int stage;**

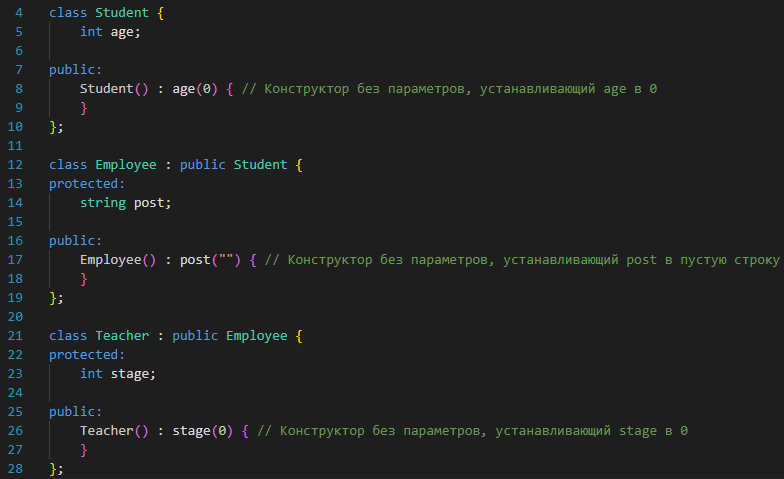
**};**

**Teacher x;**

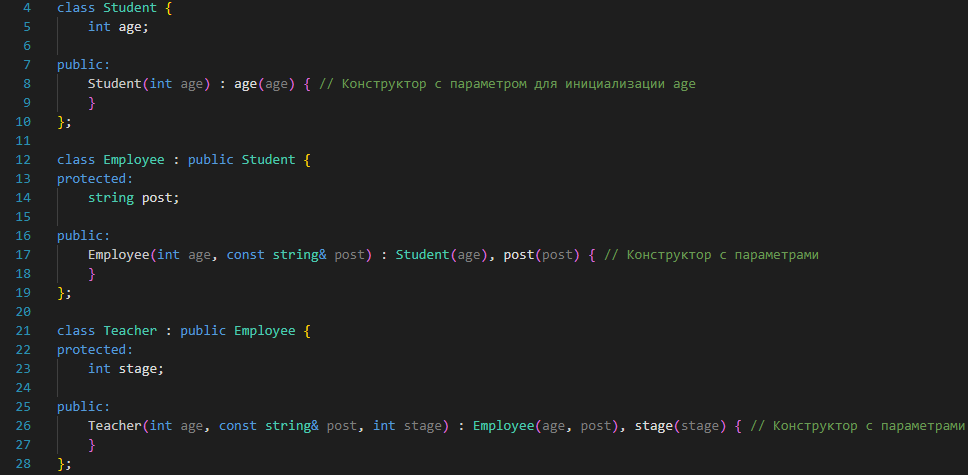
**Какие компонентные данные будет иметь объект х?**

Name и Stage, private age не наследуется, post наследуется в private секции.

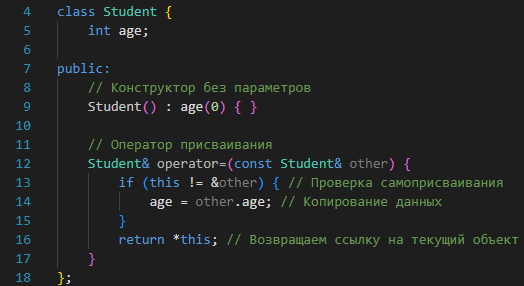
1. **Для классов Student, Employee и Teacher написать конструкторы без параметров.**

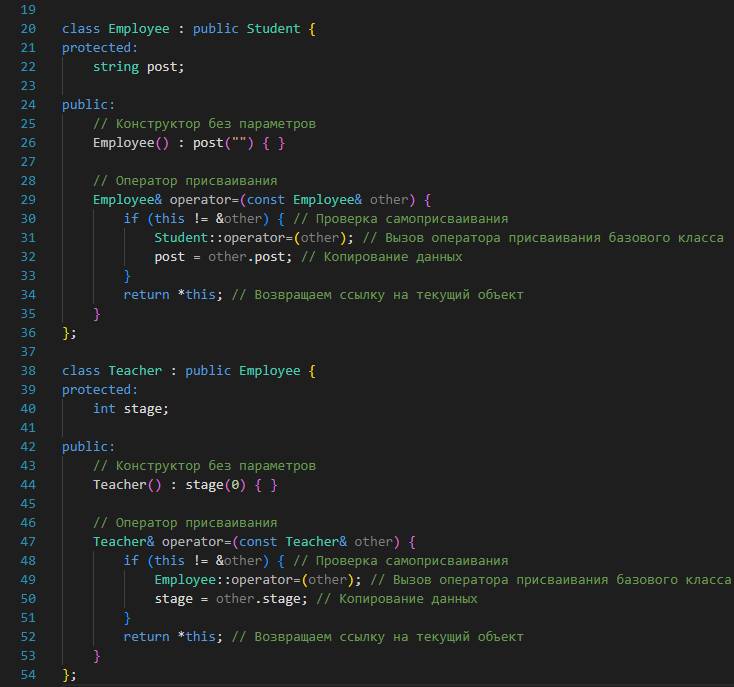


1. **Для классов Student, Employee и Teacher написать конструкторы с параметрами.**

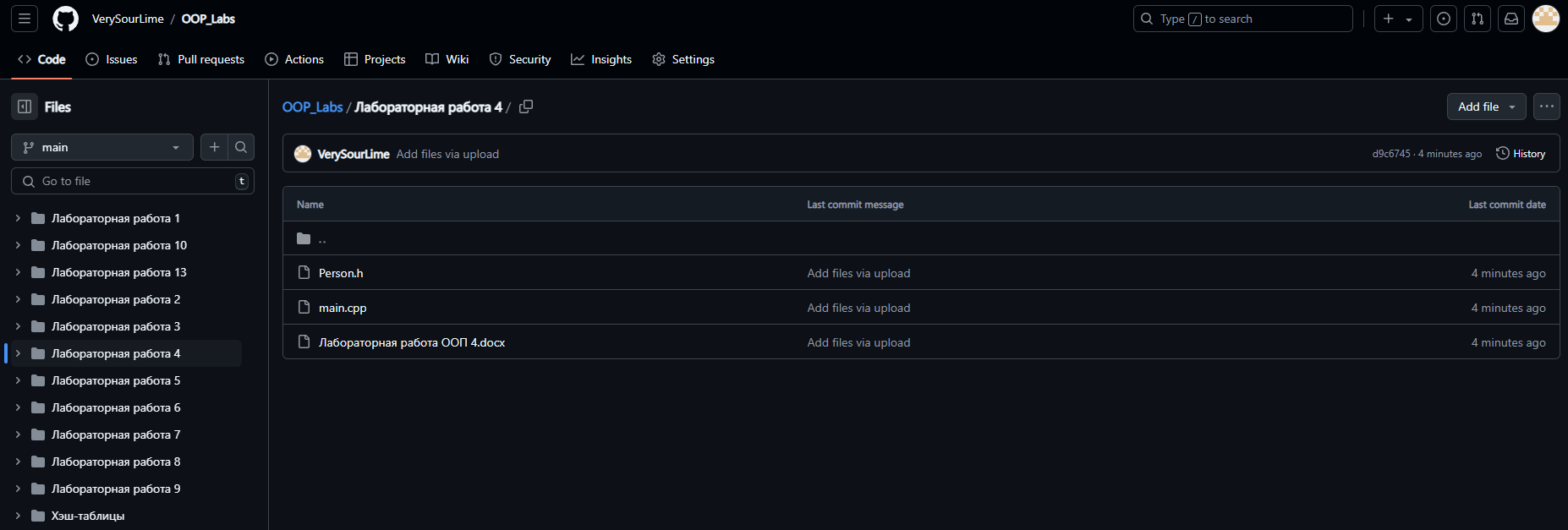


1. **Для классов Student, Employee и Teacher написать конструкторы копирования**
2. **Для классов Student, Employee и Teacher определить операцию присваивания.**





Скриншот из GitHub:

  
GitHub: https://github.com/VerySourLime/OOP\_Labs