Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы» направление подготовки: 09.03.04 – «Программная инженерия»

**Лабораторная работа №4.**

**«Простое наследование»**

Выполнил студент гр. РИС-24-2б

Молочко Артём Анатольевич

Проверил:

Доц. Каф. ИТАС

Ольга Андреевна Полякова

(оценка) (подпись)

(дата)

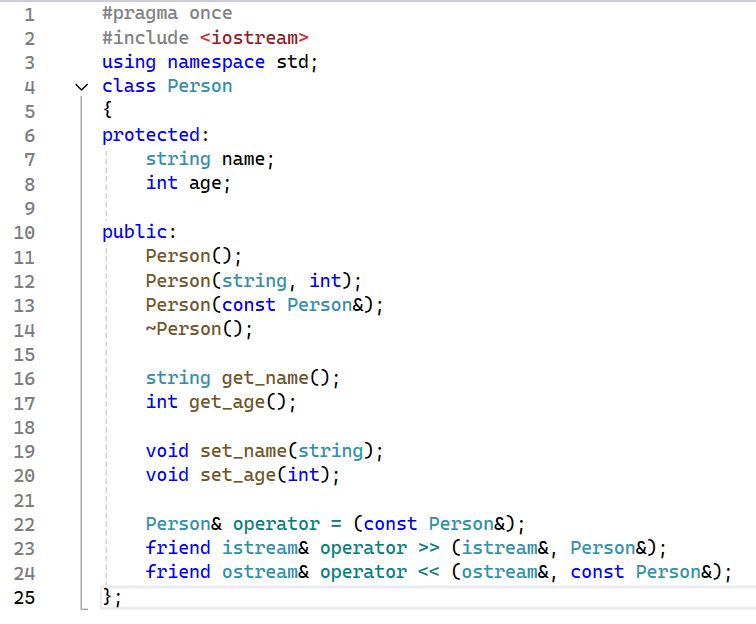
г. Пермь, 2025

**Постановка задачи**

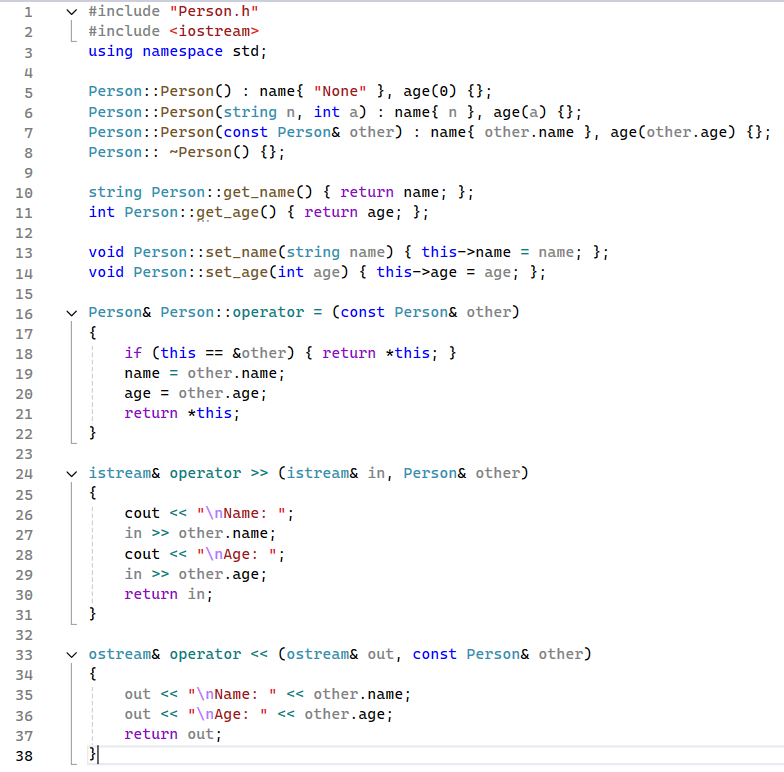
1. Создать пользовательский класс
2. Реализовать в классе три типа конструкторов:
   * Конструктор без параметров
   * Конструктор с параметрами
   * Конструктор копирования
3. Добавить деструктор класса
4. Реализовать методы для доступа к полям класса:
   * Селекторы (получение значений)
   * Модификаторы (изменение значений)
5. Перегрузить оператор присваивания (=)
6. Перегрузить операторы ввода (>>) и вывода (<<) для работы с потоками
7. Создать производный класс на основе базового
8. Разработать демонстрационную программу, которая:
   * Создает объекты классов
   * Показывает работу всех перегруженных операций
9. Реализовать функции, которые:
   * Принимают объект базового класса
   * Возвращают объект базового класса
   * Демонстрируют принцип подстановки (возможность работы с объектами производного класса как с объектами базового)
10. **Базовый класс Person (Человек)**
    * Поля:
      + name (тип string) – имя человека
      + age (тип int) – возраст
    * Методы:
      + Конструкторы: по умолчанию, с параметрами, копирования
      + Деструктор
      + Селекторы и модификаторы для полей
      + Перегрузка оператора присваивания (=)
      + Перегрузка операторов ввода/вывода (>>, <<)
11. **Производный класс Student (Студент)**
    * Наследует Person
    * Дополнительные поля:
      + subject (тип string) – название предмета
      + grade (тип int) – оценка
    * Методы:
      + Конструкторы (аналогично Person)
      + Метод bad\_mark() – выводит сообщение при неудовлетворительной оценке
      + Перегрузка операторов ввода/вывода

**Код программы**

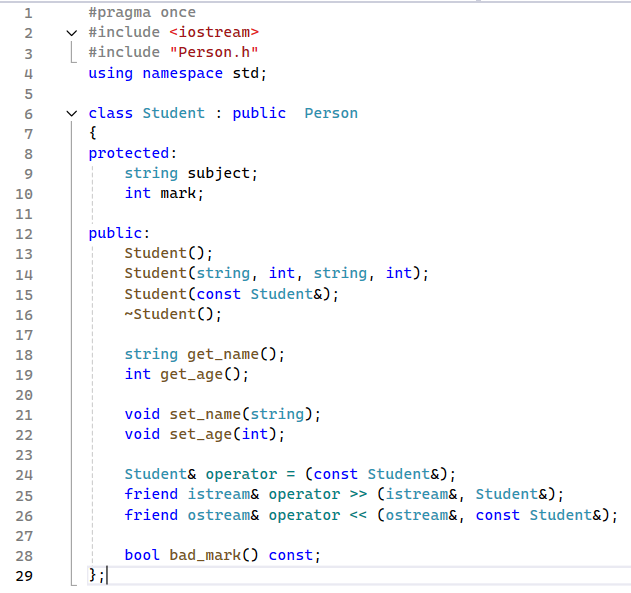
**Person.h**



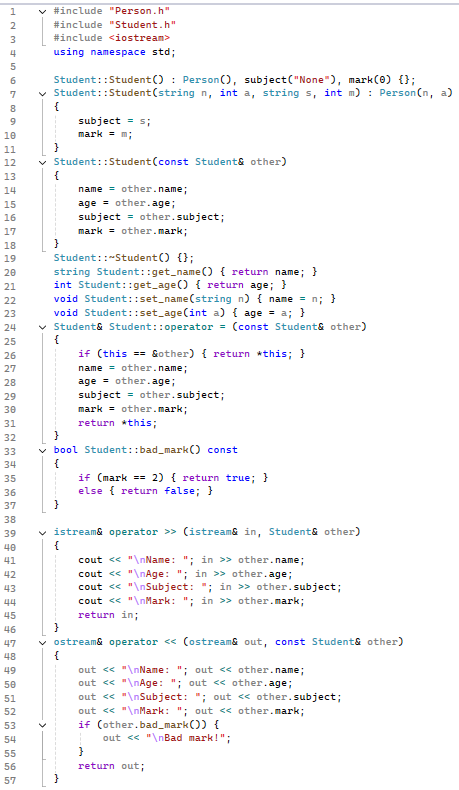
**Person.cpp**



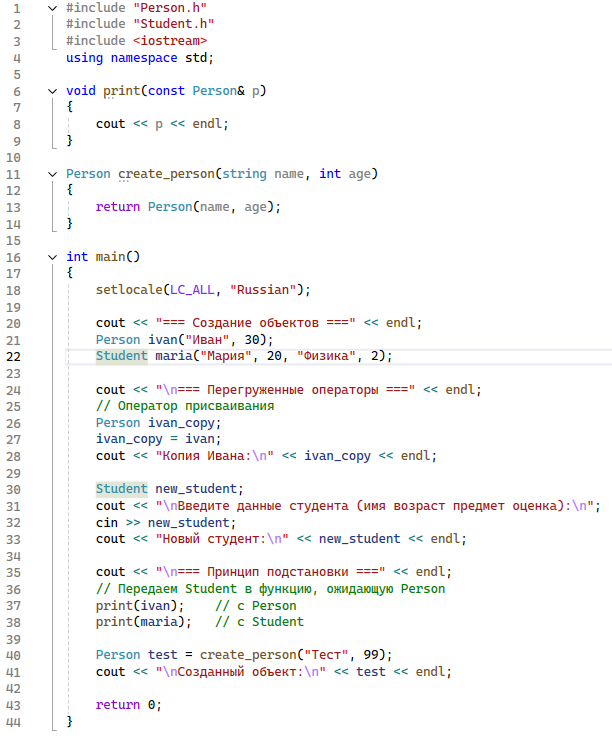
**Student.h**



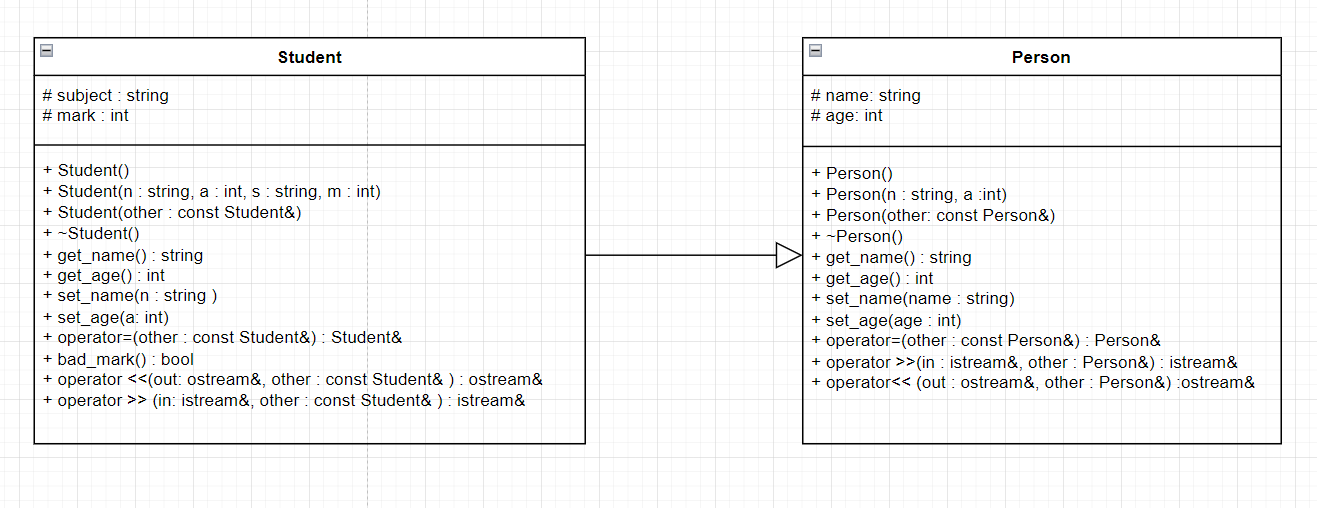
**Student.cpp**



**main.cpp**



**Диаграмма классов.**



**Контрольные вопросы**

**1. Для чего используется механизм наследования?**  
Ответ: Для создания иерархии классов и повторного использования кода.

**2. Каким образом наследуются компоненты класса, описанные со спецификатором public?**  
Ответ: Остаются public в производном классе.

**3. Каким образом наследуются компоненты класса, описанные со спецификатором private?**  
Ответ: Не наследуются и недоступны в производном классе.

**4. Каким образом наследуются компоненты класса, описанные со спецификатором protected?**  
Ответ: Остаются protected в производном классе.

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

class Производный : public Базовый { ... };

**6. Наследуются ли конструкторы?**  
Ответ: Нет, конструкторы не наследуются.

**7. Наследуются ли деструкторы?**  
Ответ: Нет, но деструктор базового класса автоматически вызывается.

**8. В каком порядке конструируются объекты производных классов?**  
Ответ: Сначала базовый класс, затем производный.

**9. В каком порядке уничтожаются объекты производных классов?**  
Ответ: Сначала производный класс, затем базовый.

**10. Что представляют собой виртуальные функции и механизм позднего связывания?**  
Ответ: Виртуальные функции позволяют переопределять методы, позднее связывание - выбор версии функции во время выполнения.

**11. Могут ли быть виртуальными конструкторы? Деструкторы?**  
Ответ: Конструкторы - нет, деструкторы - да (рекомендуется делать виртуальными).

**12. Наследуется ли спецификатор virtual?**  
Ответ: Да, если функция объявлена virtual в базовом классе.

**13. Какое отношение устанавливает открытое наследование?**  
Ответ: Отношение "является".

**14. Какое отношение устанавливает закрытое наследование?**  
Ответ: Отношение "реализовано посредством".

**15. В чем заключается принцип подстановки?**  
Ответ: Объекты производного класса могут использоваться везде, где ожидаются объекты базового класса.

**16. Какие компонентные данные будет иметь объект x класса Teacher?**

**class Student {**

**int age**

**public:**

**string name;**

**};**

**class Employee : public Student {**

**protected:**

**string post; // protected поле**

**// ...**

**};**

**class Teacher : public Employee {**

**protected:**

**int stage; // protected поле**

**// ...**

**};**

**Teacher x;**

**Ответ:**

**int age; // Унаследовано от Student (private, но недоступно напрямую)**

**string name; // Унаследовано от Student (public)**

**string post; // Унаследовано от Employee (protected)**

**int stage; // Собственное поле Teacher (protected)**

**17. Конструкторы без параметров для классов Student, Employee и Teacher**

// Student

Student::Student() : age(0), name("") {}

// Employee

Employee::Employee() : Student(), post("") {}

// Teacher

Teacher::Teacher() : Employee(), stage(0) {}

**18. Конструкторы с параметрами для классов Student, Employee и Teacher**

// Student

Student::Student(int a, string n) : age(a), name(n) {}

// Employee

Employee::Employee(int a, string n, string p) : Student(a, n), post(p) {}

// Teacher

Teacher::Teacher(int a, string n, string p, int s) : Employee(a, n, p), stage(s) {}

**19. Конструкторы копирования для классов Student, Employee и Teacher**

// Student

Student::Student(const Student& other) : age(other.age), name(other.name) {}

// Employee

Employee::Employee(const Employee& other) : Student(other), post(other.post) {}

// Teacher

Teacher::Teacher(const Teacher& other) : Employee(other), stage(other.stage) {}

**20. Операции присваивания для классов Student, Employee и Teacher**

// Student

Student& Student::operator=(const Student& other) {

if (this != &other) {

age = other.age;

name = other.name;

}

return \*this;

}

// Employee

Employee& Employee::operator=(const Employee& other) {

if (this != &other) {

Student::operator=(other);

post = other.post;

}

return \*this;

}

// Teacher

Teacher& Teacher::operator=(const Teacher& other) {

if (this != &other) {

Employee::operator=(other);

stage = other.stage;

}

return \*this;

}