Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский

политехнический университет»

Лабораторная работа №4

«Простое наследование. Принцип подстановки»

Выполнил:

студент первого курса

ЭТФ группы РИС-23-3б

Коротаев Александр Дмитриевич

Проверила:

Доцент кафедры ИТАС О. А. Полякова

Пермь, 2024

**Цель задания**

1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Создание иерархии классов с использованием простого наследования.
3. Изучение принципа подстановки.

**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.

1. Определить производный класс.
2. Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.

9. Реализовать функции, получающие и возвращающие объект базового класса. Продемонстрировать принцип подстановки.

**Задание**

Базовый класс:

ЧЕЛОВЕК (PERSON)

Имя (name) – string Возраст (age) – int

Определить методы изменения полей.

Создать производный класс STUDENT, имеющий поля Предмет – string и Оценка – int. Определить методы изменения полей и метод, выдающий сообщение о неудовлетворительной оценке.

**Анализ задачи**

1) Использование директивы #pragma once предотвращает повторную загрузку заголовочного файла, что может улучшить производительность программы, особенно при использовании множества включаемых файлов.

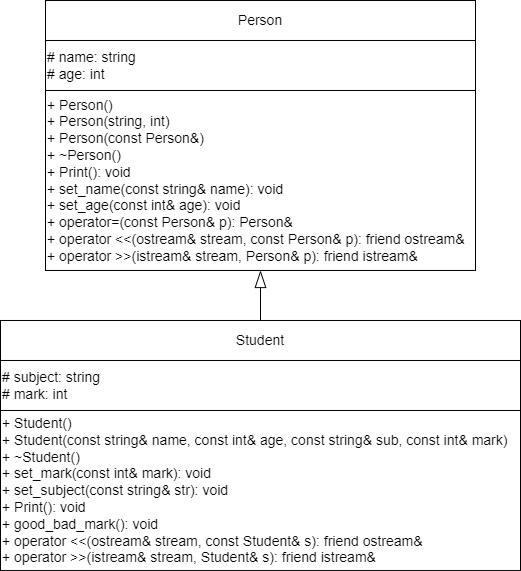
1. Необходимо реализовать класс Person и дочерний для него класс
   1. В классах надо прописать конструкторы: по умолчанию, с параметрами и копирования; а также деструктор, для предотвращения утечки памяти.
   2. Класс Person содержит защищенные данные name и age, что предотвращает их изменение извне класса. Также определены конструктор по умолчанию, конструктор с параметрами, конструктор копирования и деструктор. Предоставлены функции для установки значений и вывода информации о персоне. Определён оператор присваивания и дружественные функции для работы с потоками ввода/вывода.
   3. Класс Student является производным от Person. Он добавляет свои собственные защищенные данные subject и mark, а также предоставляет дополнительные функции для установки этих значений и вывода информации

* студенте. Определены конструктор с параметрами, функции для установки предмета и оценки, а также дружественные функции для работы с потоками ввода/вывода.
  1. Функция mark\_status() в классе Student предназначена для вывода сообщения о низкой оценке.
  2. Перегруженные операторы +, -, = и << позволяют выполнять различные операции с объектами класса. Например, оператор + позволяет

складывать целое число с первым полем объекта класса и вещественное число со вторым полем соответственно. Оператор << позволяет выводить объекты класса в поток вывода.

* 1. Методы set\_name() и set\_age() используются для установки имени
* возраста соответственно.
  1. Метод set\_mark() и set\_subject() используются для установки предмета и оценки соответственно.
  2. В главной функции показаны операции, которые можно выполнить с объектом классов

**Uml диаграмма**



**Код**

Файл ООП4.cpp

#include <iostream>

#include <string>

using namespace std;

#include "Person.h"

int main()

{

system("chcp 1251>null");

Person person1("Ктото Ктотович Ктотович", 5);

cout << person1;

Person person2;

cout << person2;

Person person3(person1);

cout << person3;

Student student1("Ктототамов Ктототам", 5, "math", 2);

cout << student1 << endl;

student1.mark\_status();

Student student2;

cout << student2 << endl << endl;

cin >> student2;

cout << student2 << endl;

student2.mark\_status();

Student student3(student1);

cout << student3 << endl;

student3.mark\_status();

return 0;

}

Файл Person.h

#pragma once

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:

string subject;

int mark;

public:

Student();

Student(string, int, string, int);

void setSubject(string);

void setMark(int);

void mark\_status();

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

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

};

Файл Person.cpp

#pragma once

#include <iostream>

#include <string>

using namespace std;

#include "Person.h"

Person::Person()

{

cout << "Вызвался конструктор без параметров для объекта " << this << endl;

this->name = "никто";

this->age = 0;

}

Person::Person(string name, int age)

{

cout << "Вызвался конструктор с параметрами для объекта " << this << endl;

this->name = name;

this->age = age;

}

Person::Person(const Person& other)

{

cout << "Вызвался конструктор копирования для объекта " << this << endl;

this->name = other.name;

this->age = other.age;

}

Person::~Person()

{

cout << "Вызвался деструктор для объекта " << 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 << "ФИО: " << person.name << endl << "Возраст: " << person.age << "\n\n";

return stream;

}

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

{

string name; int age;

cout << "Введите имя: ";

cin.seekg(cin.eof());

getline(cin, name);

cout << "Введите возраст: ";

cin >> age;

person.setName(name);

person.setAge(age);

return stream;

}

Student::Student()

{

this->name = "Никто Никтотович Никтотович";

this->age = 0;

this->subject = "Обед";

this->mark = 5;

}

Student::Student(string name, int age, string subject, int mark)

{

this->name = name;

this->age = age;

this->subject = subject;

this->mark = mark;

}

void Student::setSubject(string subject)

{

this->subject = subject;

}

void Student::setMark(int mark)

{

this->mark = mark;

}

void Student::mark\_status()

{

if (this->mark < 3)

{

cout << "Ты отчислен =(\n\n";

}

else

{

cout << "Все хорошо, пока учись =)\n\n";

}

}

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

{

stream << "Имя: " << student.name << endl << "Возраст: " << student.age << endl;

stream << "Предмет: " << student.subject << endl << "Оценка: " << student.mark;

return stream;

}

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

{

string subject; int mark;

cout << "Введите предмет: ";

cin.seekg(cin.eof());

getline(cin, subject);

cout << "Введите оценку: ";

stream >> mark;

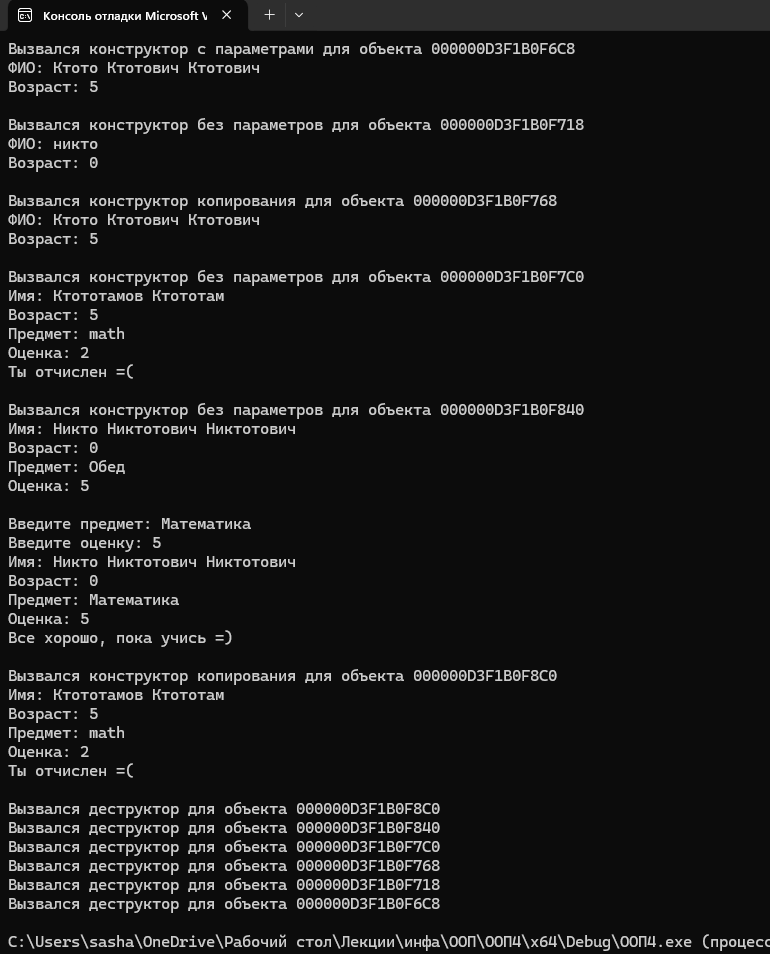
student.setSubject(subject);

student.setMark(mark);

return stream;

}

**Результат работы**



**GitHub**

<https://github.com/Korovay4ik/Laboratory-works>

**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.

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;**

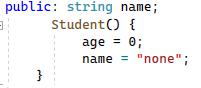
**};**

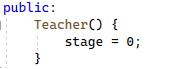
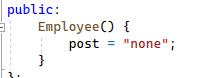
**Teacher x;**

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

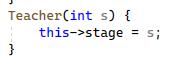
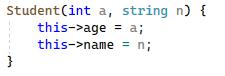
Public name (часть интерфейса) и protected post и stage.

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

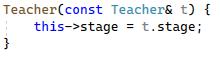
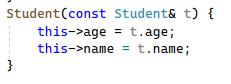




1. **Для классов Student, Employee и Teacher написать конструкторы с параметрами.**



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



1. **Для классов Student, Employee и Teacher определить операцию присваивания.**

