МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра информационные технологии и автоматизированные системы

**Дисциплина Информатика**

**Наследование**

Выполнил студент ИВТ-22-2б:

Мифтахов Марат Ринатович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Пермь, 2023

**Постановка задачи**

# Определить пользовательский класс.

# Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.

# Определить в классе деструктор.

# Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).

# Перегрузить операцию присваивания.

# Перегрузить операции ввода и вывода объектов с помощью потоков.

# Определить производный класс.

# Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.

# Реализовать функции, получающие и возвращающие объект базового класса.

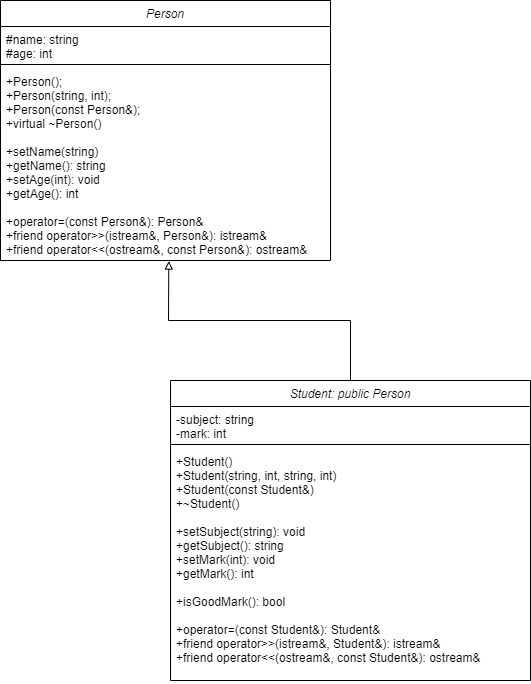
# Продемонстрировать принцип подстановки.

**Задание**

1. Класс Person
2. Поля имя — string, возраст — int
3. Определить геттеры и сеттеры
4. Создать производный класс Student
5. Поля предмет — string, возраст — int

**Диаграмма классов**

Ниже представлена UML-диаграмма классов.



**Программный код**

Файл Student.cpp:

#include "Person.h"

#include "Student.h"

///--------------------Student----------------///

Student::Student() : Person::Person() {

subject = "";

mark = 0;

}

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

this->subject = subject;

this->mark = mark;

}

Student::Student(const Student& p) {

name = p.name;

age = p.age;

subject = p.subject;

mark = p.mark;

}

Student::~Student() {}

void Student::setSubject(string subject) {

this->subject = subject;

}

string Student::getSubject() {

return subject;

}

void Student::setMark(int mark) {

this->mark = mark;

}

int Student::getMark() {

return mark;

}

bool Student::isGoodMark() {

if (mark > 3) return true;

else return false;

}

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

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

this->name = ptr.name;

this->age = ptr.age;

this->subject = ptr.subject;

this->mark = ptr.mark;

return \*this;

}

istream& operator>>(istream& in, Student& ptr) {

cout << "name:? " << endl; in >> ptr.name;

cout << "age:?" << endl; in >> ptr.age;

cout << "subject:?" << endl; in >> ptr.subject;

cout << "mark:?" << endl; in >> ptr.mark;

return in;

}

ostream& operator<<(ostream& out, const Student& p) {

return out << "name: " << p.name << " age: " << p.age << "\n" <<

"subject: " << p.subject << " mark: " << p.mark << endl;

}

///--------------------Student----------------///

Файл Person.cpp:

#include "Person.h"

#include "Student.h"

///--------------------Person----------------///

Person::Person() {

name = "";

age = 0;

}

Person::Person(string name, int age) {

this->name = name;

this->age = age;

}

Person::Person(const Person& p) {

name = p.name;

age = p.age;

}

Person::~Person() {}

void Person::setName(string name) {

this->name = name;

}

string Person::getName() {

return name;

}

void Person::setAge(int age) {

this->age = age;

}

int Person::getAge() {

return age;

}

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

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

name = p.name;

age = p.age;

return \*this;

}

ostream& operator<<(ostream& out, const Person& p) {

return out << "name: " << p.name << " age: " << p.age;

}

istream& operator>>(istream& in, Person& p) {

cout << "name:? " << endl; in >> p.name;

cout << "age:?" << endl; in >> p.age;

return in;

}

///--------------------Person----------------///

Файл Student.h:

#pragma once

#include "Person.h"

class Student : public Person {

string subject;

int mark;

public:

Student();

Student(string, int, string, int);

Student(const Student&);

~Student();

void setSubject(string);

string getSubject();

void setMark(int);

int getMark();

bool isGoodMark();

Student& operator=(const Student&);

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

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

};

Файл 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&);

virtual ~Person(); //объявление виртуальным для гарантии выполнения деструктора производного класса

void setName(string);

string getName();

void setAge(int);

int getAge();

Person& operator=(const Person&);

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

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

};

Файл main.cpp:

#include <iostream>

#include "Person.h"

#include "Student.h"

void interestingFunction(Person&);

Student secondInterestingFunction();

int main() {

Student a;

cin >> a;

cout << a;

Person a2;

cin >> a2;

cout << a2;

cout << endl;

interestingFunction(a);

a = secondInterestingFunction();

cout << endl;

cout << a;

return 0;

}

void interestingFunction(Person& person) {

person.setName("Torwalds");

cout << person;

}

Student secondInterestingFunction() {

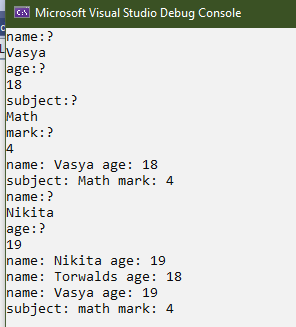
Student student("Vasya", 19, "math", 4);

return student;

}

**Вывод программы**

Ниже представлен вывод программы на консоль.

**\_**

**Контрольные вопросы**

1. Для чего используется механизм наследования?

Ответ: для создания производных классов на основе данных. Используется для расширения функционала базового класса, а также для избежания дублирования кода.

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

Ответ: Если указывается public при наследовании, то public, если private – private, если protected - protected

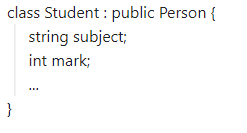
1. Каким образом наследуются компоненты класса, описанные со спецификатором private?

Ответ: Никаким образом

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

Ответ: Если указывается public при наследовании, то protected, если private – private, если protected - protected

1. Каким образом описывается производный класс?

Ответ: имя класса: спецификатор доступа, имя базового класса {…}

1. Наследуются ли конструкторы?

Ответ: нет, но вызываются при создании экземпляра производного класса

1. Наследуются ли деструкторы?

Ответ: нет, но вызываются при создании экземпляра производного класса

1. В каком порядке конструируются объекты производных классов?

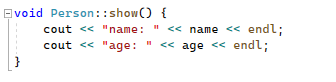
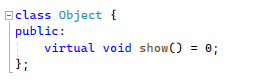
Ответ: сначала конструируется базовый класс, затем производный класс.

1. В каком порядке уничтожаются объекты производных классов?

Ответ: в обратном порядке: от производных к базовомму

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

Ответ: Виртуальные функции — функции, которые можно переопределять в производных классах. Механизм позднего связывания представляет собой формирование кода на этапе выполнения. При этом вызов метода происходит на основании типа объекта, а не на основании ссылки на базовый класс.



1. Могут ли быть виртуальными конструкторы? Деструкторы?

Ответ: конструктор виртуальным быть не может. Деструктор виртуальным быть может, так это гарантирует вызов деструктора базового класса.

1. Наследуется ли спецификатор virtual?

Ответ: Да

1. Какое отношение устанавливает между классами открытое наследование?

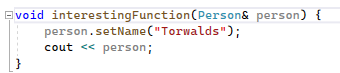
Ответ: ЯВЛЯЕТСЯ

1. Какое отношение устанавливает между классами закрытое наследование?

Ответ: РЕАЛИЗУЕТ

1. В чем заключается принцип подстановки?

Ответ: в использовании объектов производных классов по ссылке на объект базового класса

****

16. Имеется иерархия классов:

class Student

{

int age;

public:

string name;

};

class Employee : public Student

{

protected:

string post;

};

class Teacher : public Employee

{

protected: int stage;

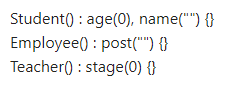
};

Teacher x;

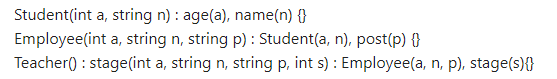
Какие компонентные данные будет иметь объект х?

Ответ: объект x будет иметь компоненты данных age, name, post и stage.

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

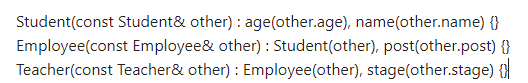
\_

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

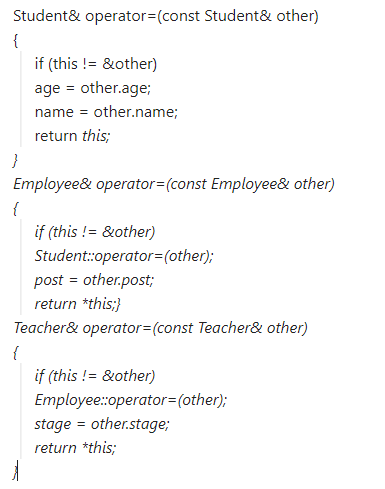


\_

19. Для классов Student, Employee и Teacher написать конструкторы копирования.



20. Для классов Student, Employee и Teacher определить операцию присваивания.

****