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

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**о работе по информатике**

Семестр: 2

На тему: «ООП. Простое наследование. Принцип подстановки»

Выполнил студент ИВТ-22-2б:

Казанцев Антон Васильевич

(дата, подпись)

Проверила:

Полякова Ольга Андреевна

(дата, подпись)

Пермь 2023

**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.

7. Определить производный класс.

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

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

*Вариант 15:*

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**

**Класс Person:**

1. Person() – конструктор без параметров

2. Person(string name, int age) – конструктор с параметрами

3. Person(const Person& copied) – конструктор копирования

4. istream& operator>>(istream& in, Person& p) – перегрузка оператора ввода

5. ostream& operator<<(ostream& out, const Person& p) – перегрузка оператора вывода

6. void setName(string name) – сеттер для поля имени

7. void setAge(int age) – сеттер для поля возраст

8. string getName() – геттер для поля имя

9. int getAge() – геттер для поля возраст

10. Person& operator=(const Person& m\_person) - перегрузка оператора присваивания

**Класс Student:**

1. Student() – конструктор без параметров

2. Student(string, int, string, int) – конструктор с параметрами

3. Student(const Student&)- конструктор копирования

4. void setSubject(string) – сеттер для поля предмет

5. void setMark(int) – сеттер для поля оценка

6. string getSubject() – геттер для поля предмет

7. int getMark() – геттер для поля оценка

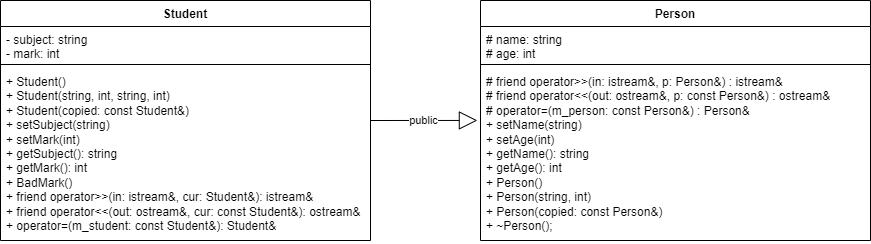
8. void BadMark() – метод выдающий сообщение о неудовлетворяющей оценке

9. friend istream& operator>>(istream&, Student&) – перегрузка оператора ввода

10. friend ostream& operator<<(ostream&, const Student&) – перегрузка оператора вывода

11. Student& operator=(const Student&) – перегрузка оператора присваивания

**UML-Диаграмма**

****

**Программное решение**

***Person.h***

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Person {//Базовый класс

protected:

string name;

int age;

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

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

Person& operator=(const Person&);

public:

void setName(string);

void setAge(int);

string getName();

int getAge();

Person();

Person(string, int);

Person(const Person&);

~Person() {};

};

***Person.cpp***

#include "Person.h"

Person::Person() {

this->name = "";

this->age = 0;

}

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

this->name = name;

this->age = age;

}

Person::Person(const Person& copied) {

age = copied.age;

name = copied.name;

}

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

cout << "Введите данные в формате:\nимя возвраст\n";

return (in >> p.name >> p.age);

}

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

return (out << p.name << ' ' << p.age << '\n');

}

void Person::setName(string name) {

this->name = name;

}

void Person::setAge(int age) {

this->age = age;

}

string Person::getName() {

return name;

}

int Person::getAge() {

return age;

}

Person& Person::operator=(const Person& m\_person) {//перегрузка оператора присваивания

if (&m\_person != this) {

name = m\_person.name;

age = m\_person.age;

}

return \*this;

}

***Student.h***

#pragma once

#include <iostream>

#include "Person.h"

using namespace std;

class Student : public Person {

private:

string subject;

int mark;

public:

Student();

Student(string, int, string, int);

Student(const Student&);

void setSubject(string);

void setMark(int);

string getSubject();

int getMark();

void BadMark();

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

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

Student& operator=(const Student&);

};

***Student.cpp***

#include "Student.h"

Student::Student() {

name = "";

subject = "";

age = 0;

mark = 0;

}

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

this->name = name;

this->age = age;

this->subject = subject;

this->mark = mark;

}

Student::Student(const Student& copied) {

name = copied.name;

age = copied.age;

subject = copied.subject;

mark = copied.mark;

}

void Student::setSubject(string subject) {

this->subject = subject;

}

void Student::setMark(int mark) {

this->mark = mark;

}

string Student::getSubject() {

return subject;

}

int Student::getMark() {

return mark;

}

void Student::BadMark() {

if (mark < 4) {

cout << this->name << " имеет плохую оценку\n";

}

}

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

cout << "Введите данные в формате:\nимя возвраст предмет оценка\n";

return (in >> cur.name >> cur.age >> cur.subject >> cur.mark);

}

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

return (out << cur.name << ' ' << cur.age << ' ' << cur.subject << ' ' << cur.mark << '\n');

}

Student& Student::operator=(const Student& m\_student) {//перегрузка оператора присваивания

if (&m\_student != this) {

name = m\_student.name;

age = m\_student.age;

subject = m\_student.subject;

mark = m\_student.mark;

}

return \*this;

}

***LabaOOP4.cpp***

#include "Person.h"

#include "Student.h"

void getObject(Person& object) {//функция принимающая объект

object.setName("Дима");

object.setAge(15);

}

Person ReturnObject() {//функция возвращающая объект

Student s("Антон", 21, "Информатика", 5);

return s;

}

int main() {

system("chcp 1251 > NULL");

Person p1("Миша", 3);//Миша ещё не ученик

cout << p1;//выводим значения

cin >> p1;//вводим новые знаения

cout << p1;//проверяем значения

Student s1("Олег", 23, "inf", 4);//У Олега, как и у любого человека есть имя и возвраст, а как у ученика - предмет и оценка

Student s2;

getObject(s2);//установка значений через функцию

cout << s2;//проверка значений

Person p2 = ReturnObject();//создание объекта через функцию

cout << p2;//проверка значений

Student s3;

cin >> s3;//считываем значений для s3

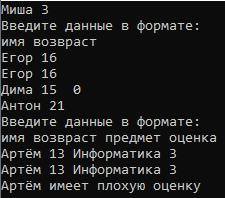
s2 = s3;//заменяем значения s2 соответствующими значениями s3

cout << s2;//проверяем значение

s2.BadMark();//проверяем оценку (выведет сообщение, если плохая)

return 0;

}

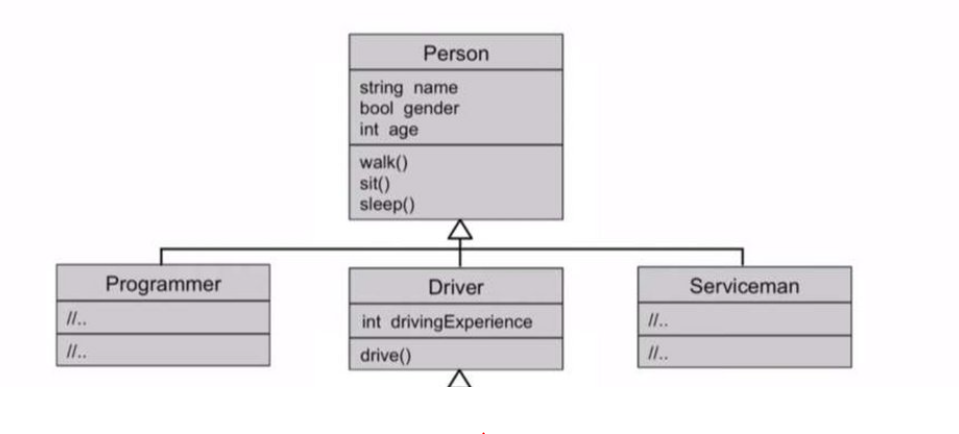


**Контрольные вопросы**

**Контрольные вопросы**

*1. Для чего используется механизм наследования?*

**Наследование** − это механизм получения нового класса на основе уже существующего. Существующий класс может быть дополнен или изменен для создания нового класса. Наследование позволяет структурировать и повторно использовать код и ускорить процесс разработки. Оно позволяет выделить общее поведение для нескольких классов (**производных**) и вынести его в отдельную сущность - **базовый** класс. С помощью наследования может быть создана иерархия классов, которые совместно используют код и интерфейсы.

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

**public** – член класса может использоваться любой функцией, которая является членом данного или производного класса, а также к public - членам возможен доступ извне через имя объекта.

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

**private** – член класса может использоваться только функциями – членами данного класса и функциями – “друзьями” своего класса. В производном классе он недоступен.

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

**protected** – то же, что и private, но дополнительно член класса с данным атрибутом доступа может использоваться функциями-членами и функциями – “друзьями” классов, производных от данного.

*5. Каким образом описывается производный класс?*

*Синтаксис определения производного класса:*

class имя\_класса : тип наследования список\_базовых\_классов

{список\_компонентов\_класса};

*Пример:*

class Person //Базовый класс

{

protected:

string name;

int age;

public:

Person();

Person(string, int);

Person(const Person&);

~Person();

};

class Student : public Person //Класс Student наследуется от класса Person

{

protected:

string subject;

int mark;

public:

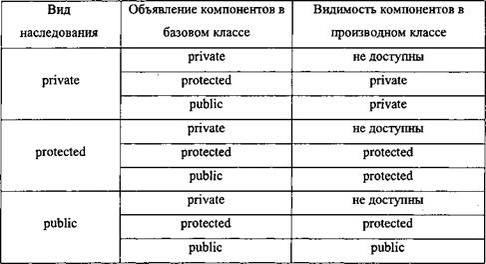
Student();

Student(string, int, string, int);

Student(const Student&);

~Student();

};



*6. Наследуются ли конструкторы?*

Конструкторы **не наследуются**. При создании производного класса наследуемые им данные-члены должны инициализироваться конструктором базового класса.

*7. Наследуются ли деструкторы?*

Деструкторы **не наследуются**. Однако они вызываются, когда объект производного класса выходит из области видимости.

*8. 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.

*9. В каком порядке уничтожаются объекты производных классов?*Уничтожаются объекты в обратном порядке: сначала производный, потом его компоненты-объекты, а потом базовый объект. Порядок уничтожения объекта противоположен по отношению к порядку его конструирования.

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

К механизму виртуальных функций обращаются в тех случаях, когда в каждом производном классе требуется свой вариант некоторой компонентной функции. Интерпретация каждого вызова виртуальной функции через указатель на базовый класс зависит от значения этого указателя, т.е. от типа объекта, для которого выполняется вызов.

*Пример:*

class Base {

public:

virtual void print() { cout << ”\nBase”; } //Виртуальная функция

. . .

};

class Derive : public Base

{

public:

void print() { cout << ”\n Derive”; } //Переопределение виртуальной функции для конкретного класса

};

int main()

{

Base B, \* bp;

Derive D, \* dp;

bp = &B;

dp = &D;

//указатель базового класса ставится на объект производного класса

Base\* p = &D;

bp –> print(); // вызывается метод для Base

dp –> print(); // вызывается метод для Derive

p –> print(); // вызывается метод для Derive

return 0;

}

**Позднее** **связывание** означает, что объект связывается с вызовом функции только во время исполнения программы, а не раньше.

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

Конструкторы не могут быть виртуальными, в отличие от деструкторов. Практически каждый класс, имеющий виртуальную функцию, должен иметь виртуальный деструктор.

*12. Наследуется ли спецификатор virtual?*

Виртуальность наследуется. После того как функция определена как виртуальная, ее повторное определение в производном классе (с тем же самым прототипом) создает в этом классе новую виртуальную функцию, причем спецификатор virtual может не использоваться.

*Пример:*

class Base {

public:

virtual void print() { cout << ”\nBase”; } //Виртуальная функция

. . .

};

class Derive : public Base

{

public:

void print() { cout << ”\n Derive”; } //Переопределение виртуальной функции для конкретного класса

};

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

Открытое наследование устанавливает между классами отношение «является»: класс-наследник является частью класса-родителя. Это означает, что везде, где может быть использован объект базового класса (при присваивании, при передаче параметров и возврате результата), вместо него разрешается использовать объект производного класса. Таким образом, порожденный класс представляет собой модификацию базового класса.

class A

{

protected:

int i;

};

class B : public A

{ ... }; //i остается protected членом B

class C : public B

{

public:

void f();

};

//поскольку i является protected для B, оно может наследоваться в C

void C::f() { i = 2; } //данная функция работает т.к. i доступна в C

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

Закрытое наследование – это наследование реализации, в этом случае принцип подстановки не соблюдается. Закрытое наследование означает, что от базового класса необходимо взять какую-то функциональность, базовый класс и потомок не имеют какой-либо концептуальной связи .Закрытое наследование не носит характера отношения подтипов. Закрытое (также как и защищенное) наследование не создает иерархии типов.

class A

{

protected:

int i;

};

class B : private A

{ ... }; //i преобразовано к private члену B

class C : public B

{

public:

void f();

};

//поскольку i является private для B, оно не может наследоваться в C (не создается иерархии классов)

void C::f() { i = 2; } //данная функция не работает, т.к. i не доступна в C

*15. В чем заключается принцип подстановки?*

Принцип подстановки: класс S может считаться подклассом T, если замена объектов T на объекты S не приведёт к изменению работы программы.

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; //Объект класса Teacher

*Какие компонентные данные будет иметь объект х?*

В класс Employee открыто наследуется public name (age не наследуется, поскольку поле private). В Teacher открыто наследуется public name (из Student), и открыто наследуется protected post (из Employee). Таким образом, у класса Teacher доступны следующие поля: public name, protected post, protected stage.

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

class Student

{

int age;

public:

string name;

public:

Student() {

age = 0;

name = "";

}

};

class Employee : public Student {

protected:

string post;

public:

Employee() {

name = "";

post = "";

}

};

class Teacher : public Employee {

protected:

int stage;

public:

Teacher() {

name = "";

post = "";

stage = 0;

}

};

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

class Student

{

int age;

public:

string name;

public:

Student() { age = 0; name = ""; }

Student(int a, string n) {

age = a;

name = n;

}

};

class Employee : public Student {

protected:

string post;

public:

Employee() { name = ""; post = ""; }

Employee(string n, string p) {

name = n;

post = p;

}

};

class Teacher : public Employee {

protected:

int stage;

public:

Teacher() { name = ""; post = ""; stage = 0; }

Teacher(string n, string p, int s) {

name = n;

post = p;

stage = s;

}

};

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

class Student

{

int age;

public:

string name;

public:

Student() { age = 0; name = ""; }

Student(const Student& s) {

age = s.age;

name = s.name;

}

};

class Employee : public Student {

protected:

string post;

public:

Employee() { name = ""; post = ""; }

Employee(const Employee& e) {

name = e.name;

post = e.post;

}

};

class Teacher : public Employee {

protected:

int stage;

public:

Teacher() { name = ""; post = ""; stage = 0; }

Teacher(const Teacher& t) {

name = t.name;

post = t.post;

stage = t.stage;

}

};

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

class Student

{

int age;

public:

string name;

public:

Student() { age = 0; name = ""; }

Student& operator=(const Student& s) {

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

age = s.age;

name = s.name;

return \*this;

}

};

class Employee : public Student

{

protected:

string post;

public:

Employee() { name = ""; post = ""; }

Employee& operator=(const Employee& e) {

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

name = e.name;

post = e.post;

return \*this;

}

};

class Teacher : public Employee

{

protected:

int stage;

public:

Teacher() { name = ""; post = ""; stage = 0; }

Teacher& operator=(const Teacher& t) {

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

name = t.name;

post = t.post;

stage = t.stage;

return \*this;

}

};