Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по лабораторной работе №4

Тема: «Простое наследование. Принцип подстановки.»

Выполнил

Студент группы РИС-22-2б

Худеньких В.Д.

Проверил доц. Кафедры ИТАС

Полякова О.А.

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.

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 int. Определить методы изменения полей и метод, выдающий сообщение о неудовлетворительной оценке.

**Контрольные вопросы**

1. Для чего используется механизм наследования?

Механизм наследования используется для создания новых классов на основе уже существующих, чтобы избежать дублирования кода и повторного описания свойств и методов базового класса в производных классах.

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

Компоненты класса, описанные со спецификатором public, наследуются открыто и доступны для использования в производном классе.

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

Компоненты класса, описанные со спецификатором private, не наследуются и не доступны для использования в производном классе.

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

Компоненты класса, описанные со спецификатором protected, наследуются защищенно и доступны для использования в производном классе и его потомках.

5. Каким образом описывается производный класс?

Производный класс описывается с помощью ключевого слова class, за которым следует имя нового класса и двоеточие, а затем указывается имя базового класса, от которого производится наследование (например, class DerivedClass : public BaseClass).

6. Наследуются ли конструкторы?

Конструкторы наследуются, но не переопределяются.

7. Наследуются ли деструкторы?

Деструкторы наследуются, но не переопределяются.

8. В каком порядке конструируются объекты производных классов?

Сначала конструируется базовый класс, затем производный класс.

9. В каком порядке уничтожаются объекты производных классов?

Сначала уничтожается производный класс, затем базовый класс.

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

Виртуальные функции позволяют вызывать методы производного класса через указатель или ссылку на базовый класс. Механизм позднего связывания обеспечивает выбор правильного метода в зависимости от типа объекта, на который указывает указатель или ссылка. Пример:

11. Могут ли быть виртуальными конструкторы? Деструкторы?

Конструкторы не могут быть виртуальными, деструкторы могут быть виртуальными. Пример:

12. Наследуется ли спецификатор virtual?

Спецификатор virtual не наследуется.

13. Какое отношение устанавливает между классами открытое наследование?

Открытое наследование устанавливает отношение "является" между базовым и производным классами, т.е. производный класс является расширением базового класса. Например:

14. Какое отношение устанавливает между классами закрытое наследование?

Закрытое наследование устанавливает отношение "реализует" между базовым и производным классами, т.е. производный класс реализует интерфейс базового класса, но не является его расширением. Например:

15. В чем заключается принцип подстановки?

Принцип подстановки гласит, что объекты производного класса могут использоваться везде, где ожидается объект базового класса, не нарушая при этом корректности программы. Например:

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?

Объект x будет иметь компоненты данных age, name, post и stage.

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

Конструкторы без параметров для классов Student, Employee и Teacher могут выглядеть так:

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

Конструкторы с параметрами для классов Student, Employee и Teacher могут выглядеть так:

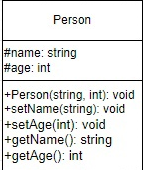
19. Для классов Student, Employee и Teacher написать конструкторы копирования.

Конструкторы копирования для классов Student, Employee и Teacher могут выглядеть так:

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

Операция присваивания для классов Student, Employee и Teacher может быть определена так:

UML – таблица



Код программы

#pragma once

#include "Person.h"

class Student : public Person

{

string subject;

int mark;

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

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

public:

Student();

Student(string, int, string, int);

~Student();

Student(const Student& s);

void print()

{

cout << "Student" << endl;

}

};

#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()

{

cout << "Student" << endl;

}

Student::Student(const Student& s)

{

name = s.name;

age = s.age;

subject = s.subject;

mark = s.mark;

}

istream& operator>>(istream& in, Student& s)

{

cout << "(string)name: ";

in >> s.name;

cout << "(int)age: ";

in >> s.age;

cout << "(string)subject: ";

in >> s.subject;

cout << "(int)mark: ";

in >> s.mark;

return in;

}

ostream& operator<<(ostream& out, const Student& s)

{

return (out << s.name << " : " << s.age << " | " << s.subject << " : " << s.mark);

}

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Person

{

protected:

string name;

int age;

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

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

public:

void set\_name(string name);

void set\_age(int age);

string get\_name();

int get\_age();

Person();

Person(string name, int age);

Person(const Person& p);

~Person();

virtual void print()

{

cout << "Person" << endl;

}

};

#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& p)

{

age = p.age;

name = p.name;

}

Person::~Person()

{

cout << "Person!" << endl;

}

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

{

cout << "(string)name: ";

in >> p.name;

cout << "(int)age: ";

in >> p.age;

return in;

}

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

{

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

}

void Person::set\_name(string name)

{

this->name = name;

}

void Person::set\_age(int age)

{

this->age = age;

}

string Person::get\_name()

{

return name;

}

int Person::get\_age() {

return age;

}

#include "Person.h"

#include "Student.h"

void f1(Person& p)

{

p.set\_age(0);

}

Student f2()

{

Student s("????", 21, "???????????", 5);

return s;

}

int main()

{

setlocale(LC\_ALL, ".1251");

cout << "??????? ??? ??????? ?????? Person, ???? ??? ??????????, ?????? ?: " << endl;

Person p1;

Person p2("?????", 18);

cout << p1 << endl << p2 << endl;

cout << "?????????? ???????????: p1 = p2" << endl;

p1 = p2;

cout << p1 << endl;

cout << "???????? ??????? ?????? Student, ???? ? ???????????, ?????? ???: " << endl;

Student s1;

cout << s1 << endl;

Student s2("??????", 17, "??????????", 5);

cout << s2 << endl;

cout << "????????? ????? ??????? ?????? Student" << endl;

cin >> s1;

cout << s1 << endl;

cout << "???????? ??????? ?????? Student ????? ?-?? f1" << endl;

Student s3 = f2();

cout << s3 << endl;

cout << "????????? ???????? ??????? ?????? Person ????? f2" << endl;

f1(p1);

cout << p1.get\_age() << endl;

cout << "??????????? ??????? print() ??? Person\* p3 = &s3: " << endl;

Person\* p3 = &s3;

p3->print();

return 0;

}