ПНИПУ

Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

Лабораторная работа №4.

Простое наследование. Принцип подстановки.

Выполнил студент группы РИС-23-3Б

Буковский Денис Владимирович

Проверила доцент кафедры ИТАС О.А. Полякова

2024

1. 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.

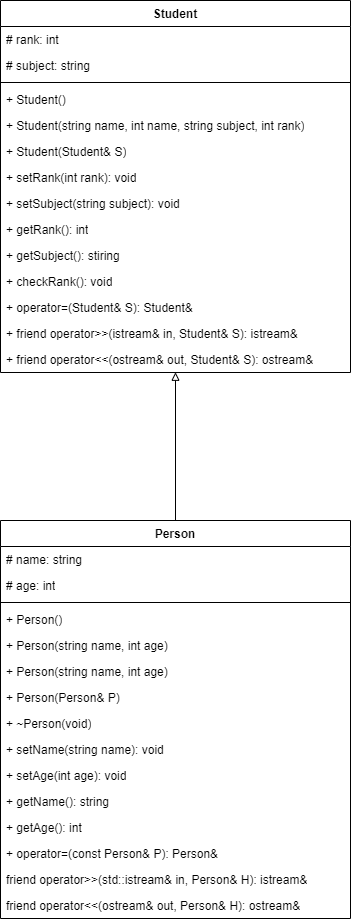
8. Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.

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

1. Анализ задачи

Создадим абстрактный класс Person. В нём определим базовые атрибуты методы класса. Затем создадим класс-наследник Student. Добавим новые методы и атрибуты. Протестируем работу программы в функции main().

1. Блок-схема



1. Код

People.h

#pragma once

#include <iostream>

class Person {

protected:

std::string name;

int age;

public:

Person();

Person(const Person& T);

Person(std::string name, int age);

~Person(void);

void setName(std::string name);

void setAge(int age);

std::string getName();

int getAge();

Person& operator=(const Person& H);

friend std::istream& operator>>(std::istream& in, Person& H);

friend std::ostream& operator<<(std::ostream& out, Person& H);

};

class Student : public Person {

protected:

int rank;

std::string subject;

public:

Student();

Student(Student& S);

Student(std::string name, int age, std::string subject, int rank);

~Student();

void setRank(int rank);

int getRank();

void setSubject(std::string subject);

std::string getSubject();

void checkRank();

Student& operator=(Student& S);

friend std::istream& operator>>(std::istream& in, Student& H);

friend std::ostream& operator<<(std::ostream& out, Student& S);

};

People.cpp

#include "People.h"

#include <iostream>

Person::Person() {

name = "";

age = 0;

std::cout << "Person object was created" << std::endl;

}

Person::Person(const Person& T) {

name = T.name;

age = T.age;

std::cout << "Person object was created" << std::endl;

}

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

this->name = name;

this->age = age;

std::cout << "Person object was created" << std::endl;

}

Person::~Person(void) {

std::cout << "Person object was destroyed" << std::endl;

}

void Person::setName(std::string name) {

this->name = name;

}

void Person::setAge(int age) {

if (age > 0) this->age = age;

}

std::string Person::getName() {

return this->name;

}

int Person::getAge() {

return this->age;

}

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

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

name = H.name;

age = H.age;

return \*this;

}

std::istream& operator>>(std::istream& in, Person& H) {

std::cout << "Enter name: ";

in >> H.name;

std::cout << "Enter age: ";

in >> H.age;

std::cout << std::endl;

return in;

}

std::ostream& operator<<(std::ostream& out, Person& H) {

out << "Name: " << H.name << std::endl << "Age: " << H.age << std::endl;

return out;

}

Student::Student() {

name = "";

age = 0;

subject = "";

rank = 0;

std::cout << "Student object was created" << std::endl;

}

Student::Student(Student& S) {

this->name = S.name;

this->age = S.age;

this->subject = S.subject;

this->rank = S.rank;

std::cout << "Student object was created" << std::endl;

}

Student::Student(std::string name, int age, std::string subject, int rank) {

this->name = name;

this->age = age;

this->subject = subject;

this->rank = rank;

std::cout << "Student object was created" << std::endl;

}

Student::~Student() {

std::cout << "Student object was destroyed" << std::endl;

}

Student& Student::operator=(Student& S) {

this->name = S.name;

this->age = S.age;

this->subject = S.subject;

this->rank = S.rank;

return \*this;

}

void Student::setRank(int rank) {

this->rank = rank;

}

int Student::getRank() {

return this->rank;

}

void Student::setSubject(std::string subject) {

this->subject = subject;

}

std::string Student::getSubject() {

return this->subject;

}

void Student::checkRank() {

switch (rank) {

case 0: {

std::cout << "Нет оценок по предмету " << subject << "!" << std::endl;

break;

}

default:

std::cout << "Оценка по предмету " << subject << " - " << rank << std::endl;

}

}

std::istream& operator>>(std::istream& in, Student& H) {

std::cout << "Enter name: ";

in >> H.name;

std::cout << "Enter age: ";

in >> H.age;

std::cout << "Enter subject: ";

in >> H.subject;

std::cout << "Enter rank: ";

in >> H.rank;

return in;

}

std::ostream& operator<<(std::ostream& out, Student& S) {

out << "Name: " << S.name << std::endl << "Age: " << S.age << std::endl << "Subject: " << S.subject << std::endl << "Rank: " << S.rank << std::endl;

return out;

}

Main.cpp

#include <iostream>

#include "People.h"

using namespace std;

int main() {

Student student1;

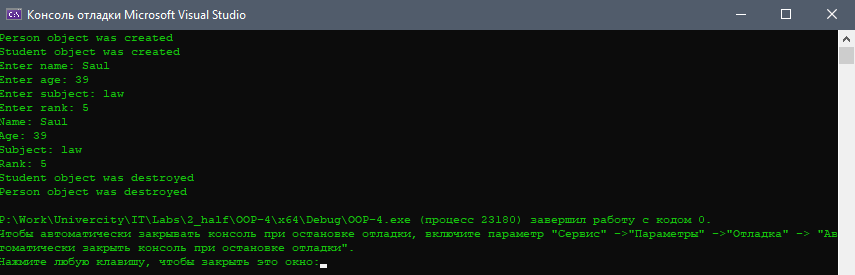
cin >> student1;

cout << student1;

return 0;

}

1. Вывод программы



1. Ответы на контрольные вопросы

1. Наследование используется для расширения функционала класса-родителя с сохранением обратной совместимости.

2. Наследуются.

3. Наследуются.

4. Не наследуются.

5. Производный класс описывается как обычный класс, с тем различием, что после его названия через двоеточие указываются класс(ы)-родитель(и) со спецификаторами доступа.

6. Конструкторы не наследуются.

7. Деструкторы также не наследуются.

8. Сначала родительский класс, затем производный.

9. Сначала производный класс, затем родительский.

10. Виртуальные функции и механизм позднего связывания – это способ поддерживать общий интерфейс, повысить гибкость наследования.

11. Конструкторы не могут быть виртуальными. Виртуальные деструкторы обязательно должны быть в абстрактном классе.

12. Спецификатор virtual наследуется.

13. Открытое наследование позволяет соблюдать принцип подстановки.

14. Закрытое наследование — это наследование реализации, при котором не соблюдается принцип подстановки.

15. Принцип подстановки состоит в том, что любой наследник класса может использоваться вместо его родителя.

16. stage, post, name, age.

17. Student(){ name = “”;}

Employee(){post = ””; name = “”;}

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

18. Student(string name){ name = name;}

Employee(string post, string name){post = post; name=name;}

Teacher(int stage, string post, string name){stage = stage; post = post; name = name;}

19.

Student(Student& S){name = S.name;}

Employee(Employee& E) { name = E.name; post = E.post; }

Teacher(Teacher& T) { name = T.name; post = T.post; stage = T.stage;}

20.

Operator=(Student& S) {name = S.name;}

Operator=(Employee& E) { name = E.name; post = E.post; }

Operator=(Teacher& T) { name = T.name; post = T.post; stage = T.stage;}