Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

направление подготовки: 09.03.01 - «Информатика и вычислительная техника»

**О Т Ч Е Т**

**по лабораторной работе “Классы” №4**

**по дисциплине**

**«Основы алгоритмизации и программирования» семестр 2**

Выполнил студент гр. ИВТ-21-1б

Ипатов Дмитрий Сергеевич

Проверил:

Ст. Преподаватель кафедры ИТАС

Яруллин Д.В.

(оценка) (подпись)

г. Пермь-2022

**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

**Анализ задачи:**

Наследование - это механизм получения нового класса на основе уже существующего. Существующий класс может быть дополнен или изменен для создания нового класса.

Существующие классы называются базовыми, а новые – производными. Производный класс наследует описание базового класса; затем он может быть изменен добавлением новых членов, изменением существующих функций-членов и изменением прав доступа. С помощью наследования может быть создана иерархия классов, которые совместно используют код и интерфейсы.

Наследуемые компоненты не перемещаются в производный класс, а остаются в базовых классах.

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

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.

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.

К механизму виртуальных функций обращаются в тех случаях, когда в каждом производном классе требуется свой вариант некоторой компонентной функции. Классы, включающие такие функции, называются полиморфными и играют особую роль в ООП.

Виртуальные функции предоставляют механизм позднего (отложенного) или динамического связывания. Любая нестатическая функция базового класса может быть сделана виртуальной, для чего используется ключевое слово virtual.

**Код программы:**

#include <iostream>

#include <string>

using namespace std;

class Person

{

protected:

string name;

int age;

public:

Person();

Person(string, int);

Person(const Person&);

~Person() {}

string GetName() { return name; }

int GetAge() { return age; }

void SetName(string name) { this->name = name; }

void SetAge(int age) { this->age = age; }

virtual Person& operator= (const Person&);

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

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

};

Person::Person()

{

this->name = "-";

this->age = 0;

}

Person::Person(string newName, int newAge)

{

this->name = newName;

this->age = newAge;

}

Person::Person(const Person& t)

{

this->name = t.name;

this->age = t.age;

}

Person& Person::operator=(const Person& t)

{

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

name = t.name;

age = t.age;

return \*this;

}

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

{

cout << "Имя: "; in >> p.name;

cout << "Возраст: "; in >> p.age;

return in;

}

ostream& operator<<(ostream& out, const Person& p) { return (out << "Имя: " << p.name << "\nВозраст: " << p.age << "\n\n"); }

class Employee : public Person

{

protected:

string post;

double salary;

public:

Employee(void);

Employee(string, double, string, int);

Employee(const Employee&);

~Employee();

void SetSalary(double salary) { this->salary = salary; }

void SetPost(string post) { this->post = post; }

double GetSalary() { return salary; }

string GetPost() { return post; }

virtual Employee& operator=(const Employee&);

double cntsalary() {return salary \* 1.1;}

friend istream& operator>>(istream& in, Employee&t);

friend ostream& operator<<(ostream& out, const Employee&t);

};

Employee::Employee()

{

this->name = "-";

this->age = 0;

this->post = "-";

this->salary = 0;

}

Employee::Employee(string post, double salary, string name, int age)

{

this->post = post;

this->salary = salary;

this->name = name;

this->age = age;

}

Employee::Employee(const Employee& t)

{

this->post = t.post;

this->salary = t.salary;

this->name = t.name;

this->age = t.age;

}

Employee::~Employee() {}

istream& operator>>(istream& in, Employee& t)

{

cout << "Пост: "; in >> t.post;

cout << "Оклад: "; in >> t.salary;

return in;

}

ostream& operator<<(ostream& out, const Employee& t) { return (out << "Имя: " << t.name << "\nВозраст: " << t.age << "\nПост: " << t.post << "\nОклад: " << t.salary << "\n\n"); }

Employee& Employee::operator=(const Employee& t)

{

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

post = t.post;

salary = t.salary;

return \*this;

}

void f1(Person& c)

{

c.SetName("Джон");

cout << c << '\n';

}

Person f2()

{

Employee l("Слесарь", 25000, "Жора Букер", 34);

return l;

}

int main()

{

setlocale(LC\_ALL, "rus");

Person chel1("Леонардо Ди Каприо", 33);

cout << chel1;

Employee chel2("Актриса", 100000, "Анна Тэйлор Джой", 25);

cout << chel2;

cout << "ЗП с учетом премии: " << chel2.cntsalary() <<"\n\n";

Employee chel3 = chel2;

cout << chel3;

Employee chel321;

f1(chel321);

chel1 = f2();

cout << chel1;

return 0;

**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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **Каким образом описывается производный класс?**

Производный класс наследует описание базового класса; затем он может быть изменен добавлением новых членов, изменением существующих функций-членов и изменением прав доступа. С помощью наследования может быть создана иерархия классов, которые совместно используют код и интерфейсы.

* 1. **Наследуются ли конструкторы?**

Нет.

* 1. **Наследуются ли деструкторы?**

Нет.

* 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?**

Уничтожаются объекты в обратном порядке: сначала производный, потом его компоненты-объекты, а потом базовый объект.

* 1. **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. Классы, включающие такие функции, называются полиморфными и играют особую роль в ООП.

Виртуальные функции предоставляют механизм позднего (отложенного) или динамического связывания. Любая нестатическая функция базового класса может быть сделана виртуальной, для чего используется ключевое слово virtual.

* 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?**

Private – не наследуется, protected – protected, public – public.

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

Private – не наследуется, protected – private, public – private.

* 1. **В чем заключается принцип подстановки?**

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. Данное положение называется принципом подстановки. Он работает и для ссылок и для указателей. Обратное неверно. Например, всякий спортсмен (производный класс) является человеком (базовый класс), но не всякий человек является спортсменом.

Закрытое наследование – это наследование реализации, в этом случае принцип подстановки не соблюдается.

* 1. **Имеется иерархия классов:**

**class Student**

**{**

**public:**

**};**

**int age;**

**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?**

Protected: int stage, string post; public: string name;

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

Student() {age = 0; name = “”;}

Employee() {name = “”, post = “”;}

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

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

Student(int a, string n) {age = a; name = n;}

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

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

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

Student(const Student& t) {age = t.age; name = t.name;}

Employee(const Employee& t) {name = t.name, post = t.post;}

Teacher(const Teacher& t) {stage = t.stage; name = t.name, post = p.post;}

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

Student& operator =(const Student&c){

Age = c.age;

Name = c.name;

Return \*this;

}

Employee& operator =(const Employee&c){

post = c.post;

Name = c.name;

Return \*this;

}

Teacher& operator =(const Teacher&c){

Stage = c.stage;

Post = c.post;

Name = c.name;

Return \*this;

}