Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе №18.4**

Дисциплина: «Основы теории алгоритмов и структуры данных»  
Тема: Простое наследование. Принцип подстановки

Вариант 14

Выполнил:

студент группы РИС-20-2б

Вичугов Алексей Дмитриевич

Проверила:

доцент кафедры ИТАС

Полякова Ольга Андреевна

Пермь, 2021

**Цель работы**

1. Изучить построение функции хеширования и алгоритмов хеширования данных.

2. Создание иерархии классов с использованием простого наследования.

3. Изучение принципа подстановки.

**Постановка задачи**

1. Определить пользовательский класс.

2. Определить в классе конструкторы.

3. Определить в классе деструктор.

4. Определить в классе компоненты-функции для просмотра и установки полей данных.

5. Перегрузить операцию присваивания.

6. Перегрузить операции ввода-вывода 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.

10. Задание варианта: базовый класс: person с полями name и age. Создать производный класс teacher с полями subject и hours. Для обоих классов определить методы изменения полей.

**Анализ задачи**

1. Описание класса:

class Person{

public:

string name;

int age;

Person();

Person(string n, int a);

Person(Person &p);

~Person();

void Init(string n, int a);

virtual void Show();

string RName();

int RAge();

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

friend ostream& operator <<(ostream &ost, Person &p);

Person& operator =(Person &p);

};

class Teacher: public Person{

private:

string pred;

int hours;

public:

Teacher();

Teacher(string pr, int h);

Teacher(Teacher& t);

~Teacher();

string RPred();

int RHour();

void TInit();

void Show();

Teacher& operator =(Person &p);

};

2. Определение компонентных функций:

2.1. Для хранения значений элементов массива и хранения нужных индексов int Person::RAge() {return age;}

string Person::RName(){return name;}

string Teacher::RPred(){return pred;}

int Teacher::RHour(){return hours;}

Person::Person(){

name = "No name";

age=0;

}

Person::Person(string n, int a){

name=n;

age=a;

}

Person::Person(Person &p){

name=p.RName();

age=p.RAge();

}

Person::~Person(){}

void Person::Init(string n, int a){

name=n;

age=a;

}

void Person::Show(){

cout << "Name: " << name << endl << "Age: " << age << endl;

}

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

name=p.RName();

age=p.RAge();

}

Teacher::Teacher(){

pred="No subjects. ";

hours=0;

}

Teacher::Teacher(string pr, int h){

pred=pr;

hours=h;

}

Teacher::Teacher(Teacher &t){

pred=t.RPred();

hours=t.RHour();

}

Teacher::~Teacher(){}

void Teacher::TInit(){

cout << "Input subjects: ";

cin >> pred;

cout << "Input hours (int): ";

cin >> hours;

while (cin.fail()){

cin.clear();

cin.ignore(5,'\n');

cout << "Incorrect input! Please, repeat. " << endl;

cout << "Input hours (int): ";

cin >> hours;

}

}

void Teacher::Show() {

cout << "Name: " << name << endl << "Age: " << age << endl << "Subjects: " << pred << endl << "Hours: " << hours << endl;

}

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

name=p.RName();

age=p.RAge();

}

3. Определение глобальных функций:

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

cout << "Input name: ";

ist >> p.name;

ist.clear();

ist.ignore(5,'\n');

cout << "Input age: ";

ist >> p.age;

while (ist.fail()){

ist.clear();

ist.ignore(5,'\n');

cout << "Incorrect input! Please, repeat. " << endl;

cout << "Input age (int): ";

ist >> p.age;

}

return ist;

}

ostream& operator <<(ostream &ost, Person &p){

ost << "Name: " << p.name << endl << "Age: " << p.age << endl;

}

4. Функция main():

main(){

Person pfirst("Andrey",18);

pfirst.Show();

Person psecond(pfirst);

cout << psecond;

Teacher tfirst;

cout << tfirst;

tfirst.Init("Vilson", 42);

tfirst.TInit();

tfirst.Show();

Teacher tsecond("Yes", 52);

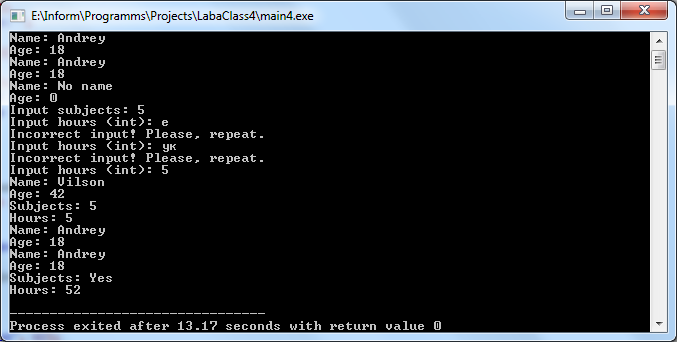
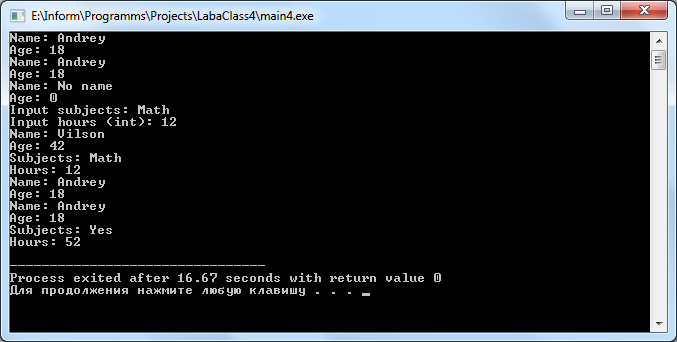
tsecond=psecond;

cout << tsecond;

tsecond.Show();

}

**Результаты работы программы**

****

**Ответы на вопросы**

1. Для создания модификации существующего класса с сохранением его полей и методов.

2. Все спецификаторы производного класса соответствуют спецификаторам базового класса.

3. Наследуются все компоненты кроме тех, которые имеют спецификатор private. Компоненты со спецификатором protected в производном классе получают спецификатор private, а компоненты со спецификатором public получают спецификатор protected.

4. Все компоненты, кроме имеющих спецификатор private, наследуются и получают спецификатор private.

5. После имени производного класса указывается двоеточие и имя базового класса. В фигурных содержится описание производного класса.

6. Нет.

7. Нет.

8. От первого родительского к последнему производному.

9. От последнего производного к первому базовому.

10. Виртуальные функции представляют собой функции, которые имеют одинаковое название, но разную реализацию во всех производных классах. Механизм позднего связывания базируется на виртуальных функциях.

11. Не могут.

12. Да.

13. Класс-наследник является частью класса-родителя.

14. Наследование реализации.

15. Вместо базового класса можно подставлять его производные классы с публичным наследованием.

16. int age; string name; string post; int stage;

17. Student() : age(0), name(""){}  
Employee() : post("") {}

Teacher(): stage(0){}

18. Student(int a, string n) : age(a), name(n){}  
Employee(string p) : post(p) {}

Teacher(int s): stage(s){}

19. Student(Student &s) : age(s.age), name(s.name){}  
Employee(Employee &e) : post(e.post) {}

Teacher(Teacher &t): stage(t.stage){}

20. Student& operator =(Student &s){age=s.age;name=s.name;}

Employee operator =(Employee &e) {post=e.post;}

Teacher operator =(Teacher &t) {stage=t.stage;}