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 12

Выполнил:

студент группы РИС-20-2б

Пономарев Егор Витальевич

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.

**Задание варианта:**

Базовый класс:

ЧЕЛОВЕК(Person)

Имя(name) - string

Возраст(age) – int

Определить методы изменения полей.

Создать произвольный класс STUDENT, имеющий поле год обучения. Определить методы изменения и увеличения года обучения.

**Анализ задачи**

1. Описание класса:

class Person

{

public:

Person(void);

public:

virtual ~Person(void);

Person(string, int);

Person(const Person&);

string Get\_name() { return name; }

int Get\_age() { return age; }

void Set\_name(string);

void Set\_age(int);

Person& operator=(const Person&);

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

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

protected:

string name;

int age;

};

class Student :

public Person

{

public:

Student(void);

public:

~Student(void);

Student(string, int, int);

Student(const Student&);

int Get\_year() { return year; }

Student& operator=(const Student&);

Student& operator++();

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

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

protected:

int year;

};

2. Определение компонентных функций:

2.1. Для хранения значений элементов массива и хранения нужных индексов int

Person::Person(string N, int A)

{

name = N;

age = A;

}

Person::Person(const Person& person)

{

name = person.name;

age = person.age;

}

void Person::Set\_age(int P)

{

age = P;

}

void Person::Set\_name(string N)

{

name = N;

}

Student::Student(void): Person()

{

year = 0;

}

Student::~Student(void)

{

}

Student::Student(string N, int A,int Y): Person(N,A)

{

year = Y;

}

3. Определение глобальных функций:

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

{

cout << "\nName: ";

in >> s.name;

cout << "\nAge: ";

in >> s.age;

cout << "\nYear: ";

in >> s.year;

return in;

}

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

{

out << "\nName: " << s.name;

out << "\nAge: " << s.age;

out << "\nYear: " << s.year;

out << "\n";

return out;

}

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

{

cout << "\nName: ";

in >> p.name;

cout << "\nAge: ";

in >> p.age;

return in;

}

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

{

out << "\nName: " << p.name;

out << "\nAge: " << p.age;

out << "\n";

return out;

}

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

{

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

name = p.name;

age = p.age;

return \*this;

}

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

{

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

name = S.name;

age = S.age;

year = S.year;

return \*this;

}

Student&Student::operator++()

{

int m = year;

m++;

year = m;

return\*this;

}

4. Функция main():

void main()

{

Person a;

cin >> a;

cout << a;

Person b("Katya", 19);

cout <<b;

a = b;

cout << a;

Student s;

cin >> s;

++s;

cout << s;

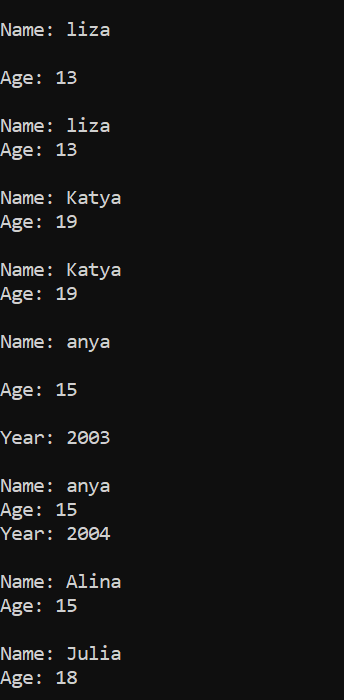
f1(s);

a = f2();

cout << a;

}

**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;}