Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе № 15.4**

Дисциплина: «Информатика»

Тема: Объектно-ориентированное программирование. Простое наследование. Принцип подстановки.

Вариант 12

Выполнила:

Студентка группы РИС-22-1б

Черкасова А.А.

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

**Пермь 2023**

**Содержание**

**Введение**

**Цель:**

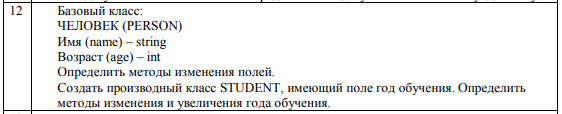
1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Создание иерархии классов с использованием простого наследования.
3. Изучение принципа подстановки.

Для достижения поставленной цели, необходимо решить следующие **задачи**:

* Провести анализ задачи
* Реализовать задачу на языке С++
* Составить блок-схему

**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 операцию ввода и вывода объектов с помощью потоков.
7. Определить произвольный 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**

1. Для решения задачи необходимо:
   1. Реализовать определение класса Person и определение необходимых методов и операторов данного класса.
   2. Реализовать определение производного от класса Person класса Student и определение необходимых методов и операторов данного класса.
   3. Реализовать применение этих функций в главной функции.
2. В ходе работы были использованы типы данных:
   1. Для конструктора без параметров Person / Student класса Person / Student не используются аргументы:

Сам конструктор заполняет атрибуты объекта класса “пустотами”

* 1. Для конструктора с параметрами Person / Student класса Person / Student используются следующие аргументы:
  2. Тип string: имя
  3. Тип int: возраст/ год обучения.

Сам конструктор заполняет атрибуты объекта класса параметрами.

* 1. Для конструктора копирования Person / Student класса Person / Student используются следующие аргументы:
  2. Адресация типа const Person& / const Student&: объект копирования.

Сам конструктор заполняет атрибуты объекта класса атрибутами объекта копирования.

* 1. Для деструктора Person / Student класса Person / Student не используются аргументы:

Сам деструктор удаляет атрибуты объекта класса.

* 1. Для селектора get\_name не используются аргументы.

Сам селектор имеет тип string и возвращает значение поля name.

* 1. Для селектора get\_age не используются аргументы.

Сам селектор имеет тип int и возвращает значение поля age.

* 1. Для модификатора set\_name используются следующие аргументы:

1. Тип string: имя.

Сам модификатор имеет тип void, поскольку при работе селектора не нужно возвращать значение.

* 1. Для модификатора set\_age используются следующие аргументы:

1. Тип int: возраст

Сам модификатор имеет тип void, поскольку при работе селектора не нужно возвращать значение.

* 1. Для метода increase\_year не используются аргументы.

Сам метод имеет тип void, поскольку при работе метода не нужно возвращать значение.

* 1. Для метода перегрузки оператора присваивания используются следующие аргументы:
  2. Тип const Person& / Student&: ссылка на объект.

Сам метод имеет тип Person& / Student& и возвращает указатель на объект this.

* 1. Для дружественной функции перегрузки оператора ввода используются следующие аргументы:
  2. Тип istream&: поток ввода.
  3. Тип const Person& / const Student&: ссылка на объект.

Сама функция имеет тип istream& и возвращает вводимые данные.

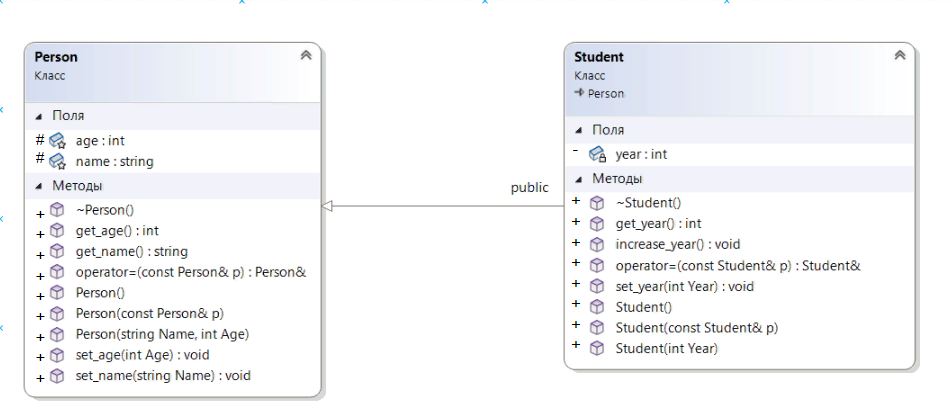
* 1. Для дружественной функции перегрузки оператора вывода используются следующие аргументы:
  2. Тип ostream&: поток вывода.
  3. Тип const Person& / const Student&: ссылка на объект.

Сама функция имеет тип ostream& и возвращает выводимые данные.

1. Для решения задачи данные были представлены в следующем виде:
   1. Для работы с данными используются атрибуты класса.
2. Для операций ввода и вывода использовались следующие операторы и функции:
   1. Ввод данных реализован с помощью оператора cin, используемых при реализации в главной функции и функций перегрузки оператора.
   2. Вывод данных реализован с помощью оператора cout, используемого при реализации в главной функции и функций перегрузки операторов ввода-вывода.
3. Поставленные задачи решены следующими действиями:
   1. Определение класса Person / Student было реализовано в заголовочном файле Person.h / Student.h, определение конструкторов, деструктора, селекторов и модификаторов класса – в файле Person.cpp / Student.cpp. Основной блок программы описан в главном файле main.cpp.
   2. При работе с объектами классов Person и Student обращение к конструктору без параметра аналогично выделению памяти под переменную. Обращение к конструктору с параметрами аналогично обращению к методу Init. Обращение к конструктору копирования аналогично оператору присваивания. Обращение к селекторам и модификаторам аналогично обращению к методам или к полям структуры.
   3. При работе с перегруженными операторами работа без объектов созданного класса происходит по умолчанию. Если при вызове перегруженных операторов используются объекты класса, то операторы работают согласно их определению в этом классе.

# 

# **UML – диаграмма**



# **Приложение А**

**Листинг программы**

*A) Заголовочный файл Person.h*

#pragma once

#include<string>

using namespace std;

class Person

{

private:

string name;

int age;

public:

Person();

Person(string Name, int Age);

Person(const Person& p);

~Person();

string get\_name();

void set\_name(string Name);

int get\_age();

void set\_age(int Age);

Person& operator= (const Person&p); // Оператор присваивания

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

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

};

*B) Файл с описанием методов класса Person.cpp*

#include "Person.h"

#include "Person.h"

#include<string>

#include <iostream>

using namespace std;

Person::Person()

{

name = " ";

age = 0;

}

Person::Person(string Name, int Age)

{

name = Name;

age = Age;

}

Person::Person(const Person& p)

{

name = p.name;

age = p.age;

}

Person::~Person()

{

}

string Person::get\_name()

{

return name;

}

void Person::set\_name(string Name)

{

name = Name;

}

int Person::get\_age()

{

return age;

}

void Person::set\_age(int Age)

{

age = Age;

}

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

{

name = p.name;

age = p.age;

return \*this;

}

istream& operator >> (istream& in, Person& p) // Оператор ввода

{

cout << "INPUT NAME: ";

in >> p.name;

cout << "INPUT AGE: ";

in >> p.age;

return in;

}

ostream& operator << (ostream& out, const Person& p) // Оператор вывода

{

out << "\nNAME: " << p.name << endl;

out << "AGE: " << p.age << endl;

return out;

}

*C) Заголовочный файл Student.h*

#pragma once

#include "Person.h"

class Student : public Person

{

private:

int year;

public:

Student();

Student(int Year);

Student(const Student& p);

~Student();

int get\_year();

void set\_year(int Year);

Student& operator= (const Student& p); // Оператор присваивания

friend istream& operator>> (istream& in, Student& p); // Оператор вывода

friend ostream& operator<< (ostream& out, const Student& p); // Оператор ввода

void increase\_year();

};

*D) Файл с описанием методов класса Student.cpp*

#include "Student.h"

#include <iostream>

using namespace std;

Student::Student()

{

year = 0;

}

Student::Student(int Year)

{

year = Year;

}

Student::Student(const Student& p)

{

year = p.year;

name = p.name;

age = p.age;

}

Student::~Student()

{

}

int Student::get\_year()

{

return year;

}

void Student::set\_year(int Year)

{

year = Year;

}

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

{

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

name = p.name;

age = p.age;

year = p.year;

return \*this;

}

istream& operator >> (istream& in, Student& p) // Оператор ввода

{

cout << "\nINPUT NAME: ";

in >> p.name;

cout << "INPUT AGE: ";

in >> p.age;

cout << "INPUT YEAR: ";

in >> p.year;

return in;

}

ostream& operator << (ostream& out, const Student& p) // Оператор вывода

{

out << "\nNAME: " << p.name << endl;

out << "AGE: " << p.age << endl;

out << "YEAR: " << p.year << endl;

return out;

}

void Student::increase\_year()

{

year += 1;

}

*E) Файл с главной программой main.cpp*

#include <iostream>

#include "Student.h"

#include "Person.h"

#include <string>

using namespace std;

void func1(Person& p) // Объект базового класса как параметр

{

p.set\_name("Вася");

p.set\_age(18);

cout << p;

}

Person func2() // Возвращает объект базового класса как результат

{

Student year(4);

return year;

}

int main()

{

system("chcp 1251>nul");

cout << "class Student" << endl;

Student A;

cin >> A;

cout << "\nObject A :\n" << A;

A.increase\_year();

cout << "\nВывод объекта A с годом +1: \n" << A;

cout << "\nПодстановка" << endl;

func1(A);

Person res = func2(); // подставновка

cout << "\nObject A:\n" << A;

cout << "\nResult \n" << res;

cout << "class Person" << endl;

Person One;

cout << "\n Инициализация через конструктор по умолчанию\n" << One;

string name;

int age;

cout << "\nEnter Name: ";

cin >> name;

cout << "Enter Age: ";

cin >> age;

Person Two;

Two.set\_name(name);

Two.set\_age(age);

cout << "\n Инициализация через модификатор\n" << Two;

Person Three("Петя", 20);

cout << "\n Инициализация через конструктор с параметрами\n" << Three;

Person Four(Three);

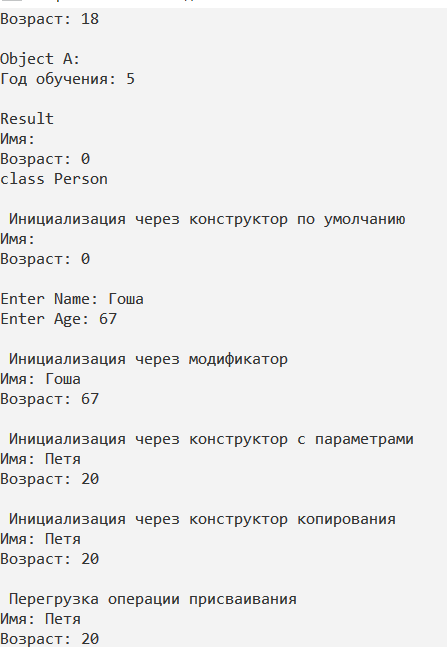
cout << "\n Инициализация через конструктор копирования\n" << Four;

Two = Four;

cout << "\n Перегрузка операции присваивания\n" << Two;}

# **Приложение Б**

**Результаты выполнения программы**



**Ответы на вопросы**

1. Для чего используется механизм наследования?

Наследование позволяет структурировать и повторно использовать код и ускорить процесс разработки. Оно позволяет выделить общее для нескольких классов поведение и вынести его в отдельную сущность - базовый класс.

2. 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.

3. 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.

4. 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.

5. Каким образом описывается производный класс?

Синтаксис определения производного класса:

class имя\_класса : список\_базовых\_классов

{список\_компонентов\_класса};

class Base{ //базовый класс

…

};

class Derive: Base { //производный класс

…

};

6. Наследуются ли конструкторы?

Конструкторы не наследуются, при создании производного класса

наследуемые им данные-члены должны инициализироваться конструктором базового класса.

7. Наследуются ли деструкторы?

Деструкторы не наследуются. Однако они вызываются, когда объект производного класса выходит из области видимости.

8. 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.

9. В каком порядке уничтожаются объекты производных классов?

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

10. 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. Интерпретация каждого вызова виртуальной функции через указатель на базовый класс зависит от значения этого указателя, т.е. от типа объекта, для которого выполняется вызов.

class Base {

public:

virtual void print(){cout<<”\nBase”;}

. . .

};

class Derive : public Base {

public:

void print(){cout<<”\n Derive”;}

};

int main() {

Base B,\*bp;

Derive D,\*dp;

bp=&B; //указатель базового класса ставится на объект

dp=&D; //производного класса

Base \*p = &D;

bp –>print(); // вызывается метод для Base

dp –>print(); // вызывается метод для Derive

p –>print(); // вызывается метод для Derive

return 0;

}

11. Могут ли быть виртуальными конструкторы? Деструкторы?

Виртуальными могут быть только нестатические функции-члены, то есть конструкторы и деструкторы не могут быть виртуальными, так как они ненаследуемы.

12. Наследуется ли спецификатор virtual?

Виртуальность наследуется. После того как функция определена как виртуальная, ее повторное определение в производном классе (с тем же самым прототипом) создает в этом классе новую виртуальную функцию, причем спецификатор virtual может не использоваться.

13. Какое отношение устанавливает между классами открытое наследование?

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.Таким образом, порожденный класс представляет собой модификацию базового класса.

class A { protected: int i; };

class B: public A {...};// i остается protected членом B

class C: public B {public:void f();};

//поскольку i является protected для B, оно может наследоваться вC

void C::f(){ i = 2;} //данная функция работает т.к. i доступна в C

14. Какое отношение устанавливает между классами закрытое наследование?

Закрытое наследование означает, что от базового класса необходимо взять какую-то функциональность, базовый класс и потомок не имеют какой-либо концептуальной связи .Закрытое наследование не носит характера отношения подтипов. Закрытое (также как и защищенное) наследование не создает иерархии типов.

class A { protected: int i; };

class B: private A {...};// i преобразовано к private члену B

class C: public B {public:void f();};

//поскольку i является private для B, оно не может наследоваться вC (не создается иерархии объектов)

void C::f(){ i = 2;}// данная функция не работает т.к. i не доступна

15. В чем заключается принцип подстановки?

Принцип подстановки: класс S может считаться подклассом T, если замена объектов T на объекты S не приведёт к изменению работы программы.

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?

В классе Employee открыто наследуется public name (age не наследуется, поскольку поле private). В Teacher открыто наследуется public name, и открыто наследуется protected post. Таким образом, у класса Teacher доступны следующие поля: public name, protected post, protected stage.

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

Student::Student(){

age = 0;

name = “NULL”; }

Employee::Employee() {

name = “NULL”;

post = “NULL”; }

Teacher::Teacher(){

name = “NULL”;

post = “NULL”;

stage = 0; }

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

Student::Student(int age, string name){

this->age = age;

this->name = name; }

Employee::Employee(string name, string spost) {

this->name = name;

this->post = post; }

Teacher::Teacher(string name, string post, int stage){

this->name = name;

this->post = post;

this->stage = stage; }

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

Student::Student(const Student& s){

age = s.age;

name = s.name; }

Employee::Employee(const Employee& e) {

name = e.name;

post = e.post; }

Teacher::Teacher(const Teacher& t){

name = t.name;

post = t.post;

stage = t.stage; }

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

Student& Student::operator = (const& Student s) {

if (&s==this) return\*this; //проверка на самоприсваивание

name = s.name;

age = s.age;

return \*this; }

Employee& Employee::operator = (const& Employee e) {

if (&e==this) return\*this; //проверка на самоприсваивание

name = e.name;

age = e.age;

return \*this; }

Teacher& Teacher::operator = (const& Teacher t) {

if (&t==this) return\*this; //проверка на самоприсваивание

name = t.name;

age = t.age;

return \*this; }