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 №18.4**

Дисциплина: основы алгоритмизации и программирования

Тема: “Классы и объекты. Принцип подстановки.”

Вариант 4

Выполнил:

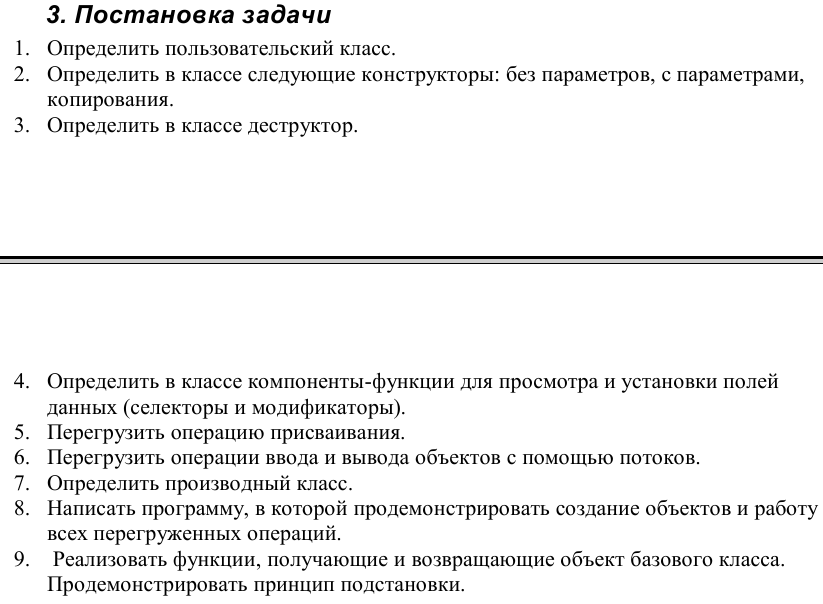
студент группы ИВТ-20-2Б Чувашев Максим

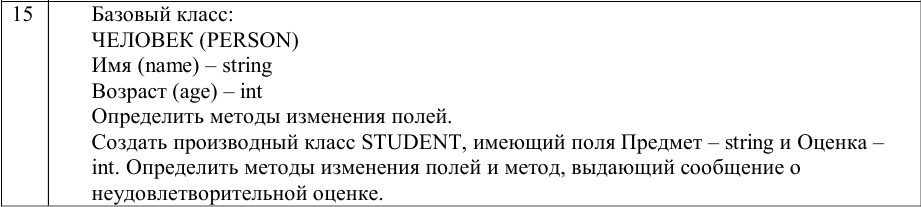
Проверила: доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**





**Анализ задачи**

1. Используемые типы данных:
   1. Double
   2. Int
   3. PERSON
   4. String
2. Действия над используемыми данными
   1. Int, double – арифметические операции.
3. Структуры
   1. Person – класс для хранения сведений о человеке. Конструктор создает сведения о человеке

class PERSON

{

//конструктор без параметров

public:

PERSON(void);

public:

//деструктор

virtual ~PERSON(void);

//констрктор с параметрами

PERSON(string,int);

//конструктор копирования

PERSON(const PERSON&);

//Геттеры

string Get\_name()

{

return name;

}

int Get\_age()

{

return age;

}

//Сеттеры

void Set\_name(string);

void Set\_age(int);

//перегрузка операции присваивания

PERSON& operator=(const PERSON&);

//глобальные операторы-функции ввода-вывода

friend istream& operator>>(istream& in, PERSON& c);

friend ostream& operator<<(ostream& out, const PERSON& c);

//атрибуты

protected:

string name;

int age;

};

//конструктор без параметров

PERSON::PERSON(void)

{

name = "";

age = 0;

}

//деструктор

PERSON::~PERSON(void)

{

}

//конструктор с параметрами

PERSON::PERSON(string M,int C)

{

name = M;

age = C;

}

//конструктор копирования

PERSON::PERSON(const PERSON& person)

{

name = person.name;

age = person.age;

}

//модификаторы

void PERSON::Set\_age(int C)

{

age = C;

}

void PERSON::Set\_name(string M)

{

name = M;

}

//перегрузка операции присваивания

PERSON& PERSON::operator=(const PERSON&c)

{

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

name = c.name;

age = c.age;

return \*this;

}

* 1. Student : public Person – класс для хранения сведений о студенте. Наследуется для класса Person.

#pragma once

#include "PERSON.h"

//класс STUDENT наследуется от класса PERSON

class STUDENT :public PERSON

{

public:

STUDENT(void);//конструктор без параметров

public:

~STUDENT(void);//деструктор

STUDENT(string,int,int, string);//конструктор с параметрами

STUDENT(const STUDENT& );//конструктор копирования

string Get\_subject() { return subject; }//модификатор

void Set\_subject(string);//селектор

int Get\_mark() { return mark; }//модификатор

void Set\_mark(int);//селектор

STUDENT& operator=(const STUDENT&);//операция присваивания

void Check\_Mark(const STUDENT&);

friend istream& operator>>(istream& in, STUDENT& l);//операция ввода

friend ostream& operator<<(ostream& out,const STUDENT& l); //операция вывода

protected:

int mark;//поле оценки студента

string subject;//поле предмета

};

//конструктор без параметров

STUDENT::STUDENT(void):PERSON()

{

mark = 2;

}

//дестрктор

STUDENT::~STUDENT(void)

{

}

//конструктор с параметрами

STUDENT::STUDENT(string M, int C, int mark, string subject) :PERSON(M, C)

{

this->name = M;

this->age = C;

this->mark = mark;

this->subject = subject;

}

//конструктор копирования

STUDENT::STUDENT(const STUDENT &other)

{

this->name = other.name;

this->age = other.age;

this->mark = other.mark;

this->subject = other.subject;

}

//модификатор

void STUDENT::Set\_mark(int)

{

this->mark = mark;

}

void STUDENT::Set\_subject(string)

{

this->subject = subject;

}

//оперция присваивания

STUDENT& STUDENT::operator=(const STUDENT& l)

{

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

name = l.name;

age = l.age;

mark = l.mark;

subject = l.subject;

return \*this;

}

1. Организация ввода-вывода:
   1. С помощью оператора ввода cin
   2. С помощью оператора вывода cout
   3. Перегрузка оператора << класса person

//глобальная функция для вывода

ostream& operator<<(ostream& out, const PERSON& c)

{

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

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

out << "\n";

return out;

}

* 1. Перегрузка оператора << для класса Student

ostream& operator<<(ostream& out, const STUDENT& l)

{

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

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

out << "\nmark : " << l.mark;

out << "\nsubject:" << l.subject;

out << "\n";

return out;

}

1. Действия для решения задачи. Демонстрация работы класса в функции main().

void main()

{

system("chcp 1251>nul");

setlocale(LC\_ALL, "Ru");

//работа с классом PERSON

PERSON a;

cin >> a;

cout << a;

PERSON b("Pavel", 18);

cout << b;

a = b;

cout << a;

//работа с классом STUDENT

STUDENT c;

cin >> c;

cout << c;

c.Check\_Mark(c);

//принцип подстановки

f1(c);//передаем объект класса STUDENT

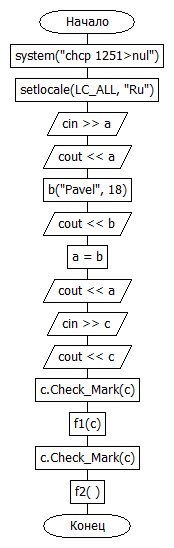
c.Check\_Mark(c);

f2();//создаем в функции объект класса STUDENT

}

**Блок-схема**





**Person.h**



**Student.h**



**Person.h**



**Student.h**





**Код**

**Sourse.cpp**

#include <iostream>

#include "PERSON.h"

#include "Student.h"

using namespace std;

// глобальные функции

void f1(STUDENT&c)

{

c.Set\_name("Maxim");

cout << c;

}

PERSON f2()

{

STUDENT l("Alex", 18, 2, "inform");

cout << l;

l.Check\_Mark(l);

return l;

}

void main()

{

system("chcp 1251>nul");

setlocale(LC\_ALL, "Ru");

//работа с классом PERSON

PERSON a;

cin >> a;

cout << a;

PERSON b("Pavel", 18);

cout << b;

a = b;

cout << a;

//работа с классом STUDENT

STUDENT c;

cin >> c;

cout << c;

c.Check\_Mark(c);

//принцип подстановки

f1(c);//передаем объект класса STUDENT

c.Check\_Mark(c);

f2();//создаем в функции объект класса STUDENT

}

**Person.h**

#pragma once

#include <string>

#include <iostream>

using namespace std;

class PERSON

{

//конструктор без параметров

public:

PERSON(void);

public:

//деструктор

virtual ~PERSON(void);

//констрктор с параметрами

PERSON(string,int);

//конструктор копирования

PERSON(const PERSON&);

//Геттеры

string Get\_name()

{

return name;

}

int Get\_age()

{

return age;

}

//Сеттеры

void Set\_name(string);

void Set\_age(int);

//перегрузка операции присваивания

PERSON& operator=(const PERSON&);

//глобальные операторы-функции ввода-вывода

friend istream& operator>>(istream& in, PERSON& c);

friend ostream& operator<<(ostream& out, const PERSON& c);

//атрибуты

protected:

string name;

int age;

};

**Student.h**

#pragma once

#include "PERSON.h"

//класс STUDENT наследуется от класса PERSON

class STUDENT :public PERSON

{

public:

STUDENT(void);//конструктор без параметров

public:

~STUDENT(void);//деструктор

STUDENT(string,int,int, string);//конструктор с параметрами

STUDENT(const STUDENT& );//конструктор копирования

string Get\_subject() { return subject; }//модификатор

void Set\_subject(string);//селектор

int Get\_mark() { return mark; }//модификатор

void Set\_mark(int);//селектор

STUDENT& operator=(const STUDENT&);//операция присваивания

void Check\_Mark(const STUDENT&);

friend istream& operator>>(istream& in, STUDENT& l);//операция ввода

friend ostream& operator<<(ostream& out,const STUDENT& l); //операция вывода

protected:

int mark;//поле оценки студента

string subject;//поле предмета

};

**Person.cpp**

#include "PERSON.h"

//конструктор без параметров

PERSON::PERSON(void)

{

name = "";

age = 0;

}

//деструктор

PERSON::~PERSON(void)

{

}

//конструктор с параметрами

PERSON::PERSON(string M,int C)

{

name = M;

age = C;

}

//конструктор копирования

PERSON::PERSON(const PERSON& person)

{

name = person.name;

age = person.age;

}

//модификаторы

void PERSON::Set\_age(int C)

{

age = C;

}

void PERSON::Set\_name(string M)

{

name = M;

}

//перегрузка операции присваивания

PERSON& PERSON::operator=(const PERSON&c)

{

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

name = c.name;

age = c.age;

return \*this;

}

//глобальная функция для ввода

istream& operator>>(istream&in,PERSON&c)

{

cout << "\nName:";

in >> c.name;

cout << "\nAge:";

in >> c.age;

while (c.age < 14 || c.age > 40)

{

cout << "\nНе уверен что в таком возрасте учатся в университете\n";

cout << "\nAge:";

in >> c.age;

}

return in;

}

//глобальная функция для вывода

ostream& operator<<(ostream& out, const PERSON& c)

{

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

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

out << "\n";

return out;

}

**Student.cpp**

#include "Student.h"

//конструктор без параметров

STUDENT::STUDENT(void):PERSON()

{

mark = 2;

}

//дестрктор

STUDENT::~STUDENT(void)

{

}

//конструктор с параметрами

STUDENT::STUDENT(string M, int C, int mark, string subject) :PERSON(M, C)

{

this->name = M;

this->age = C;

this->mark = mark;

this->subject = subject;

}

//конструктор копирования

STUDENT::STUDENT(const STUDENT &other)

{

this->name = other.name;

this->age = other.age;

this->mark = other.mark;

this->subject = other.subject;

}

//модификатор

void STUDENT::Set\_mark(int)

{

this->mark = mark;

}

void STUDENT::Set\_subject(string)

{

this->subject = subject;

}

//оперция присваивания

STUDENT& STUDENT::operator=(const STUDENT& l)

{

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

name = l.name;

age = l.age;

mark = l.mark;

subject = l.subject;

return \*this;

}

//операция ввода

istream& operator>>(istream& in, STUDENT& l)

{

cout << "\nName:";

in >> l.name;

cout << "\nAge:";

in >> l.age;

while (l.age < 14 || l.age > 40)

{

cout << "\nНе уверен что в таком возрасте учатся в университете\n";

cout << "\nAge:";

in >> l.age;

}

cout << "\nsubject:";

in >> l.subject;

cout << "\nmark:";

in >> l.mark;

while (l.mark < 2 || l.mark > 5)

{

cout << "\nВведена некорректная оценка!\n";

cout << "\nMark:";

in >> l.mark;

}

return in;

}

//операция вывода

ostream& operator<<(ostream& out, const STUDENT& l)

{

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

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

out << "\nmark : " << l.mark;

out << "\nsubject:" << l.subject;

out << "\n";

return out;

}

void STUDENT::Check\_Mark(const STUDENT& stud)

{

switch (stud.mark)

{

case 2: cout << "\nОценка студента " << stud.name << " по предмету " << stud.subject << " неудовлетворительна!\n"; break;

case 3:

case 4:

case 5: cout << "\nОценка студента " << stud.name << " по предмету " << stud.subject << " удовлетворительна!\n"; break;

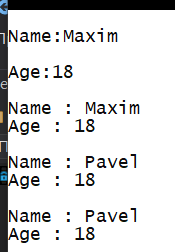
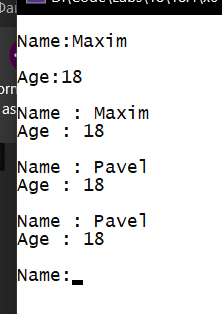
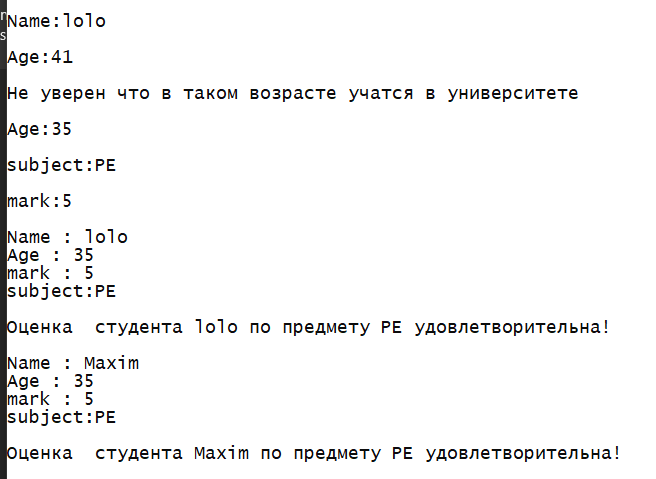
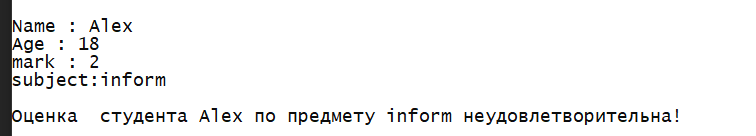
default:

break;

}

}

**Скриншоты**

1. 
2. 
3. 
4. 

**Ответы на контрольные вопросы**



1. Наследование позволяет реализовать типизированное повторное использование объектов, создание иерархии «тип-подтип» с сохранением подтипом всех свойств своего предка. Можно рассматривать наследование для всех трех составляющих объекта.



2. Когда вы открыто наследуете родительский класс, то унаследованные public-члены остаются public.



3. Когда вы открыто наследуете родительский класс, то унаследованные private-члены остаются недоступными для дочернего класса.



4. К protected-членам родительского класса доступ открыт для членов дочернего класса.



5. 



6. Конструкторы не наследуются.



7. Деструкторы не наследуются.



8. 



9. 

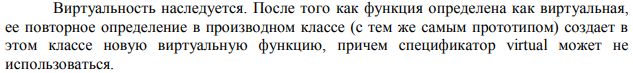


10. 

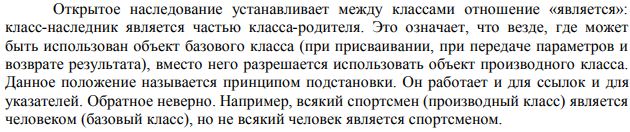


11. 



12. 



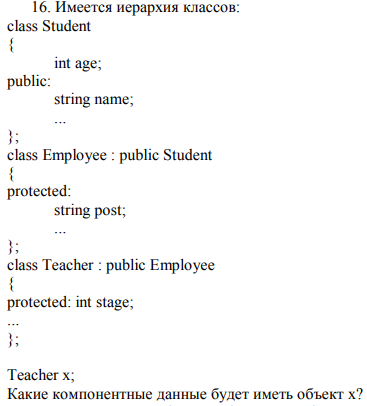
13. 



14. 



15. Пункт 13 и 14 – принцип подстановки.



16. age, name, post, stage.



17. Student(); Employee() : Student(); Teacher() : Employee();