Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: Простое наследование. Принцип подстановки.

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Токарев Павел Аркадьевич

Проверил

Доцент кафедры ИТАС

Полякова Ольга Андреевна

Г. Пермь-2023

**Постановка задачи**

1.Определить новый класс.

2.Определить в классе конструкторы: без параметров, с параметрами, копирования и деструктор.

3.Определить в классе селекторы и модификаторы

4.Перегрузить операцию присваивания, операции ввода и вывода, операцию вычитания временных интервалов и операцию их сравнения(!=).

5.В главной функции продемонстрировать методы класса.

**Анализ**

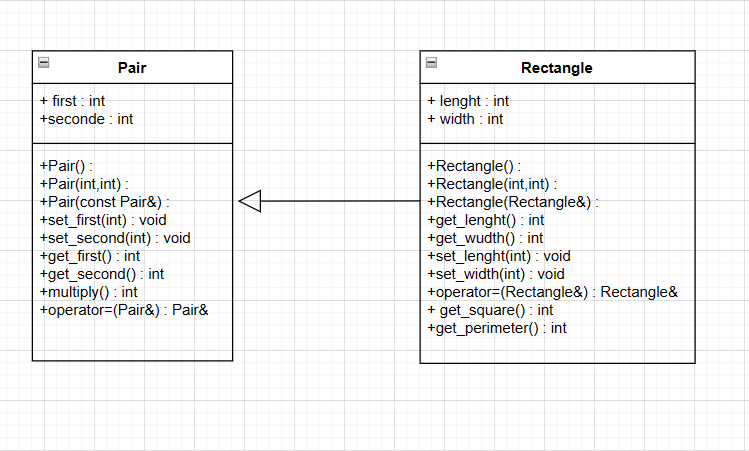
1.Определим класс Time, работающий с временными интервалами. В нём определим 3 вида конструкторов, деструктор, селекторы и модификаторы.

2.Перегрузим операторы ввода и вывод, как внешние дружественные функции.

3.Перегрузим оператор присваивания. Теперь она возвращает ссылку на объект, которому присваивают.

4.В функции main покажем создание объектов и все перегруженные операции.

**Диаграмма UML**

****

**Код программы**

main.cpp

#include"Pair.h"

#include"Rectangle.h"

#include <iostream>

using namespace std;

int main()

{

Pair a;

cin >> a;

cout << a.multyply();

cout << endl;

cout << a;

Rectangle b(24,9);

Rectangle c = b;

cout << c.get\_square()<<endl;

cout << c.get\_perimeter() << endl;

}

Pair.h

#pragma once

#include<iostream>

using namespace std;

class Pair

{

public:

int first;

int second;

Pair();

Pair(int, int);

Pair(const Pair&);

~Pair();

int get\_first();

int get\_second();

void set\_first(int);

void set\_second(int);

Pair& operator=(const Pair&);

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

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

int multyply();

};

Pair.cpp

#include "Pair.h"

#include<iostream>

using namespace std;

Pair::Pair() {

first = 0;

second = 0;

}

Pair::Pair(int tmp1, int tmp2) {

first = tmp1;

second = tmp2;

}

Pair::Pair(const Pair& t) {

first = t.first;

second = t.second;

}

Pair::~Pair() {};

int Pair::get\_first() {

return first;

}

int Pair::get\_second() {

return second;

}

void Pair::set\_first(int tmp) {

first = tmp;

}

void Pair::set\_second(int tmp) {

second = tmp;

}

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

first = t.first;

second = t.second;

return \*this;

}

istream& operator>>(istream& in, Pair& t) {

cout << "first:";

in >> t.first;

cout << "second:";

in >> t.second;

return in;

}

ostream& operator<<(ostream& out, const Pair& t) {

out << "first:" << t.first;

out << " ";

out << "second:" << t.second;

out << endl;

return out;

}

int Pair::multyply() {

return(first \* second);

}

Rectangle.h

#pragma once

#include"Pair.h"

class Rectangle:

public Pair

{

public:

Rectangle(void);

Rectangle(int, int);

Rectangle(const Rectangle& t);

int get\_leght();

int get\_width();

void set\_lenght(int);

void set\_width(int);

Rectangle& operator=(const Rectangle&);

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

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

int get\_square();

int get\_perimeter();

};

Rectangle.cpp

#include "Rectangle.h"

#include<iostream>

using namespace std;

Rectangle::Rectangle(void):Pair() {

}

Rectangle::Rectangle(int tmp1, int tmp2):Pair(tmp1,tmp2) {

}

Rectangle::Rectangle(const Rectangle& t) {

first = t.first;

second = t.second;

}

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

first = t.first;

second = t.second;

return \*this;

}

int Rectangle::get\_perimeter() {

return(2 \* first + 2 \* second);

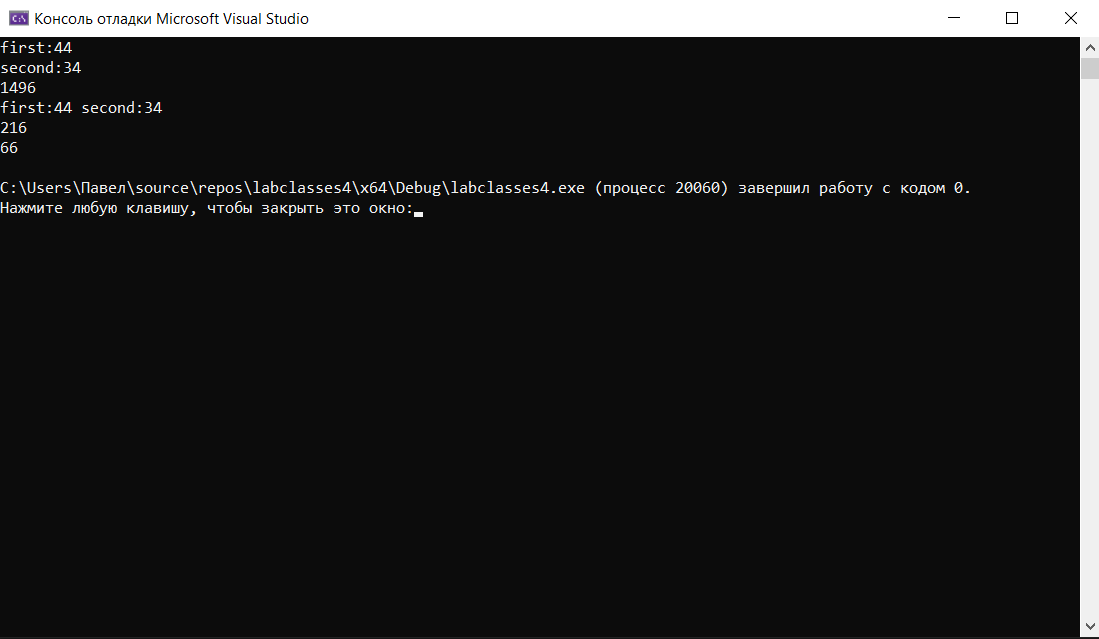
}

int Rectangle::get\_square() {

return(first \* second);

}

**Результаты работы программы**

****

**Анализ результатов**

Удалось успешно реализовать наследование классов. В программе корректно работают все необходимые функции. Без ошибок работают перегруженный функции-операторы.

**Контрольные вопросы**

1.Наследование применяется, когда необходимо получить новый класс на основе уже существующего.

2.Public член класса может использоваться как базовым, так и производными классами.

3.Private член класса может использоваться только функциями данного класса. В производном классе он недоступен.

4.Protected член класса может использоваться только функциями данного класса. В производном классе он недоступен, однако может использовать методами и дружественными функциями производного класса.

5. class (имя производного класса): (спецификатор доступа) (имя базового класса) {тело класса}

6.Конструкторы не наследуются. При определении объекта производного класса выполняется конструктор базового класса до момента вызова конструктора производного класса.

7.Деструкторы не наследуются. Деструкторы определяются аналогично конструкторам.

8.Сначало конструируются поля базового класса, потом производного.

9.Сначала уничтожаются поля базового класса, потом производного.

10.Вируальные функции применяются, когда в каждом производном классе необходим своя уникальная реализация данной функции. Позднее связывание означает, что объект связывается с вызовом функции только во время исполнения программы, а не раньше.

11.Конструкторы не могут быть виртуальными, а деструкторы могут. Практически у каждого класса, имеющего виртуальную функцию имеется виртуальный деструктор.

12.Спецификатор virtual может наследоваться.

13.Открытое наследование устанавливает такую зависимость, что производный класс является частью базового.

14.При закрытом наследовании производный класс не может использовать методы базового класса. При таком наследовании все методы реализуются заново.

15.Принцип подстановки : класс A будет считаться подклассом B, если замена объектов A на B не приведёт к изменениям работы программы.

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

17. Student::Student(){  
 age = 0;  
 name = “NULL”; }  
Employee::Employee() {  
 name = “NULL”;  
 post = “NULL”;}  
Teacher::Teacher(){  
 name = “NULL”;  
 post = “NULL”;  
 stage = 0; }

18. 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. 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. 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; }