Министерство образования и науки Российской Федерации

Федеральное государственное образовательное автономное учреждение высшего образования

"Пермский национальный исследовательский политехнический университет"

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

Дисциплина: Основы алгоритмизации и программирования

Тема: Простое наследование. Принцип подстановки.

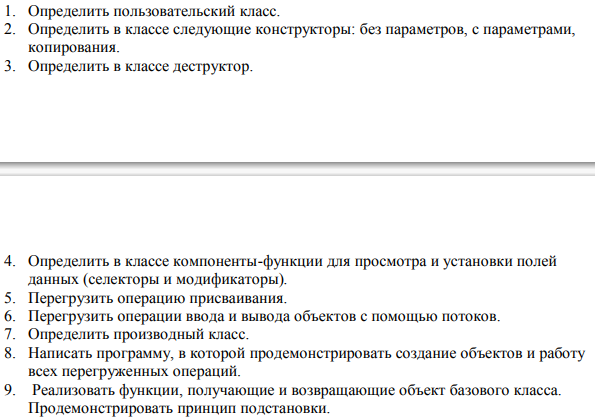
|  |  |
| --- | --- |
| Выполнил работу | |
| Студент группы РИС-22-1б | |
| Деревнин И.В. | |
|  | |
| Проверил работу | |
| Доцент кафедры ИТАС | |
| Полякова О.А. | |
|  | |

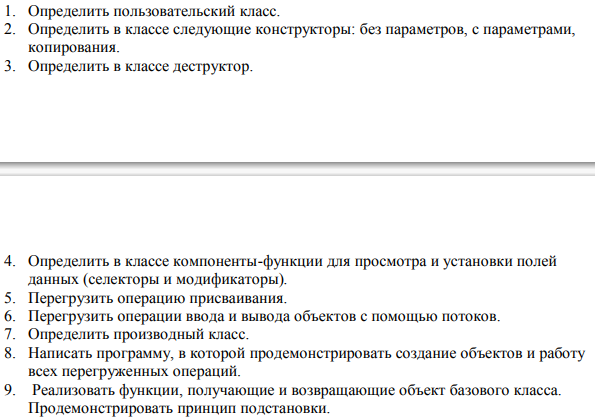
Пермь – 2023

**Анализ предметной области**

**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 операции: присваивания, вывода и ввода объектов с помощью потоков.
6. Определить производный класс.
7. Написать демонстрационную программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.
8. Реализовать функции, получающие и возвращающие объект базового класса. Продемонстрировать принцип подстановки.

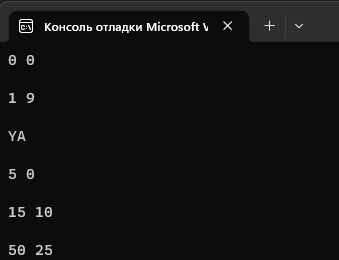




**Анализ задачи**

Класс Pair представляет из себя класс пары, который может существовать отдельно, но имеет очень ограниченный функционал. Класс Pair – родитель от которого можно унаследовать любой класс и добавить к нему недостающие характеристики. Класс rational – класс предок, который унаследован от класса Pair и расширяет его функционал. У него добавляются 2 поля –chislitel и znamenatel, а также метод, который проверяет знаменатель.

**Тестирование программы**

****

*Рис. 1 – Тестирование программы и классов pair и rational.*

# Ответы на контрольные 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

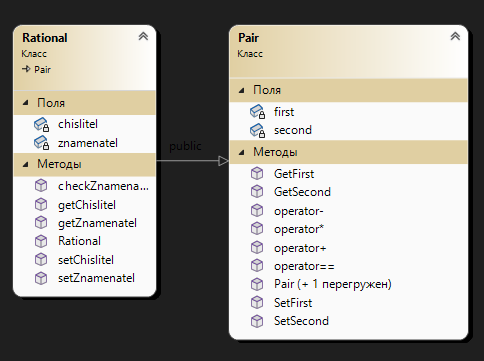
name = t.name;

age = t.age;

return \*this; }

**Приложения**

Приложение UML-диаграмма



Приложение Б – код программы

Main.cpp:

#include <iostream>

#include "Header.h"

using namespace std;

class Rational : public Pair {

private:

int chislitel;

int znamenatel;

public:

Rational(int ch = 0, int zn = 0);

void setChislitel(int ch);

void setZnamenatel(int zn);

int getChislitel();

int getZnamenatel();

void checkZnamenatel();

friend istream& operator>>(istream& is, Rational& s);

};

Rational::Rational(int f, int s) : Pair(f, s) {

chislitel = f;

znamenatel = s;

}

void Rational::setChislitel(int ch) {

chislitel = ch;

}

void Rational::setZnamenatel(int zn) {

znamenatel = zn;

}

int Rational::getChislitel() {

return chislitel;

}

int Rational::getZnamenatel() {

return znamenatel;

}

void Rational::checkZnamenatel() {

if (znamenatel < 4) {

cout << "Оценка студента неудовлетворительная!" << endl;

}

else {

cout << "Оценка студента удовлетворительная!" << endl;

}

}

istream& operator>>(istream& is, Rational& s) {

int ch, zn;

cout << "Введите числитель: ";

is >> ch;

s.SetFirst(ch);

cout << "Введите знаменатель: ";

is >> zn;

s.SetSecond(zn);

return is;

}

Rational operator- (const Rational& ch, const Rational& zn) {

return Rational(ch - zn);

}

Rational operator+ (const Rational& ch, const Rational& zn) {

return Rational(ch + zn);

}

Rational operator\* (const Rational& ch, const Rational& zn) {

return Rational(ch + zn);

}

int main() {

Pair a(1, 3);

Pair b(1, 3);

Pair c = a - b;

cout << c.GetFirst();

cout << " ";

cout << c.GetSecond();

cout << "\n\n";

Pair d = a \* b;

cout << d.GetFirst();

cout << " ";

cout << d.GetSecond();

cout << "\n\n";

if (a == b) cout << "YA";

else cout << "NU";

cout << "\n\n";

Rational chislo1(10, 5);

Rational chislo2(5, 5);

Rational chislo3(chislo1.getChislitel() - chislo2.getChislitel(), chislo1.getZnamenatel() - chislo2.getZnamenatel());

cout << chislo3.getChislitel() << " " << chislo3.getZnamenatel();

cout << "\n\n";

Rational chislo4(chislo1.getChislitel() + chislo2.getChislitel(), chislo1.getZnamenatel() + chislo2.getZnamenatel());

cout << chislo4.getChislitel() << " " << chislo4.getZnamenatel();

cout << "\n\n";

Rational chislo5(chislo1.getChislitel() \* chislo2.getChislitel(), chislo1.getZnamenatel() \* chislo2.getZnamenatel());

cout << chislo5.getChislitel() << " " << chislo5.getZnamenatel();

return 0;

}

pair.h:

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Pair {

private:

int first;

int second;

public:

Pair()

{

}

Pair(int f, int s);

void SetFirst(int f);

void SetSecond(int s);

int GetFirst();

int GetSecond();

Pair operator+(const Pair& a);

Pair operator\*(const Pair&);

Pair operator-(const Pair&);

bool operator==(const Pair&);

};

pair.cpp:

#include "Header.h"

#include <iostream>

using namespace std;

Pair::Pair(int f, int s) {

first = f;

second = s;

}

void Pair::SetFirst(int f) {

first = f;

}

void Pair::SetSecond(int s) {

second = s;

}

int Pair::GetFirst() {

return first;

}

int Pair::GetSecond() {

return second;

}

Pair Pair::operator+(const Pair& a) {

Pair t;

t.first = first + a.first;

t.second = second + a.second;

return t;

}

Pair Pair :: operator\*(const Pair& a) {

Pair t;

t.first = first \* a.first;

t.second = second \* a.second;

return t;

}

Pair Pair :: operator-(const Pair& a) {

Pair t;

t.first = first - a.first;

t.second = second - a.second;

return t;

}

bool Pair :: operator==(const Pair& a)

{

if ((first == a.first) && (second == a.second)) return true;

else return false;

}