Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет Кафедра «Информационные технологии и автоматизированные системы» направление подготовки: 09.03.04 - «Программная инженерия»

**«Лабораторная работа №4**

**"Простое наследование. Принцип подстановки."»**

Выполнил студент гр. РИС-24-3б Носков Егор Михайлович

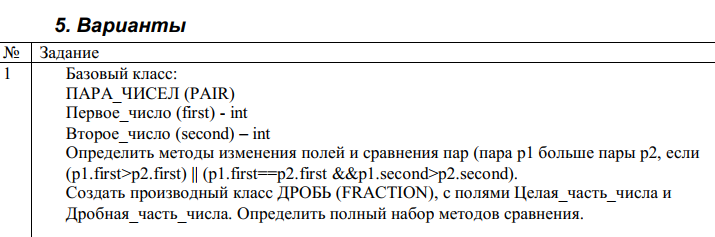
Проверил: Доцент кафедры ИТАС Ольга Андреевна Полякова

|  |  |
| --- | --- |
|  | |
| (оценка) (подпись) | |
|  |
| (дата) |

Г. Пермь, 2024

**Постановка задачи**

**Вариант 1:**

****

**Программа на языке C++**

Pair.h:

#pragma once

#include <iostream>

using namespace std;

class Pair

{

protected:

int first;

int second;

public:

Pair(void);

Pair(int, int);

Pair(const Pair&);

virtual ~Pair(void);

int get\_first() { return first; }

int get\_second() { return second; }

void set\_first(int);

void set\_second(int);

bool operator>(const Pair&);

bool operator==(const Pair&);

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

friend std::ostream& operator<<(ostream& out, const Pair& A);

void printinfo(const Pair&);

};

Pair.cpp:

#include "Pair.h"

Pair::Pair(void)

{

first=0;

second = 0;

}

Pair::~Pair(void) {}

Pair::Pair(int A, int B) { first = A; second = B; }

Pair::Pair(const Pair& pair) { this->first = pair.first; this->second = pair.second; }

void Pair::set\_first(int A) { first = A; }

void Pair::set\_second(int A) { second = A; }

bool Pair::operator>(const Pair& A) {if ((this->first > A.first) || ((this->first == A.first) && this->second > A.second)) return 1; else return 0;}

bool Pair::operator==(const Pair& A){ if ((this->first == A.first) && (this->second == A.second)) return 1; else return 0; }

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

cout << "Введите первое число: "; in >> A.first;

cout << "Введите второе число: "; in >> A.second;

return in;

}

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

out << "Пара чисел:" << endl;

out << "Первое число: " << A.first << endl;

out << "Второе число: " << A.second << endl;

return out;

}

void Pair::printinfo(const Pair& A) {cout << A << endl;}

Fraction.h:

#pragma once

#include "Pair.h"

class Fraction :

public Pair

{

protected:

int integerpart;

int fraction;

public:

Fraction(void);

~Fraction(void);

Fraction(int, int, int, int);

Fraction(const Fraction&);

int get\_integerpart() { return integerpart; }

int get\_fraction() { return fraction; }

void set\_integerpart(int);

void set\_fraction(int);

bool operator>(const Fraction&);

bool operator==(const Fraction&);

friend istream& operator>>(istream& in, Fraction& A);

friend std::ostream& operator<<(ostream& out, const Fraction& A);

void printinfo(const Fraction&);

};

Fraction.cpp:

#include "Fraction.h"

Fraction::Fraction(void) : Pair() { integerpart = 0; fraction = 0; }

Fraction::~Fraction(void) {}

Fraction::Fraction(int A, int B, int C, int D) : Pair(A, B) { integerpart = C; fraction = D; }

Fraction::Fraction(const Fraction& A) { this->first = A.first; this->second = A.second; this->integerpart = A.integerpart; this->fraction = A.fraction; }

void Fraction::set\_integerpart(int A) { this->integerpart = A; }

void Fraction::set\_fraction(int A) { this->fraction = A; }

bool Fraction::operator>(const Fraction& A) { if ((this->integerpart > A.integerpart) || ((this->integerpart == A.integerpart) && this->fraction > A.fraction)) return 1; else return 0; }

bool Fraction::operator==(const Fraction& A) { if ((this->integerpart == A.integerpart) && (this->fraction == A.fraction)) return 1; else return 0; }

istream& operator>>(istream& in, Fraction& A) {

cout << "Введите первое число: "; in >> A.first;

cout << "Введите второе число: "; in >> A.second;

cout << "Введите целую часть дроби: "; in >> A.integerpart;

cout << "Введите дробную часть: "; in >> A.fraction;

return in;

}

ostream& operator<<(ostream& out, const Fraction& A) {

out << "Пара чисел:" << endl;

out << "Первое число: " << A.first << endl;

out << "Второе число: " << A.second << endl;

out << "Дробь: " << A.integerpart << "." << A.fraction << endl;

return out;

}

void Fraction::printinfo(const Fraction& A) { cout << A << endl; }

Pair\_main.cpp:

#include <iostream>

#include "Pair.h"

#include "Fraction.h"

using namespace std;

void f1(Pair& A) { A.set\_first(55); cout << A << endl; }

Pair f2() { Fraction A(67, 80,90,34); return A; }

int main() {

setlocale(LC\_ALL, "RU");

Pair A;

cin >> A;

Pair B(56, 75);

cout << A << B;

if (A > B) {

cout << "Первая пара больше второй"<<endl;

}

else {

if (A == B) cout << "Первая пара равна второй"<<endl; else cout << "Первая пара меньше второй"<<endl;

}

Fraction C(50, 40, 30, 70);

Fraction D;

cin >> D;

cout << C << D;

if (C > D) {

cout << "Первая дробь больше второй"<<endl;

}

else {

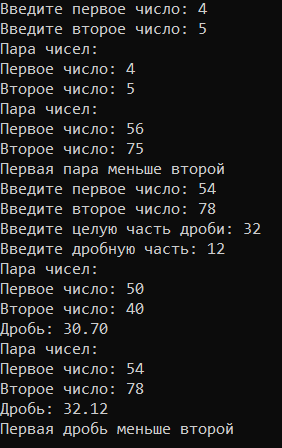
if (C == D)cout << "Первая дробь равна второй"<<endl; else cout << "Первая дробь меньше второй"<<endl;

}

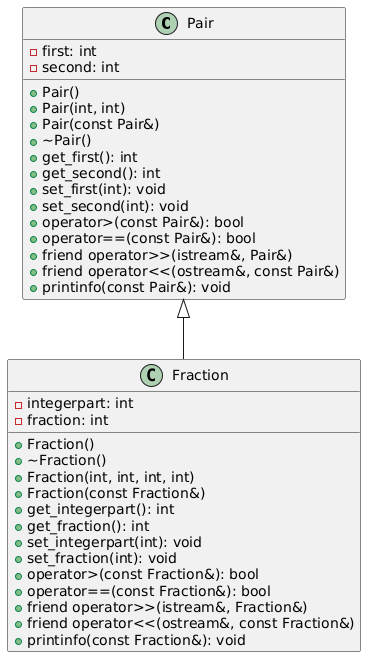
return 0;

}

Вывод программы:



**UML-диаграмма классов**



**Контрольные вопросы**

1. **Для чего используется механизм наследования?**  
   Механизм наследования используется для создания новых классов на основе существующих, что позволяет повторно использовать код, организовывать иерархии классов и реализовывать полиморфизм.
2. **Каким образом исследуются компоненты класса, описанные со спецификатором public?**  
   Компоненты класса, описанные со спецификатором public, доступны из любого места программы, включая другие классы и функции.
3. **Каким образом исследуются компоненты класса, описанные со спецификатором private?**  
   Компоненты класса, описанные со спецификатором private, доступны только внутри самого класса и его дружественных функций или классов.
4. **Каким образом исследуются компоненты класса, описанные со спецификатором protected?**  
   Компоненты класса, описанные со спецификатором protected, доступны внутри самого класса, его друзей, а также в производных классах.
5. **Каким образом описывается производный класс?**  
   Производный класс описывается с использованием ключевого слова class (или struct) и указания базового класса после двоеточия, например:

class Derived : public Base { ... };

1. **Наследуются ли конструкторы?**  
   Конструкторы не наследуются автоматически, но в C++11 и выше можно использовать ключевое слово using для наследования конструкторов базового класса.
2. **Наследуются ли деструкторы?**  
   Деструкторы не наследуются явно, но производный класс всегда вызывает деструктор базового класса при уничтожении объекта.
3. **В каком порядке конструируются объекты производных классов?**  
   Объекты производных классов конструируются в порядке от базового класса к производному (сверху вниз по иерархии).
4. **В каком порядке уничтожаются объекты производных классов?**  
   Объекты производных классов уничтожаются в порядке от производного класса к базовому (снизу вверх по иерархии).
5. **Что представляют собой виртуальные функции и механизм позднего связывания?**  
   Виртуальные функции позволяют переопределять методы в производных классах. Механизм позднего связывания (динамический полиморфизм) обеспечивает вызов правильной версии функции во время выполнения, а не компиляции.
6. **Могут ли быть виртуальными конструкторы? Деструкторы?**  
   Конструкторы не могут быть виртуальными. Деструкторы могут и часто должны быть виртуальными, особенно в базовых классах, чтобы обеспечить корректное удаление объектов производных классов.
7. **Наследуется ли спецификатор virtual?**  
   Да, если функция объявлена как виртуальная в базовом классе, она остаётся виртуальной во всех производных классах, даже без явного указания virtual.
8. **Какое отношение устанавливает между классами открытое наследование?**  
   Открытое наследование (public) устанавливает отношение "является" (is-a), где производный класс является разновидностью базового класса.
9. **Какое отношение устанавливает между классами закрытое наследование?**  
   Закрытое наследование (private) устанавливает отношение "реализовано посредством", где производный класс использует функциональность базового класса, но не раскрывает его интерфейс.
10. **В чем заключается принцип подстановки?**  
    Принцип подстановки (Liskov Substitution Principle) гласит, что объекты производного класса должны быть заменяемы на объекты базового класса без изменения корректности программы.
11. **Какие компонентные данные будет иметь объект x?**  
    Объект x класса Teacher будет содержать следующие компонентные данные:
    * age (унаследовано от Student, private)
    * name (унаследовано от Student, public)
    * post (унаследовано от Employee, protected)
    * stage (из класса Teacher, protected)
12. **Для классов Student, Employee и Teacher написать конструкторы без параметров.**

class Student {

public:

Student() : age(0), name("") {}

};

class Employee : public Student {

public:

Employee() : post("") {}

};

class Teacher : public Employee {

public:

Teacher() : stage(0) {}

};

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

class Student {

public:

Student(int a, const string& n) : age(a), name(n) {}

};

class Employee : public Student {

public:

Employee(int a, const string& n, const string& p) : Student(a, n), post(p) {}

};

class Teacher : public Employee {

public:

Teacher(int a, const string& n, const string& p, int s) : Employee(a, n, p), stage(s) {}

};

1. **Для классов Student, Employee и Teacher написать конструкторы копирования.**

class Student {

public:

Student(const Student& other) : age(other.age), name(other.name) {}

};

class Employee : public Student {

public:

Employee(const Employee& other) : Student(other), post(other.post) {}

};

class Teacher : public Employee {

public:

Teacher(const Teacher& other) : Employee(other), stage(other.stage) {}

};

1. **Для классов Student, Employee и Teacher определить операцию присваивания.**

class Student {

public:

Student& operator=(const Student& other) {

if (this != &other) {

age = other.age;

name = other.name;

}

return \*this;

}

};

class Employee : public Student {

public:

Employee& operator=(const Employee& other) {

if (this != &other) {

Student::operator=(other);

post = other.post;

}

return \*this;

}

};

class Teacher : public Employee {

public:

Teacher& operator=(const Teacher& other) {

if (this != &other) {

Employee::operator=(other);

stage = other.stage;

}

return \*this;

}

};

**Ссылка на репозиторий с отчетом Github**

****