Министерство науки и высшего образования РФ

Федеральное бюджетное государственное образовательное учреждение высшего профессионального образования

Пермский национальный исследовательский политехнический университет (ПНИПУ)

Факультет: Электротехнический (ЭТФ)

Направление: 09.03.01 «Информатика и вычислительная техника (Автоматизированные системы обработки информации и управления)»

Кафедра: «Информационные технологии и автоматизированных систем» (ИТАС)

Информатика

Лабораторная работа №5

Наследование. Виртуальные функции. Полиморфизм

Студент: Балтаев Э. А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, подпись)

Группа: АСУ-20-1бзу

Работу проверил:

доцент кафедры ИТАС

Полякова О. А.

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Подпись:\_\_\_\_\_\_\_\_\_\_\_

Пермь 2021г.

Постановка задачи

1. Определить абстрактный класс

2. Определить иерархию классов, в основе которой будет находиться абстрактный класс (см. лабораторную работу №4)

3. Определить класс Вектор, элементами которого будут указатели на объекты иерархии классов

4. Перегрузить для класса Вектор операцию вывода объектов с помощью потоков

5. В основной функции продемонстрировать перегруженные операции и полиморфизм Вектора.

Задание варианта

Базовый класс:

ПАРА\_ЧИСЕЛ (PAIR)

Первое\_число (first) – int

Второе\_число (second) – int

Определить методы проверки на равенство и операцию перемножения полей. Реализовать операцию вычитания пар по формуле (a,b)-(c-d)=(a-b, c-d).

Создать производный класс ПРОСТАЯ\_ДРОБЬ (RATIONAL), с полями Числитель и Знаменатель. Переопределить операцию вычитания и определить операции сложения и умножения простых дробей.

Описание классов

1) Базовый абстрактный

#pragma once

#include <iostream>

using namespace std;

class Object

{

public:

Object() {}

~Object() {}

virtual void Show() = 0;

};

2) Базовый

#pragma once

#include "Object.h"

class Pair :

public Object

{

protected:

int first;

int second;

public:

Pair() { first = second = 0; }

Pair(int f, int s) { first = f; second = s; }

Pair(const Pair& p) { first = p.first; second = p.second; }

~Pair() {}

int get\_first() { return first; }

int get\_second() { return second; }

void set\_first(int f) { first = f; }

void set\_second(int s) { second = s; }

void Show();

Pair& operator = (const Pair& p);

bool operator == (const Pair& p);

Pair operator \* (int val);

Pair operator - (const Pair& p);

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

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

};

3) Дочерний

#pragma once

#include "Pair.h"

class Rational :

public Pair

{

public:

Rational() : Pair() {}

Rational(int f, int s) : Pair() { set\_first(f); set\_second(s); }

Rational(const Rational& r) : Pair(r) {}

~Rational() {}

void set\_first(int f);

void set\_second(int s);

void Show();

Rational& operator= (const Rational& r);

Rational operator- (const Rational& r);

Rational operator+ (const Rational& r);

Rational operator\* (const Rational& r);

friend istream& operator>> (istream& in, Rational& r);

friend ostream& operator<< (ostream& out, const Rational& r);

};

4) Вектор

#pragma once

#include "Object.h"

class Vector

{

private:

Object\*\* beg;

int curr;

int size;

public:

Vector();

Vector(int s);

~Vector();

int get\_size() { return size; }

void add(Object\* obj);

friend ostream& operator<< (ostream& out, const Vector& v);

};

Определение компонентных функций

1) Базовый

#include "Pair.h"

void Pair::Show()

{

cout << \*this << endl;

}

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

{

if (&p == this)

return \*this;

first = p.first;

second = p.second;

return \*this;

}

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

{

if (first == p.first && second == p.second)

return true;

return false;

}

Pair Pair::operator\*(int val)

{

Pair temp(first \* val, second \* val);

return temp;

}

Pair Pair::operator-(const Pair& p)

{

Pair temp(first - second, p.first - p.second);

return temp;

}

istream& operator>>(istream& in, Pair& p)

{

cout << "first? "; in >> p.first;

cout << "second? "; in >> p.second;

return in;

}

ostream& operator<<(ostream& out, const Pair& p)

{

out << "(" << p.first << "," << p.second << ")";

return out;

}

2) Дочерний

#include "Rational.h"

void Rational::set\_first(int f)

{

first = f;

}

void Rational::set\_second(int s)

{

if (s == 0) {

cout << "Ошибка знаменателя, равен 0" << endl;

second = 1;

}

else

second = s;

}

void Rational::Show()

{

cout << \*this << endl;

}

Rational& Rational::operator=(const Rational& r)

{

if (&r == this)

return \*this;

first = r.first;

second = r.second;

return \*this;

}

Rational Rational::operator-(const Rational& r)

{

Rational temp;

// если знаменатели равны)

if (second == r.second) {

temp.first = first - r.first;

temp.second = second;

}

else {

temp.second = second \* r.second;

temp.first = first \* r.second - r.first \* second;

}

return temp;

}

Rational Rational::operator+(const Rational& r)

{

Rational temp;

// если знаменатели равны)

if (second == r.second) {

temp.first = first + r.first;

temp.second = second;

}

else {

temp.second = second \* r.second;

temp.first = first \* r.second + r.first \* second;

}

return temp;

}

Rational Rational::operator\*(const Rational& r)

{

Rational temp;

temp.first = first \* r.first;

temp.second = second \* r.second;

return temp;

}

istream& operator>>(istream& in, Rational& r)

{

int f, s;

cout << "num? "; in >> f; r.set\_first(f);

cout << "den? "; in >> s; r.set\_second(s);

return in;

}

ostream& operator<<(ostream& out, const Rational& r)

{

if (r.first > 0 && r.second || r.first < 0 && r.second < 0) {

out << abs(r.first) << "/" << abs(r.second);

}

else {

if (r.first < 0 && r.second > 0)

out << r.first << "/" << r.second;

else if (r.first > 0 && r.second < 0)

out << "-" << r.first << "/" << abs(r.second);

}

return out;

}

3) Вектор

#include "Vector.h"

Vector::Vector()

{

size = 10;

beg = new Object \* [size];

curr = 0;

}

Vector::Vector(int s)

{

size = s;

beg = new Object \* [size];

curr = 0;

}

Vector::~Vector()

{

delete[] beg;

}

void Vector::add(Object\* obj)

{

if (curr == size) {

cout << "Ошибка: заполнено" << endl;

return;

}

beg[curr] = obj;

curr++;

}

ostream& operator<<(ostream& out, const Vector& v)

{

if (v.curr == 0)

cout << "Пусто" << endl;

for (int i = 0; i < v.curr; i++)

v.beg[i]->Show();

return out;

}

Определение главного файла

#include <iostream>

#include "Vector.h"

#include "Rational.h"

int main()

{

system("chcp 1251");

cout << "\nСоздадим пары\n";

Pair p1(4, 5);

Pair p2;

cin >> p2;

cout << "\nСоздадим простые дроби\n";

Rational r1(11, 2);

Rational r2;

cin >> r2;

cout << "\nВектор: создаем пустой и добавляем 5 элементов\n";

Vector v(4);

cout << v << endl;

v.add(&r1);

v.add(&p1);

v.add(&p2);

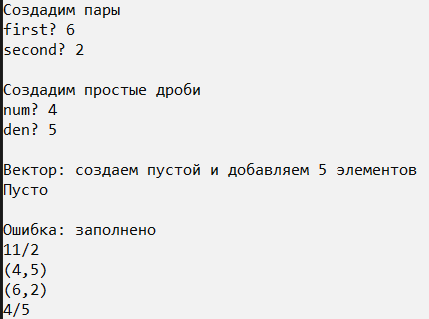
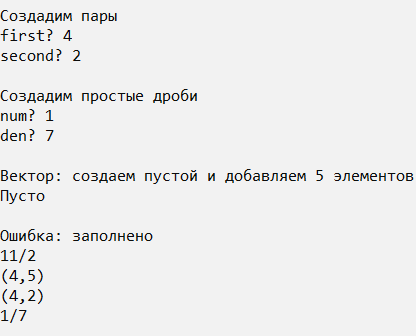
v.add(&r2);

v.add(new Pair(2, 1));

cout << v << endl;

}

Результаты работы программы

Ответы на контрольные вопросы

1) Чисто виртуальный – не имеет определения (=0), при этом класс будет являться абстрактным.

2) У которого хотя бы один чистый виртуальный метод

3) Чтобы создавать некоторый базовый класс для иерархии, невозможность создать объект данного класса и помещать дочерние классы в к классы-контейнеры

4) Данные функции будут работать со всеми типами в пределах иерархии

5) Принцип – просто берет часть родителя из дочернего, тогда как полиморфизм полностью определяет дочерний.

6)

Животное (абстрактный)

- Млекопитающие

- Водные

7)

Животное - перемещается() = 0

- Млекопитающие перемещается() { вывод(ходит по земле); }

- Водные перемещается() { вывод(плавает по воде); }

8) Например при нахождении в контейнере объектов разных типов иерархии, и выполнение полиморфной функции, то у каждого класса будет выполняться своя переорпределенная функция