Пермский Национальный Исследовательский  
Политехнический Университет

**Лабораторная работа № 5**

Информатика

за 2 семестр

Вариант № 1

Выполнил:

Студент группы РИС 20-1-бз

Акиев В.В.

20-ЭТФ-641

Проверила:

Доцент кафедры ИТАС

Полякова О.А.

2021

**Наследование. Виртуальные функции. Полиморфизм.**

1. **Цель задания**.
   1. Создание консольного приложения, состоящего из нескольких файлов в системепрограммированияVisualStudio.
   2. Создание иерархии классов с использованием простого наследования иабстрактногокласса.
   3. Изучениеполиморфизмаивиртуальныхметодов.
2. **Задание.**

Базовыйкласс:

ПАРА\_ЧИСЕЛ(PAIR)

Первое\_число (first) – int

Второе\_число(second) – int

Определить методы изменения полей и сравнения пар (пара p1 больше пары р2, если(p1.first>p2.first)||(p1.first==p2.first&&p1.second>p2.second).

Создать производный класс ДРОБЬ (FRACTION), с полями Целая\_часть\_числа и Дробная\_часть\_числа. Определить полный набор методов сравнения.

1. **Файл Лабораторная\_5.cpp**

#include <iostream>

#include "Pair.h"

#include "Fraction.h"

#include "Vector.h"

#include "object.h"

using namespace std;

void main()

{

Vector v(5);//вектор из 5 элементов

Pair a;//объект класса Car

cin >> a;

Fraction b;// объект класса Lorry

cin >> b;

Object\* p = &a;//ставим указатель на объект класса Car

v.Add(p);//добавляем объект в вектор

p = &b;//ставим указатель на объект класса Lorry

v.Add(p); //добавляем объект в вектор

cout << v;//вывод вектора

}

1. **Файл Vector.cpp**

#include "Vector.h"

//конструктор без параметров

Vector::Vector(void)

{

beg = 0;

size = 0;

cur = 0;

}

//деструктор

Vector::~Vector(void)

{

if (beg != 0)delete[] beg;

beg = 0;

}

//конструктор с параметрами

Vector::Vector(int n)

{

beg = new Object \* [n];

cur = 0;

size = n;

}

//добавление объекта, на который указывает указатель p в вектор

void Vector::Add(Object\* p)

{

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

//операция вывода

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

{

if (v.size == 0) out << "Empty" << endl;

Object\*\* p = v.beg;//указатель на указатель типа Object

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

{

(\*p)->Show();//вызов метода Show() (позднее связывание)

p++;//передвигаем указатель на следующий объект

}

return out;

}

1. **Файл object.cpp**

#include "object.h"

Object::Object(void)

{

}

Object::~Object(void)

{

}

1. **Файл object.h**

#pragma once

class Object

{

public:

Object(void);

public:

~Object(void);

virtual void Show() = 0;//чисто виртуальная функция

};

1. **Файл Vector.h**

#include "object.h"

#include <string>

#include <iostream>

using namespace std;

class Vector

{

public:

Vector(void);//конструктор без параметров

Vector(int);//конструктор копирования

public:

~Vector(void);//деструктор

void Add(Object\*);//добавление элемента в вектор

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

private:

Object\*\* beg;//указатель на первый элемент вектора

int size;//размер

int cur;//текущая позиция

};

1. **Файл Fraction.cpp**

#include "Fraction.h"

#include <math.h>;

#include <tchar.h>

#include <iostream>

#include <iomanip>

#include <sstream>

int fractional\_part\_as\_int(double number) {

int number\_of\_decimal\_places;

std::stringstream ss;

ss << std::setprecision(9) << number;

std::string strNum = ss.str();

size\_t pos = strNum.find('.');

if (pos != strNum.npos)

number\_of\_decimal\_places = strNum.size() - 1 - pos;

else

number\_of\_decimal\_places = 0;

double dummy;

double frac = modf(number, &dummy);

return round(frac \* pow(10, number\_of\_decimal\_places));

}

//не параметризованный конструктор

Fraction::Fraction(void) :Pair()

{

intPart = 0;

fractPart = 0;

}

//деструктор

Fraction::~Fraction()

{

}

//параметризованный конструктор

Fraction::Fraction(int f, int s, int ip, int fp) :Pair(f, s)

{

intPart = ip;

fractPart = fp;

}Fraction::Fraction(int f, int s, double num) : Pair(f, s)

{

intPart = (int)num;

fractPart = fractional\_part\_as\_int(num);

}

//клонирование

Fraction::Fraction(const Fraction& Fraction)

{

first = Fraction.first;

second = Fraction.second;

intPart = Fraction.intPart;

fractPart = Fraction.fractPart;

}

//модификатор first

void Fraction::Set\_intPart(int i)

{

intPart = i;

}

//модификатор second

void Fraction::Set\_fractPart(int f)

{

fractPart = f;

}

//оператор присваивания

Fraction& Fraction::operator=(const Fraction& Fraction)

{

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

first = Fraction.first;

second = Fraction.second;

intPart = Fraction.intPart;

fractPart = Fraction.fractPart;

return \*this;

}

//операторы сравнения

bool operator==(const Fraction& f1, const Fraction& f2)

{

return ((f1.first == f2.first && f1.second == f2.second) && (f1.intPart == f2.intPart && f1.fractPart == f2.fractPart));

}

bool operator!=(const Fraction& f1, const Fraction& f2)

{

return !(f1 == f2);

}

bool operator>(const Fraction& f1, const Fraction& f2)

{

return ((f1.first > f2.first) || (f1.first == f2.first && f1.second > f2.second)) && ((f1.intPart > f2.intPart) || (f1.intPart == f2.intPart && f1.fractPart > f2.fractPart));

}

bool operator>=(const Fraction& f1, const Fraction& f2)

{

return (f1 > f2) || (f1 == f2);

}

bool operator<(const Fraction& f1, const Fraction& f2)

{

return ((f1.first < f2.first) || (f1.first == f2.first && f1.second < f2.second)) && ((f1.intPart < f2.intPart) || (f1.intPart == f2.intPart && f1.fractPart < f2.fractPart));

}

bool operator<=(const Fraction& f1, const Fraction& f2)

{

return (f1 < f2) || (f1 == f2);

}

//оператор ввода

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

{

cout << "\nFirst: "; in >> Fraction.first;

cout << "\nSecond: "; in >> Fraction.second;

cout << "\nInt part: "; in >> Fraction.intPart;

cout << "\nFraction part: "; in >> Fraction.fractPart;

return in;

}

//оператор вывода

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

{

out << "\nFirst: " << Fraction.first;

out << "\nSecond: " << Fraction.second;

out << "\nFraction: " << Fraction.intPart << "." << Fraction.fractPart;

out << "\n";

return out;

}

void Fraction::Show()

{

cout << "\nFirst: " << first;

cout << "\nSecond: " << second;

cout << "\nFraction: " << intPart << "." << fractPart;

cout << "\n";

}

1. **Файл Pair.cpp**

#include "Pair.h"

//не параметризованный конструктор

Pair::Pair(void)

{

first = 0;

second = 0;

}

//деструктор

Pair::~Pair()

{

}

//параметризованный конструктор

Pair::Pair(int f, int s)

{

first = f;

second = s;

}

//клонирование

Pair::Pair(const Pair& pair)

{

first = pair.first;

second = pair.second;

}

//модификатор first

void Pair::Set\_first(int f)

{

first = f;

}

//модификатор second

void Pair::Set\_second(int s)

{

second = s;

}

//оператор присваивания

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

{

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

first = pair.first;

second = pair.second;

return \*this;

}

void Pair::Show()

{

cout << "\nFirst: " << first;

cout << "\nSecond: " << second;

cout << "\n";

}

//операторы сравнения

bool operator==(const Pair& p1, const Pair& p2)

{

return (p1.first == p2.first && p1.second == p2.second);

}

bool operator!=(const Pair& p1, const Pair& p2)

{

return !(p1 == p2);

}

bool operator>(const Pair& p1, const Pair& p2)

{

return (p1.first > p2.first) || (p1.first == p2.first && p1.second > p2.second);

}

bool operator>=(const Pair& p1, const Pair& p2)

{

return (p1 > p2) || (p1 == p2);

}

bool operator<(const Pair& p1, const Pair& p2)

{

return (p1.first < p2.first) || (p1.first == p2.first && p1.second < p2.second);

}

bool operator<=(const Pair& p1, const Pair& p2)

{

return (p1 < p2) || (p1 == p2);

}

//оператор ввода

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

{

cout << "\nFirst: "; in >> pair.first;

cout << "\nSecond: "; in >> pair.second;

return in;

}

//оператор вывода

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

{

out << "\nFirst: " << pair.first;

out << "\nSecond: " << pair.second;

out << "\n";

return out;

}

1. **Файл Pair.h**

#pragma once

#include <iostream>

#include "object.h"

using namespace std;

class Pair :

public Object

{

protected:

//атрибуты

int first;

int second;

public:

//деконструктор

virtual ~Pair();

//конструкторы

Pair(void);

Pair(int, int);

Pair(const Pair&);

//селекторы

int Get\_first() { return first; }

int Get\_second() { return second; }

//модификаторы

void Set\_first(int);

void Set\_second(int);

//перегрузка присваивания

Pair& operator=(const Pair&);

//перегрузка операторов сравнения

friend bool operator> (const Pair& p1, const Pair& p2);

friend bool operator<= (const Pair& p1, const Pair& p2);

friend bool operator< (const Pair& p1, const Pair& p2);

friend bool operator>= (const Pair& p1, const Pair& p2);

friend bool operator== (const Pair& p1, const Pair& p2);

friend bool operator!= (const Pair& p1, const Pair& p2);

//перегрузка ввода вывода

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

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

virtual void Show();

};

1. **Файл Fraction.h**

#include "Pair.h";

class Fraction :

public Pair

{

protected:

//атрибуты

int intPart;

int fractPart;

public:

//деконструктор

virtual ~Fraction();

//конструкторы

Fraction(void);

Fraction(int, int, int, int);

Fraction(int, int, double);

Fraction(const Fraction&);

//селекторы

int Get\_intPart() { return intPart; }

int Get\_fractPart() { return fractPart; }

//модификаторы

void Set\_intPart(int);

void Set\_fractPart(int);

void Set\_All(double);

//перегрузка присваивания

Fraction& operator=(const Fraction&);

//перегрузка операторов сравнения

friend bool operator> (const Fraction& p1, const Fraction& p2);

friend bool operator<= (const Fraction& p1, const Fraction& p2);

friend bool operator< (const Fraction& p1, const Fraction& p2);

friend bool operator>= (const Fraction& p1, const Fraction& p2);

friend bool operator== (const Fraction& p1, const Fraction& p2);

friend bool operator!= (const Fraction& p1, const Fraction& p2);

//перегрузка ввода вывода

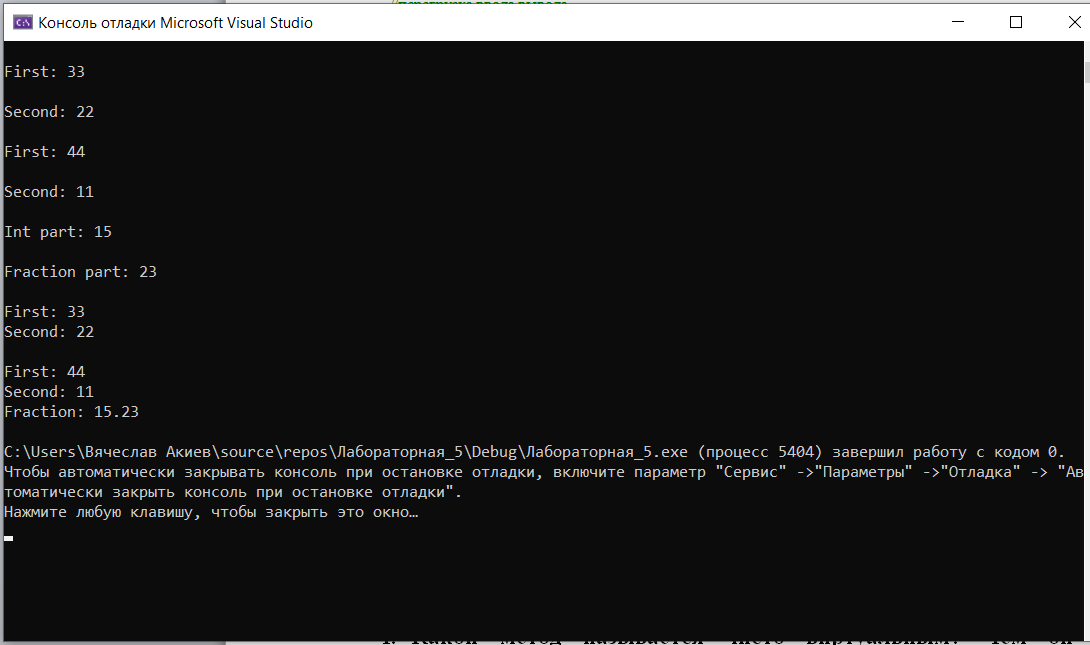
friend istream& operator>> (istream& in, Fraction& p);

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

virtual void Show();

};

1. **Результат работы программы**

****

1. **Контрольные вопросы.** 
   1. **Какой метод называется чисто виртуальным? Чем он отличается от виртуальногометода?**

Чисто виртуальный метод – метод содержащий признак = 0 вместо тела. Отличается от виртуального тем что при наследовании необходимо переопределять этот метод.

* 1. **Какой класс называетсяабстрактным?**

Класс называется абстрактным если содержит хотя бы 1 чисто виртуальный метод

* 1. **Для чего предназначены абстрактные классы?**

Абстрактные классы предназначены для представления общих понятий, которые предполагается конкретизировать в производных классах.

* 1. **Что такое полиморфные функции?**

Это функцию, параметром которой является указатель на абстрактный класс.

* 1. **Чем полиморфизм отличается от принципа подстановки?**

Возможностью работать с объектом любого типа в пределах одной иерархии

* 1. **Привести примеры иерархий с использованием абстрактных классов**.

Абстрактный класс 4-х угольник

класс выпуклый:4-х угольник

класс трапеция: выпуклый

класс параллелограмм: трапеция

Абстрактный класс животные

класс собака: животные

класс кошка: животные

класс птица: животные

* 1. **Привести примеры полиморфных функций.**

1. Перезагрузка функции – Когда их аргументы уникальны, вы можете использовать две основные функции с одинаковым названием в C++. Различные методы вызываются в зависимости от количества и типа параметров, и эта концепция известна как перегрузка функций.

2. Перегрузка операторов - похожа на перегрузку методов, поскольку она использует один и тот же знак, но разные операнды для разных методов операторов.

3. Переопределение функции - Базовый класс и его подклассы-потомки могут иметь методы с одинаковыми именами. Когда мы используем экземпляр из подкласса для вызова метода, запускается функция расширенного класса, а не родительского класса. В результате различные функции будут выполняться в зависимости от объекта, вызывающего метод. В C++ это называется переопределением метода.

4. Виртуальные функции - Если мы используем «указатель» ссылки на базовый класс для ссылки на объект производного класса, мы даже не сможем обойти методы в C++. В этих обстоятельствах использование виртуальных методов в родительском классе гарантирует возможность переопределения метода.

* 1. **В каких случаях используется механизм позднего связывания?**

Когда мы не можем заранее определить от какого класса из иерархии будет вызвана функция