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 №5

По теме **«Наследование. Виртуальные функции. Полиморфизм»**

Вариант №10

Выполнял:

студент группы РИС-24-1б

Морозова Н.С.

Проверял:

Доцент кафедры ИТАС

Полякова О.А.

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:

Базовый класс - ТРОЙКА\_ЧИСЕЛ (TRIAD)

Первое\_число (first) - int,

Второе\_число (second) – int,

Третье\_число (third) - int.

Определить методы изменения полей и увеличения полей на 1.

Создать производный класс DATE с полями год, месяц и число. Переопределить методы увеличения полей на 1 и определить метод увеличения даты на n дней.

Анализ задачи:

1. Создать файлы Object.h и Object.cpp для определения и описания базового класса Object, внутри класса создать чисто виртуальный метод Show().
2. Внутри производных классов Date и Traid переопределить функцию Show() для вывода данных (все остальные метода как в л/р 4).
3. Создать класс Vector в файле Vector.h и описать его методы в файле Vector.cpp, реализовать в этом классе перегрузку оператора вывода с помощью потоков.

Решение

Код

Файл Lab\_5.cpp

#include <iostream>

#include <Windows.h>

#include "Object.h"

#include "traid.h"

#include "Date.h"

#include "vector.h"

using namespace std;

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Vector v(3);

Traid tr;

cin >> tr;

Date d;

cin >> d;

cout << "Вывод через вектор" << endl;

Object \*ob = &tr;

v.Add(ob);

ob = &d;

v.Add(ob);

cout << v << endl;

tr.plus();

d.change();

cout << v << endl;

return 0;

}

Файл traid.h

#pragma once

#include "Object.h"

#include <iostream>

using namespace std;

class Traid:public Object

{

protected:

int first;

int second;

int third;

public:

Traid();

Traid(int, int, int);

Traid(const Traid&);

virtual ~Traid();

int Get\_first() { return first; }

int Get\_second() { return second; }

int Get\_third() { return third; }

void Set\_first(int F) { first = F; }

void Set\_second(int S) { second = S; }

void Set\_third(int T) { third = T; }

Traid& operator=(const Traid&);

friend istream& operator>>(istream& in, Traid& c);

friend ostream& operator<<(ostream& out, const Traid& c);

void Show();

void change(); //фунцкия изменения полей

void plus(); // прибавление 1

};

Файл traid.cpp

#include "traid.h"

#include <iostream>

Traid::Traid(void)

{

first = 0;

second = 0;

third = 0;

}

Traid::Traid(int F, int S, int T)

{

first = F;

second = S;

third = T;

}

Traid::Traid(const Traid& traid)

{

first = traid.first;

second = traid.second;

third = traid.third;

}

Traid::~Traid(void) { cout << "Деструктор" << endl; }

Traid& Traid::operator=(const Traid& t)

{

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

first = t.first;

third = t.third;

second = t.second;

return \*this;

}

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

cout << "first: ";

in >> t.first;

cout << "second: ";

in >> t.second;

cout << "third: ";

in >> t.third;

return in;

}

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

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

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

out << "\nthird: " << t.third;

out << endl;

return out;

}

void Traid:: change()

{//фунцкия изменения полей

int n, dat;

cout << "Какое поле поменять? \n(first - 1, second - 2, third - 3) ";

cin >> n;

cout << "Введите значение: ";

cin >> dat;

switch (n)

{

case 1: first = dat; break;

case 2: second = dat; break;

case 3: third = dat; break;

default: cout << "Ошибка!"; break;

}

}

void Traid:: plus()

{// прибавление 1

cout << "Прибавление 1" << endl;

first += 1;

second += 1;

third += 1;

}

void Traid:: Show()

{ //переопределение функции

cout << "\nfirst: " << first;

cout << "\nsecond: " << second;

cout << "\nthird: " << third;

cout << endl;

}

Файл Date.h

#pragma once

#include "traid.h"

#include <iostream>

using namespace std;

class Date: public Traid

{

protected:

int year;

int month;

int day;

public:

Date();

Date(int, int, int);

Date(const Date&);

~Date();

//селекторы

int Get\_year() { return year; }

int Get\_month() { return month; }

int Get\_day() { return day; }

//модификаторы

void Set\_year(int Y) { year = Y; }

void Set\_month(int M) { month = M; }

void Set\_day(int D) { day = D; }

Date& operator=(const Date&);

friend istream& operator>>(istream& in, Date& c);

friend ostream& operator<<(ostream& out, const Date& c);

void Show();

void change(); //фунцкия изменения полей

void plus(); // прибавление 1

};

Файл Date.cpp

#include "traid.h"

#include "Date.h"

#include <iostream>

Date::Date(void)

{

year = 0;

month = 0;

day = 0;

}

Date::Date(int Y, int M, int D) {

year = Y;

month = M;

month = D;

}

Date::Date(const Date& date) {

year = date.year;

month = date.month;

day = date.day;

}

Date::~Date(void) { cout << "Деструктор" << endl; }

Date& Date::operator=(const Date& d) {

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

year = d.year;

month = d.month;

day = d.day;

return \*this;

}

istream& operator>>(istream& in, Date& d) {

cout << "\nГод: ";

in >> d.year;

cout << "Месяц: ";

in >> d.month;

cout << "Число: ";

in >> d.day;

return in;

}

ostream& operator<<(ostream& out, const Date& d) {

out << "\nГод: " << d.year;

out << "\nМесяц: " << d.month;

out << "\nЧисло: " << d.day;

out << endl;

return out;

}

void Date:: change() {//фунцкия изменения полей

int n;

do

{

cout << "Сколько дней прибавить к дате? ";

cin >> n;

} while (n > 30);

day += n;

}

void Date:: plus() {// прибавление 1

cout << "Прибавление 1" << endl;

if (day + 1 == 32)

{

month += 1;

day = 1;

}

else day += 1;

if (month + 1 == 13)

{

month = 1;

year += 1;

}

else month += 1;

year += 1;

}

void Date::Show(){

cout << "\nГод: " << year;

cout << "\nМесяц: " << month;

cout << "\nЧисло: " << day;

cout << endl;

}

Файл Object.h

#pragma once

class Object{

public:

Object();

virtual ~Object();

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

};

Файл Object.cpp

#include "Object.h"

#include <iostream>

Object::Object() {} // реализация конструктора

Object::~Object() {} // реализация деструктора

Файл Vector.h

#pragma once

#include "object.h"

#include <string>

#include <iostream>

using namespace std;

class Vector{

private:

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

int size;//размер

int cur;//текущая позиция

public:

Vector();//конструктор без параметров

Vector(int);//конструктор копирования

~Vector();//деструктор

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

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

};

Файл Vector.cpp

#include "Vector.h"

Vector::Vector(void) { beg = 0; size = 0; cur = 0; }//конструктор без параметров

Vector::~Vector(void) { if (beg != 0) delete[] beg; beg = 0; }//деструктор

Vector::Vector(int n) {//конструктор с параметрами

beg = new Object \* [n];

cur = 0;

size = n;

}

void Vector::Add(Object \*p){//добавление объекта

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

ostream& operator<<(ostream& out, const Vector& v){//операция вывода

if (v.size == 0)

out << "Empty" << endl;

Object\*\* p = v.beg;

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

(\*p)->Show();//позднее связывание

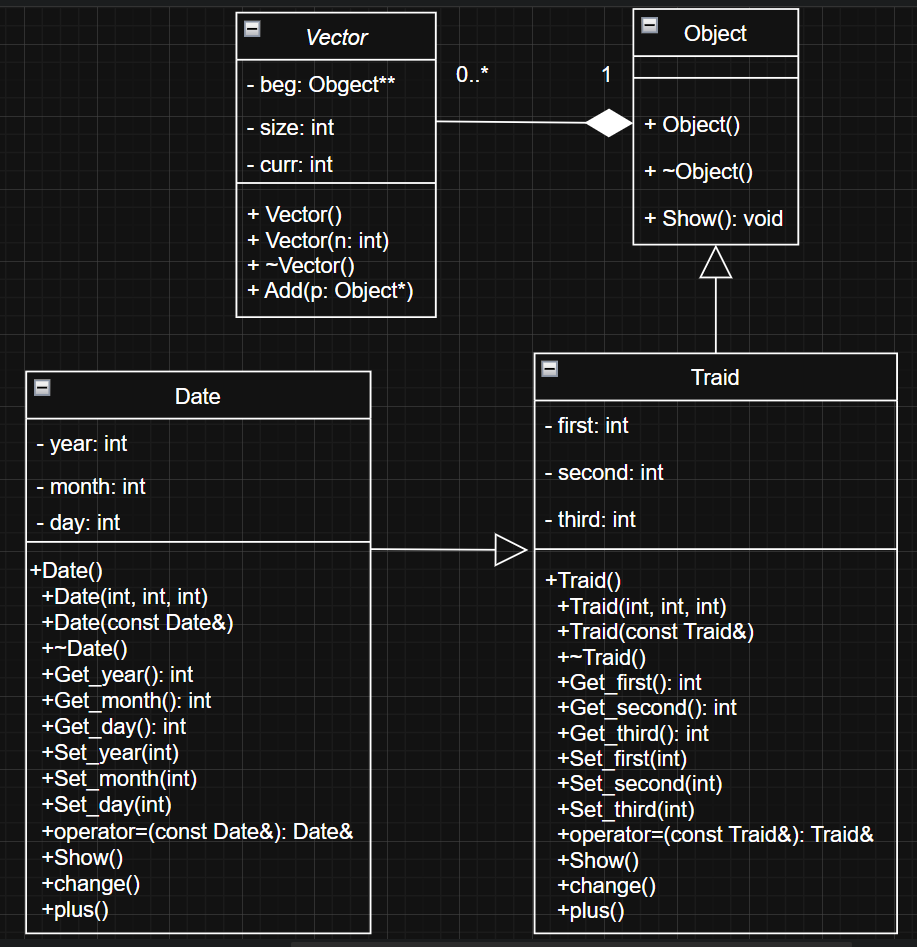
p++;

}

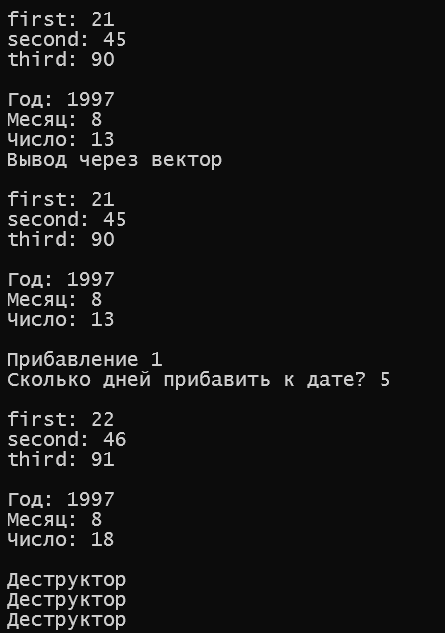
return out;

}

UML-диаграмма



Пример работы программы:



Контрольные вопросы:

1. Какой метод называется чисто виртуальным?

Чисто виртуальный метод — это метод, который объявлен в базовом классе, но не имеет реализации в нём. Он обозначается с помощью = 0 в конце объявления.

1. Чем он отличается от виртуального метода?

Виртуальный метод имеет реализацию в базовом классе и может быть переопределен в производных классах.

Чисто виртуальный метод не имеет реализации в базовом классе и обязывает производные классы предоставить свою реализацию.

1. Какой класс называется абстрактным?

Абстрактный класс — это класс, содержащий хотя бы один чисто виртуальный метод. Создавать объекты абстрактного класса нельзя, он предназначен только для наследования.

1. Для чего предназначены абстрактные классы?

Определяют интерфейс для производных классов. Обеспечивают полиморфное поведение через виртуальные функции. Запрещают создание объектов самого абстрактного класса, оставляя реализацию наследникам.

1. Что такое полиморфные функции?

Полиморфные функции — это функции, которые могут работать с разными типами данных через единый интерфейс. В C++ полиморфизм достигается через виртуальные функции и указатели/ссылки на базовый класс.

1. Чем полиморфизм отличается от принципа подстановки?

Принцип подстановки (LSP) — это правило, что объекты производного класса должны заменять объекты базового класса без изменения корректности программы.

Полиморфизм — это механизм, позволяющий вызывать разные реализации методов в зависимости от типа объекта.

LSP — это принцип проектирования, а полиморфизм — это механизм языка.

1. Привести примеры иерархий с использованием абстрактных классов.
2. class Shape { // абстрактный класс

public:

virtual double area() const = 0;

virtual ~Shape() = default;

};

class Circle : public Shape {

double radius;

public:

Circle(double r) : radius(r) {}

double area() const override { return 3.14 \* radius \* radius; }

};

class Square : public Shape {

double side;

public:

Square(double s) : side(s) {}

double area() const override { return side \* side; }

};

1. class Animal {

public:

virtual void makeSound() const = 0;

};

class Dog : public Animal {

public:

void makeSound() const override { std::cout << "Woof!\n"; }

};

class Cat : public Animal {

public:

void makeSound() const override { std::cout << "Meow!\n"; }

};

1. Привести примеры полиморфных функций.

void printArea(const Shape& shape) {

cout << "Area: " << shape.area() << "\n";

}

int main() {

Circle c(5);

Square s(4);

printArea(c); // Полиморфизм: вызовется Circle::area()

printArea(s); // Полиморфизм: вызовется Square::area()

}

1. В каких случаях используется механизм позднего связывания?

Позднее связывание (динамическая диспетчеризация) используется:

* При вызове виртуальных функций через указатель/ссылку на базовый класс.
* Когда нужно, чтобы вызывалась реализация производного класса, а не базового.
* В полиморфных иерархиях, где поведение метода зависит от типа объекта.