Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

Тема: «Лабораторная работа №5»

Выполнил

Студент группы РИС-22-2б

Прядеин И.А.

Проверил доцент кафедры

ИТАС

Полякова О. А.

Пермь 2023

Постановка задачи

1. Определить абстрактный класс.
2. Определить иерархию классов, в основе которой будет находиться абстрактный класс.
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.

**Вариант 8:**

Базовый класс:

ТРОЙКА\_ЧИСЕЛ (TRIAD)

Первое\_число (first) - int

Второе\_число (second) - int

Третье\_число (third) - int

Определить методы изменения полей и сравнения триады.

Создать производный класс DATE с полями год, месяц и число. Определить полный набор сравнения дат.

**Исходный код программы:**

**Файл “Object.h”:**

#pragma once

class Object {

public:

virtual void Show() =0;

};

**Файл “Triad.h”:**

#pragma once

#include "Object.h"

#include <iostream>

using namespace std;

class Triad : public Object {

public:

Triad();

virtual ~Triad();

void Show();

Triad(int, int, int);

Triad(const Triad&);

int get\_first() {

return first;

}

int get\_second() {

return second;

}

int get\_third() {

return third;

}

void set\_first(int first) {

this->first = first;

}

void set\_second(int second) {

this->second = second;

}

void set\_third(int third) {

this->third = third;

}

Triad& operator =(const Triad&);

friend istream& operator >>(istream&in, Triad&other);

friend ostream& operator <<(ostream&out, Triad&other);

protected:

int first, second, third;

};

**Файл “Triad.cpp”:**

#include "Triad.h"

#include <iostream>

using namespace std;

Triad::Triad() {

first = 0;

second = 0;

third = 0;

}

Triad::Triad(int f, int s, int t) {

first = f;

second = s;

third = t;

}

Triad::~Triad() {

}

Triad::Triad(const Triad& triad) {

first = triad.first;

second = triad.second;

third = triad.third;

}

Triad& Triad::operator =(const Triad& other) {

if (&other == this)

return \*this;

first = other.first;

second = other.second;

third = other.third;

return \*this;

}

istream& operator >>(istream&in, Triad& other) {

cout << "First: ";

in >> other.first;

cout << "Second: ";

in >> other.second;

cout << "Third: ";

in >> other.third;

return in;

}

ostream& operator <<(ostream&out, Triad& other) {

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

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

out << "\nThird : " << other.third;

out << "\n";

return out;

}

void Triad::Show() {

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

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

cout << "\nThird : " << third;

cout << "\n";

}

**Файл “Date.h”:**

#pragma once

#include "Triad.cpp"

class Date : public Triad {

public:

Date();

~Date();

void Show();

Date(int, int, int);

Date(const Date &);

Date& operator =(const Date&);

bool operator ==(const Date&);

bool operator !=(const Date&);

bool operator >(const Date&);

bool operator <(const Date&);

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

friend ostream& operator<<(ostream&out, Date&o);

};

**Файл “Date.cpp”:**

#include "Date.h"

Date::Date() : Triad() {

}

Date::~Date() {

}

Date::Date(int f, int s, int t) : Triad(f, s, t) {

}

Date::Date(const Date& other) {

first = other.first;

second = other.second;

third = other.third;

}

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

if (&other == this)

return \*this;

first = other.first;

second = other.second;

third = other.third;

return \*this;

}

bool Date::operator ==(const Date& other) {

return (first == other.first && second == other.second && third == other.third);

}

bool Date::operator !=(const Date& other) {

return !(first == other.first && second == other.second && third == other.third);

}

bool Date::operator >(const Date& other) {

if (third > other.third)

return true;

else

return false;

if (second > other.second)

return true;

else

return false;

if (first > other.first)

return true;

else

return false;

}

bool Date::operator <(const Date& other) {

if (third < other.third)

return true;

else

return false;

if (second < other.second)

return true;

else

return false;

if (first < other.first)

return true;

else

return false;

}

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

cout << "\nDay: ";

cin >> d.first;

while (d.first > 31) {

cout << "this day does not exist, please enter a new value: ";

cin >> d.first;

}

cout << "Month: ";

cin >> d.second;

while (d.second > 12) {

cout << "this month does not exist, please enter a new value: ";

cin >> d.second;

}

cout << "Year: ";

cin >> d.third;

return in;

}

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

cout << "Date: " << d.first << "." << d.second << "." << d.third;

return out;

}

void Date::Show() {

cout << "Date: " << first << "." << second << "." << third << endl;

}

**Файл “Vector.h”:**

#pragma once

#include "Object.h"

#include <iostream>

using namespace std;

class Vector {

public:

Vector();

Vector(int);

~Vector();

void Add(Object \*);

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

private:

Object\*\* beg;

int size;

int cur;

};

**Файл “Vector.cpp”:**

#include "Vector.h"

Vector::Vector() {

beg = 0;

size = 0;

cur = 0;

}

Vector::Vector(int n) {

beg = new Object\*[n];

cur = 0;

size = n;

}

Vector::~Vector() {

if (beg != 0)

delete[] beg;

beg = 0;

}

void Vector::Add(Object \*p) {

if (cur < size) {

beg[cur] = p;

cur++;

}

}

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

if (v.size == 0)

out << "Empty" << endl;

Object \*\*p = v.beg;

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

cout << i << ": ";

(\*p)->Show();

p++;

}

return out;

}

**Файл “Lab\_5.cpp”:**

#include "Object.h"

#include "Date.cpp"

#include "Vector.cpp"

#include <iostream>

using namespace std;

int main() {

Vector v(5);

Triad a;

cin >> a;

Date b;

cin >> b;

Object \*p = &a;

v.Add(p);

p = &b;

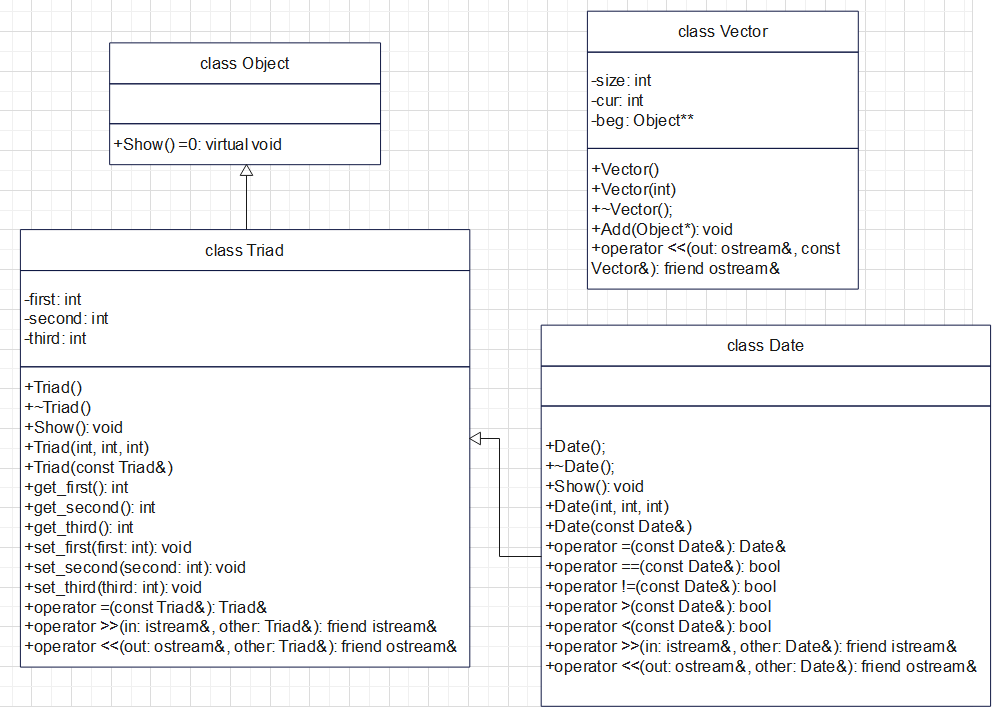
v.Add(p);

cout << v;

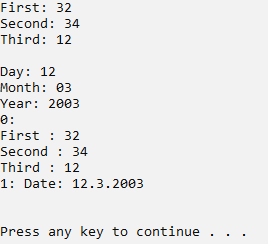
return 0;

}

**UML диаграмма:**



**Скриншот результата выполнения программы**



**Ответы на вопросы**

1. Какой метод называется чисто виртуальным? Чем он отличается от виртуального метода?

**Ответ:** Чистые виртуальные функции - функции, которые не имеют определения.

Обычная виртуальная функция превращается в чистую, если в конец объявления виртуальной функции добавить "= 0":

class Object {

public:

virtual void output() = 0;

};

1. Какой класс называется абстрактным?

**Ответ:** Абстрактные класс - класс, который содержит или наследует хотя бы одну чистую виртуальную функцию без переопределения.

1. Для чего предназначены абстрактные классы?

**Ответ:** Абстрактные классы используются в качестве обобщённых концепций, на основе которых можно создавать более конкретные производные классы.

1. Что такое полиморфные функции?

**Ответ:** Полиморфные функции - функции, способные обрабатывать данные разных типов.

1. Чем полиморфизм отличается от принципа подстановки?

**Ответ:** Принцип подстановки гарантирует, что объекты могут заменять друг друга без нарушения работы программы, а полиморфизм позволяет использовать различные объекты с одним интерфейсом и получать разное поведение в зависимости от типа данных.

1. Привести примеры иерархий с использованием абстрактных классов.

**Ответ:**

class Book {

protected:

string title;

public:

Book(string t) {

title = t;

}

virtual void Show() {

cout << “Title: “ << title << endl;

}

};

class Page : public Book {

protected:

int chapter;

public:

Page(string t, int c) : Book(t) {

chapter = c;

}

void Show() {

cout << “Title: “ << title << endl;

cout << “Chapter: “ << chapter << endl;

}

};

1. Привести примеры полиморфных функций.
2. В каких случаях используется механизм позднего связывания?

**Ответ:** Механизм позднего связывания используется в ООП в тех случаях, когда необходимо обеспечить гибкость и расширяемость программного кода за счет возможности взаимозаменяемости объектов.