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

Лабораторная работа №5

Наследование. Виртуальные функции. Полиморфизм.

Выполнил

Студент группы РИС-22-2б

Вековшинин Д. А.

Проверила

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 (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

Вариант 10: Базовый класс:

ТРОЙКА\_ЧИСЕЛ (TRIAD)

Первое\_число (first) - int

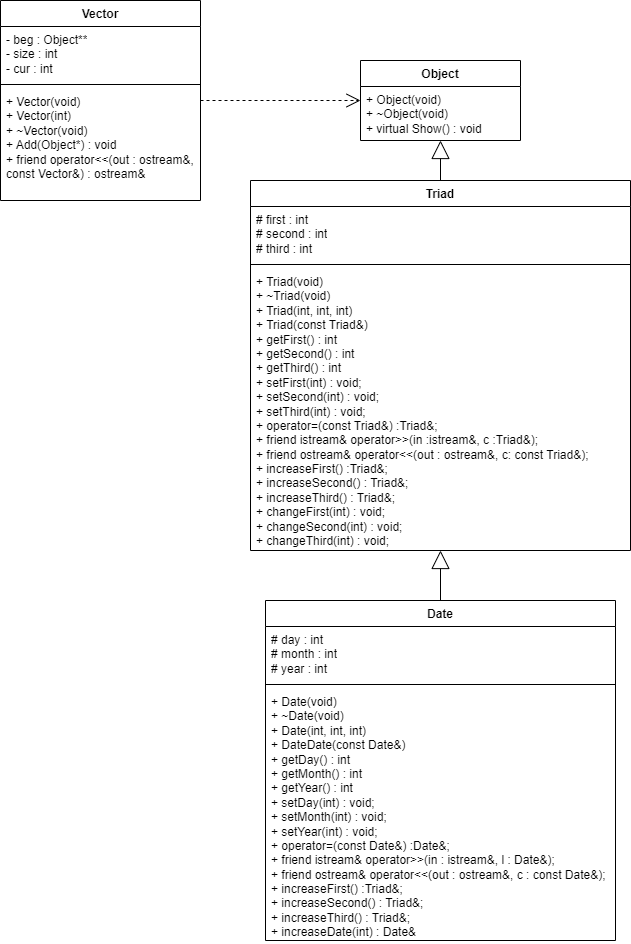
Второе\_число (second) – int

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.

UML-диаграмма



Определение компонентных функций класса Triad

//конструктор без параметров

Triad::Triad(void)

{

first = 0;

second = 0;

third = 0;

}

//деструктор

Triad::~Triad(void)

{

}

//конструктор с параметрами

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

{

first = F;

second = S;

third = T;

}

//конструктор копирования

Triad::Triad(const Triad& car)

{

first = car.first;

second = car.second;

third = car.third;

}

//модификаторы

void Triad::setFirst(int F)

{

first = F;

}

void Triad::setSecond(int S)

{

second = S;

}

void Triad::setThird(int T)

{

third = T;

}

//перегрузка операции присваивания

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

{

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

first = t.first;

second = t.second;

third = t.third;

return \*this;

}

//методы увеличения полей на 1

Triad& Triad::increaseFirst()

{

first++;

return \*this;

}

Triad& Triad::increaseSecond()

{

second++;

return \*this;

}

Triad& Triad::increaseThird()

{

third++;

return \*this;

}

//методы изменения полей

void Triad::changeFirst(int F)

{

first=F;

}

void Triad::changeSecond(int S)

{

second=S;

}

void Triad::changeThird(int T)

{

third=T;

}

void Triad::Show()

{

cout << "\nFIRST : " << first;

cout << "\nSECOND : " << second;

cout << "\nTHIRD : " << third;

cout << "\n";

}

Определение компонентных функций класса Date

//конструктор без параметров

Date::Date(void) :Triad()

{

day = 0;

year = 0;

month = 0;

}

//дестрктор

Date::~Date(void)

{

}

//конструктор с параметрами

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

{

day = D;

month = M;

year = Y;

}

//конструктор копирования

Date::Date(const Date& D)

{

day = D.day;

month = D.month;

year = D.year;

}

//модификаторы

void Date::setDay(int D)

{

day = D;

}

void Date::setMonth(int M)

{

day = M;

}

void Date::setYear(int Y)

{

day = Y;

}

//оперция присваивания

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

{

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

day = d.day;

month = d.month;

year = d.year;

return \*this;

}

//методы увеличения полей на 1

Date& Date::increaseFirst()

{

day++;

return \*this;

}

Date& Date::increaseSecond()

{

month++;

return \*this;

}

Date& Date::increaseThird()

{

year++;

return \*this;

}

//метод увеличения даты на n дней

Date& Date::increaseDate(int n)

{

day+=n;

return \*this;

}

void Date::Show()

{

cout << "\nDAY : " << day;

cout << "\nMONTH : " << month;

cout << "\nYEAR : " << year;

cout << "\n";

}

Определение глобальных функций

//глобальная функция для ввода

istream& operator>>(istream& in, Triad& t)

{

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

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

cout << "\nThird: "; in >> t.third;

return in;

}

//глобальная функция для вывода

ostream& operator<<(ostream& out, const Triad& t)

{

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

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

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

out << "\n";

return out;

}

//операция ввода

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

{

cout << "\nDay: "; in >> d.day;

cout << "\nMonth: "; in >> d.month;

cout << "\nYear: "; in >> d.year;

return in;

}

//операция вывода

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

{

out << "\nDAY : " << d.day;

out << "\nMONTH : " << d.month;

out << "\nYEAR : " << d.year;

out << "\n";

return out;

}

void f1(Triad& c)

{

c.setFirst(99999);

cout << c;

}

Triad f2()

{

Date d(10, 5, 2001);

return d;

}

Функция main()

void main()

{

Vector v(5);//вектор из 5 элементов

Triad a;//объект класса Car

cin >> a;

Date b;// объект класса Lorry

cin >> b;

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

v.Add(p);//добавляем объект в вектор

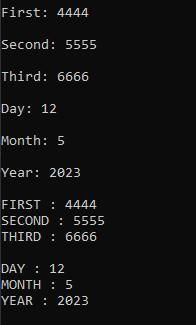
p = &b;//ставим указатель на объект класса Lorry

v.Add(p); //добавляем объект в вектор

cout << v;//вывод вектора

}

Результаты работы программы



Ответы на контрольные вопросы

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

Чисто виртуальный метод содержит признак = 0 вместо тела, например:

virtual void f(int) = 0;

Чисто виртуальный метод должен переопределяться в производном классе (возможно, опять как чисто виртуальный).

1. Какой класс называется абстрактным?

Класс, содержащий хотя бы один чисто виртуальный метод, называется абстрактным.

class Object

{

public:

Object(void);

~Object(void);

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

};

1. Для чего предназначены абстрактные классы?

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

class Object

{

public:

Object(void);

~Object(void);

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

};

Функция Show конкретизируется в классе Triad:

void Triad::Show()

{

cout << "\nFIRST : " << first;

cout << "\nSECOND : " << second;

cout << "\nTHIRD : " << third;

cout << "\n";

}

1. Что такое полиморфные функции?

Функции, работающие с объектом любого типа в пределах одной иерархии.

void Vector::Add(Object\* p)

{

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

1. Чем полиморфизм отличается от принципа подстановки?

Принцип подстановки заключается в том, что везде, где может быть использован объект базового класса, вместо него разрешается использовать объект производного класса. Полиморфизм позволяет создавать функции, работающие с объектом любого типа в пределах одной иерархии.

1. Привести примеры иерархий с использованием абстрактных классов.

class Object

{

public:

Object(void);

~Object(void);

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

};

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. Привести примеры полиморфных функций.

void Vector::Add(Object\* p)

{

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

1. В каких случаях используется механизм позднего связывания?

Когда необходимо создать экземпляр определенного типа и вызвать его члены во время выполнения без кодирования факта его существования жестким образом на этапе компиляции.