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 №4»

Выполнил

Студент группы РИС-22-2б

Прядеин И.А.

Проверил доцент кафедры

ИТАС

Полякова О. А.

Пермь 2023

Постановка задачи

1. Определить пользовательский класс.
2. Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.
3. Определить в классе деструктор.
4. Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).
5. Перегрузить операцию присваивания.
6. Перегрузить операции ввода и вывода объектов с помощью потоков.
7. Определить производный класс.
8. Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.
9. Реализовать функции, получающие и возвращающие объект базового класса. Продемонстрировать принцип подстановки.

**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 операций сравнения дат.

**Исходный код программы:**

**Файл “Triad.h”:**

#include <iostream>

using namespace std;

class Triad {

public:

Triad();

virtual ~Triad();

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;

}

**Файл “Date.h”:**

#include "Triad.cpp"

class Date : public Triad {

public:

Date();

~Date();

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;

}

#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;

}

**Файл “Lab\_4.cpp”:**

#include <iostream>

#include "Date.cpp"

using namespace std;

int main() {

Triad a;

cin >> a;

cout << a;

Triad b(34, 23, 3);

cout << b;

a = b;

cout << a;

Date d1;

Date d2(22, 03, 2003);

Date d3;

cin >> d1;

cin >> d3;

cout << "\nDate 1: " << d1 << endl;

cout << "Date 2: " << d2 << endl;

cout << "Date 3: " << d3 << endl;

cout << "\nd1 == d2: " << (d1 == d2) << endl;

cout << "d1 != d3: " << (d1 != d3) << endl;

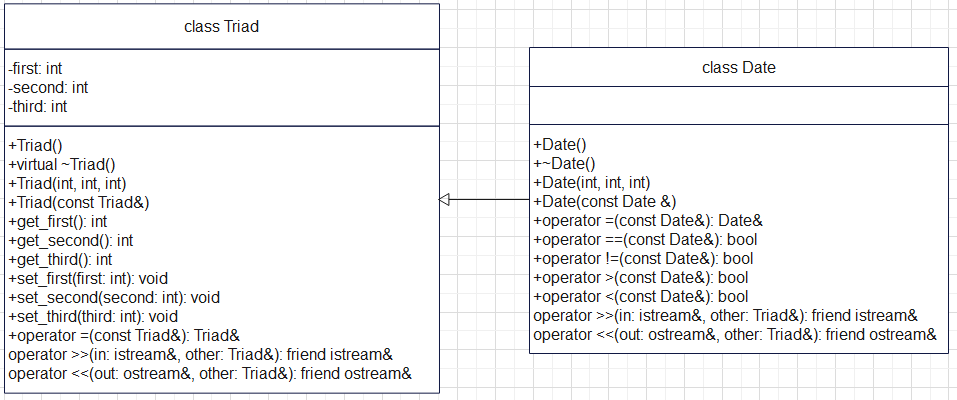
cout << "d2 > d3: " << (d2 > d3) << endl;

cout << "d3 < d1: " << (d3 < d1) << endl;

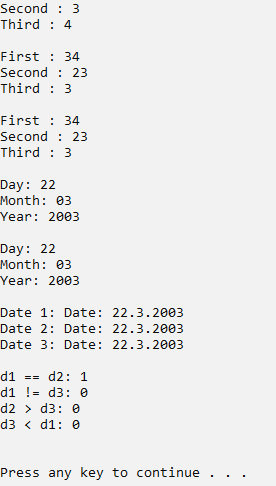
return 0;

}

**UML диаграмма:**



**Скриншот результата выполнения программы**



**Ответы на вопросы**

1. Для чего используется механизм наследования?

**Ответ:** Для создания новых классов, на основе существующих.

class My\_class {

public:

int a;

};

class sub\_class : My\_class {

public:

int b;

};

1. Каким образом наследуются компоненты класса, описанные спецификатором public?

**Ответ:** Компоненты класса, описанные со спецификатором public становятся доступными в базовом классе. Если тип наследования public, то становятся доступными для обращения из экземпляра класса.

class My\_class {

public:

int b;

};

class sub\_class : public My\_class {

private:

int c;

};

1. Каким образом наследуются компоненты класса, описанные со спецификатором private?

**Ответ:** Компоненты, описанные со спецификатором private не доступны в производном классе.

class My\_class {

private:

int a;

public:

int b;

};

class sub\_class : My\_class {

private:

int c;

};

1. Каким образом наследуются компоненты класса, описанные со спецификатором protected?

**Ответ:** Компоненты, описанные со спецификатором protected, при типе наследования public/protected остаются protected - доступны в производном классе и в классах-наследниках производного. При типе наследования private становятся private - доступны только внутри производного класса.

class My\_class {

protected:

int a;

int b;

};

class sub\_class : public My\_class { // Поля a и b остаются protected

private:

int c;

};

1. Каким образом описывается производный класс?

**Ответ:** class <имя>: <тип наследоавния> <имя базового класса>, ...

class sub\_class : public base\_class

1. Наследуются ли конструкторы?

**Ответ:** Конструкторы не наследуются, но могут вызываться для передачи параметров.

class My\_class {

public:

int a;

My\_class(int a) {

this->a = a;

};

};

class sub\_class : My\_class {

public:

sub\_class(int sa) : My\_class(sa) {

}

};

1. Наследуются ли деструкторы?

**Ответ:** Деструкторы не наследуются.

1. В каком порядке конструируются объекты производных классов?

**Ответ:** Сначала конструируются компоненты базового класса, потом базовый класс, далее компоненты производного класса, затем производный класс. Компоненты конструируются в том порядке, в котором они записаны.

1. В каком порядке уничтожаются объекты производных классов?

**Ответ:** Объекты производных классов уничтожаются в обратном порядке создания. Сначала производный класса, затем его компоненты, далее базовый класс и его компоненты.

1. Что представляют собой виртуальные функции и механизм позднего связывания?

**Ответ:** Виртуальные функции - функции, которые могут быть переопределены в производном. Используя виртуальную функцию, можно вызывать функции производного класса, используя указатель базового класса.

Виртуальные функции представляют механизм позднего или динамического связывания.

class Base {

public:

virtual void print () {

cout << “Base << endl;

}

};

class Sub : public Base {

pubic:

void print () {

cout << “Sub” << endl;

};

1. Могут ли быть виртуальными конструкторы? Деструкторы?

**Ответ:** Конструкторы не могут быть виртуальными, в отличие от деструкторов.

1. Наследуется ли спецификатор virtual?

**Ответ:** Спецификатор virtual наследуется.

1. Какое отношение устанавливает между классами открытое наследование?

**Ответ:** Открытое наследование является наследованием типа. Т.е. производный класс является частным случаем базового класса. Или производный класс может замещать методы базового класса.

1. Какое отношение устанавливает между классами закрытое наследование?

**Ответ:** При закрытом наследовании производный класс имеет доступ ко всему функционалу, за исключением private компонентов базового класса, но ни производные классы, ни экземпляры класса не имеют доступ к этому функционалу.

1. В чем заключается принцип подстановки?

**Ответ:** Если класс A является частным случаем класса B, то везде, где был использован класс B, можно использовать класс A. Принцип подстановки возможен, если не были переопределены методы базового класса.

1. Имеется иерархия классов:

class Student {

int age;

public:

string name;

...

};

class Employee : public Student {

protected:

string post;

...

};

class Teacher : public Employee {

protected:

int stage;

…

};

Teacher x;

Какие компонентные данные будет иметь объект x?

**Ответ:** Через объект x можно обратиться только к полю name. Но компонентные функции и друзья класса Teacher имеют доступ к protected полям string post и int stage.

1. Для классов Student, Employee и Teacher написать конструкторы без параметров.

**Ответ:**

class Student {

int age;

public:

string name;

Student() {

age = 0;

name = “”;

}

};

class Employee : public Student {

protected:

string post;

public:

Employee() : Student() {

post = “”;

}

};

class Teacher : public Employee {

protected:

int stage;

public:

Teacher() : Employee() {

stage = 0;

}

};

1. Для классов Student, Employee и Teacher написать конструкторы с параметрами.

**Ответ:**

class Student {

int age;

public:

string name;

Student(int a, string n) {

age = a;

name = n;

}

};

class Employee : public Student {

protected:

string post;

public:

Employee(int a, string n, string p) : Student(a, n) {

post = p;

}

};

class Teacher : public Employee {

protected:

int stage;

public:

Teacher(int a, string n, string p, int s) : Employee(a, n, p) {

stage = s;

}

};

1. Для классов Student, Employee и Teacher написать конструкторы копирования.

class Student {

int age;

public:

string name;

Student(const Student& other) {

age = other.age;

name = other.name;

}

};

class Employee : public Student {

protected:

string post;

public:

Employee(const Employee& other) : Student(other) {

post = other.post;

}

};

class Teacher : public Employee {

protected:

int stage;

public:

Teacher(const Teacher& other) : Employee(other) {

stage = other.stage;

}

};

1. Для классов Student, Employee и Teacher определить операцию присваивания.

**Ответ:**

class Student {

int age;

public:

string name;

void Set(int age, string name) {

cin >> age;

cin >> name;

};

};

class Employee : public Student {

protected:

string post;

public:

void Set(int age, string name, string post) : Set(age, name) {

cin >> post;

}

};

class Teacher : public Employee {

protected:

int stage;

public:

void Set(int age, string name, string post, int stage) : Set(age, name, post) {

cin >> stage;

}

};