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 " Простое наследование. Принцип подстановки."

Выполнил студент гр. РИС-24-3б

Жиряков Леонид Антонович

Проверил:

Доцент кафедры ИТАС   
Ольга Андреевна Полякова

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (оценка) (подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

г. Пермь, 2024

Вариант 8.

Постановка задачи:

Общая:

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.

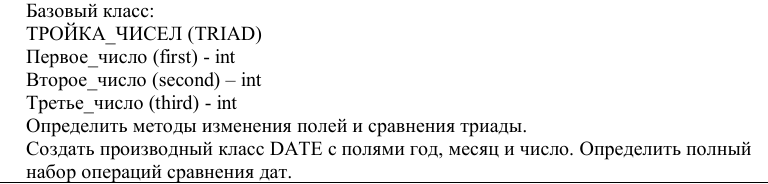
6. Перегрузить операции ввода и вывода объектов с помощью потоков.

7. Определить производный класс.

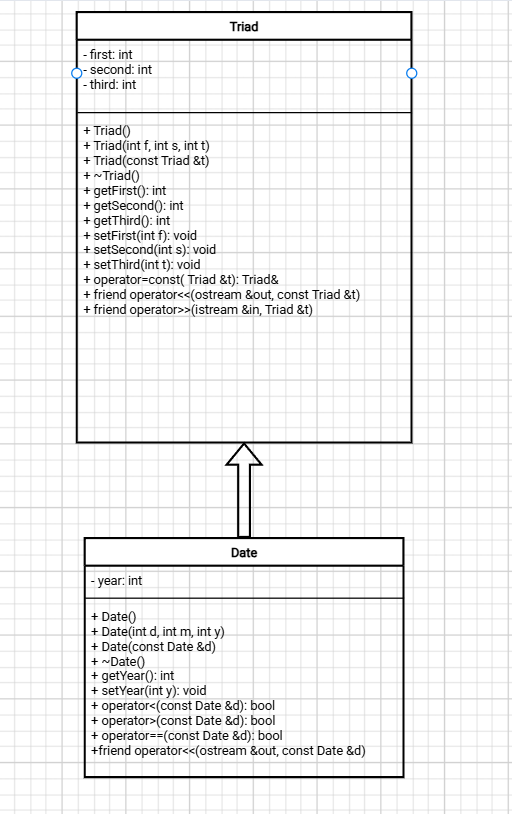
8. Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.

9. Реализовать функции, получающие и возвращающие объект базового класса. Продемонстрировать принцип подстановки.

Персональная:



UML – диаграмма:



Программная реализация

**Заголовочный файл**

#pragma once

#include <iostream>

using namespace std;

class Triad

{

private:

int first;

int second;

int third;

public:

// Конструкторы

Triad(); // без параметров

Triad(int f, int s, int t); // с параметрами

Triad(const Triad& t); // копирования

// Деструктор

~Triad();

// Селекторы

int getFirst() const;

int getSecond() const;

int getThird() const;

// Модификаторы

void setFirst(int f);

void setSecond(int s);

void setThird(int t);

// Операция присваивания

Triad& operator=(const Triad& t);

// Перегрузка операторов

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

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

};

#pragma once

#include "Triad.h"

class Date : public Triad

{

private:

int year;

public:

// Конструкторы

Date(); // без параметров

Date(int d, int m, int y); // с параметрами

Date(const Date& d); // копирования

// Деструктор

~Date();

// Селекторы

int getYear() const;

// Модификаторы

void setYear(int y);

// Операция сравнения

bool operator<(const Date& d) const;

bool operator>(const Date& d) const;

bool operator==(const Date& d) const;

// Перегрузка оператора вывода

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

};

**Реализация методов**

#include "Triad.h"

// Конструктор без параметров

Triad::Triad() : first(0), second(0), third(0) {}

// Конструктор с параметрами

Triad::Triad(int f, int s, int t) : first(f), second(s), third(t) {}

// Конструктор копирования

Triad::Triad(const Triad& t) : first(t.first), second(t.second), third(t.third) {}

// Деструктор

Triad::~Triad() {}

// Селекторы

int Triad::getFirst() const { return first; }

int Triad::getSecond() const { return second; }

int Triad::getThird() const { return third; }

// Модификаторы

void Triad::setFirst(int f) { first = f; }

void Triad::setSecond(int s) { second = s; }

void Triad::setThird(int t) { third = t; }

// Операция присваивания

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

{

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

first = t.first;

second = t.second;

third = t.third;

return \*this;

}

// Перегрузка оператора вывода

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

{

out << "(" << t.first << ", " << t.second << ", " << t.third << ")";

return out;

}

// Перегрузка оператора ввода

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

{

in >> t.first >> t.second >> t.third;

return in;

}

#include "Date.h"

// Конструктор без параметров

Date::Date() : Triad(0, 0, 0), year(0) {}

// Конструктор с параметрами

Date::Date(int d, int m, int y) : Triad(d, m, 0), year(y) {}

// Конструктор копирования

Date::Date(const Date& d) : Triad(d), year(d.year) {}

// Деструктор

Date::~Date() {}

// Селектор

int Date::getYear() const { return year; }

// Модификатор

void Date::setYear(int y) { year = y; }

// Операция сравнения

bool Date::operator<(const Date& d) const

{

if (getYear() != d.getYear())

return getYear() < d.getYear();

else if (getFirst() != d.getFirst())

return getFirst() < d.getFirst();

else

return getSecond() < d.getSecond();

}

bool Date::operator>(const Date& d) const

{

return d < \*this;

}

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

{

return getYear() == d.getYear() && getFirst() == d.getFirst() && getSecond() == d.getSecond();

}

// Перегрузка оператора вывода

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

{

out << d.getFirst() << "/" << d.getSecond() << "/" << d.getYear();

return out;

}  
**Функция main**

#include "Date.h"

int main()

{

setlocale(LC\_ALL, "Russian");

Triad triad1(1, 2, 3);

Triad triad2;

cout << "Триада 1: " << triad1 << endl;

cout << "Введите значения для Триады 2: ";

cin >> triad2;

cout << "Триада 2: " << triad2 << endl;

Date date1(28, 12, 2005);

Date date2(1, 9, 2001);

cout << "Дата 1: " << date1 << endl;

cout << "Дата 2: " << date2 << endl;

if (date1 > date2)

{

cout << "Дата 1 более поздняя, чем Дата 2." << endl;

}

else if (date1 < date2)

{

cout << "Дата 1 более ранняя, чем Дата 2." << endl;

}

else

{

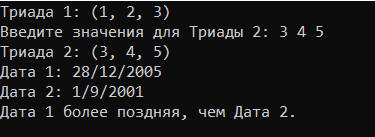
cout << "Дата 1 совпадает с Датой 2." << endl;

}

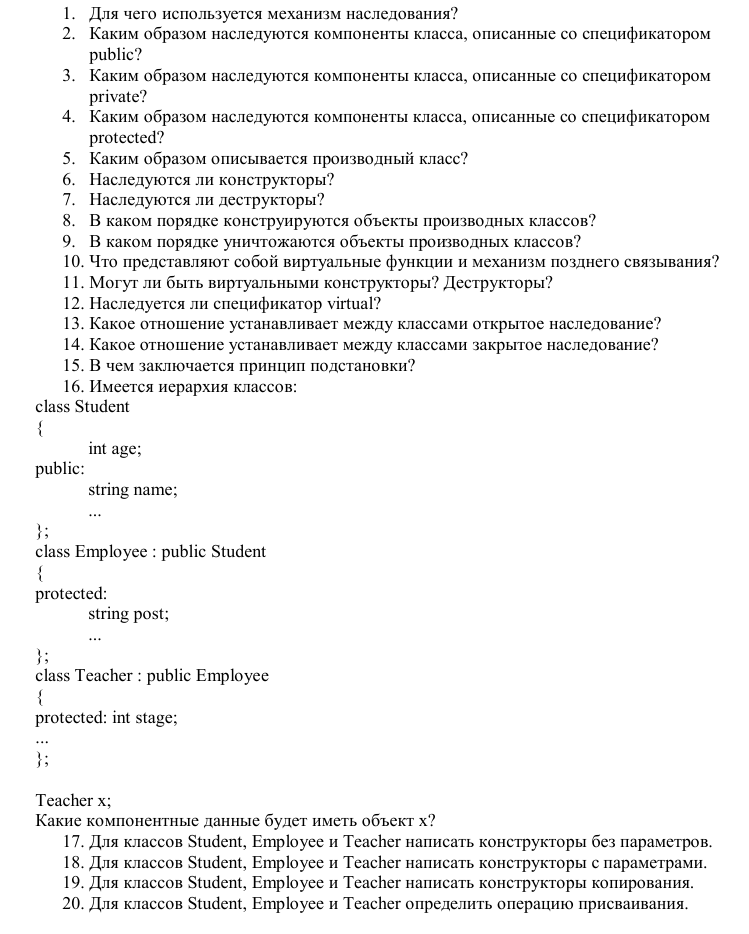
return 0;

}

Результат работы:



Контрольные вопросы:



Ответы:

1. **Для чего используется механизм наследования?**  
   Наследование используется для создания нового класса на основе уже существующего, позволяя расширять или изменять функциональность базового класса.
2. **Каким образом наследуются компоненты класса, описанные со спецификатором public?**  
   Компоненты с спецификатором public наследуются в производном классе с тем же уровнем доступа (public).
3. **Каким образом наследуются компоненты класса, описанные со спецификатором private?**  
   Компоненты с спецификатором private не доступны в производном классе и не наследуются.
4. **Каким образом наследуются компоненты класса, описанные со спецификатором protected?**  
   Компоненты с спецификатором protected наследуются в производном классе и могут использоваться в производных классах, но недоступны извне.
5. **Каким образом описывается производный класс?**  
   Производный класс описывается с помощью синтаксиса: class имя\_класса : список\_базовых\_классов { список\_компонентов\_класса };.
6. **Наследуются ли конструкторы?**  
   Конструкторы не наследуются в производном классе.
7. **Наследуются ли деструкторы?**  
   Деструкторы не наследуются, но они могут быть виртуальными.
8. **В каком порядке конструируются объекты производных классов?**  
   Объекты конструируются в порядке: сначала базовый класс, затем компоненты-объекты, а затем сам производный класс.
9. **В каком порядке уничтожаются объекты производных классов?**  
   Объекты уничтожаются в обратном порядке: сначала производный класс, затем его компоненты-объекты, а потом базовый объект.
10. **Что представляют собой виртуальные функции и механизм позднего связывания?**  
    Виртуальные функции — это функции, которые могут иметь разные реализации в производных классах. Механизм позднего связывания позволяет вызывать правильную функцию в зависимости от типа объекта, на который указывает указатель, а не от типа самого указателя.
11. **Могут ли быть виртуальными конструкторы? Деструкторы?**  
    Конструкторы не могут быть виртуальными, но деструкторы могут быть виртуальными.
12. **Наследуется ли спецификатор virtual?**  
    Да, спецификатор virtual наследуется, и повторное определение функции в производном классе создает новую виртуальную функцию.
13. **Какое отношение устанавливает между классами открытое наследование?**  
    Открытое наследование устанавливает отношение «является»: класс-наследник является частью класса-родителя.
14. **Какое отношение устанавливает между классами закрытое наследование?**  
    Закрытое наследование не соблюдает принцип подстановки, и методы базового класса не доступны в производном классе.
15. **В чем заключается принцип подстановки?**  
    Принцип подстановки гласит, что объект производного класса может быть использован вместо объекта базового класса везде, где это возможно.
16. **Какие компонентные данные будет иметь объект х?**  
    Объект x будет иметь:
    * age (int, от базового класса Student),
    * name (string, от базового класса Student),
    * post (string, от производного класса Employee),
    * stage (int, от производного класса Teacher).

### 17.Конструкторы без параметров:

class Student

{

public:

Student()

{

// Инициализация по умолчанию

}

};

class Employee

{

public:

Employee()

{

// Инициализация по умолчанию

}

};

class Teacher

{

public:

Teacher()

{

// Инициализация по умолчанию

}

};

### 18.Конструкторы с параметрами:

class Student

{

private:

std::string name;

int age;

public:

Student(std::string n, int a) : name(n), age(a) {}

};

class Employee

{

private:

std::string name;

int age;

double salary;

public:

Employee(std::string n, int a, double s) : name(n), age(a), salary(s) {}

};

class Teacher

{

private:

std::string name;

int age;

std::string subject;

public:

Teacher(std::string n, int a, std::string sub) : name(n), age(a), subject(sub) {}

};

### 19. Конструкторы копирования:

class Student

{

private:

std::string name;

int age;

public:

Student(std::string n, int a) : name(n), age(a) {}

Student(const Student& s) : name(s.name), age(s.age) {}

};

class Employee

{

private:

std::string name;

int age;

double salary;

public:

Employee(std::string n, int a, double s) : name(n), age(a), salary(s) {}

Employee(const Employee& e) : name(e.name), age(e.age), salary(e.salary) {}

};

class Teacher

{

private:

std::string name;

int age;

std::string subject;

public:

Teacher(std::string n, int a, std::string sub) : name(n), age(a), subject(sub) {}

Teacher(const Teacher& t) : name(t.name), age(t.age), subject(t.subject) {}

};

### 20. Операция присваивания:

class Student

{

private:

std::string name;

int age;

public:

Student(std::string n, int a) : name(n), age(a) {}

Student& operator=(const Student& s)

{

if (this != &s)

{

name = s.name;

age = s.age;

}

return \*this;

}

};

class Employee

{

private:

std::string name;

int age;

double salary;

public:

Employee(std::string n, int a, double s) : name(n), age(a), salary(s) {}

Employee& operator=(const Employee& e)

{

if (this != &e)

{

name = e.name;

age = e.age;

salary = e.salary;

}

return \*this;

}

};

class Teacher

{

private:

std::string name;

int age;

std::string subject;

public:

Teacher(std::string n, int a, std::string sub) : name(n), age(a), subject(sub) {}

Teacher& operator=(const Teacher& t)

{

if (this != &t)

{

name = t.name;

age = t.age;

subject = t.subject;

}

return \*this;

}

};

Ссылка:

**https://github.com/LeonidZhir/-**