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 №4.**

**«Простое наследование. Принцип подстановки.»**

Выполнил студент гр. РИС-24-1б

Чижов Денис Николаевич

Проверил:

Доц. Каф. ИТАС 

Ольга Андреевна Полякова

(оценка) (подпись)

(дата)

г. Пермь, 2025

**Задача:**

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.

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.

**Анализ:**

**1. Базовый класс Triad**

* **Поля:**
  + first, second, third (целые числа).
* **Конструкторы:**
  + Без параметров (инициализация нулями).
  + С параметрами (установка значений).
  + Копирования (создание копии объекта).
* **Деструктор:**
  + Пустой, так как нет динамической памяти.
* **Селекторы и модификаторы:**
  + get\_first(), set\_second() и т. д. – для доступа к полям.
* **Перегрузка операторов:**
  + =, <, >, == (сравнение по **сумме чисел**).
  + >> и << (ввод/вывод в формате число1, число2, число3).

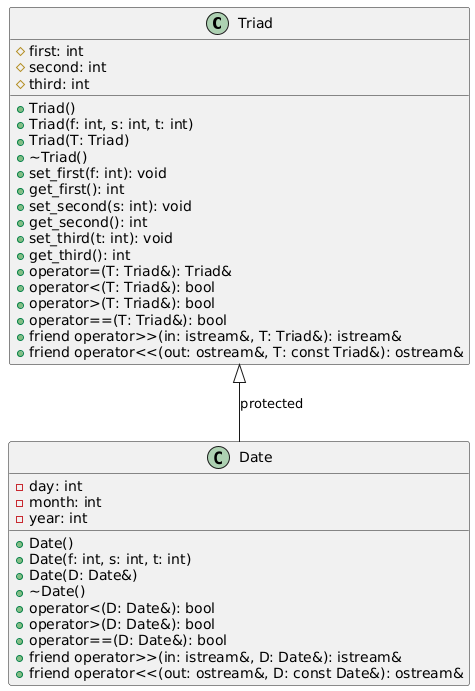
### ****2. Производный класс**** Date

* **Наследование:** protected Triad (ограниченный доступ к полям Triad).
* **Поля:**
  + day, month, year (используют first, second, third из Triad).
* **Конструкторы:**
  + Аналогичны Triad, но работают с датами.
* **Перегрузка операторов:**
  + <, >, == – **корректное сравнение дат** (сначала год, потом месяц, потом день).
  + >> и << – ввод/вывод в формате день.месяц.год.

### ****3. Демонстрация в**** main()

* **Тестирование Triad:**
  + Создание объектов a, b, c.
  + Ввод, сравнение, копирование, вывод.
* **Тестирование Date:**
  + Аналогично, но с датами

**UML диаграмма:**

****

**Код:**

Triad.h:

#pragma once

#include <iostream>

using namespace std;

class Triad {

protected:

int first;

int second;

int third;

public:

//Конструкторы

Triad() { first = 0; second = 0; third = 0; }; //Конструктор без параметров

Triad(int f, int s, int t) { first = f; second = s; third = t; };//Конструктор с параметрами

Triad(const Triad& T) { first = T.first; second = T.second; third = T.third; }; //Конструктор копирования

//Деструктор

~Triad() {};

//Селекторы и модификаторы

void set\_first(int f) { first = f; }; //модификатор first

long get\_first() { return first; }; //селектор first

void set\_second(int s) { second = s; }; //модификатор second

int get\_second() { return second; }; //селектор second

void set\_third(int t) { third = t; }; //модификатор second

int get\_third() { return third; }; //селектор second

//Перегрузки операторов

Triad& operator=(const Triad&);

bool operator<(const Triad&);

bool operator>(const Triad&);

bool operator==(const Triad&);

//Глобальные функции ввода-вывода

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

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

};

Triad.cpp:

#include "Triad.h"

#include <iostream>

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

{

//проверка на самоприсваивание

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

first = T.first;

second = T.second;

third = T.third;

return \*this;

}

bool Triad::operator<(const Triad& T)

{

int f = first + second + third;

int s = T.first + T.second + T.third;

if (f < s) { return true; }

else { return false; }

}

bool Triad::operator>(const Triad& T)

{

int f = first + second + third;

int s = T.first + T.second + T.third;

if (f > s) { return true; }

else { return false; }

}

bool Triad::operator==(const Triad& T)

{

int f = first + second + third;

int s = T.first + T.second + T.third;

if (f == s) { return true; }

else { return false; }

}

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

in >> T.first;

in >> T.second;

in >> T.third;

cout << endl;

return in;

}

ostream& operator<<(ostream& out, const Triad& T) {

return (out << T.first << " , " << T.second << " , " << T.third << endl);

}

Date.h:

#pragma once

#include "Triad.h"

class Date : protected Triad{

public:

Date() { day = first; month = second; year = third; }; //Конструктор без параметров

Date(int f, int s, int t) { day = f; month = s; year = t; };

Date(const Date& D) { first = D.first; second = D.second; third = D.third; day = D.day; month = D.month; year = D.year;};

~Date() {}; //Деструктор

bool operator<(const Date&);

bool operator>(const Date&);

bool operator==(const Date&);

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

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

private:

int day;

int month;

int year;

};

Date.cpp:

#include "Date.h"

#include <iostream>

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

cout << "Введите день:";

in >> D.first;

D.day= D.first;

cout << "Введите месяц:";

in >> D.second;

D.month = D.second;

cout << "Введите год:";

in >> D.third;

D.year = D.third;

cout << endl;

return in;

}

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

return (out << D.day << '.' << D.month << '.' << D.year << endl);

}

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

if (year < D.year) { return true; }

else {

if (month < D.month) { return true; }

else {

if (day < D.day) { return true; }

else { return false; }

}

}

}

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

if (year > D.year) { return true; }

else {

if (month > D.month) { return true; }

else {

if (day > D.day) { return true; }

else { return false; }

}

}

}

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

if (year == D.year) { return true; }

else {

if (month == D.month) { return true; }

else {

if (day == D.day) { return true; }

else { return false; }

}

}

}

Main.cpp:

#include "Triad.h"

#include "Date.h"

#include "iostream"

#include "clocale"

using namespace std;

int main() {

setlocale(LC\_ALL, "Ru");

//проверка Triad

Triad a;

Triad b;

Triad c;

cin >> a;

cin >> b;

if (a > b) {

cout << "A больше B" << endl;

}

else { cout << "A не больше B" << endl; }

if (a < b) {

cout << "A меньше B" << endl;

}

else { cout << "A не меньше B" << endl; }

if (a == b) {

cout << "A равно B" << endl;

}

else { cout << "A не равно B" << endl; }

c = a;

cout << a;

cout << b;

cout << c;

//проверка Date

Date D, E, F;

cin >> D;

cin >> E;

F = D;

if (D > E) {

cout << "D больше E" << endl;

}

else { cout << "D не больше E" << endl; }

if (D < E) {

cout << "D меньше E" << endl;

}

else { cout << "D не меньше E" << endl; }

if (D == E) {

cout << "D равно E" << endl;

}

else { cout << "D не равно E" << endl; }

cout << D;

cout << E;

cout << F;

return 0;

}

**Вывод:**

Программа работает. Задача решена.

**Вопросы:**

**1. Для чего используется механизм наследования?**

Наследование позволяет создавать новые классы (производные) на основе существующих (базовых), что обеспечивает:

* **Повторное использование кода** (наследование членов базового класса).
* **Расширение функциональности** (добавление новых методов и полей).
* **Полиморфизм** (возможность переопределения методов для разных классов в иерархии).

**2. Каким образом наследуются компоненты класса, описанные со спецификатором public?**

* Если наследование **публичное (public)**, то:
  + public → public (в производном классе).
  + protected → protected.
  + private → недоступно.
* Если наследование **защищенное (protected)**, то:
  + public и protected → protected.
  + private → недоступно.
* Если наследование **приватное (private)**, то:
  + Все (public, protected) → private.
  + private → недоступно.

**3. Наследование private-компонентов**

* private-члены **никогда не наследуются** напрямую.
* Доступ к ним возможен только через public/protected методы базового класса.

**4. Наследование protected-компонентов**

* В публичном наследовании protected → protected.
* В защищенном/приватном наследовании protected → protected/private.

**5. Как описывается производный класс?**

class ПроизводныйКласс : [тип\_наследования] БазовыйКласс {

// поля и методы

};

Пример:

class Employee : public Student { ... };

**6. Наследуются ли конструкторы?**

* **Нет**, конструкторы не наследуются автоматически.
* Их нужно вызывать явно в списке инициализации производного класса.

**7. Наследуются ли деструкторы?**

* Деструкторы **не наследуются**, но вызываются автоматически при уничтожении объекта (в обратном порядке конструкторам).

**8. Порядок конструирования объектов**

1. Конструктор **базового класса**.
2. Конструкторы **полей производного класса** (в порядке объявления).
3. Конструктор **производного класса**.

**9. Порядок уничтожения объектов**

1. Деструктор **производного класса**.
2. Деструкторы **полей производного класса** (в обратном порядке объявления).
3. Деструктор **базового класса**.

**10. Виртуальные функции и позднее связывание**

* **Виртуальная функция** (virtual) может быть переопределена в производном классе.
* **Позднее связывание** — решение о вызове функции принимается во время выполнения (через таблицу виртуальных функций).

Пример:

class Base {

public:

virtual void show() { cout << "Base"; }

};

class Derived : public Base {

public:

void show() override { cout << "Derived"; }

};

**11. Могут ли быть виртуальными конструкторы? Деструкторы?**

* **Конструкторы**: **Нет** (объект еще не создан).
* **Деструкторы**: **Да**, и они **должны** быть виртуальными, если возможен полиморфизм.

**12. Наследуется ли спецификатор virtual?**

* **Да**, если функция объявлена virtual в базовом классе, то она остается виртуальной во всей иерархии.

**13. Открытое наследование (public)**

* Устанавливает отношение **«является»** (например, Employee является Student).
* Сохраняет уровень доступа базового класса.

**14. Закрытое наследование (private)**

* Устанавливает отношение **«реализован посредством»**.
* Все члены базового класса становятся private в производном.

**15. Принцип подстановки (Liskov Substitution Principle, LSP)**

* Объекты производного класса должны **корректно работать** везде, где используется базовый класс.

**16. Компонентные данные объекта Teacher x**

class Student { public: int age; string name; };

class Employee : public Student { protected: string post; };

class Teacher : public Employee { protected: int stage; };

Объект x содержит:

* age (public, унаследовано от Student)
* name (public, унаследовано от Student)
* post (protected, унаследовано от Employee)
* stage (protected, собственное поле Teacher)

**17. Конструкторы без параметров**

class Student {

public:

Student() : age(0), name("") {}

};

class Employee : public Student {

public:

Employee() : post("") {}

};

class Teacher : public Employee {

public:

Teacher() : stage(0) {}

};

**18. Конструкторы с параметрами**

class Student {

public:

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

};

class Employee : public Student {

public:

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

};

class Teacher : public Employee {

public:

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

};

**19. Конструкторы копирования**

class Student {

public:

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

};

class Employee : public Student {

public:

Employee(const Employee &other) : Student(other), post(other.post) {}

};

class Teacher : public Employee {

public:

Teacher(const Teacher &other) : Employee(other), stage(other.stage) {}

};

**20. Перегрузка операции присваивания**

class Student {

public:

Student& operator=(const Student &other) {

if (this != &other) {

age = other.age;

name = other.name;

}

return \*this;

}

};

class Employee : public Student {

public:

Employee& operator=(const Employee &other) {

if (this != &other) {

Student::operator=(other);

post = other.post;

}

return \*this;

}

};

class Teacher : public Employee {

public:

Teacher& operator=(const Teacher &other) {

if (this != &other) {

Employee::operator=(other);

stage = other.stage;

}

return \*this;

}

};