

Федеральное агентство связи

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М. А. БОНЧ-БРУЕВИЧА» (СПбГУТ)**

Факультет информационных технологий и программной инженерии Кафедра: Программная инженерия. Разработка программного обеспечения и приложений искусственного интеллекта в киберфизических системах

ЛАБОРАТОРНАЯ РАБОТА №2-4

по дисциплине «Объектно-ориентированное программирование»

Тема: отношение включения

Выполнил: студент 2-го курса группы ИКПИ-42 Терещенко Максим Андреевич

Преподаватель: Петрова Ольга Борисовна

Санкт-Петербург 2025

Постановка задачи

Реализовать иерархию классов на языке C++ с использованием принципов объектно-ориентированного программирования: инкапсуляции, наследования и полиморфизма. Создать базовый класс `COne`, производные классы `CTwo`, `CThree`, `CFour`. Обеспечить корректное копирование, присваивание и освобождение динамической памяти. Продемонстрировать работу виртуальных функций при вызове методов через указатели на базовый класс.

Анализ задачи

Алгоритмическая часть

1. Базовый класс `COne`:

- содержит два поля: `int n` и `std::string s`;
- реализует конструкторы, деструктор, оператор присваивания;
- предоставляет методы доступа `get/set` и метод `print()`.

2. Класс `CTwo`:

- хранит указатель `COne* p` и число `double d`;
- владеет объектом `COne` через динамическую память;
- реализует глубокое копирование (`copy constructor` и `operator=`);
- определяет виртуальный метод `print()`.

3. Класс `CThree`:

- наследуется от `CTwo`;
- добавляет строковое поле `extra`;
- переопределяет метод `print()`.

4. Класс `CFour`:

- наследуется от `CThree`;
- добавляет специфичные поля;
- также переопределяет метод `print()`.

Формулы

- Для демонстрации полиморфизма используется механизм виртуальных функций:

```
CTwo* arr[i] → print();
```

вызовет `print()` именно того класса, на который реально указывает `arr[i]` .

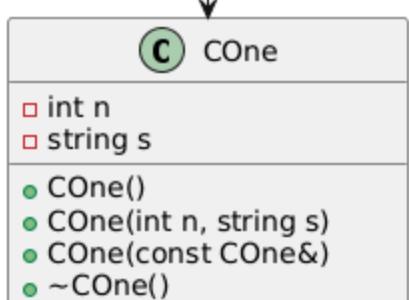
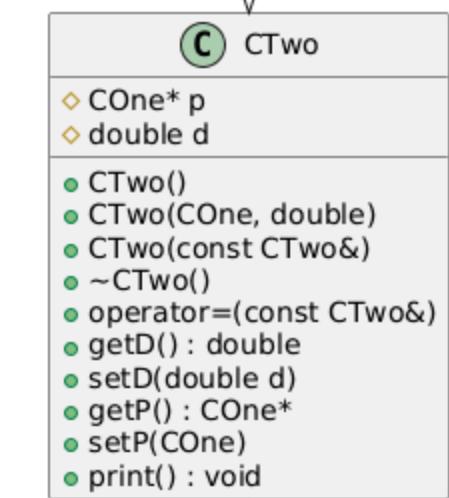
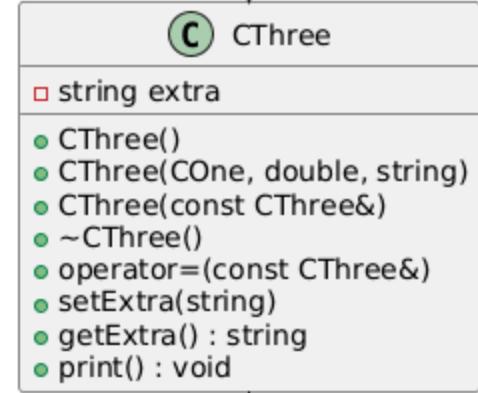
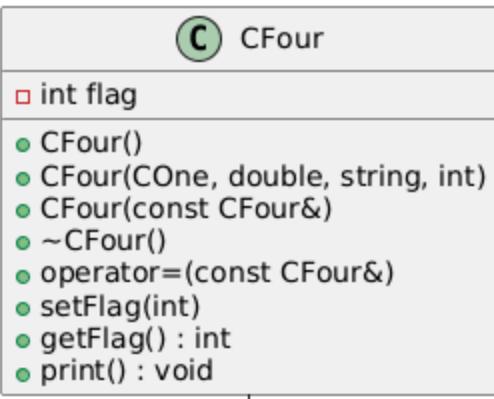
Таблица идентификаторов

Идентификатор	Описание
<code>n</code>	Приватное поле класса <code>COne</code> , хранящее целое число.
<code>s</code>	Приватное поле класса <code>COne</code> , хранящее строку типа <code>std::string</code> .
<code>COne()</code>	Конструктор по умолчанию класса <code>COne</code> , инициализирует <code>n=0</code> , <code>s=""</code> .
<code>COne(int n_, const std::string& s_)</code>	Конструктор, задающий значения полей <code>n</code> и <code>s</code> .
<code>COne(const COne& other)</code>	Конструктор копирования класса <code>COne</code> .
<code>~COne()</code>	Деструктор класса <code>COne</code> .
<code>operator=(const COne& other)</code>	Перегруженный оператор присваивания для класса <code>COne</code> .
<code>getN() const</code>	Метод, возвращающий значение поля <code>n</code> .
<code>setN(int n_)</code>	Метод, устанавливающий значение поля <code>n</code> .
<code>getS() const</code>	Метод, возвращающий значение поля <code>s</code> .
<code>setS(const std::string& s_)</code>	Метод, устанавливающий значение поля <code>s</code> .
<code>print() const</code>	Метод для вывода полей класса <code>COne</code> в формате <code>n=<значение>, s=<значение></code> .
<code>p</code>	Защищённое поле класса <code>CTwo</code> , указатель на объект класса <code>COne</code> .
<code>d</code>	Защищённое поле класса <code>CTwo</code> , хранящее число типа <code>double</code> .
<code>CTwo()</code>	Конструктор по умолчанию класса <code>CTwo</code> , инициализирует <code>p=nullptr</code> , <code>d=0.0</code> .

Идентификатор	Описание
CTwo(const COne& obj, double d_)	Конструктор, создающий объект CTwo с копией объекта COne и значением d .
CTwo(const CTwo& other)	Конструктор копирования класса CTwo .
~CTwo()	Виртуальный деструктор класса CTwo , освобождающий память под р .
operator=(const CTwo& other)	Перегруженный оператор присваивания для класса CTwo .
getD() const	Метод, возвращающий значение поля d .
setD(double d_)	Метод, устанавливающий значение поля d .
getP() const	Метод, возвращающий указатель р на объект COne .
setP(const COne& obj)	Метод, устанавливающий значение указателя р с копированием объекта COne .
print() const	Виртуальный метод для вывода полей класса CTwo и объекта COne , на который указывает р .
extra	Приватное поле класса CThree , хранящее строку типа std::string .
CThree()	Конструктор по умолчанию класса CThree , инициализирует extra="" .
CThree(const COne& obj, double d_, const std::string& e)	Конструктор, создающий объект CThree с использованием базового класса CTwo и поля extra .
CThree(const CThree& other)	Конструктор копирования класса CThree .
~CThree()	Деструктор класса CThree .
operator=(const CThree& other)	Перегруженный оператор присваивания для класса CThree .
setExtra(const std::string& e)	Метод, устанавливающий значение поля extra .
getExtra() const	Метод, возвращающий значение поля extra .

Идентификатор	Описание
print() const	Переопределённый метод для вывода полей класса CThree и базового класса CTwo .
flag	Приватное поле класса CFour , хранящее целое число.
CFour()	Конструктор по умолчанию класса CFour , инициализирует flag=0 .
CFour(const COne& obj, double d_, const std::string& e, int f)	Конструктор, создающий объект CFour с использованием базового класса CThree и поля flag .
CFour(const CFour& other)	Конструктор копирования класса CFour .
~CFour()	Деструктор класса CFour .
operator=(const CFour& other)	Перегруженный оператор присваивания для класса CFour .
setFlag(int f)	Метод, устанавливающий значение поля flag .
getFlag() const	Метод, возвращающий значение поля flag .
print() const	Переопределённый метод для вывода полей класса CFour и базового класса CThree .
printAll(CTwo* arr[], int n)	Функция, выводящая содержимое массива указателей на объекты CTtwo и его производные.

UML-диаграмма классов



КОМПОЗИЦИЯ

- operator=(const COne&)
- getN() : int
- setN(int n)
- getS() : string
- setS(string s)
- print() : void

Копии экранов выполнения

```
PS C:\Educational activities\2_course\OOP\lab2> ./lab4
CTwo: d=3.14
COne: n=42, s=hello
-----
CThree: extra=extra data
CTwo: d=2.71
COne: n=42, s=hello
-----
CFour: flag=7
CThree: extra=deep extra
CTwo: d=1.61
COne: n=42, s=hello
-----
```

Тесты для лабораторной работы

№	Входные данные	Действие	Ожидаемый результат
1	COne one(42, "hello");	Вызов метода print() для объекта one	Вывод: COne: n=42, s=hello
2	CThree obj3(one, 2.71, "extra data");	Вызов метода print() для объекта CThree	Вывод: CThree: extra=extra data CTwo: d=2.71 COne: n=42, s=hello
3	CThree obj3(one, 2.71, "extra data");	Вызов конструктора	Вывод для нового объекта должен быть

№	Входные данные	Действие	Ожидаемый результат
		копирования для объекта CThree И вывод на экран	идентичен: CThree: extra=extra data CTwo: d=2.71 COne: n=42, s=hello
4	Массив указателей на объекты: arr[0] = &obj2 , arr[1] = &obj3 , arr[2] = &obj4	Вызов метода printAll() для всех объектов в массиве	Вывод для каждого объекта: CTwo: d=3.14 ----- CThree: extra=extra data CTwo: d=2.71 ----- CFour: flag=7 CThree: extra=deep extra CTwo: d=1.61

1. Тест на корректность вывода для объекта COne

Изменения в программе:

```
COne one(42, "hello");
one.print();
```

Ожидаемый результат:

```
COne: n=42, s=hello
```

Скриншот:

```
(base) PS D:\Educational activities\Proga\university_labs\2_course\OOP\lab2> ./lab4
COne: n=42, s=hello
```

2. Тест на полиморфизм для объекта CThree

Изменения в программе:

```
COne one(42, "hello");
CThree obj3(one, 2.71, "extra data");
obj3.print();
```

Ожидаемый результат:

```
CThree: extra=extra data
CTwo: d=2.71
COne: n=42, s=hello
```

Скриншот:

```
(base) PS D:\Educational activities\Proga\university_labs\2_course\OOP\lab2> ./lab4
COne: n=42, s=hello
CThree: extra=extra data
CTwo: d=2.71
COne: n=42, s=hello
```

3. Тест на конструктор копирования для объекта CThree

Изменения в программе:

```
COne one(42, "hello");
CThree obj3(one, 2.71, "extra data");
CThree obj5 = obj3;
obj5.print();
```

Ожидаемый результат:

```
CThree: extra=extra data
CTwo: d=2.71
COne: n=42, s=hello
```

Скриншот:

```
(base) PS D:\Educational activities\Proga\university_labs\2_course\OOP\lab2> ./lab4
CThree: extra=extra data
CTwo: d=2.71
COne: n=42, s=hello
```

4. Тест на полиморфизм и вызов метода printAll() для всех объектов

Изменения в программе:

```
COne one(42, "hello");

CTwo obj2(one, 3.14);
CThree obj3(one, 2.71, "extra data");
CFour obj4(one, 1.61, "deep extra", 7);

CTwo* arr[3];
arr[0] = &obj2;
arr[1] = &obj3;
arr[2] = &obj4;

printAll(arr, 3);
```

Ожидаемый результат:

```
CTwo: d=3.14
-----
CThree: extra=extra data
CTwo: d=2.71
-----
CFour: flag=7
CThree: extra=deep extra
CTwo: d=1.61
-----
```

Скриншот:

```
(base) PS D:\Educational activities\Proga\university_labs\2_course\OOP\lab2> ./lab4
CTwo: d=3.14
COne: n=42, s=hello
-----
CThree: extra=extra data
CTwo: d=2.71
COne: n=42, s=hello
-----
CFour: flag=7
CThree: extra=deep extra
CTwo: d=1.61
COne: n=42, s=hello
-----
```

Выводы

В данной работе:

- Реализована иерархия классов с инкапсуляцией, наследованием и полиморфизмом.
- Продемонстрирована работа виртуальных функций при использовании указателей на базовый класс.
- Построена UML-диаграмма для визуализации структуры классов.

Приложение: код программы

Файл COne.h

```
#ifndef CONE_H
#define CONE_H

#include <iostream>
#include <string>

class COne {
private:
    int n;
    std::string s;
public:
    COne();
    COne(int n_, const std::string& s_);
    COne(const COne& other);
    ~COne();

    COne& operator=(const COne& other);

    int getN() const;
    void setN(int n_);

    std::string getS() const;
    void setS(const std::string& s_);

    void print() const;
};

#endif
```

Файл COne.cpp

```
#include "COne.h"

COne::COne() : n(0), s("") {}

COne::COne(int n_, const std::string& s_) : n(n_), s(s_) {}

COne::COne(const COne& other) : n(other.n), s(other.s) {}

COne::~COne() {}

COne& COne::operator=(const COne& other) {
    if (this != &other) {
        n = other.n;
        s = other.s;
    }
    return *this;
}

int COne::getN() const { return n; }

void COne::setN(int n_) { n = n_; }

std::string COne::getS() const { return s; }

void COne::setS(const std::string& s_) { s = s_; }

void COne::print() const {
    std::cout << "COne: n=" << n << ", s=" << s << std::endl;
}
```

Файл CTwo.h

```
#ifndef CTWO_H
#define CTWO_H

#include "COne.h"

class CTwo {
protected:
    COne* p;
    double d;
public:
    CTwo();
    CTwo(const COne& obj, double d_);
    CTwo(const CTwo& other);
    virtual ~CTwo();

    CTwo& operator=(const CTwo& other);

    double getD() const;
    void setD(double d_);

    COne* getP() const;
    void setP(const COne& obj);

    virtual void print() const;
};

#endif
```

Файл CTwo.cpp

```
#include "CTwo.h"

CTwo::CTwo() : p(nullptr), d(0.0) {}

CTwo::CTwo(const COne& obj, double d_) : d(d_) {
    p = new COne(obj);
}

CTwo::CTwo(const CTwo& other) : d(other.d) {
    if (other.p)
        p = new COne(*other.p);
    else
        p = nullptr;
}

CTwo::~CTwo() {
    delete p;
}

CTwo& CTwo::operator=(const CTwo& other) {
    if (this != &other) {
        d = other.d;
        delete p;
        p = (other.p) ? new COne(*other.p) : nullptr;
    }
    return *this;
}

double CTwo::getD() const { return d; }

void CTwo::setD(double d_) { d = d_; }

COne* CTwo::getP() const { return p; }

void CTwo::setP(const COne& obj) {
    delete p;
    p = new COne(obj);
}

void CTwo::print() const {
    std::cout << "CTwo: d=" << d << std::endl;
    if (p) p->print();
```

```
}
```

Файл CThree.h

```
#ifndef CTHREE_H
#define CTHREE_H

#include "CTwo.h"

class CThree : public CTwo {
private:
    std::string extra;
public:
    CThree();
    CThree(const COne& obj, double d_, const std::string& e);
    CThree(const CThree& other);
    ~CThree();

    CThree& operator=(const CThree& other);

    void setExtra(const std::string& e);
    std::string getExtra() const;

    void print() const override;
};

#endif
```

Файл CThree.cpp

```
#include "CThree.h"

CThree::CThree() : CTwo(), extra("") {}

CThree::CThree(const COne& obj, double d_, const std::string& e)
    : CTwo(obj, d_), extra(e) {}

CThree::CThree(const CThree& other) : CTwo(other), extra(other.extra) {}

CThree::~CThree() {}

CThree& CThree::operator=(const CThree& other) {
    if (this != &other) {
        CTwo::operator=(other);
        extra = other.extra;
    }
    return *this;
}

void CThree::setExtra(const std::string& e) { extra = e; }
std::string CThree::getExtra() const { return extra; }

void CThree::print() const {
    std::cout << "CThree: extra=" << extra << std::endl;
    CTwo::print();
}
```

Файл CFour.h

```
#ifndef CFOUR_H
#define CFOUR_H

#include "CThree.h"

class CFour : public CThree {
private:
    int flag;
public:
    CFour();
    CFour(const COne& obj, double d_, const std::string& e, int f);
    CFour(const CFour& other);
    ~CFour();

    CFour& operator=(const CFour& other);

    void setFlag(int f);
    int getFlag() const;

    void print() const override;
};

#endif
```

Файл CFour.cpp

```
#include "CFour.h"

CFour::CFour() : CThree(), flag(0) {}

CFour::CFour(const COne& obj, double d_, const std::string& e, int f)
: CThree(obj, d_, e), flag(f) {}

CFour::CFour(const CFour& other) : CThree(other), flag(other.flag) {}

CFour::~CFour() {}

CFour& CFour::operator=(const CFour& other) {
    if (this != &other) {
        CThree::operator=(other);
        flag = other.flag;
    }
    return *this;
}

void CFour::setFlag(int f) { flag = f; }
int CFour::getFlag() const { return flag; }

void CFour::print() const {
    std::cout << "CFour: flag=" << flag << std::endl;
    CThree::print();
}
```

Файл main.cpp

```
#include "CFour.h"

void printAll(CTwo* arr[], int n) {
    for (int i = 0; i < n; ++i) {
        arr[i]->print();
        std::cout << "-----" << std::endl;
    }
}

int main() {
    COne one(42, "hello");

    CTwo obj2(one, 3.14);
    CThree obj3(one, 2.71, "extra data");
    CFour obj4(one, 1.61, "deep extra", 7);

    CTwo* arr[3];
    arr[0] = &obj2;
    arr[1] = &obj3;
    arr[2] = &obj4;

    printAll(arr, 3);

    return 0;
}
```