ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»

(СПбГУТ)

Факультет инфокоммуникационных Сетей и систем (иксс)

кафедра программной инженерии и вычислительной техники (пи и вт)

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №3.

Тема: «ОТНОШЕНИЕ НАСЛЕДОВАНИЯ»

Вариант №14

Выполнил:

Студент группы ИКПИ-05

Принял:

Доцент кафедры ПИиВТ

Молошников Ф.А.

Подпись\_\_\_\_\_\_\_\_\_\_\_\_

Коробов С.А.

Подпись\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_»\_\_\_\_\_\_\_\_2021

СОДЕРЖАНИЕ

[1. Постановка Задачи 3](#_Toc82350657)

[1.1. Результаты работы программы: 3](#_Toc82350658)

[1.2. Программа на языке c++ 3](#_Toc82350659)

[1.2.1. main.cpp 3](#_Toc82350660)

[1.2.2. COneCTwo.h 4](#_Toc82350661)

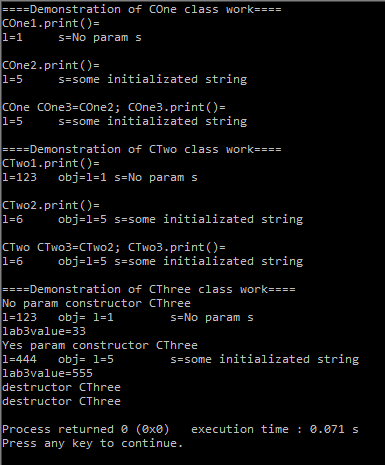
[1.2.3. COneCTwo.cpp 5](#_Toc82350662)

[2. Выводы 7](#_Toc82350663)

1. Постановка Задачи

Дополнить систему, состоящую из двух классов COne и CTwo, которые были разработаны в лабораторной работе 2, новым классом CThree. Новый класс должен быть связан public наследованием с классом CTwo. Класс CThree должен имеет одно поля, которое выбирается студентом самостоятельно. Для разрабатываемого класса написать конструкторы умолчания, с параметрами и конструктор копирования, деструктор, методы доступа и метод print(). Написать тестовую программу для проверки работоспособности разработанных классов.

* 1. Результаты работы программы



* 1. Программа на языке c++
     1. main.cpp

#include <iostream>

#include "COneCTwo.h"

using namespace std;

int main()

{

/\* Демонстрация работы класса COne\*/

cout << "====Demonstration of COne class work====" << endl;

COne COne1;

COne COne2(5, "some initializated string");

cout << "COne1.print()=" << endl;

COne1.print();

cout<< endl;

cout << "COne2.print()=" << endl;

COne2.print();

cout<< endl;

cout << "COne COne3=COne2; COne3.print()=" << endl;

COne COne3=COne2;

COne3.print();

cout<< endl;

cout << "====Demonstration of CTwo class work====" << endl;

CTwo CTwo1;

// cout <<"-------"<<endl;

CTwo CTwo2(6, COne2);

cout << "CTwo1.print()=" << endl;

CTwo1.print();

cout<< endl;

cout << "CTwo2.print()=" << endl;

CTwo2.print();

cout<< endl;

cout << "CTwo CTwo3=CTwo2; CTwo3.print()=" << endl;

// CTwo CTwo3;

CTwo CTwo3=CTwo2;

CTwo3.print();

cout<< endl;

cout << "====Demonstration of CThree class work====" << endl;

CThree CThree1;

CThree1.print();

CThree CThree2(444, COne2, 555);

CThree2.print();

return 0;

}

* + 1. COneCTwo.h

#ifndef CONECTWO\_H\_INCLUDED

#define CONECTWO\_H\_INCLUDED

#include <string>

class COne

{

private:

long l;

std::string s;

public:

/\* Конструкторы и деструктор\*/

COne();

COne(long Nl, std::string Ns);

COne(const COne &COne1);

~COne();

/\* Getters Setters\*/

// long Getl();

// std::string Gets();

void print();

void Setl(long Nl);

void Sets(std::string Ns);

/\* Перегрузка = \*/

COne& operator=(const COne &COne2);

};

class CTwo

{

protected: //Пусть будет protected, иначе не сможем переопределить print()

long l;

COne obj;

public:

/\* Конструкторы и деструктор\*/

CTwo();

CTwo(long Nl, COne Nobj);

CTwo(const CTwo &CTwo1);

~CTwo();

/\* Getters Setters\*/

void print();

void Setl(long Nl);

void Setobj(COne Nobj);

/\* Перегрузка = \*/

CTwo& operator=(const CTwo &CTwo2);

};

class CThree: public CTwo

{

private:

int lab3value;

public:

/\* Конструкторы и деструктор\*/

CThree();

CThree(long Nl, COne Nobj, int Nlab3value);

CThree(const CThree &CThree1);

~CThree();

/\* Getters Setters\*/

void print();

int Getlab3value();

void Setlab3value(int Nlab3value);

};

#endif // CONECTWO\_H\_INCLUDED

* + 1. COneCTwo.cpp

#include "COneCTwo.h"

#include "iostream"

/\* ======================COne=====================\*/

/\* Конструкторы и деструктор\*/

COne::COne():l(1), s("No param s")

{

// std::cout<<"No param constructor COne"<<std::endl;

}

COne::COne(long Nl, std::string Ns):l(Nl), s(Ns)

{

// std::cout<<"Yes param constructor COne"<<std::endl;

}

COne::COne(const COne &COne1)

{

// std::cout<<"Copy constructor COne"<<std::endl;

l=COne1.l;

s=COne1.s;

}

COne::~COne()

{

// std::cout<<"destructor COne"<<std::endl;

}

/\* Getters Setters\*/

//long COne::Getl()

//{

// return l;

//}

//std::string COne::Gets()

//{

// return s;

//}

void COne::print()

{

std::cout << "l=" << l << "\ts=" << s << std::endl;

}

void COne::Setl(long Nl)

{

l=Nl;

}

void COne::Sets(std::string Ns)

{

s=Ns;

}

/\* Перегрузка = \*/

COne& COne::operator=(const COne &COne2)

{

this->l=COne2.l;

this->s=COne2.s;

return \*this;

}

/\* ======================CTwo=====================\*/

/\* Конструкторы и деструктор\*/

CTwo::CTwo():l(123), obj()

{

// std::cout<<"No param constructor CTwo"<<std::endl;

}

CTwo::CTwo(long Nl, COne Nobj):l(Nl), obj(Nobj)

{

// std::cout<<"Yes param constructor CTwo"<<std::endl;

}

CTwo::CTwo(const CTwo &CTwo1)

{

// std::cout<<"Copy constructor CTwo"<<std::endl;

this->l=CTwo1.l;

this->obj=CTwo1.obj;

}

CTwo::~CTwo()

{

// std::cout<<"destructor CTwo"<<std::endl;

}

/\* Getters Setters\*/

void CTwo::print()

{

std::cout << "l=" << l << "\tobj=";

obj.print();

}

void CTwo::Setl(long Nl)

{

l=Nl;

}

void CTwo::Setobj(COne Nobj)

{

obj=Nobj;

}

/\* ======================CThree=====================\*/

/\* Конструкторы и деструктор\*/

CThree::CThree(): lab3value(33) //не вызываем конструкторы родительских(базовых) классов, поскольку они здесь вызовутся по умолчанию

{

std::cout<<"No param constructor CThree"<<std::endl;

}

CThree::CThree(long Nl, COne Nobj, int Nlab3value): CTwo(Nl, Nobj), lab3value(Nlab3value) //вызываем родительский конструктор (всегда выполняется первым, независимо от положения в списке членов инициализации)

{

std::cout<<"Yes param constructor CThree"<<std::endl;

}

CThree::CThree(const CThree &CThree1)

{

std::cout<<"Copy constructor CThree"<<std::endl;

this->lab3value=CThree1.lab3value;

}

CThree::~CThree()

{

std::cout<<"destructor CThree"<<std::endl;

}

/\* Getters Setters\*/

void CThree::print() //переопределенный метод. различие между виртуальным методом и обычным переопределенным проявляется, когда совершется вызов метода при разыменовании указателя на объект. Причем указатель может быть как указателем на дочернего типа, так и указателем родительского типа (оба указывают на один и тот же дочерний объека). В случае указателя родительсого типа поведение виртуального метода и обычного переопределенного будет разным, а именно в случае виртуального метода будет выбираться самый дочерний метод, что и позволяет реализовывать полиморфизм.

{ //короче, виртуальный метод меньше привязан к типу, чем обычный переопределенный.

std::cout<<"l="<< l << "\tobj= ";

obj.print();

std::cout<< "lab3value=" << lab3value <<std::endl;

// std::cout<<std::endl;

}

int CThree::Getlab3value()

{

return lab3value;

}

void CThree::Setlab3value(int Nlab3value)

{

lab3value=Nlab3value;

}

1. Выводы

Изучено отношение наследования классов в языке си++.

Получено представление о значении ключей доступа при наследовании классов в языке Си++.