ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | Е.О. Шумова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| «Множественное наследование в языке С++» |
| по курсу: ОБЪЕКТНО ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134к |  |  |  | Костяков Н.А. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

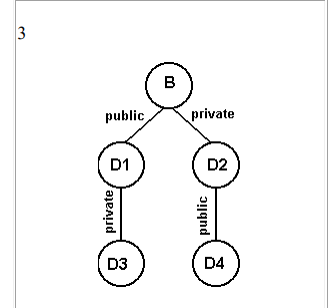
Санкт-Петербург 2022

**Цель работы** Получение практических навыков при использовании множественного наследования в языке С++.

**Описание работы**

В работе необходимо построить иерархию классов согласно схеме наследования, приведенной в варианте задания. Каждый класс должен содержать: - инициализирующий конструктор с выводом значения, полученного в качестве формального параметра - функцию show для вывода значений. - деструктор. При определении производного класса указывать базовые классы, которые являются непосредственно родительскими для данного класса. Функция main должна иллюстрировать иерархию наследования. В функции main создаем объекты производных классов ТОЛЬКО «самого нижнего» уровня в иерархии наследования.

**Вариант 3**



**Листинг:**

#include <iostream>

class B1

{

public:

B1(int z=0);

~B1();

virtual void show();

int z;

protected:

};

B1::B1(int z)

{

this->z = z;

std::cout << "Calling B1 Constructor with param x = "<<z<<"\n";

}

B1::~B1()

{

std::cout << "Calling destructor B1\n";

}

void B1::show()

{

std::cout << "B1 show " << z<< std::endl;

}

class D1:public B1

{

public:

D1(int y, int z);

~D1();

virtual void show() override;

int y;

private:

protected:

};

void D1::show()

{

std::cout << "D1 show " << y << " " << std::endl;

B1::show();

}

D1::D1(int y, int z): B1 (z)

{

this->y = y;

std::cout << "Calling D1 Constructor with param " << y<<" "<< z << "\n";

}

D1::~D1()

{

std::cout << "Calling destructor D1\n";

}

class D2 : private B1

{

public:

D2(int y, int z);

~D2();

void show() override;

int y, z;

private:

int i;

protected:

int j;

};

void D2::show()

{

std::cout << "D2 show private " << y << std::endl;

B1::show();

}

D2::D2(int y, int z) : B1(z)

{

this->y = y;

std::cout << "Calling D2 Constructor with param " << y<< " "<< z << "\n";

}

D2::~D2()

{

std::cout << "Calling destructor D2\n";

}

class D3:private D1

{

public:

D3(int x, int y, int z);

~D3();

void show() override;

private:

int x;

};

D3::D3( int x, int y, int z) : D1(y, z)

{

this->x = x;

std::cout << "calling D3 constructor with params: "<< x <<" "<<y<<" "<<z << '\n';

}

D3::~D3()

{

std::cout << "\nCalling destructor D3\n";

}

void D3::show()

{

std::cout << "D3 show " << x << std::endl;

D1::show();

}

class D4:public D2

{

public:

D4(int x, int y, int z);

~D4();

void show() override;

int x, y, z;

private:

int i;

protected:

int j;

};

void D4::show()

{

std::cout << "D4 show private " << x << std::endl;

D2::show();

}

D4::D4(int x, int y , int z) : D2(y , z)

{

this->x = x;

std::cout << "Calling D4 Constructor with params: " << x << " "<< y<<" "<<z << "\n";

}

D4::~D4()

{

std:: cout << "\nCalling destructor D4\n";

}

int main()

{

system("color f1");

std::cout << "classes B1->D2->D4\n";

D4 point(1 , 2, 3);

point.show();

std::cout << "\n\n";

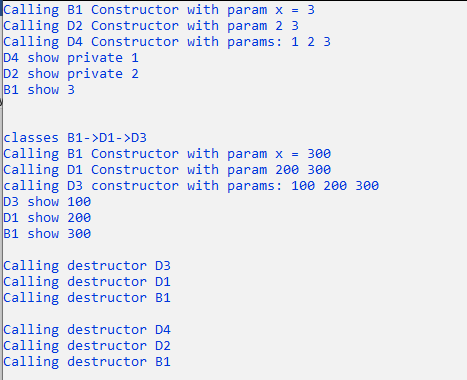
std::cout << "classes B1->D1->D3\n";

D3 point2(100, 200, 300);

point2.show();

}

**Результат работы**



**Вывод**

Я изучил работу с наследованиями и множественными наследованиями