КАФЕДРА №

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №5

Множественное наследование в языке C++

по курсу: Объектно-ориентированное программирование

РАБОТУ ВЫПОЛНИЛ

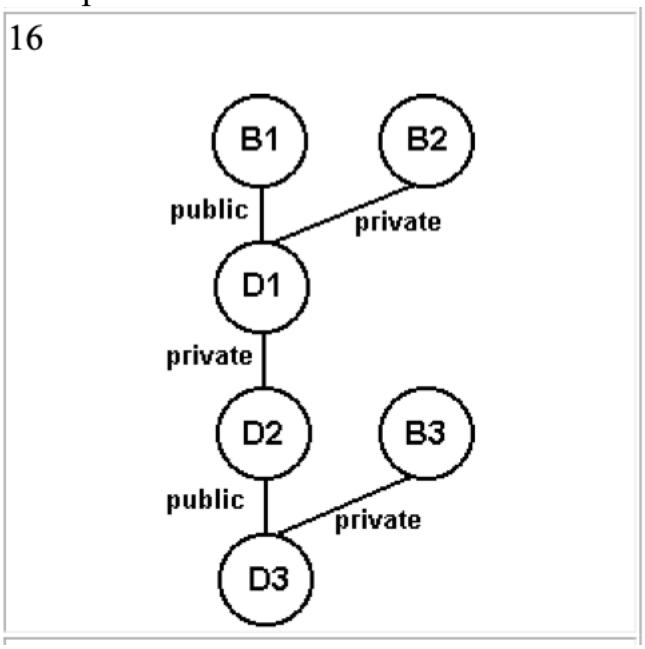
СТУДЕНТ ГР. №

подпись, дата	инициалы, фамилия

Санкт-Петербург 2022

Условие

16 вариант



Цель работы:
Получение практических навыков при использовании множественного наследования в языке C++.

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.

Листинг программы

main.cpp

```
/*
  16 вариант

*/

#include <iostream>
using namespace std;

#include <cmath>

#include "b_classes.h"
#include "d_classes.h"

int main() {
    // смена кодировки
  system("chcp 65001"); // setlocale(LC_ALL, "Russian");
```

```
  D3 d3(10, 20, 30, 40, 50, 60);
  d3.show_D3();

    return 0;
```

b_classes.h

```
#include <iostream>
using namespace std;

class B1 {
public:
  B1(int x);
  B1();
  ~B1();
  void show_B1();
  void set_B1(int);

public:
  int b1;

};

B1::B1(int x) {
  b1 = x;
  cout << "Конструктор с аргументами B1" << endl;
}

B1::B1() {
  b1 = 0;
  cout << "Конструктор по умолчанию B1" << endl;
}


B1::~B1() {
  cout << "Деструктор B1" << endl;
}

void B1::show_B1() {
  cout << "B1 = " << b1 << endl;
}

void B1::set_B1(int x) {
  b1 = x;
}

/**********************/
class B2 {
public:
  B2(int x);
  ~B2();
  void show_B2();
  void set_B2(int);

public:
  int b2;

};
```

```
B2::B2(int x) {
  b2 = x;
  cout << "Конструктор с аргументами B2" << endl;
}

B2::~B2() {
  cout << "Деструктор B2" << endl;
}

void B2::show_B2() {
  cout << "B2 = " << b2 << endl;
}

void B2::set_B2(int x) {
  b2 = x;
}

/**********************/
class B3 {
public:
  B3(int x);
  ~B3();
  void show_B3();
  void set_B3(int);

public:
  int b3;

};

B3::B3(int x) {
  b3 = x;
  cout << "Конструктор с аргументами B3" << endl;
}

B3::~B3() {
  cout << "Деструктор B3" << endl;
}

void B3::show_B3() {
  cout << "B3 = " << b3 << endl;
}

void B3::set_B3(int x) {
  b3 = x;
```

d_classes.h

```
#include <iostream>
using namespace std;

class D1: public B1, private B2 {
public:
  D1(int, int, int);
  ~D1();
  void show_D1();

public:
```

```
  int d1;

};

D1::D1(int x, int y, int z): B1(y), B2(z) {
  d1 = x;
  cout << "Конструктор с аргументами D1" << endl;
}

D1::~D1() {
  cout << "Деструктор D1" << endl;
}

void D1::show_D1() {
  cout << "D1 = " << d1 << endl;
  show_B1();
  show_B2();

}

/*-----------------------------------------------------------------
------------*/

class D2: private D1 {
public:
  D2(int, int, int, int);
  ~D2();
  void show_D2();

public:
  int d2;

};

D2::D2(int x, int y, int z, int i): D1(y, z, i) {
  d2 = x;
  cout << "Конструктор с аргументами D2" << endl;
}

D2::~D2() {
  cout << "Деструктор D2" << endl;
}

void D2::show_D2() {
  cout << "D2 = " << d2 << endl;
  show_D1();

}

/*-----------------------------------------------------------------
------------*/

class D3: public D2, private B3 {
public:
  D3(int, int, int, int, int, int);
  ~D3();
  void show_D3();

public:
  int d3;
```

```
};

D3::D3(int x, int y, int z, int i, int j, int k): D2(y, z, i, j),
B3(k) {
  d3 = x;
  cout << "Конструктор с аргументами D3" << endl;
}

D3::~D3() {
  cout << "Деструктор D3" << endl;
}

void D3::show_D3() {
  cout << "D3 = " << d3 << endl;
  show_D2();
  show_B3();

}
```

Скриншоты

Вывод

Мы получили практические навыки при использовании множественного наследования в языке C++.