Лабораторная работа № 4

**Тема**: Дружеские классы и функции (friend). Правила доступа к членам класса. Полиморфизм.

**Цель**: Изучить технику позднего связывания на примере использования виртуальных функций. Научиться пользоваться возможностью "дружбы" с классом.

**теоретические** **сведения**

**1 Дружественные функции и дружественные классы**

**2 Чистые виртуальные функцii**

**3 Абстрактные классы**

Задание:

1. Напишите код, моделирующий работу двигателя внутреннего сгорания. Придумайте 2 класса, один с именем TEngine (двигатель), другой с именем TFuel (горючее), в котором есть закрытый действительный член двойной точности level, показывающий, сколько горючего залито в баки. С помощью друзей объявите ваши классы так, чтобы класс TEngine имел непосредственный доступ к закрытому члена level класса TFuel.

2. Используя множественную преемственность напишите класс, который способен отразить содержание каталога и запомнить строку, который содержит спецсимволы. Например, инициализация нового объекта-каталога строкой "\* .cpp" приведет к отображению всех файлов с расширением .cpp, размещаемых в текущем каталоге, и к запоминанию самого этого строки следующих ссылок.

3. Напишите несколько связанных между собой классов для программы, которая вводит данные с экрана, вроде базы данных, содержащей имена и адреса. Создайте классы, спрашивают и возвращают значения различных типов данных в определенном месте экрана.

4. Создать абстрактный базовый класс Figure с виртуальными методами вычета площади и периметра. Создать производные классы: Rectandle (прямоугольник), Circle (круг), Trapezium (трапеция) со своими функциями площади и периметра. Самостоятельно определить, какие поля необходимы, какие из них можно задать в базовом классе, а какие - в производных. Площадь трапеции: S = (a + b) \* h / 2.

1. Напишите код, моделирующий работу двигателя внутреннего сгорания. Придумайте 2 класса, один с именем TEngine (двигатель), другой с именем TFuel (горючее), в котором есть закрытый действительный член двойной точности level, показывающий, сколько горючего залито в баки. С помощью друзей объявите ваши классы так, чтобы класс TEngine имел непосредственный доступ к закрытому члена level класса TFuel.

// Листинг TEngine.h

#ifndef TENGINE\_H

#define TENGINE\_H

#include <iostream> // библиотека ввода-вывода

class TEngine; // обьявление класса

class TFuel // класс бензобак

{

friend class TEngine; // TEngine друг для TFuel

public:

TFuel( ); // контсруктор

virtual ~TFuel(); //деструктор

private:

double \_level; // уровень горючего

};

class TEngine // класс двигатель

{

public:

TFuel mot; // обьект класса бензобак

// контсруктор принимающий аргументы

TEngine(const char\* str,int cylinder\_count,int engine\_power,int weight, double level);

virtual ~TEngine();

void show();

private:

char\* \_name; // система

int \_cylinder\_count; // кол-во цилиндров

int \_engine\_power; // мощность

int \_weight; // вес двигателя

};

#endif // TENGINE\_H

// Листинг TEngine.cpp

#include "TEngine.h"

#include <iostream> // библиотека ввода-вывода

#include <cstring> //для strdup

TEngine::TEngine(const char\* str,int cylinder\_count,int engine\_power,int weight, double level)

{

\_name = strdup(str); // выделение памяти под name

\_cylinder\_count=cylinder\_count;// присваивание кол. цилиндров

\_engine\_power = engine\_power; // присваивание мощность мотора

\_weight = weight; // присваивание вес двигателя

mot.\_level = level; // присваивание уровню бензобака

}

TEngine::~TEngine()

{

delete \_name; // удаление указателя

}

void TEngine::show() // отображение хранящихся значений

{

std::cout << "Система питания - : " << \_name;

std::cout << "\nКоличество цилиндров - : " << \_cylinder\_count;

std::cout << "\nМощность двигателя - : " << \_engine\_power;

std::cout << "\nВес двигателя - : " << \_weight;

std::cout << "\nКол-во горючего в баке -: " << mot.\_level;

}

TFuel::TFuel(){}

TFuel::~TFuel(){}

// Листинг main.cpp

#include <iostream> // библиотека ввода-вывода

#include <TEngine.h> // заголовок класса мотор

using namespace std;

int main()

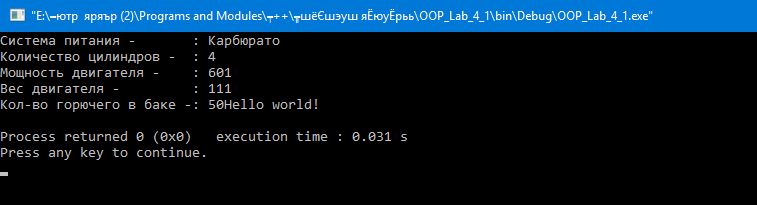
{

setlocale(LC\_ALL, "rus"); // локализация

TEngine Vaz\_21011("Карбюрато", 4 ,601, 111,50 ); // обьект класса мотор

Vaz\_21011.show(); // отображение хранящихся значений

return 0;

}

2. Используя множественную преемственность напишите класс, который способен отразить содержание каталога и запомнить строку, который содержит спецсимволы. Например, инициализация нового объекта-каталога строкой "\* .cpp" приведет к отображению всех файлов с расширением .cpp, размещаемых в текущем каталоге, и к запоминанию самого этого строки следующих ссылок

//Листинг Command.h

#ifndef COMMAND\_H

#define COMMAND\_H

class Entered\_command

{

public:

Entered\_command();

virtual ~Entered\_command();

virtual void delete\_command(){delete \_command;} // освобождение выделенной памяти

virtual void set\_command (const char\* command); // установка значения в \_command

virtual char\* return\_command(){return \_command;} // получения значения в \_command

private:

char\* \_command; // указатель для хранения команды

};

#endif // COMMAND\_H

//Листинг Command.cpp

#include "Command.h"

#include <cstring>

Entered\_command::Entered\_command() { } // Конструктор

Entered\_command::~Entered\_command() { } // Деструктор

void Entered\_command::set\_command(const char\* command)

{

\_command = strdup(command); // выделение памяти по строку и копирование значения

}

//Листинг Ras.h

#ifndef RAS\_H

#define RAS\_H

class Entered\_ras

{

public:

Entered\_ras();

virtual ~Entered\_ras();

virtual void delete\_ras(){delete \_ras\_value;} // освобождение выделенной памяти

virtual void set\_ras (const char\* ras\_value); //установка значения в ras\_value

virtual char\* return\_ras(){return \_ras\_value;} // получения значения в \_ras\_value

private:

char\* \_ras\_value; // указатель расширения для поиска

};

#endif // RAS\_H

//Листинг Ras.cpp

#include "Ras.h"

#include <cstring>

Entered\_ras::Entered\_ras(){ } // Конструктор

Entered\_ras::~Entered\_ras(){}// Деструктор

void Entered\_ras::set\_ras(const char \* ras\_value)

{

\_ras\_value = strdup(ras\_value); // выделение памяти по строку и копирование значения

}

//Листинг Derive\_Class.h

#ifndef DERIVED\_CLASS\_H

#define DERIVED\_CLASS\_H

#include <Command.h> // заголовочный файл команды

#include <Ras.h> // заголовочный файл расширения

class Derived\_Class:public Entered\_command,public Entered\_ras

{

public:

Derived\_Class();

virtual ~Derived\_Class();

void show(); // метод отображения

private:

char \_result\_command[50]; // объединеная строка из команды и расширения

};

#endif // DERIVED\_CLASS\_H

//Листинг Derive\_Class.cpp

#include "Derived\_Class.h"

#include <cstdlib>

#include <cstring>

Derived\_Class::Derived\_Class() // конструктор класса Derived\_Class

{

for(int i=0; i<25; i++) // обнуление массива

\_result\_command[i] = 0;

}

Derived\_Class::~Derived\_Class() //деструктор класса

{

delete\_command(); // функция которая удаляет член класса Entered\_command

delete\_ras(); // функция которая удаляет член класса Entered\_command

}

void Derived\_Class::show() // метод отображения

{

strcat(\_result\_command, return\_command()); // конкатенация команды

strcat(\_result\_command, return\_ras()); // конкатенация расширения

system(\_result\_command); // отображения всех файлов в каталоге

}

//Листинг main.cpp

#include <iostream> // библиотека ввода-вывода

#include <Derived\_Class.h> // потомок от классов команд и расширения

#include <cstdlib> // стандартная библиотека для system

using namespace std;

int main()

{

Derived\_Class \*object = new Derived\_Class; // указательна на обьект класса и в.п.

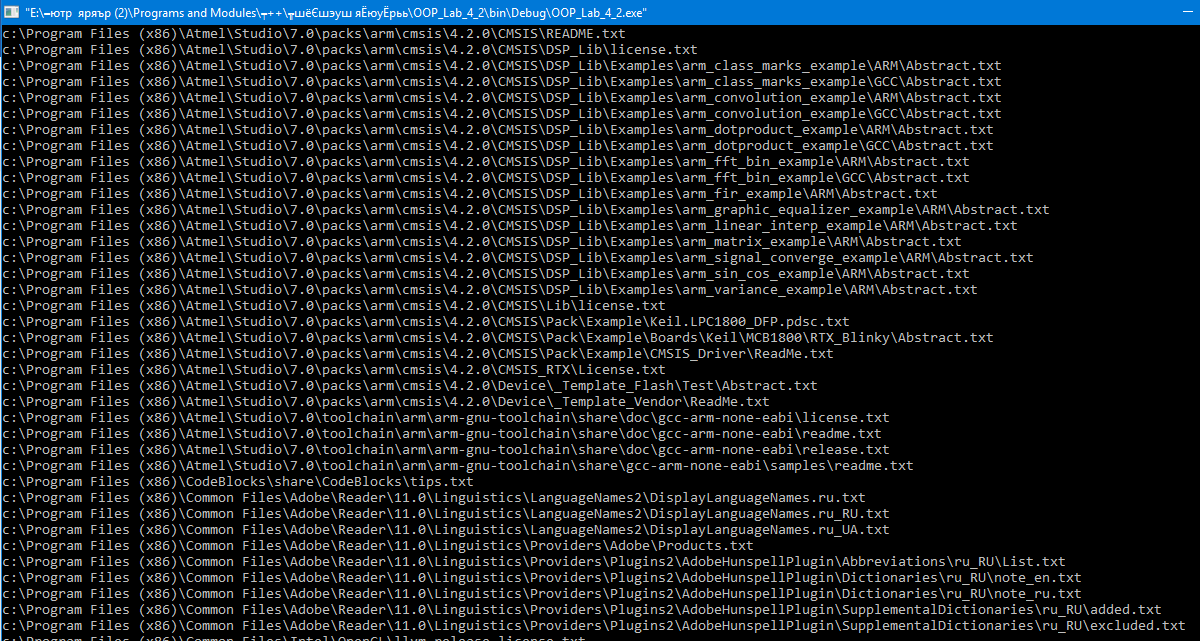
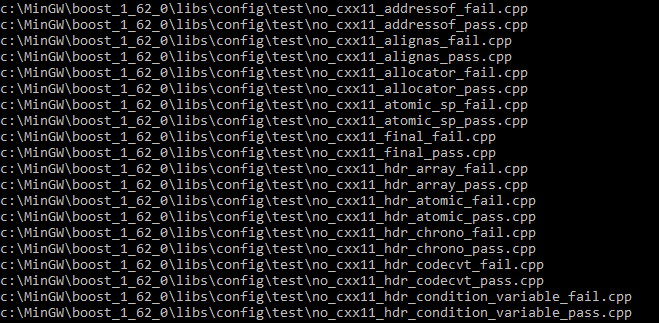
object->set\_command("where/R c:\\ \*."); //вызов вирт функции с класса команд

object->set\_ras("txt"); //вызов вирт функции с класса расширений

object->show(); // отображения каталога

return 0;

}



1. Напишите несколько связанных между собой классов для программы, которая вводит данные с экрана, вроде базы данных, содержащей имена и адреса. Создайте классы, спрашивают и возвращают значения различных типов данных в определенном месте экрана.

// Листинг Base\_Type\_Class.h

#ifndef BASE\_TYPE\_CLASS\_H

#define BASE\_TYPE\_CLASS\_H

class Object\_Name // класс имени

{

public:

Object\_Name(); // конструктор

virtual ~Object\_Name(); // деструктор

virtual void delete\_name(){ delete \_name;} // функция удаления указателя

virtual void set\_name(const char\* name); // установка значений в указатель

virtual char\* return\_name(){return \_name;} // получение значения

private:

char\* \_name; // поле « имя »

};

class Object\_Address // класс адресс

{

public:

Object\_Address(); // конструктор

virtual ~Object\_Address(); // деструктор

virtual void delete\_address(){ delete \_address;} // удаление указателя

virtual void set\_address( const char\* address); // установка значения

virtual char\* return\_address(){return \_address;} // получение значения

private:

char\* \_address; // поле « адреса »

};

class Object\_Age // класс возраст

{

public:

Object\_Age(); // конструктор

virtual ~Object\_Age(); // деструктор

virtual void set\_age( int age); // установка значения

virtual int return\_age(){return \_age;} // получения значения

private:

int \_age; // поле « возраст »

};

#endif // BASE\_TYPE\_CLASS\_H

//Листинг Base\_Type\_Class.cpp

#include "Base\_Type\_Class.h"

#include <cstring> // для strdup

Object\_Name::Object\_Name(){} // конструктор

Object\_Name::~Object\_Name(){} // деструктор

// установка имя

void Object\_Name::set\_name(const char\* name){ name = strdup(name); }

Object\_Address::Object\_Address(){} // конструктор

Object\_Address::~Object\_Address(){}// деструктор

// установка адресса

void Object\_Address::set\_address(const char\* address){\_address = strdup(address);}

Object\_Age::Object\_Age(){} // конструктор

Object\_Age::~Object\_Age(){}// деструктор

void Object\_Age::set\_age(int age) { \_age = age;} // установка возраста

// Листинг Derived.h

#ifndef DERIVED\_CLASS\_H

#define DERIVED\_CLASS\_H

#include <Base\_Type\_Class.h>

class Derived\_Class: public Object\_Name, public Object\_Address, public Object\_Age

{

public:

Derived\_Class(); // конструктор

virtual ~Derived\_Class(); // деструктор

void set\_kortege(const char\* name,

const char\* address,

int age );

void show(); // отображение

private:

static int numeric; // счетчик

};

#endif // DERIVED\_CLASS\_H

// Листинг Derived.cpp

#include "Derived\_Class.h"

#include <iostream> // библиотека ввода-вывода

int Derived\_Class::numeric = 0;

Derived\_Class::Derived\_Class(){numeric++; } // инкремент

Derived\_Class::~Derived\_Class()

{

delete\_address(); // удаление указателя

delete\_name(); // удаление имени

numeric--; // декремент

}

void Derived\_Class::show() // отображение

{

std::cout << " № " << numeric << " | "

<< return\_name()<< " |"

<< return\_address()<< " |"

<< return\_age()<< " age old\n";

}

//Листинг main.cpp

#include <iostream>

#include <Derived\_Class.h>

using namespace std;

int main()

{

setlocale(LC\_ALL,"rus");

Derived\_Class \*object\_1 = new Derived\_Class; // указатель на класс и в.п.

object\_1->set\_name("Arslan"); // установка имени

object\_1->set\_address("Nurana #59"); // установка адресса

object\_1->set\_age(25); // установка возраста

object\_1->show(); // отображение

Derived\_Class \*object\_2 = new Derived\_Class; // указатель на класс и в.п.

object\_2->set\_name("Victor"); // установка имени

object\_2->set\_address("Universitetskaya #16");// установка адресса

object\_2->set\_age(20); // установка возраста

object\_2->show(); // отображение

Derived\_Class \*object\_3 = new Derived\_Class; // указатель на класс и в.п.

object\_3->set\_name("Pavel"); // установка имени

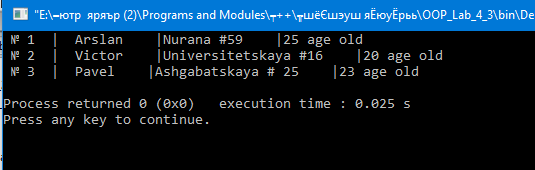
object\_3->set\_address("Ashgabatskaya # 25");// установка адресса

object\_3->set\_age(23); // установка возраста

object\_3->show(); //отображение

return 0;

}



4. Создать абстрактный базовый класс Figure с виртуальными методами вычета площади и периметра. Создать производные классы: Rectandle (прямоугольник), Circle (круг), Trapezium (трапеция) со своими функциями площади и периметра. Самостоятельно определить, какие поля необходимы, какие из них можно задать в базовом классе, а какие - в производных. Площадь трапеции: S = (a + b) \* h / 2.

// Листинг Figure\_Base.h

#ifndef FIGURE\_BASE\_H

#define FIGURE\_BASE\_H

class Figure\_Base

{

public:

Figure\_Base(); // конструктор

virtual ~Figure\_Base(); // деструктор

virtual void show()=0; // отображение значений хранящихся в каждом объекте

virtual void square()=0; // функция для подсчета площади фигуры и присваив. square

virtual void perimeter()=0; // функция для подсчета периметра фигуры и присваив. perimeter

double \_square; // площадь

double \_perimeter; // периметр

};

#endif // FIGURE\_BASE\_H

// Листинг Figure\_Base.cpp

#include "Figure\_Base.h"

Figure\_Base::Figure\_Base() {}

Figure\_Base::~Figure\_Base(){}

//Листинг Figure\_Derived.h

#ifndef FIGURE\_DERIVED\_H

#define FIGURE\_DERIVED\_H

#include <Figure\_Base.h>

// класс прямоугольника

class Rectangle : public Figure\_Base

{

public:

Rectangle(double side\_a, double side\_b);// конструктор принимающий два аргумента

virtual ~Rectangle();

void show(); // отображение значений хранящихся в объекте прямоугольник

void square(); // для подсчета площади прямоугольника и присваив. square

void perimeter();//для подсчета периметра прям. и присваив. perimeter

private:

double \_side\_a; // первая сторона

double \_side\_b; // вторая сторона

};

// класс круга

class Circle : public Figure\_Base

{

public:

Circle (double radiuse);// конструктор принимающий один аргумент

virtual ~Circle ();

void show(); // отображение значений хранящихся в объекте прямоугольник

void square(); // для подсчета площади прямоугольника и присваив. square

void perimeter();//для подсчета периметра прям. и присваив. perimeter

private:

double \_radiuse; // радиус круга

};

// класс трапеции

class Trapezium : public Figure\_Base

{

public:

Trapezium ( double side\_a,

double side\_b,

double height);// конструктор принимающий три аргумента

virtual ~Trapezium ();

void show();// отображение значений хранящихся в объекте прямоугольник

void square();//для подсчета площади прямоугольника и присваив. square

void perimeter();//для подсчета периметра прям. и присваив. perimeter

private:

double \_side\_a; // первая сторона

double \_side\_b; // вторая сторона

double \_height; // высота трапеции

};

#endif // FIGURE\_DERIVED\_H

//Листинг Figure\_Derived.cpp

#include "Figure\_Derived.h"

#include <iostream>

// конструктор класса Regctangle

Rectangle::Rectangle(double side\_a, double side\_b)

{

\_side\_a = side\_a;

\_side\_b = side\_b;

\_square = 0;

\_perimeter =0;

}

// деструктор класса Rectangle

Rectangle::~Rectangle()

{

std :: cout << "Вызов Деструктора";

}

// функция для подсчета площади прямоугольника и присваив. square

void Rectangle::square()

{

\_square = \_side\_a\*\_side\_b;

}

// функция для подсчета периметра прямоугольника и присваив. perimeter

void Rectangle ::perimeter()

{

\_perimeter = 2\*(\_side\_a+\_side\_b);

}

// отображение значений хранящихся в объекте прямоугольник

void Rectangle::show()

{

std::cout << "\nПервая сторона прямоугольника : = " << \_side\_a;

std::cout << "\nВторая сторона прямоугольника : = " << \_side\_b;

std::cout << "\nПлощадь прямоугольника : = " << \_square;

std::cout << "\nПериметр прямоугольника : = " << \_perimeter;

}

// конструктор класса Circle

Circle::Circle(double radiuse )

{

\_radiuse = radiuse;

\_square = 0;

\_perimeter =0;

}

// деструктор класса Circle

Circle::~Circle()

{

std :: cout << "Вызов Деструктора";

}

// функция для подсчета площади прямоугольника и присваив. square

void Circle::square()

{

\_square = 3.14\*\_radiuse\*\_radiuse ;

}

// функция для подсчета периметра прямоугольника и присваив. perimeter

void Circle ::perimeter()

{

\_perimeter = 2\*3.14\*\_radiuse;

}

// отображение значений хранящихся в объекте прямоугольник

void Circle::show()

{

std::cout << "\nРадиус круга : = " << \_radiuse;

std::cout << "\nДлина окружности : = " << \_perimeter;

std::cout << "\nПлощадь круга : = " << \_square;

}

// конструктор класса Trapezium

Trapezium::Trapezium(double side\_a, double side\_b, double height)

{

\_side\_a = side\_a;

\_side\_b = side\_b;

\_height = height;

\_square = 0;

\_perimeter = 0;

}

// деструктор класса Trapezium

Trapezium::~Trapezium()

{

std :: cout << "Вызов Деструктора";

}

// функция для подсчета площади прямоугольника и присваив. square

void Trapezium::square()

{

\_square = ((\_side\_a+\_side\_b)\*\_height)/2 ;

}

// функция для подсчета периметра прямоугольника и присваив. perimeter

void Trapezium ::perimeter()

{

\_perimeter = 2\*(\_side\_a+\_side\_b);

}

// отображение значений хранящихся в объекте прямоугольник

void Trapezium::show()

{

std::cout << "\nПервая сторона трапеции : = " << \_side\_a;

std::cout << "\nВторая сторона трапеции : = " << \_side\_b;

std::cout << "\nВысота трапеции : = " << \_height;

std::cout << "\nПлощадь трапеции : = " << \_square;

std::cout << "\nПериметр трапеции : = " << \_perimeter;

}

//Листинг main.cpp

#include <iostream>

#include <Figure\_Derived.h>

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

// Экземпляр класса прямоугольник

Rectangle \*rectangle = new Rectangle(10,20);

// объявление указателя и выделение памяти для обьекта

rectangle->show();

// отображение значений которые были проинициализированы во время создания обьекта

cout<<"\n\n\n";

rectangle->perimeter(); // подсчет периметра

rectangle->square(); // подсчет площади

rectangle->show(); // отображение конечного результата хранящегося в обьекте

cout<<"\n\n\n";

// Экземпляр класса Circle

Circle \*circle = new Circle(12);

// объявление указателя и выделение памяти для обьекта

circle->show();

// отображение значений которые были проинициализированы во время создания обьекта

cout<<"\n\n\n";

circle->perimeter(); // подсчет периметра

circle->square(); // подсчет площади

circle->show(); // отображение конечного результата хранящегося в обьекте

cout<<"\n\n\n";

// Экземпляр класса Circle

Trapezium \*trapezium = new Trapezium(12, 10,6);

// объявление указателя и выделение памяти для обьекта

trapezium->show(); // отображение значений которые были проинициализированы во время создания обьекта

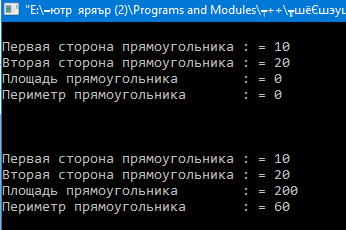
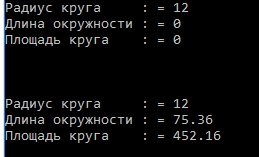
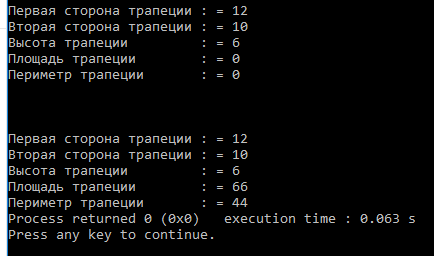
cout<<"\n\n\n";

trapezium->perimeter(); // подсчет периметра

trapezium->square(); // подсчет площади

trapezium->show(); // отображение конечного результата хранящегося в обьекте

return 0;

}

Контрольные вопросы к Л.Р. 4:

1. Какие функции называют дружественными для любого класса?

***Дружественная функция — это функция, которая не является членом класса, но имеет доступ к членам класса, объявленным в полях private или protected***

1. Какие классы называют дружественными для любого класса?

*Класс объявленные дружественным иному классу, то он получает доступ ко всем его закрытых и защищенных членов*.

1. Что нужно сделать, чтобы объявить класс дружественным иному классу?

*Нужно отметить ключевое слово friend в классе, к которому необходимо получить доступ в другом классе.*

1. Какие классы называют взаимно дружественными?

*Два класса обьявлены друг-другу друзьями.*

1. Какие функции называют чистыми виртуальными?

*Функция - которая объявляется в базовом классе, но не имеет в нем определения. Поскольку она не имеет определения, то есть тела в этом базовом классе, то всякий производный класс обязан иметь свою собственную версию определения.*

1. Какой класс называют абстрактным?

*Если класс содержит хотя бы одну чистую виртуальную функцию, он называется абстрактным*

1. Что такое множественная преемственность?

*Производный класс может иметь несколько базовых классов.*

1. Что такое виртуальные базовые классы?

*Обеспечивающие механизм для предотвращения дублирования элементов в классе.*

1. Как вызываются конструкторы виртуальных базовых классов?

*При наследовании виртуального базового класса производным, конструктор виртуального базового класса будет инициализироваться первым*