ПЗ 1.1 Классы. Программирование линейных процессов. Конструкторы, деструкторы, дружественные функции.

**Цель работы:** сформировать навыки решения задач с применением ООП в линейных процессах.

**Задачи:** Изучить особенности работы с ООП.

**Теоретические основы**:

### Классы: основные понятия, данные, методы, конструкторы, свойства

#### Классы

##### Основные понятия

Класс – это структурированный тип данных объектного тина, объединяющий переменные и методы, работающие с ними и реализованный на принципах инкапсуляции, наследования и полиморфизма.

Класс — это обобщенное понятие, определяющие характеристики и поведение некоторого множества объектов, называемых экземплярами класса. "Классический" класс содержит данные, определяющие *свойства объектов класса*, и методы, определяющие их поведение. Для Windows-приложений в класс добавляется третья составляющая - события, на которые может *реагировать объект* класса. Все классы библиотеки .Net, а также все классы, которые создает программист в среде .Net, имеют одного общего предка – класс object.

Все программы, рассмотренные ранее, состояли из одного класса с одним методом Main и несколькими вспомогательными статическими методами. Теперь рассмотрим понятие "класс" более подробно.

Описание класса содержит ключевое слово class, за которым следует его *имя*, а далее в фигурных скобках – *тело* класса. Кроме того, для класса можно задать его базовые классы (предки) и ряд необязательных атрибутов и спецификаторов, определяющих различные характеристики класса:

[ атрибуты ] [ спецификаторы ] class имя\_класса [ : предки ]

{тело\_класса}

Простейший пример класса:

class Demo{}

Спецификаторы определяют свойства класса, а также доступность класса для других элементов программы. Возможные значения спецификаторов перечислены в следующей таблице:

|  |  |  |
| --- | --- | --- |
| **№** | **Спецификатор** | **Описание** |
| 1 | new | Задает новое описание класса взамен унаследованного от предка. Используется для вложения классов (в иерархии объектов). |
| 2 | public | Доступ к классу не ограничен |
| 3 | protected | Доступ только из данного или производного класса. Используется для вложенных классов. |
| 4 | internal | Доступ только из данной программы (сборки). |
| 5 | protected internal | Доступ только из данного и производного класса и из данной программы (сборки). |
| 6 | private | Доступ только из элементов класса, внутри которых описан данный класс. Используется для вложенных классов. |
| 7 | static | Статический класс. Позволяет обращаться к методам класса без создания экземпляра класса |
| 8 | sealed | «Бесплодный» класс. Запрещает наследование данного класса. Применяется в иерархии объектов. |
| 9 | abstract | Абстрактный класс. Применяется в иерархии объектов. |

Спецификаторы 2-6 называются спецификаторами *доступа*. Они определяют, откуда можно непосредственно обращаться к данному классу. Спецификаторы доступа могут комбинироваться с остальными спецификаторами.

**Замечание**. Атрибуты будут рассмотрены позже.

Класс можно описывать непосредственно внутри пространства имен или внутри другого класса. В последнем случае класс называется вложенным. В зависимости от места описания класса некоторые из этих спецификаторов могут быть запрещены. В данном разделе мы рассмотрим классы, которые описываются непосредственно в пространстве имен. Для таких классов допускаются только два спецификатора: public и internal. По умолчанию, то есть если ни один спецификатор доступа не указан, подразумевается спецификатор internal.

Объекты создаются явным или неявным образом, то есть либо программистом, либо системой. Программист создает экземпляр класса с помощью операции new, например:

Demo a = new Demo (); // Создается экземпляр класса Demo

Если достаточный для хранения объекта объем памяти выделить не удалось, то генерируется исключение OutOfMemoryException.

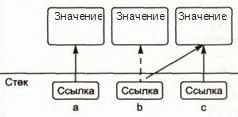
Для каждого объекта при его создании в памяти выделяется отдельная область, в которой хранятся его данные. В классе могут присутствовать *статические элементы*, которые существуют в единственном экземпляре для всех объектов класса. Статические данные часто называют данными класса, а остальные - данными экземпляра. Для работы с данными класса используются статические методы класса, для работы с данными экземпляра - *методы экземпляра*, или просто методы.

До сих пор мы использовали в программах только данные (переменные и константы) и методы. В общем случае класс может содержать следующие функциональные элементы:

1. Данные: переменные или константы.
2. Методы, реализующие не только вычисления, но и другие действия, выполняемые классом или его экземпляром.
3. Конструкторы (реализуют действия по *инициализации экземпляров* или класса в целом).
4. Свойства (определяют характеристики класса в соответствии со способами их задания и получения).
5. Деструкторы (определяют действия, которые необходимо выполнить до того, как объект будет уничтожен).
6. Индексаторы (обеспечивают возможность доступа к элементам класса по их порядковому номеру).
7. Операции (задают действия с объектами с помощью знаков операций).
8. События (определяют уведомления, которые может генерировать класс).
9. Типы (типы данных, внутренние по отношению к классу).

В данном разделе мы рассмотрим первые четыре категории элементов класса

Прежде чем приступить к проектированию классов, необходимо поговорить о присваивании и сравнении объектов. Механизм выполнения присваивания один и тот же для величин любого типа, как ссылочного, так и размерного, однако результаты различаются. При присваивании значения копируется значение, а при присваивании ссылки - ссылка, поэтому после присваивания одного объекта другому мы получим две ссылки, указывающие на одну и ту же область памяти:



Пусть были созданы три объекта а, b и с, а затем выполнено присваивание b = с. Теперь ссылки b и с указывают на один и тот же объект. Старое значение b становится недоступным и очищается сборщиком мусора.

Аналогичная ситуация с операцией проверки на равенство. Величины *значимого типа* равны, если равны их значения. Величины ссылочного типа равны, если они ссылаются на одни и те же данные. Так, объекты b и с равны, т.к. они ссылаются на одну и ту же область памяти. Но, а не равно b даже при равенстве их значений.

##### ***Данные: поля и константы***

Данные, содержащиеся в классе, могут быть *переменными* или *константами* и задаются в соответствии с правилами, рассмотренными в теме "Идентификаторы". При описании данных также можно указывать атрибуты и спецификаторы, задающие различные характеристики элементов. Синтаксис описания элемента данных приведен ниже:

[атрибуты] [спецификаторы] [const] тип имя [ = начальное\_значение ];

Рассмотрим возможные спецификаторы для данных:

|  |  |  |
| --- | --- | --- |
| **№** | **Спецификатор** | **Описание** |
| 1 | new | Новое описание поля, скрывающее унаследованный элемент класса |
| 2 | public | Доступ к элементу не ограничен |
| 3 | protected | Доступ только из данного и производных классов |
| 4 | internal | Доступ только из данной сборки |
| 5 | protected internal | Доступ только из данного и производных классов и из данной сборки |
| 6 | private | Доступ только из данного класса |
| 7 | static | Одно поле для всех экземпляров класса |
| 8 | readonly | Поле доступно только для чтения (значение таких полей можно установить либо при описании, либо в конструкторе) |
| 9 | *volatile* | Поле может изменяться другим процессом или системой |

**Замечание**. Атрибуты будут рассмотрены позже.

Для констант можно использовать только спецификаторы 1-6.

По умолчанию элементы класса считаются закрытыми private. Для полей класса этот вид доступа является предпочтительным, поскольку поля определяют внутреннее строение класса, которое должно быть скрыто от пользователя. Все методы класса имеют непосредственный доступ к его закрытым полям.

Поля, описанные со спецификатором static, а также константы существуют в единственном экземпляре для всех объектов класса, поэтому к ним обращаются не через имя экземпляра, а через имя класса. Обращение к полю класса выполняется с помощью операции доступа (точка). Справа от точки задается имя поля, слева - имя экземпляра для обычных полей или имя класса для статических. Рассмотрим пример создания класса Demo и два способа обращения к его полям.

class Circle

{

public int x=0;

public int y=0;

public int radius=3;

public const double pi = 3.14;

public static string name = "Окружность";

double p;

double s;

}

class Program

{

static void Main()

{

Circle cr = new Circle(); //создание экземпляра класса

Console.WriteLine("pi=" + Circle.pi);// обращение к константе

Console.Write(Circle.name);// обращение к статическому полю

//обращение к обычным полям

Console.WriteLine(" с центром в точке ({0},{1}) и радиусом {2}", cr.x, cr.y, cr.radius);

// Console.WriteLine(cr.p); - вызовет ошибку, т.к. поле p имеет тип private

Console.Write("Введите коэффициент= ");

int kof = int.Parse(Console.ReadLine());

cr.x -= kof; cr.y += kof; cr.radius \*= kof;

Console.WriteLine(" Новая окружность с центром в точке ({0},{1}) и радиусом {2}",

cr.x, cr.y, cr.radius);

//cr.s = 2 \* Circle.pi \* cr.radius; - вызовет ошибку, т.к. поле s имеет тип private

}

}

##### ***Методы***

**Замечание**. Создание и использование методов было рассмотрено нами ранее. Теперь рассмотрим использование методов в контексте создания классов.

Методы находятся в памяти в единственном экземпляре и используются всеми объектами одного класса совместно, поэтому необходимо обеспечить работу методов нестатических экземпляров с полями именно того объекта, для которого они были вызваны. Для этого в любой нестатический метод автоматически передается скрытый параметр this, в котором хранится ссылка на вызвавший функцию экземпляр.

В явном виде параметр this применяется для того, чтобы возвратить из метода ссылку на вызвавший объект, а также для идентификации поля в случае, если его имя совпадает с именем параметра метода, например:

class Circle

{

public int x=0;

public int y=0;

public int radius=3;

public const double pi = 3.14;

public static string name = "Окружность";

public Circle T() //метод возвращает ссылку на экземпляр класса

{

return this;

}

public void Set(int x, int y, int r)

{

this.x = x;

this.y = y;

radius=r;

}

}

class Program

{

static void Main()

{

Circle cr = new Circle(); //создание экземпляра класса

Console.WriteLine("pi=" + Circle.pi);// обращение к константе

Console.Write(Circle.name);// обращение к статическому полю

//обращение к обычным полям

Console.WriteLine(" с центром в точке ({0},{1}) и радиусом {2}", cr.x, cr.y, cr.radius);

cr.Set(1, 1, 10);

Console.WriteLine("Новая окружность с центром в точке ({0},{1}) и радиусом {2}",

cr.x, cr.y, cr.radius);

Circle b=cr.T();//получаем ссылку на объект cr, аналог b=c

Console.WriteLine("Новая ссылка на окружность с центром в точке ({0},{1})

и радиусом {2}", b.x, b.y, b.radius);

}

}

##### ***Конструкторы***

Конструктор предназначен для инициализации объекта. Конструкторы делятся на конструкторы класса (для статических классов) и конструкторы экземпляра класса (всех остальных классов).

###### ***Конструкторы экземпляра***

Конструктор экземпляра вызывается автоматически при создании объекта класса с помощью операции new. Имя конструктора совпадает с именем класса. Рассмотрим основные свойства конструкторов:

1. Конструктор не возвращает значение, даже типа void.
2. Класс может иметь несколько конструкторов с разными параметрами для разных видов инициализации.
3. Если программист не указал ни одного конструктора или какие-то поля не были инициализированы, полям *значимых типов* присваивается нуль, полям ссылочных типов - значение null.

До сих пор мы задавали начальные значения полей класса при описании класса. Это удобно в том случае, когда для всех экземпляров класса начальные значения некоторого поля одинаковы. Если же при создании объектов требуется присваивать полю разные значения, это следует делать с помощью явного задания конструктора. В следующем примере добавлен конструктор и метод Print для вывода информации об объекте:

class Circle

{

public int x;

public int y;

public int radius;

public const double pi = 3.14;

public static string name = "Окружность";

public Circle(int x, int y, int r)//конструктор

{

this.x = x;

this.y = y;

radius = r;

}

public void Print()

{

Console.Write(name);

Console.WriteLine(" с центром в точке ({0},{1}) и радиусом {2}", x, y, radius);

Console.WriteLine();

}

}

class Program

{

static void Main()

{

Circle a = new Circle(0, 0, 1); //вызов конструктора

a.Print();

Circle b=new Circle(10, 10, 5);//вызов конструктора

b.Print();

}

}

Часто бывает удобно задать в классе несколько конструкторов, чтобы обеспечить возможность инициализации объектов разными способами. Для этого конструкторы должны иметь разные сигнатуры. Рассмотрим это на примере:

class Circle

{

public int x;

public int y;

public int radius;

public const double pi = 3.14;

public static string name = "Окружность";

public Circle(int x, int y, int r)//конструктор 1

{

this.x = x;

this.y = y;

radius = r;

}

public Circle(int r)//конструктор 2

{

radius = r;

}

public void Print()

{

Console.Write(name);

Console.WriteLine(" с центром в точке ({0},{1}) и радиусом {2}", x, y, radius);

Console.WriteLine();

}

}

class Program

{

static void Main()

{

Circle a = new Circle(0, 0, 1); //вызов конструктора 1

a.Print();

Circle b=new Circle(5);//вызов конструктора 2

b.Print();

}

}

Обратите внимание на то, что в конструкторе 2 не были инициализированы поля x, y, поэтому им присваивается значение 0.

Если один из конструкторов выполняет какие-либо действия, а другой должен делать то же самое плюс еще что-нибудь, то удобно вызвать первый конструктор из второго. Для этого используется уже известное вам ключевое слово this в другом контексте, например:

public Circle(int x, int y, int r):this(r) //конструктор 1

{

this.x = x;

this.y = y;

}

public Circle(int r) //конструктор 2

{

radius = r;

}

Конструкция, находящаяся после двоеточия, называется *инициализатором*, то есть тем кодом, который исполняется до начала выполнения тела конструктора. В нашем случае конструктор 1 до выполнения своего кода вызывает конструктор 2.

###### ***Конструкторы класса***

Статические классы содержат только статические члены, в том числе и конструктор, которые хранятся в памяти в единственном экземпляре. Поэтому создавать экземпляры класса нет смысла.

Чтобы подчеркнуть этот факт, в первой версии С# для статических классов создавали два конструктора, один - пустой закрытый (private) конструктор, второй - статический конструктор, не имеющий параметров. Первый конструктор предотвращал попытки создания экземпляров класса. Второй конструктор автоматически вызывается системой до первого обращения к любому элементу статического класса, выполняя необходимые действия по инициализации. Вышесказанное отражено в следующем примере:

class Demo

{

static int a;

static int b;

private Demo(){} //закрытый конструктор

static Demo() //статический конструктор

{

a=10;

b=2;

}

public static void Print ()

{

Console.WriteLine("{0}+{1}={2}",a,b,a+b);

Console.WriteLine("{0}\*{1}={2}",a,b,a\*b);

Console.WriteLine("{0}-{1}={2}",a,b,a-b);

}

}

class Program

{

static void Main()

{

//Demo S=new Demo(); //ошибка содать экземпляр класса нельзя

Demo.Print();

}

}

В версию 2.0 введена возможность описывать статический класс, то есть класс с модификатором static. Экземпляры такого класса создавать запрещено, и кроме того, от него запрещено наследовать. Все элементы такого класса должны явным образом объявляться с модификатором static (константы и *вложенные типы* классифицируются как *статические элементы* автоматически). Конструктор экземпляра для статического класса задавать запрещается.

static class Demo

{

static int a=20;

static int b=10;

public static void Print ()

{

Console.WriteLine("{0}+{1}={2}",a,b,a+b);

Console.WriteLine("{0}\*{1}={2}",a,b,a\*b);

Console.WriteLine("{0}-{1}={2}",a,b,a-b);

}

}

class Program

{

static void Main()

{

Demo.Print();

}

}

**Задания по вариантам к ПЗ 1.1**:

Классы. Программирование линейных алгоритмов с использованием функций инициализации set() и вывода результатов print().

Пользовательский класс должен содержать необходимые элементы-данные, метод установки их начальных значений:

Формирование класса начните с конструктора класса;

Void set(double X, …);

метод печати:

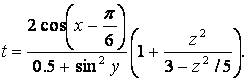
Void print(void);

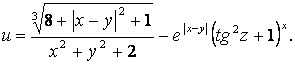
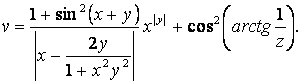
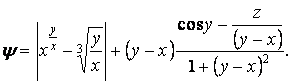
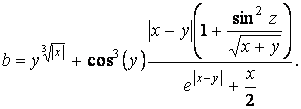
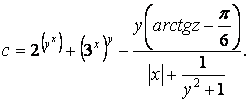
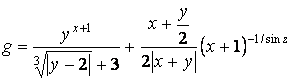
метод, решающий поставленную задачу:

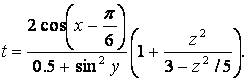
Void Run(void);

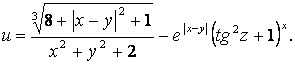
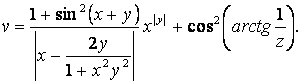
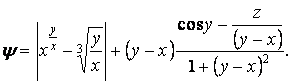
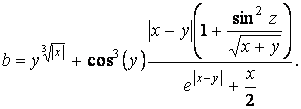
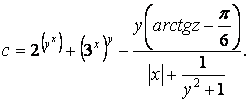
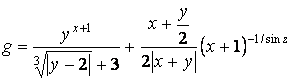
Код методов – вне пространства определения класса. Программа должна включать в себя статический и динамический способы создания объектов, и для каждого объекта использовать прямую и косвенную адресацию при вызове методов класса.

Варианты заданий

  
  
При x=14.26, y=-1.22, z=3.5http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image002_0000.gif **t=0.564849.**

1.   
     
   При x=-4.5, y=0.75http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image002_0001.gif,z=0.845http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image010.gif **u=-55.6848.**
2.   
     
   При x=3.74http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image014.gif, y=-0.825, z=0.16http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image010_0000.gif, **v=1.0553.**
3. http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image017.gif  
     
   При x=0.4http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image019.gif ,y=-0.875, z=-0.475http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image021.gif**w=1.9873.**
4. http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image023.gif  
     
   При x=-15.246, y=4.642http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image014_0000.gif, z=20.001http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image026.gif **http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image028.gif=-182.036.**
5. http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image030.gif  
     
   При x=16.55http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image021_0000.gif, y=-2.75, z=0.15 **http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image033.gif=-40.630.**
6. http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image035.gif  
     
   При x=0.1722, y=6.33, z=3.25http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image037.gif **http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image039.gif=-205.305.**
7. http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image041.gif  
     
   При x=-2.235http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image043.gif, y=2.23, z=15.221 **http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image045.gif=39.374.**
8.   
     
   При x=1.825http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image010_0001.gif, y=18.225, z=-3.298http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image050.gif **http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image052.gif=1.2131.**
9.   
     
   При x=6.251, y=0.827, z=25.001 **b=0.7121.**
10.   
      
    При x=3.251, y=0.325, z=0.466http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image037_0000.gif**c=4.25.**
11. http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image059.gif.  
      
    При x=17.421, y=10.365http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image021_0001.gif, z=0.828http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image062.gif **f=0.33056.**
12. .  
      
    При x=12.3http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image066.gif, y=15.4, z=0.252http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image068.gif**g=82.8257.**
13. http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image070.gif.  
      
    При x=2.444, y=0.869http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image043_0000.gif, z=-0.13http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image068_0000.gif **http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image074.gif-0.49871.**

  
  
При x=14.26, y=-1.22, z=3.5http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image002_0000.gif **t=0.564849.**

1.   
     
   При x=-4.5, y=0.75http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image002_0001.gif,z=0.845http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image010.gif **u=-55.6848.**
2.   
     
   При x=3.74http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image014.gif, y=-0.825, z=0.16http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image010_0000.gif, **v=1.0553.**
3. http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image017.gif  
     
   При x=0.4http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image019.gif ,y=-0.875, z=-0.475http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image021.gif**w=1.9873.**
4. http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image023.gif  
     
   При x=-15.246, y=4.642http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image014_0000.gif, z=20.001http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image026.gif **http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image028.gif=-182.036.**
5. http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image030.gif  
     
   При x=16.55http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image021_0000.gif, y=-2.75, z=0.15 **http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image033.gif=-40.630.**
6. http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image035.gif  
     
   При x=0.1722, y=6.33, z=3.25http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image037.gif **http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image039.gif=-205.305.**
7. http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image041.gif  
     
   При x=-2.235http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image043.gif, y=2.23, z=15.221 **http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image045.gif=39.374.**
8.   
     
   При x=1.825http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image010_0001.gif, y=18.225, z=-3.298http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image050.gif **http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image052.gif=1.2131.**
9.   
     
   При x=6.251, y=0.827, z=25.001 **b=0.7121.**
10.   
      
    При x=3.251, y=0.325, z=0.466http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image037_0000.gif**c=4.25.**
11. http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image059.gif.  
      
    При x=17.421, y=10.365http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image021_0001.gif, z=0.828http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image062.gif **f=0.33056.**
12. .  
      
    При x=12.3http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image066.gif, y=15.4, z=0.252http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image068.gif**g=82.8257.**
13. http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image070.gif.  
      
    При x=2.444, y=0.869http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image043_0000.gif, z=-0.13http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image068_0000.gif **http://gr682423.narod.ru/semestr_7/itpsu/images/lab_clip_image074.gif-0.49871.**