Лабораторная работа №4

ХАРЬКОВСКИЙ НАЦИОНАЛЬНЫЙ АВТОМОБИЛЬНО-ДОРОЖНЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И МЕХАТРОНИКИ

Кафедра информационных технологий и мехатроники

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по проведению лабораторных работ по дисциплине
«Объектно-ориентированное программирование»
для студентов специальности 6.050101 “Компьютерные науки”

Разработчик - доцент кафедры информационных технологий и мехатроники
кандидат технических наук, старший научный сотрудник
Тимонин Владимир Алексеевич

Харків 2015

Лабораторная работа №4
Исследование возможностей интегрированной среды разработки Visual C# для создания приложений с классами. Перегрузка операторов.

Цель работы – исследовать возможности интегрированной среды разработки Visual Studio 2010 и получить практические навыки по созданию приложений, использующие перегрузку операторов в классах.

1. Теоретические сведения

Перегрузка операторов – один из способов реализации полиморфизма, заключающийся в возможности одновременного существования в одной области видимости нескольких различных вариантов применения оператора, имеющих одно и то же имя, но различающихся типами параметров, к которым они применяются.

Для перегрузки операторов используется ключевое слово **operator**, позволяющее создать операторный метод, который определяет действие оператора, связанное с его классом. Существует две формы методов **operator**: одна используется для унарных операторов, а другая — для бинарных.

Общий формат перегрузки для унарного оператора имеет вид:

```
public static тип_возврата operator op(тип_параметра операнд)
        {
          // операции
        }
```

Общий формат перегрузки для бинарного оператора имеет вид:

```
public static тип_возврата operator op(тип_параметра1 операнд1,
                                       тип_параметра2 операнд2)
        {
           // операции
        }
```

где элемент **op** — это оператор (например " + " или " / "), который перегружается. Элемент **тип_возврата** — это тип значения, возвращаемого при выполнении заданной операции. Несмотря на то, что можно выбрать любой тип, тип возвращаемого значения чаще всего будет совпадать с типом класса, для которого этот оператор перегружается. Такая корреляция облегчает использование перегруженного оператора в выражениях. Для унарных операторов операнд передается в элементе **операнд**, а для бинарных — в элементах **операнд1** и **операнд2**. Для унарных операторов тип операнда должен совпадать с классом, для которого определен оператор. Для бинарных операторов - тип хотя бы одного операнда должен совпадать с соответствующим классом.

1.1. Перегрузка бинарных операторов

Пример 1. В нижеприведенном приложении создается класс ThreeD поддержки координат объекта в трехмерном пространстве. Перегруженный оператор "+" выполняет сложение отдельных координат двух ThreeD-объектов, а перегруженный оператор "-" вычитает координаты одного ThreeD-объекта из координат другого. Результат выполнения приложения показан на рис.1.

```
class ThreeD
  {
    int x, y, z; // 3-х-мерные координаты
    public ThreeD() { x = y = z = 0; }
    public ThreeD(int i, int j, int k)
    {
```

```
        x = i; y = j; z = k;
    }
    // Перегрузка бинарного оператора "+"
    public static ThreeD operator +(ThreeD opl, ThreeD op2)
    {
        ThreeD result = new ThreeD();
        // Суммирование координат двух точек и возврат результата
        result.x = opl.x + op2.x;
        result.y = opl.y + op2.y;
        result.z = opl.z + op2.z;
        return result;
    }
    // Перегрузка бинарного оператора "-"
    public static ThreeD operator -(ThreeD opl, ThreeD op2)
    {
        ThreeD result = new ThreeD();
        // Вычитание координат двух точек и возврат результата
        result.x = opl.x - op2.x;
        result.y = opl.y - op2.y;
        result.z = opl.z - op2.z;
        return result;
    }
    // Отображение координат X, Y, Z
    public void show()
    {
        Console.WriteLine(x + ", " + y + ", " + z);
    }
  }
class Program
  {
    static void Main(string[] args)
    {
        ThreeD a = new ThreeD(1, 2, 3);
        ThreeD b = new ThreeD(0, 10, 10);
        ThreeD c = new ThreeD();
        Console.Write("\n  Координаты точки a: " ) ;
        a.show();
        Console.Write("\n  Координаты точки b: " ) ;
        b.show() ;
        c = a + b; // Складываем a и b.
        Console.Write("\n  Результат сложения (a + b): " ) ;
        c.show();
        c = a + b + c; // Складываем a, b и c.
        Console.Write("\n  Результат сложения (a + b + c): " ) ;
        c.show();
        c = c - a; // Вычитаем a из c.
        Console.Write("\n  Результат вычитания (c - a): " ) ;
        c.show();
        c = c - b; // Вычитаем b из c.
        Console.Write("\n  Результат вычитания (c - b): " ) ;
        c.show();
    }
```

}

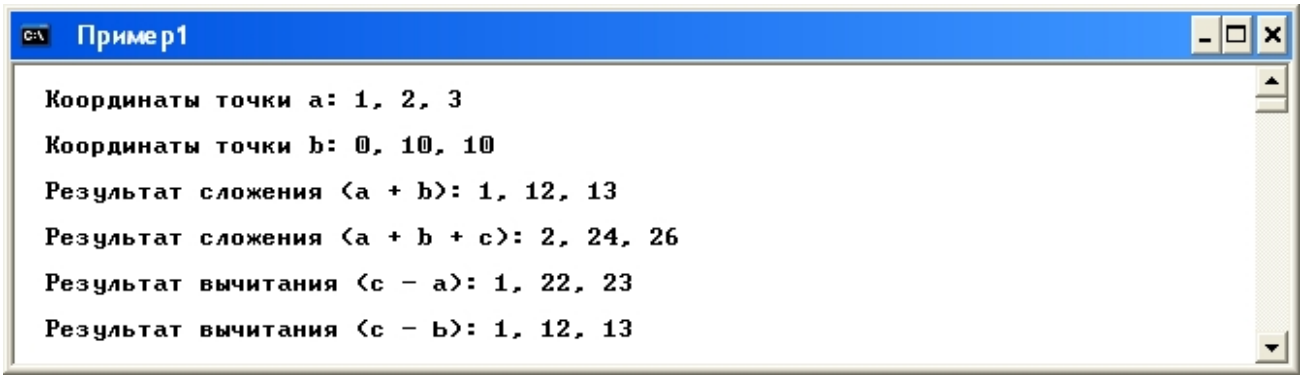

Рис.1. Результат выполнения приложения«Пример1»

При воздействии перегруженного оператора " + " на два объекта типа ThreeD величины соответствующих координат суммируются, как показано в методе operator+(). Этот метод не модифицирует значения ни одного из операндов и возвращает новый объект типа ThreeD, который содержит результат выполнения рассматриваемой операции.

Тот факт, что метод operator+() возвращает объект типа ThreeD, позволяет использовать оператор "+" в таких составных выражениях, как a+b+c, где часть этого выражения a+b, генерирует результат типа ThreeD, который затем суммируется с объектом c. И если бы выражение генерировало значение иного типа (а не типа ThreeD), такое составное выражение попросту не работало бы.

При сложении координат внутри метода operator+() выполняется целочисленное сложение, поскольку отдельные координаты представляют собой целочисленные величины. Факт перегрузки оператора "+" для объектов типа ThreeD не влияет на оператор "+", применяемый к целым числам.

Оператор " - " в операторном методе operator-() работает подобно оператору "+" за исключением того, что здесь важен порядок следования операндов. Для всех бинарных операторов первый параметр операторного метода будет содержать левый операнд, а второй параметр — правый. При реализации перегруженных версий некоммутативных операторов необходимо помнить, какой операнд является левым, а какой — правым.

1.2. Перегрузка унарных операторов

Унарные операторы перегружаются точно так же, как и бинарные. Главное отличие состоит в том, что в этом случае существует только один операнд. Например, метод, который перегружает унарный "минус" для класса ThreeD:

```
public static ThreeD operator -(ThreeD op)
{
   ThreeD result = new ThreeD();
   result.x = -op.x;
   result.y = -op.y;
   result.z = -op.z;
   return result;
}
```

Здесь создается новый объект, который содержит поля операнда, но со знаком "минус". Созданный таким образом объект и возвращается операторным методом operator-(). Обратите внимание на то, что сам операнд остается немодифицированным. Такое поведение соответствует обычному действию унарного "минуса". Например, в выражении a = -b; a получает значение b, взятое с противоположным знаком, но само b при этом не меняется.

В операторах инкремента (++) и декремента (--)операторный метод изменяет содержимое операнда. Поскольку обычно эти операторы выполняют функции инкрементирования и декрементирования значений, соответственно, то перегруженные операторы "++" и " -- " , как

правило, инкрементируют свой операнд. Таким образом, при перегрузке этих операторов операнд обычно модифицируется. Например, метод, который перегружает оператор ++ для класса ThreeD:

```
public static ThreeD operator ++(ThreeD op)
{
   // Оператор "++" модифицирует аргумент
   op.x++;
   op.y++;
   op.z++;
   return op;
}
```

В результате выполнения этого операторного метода объект, на который ссылается операнд op, модифицируется, т.е. операнд, подвергнутый операции "++", инкрементируется. Кроме того, модифицированный объект возвращается этим методом, благодаря чему оператор "++" можно использовать в более сложных выражениях.

Пример 2. Ниже приведена расширенная версия предыдущего примера программы, которая, помимо прочего, демонстрирует определение и использование унарных операторов "-" и "++". Результат выполнения приложения показан на рис.2.

```
class ThreeD
  {
    int x, y, z; // 3-х-мерные координаты
    public ThreeD() { x = y = z = 0; }
    public ThreeD(int i, int j, int k)
    {
       x = i; y = j; z = k;
    }
    // Перегрузка бинарного оператора "+"
    public static ThreeD operator +(ThreeD opl, ThreeD op2)
    {
       ThreeD result = new ThreeD();
       // Суммирование координат двух точек и возврат результата
       result.x = opl.x + op2.x;
       result.y = opl.y + op2.y;
       result.z = opl.z + op2.z;
       return result;
    }
    // Перегрузка бинарного оператора "-"
    public static ThreeD operator -(ThreeD opl, ThreeD op2)
    {
       ThreeD result = new ThreeD();
       // Вычитание координат двух точек и возврат результата
       result.x = opl.x - op2.x;
       result.y = opl.y - op2.y;
       result.z = opl.z - op2.z;
       return result;
    }
    // Перегрузка унарного оператора "-".
    public static ThreeD operator -(ThreeD op)
    {
       ThreeD result = new ThreeD();
       result.x = -op.x;
```

```
        result.y = -op.y;
        result.z = -op.z;
        return result;
      }
      // Перегрузка унарного оператора "++".
      public static ThreeD operator ++(ThreeD op)
      {
        // Оператор "++" модифицирует аргумент
        op.x++;
        op.y++;
        op.z++;
        return op;
      }
      // Отображаем координаты X, Y, Z
      public void show()
      {
        Console.WriteLine(x + ", " + y + ", " + z);
      }
    }
class Program
    {
      static void Main(string[] args)
      {
        ThreeD a = new ThreeD(1, 2, 3);
        ThreeD b = new ThreeD(0, 10, 10);
        ThreeD c = new ThreeD();
        Console.Write("\n  Координаты точки a: " ) ;
        a.show();
        Console.Write("\n  Координаты точки b: " ) ;
        b.show() ;
        c = a + b; // Складываем a и b.
        Console.Write("\n  Результат сложения (a + b): " ) ;
        c.show();
        c = a + b + c; // Складываем a, b и c.
        Console.Write("\n  Результат сложения (a + b + c): " ) ;
        c.show();
        c = c - a; // Вычитаем a из c.
        Console.Write("\n  Результат вычитания (c - a): " ) ;
        c.show();
        c = c - b; // Вычитаем b из c.
        Console.Write("\n  Результат вычитания (c - Ь): " ) ;
        c.show();
        c = -a; // Присваивание -a объекту c.
        Console.Write("\n  Результат присваивания -a: ");
        c.show();
        a++; // Инкрементирование a.
        Console.Write("\n  Результат инкрементирования a++: ");
        a.show();
      }
    }
```

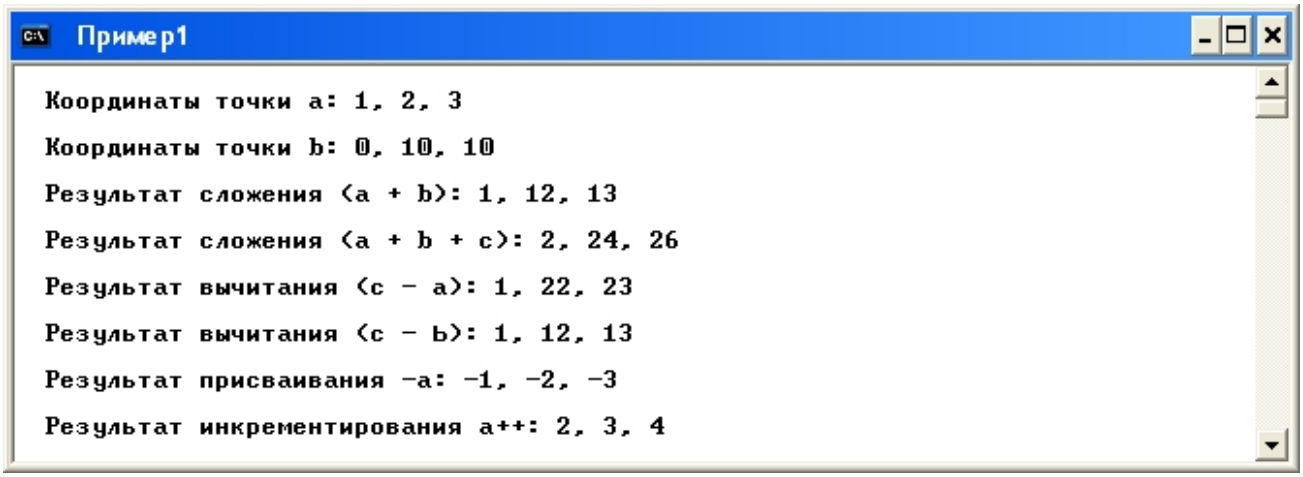

Рис.2. Результат выполнения модифицированного приложения«Пример1»

Как известно, операторы "++" и "--" имеют как префиксную, так и постфиксную форму. Например, инструкции ++a и a++ представляют собой допустимое использование оператора инкремента. Однако при перегрузке оператора "++" обе формы вызывают один и тот же метод. Следовательно, в этом случае невозможно отличить префиксную форму оператора "++" от постфиксной. Это касается и перегрузки оператора "--".

1.3. Выполнение операций над значениями встроенных C#-типов

Для любого заданного класса и оператора любой операторный метод сам может перегружаться Одна из самых распространенных причин этого — разрешить операции между объектами этого класса и другими (встроенными) типами данных. В качестве примера возьмем класс ThreeD. В примере перегруженный оператор "+" суммирует координаты одного ThreeD-объекта с координатами другого. Однако это не единственный способ определения операции сложения для класса ThreeD. Например, может потребоваться суммирование какого-либо целого числа с каждой координатой ThreeD-объекта. Ведь тогда эту операцию можно использовать для смещения осей. Для ее реализации необходимо перегрузить оператор "+" еще раз, например, так:

```
public static ThreeD operator +(ThreeD opl, int op2)
{
   ThreeD result = new ThreeD();
    result.x = opl.x + op2;
    result.y = opl.y + op2;
    result.z = opl.z + op2;
    return result;
}
```

Обратите внимание на то, что второй параметр имеет тип int. Таким образом, этот метод позволяет сложить int-значение с каждым полем ThreeD-объекта. Это вполне допустимо, поскольку, как разъяснялось выше, при перегрузке бинарного оператора тип только одного из его операндов должен совпадать с типом класса, для которого перегружается этот оператор. Другой операнд может иметь любой тип.

Пример 3. Ниже приведена версия класса ThreeD, которая имеет два перегруженных метода operator+(). Результат выполнения приложения показан на рис.3.

```
class ThreeD
  {
    int x, y, z; // 3-х-мерные координаты
    public ThreeD() { x = y = z = 0; }
    public ThreeD(int i, int j, int k)
```

```
    {
        x = i; y = j; z = k;
    }
    // Перегружаем бинарный оператор " + " для варианта "объект + объект"
    public static ThreeD operator +(ThreeD opl, ThreeD op2)
    {
        ThreeD result = new ThreeD();
         result.x = opl.x + op2.x;
         result.y = opl.y + op2.y;
         result.z = opl.z + op2.z;
         return result;
    }
    // Перегружаем бинарный оператор "+" для варианта "объект + int-значение"
    public static ThreeD operator +(ThreeD opl, int op2)
    {
        ThreeD result = new ThreeD();
        result.x = opl.x + op2;
        result.y = opl.y + op2;
        result.z = opl.z + op2;
        return result;
    }
    // Отображаем координаты X, Y, Z
    public void show()
    {
        Console.WriteLine(x + ", " + y + ", " + z);
    }
}
class Program
{
    static void Main(string[] args)
    {
        ThreeD a = new ThreeD(1, 2, 3);
        ThreeD b = new ThreeD(10, 10, 10);
        ThreeD c = new ThreeD();
        Console.Write("\n  Координаты точки a: ");
        a.show();
        Console.Write("\n  Координаты точки b: ");
        b.show();
        c = a + b; // объект + объект
        Console.Write("\n  Результат сложения (a + b): ");
        c.show();
        c = b + 10; // объект + int-значение
        Console.Write("\n  Результат сложения (b + 10): ");
        c.show();
    }
}
```

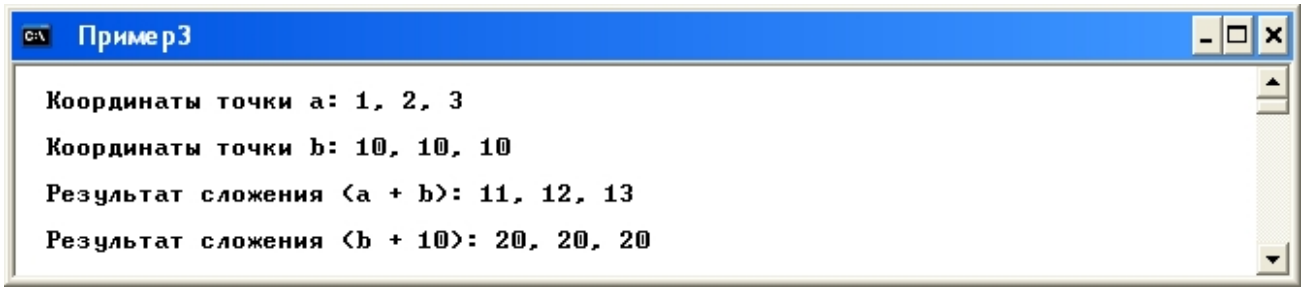

Рис.3. Результат выполнения приложения«Пример3»

Как подтверждают результаты выполнения этой программы, если оператор "+" применяется к двум объектам, их соответствующие координаты суммируются. А если оператор "+" применяется к объекту и целому числу, то значения координат объекта увеличиваются на это целое число.

Необходимо заметить, что метод operator+(ThreeD, int) позволяет выполнять инструкции:

ob1 = ob2 + 10;

но он не позволяет выполнять инструкции:

ob1 = 10 + ob2;

Дело в том, что целочисленное значение принимается в качестве второго аргумента, которым является правый операнд. А в предыдущей инструкции целочисленный аргумент находится слева. Чтобы сделать допустимыми две формы инструкций, необходимо перегрузить оператор "+" еще раз. Новая версия должна будет в качестве первого параметра принимать значение типа int, а в качестве второго — объект типа ThreeD. И тогда старая версия метода operator+() будет обрабатывать вариант "объект + int-значение", а новая — вариант "int-значение + объект". Перегрузка оператора "+" (или любого другого бинарного оператора), выполненная подобным образом, позволит значению встроенного типа находиться слева или справа от оператора.

Пример 4. Ниже приводится версия класса ThreeD, которая перегружает оператор "+" с учетом описанных выше вариантов приема аргументов. Результат выполнения приложения показан на рис4.

```
class ThreeD
  {
    int x, y, z; // 3-х-мерные координаты
    public ThreeD() { x = y = z = 0; }
    public ThreeD(int i, int j, int k)
    {
      x = i; y = j; z = k;
    }
    // Перегружаем бинарный оператор " + " для варианта "объект + объект"
    public static ThreeD operator +(ThreeD opl, ThreeD op2)
    {
     ThreeD result = new ThreeD();
      result.x = opl.x + op2.x;
      result.y = opl.y + op2.y;
      result.z = opl.z + op2.z;
      return result;
    }
    // Перегружаем бинарный оператор "+" для варианта "объект + int-значение"
    public static ThreeD operator +(ThreeD opl, int op2)
    {
      ThreeD result = new ThreeD();
      result.x = opl.x + op2;
      result.y = opl.y + op2;
```

```
        result.z = opl.z + op2;
        return result;
    }
    // Перегружаем бинарный оператор "+" для варианта "int-значение + объект"
    public static ThreeD operator +(int opl, ThreeD op2)
    {
        ThreeD result = new ThreeD();
        result.x = op2.x + opl;
        result.y = op2.y + opl;
        result.z = op2.z + opl;
        return result;
    }
    // Отображаем координаты X, Y, Z
    public void show()
    {
        Console.WriteLine(x + ", " + y + ", " + z);
    }
}
class Program
{
    static void Main(string[] args)
    {
        ThreeD a = new ThreeD(1, 2, 3);
        ThreeD b = new ThreeD(10, 10, 10);
        ThreeD c = new ThreeD();
        Console.Write("\n  Координаты точки a: ");
        a.show();
        Console.Write("\n  Координаты точки b: ");
        b.show();
        c = a + b; // объект + объект
        Console.Write("\n  Результат сложения (a + b): ");
        c.show();
        c = b + 10; // объект + int-значение
        Console.Write("\n  Результат сложения (b + 10): ");
        c.show();
        c = 15 + b; // int-значение + объект
        Console.Write("\n  Результат сложения (15 + b): ");
        c.show();
    }
}
```

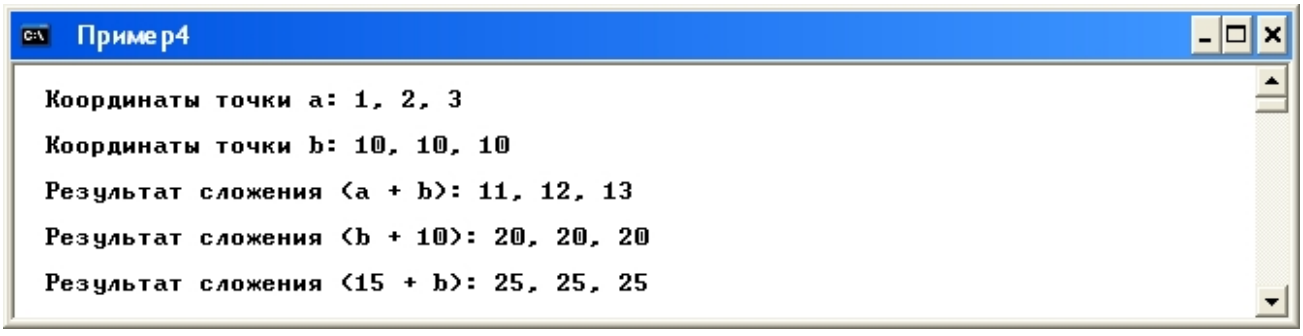

Рис.4. Результат выполнения приложения«Пример4»

1.4. Перегрузка операторов отношений

Операторы отношений (например, "==" или "<") также можно перегружать, причем сделать это совсем нетрудно. Как правило, перегруженный оператор отношения возвращает одно из двух возможных значений: true или false. Это позволяет использовать перегруженные операторы отношений в условных выражениях. Если бы они возвращали результат другого типа, это бы весьма ограничило круг их применения.

Пример 5. Нижеприведенная версия класса ThreeD перегружает операторы "<" и ">". Результат выполнения приложения показан на рис.5.

```
class ThreeD
  {
    int x, y, z; // 3-х-мерные координаты
    public ThreeD() { x = y = z = 0; }
    public ThreeD(int i, int j, int k)
    {
      x = i; y = j; z = k;
    }
    // Перегрузка оператора "<"
    public static bool operator <(ThreeD opl, ThreeD op2)
    {
      if ((opl.x < op2.x) && (opl.y < op2.y) &&
      (opl.z < op2.z))
        return true;
      else
        return false;
    }
    // Перегрузка оператора ">"
    public static bool operator >(ThreeD opl, ThreeD op2)
    {
      if ((opl.x > op2.x) && (opl.y > op2.y) &&
      (opl.z > op2.z))
        return true;
      else
        return false;
    }
    // Отображаем координаты X, Y, Z
    public void show()
    {
      Console.WriteLine(x + ", " + y + ", " + z);
    }
  }
  class Program
  {
    static void Main(string[] args)
    {
      ThreeD a = new ThreeD(5, 6, 7);
      ThreeD b = new ThreeD(10, 10, 10);
      ThreeD c = new ThreeD(1, 2, 3);
      Console.Write("\n  Координаты точки a: ");
      a.show();
      Console.Write("\n  Координаты точки b: ");
```

```
        b.show();
        Console.Write("\n  Координаты точки c: ");
        c.show();
        if (a > c) Console.WriteLine("\n  a > c - ИСТИНА");
        if (a < c) Console.WriteLine("\n  a < c - ИСТИНА");
        if (a > b) Console.WriteLine("\n  a > b - ИСТИНА");
        if (a < b) Console.WriteLine("\n  a < b - ИСТИНА");
    }
}
```

Рис.5. Результат выполнения приложения«Пример5»

На перегрузку операторов отношений налагается серьезное ограничение: их следует перегружать парами (“==” и “!=”, “<” и “>”, “<=” и “>=”). Например, перегружая оператор "<", вы также должны перегрузить оператор ">", и наоборот. Перегружая операторы "==" и " ! = ", следует перегрузить также методы Object.Equals() и Object.GetHashCode().

1.5. Перегрузка операторов true и false

Ключевые слова true и false в целях перегрузки также можно использовать в качестве унарных операторов. Перегруженные версии этих операторов обеспечивают специфическое определение понятий ИСТИНА и ЛОЖЬ в отношении создаваемых программистом классов. Если для класса реализовать таким образом ключевые слова true и false, то затем объекты этого класса можно использовать для управления инструкциями if, while, for и do-while, а также в ?-выражении. Их можно даже использовать для реализации специальных типов логики (например, нечеткой логики).

Операторы true и false должны быть перегружены в паре. Нельзя перегружать только один из них. Оба они выполняют функцию унарных операторов и имеют такой формат:

```
public static bool operator true{тип_параметра op)
        {
            // Возврат значения true или false
        }
public static bool operator false{тип_параметра op)
        {
            // Возврат значения true или false
        }
```

Пример 6. В следующем примере демонстрируется один из возможных способов реализации операторов true и false для класса ThreeD. Предполагается, что ThreeD-объект истинен, если, по крайней мере, одна его координата не равна нулю. Если все три координаты равны нулю, объект считается ложным. Результат выполнения приложения показан на рис.6.

```
class ThreeD
  {
    int x, y, z; // 3-х-мерные координаты
```

```
    public ThreeD() { x = y = z = 0; }
    public ThreeD(int i, int j, int k)
    {
        x = i; y = j; z = k;
    }
    // Перегружаем оператор true
    public static bool operator true(ThreeD op)
    {
        if((op.x != 0) || (op.y != 0) || (op.z != 0))
            return true; // Хотя бы одна координата не равна 0
         else return false;
    }
    // Перегружаем оператор false
    public static bool operator false(ThreeD op)
    {
        if((op.x == 0) && (op.y == 0) && (op.z == 0))
            return true; // Все координаты равны нулю
        else return false;
    }
    // Перегружаем унарный оператор " -- "
    public static ThreeD operator --(ThreeD op)
    {
        op.x--;
        op.y--;
        op.z--;
        return op;
    }
    // Отображаем координаты X, Y, Z
    public void show()
    {
        Console.WriteLine("  " + x + ", " + y + ", " + z);
    }
}
class Program
{
    static void Main(string[] args)
    {
        ThreeD a = new ThreeD(5, 6, 7);
        ThreeD b = new ThreeD(10, 10, 10);
        ThreeD c = new ThreeD(0, 0, 0);
        Console.Write("\n  Координаты точки a: ");
        a. show();
        Console.Write("\n  Координаты точки b: ");
        b.show();
        Console.Write("\n  Координаты точки c: ");
        c.show();
        if (a) Console.WriteLine("\n  a - это ИСТИНА.");
        else Console.WriteLine("  a - это ЛОЖЬ.");
        if(b) Console.WriteLine("  b - это ИСТИНА.");
        else Console.WriteLine("  b - это ЛОЖЬ.");
        if(c) Console.WriteLine("  c - это ИСТИНА.");
        else Console.WriteLine("  c - это ЛОЖЬ.");
```

```
        Console.WriteLine("\n  Управляем циклом, используя объект класса ThreeD.");
        do {
           b.show();
           b--;
        } while(b);
     }
  }
```

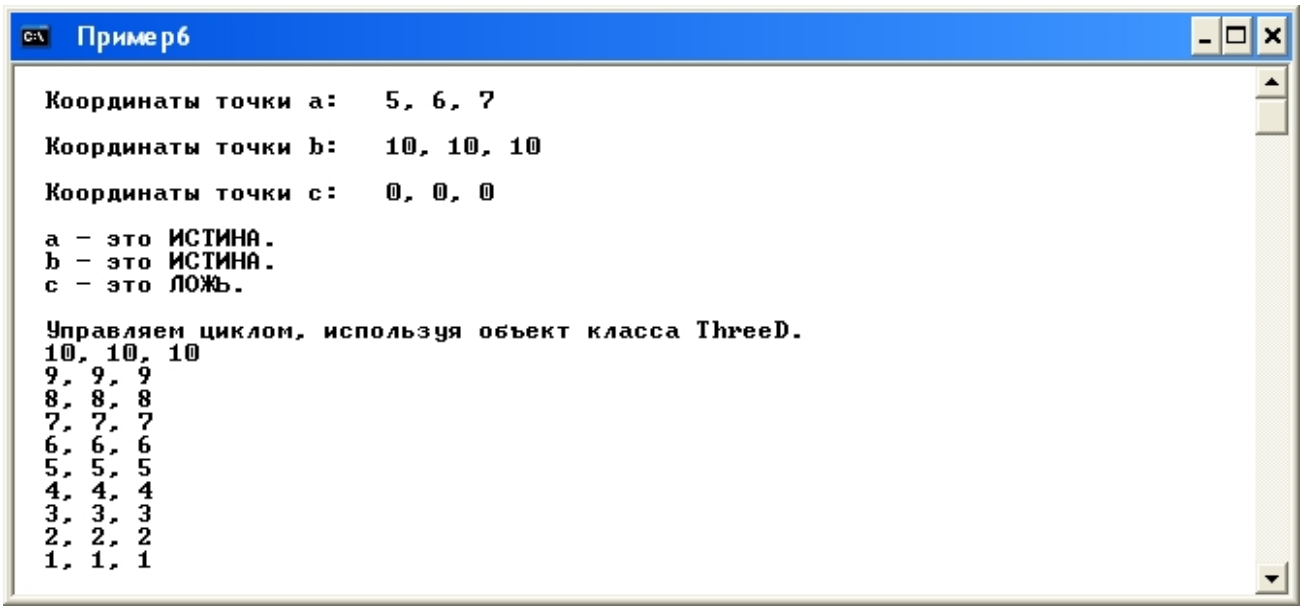

Рис.6. Результат выполнения приложения«Пример6»

Обратите внимание на то, что объекты класса ThreeD используются для управления if-инструкциями и while-цикла. Что касается if-инструкций, то ThreeD-объект оценивается с помощью ключевого слова true. В случае истинности результата этой операции выполняется соответствующая инструкция. В случае do-while-цикла каждая его итерация декрементирует значение объект b. Цикл выполняется до тех пор, пока значение объекта b оценивается как ИСТИНА (т.е. содержит по крайней мере одну ненулевую координату). Когда все координаты объекта b станут равными нулю, он (объект) будет считаться ложным (благодаря оператору true), и цикл прекратится.

1.6. Перегрузка логических операторов

В C# определены следующие логические операторы: &, |, !, && и ||, из которых перегруженными могут быть только &, | , !. Однако при соблюдении определенных правил можно использовать и операторы && и || , действующие по сокращенной схеме.

1.6.1. Простой случай перегрузки логических операторов

Если не планируется использовать логические операторы, работающие по сокращенной схеме, то можете перегружать операторы & и | по своему усмотрению, причем каждый вариант должен возвращать результат типа bool. Перегруженный оператор ! также, как правило, возвращает результат типа bool.


Пример 7. Рассмотрим пример перегрузки логических операторов &, |, ! для объектов типа ThreeD. Как и прежде, в каждом из них предполагается, что ThreeD-объект является истинным, если хотя бы одна его координата не равна нулю. Если же все три координаты равны нулю, объект считается ложным.

```
class ThreeD
  {
```

```
    int x, y, z; // 3-х-мерные координаты
    public ThreeD() { x = y = z = 0; }
    public ThreeD(int i, int j, int k)
    {
        x = i; y = j; z = k;
    }
    // Перегрузка оператора "|"
    public static bool operator |(ThreeD opl, ThreeD op2)
    {
        if( ((opl.x != 0) || (opl.y != 0) || (opl.z != 0)) |
            ((op2.x != 0) || (op2.y != 0) || (op2.z != 0)))
            return true;
        else return false;
    }
    // Перегрузка оператора "&"
    public static bool operator &(ThreeD opl, ThreeD op2)
    {
        if (((opl.x != 0) && (opl.y != 0) && (opl.z != 0)) &
            ((op2.x != 0) && (op2.y != 0) && (op2.z != 0)))
            return true;
        else return false;
    }
    // Перегрузка оператора "!"
    public static bool operator !(ThreeD op)
    {
        if ((op.x != 0) || (op.y != 0) || (op.z != 0))
            return false;
        else return true;
    }
    // Отображаем координаты X, Y, Z
    public void show()
    {
        Console.WriteLine("  " + x + ", " + y + ", " + z);
    }
}
class Program
{
    static void Main(string[] args)
    {
        ThreeD a = new ThreeD(5, 6, 7);
        ThreeD b = new ThreeD(10, 10, 10);
        ThreeD c = new ThreeD(0, 0, 0);
        Console.Write("\n  Координаты точки a: ");
        a. show();
        Console.Write("\n  Координаты точки b: ");
        b.show();
        Console.Write("\n  Координаты точки c: ");
        c.show();
        if (!a) Console.WriteLine("\n  a - ЛОЖЬ.");
        if (!b) Console.WriteLine("\n  b - ЛОЖЬ.");
        if (!c) Console.WriteLine("\n  c - ЛОЖЬ.");
        Console.WriteLine();
```

```
        if (a & b) Console.WriteLine("  a & b - ИСТИНА.");
        else Console.WriteLine("  a & b - ЛОЖЬ.");
        if (a & c) Console.WriteLine("  a & c - ИСТИНА.");
        else Console.WriteLine("  a & c - ЛОЖЬ.");
        if (a | b) Console.WriteLine("  a | b - ИСТИНА.");
        else Console.WriteLine("  a | b - ЛОЖЬ.");
        if (a | c) Console.WriteLine("  a | c - ИСТИНА.");
        else Console.WriteLine("  a | c - ЛОЖЬ.");
    }
}
```

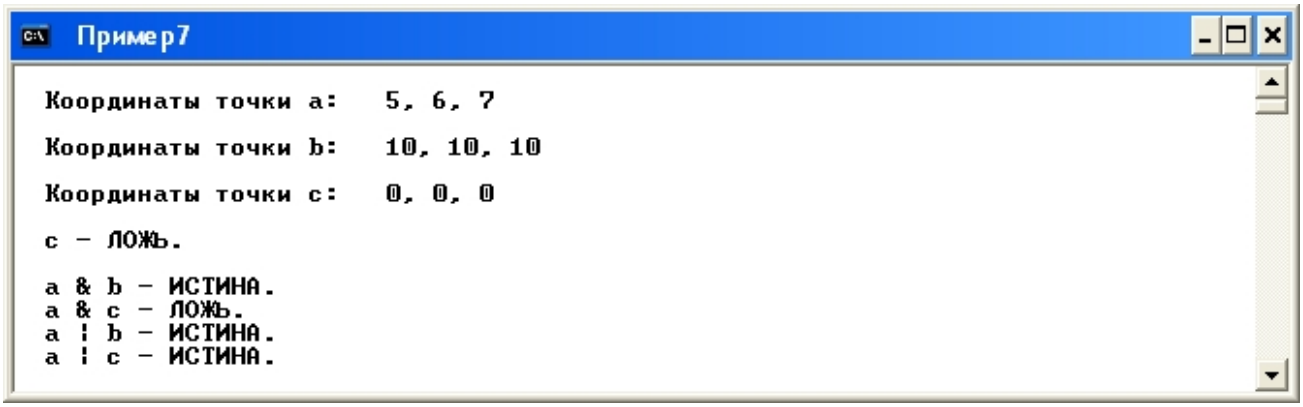

Рис.7. Результат выполнения приложения«Пример7»

В этом примере методы operator | (), operator &() и operator ! () возвращают результат типа bool. Это необходимо в том случае, если перечисленные операторы должны использоваться в своем обычном "амплуа" (т.е. там, где ожидается результат типа bool). Вспомните, что для всех встроенных типов результат выполнения логической операции представляет собой значение типа bool. Таким образом, вполне логично, что перегруженные версии этих операторов должны возвращать значение типа bool. К сожалению, такая логика работает в случае, если нет необходимости в операторах, работающих по сокращенной схеме вычислений.

1.6.2. Включение операторов, действующих по сокращенной схеме вычислений

Чтобы иметь возможность использовать операторы && и ||, действующие по сокращенной схеме вычислений, необходимо соблюдать четыре правила. Во-первых, класс должен перегружать операторы & и |. Во-вторых, &- и |-методы должны возвращать объект класса, для которого перегружаются эти операторы. В-третьих, каждый параметр должен представлять собой ссылку на объект класса, для которого перегружаются эти операторы. В-четвертых, тот же класс должен перегружать операторы true и false. При соблюдении всех этих условий операторы сокращенной схемы действия автоматически становятся доступными для применения.

Пример 7.2. В следующей программе показано, как реализовать операторы & и | для класса ThreeD, чтобы можно было использовать операторы && и ||, действующие по сокращенной схеме вычислений Это более удачный способ реализации операторов !, | и & для класса ThreeD. Эта версия автоматически делает работоспособными операторы && и ||.

```
class ThreeD
  {
    int x, y, z; // 3-х-мерные координаты
    public ThreeD() { x = y = z = 0; }
    public ThreeD(int i, int j, int k)
```

```
    {
        x = i; y = j; z = k;
    }
    // Перегружаем оператор "|" для вычислений по сокращенной схеме
    public static ThreeD operator |(ThreeD opl, ThreeD op2)
    {
        if( ((opl.x != 0) || (opl.y != 0) || (opl.z != 0)) |
            ((op2.x != 0) || (op2.y != 0) || (op2.z != 0)) )
            return new ThreeD(1,1,1);
        else return new ThreeD(0,0,0);
    }
    // Перегружаем оператор "&" для вычислений по сокращенной схеме
    public static ThreeD operator &(ThreeD opl, ThreeD op2)
    {
        if( ((opl.x != 0) && (opl.y != 0) && (opl.z != 0)) &
            ((op2.x != 0) && (op2.y != 0) && (op2.z != 0)))
            return new ThreeD(1,1,1);
        else return new ThreeD(0,0,0);
    }
    // Перегружаем оператор "!"
    public static bool operator !(ThreeD op)
    {
        if(op) return false;
            else return true;
    }
    // Перегружаем оператор true
    public static bool operator true(ThreeD op)
    {
        if((op.x != 0) || (op.y != 0) || (op.z != 0))
            return true; // Хотя бы одна координата не равна нулю
            else return false;
    }
    // Перегружаем оператор false
    public static bool operator false(ThreeD op)
    {
        if ((op.x == 0) && (op.y == 0) && (op.z == 0))
            return true; // Все координаты равны нулю
        else return false;
    }
    // Отображаем координаты X, Y, Z
    public void show()
    {
        Console.WriteLine("  " + x + ", " + y + ", " + z);
    }
}
class Program
{
    static void Main(string[] args)
    {
        ThreeD a = new ThreeD(5, 6, 7);
        ThreeD b = new ThreeD(10, 10, 10);
        ThreeD c = new ThreeD(0, 0, 0);
```

```
        Console.Write("\n  Координаты точки a: ");
        a. show();
        Console.Write("\n  Координаты точки b: ");
        b.show();
        Console.Write("\n  Координаты точки c: ");
        c.show();
        Console.WriteLine();
        if (a) Console.WriteLine("  a - ИСТИНА.");
        if(b) Console.WriteLine("  b - ИСТИНА.");
        if(c) Console.WriteLine("  c - ИСТИНА.");
        if(!a) Console.WriteLine("  a - ЛОЖЬ.");
        if(!b) Console.WriteLine("  b - ЛОЖЬ.");
        if (!c) Console.WriteLine("  c - ЛОЖЬ.");
        Console.WriteLine("  Используем операторы & и | ");
        if(a & b) Console.WriteLine("  a & b - ИСТИНА.");
           else Console.WriteLine("  a & b - ЛОЖЬ.");
        if (a & c) Console.WriteLine("  a & c - ИСТИНА.");
           else Console.WriteLine("  a & c - ЛОЖЬ.");
        if(a | b) Console.WriteLine("  a | b - ИСТИНА.");
           else Console.WriteLine("  a | b - ЛОЖЬ.");
        if (a | c) Console.WriteLine("  a | c - ИСТИНА.");
           else Console.WriteLine("  a | c - ЛОЖЬ.");
        // Теперь используем операторы && и || по сокращенной схеме вычислений
        Console.WriteLine("  Используем сокращенные операторы && и || ");
        if(a && b) Console.WriteLine("  a && b - ИСТИНА.");
           else Console.WriteLine("  a && b - ЛОЖЬ.");
        if(a && c) Console.WriteLine("  a && c - ИСТИНА.");
           else Console.WriteLine("  a && c - ЛОЖЬ.");
        if(a || b) Console.WriteLine("  a || b - ИСТИНА.");
           else Console.WriteLine("  a || b - ЛОЖЬ.");
        if (a || c) Console.WriteLine("  a || c - ИСТИНА.");
           else Console.WriteLine("  a || c - ЛОЖЬ.");
    }
}
```

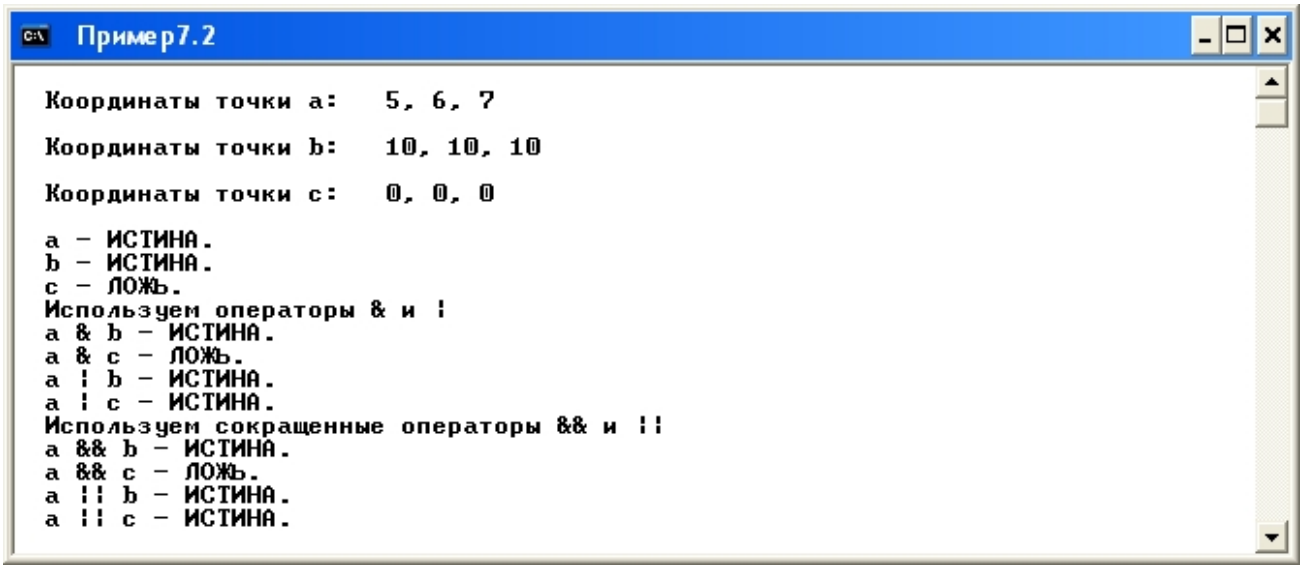

Рис.7. Результат выполнения приложения«Пример7.2»

Теперь остановимся подробнее на реализации операторов & и |. Для удобства приведем здесь их операторные методы.

```
// Перегружаем оператор "|" для вычислений по сокращенной схеме
public static ThreeD operator |(ThreeD op1, ThreeD op2)
{
   if( ((op1.x != 0) || (op1.y != 0) || (op1.z != 0)) |
      ((op2.x != 0) || (op2.y != 0) || (op2.z != 0)) )
      return new ThreeD(1,1,1);
   else return new ThreeD(0,0,0);
}
// Перегружаем оператор "&" для вычислений по сокращенной схеме
public static ThreeD operator &(ThreeD op1, ThreeD op2)
{
   if( ((op1.x != 0) && (op1.y != 0) && (op1.z != 0)) &
      ((op2.x != 0) && (op2.y != 0) && (op2.z != 0)))
      return new ThreeD(1,1,1);
   else return new ThreeD(0,0,0);
}
```

Оба операторных метода сейчас возвращают объект типа ThreeD. Если результатом операции оказывается значение ИСТИНА, то создается и возвращается истинный ThreeD-объект (т.е. объект, в котором не равна нулю, хотя бы одна координата). Если же результатом операции оказывается значение ЛОЖЬ, то создается и возвращается ложный ThreeD-объект (т.е. объект, в котором равны нулю все координаты). Таким образом, в инструкции:

```
if ( a & b) Console.WriteLine("a & b - ИСТИНА.");
  else Console.WriteLine("a & b - ЛОЖЬ.");
```

результатом операции a& b является ThreeD-объект, который в данном случае оказывается истинным. Поскольку в классе ThreeD определены операторы true и false, результирующий объект подвергается воздействию оператора true, вследствие чего возвращается результат типа bool. В данном случае результат равен значению true, и поэтому инструкция выводит сообщение " a & b - ИСТИНА. ".

Поскольку в этом примере программы все необходимые правила соблюдены, для объектов класса ThreeD теперь доступны логические операторы сокращенного действия. Их работа заключается в следующем. Первый операнд тестируется с помощью операторного метода operator true (для оператора "||") или операторного метода operator false (для оператора "&&"). Если этот тест в состоянии определить результат всего выражения, то оставшиеся &- или |-операции уже не выполняются. В противном случае для определения результата используется соответствующий перегруженный оператор "&" или " | " . Таким образом, использование &&- или || -операторов приводит к выполнению соответствующих &- или | -операций только в том случае, когда первый операнд не предопределяет результат всего выражения. Рассмотрим, например, следующую инструкцию из нашей программы:

```
if(a || c) Console.WriteLine("a || c - ИСТИНА.");
```

Сначала к объекту a применяется оператор true. Поскольку в данной ситуации a — истинный объект, в использовании операторного | -метода необходимости нет.

Но эту инструкцию можно переписать и по-другому:

```
if ( c || a) Console.WriteLine("c || a - ИСТИНА.");
```

В этом случае оператор true сначала будет применен к объекту c, а он в данной ситуации является ложным. Тогда будет вызван операторный| -метод, чтобы определить, является ли истинным объект a, что здесь как раз соответствует действительности.

2. Рабочее задание

Задание 1. Руководствуясь теоретическим материалом раздела 1 изучить возможности языка C# по созданию приложений, использующие перегрузку операторов класса, и выполнить примеры, описанные в этом разделе.

Задание 2. Разработать приложение «Работа с матрицами», с помощью которого можно продемонстрировать действия с матрицами: сложение матриц, вычитание матриц, получение противоположной матрицы, умножение матрицы на число, умножение матриц, транспонирование матрицы.

Примечание. Произведение матриц определяется следующим образом. Пусть заданы две матрицы *A* и *B*, причем число столбцов первой из них равно числу строк второй. Если

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nk} \end{pmatrix},$$

то произведением матриц A и B, называется матрица

$$C = \begin{pmatrix} c_{11} & \dots & c_{1k} \\ c_{21} & \dots & c_{2k} \\ \dots & \dots & \dots \\ c_{m1} & \dots & c_{mk} \end{pmatrix},$$

элементы которой вычисляются по формуле:

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{in}b_{nj}, \; i=1, \dots, m, \; j=1, \dots, k.$$

Матрица, получающаяся из прямоугольной матрицы A заменой строк столбцами, называется транспонированной по отношению к исходной матрице.

Задание 3. Разработать приложение «Пирамида в коробке», с помощью которого можно определить возможность размещения пирамиды в коробке. Размеры пирамиды (количество граней, длина, высота) и коробки (длина, ширина и высота) задаются с клавиатуры.

Задание 4. Разработать приложение «Упаковка», с помощью которого можно определить максимальное количество елочных шаров, размещенных в коробке. Размеры шаров и коробки (длина, ширина и высота) задаются с клавиатуры.

3. Контрольные вопросы

1. Что такое перегрузка операторов?
2. Перечислите формы методов operator и запишите их синтаксис.
3. Как осуществляется перегрузка бинарных операторов?
4. Как осуществляется перегрузка унарных операторов?
5. В чем особенности перегрузки операторов инкремента и декремента?

6. Как осуществляется перегрузка операторов отношения?
7. Как осуществляется перегрузка логических операторов?

Литература

1. Голощапов А.Л. Microsoft Visual Studio 2010. – СПб.:БХВ-Петербург, 2011. – 544 с.: ил.
2. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 1. Пер. с англ. - М.: «Русская Редакция», 2002.- 576 с.: ил.
3. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах. Том 2. Пер. с англ. - М.: «Русская Редакция», 2002.- 624 с.: ил.
4. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4.0. Пер. с англ. - М.: Издательский дом "Вильямс", 2011. — 1392 с.: ил.
5. Фленов М.Е. Библия C#. - СПб.: БХВ-Петербург, 2011.– 560с.: ил.
6. Шилдт Г. C# Учебный курс. – СПб.: Питер, Издательская группа BHV, 2003. – 512 с.: ил.