Структуры

Определение структуры

Структура — тип данных, аналогичный классу, отличия:

- является *значимым*, а не ссылочным типом данных;
- не может участвовать в иерархиях наследования, может только реализовывать интерфейсы;
- в структуре запрещено определять конструктор по умолчанию, поскольку он определен неявно и присваивает всем ее элементам нули соответствующего типа;
- в структуре запрещено определять деструкторы, поскольку это бессмысленно.

Область применения структур: типы данных, имеющие небольшое количество полей, с которыми удобнее работать как со значениями, а не как со ссылками (снижаются накладные расходы на динамическое выделение памяти)

Синтаксис структуры

[атрибуты] [спецификаторы] **struct** имя_структуры [: интерфейсы]

тело_структуры [;]

- *Спецификаторы* доступа - public, internal и private (последний — только для вложенных структур).
- *Интерфейсы*, реализуемые структурой, перечисляются через запятую.
- *Тело структуры* может состоять из констант, полей, методов, свойств, событий, индексаторов, операций, конструкторов и вложенных типов.

Пример структуры

```
struct Complex
    {  public double re, im;
       public Complex( double re_, double im_ )
       {     re = re_; im = im_;  }
       public static Complex operator + ( Complex a, Complex b )
       {    return new Complex( a.re + b.re, a.im + b.im );        }
       public override string ToString()
       { return ( string.Format( "({0,2:0.##}; {1,2:0.##})", re, im ) );
       }
    }
class Class1
    {   static void Main()
       { Complex a = new Complex( 1.2345, 5.6 );
          Console.WriteLine( "a = " + a );
          Complex [] mas = new Complex[4]; …
   }}
```

Результат работы программы:
a = (1,23; 5,6)

Описание элементов структур

- поскольку структуры не могут участвовать в иерархиях, для их элементов не могут использоваться спецификаторы protected и protected internal;
- структуры не могут быть абстрактными (abstract), к тому же по умолчанию они бесплодны (sealed);
- методы структур не могут быть абстрактными и виртуальными;
- переопределяться (то есть описываться со спецификатором override) могут только методы, унаследованные от базового класса object;
- параметр this интерпретируется как значение, поэтому его можно использовать для ссылок, но не для присваивания;
- при описании структуры нельзя задавать значения полей по умолчанию.

Перечисления

Определение перечисления

Перечисление – набор связанных констант:

```
enum Menu { Read, Write, Append, Exit };

enum Радуга { Красный, Оранжевый, Желтый, Зеленый, Синий,
    Фиолетовый };

enum Nums { two = 2, three, four, ten = 10, eleven, fifty = ten + 40 };

enum Flags : byte

{    b0, b1, b2, b3 = 0x04, b4 = 0x08, b5 = 0x10, b6 = 0x20, b7 = 0x40    };
```

- Имена перечисляемых констант внутри каждого перечисления должны быть уникальными, а значения могут совпадать.
- Все перечисления являются потомками базового класса System.Enum

Преимущества перечислений перед описанием именованных констант:

- связанные константы нагляднее;
- компилятор выполняет проверку типов;
- интегрированная среда разработки подсказывает возможные значения констант.

С переменными перечисляемого типа можно выполнять:

- арифметические операции (+, –, ++, ––),
- логические поразрядные операции (^, &, |, ~),
- сравнения (<, <=, >, >=, ==, !=)
- получать размер в байтах (sizeof).

- При использовании переменных перечисляемого типа в целочисленных выражениях и операциях присваивания требуется явное *преобразование типа*.
- Переменной перечисляемого типа можно присвоить любое значение, представимое с помощью базового типа.

```
enum Flags : byte
{  b0, b1, b2, b3 = 0x04, b4 = 0x08, b5 = 0x10, b6 = 0x20, b7 = 0x40     };
Flags a = Flags.b2 | Flags.b4;
++a;
int x = (int) a;
Flags b = (Flags) 65;
```