Структуры и перечисления

Лекция №9

Структуры

Структура – это значимый тип данных, экземпляр структуры размещается в стеке, а не в динамической памяти.

Синтаксис структуры:

[спецификаторы] struct <имя> [: интерфейсы]

{тело структуры}

Спецификаторы – public, internal, для вложенных структур можно private.

Тело может содержать константы, поля, методы, свойства, события, индексаторы, операции, конструкторы. Их описание и использование аналогично соответствующим элементам класса.

Правила описания структур:

- Структуры не могут участвовать в иерархиях наследования, но может реализовывать интерфейсы (однако, структуры наследуют класс object).

В структуре нельзя определять конструктор без параметров. Конструктор по умолчанию автоматически определяется для всех структур, и его изменить нельзя.

- В структуре нельзя определять деструкторы.

- Структуры не могут быть абстрактными.
- Структура не может содержать абстрактные или виртуальные методы.

- Переопределяться со спецификатором override могут только методы, унаследованные от класса object.

• Нельзя задавать значения полей по умолчанию.

Например,

```
struct   Student
    {   public string fam;
        public string dat_r;
        public int stip;
        public Student(string f, óuring dr,int óõ(
        { fam = f; dat_r = dr; stip = st; }
}
```

Объект структуры можно создать с помощью операции **new**, подобно любому объекту класса:

<имя структуры> <имя экземпляра> = new <имя структ.>( );

В этом случае будет вызван конструктор по умолчанию, который инициализирует все поля нулями.

<имя структуры> <имя экземпляра> =

new <имя структ.>(<параметры>);

В этом случае будет вызван конструктор определенный в структуре.

Например,
```
Student St1 = new Student( );
Student St2 = new Student("Иванов","12.03.87",200);
```

Можно объявить структуру, не используя new:

<имя структуры> <имя экземпляра>;

В этом случае придется выполнить инициализацию вручную.

Например, Student St3; St3.fam = "Петров";

При присваивании одной структуры другой создается копия этого объекта.

Пример из Шилдта:

```
struct MyStruct {  public int x; }

 public static void Main( ) {
 MyStruct a;
 MyStruct b;

  a.x = 10;
  b.x = 20;
  Console .WriteLine ("a.x {0}, b.x {1} ", a.x, b.x);

  a =  b;
  b.x = 30;
  Console.WriteLine("a.x {0}, b.x {1}", a.x, b.x);}
```

Результаты:

a .x 10, b.x 20

a .x 20, b.x 30

Если бы a и b были объектами класса , результат был бы следующим:

a . x 10, b.x 20

a . x 30, b.x 30

Преимущества использования структур:

Структуры обрабатываются напрямую, а не через ссылки. Таким образом, структура не требует отдельной ссылочной переменной. Т.е. при использовании структур расходуется меньший объем памяти.

Благодаря прямому доступу к структурам, при работе с ними не снижается производительность, что имеет место при доступе к объектам классов.

Итак, если нужно хранить небольшую группу связанных данных, но не нужно обеспечивать наследование и использовать другие достоинства ссылочных типов, предпочтительно использовать структуру.

Особенно эффективным может быть использование массивов структур вместо массивов классов. Ведь для массива из 100 экземпляров класса создается 101 объект, а для массива структур – 1 объект.

Все значимые типы в C# являются структурами.

В пространстве имен **System** определены такие структуры:

Boolean

DateTime

Int64

UInt64

Byte

Decimal

Intl6

Single

UIntl6

Char

Double

Int32

TimeSpan

UInt32

и др.

Перечисления

Перечисление **(enumeration)** — это множество именованных целочисленных констант.

Синтаксис перечисления:

[спецификаторы] enum <имя> [: базовый тип]

{тело перечисления}

Допускаются спецификаторы new, public, protected, internal, private.

Базовый тип – это тип элементов, из которых построено перечисление. По умолчанию **int**.

Тело перечисления состоит из имен констант, которым может быть присвоено значение, разделенных запятыми.

Если значение не указано, оно вычисляется прибавлением единицы к предыдущей константе.

По умолчанию константам присваиваются последовательные значения начиная с нуля.

Например,

```
public enum god {Январь, Февраль, Март, Апрель, Май, Июнь,
                 Июль, Август, Сентябрь, Октябрь, Ноябрь, Декабрь }
```

Описывать перечисление можно как в пространстве имен, так и внутри класса или структуры.

Например, дополним структуру студент:

```
double[ ] x;  // поле с оценками за сессию

public enum Экзамен
        { Информатика, Математика, Физика, История};

public double this[Экзамен   i]
    { get   { return   x[(int) i]; }     set { x[(int) i] = value; } }
```

Тогда допустимы следующие операторы:

```
Student St2 = new Student("Иванов","12.03.87",200);

St2[Student.Экзамен.Математика] = 9;
St2[Student.Экзамен.Физика] = 10;

Console.WriteLine(Student.Экзамен.Физика + "....." +
                  St2[Student.Экзамен.Физика]);
```

При использовании переменных перечисляемого типа в целочисленных операциях и выражениях требуется явное преобразование типа.

С переменными перечисляемого типа можно выполнять арифметические операции, логические поразрядные операции, сравнивать их с помощью операций отношения.

Например,

```
for (god   g = god.Март;  g < god.Сентябрь;  g++)
        Console.WriteLine(g);
```

Результат:

Март

Апрель

Май

Июнь

Июль

Август

Все перечисления являются потомками базового класса **System.Enum.**

Приведем описание некоторых методов этого класса:

Статический метод

GetName(Type t, object v)

возвращает строку - имя константы по ее значению (t – тип перечисления, v- значение).

Например, `Enum.GetName(typeof(god), 5)`

Результат: Июнь

Статический метод

GetNames(Type t)

возвращает строковый массив из имен констант, составляющих перечисление.

Например, пусть имеется перечисление

```
public enum KodTovara
    { Стол =104,  Стул=203,  Шкаф,  Диван=378 };
```

Тогда результатом оператора

```
string[ ]   nt  =  Enum.GetNames(typeof(KodTovara));
```

будет массив строк **nt** из четырех элементов:

Стол Стул Шкаф Диван

Статический метод

GetValues(Type t)

возвращает массив значений констант, составляющих перечисление. Результат имеет тип **Array**.

Например, выполнение оператора

```
Array   kt =  Enum.GetValues(typeof(KodTovara));
```

или оператора

```
int[ ]   kt =(int[ ]) Enum.GetValues(typeof(KodTovara));
```

приведет к формированию массива kt из четырех элементов:

104 203 204 378

В первом операторе формируется массив с типом элементов **KodTovara.**

Статический метод

IsDefined(Type t, object v)

возвращает значение **true**, если параметр **v** содержит

- значение константы, входящей в перечисление,

или

- символическое имя константы, входящей в перечисление

и **false** в противном случае.

Например,

Enum.IsDefined(typeof(KodTovara),"Стол") **true**

Enum.IsDefined(typeof(KodTovara),104) **true**

Enum.IsDefined(typeof(KodTovara),"Трюмо") **false**

Статический метод

Enum.Parse(Type t, string s)

конвертирует строку, представляющую имя константы из перечисления типа t или значение константы, в соответствующий объект перечисления.

Например,

```
KodTovara tovar =
  (KodTovara) Enum.Parse(typeof(KodTovara), "104");
Console.WriteLine(tovar);
```

или

```
KodTovara tovar=
(KodTovara) Enum.Parse(typeof(KodTovara), "Стол");
```

Можно так:

KodTovara tovar1 = (KodTovara) 104;

Переменной перечисляемого типа можно присвоить не только одно из значений из перечисления, а любое значение, представимое с помощью базового типа.

Например,

KodTovara tovar1 = (KodTovara) 100;

Оператор

Console.WriteLine(Enum.IsDefined(typeof(KodTovara),tovar1));

выведет **false**

а оператор

Console.WriteLine(Enum.GetName(typeof(KodTovara), 100));

выведет пустую строку.

В пространстве имен **System** определены стандартные перечисления **ConsoleColor и ConsoleKey.**

ConsoleColor содержит константы, определяющие цвет выводимых символов и цвет фона.

Например, **ConsoleColor.Gray .**

Используется для установки цвета фона:

Console.BackgroundColor = ConsoleColor.Red;

или цвета выводимых символов:

Console.ForegroundColor = ConsoleColor.White;

Перечисление **ConsoleKey** содержит константы для определения стандартных клавиш.

Например: **ConsoleKey.Enter, ConsoleKey.PageUp**

Используется для определения, какая была нажата клавиша.

Метод **Console.ReadKey( )** возвращает результат типа **ConsoleKeyInfo.** Это структура, содержащая информацию о нажатой клавише.

Свойство **Key** этой структуры имеет тип **ConsoleKey** и содержит константу, определяющую нажатую клавишу.

Следующий пример показывает, как можно проанализировать, была ли нажата клавиша **Home.**

```
ConsoleKeyInfo  k = Console.ReadKey( );
 if  (k.Key  = = ConsoleKey.Home)
        Console.WriteLine(" Нажата клавиша Home ");
 else
  Console.WriteLine(" Нажата другая клавиша ");
```

Свойство **KeyChar** возвращает **Unicode-**символ, представляющий текущий объект типа **ConsoleKeyInfo.**

```
ConsoleKeyInfo   k1 =   Console.ReadKey(true);
      if   (k1.KeyChar == 'A')
  Console.WriteLine("нажата клавиша с буквой А");
```

Пример.

Подготовить текстовый файл, содержащий информацию о студентах: фамилия и инициалы, факультет, дата рождения, средний балл (разделителем в файле служит точка с запятой).

Разработать программу, которая выполняет следующие действия:

- Считывает информацию в массив структур.

(Поле с факультетом должно быть перечисляемого типа: в программе следует создать перечисление с факультетами)

- Выводит информацию о студентах заданного факультета.

Выбор факультета должен осуществляться посредством меню (Все названия факультетов из перечисления выводятся на экран, и пользователь получает возможность выбора, указывая порядковый номер, при этом выбираемый факультет подсвечивается красным цветом, выбор заканчивается после нажатия клавиши **Enter**).

- Записывает в новый текстовый файл информацию о студентах в виде:

Факультет: ЭФ

№	Фамилия	Средний балл	Возраст
1	Иванов	9	15
2	Петров	5	18

Факультет: ФАИС

№	Фамилия	Средний балл	Возраст
1	Сидоров	6.5	19
2	Ребров	5	20

Если информации по какому-то факультету нет, в файл записывается заголовок с факультетом и вместо таблицы фраза «данных нет».

Информация в каждой таблице должна быть отсортирована по возрастанию возраста.

Добавляем в программу инструкцию **using System.IO;**

```
public enum Facultet
  { ГЭФ, ЭФ, ФАИС, МТФ, МСФ };
```

```
 struct Student : IComparable
     {     public string       fam;
           public Facultet   fcltt;
           public string       dat_r;
           public double      sr_ball;

public double vozrast
 { get
 { return (DateTime.Now.Year - Convert.ToDateTime(dat_r).Year);}
}
```

```
public int CompareTo(Object   obj)
   {  Student   st  =  ( Student)   obj  ;
       if (vozrast > st.vozrast)   return 1;
      else    { if  (vozrast < st.vozrast)   return -1;    else   return 0; }
   }
   }

class Program
    {
        static   void   Main(string[ ] args)
                {
```

```
StreamReader    f = new   StreamReader("baza.dat");
        string s = f.ReadLine( ); int j = 0;
        while (s != null)
        {
            s = f.ReadLine( );
            j++;
        }
f.Close( );

 Student[ ]    students = new Student[j];
```

```
string[ ] dano=new string[4];

 f = new StreamReader("baza.dat");

  s = f.ReadLine( ); j = 0;

   while (s != null)

              {

                 dano = s.Split(';');

                   students[j].fam=dano[0];

students[j].fcltt =(Facultet) Enum.Parse(typeof(Facultet), dano[1]);
```

```
students[j].dat_r = dano[2];
  students[j].sr_ball = Convert.ToDouble(dano[3]);
   s = f.ReadLine( );
    j++;
            }
   f.Close( );
    string[ ]    F = Enum.GetNames(typeof(Facultet));
     int v =  -1, p=0;
     Console.Title =  "Студенты";
      Console.Clear( );
```

```
for (int  i = 0; i < F.Length; i++)   Console.WriteLine(F[i]);

 bool       ff = true;
  while (ff)
     {Console.ForegroundColor = ConsoleColor.Gray;

        if (  !(v == 0 && p == 0))
                        {
                                Console.SetCursorPosition(0, p);

                                 Console.WriteLine(F[p]);
                          }
```

```
Console.SetCursorPosition(0, F.Length);
 ConsoleKeyInfo k = Console.ReadKey();

  if (v = =  -1)   p = 0;   else     p = v;

    if (k.Key = = ConsoleKey.Enter)    ff = false;

else
   {if    (Char.GetNumericValue(k.KeyChar) >= 0   &&
                Char.GetNumericValue(k.KeyChar) < F.Length)
{
    v = Convert.ToInt32(Char.GetNumericValue(k.KeyChar));
```

```
  Console.SetCursorPosition(0, v);
  Console.ForegroundColor = ConsoleColor.Red;
  Console.WriteLine(F[v]);

}
    else    v =  -1;
 }
}
      if (v != -1)
                {
                    Console.Clear();
                    Console.WriteLine(" Факультет: " + F[v]);
```

```
// шапка таблицы…..
    j=0;
    for  (int i = 0; i < students.Length; i++)
        {

        if (Enum.GetName(typeof(Facultet),  students[i].fcltt) == F[v])
            {   j++;
              Console.WriteLine("│{0,2} │ {1,15} │ {2,15} │ {3,11} │",
                        j, students[i].fam, students[i].dat_r, students[i].sr_ball);
              }
        }
   }
```

```
else
            Console.WriteLine(" Факультет не выбран ");

   Array.Sort(students);
Console.ReadKey( );
StreamWriter     f1 = new      StreamWriter("baza1.txt");
  for (int   ii = 0;  ii < F.Length;  ii++)
    {
        f1.WriteLine(" Факультет: " + F[ii]);

         int q = 0;
```

```
for (int i = 0; i < students.Length; i++)
  {
     if (students[i].fcltt.ToString( ) = =  F[ii])
        {
          if (q = = 0)
              {// шапка таблицы…..
               }

         q++;
        f1.WriteLine("│{0,2} │ {1,15} │ {2,15} │ {3,11} │",
         q, students[i].fam, students[i].sr_ball, students[i].vozrast);
           }
  }
```

```
if (q = = 0)
      f1.WriteLine("данных нет.");

 f1.Close( );
 }
 }
```