Структуры. Перечисления

Перечисления

Перечисления - множество именованных целочисленных констант, которые образуют тип данных.

Общая форма объявления типа перечисление имеет вид:

enum имя { список_констант };

где

имя – это имя (название) типа перечисления;
список_констант – список идентификаторов, разделенных запятой.

Чтобы изменить базовый тип констант с int на другой, нужно указать этот тип сразу после имени перечисления**. Общая форма такого объявления будет иметь вид:**

enum имя : новый_тип { список_констант };

где

имя – это имя (название) типа перечисления;
новый_тип – собственно, новый тип, который устанавливается вместо базового типа;
список_констант – список идентификаторов, разделенных запятой.

```
enum Days
{
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday,
    Sunday
}
```

```
enum Time : byte
{
    Morning,
    Afternoon,
    Evening,
    Night
}
```

Инициализация перечисления

```
enum Operation
{
    Add = 1,   // каждый следующий элемент по умолчанию увеличивается на единицу
    Subtract, // этот элемент равен 2
    Multiply, // равен 3
    Divide    // равен 4
}
```

```
enum Operation
{
    Add = 2,
    Subtract = 4,
    Multiply = 8,
    Divide = 16
}
```

Каждое перечисление фактически определяет новый тип данных.

```
class Program
{
    enum Operation
    {
        Add = 1,
        Subtract,
        Multiply,
        Divide
    }
    static void Main(string[] args)
    {
        Operation op;
        op = Operation.Add;
        Console.WriteLine(op); // Add

        Console.ReadLine();
    }
}
```

```
Operation op;
op = Operation.Multiply;
Console.WriteLine((int)op); // 3
```

```
enum Operation
{
    Add = 1,
    Subtract,
    Multiply,
    Divide
}
class Program
{
    static void Main(string[] args)
    {
        Operation op;
        op = Operation.Add;
        Console.WriteLine(op); // Add

        Console.ReadLine();
    }
}
```

```
class Program
{
    enum Operation
    {
        Add = 1,
        Subtract,
        Multiply,
        Divide
    }

    static void MathOp(double x, double y, Operation op)
    {
        double result = 0.0;

        switch (op)
        {
            case Operation.Add:
                result = x + y;
                break;
            case Operation.Subtract:
                result = x - y;
                break;
            case Operation.Multiply:
                result = x * y;
                break;
            case Operation.Divide:
                result = x / y;
                break;
        }
```

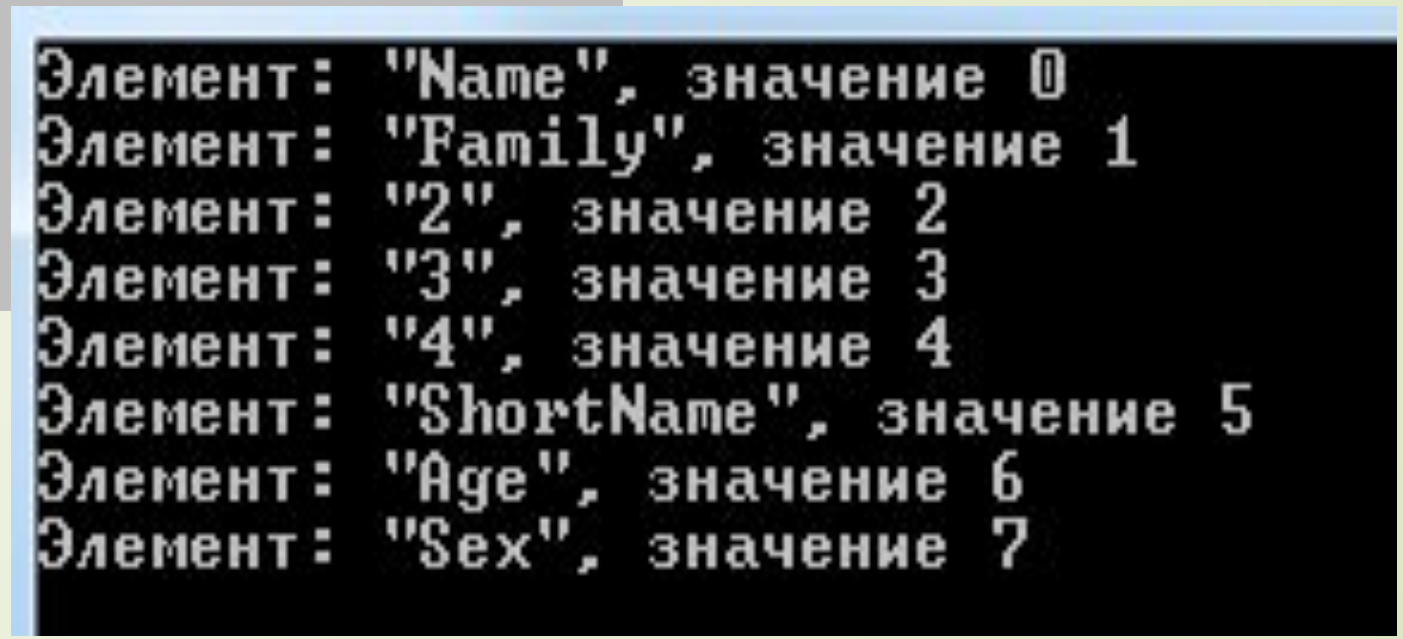
```
static void Main(string[] args)
    {
        // Тип операции задаем с помощью константы
Operation.Add, которая равна 1
        MathOp(10, 5, Operation.Add);
        // Тип операции задаем с помощью константы
Operation.Multiply, которая равна 3
        MathOp(11, 5, Operation.Multiply);

        Console.ReadLine();
    }
}
```

```
// Создать перечисление
    enum UI : long { Name, Family, ShortName = 5, Age, Sex }

    class Program
    {
        static void Main()
        {
            UI user1;
            for (user1 = UI.Name; user1 <= UI.Sex; user1++)
                Console.WriteLine("Элемент: \"{0}\", значение
{1}",user1,(int)user1);

            Console.ReadLine();
        }
    }
}
```



```
enum apple { Jonathan, GoldenDel, RedDel, Winsap, Cortland,
Mclntosh };

public static void Main()
{
    string[] color = { "красный", "желтый", "красный", "красный",
"красный", "красно-зеленый" };
        apple i; // Объявляем переменную перечислимого типа
        // Используем переменную i для обхода всех членов
перечисления
        for (i = apple.Jonathan; i <= apple.Mclntosh; i++)
            Console.WriteLine(i + " имеет значение " + (int)i);
        Console.WriteLine();
        // Используем перечисление для индексации массива
        for (i = apple.Jonathan; i <= apple.Mclntosh; i++)
                Console.WriteLine("Цвет сорта " + i + " - " +
color[(int)i]);
}
```

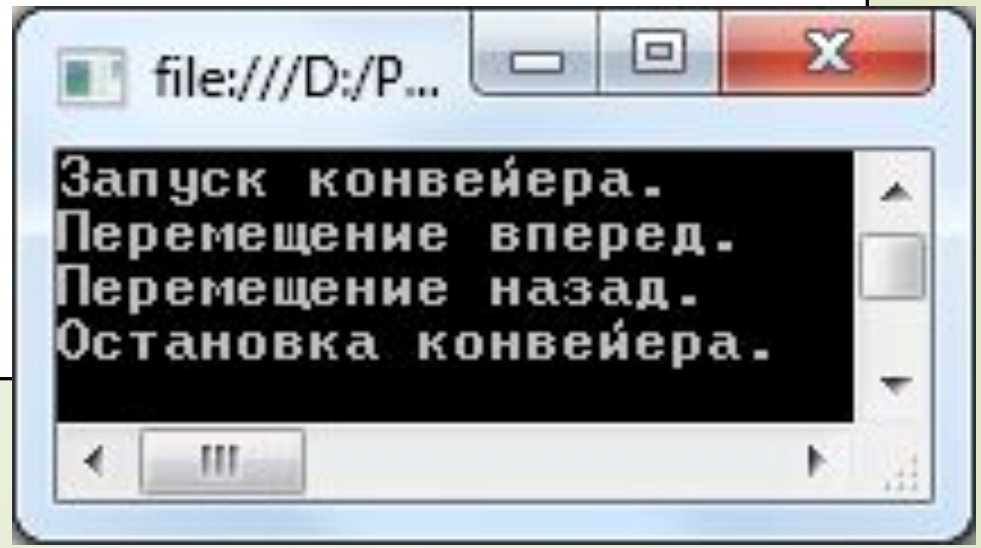
```
Jonathan имеет значение 0
GoldenDel имеет значение 1
RedDel имеет значение 2
Winsap имеет значение 3
Cortland имеет значение 4
Mclntosh имеет значение 5

Цвет сорта Jonathan - красный
Цвет сорта GoldenDel - желтый
Цвет сорта RedDel - красный
Цвет сорта Winsap - красный
Цвет сорта Cortland - красный
Цвет сорта Mclntosh - красно-зеленый
```

```
enum action { старт, стоп, вперед, назад };

static void conveyor(action com)
{
      switch (com)
            {
      case action.старт: Console.WriteLine("Запуск конвейера. "); break;
      case action.стоп: Console.WriteLine("Остановка конвейера."); break;
      case action.вперед: Console.WriteLine("Перемещение вперед."); break;
      case action.назад: Console.WriteLine("Перемещение назад."); break;
            }
}

        public static void Main()
        {
            conveyor(action.старт);
            conveyor(action.вперед);
            conveyor(action.назад);
            conveyor(action.стоп);
            Console.ReadLine();
        }
```



Структуры

В языке программирования C#, с целью удобного группирования данных, используются так называемые структуры.

Использование структуры в программе происходит в 2 этапа:

- объявление типа структуры;
- объявление структурной переменной.

Структуры объявляются с использованием ключевого слова **struct**

Общая форма объявления типа структуры:

```
struct имя_типа_структуры : интерфейсы
{
    // объявление членов и методов структуры
    // ...
}
```

где

имя_типа_структуры – название структурного типа на основе которого будут объявляться объекты (переменные, экземпляры структуры);

интерфейсы – список интерфейсов, методы которых нужно реализовать в теле структуры.

Структуры **не могут наследовать (либо наследоваться)** другие структуры или классы. Структура может реализовать один или несколько интерфейсов. Они указываются после имени структуры и отделяются запятыми. Как и у классов, членами структур могут быть методы, поля, индексаторы, свойства, операторные методы и события. Структуры могут также определять конструкторы, но не деструкторы. Однако для структуры нельзя определить конструктор по умолчанию (без параметров). Дело в том, что конструктор по умолчанию автоматически определяется для всех структур, и его изменить нельзя.

Объект структуры можно создать с помощью оператора new, подобно любому объекту класса, но это не обязательно. Если использовать оператор new, вызывается указанный конструктор, а если не использовать его, объект все равно будет создан, но не инициализирован.

По умолчанию, поля структурной переменной имеют тип доступа private. Чтобы можно было иметь доступ к полям структурной переменной непосредственно, используется ключевое слово public.

```
struct Student
{
    public string name; // фамилия
    public string surname; // имя
    public int year; // год вступления в учебное
заведение
    public int birth_year; // год рождения
    public string address; // адрес проживания
    public string book_number; // номер
зачетной книжки
    public float rank; // рейтинг
}
```

Пусть дано объявление структурной переменной, описывающей одну запись в телефонном справочнике:

```
struct Telephone
{
    public string number; // номер телефона
    public string name; // имя абонента
    public string surname; // фамилия абонента
    public string address; // адрес
    public int code; // почтовый код
}
```

Объявление и использование структурной переменной типа Telephone будет следующим:

```
// Объявление структурной переменной с именем T1 типа Telephone
Telephone T1;

// заполнение полей структурной переменной T1
T1.name = "Johnson";
T1.surname = "John";
T1.number = "77777777";
T1.code = 89300;
T1.address = "Boston";
```

```
using System;

namespace Structures
{
    class Program
    {
        static void Main(string[] args)
        {
            Book book;
            book.name = "Война и мир";
            book.author = "Л. Н. Толстой";
            book.year = 1869;

            //Выведем информацию о книге book на экран
            book.Info();

            Console.ReadLine();
        }
    }

    struct Book
    {
        public string name;
        public string author;
        public int year;

        public void Info()
        {
            Console.WriteLine("Книга '{0}' (автор {1}) была
издана в {2} году", name, author, year);
```

Структуру можно задать как внутри пространства имен (как в данном случае), так и внутри класса, но не внутри метода.

Массив структур:

```
Book[] books=new Book[3];
books[0].name = "Война и мир";
books[0].author = "Л. Н. Толстой";
books[0].year = 1869;

books[1].name = "Преступление и наказание";
books[1].author = "Ф. М. Достоевский";
books[1].year = 1866;

books[2].name = "Отцы и дети";
books[2].author = "И. С. Тургенев";
books[2].year = 1862;

foreach (Book b in books)
{
    b.Info();
}
```

Конструкторы в структурах

Кроме обычных методов структура может содержать специальный метод - конструктор, который выполняет некую начальную инициализацию объекта, например, присваивает всем полям некоторые значения по умолчанию. В принципе для структуры необязательно использовать конструктор:

```
Book book1;
```

Однако в этом случае, чтобы использовать структуру, нам надо будет первоначально проинициализировать все ее поля:

```
book1.name = "Война и мир";
book1.author = "Л. Н. Толстой";
book1.year = 1869;
```

Вызов же конструктора по умолчанию позволяет автоматически проинициализировать поля структуры значениями по умолчанию (например, для числовых данных - это число 0):

Book book2=new Book(); // использование конструктора

Вызов конструктора имеет следующий синтаксис:

new название_структуры ([список_параметров]).

```
struct Book
{
    public string name;
    public string author;
    public int year;

    // конструкор
    public Book(string n, string a, int y)
    {
        name = n;
        author = a;
        year = y;
    }
    public void Info()
    {
        Console.WriteLine("Книга '{0}' (автор {1}) была издана
в {2} году", name, author, year);
    }
}
```

Конструктор по сути является обычным методом, только не имеет возвращаемого значения, и его название всегда совпадает с именем структуры или класса. Теперь используем его:

```
Book book=new Book("Война и мир", "Л. Н. Толстой", 1869);
book.Info();
```

Теперь нам не надо вручную присваивать значения полям структуры - их инициализацию выполнил конструктор.