*Перечисление* представляет собой множество именованных целочисленных констант.

Перечислимый тип данных объявляется с помощью ключевого слова enum.

# Перечисления

Ниже приведена

общая форма объявления перечисления:

***enum имя {список\_перечисления};***

где *имя* — это имя типа перечисления, а *список\_перечисления* — список идентификаторов,

разделяемый запятыми.

В приведенном ниже примере объявляется перечисление Apple различных сортов

яблок.

enum Apple

{

Jonathan, GoldenDel, RedDel, Winesap,

Cortland, McIntosh

};

Следует особо подчеркнуть, что каждая символически обозначаемая константа

в перечислении имеет целое значение. Тем не менее неявные преобразования перечислимого

типа во встроенные целочисленные типы и обратно в C# не определены,

а значит, в подобных случаях требуется явное приведение типов. Кроме того, приведение

типов требуется при преобразовании двух перечислимых типов. Но поскольку

перечисления обозначают целые значения, то их можно, например, использовать для

управления оператором выбора switch или же оператором цикла for.

По умолчанию значение первой символически обозначаемой константы в перечислении

равно нулю. Следовательно, в приведенном выше примере перечисления Apple

константа Jonathan равна нулю, константа GoldenDel — 1, константа RedDel — 2

и т.д.

Доступ к членам перечисления осуществляется по имени их типа, после которого

следует оператор-точка. Например, при выполнении фрагмента кода

Console.WriteLine(Apple.RedDel + " имеет значение " +

(int) Apple.RedDel);

выводится следующий результат.

RedDel имеет значение 2

Как показывает результат выполнения приведенного выше фрагмента кода, для вывода

перечислимого значения используется его имя. Но для получения этого значения

требуется предварительно привести его к типу int.

(***glava12\_9***)

class EnumDemo

{

enum Apple { Johnatan, GoldenDel, RedDel, Winesap, Cortland, McIntosh };

static void Main()

{

string[] color =

{

"red",

"yellow",

"red",

"red",

"red",

"redish - green"

};

Apple i; //variable of enum type

//use var i for cycle access to enum members

for (i = Apple.Johnatan; i <= Apple.McIntosh; i++)

Console.WriteLine(i + " has value " + (int)i);

Console.WriteLine();

//use enum to index array

for (i = Apple.Johnatan; i <= Apple.McIntosh; i++)

Console.WriteLine("Sort color " + i

+ " - " + color[(int)i]);

}

}

Обратите внимание на то, как переменная типа Apple управляет циклами for.

Значения символически обозначаемых констант в перечислении Apple начинаются с

нуля, поэтому их можно использовать для индексирования массива, чтобы получить

цвет каждого сорта яблок. Обратите также внимание ***на необходимость производить***

***приведение типов, когда перечислимое значение используется для индексирования***

***массива***. Как упоминалось выше, в C# не предусмотрены неявные преобразования

перечислимых типов в целочисленные и обратно, поэтому для этой цели требуется

явное приведение типов.

И еще одно замечание: все перечисления неявно наследуют от класса System.Enum,

который наследует от класса System.ValueType, а тот, в свою очередь, — от класса

object.

# Инициализация перечисления

Значение одной или нескольких символически обозначаемых констант в перечислении

можно задать с помощью инициализатора. Для этого достаточно указать после

символического обозначения отдельной константы знак равенства и целое значение.

Каждой последующей константе присваивается значение, которое на единицу больше

значения предыдущей инициализированной константы. Например, в приведенном

ниже фрагменте кода константе RedDel присваивается значение 10.

enum Apple

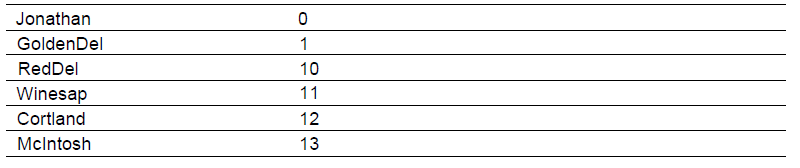
{

Jonathan, GoldenDel, RedDel = 10, Winesap,

Cortland, McIntosh

};

В итоге все константы в перечислении принимают приведенные ниже значения.



# Указание базового типа перечисления

По умолчанию в качестве базового для перечислений выбирается тип int, тем не

менее перечисление может быть создано любого целочисленного типа, за исключением

char. Для того чтобы указать другой тип, кроме int, достаточно поместить этот

тип после имени перечисления, отделив его двоеточием. В качестве примера ниже задается

тип byte для перечисления Apple.

enum Apple : byte

{

Jonathan, GoldenDel, RedDel,

Winesap, Cortland, McIntosh

};

Теперь константа Apple.Winesap, например, имеет количественное значение типа

byte.

# Применение перечислений

На первый взгляд перечисления могут показаться любопытным, но не очень нужным

элементом С#, но на самом деле это не так. Перечисления очень полезны, когда

в программе требуется одна или несколько специальных символически обозначаемых

констант. Допустим, что требуется написать программу для управления лентой конвейера

на фабрике. Для этой цели можно создать метод Conveyor(), принимающий в

качестве параметров следующие команды: *"*старт*"*, *"*стоп*"*, *"*вперед*"* и *"*назад*"*. Вместо

того чтобы передавать методу Conveyor() целые значения, например, 1 — в качестве

команды *"*старт*"*, 2 — в качестве команды *"*стоп*"* и так далее, что чревато ошибками,

можно создать перечисление, чтобы присвоить этим значениям содержательные символические

обозначения. Ниже приведен пример применения такого подхода.

(***glava12\_10***)

class ConveyorControl

{

//enum conveyor commands

public enum Action { Start, Stop, Forward, Reverse };

public void Conveyor(Action com)

{

switch(com)

{

case Action.Start:

Console.WriteLine("Start conveyor.");

break;

case Action.Stop:

Console.WriteLine("Stop conveyor.");

break;

case Action.Forward:

Console.WriteLine("Move conveyor forward.");

break;

case Action.Reverse:

Console.WriteLine("Move conveyor backward.");

break;

}

}

}

class ConvetorDemo

{

static void Main()

{

ConveyorControl c = new ConveyorControl();

c.Conveyor(ConveyorControl.Action.Start);

c.Conveyor(ConveyorControl.Action.Forward);

c.Conveyor(ConveyorControl.Action.Reverse);

c.Conveyor(ConveyorControl.Action.Stop);

}

}

Метод Conveyor() принимает аргумент типа Action, и поэтому ему могут быть

переданы только значения, определяемые в перечислении Action. Например, ниже

приведена попытка передать методу Conveyor() значение 22.

c.Conveyor(22); // Error!

Эта строка кода не будет скомпилирована, поскольку отсутствует предварительно

заданное преобразование типа int в перечислимый тип Action. Именно это и

препятствует передаче неправильных команд методу Conveyor(). Конечно, такое

преобразование можно организовать принудительно с помощью приведения типов,

но это было бы преднамеренным, а не случайным или неумышленным действием.

Кроме того, вероятность неумышленной передачи пользователем неправильных команд

методу Conveyor() сводится с минимуму благодаря тому, что эти команды обозначены

символическими именами в перечислении.

В приведенном выше примере обращает на себя внимание еще одно интересное

обстоятельство: перечислимый тип используется для управления оператором switch.

Как упоминалось выше, перечисления относятся к целочисленным типам данных,

и поэтому их вполне допустимо использовать в операторе switch.