**Урок № 6. C# Теория**

* **Перечисление** — это структура, которая позволяет создать переменную с фиксированным набором возможных значений. Наиболее распространенным примером является использование перечисления для определения дня недели. Есть только семь возможных значений для дней недели, и вы можете быть достаточно уверены, что эти значения никогда не изменятся.
* Необходимо определять перечисления непосредственно в пространстве имен, чтобы все классы в этом пространстве имен имели к нему доступ, если это необходимо.
* По умолчанию значения перечисления начинаются с 0, и каждый последующий элемент увеличивается на значение **1**.

**Создание и использование**

* Чтобы создать перечисление, вы объявляете его в файле кода со следующим синтаксисом, который демонстрирует создание перечисления с именем Day, содержащего дни недели:

|  |
| --- |
| **enum** Day { Воскресенье, Понедельник, Вторник, Среда, Четверг, Пятница,  Суббота }; |

По умолчанию значения перечисления начинаются с 0, и каждый последующий элемент увеличивается на значение **1**:

Воскресенье = 0

Понедельник = 1

Вторник = 2

и т. д.

* Вы можете изменить значение по умолчанию, указав начальное значение для вашего перечисления, как в следующем примере:

|  |
| --- |
| День перечисления { Воскресенье = 1, Понедельник, Вторник, Среда, Четверг, Пятница, Суббота }; |

* В этом примере *воскресенье* имеет значение **1** вместо значения по умолчанию **0**. Теперь понедельник — 2, вторник — 3 и т.д.
* Ключевое слово enum используется для указания «типа» переменной Day. В данном случае это тип перечисления.

Использование:

|  |
| --- |
| *// инициализируем переменную типа enum значением по имени*  Day favoriteDay = Day.Friday;  *// инициализируем переменную типа enum значением по индексу*  Day favoriteDay = (Day)4; |

**Последовательности**

Для работы с последовательностями подключите две директивы:

|  |
| --- |
| **using** System.Collections.Generic;  **using** System.Linq; |

**Генераторы последовательностей**

Пример: Генерация последовательности в указанном итнтервале:

Range (int start, int count);

start — The value of the first integer in the sequence.  
count — The number of sequential integers to generate.

|  |
| --- |
| IEnumerable<**int**> seq; *// definition*  seq = System.Linq.Enumerable.Range(3, 10);  **foreach** (**var** x **in** seq)  Console.Write($"{x} "); *// 3 4 5 6 7 8 9 10 11 12* |

**Генерация последовательностей с лямбда-выражениями:**

Пример: Сгенерировать последовательность целых чисел от 1 до 10 и вычислить квадраты этих чисел:

|  |
| --- |
| IEnumerable<**int**> squares;  squares=Enumerable.Range(1, 10).**Select**(x => x \* x);  **foreach** (**int** num **in** squares)  {  Console.WriteLine(num);  } |

Последовательность чисел в интервале от 1 до 9 с правилом: (a+b)\*2 (т.е. для первых двух чисел последовательности имеем (0+1)\*2 = 2)

|  |
| --- |
| IEnumerable<**int**> seq;  seq= Enumerable.Range(1, 9).**Select**((a, b) => (a+b)\*2);  **foreach** (**int** num **in** seq)  {  Console.Write(num); *// 2 6 10 14 18 22 26 30 34*  } |

**Задания и лабораторные си шарп**

**Лабораторная работа 1. Перечисления**

**Выполнить:** Попросите пользователя ввести отметку (значения 1, 2, 3, 4, 5). Проверьте введенное число, и распечатайте характеристику отметки (*very\_bad* — 1, *bad* — 2, *satisfactory* — 3, *good* — 4, *excellent* — 5). А также необходимо вывести отметку соответствующую характеристике *удовлетворительно (satisfactory)*.

**Указание:** Создайте функцию CheckMark с оператором switch, чтобы проверить введенную отметку.

**Пример выполнения:**

Введите отметку

2

Характеристика отметки 2: bad

Отметка для satisfactory: 3

[Название проекта: Lesson\_6Lab1, название файла L6Lab1.cs]

 Алгоритм:

1. Создайте консольное приложение с именем Lesson\_6Lab1.
2. В окне *Обозреватель решений* (Solution Explorer) найдите файл Program.cs и переименуйте его в L6Lab1.cs.
3. Подключите класс Console, чтобы постоянно не писать его название при обращении к его методам:

...

**using static** System.Console;

...

1. После открывающей скобки класса и непосредственно перед методом Main() введите следующий код для создания перечисления Marks, представляющего характеристики отметок:

...

**enum** Marks { very\_bad, bad, satisfactory, good, excellent, noSuchMark };

...

1. В функцию Main() добавьте следующий код:

...

**int** x = (**int**)Marks.satisfactory;

Console.WriteLine($"And also the characteristic satisfactory is for: {x} mark");

...

1. Этот код показывает два аспекта использования перечислений. Во-первых, можно исполльзовать предлагаемое автозаполнение и з выпадающего списка Marks.satisfactory. Во-вторых, можно использовать номер необходимого значения перечисления, сконвертировав его в тип int.
2. Запустите приложение, щелкнув CTRL+F5.
3. Под этим кодом добавьте код, в котором попросите пользователя ввести отметку (1—5):

...

WriteLine("Please enter a number");

**int** mark = **int**.Parse(ReadLine());

...

1. Создайте свою функцию CheckMark с двумя параметрами: параметр mark типа int — вводимый параметр, и параметр characteristic с ключевым словом ref и типом данных Marks.

static **void** CheckMark(**int** mark, ref Marks characteristic)

{

...

}

1. В функцию добавьте оператор switch для проверки введенного числа и присваивания соответствующей характеристики переменной characteristic:

switch (mark)

{

case 1:

characteristic = Marks.very\_bad;

break;

case 2:

characteristic = Marks.bad;

break;

case 3:

characteristic = Marks.satisfactory;

break;

case 4:

characteristic = Marks.good;

break;

case 5:

characteristic = Marks.excellent;

break;

default:

WriteLine("Invalid selection. Please select 1, 2, 3, 4 or 5.");

characteristic = Marks.noSuchMark;

break;

}

1. В функции main инициализируйте переменные, которые будут использоваться в качестве параметров функции:

...

WriteLine("Введите отметку");

**int** mark = **int**.Parse(ReadLine()); // этот код был

Marks characteristic = Marks.very\_bad; //нужно присвоить какое-то значение, неважно какое

1. Вызовите функцию и посмотрите на результат:

...

CheckMark(mark, ref characteristic);

WriteLine($"The characteristic for {mark} is: {characteristic}");

1. Запустите программу.

**Задание 1:**

**Выполнить:** Создайте консольное приложение Lesson\_6Task1. Удалите полностью функцию Main из проекта. Скопируйте код из [текстового файла](https://labs-org.ru/labs/DayOfWeek_task.txt) и вставьте его в основной класс (class Program) созданного проекта. Следуйте всем тегам TODO и выполняйте то, что там написано. Убедитесь, что проект работает правильно.  
    
**Пример выполнения:**

Введите день недели:

2

Вторник

[Название проекта: Lesson\_6Task1, название файла L6Task1.cs]

**ПОСЛЕДОВАТЕЛЬНОСТИ**

**Задание 2:**

**Выполнить:** Создайте последовательность из N целых чисел, начинающихся с a. a и N — это натуральные числа.

**Пример выполнения:**

Введите натуральные числа N и a:

2

5

Результат:

2 3 4 5 6

[Название проекта: Lesson\_6Task2, название файла L6Task2.cs]

**Задание 3:**

**Выполнить:** Создайте последовательность из **9** целых чисел, начиная с **1** с шагом=**2**.

**Указание:**чтобы получить шаг, вы можете использовать лямбда-выражение ... .Select((a,b)=> ...);. Подумайте, что нужно вставить вместо ....

**Пример выполнения:**

Результат:

1 3 5 7 9 11 13 15 17

[Название проекта: Lesson\_6Task3, название файла L6Task3.cs]

**ПРИЛОЖЕНИЯ ДЛЯ WINDOWS FORMS**

**Lab 6. Элементы управления Timer и PictureBox**

**Выполнить:** Разработайте приложение, в котором при нажатии на кнопку изображение будет перемещаться по экрану (например, по горизонтали).

[](https://labs-org.ru/wp-content/uploads/2016/06/1.png)

Форма «Анимация»

[Solution and Project name: Lesson\_6Lab2, file name L6Lab2.cs]

 Решение:

***Инструментарий***:

|  |  |
| --- | --- |
| **объект** | **свойство name** |
| форма |  |
| контейнер для картинки (PictureBox) | *pct* |
| кнопка (button) | *btnStart* |
| кнопка (button) | *btnExit* |
| таймер (timer) | *tmr* |

1. Создайте новый проект и расположите на новой форме элементы управления *PictureBox* (назовите его *pct*) и два элемента *Button*:*btnStart*и *btnExit*, как показано на рисунке. Создайте также элемент управления *Timer  (tmr),* который будет передвигать *PictureBox*через определенные промежутки времени.

Элемент *Timer*скрытый, он появляется ниже дизайна формы.

1. В окне свойств элемента *PictureBox* выберите свойство *Image* и в открывшемся диалоговом окне отметьте пункт *Local Resource (Локальный ресурс)*. Нажмите кнопку *Import* и выберите любой файл картинки.
2. Для того чтобы *PictureBox* перемещался вправо, необходимо увеличивать его свойство *Left*– координату левого края элемента управления. Поэтому запрограммируйте событие *Tick* элемента *Timer* следующим образом:

|  |
| --- |
| **private** **void** tmr\_Tick(**object** sender, EventArgs e)  {  pct.Left = pct.Left + 20;  } |

1. Теперь необходимо запустить *Таймер*. Это можно сделать нажатием кнопки *Старт*. Поместите следующий код в обработку события *Click* кнопки *Старт*:

|  |
| --- |
| **private** **void** btnStart\_Click(**object** sender, EventArgs e)  {  tmr.Enabled = **true**;  tmr.Interval = 50;  } |

 Запустите и отладьте приложение. Сохраните его.

***Дополнительное задание***.

1. Исправьте приложение так, чтобы *Графическое окно (PictureBox)* не выходило за край формы (останавливалось бы у правого края окна), а при нажатии на кнопку *Старт* перемещалось бы в исходное положение.
2. Измените приложение так, чтобы при первом нажатии на кнопку *Старт* графическое окно начинало двигаться, а при повторном – останавливалось. При этом должна меняться надпись на кнопке: *Старт* – при запуске, *Стоп* – при остановке.

***Вопросы для самоконтроля:***

1. Каково основное назначение элемента управления *PictureBox*?
2. Для чего используется элемент управления *Timer* при создании анимации?
3. Какое свойство *PictureBox* необходимо изменить для того, чтобы переместить этот элемент управления по вертикали?