ЛЯМБДА-ВЫРАЖЕНИЯ

Лямбда-выражения представляют упрощенную запись анонимных методов. С помощью лямбда-выражений можно написать локальные функции, которые затем можно передавать в другие функции в качестве аргументов или возвращать из них в качестве значения.

Лямбда-выражения особенно полезны при написании выражений запросов LINQ.

Лямбда-выражения имеют следующий синтаксис: слева от лямбда-оператора => (читается как "переходит" или "становится") определяется список параметров, а справа блок выражений, использующий эти параметры: (список_параметров) => выражение.

```
delegate int Square(int x); // объявляем делегат, принимающий int и возвращающий int
static void Main(string[] args)
    {

    Square squareInt = i => i * i; // объекту делегата присваивается лямбда-выражение

    int z = squareInt(6); // используем делегат
    // z равно 36
}
```

При использовании надо учитывать, что каждый параметр в лямбда-выражении неявно преобразуется в соответствующий параметр делегата, поэтому типы параметров должны быть одинаковыми, количество параметров должно быть таким же, как и у делегата и возвращаемое значение лямбда-выражений должно быть тем же, что и у делегата.

```
delegate bool StringEqual(string s1, string s2);
…
StringEqual stEq = (s1, s2) => s1 == s2;
```

Иногда компилятору бывает трудно или даже невозможно вывести типы входных параметров. В этом случае типы можно указать в явном виде:

```
(x, y) => x == y
```

```
(int x, string s) => s.Length > x
```

Бывает, что параметров не требуется. В этом случае вместо параметра в лямбда-выражении используются пустые скобки:

```
delegate void message(); // делегат без параметров
static void Main(string[] args)
    {
    message GetMessage = () => { Console.WriteLine("Лямбда-
    выражение"); };

    GetMessage();
    }
```

Лямбда-выражение необязательно должно принимать блок операторов и выражений. Оно может также принимать ссылку на метод:

```
delegate void message(); // делегат без параметров
    static void Main(string[] args)
    {
        message GetMessage = () => Show_Message();

        GetMessage();
    }
    private static void Show_Message()
    {
        Console.WriteLine("Привет мир!");
    }
```

Как и делегаты, лямбда-выражения можно передавать в качестве параметров методу.

```
delegate bool IsEqual(int x);
static void Main(string[] args)
{
    int[] integers = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    // найдем сумму чисел больше 5
    int result1 = Sum(integers, x => x > 5);
    Console.WriteLine(result1); // 30
    Console.Read();
}
private static int Sum (int[] numbers, IsEqual func)
{
    int result = 0;
    foreach(int i in numbers)
    {
        if (func(i))
            result += i;
    }
    return result;
}
```

Лямбда-выражения могут ссылаться на *внешние переменные*, находящиеся в области метода, в котором определена лямбда-функция, или в области типа, который содержит лямбда-выражение. Переменные, полученные таким способом, сохраняются для использования в лямбда-выражениях, даже если бы в ином случае они оказались за границами области действия и уничтожились сборщиком мусора. Внешней переменной должно быть присвоено определённое значение, прежде чем она сможет использоваться в лямбда-выражениях

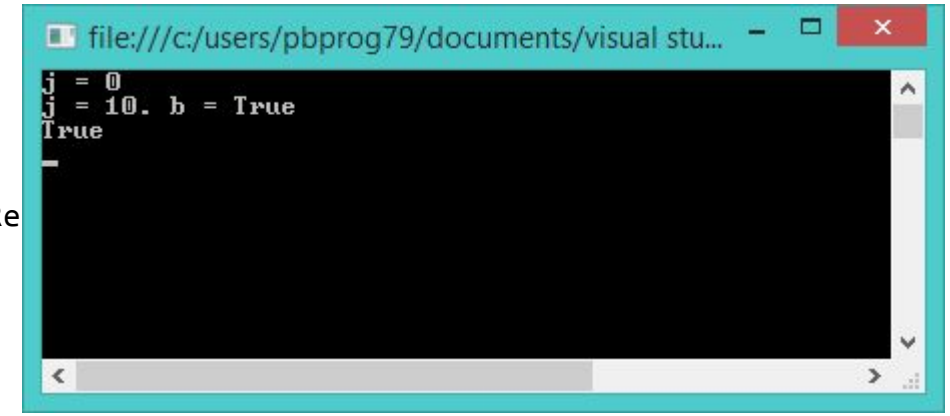
```
delegate bool D();
delegate bool D2(int i);

class Test
{
D del;
D2 del2;
public void TestMethod(int input)
{
int j = 0;
// Инициализация лямбда-выражения
// Обратите внимание, что получается доступ к двум внешним переменным
// del будет вызвана вместе с этим методом.
del = () => { j = 10; return j > input; };

// del2 будет вызвана после того, как TestMethod завершится
del2 = (x) => { return x == j; };

// Output: j = 0
// Делегат пока еще не вызван.
Console.WriteLine("j = {0}", j);
// вызов делегата
bool boolResult = del();
// Output: j = 10 b = True
Console.WriteLine("j = {0}. b = {1}", j, boolRe
}

static void Main()
{
```



Следующие правила применимы к области действия переменной в лямбда-выражениях.

- Захваченная переменная не будет уничтожена сборщиком мусора до тех пор, пока делегат, который на нее ссылается, не перейдет в статус подлежащего уничтожению при сборке мусора.

- Переменная, объявленная в лямбда-выражении, невидима во внешнем методе.

- Лямбда-выражение не может непосредственно захватывать параметры **ref** или **out** из метода, в котором они находятся.

Следующие правила применимы к области действия переменной в лямбда-выражениях.

- Оператор **Return** в лямбда-выражении не приводит к возврату (завершению) метода, в котором объявлено/вызвано лямбда-выражение.

- Лямбда-выражение не может содержать оператора **goto**, оператора **break** или оператора **continue** внутри лямбда-функции, если целевой объект перехода находится вне блока. Если целевой объект находится внутри блока, то наличие оператора перехода за пределами лямбда-функции также будет ошибкой.