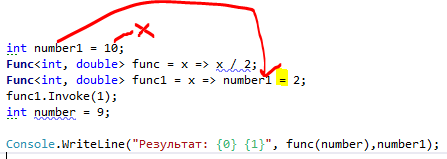


При захваченной переменой е значение меняется

Полная запись лямбда выражения



Деление будет произведено между int так как передаем int



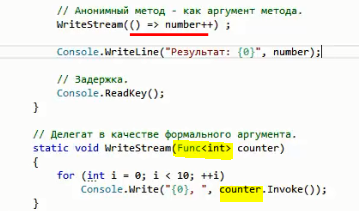


Деление будет произведено между double так как передаем double





Метод принимает другой метод (лямда выражение)

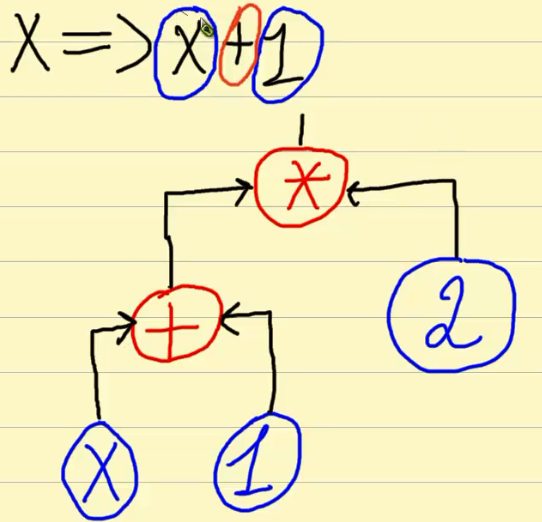
)

Expression

Деревья Выражений

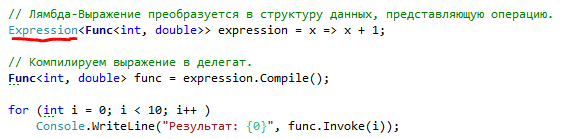


Одно лямбда выражение вернуло результат (сложение), после к результат у было применено второе лямбда выражение (умножение)



Нужно создать переменную типа Expression параметризированную классом делегата

expression.Compile(); - возвращает ссылку на класс делегат



ParameterExpression

// Представляем параметр в списке параметров лямбда-выражения.

ParameterExpression n = Expression.Parameter(typeof(int), "n"); - Эта строка говорит, что нам нужна переменная n типа int.

ParameterExpression t = Expression.Parameter(typeof(int), "t"); - аналогия

ConstantExpression

var cons1 = Expression.Constant(1); - создает константу

var c2 = Expression.Constant(2);

BinaryExpression

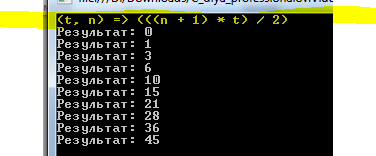
// Прибавляем число 1 к параметру n.

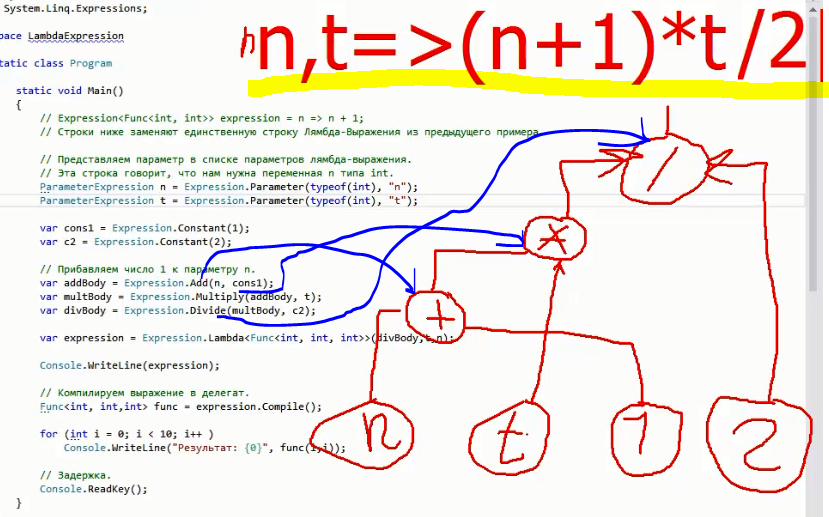
BinaryExpression addBody = Expression.Add(n, cons1); - сложение

BinaryExpression multBody = Expression.Multiply(addBody, t); - умножение

BinaryExpression divBody = Expression.Divide(multBody, c2); - деление

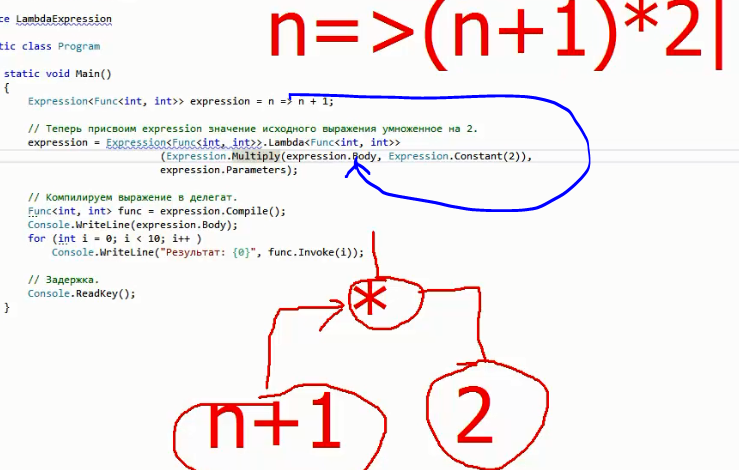
Expression<**Func**<int, int, int>> expression = Expression.Lambda<**Func**<int, int, int>>(divBody, t, n); - сформировать дерево выражений (необходимо передать тело выражения и параметры)



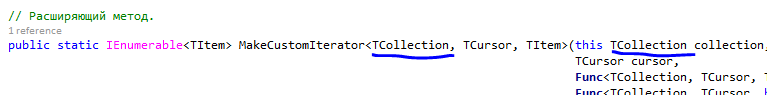


Второй способ

expression.Body – возвращает тело лямбда выражения

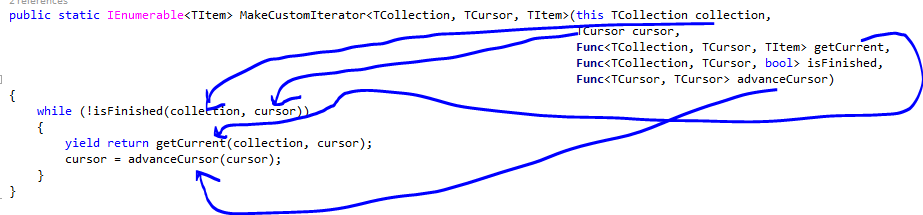


Если в расширяющем методе this является параметризированным (обобщенным) типом, то данный метод может быть применен ко всем типам

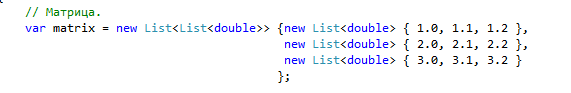


Итератор

Имеется расширяющий метод

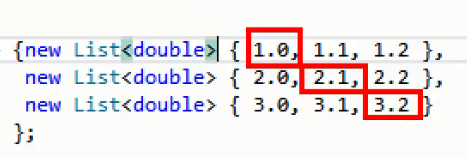


Матрица по которой будем ходить

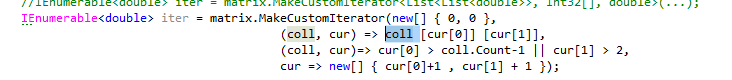


1-Передаем в него курсор 0,0, т.е. изначально 1 элемент 1-го листа в массиве

2-Вторым параметром коллекцию и курсор (к коллекции будет подставляться значения курсора пример - matrix[0][0], потом matrix[1][1] и т.д.)

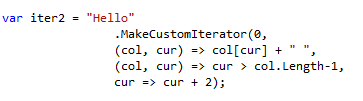


3-проверяем если matrix[будет 3][или здесь 3], то это выход за пределы коллекции и будет возвращен false (выход из цикла)





Или

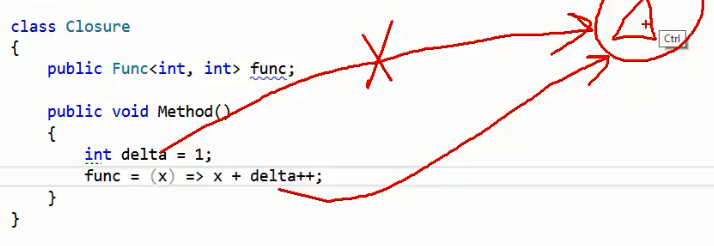




При замыкании переменной в методе

При захвате переменной delta будет создан объект (упакованный int) на куче и обе переменных как в методе так и в лямбда выражении будут ссылаться на этот объект на куче

Гарбочка лектор его не удалит пока существует одна из ссылок



Рекурсия в лямбда выражении

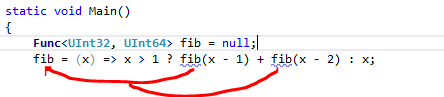


Мемоизация

- замена делегата другим делегатом.

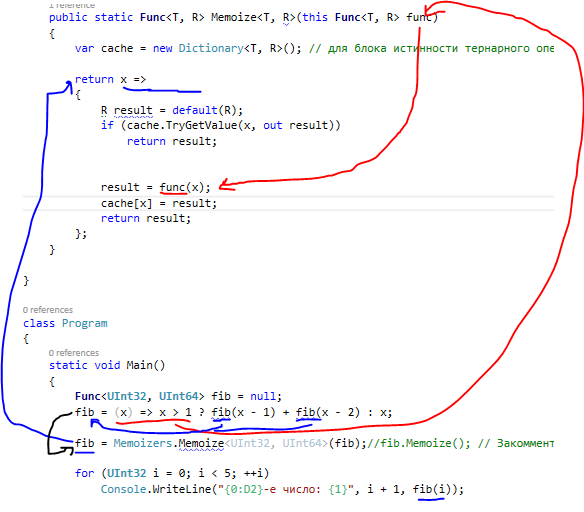
Числа фабианчи, равно сумме цифр (5 его числа 5-1=4 и 5-2=3 или 2 2-1=1 и 2-2=0)

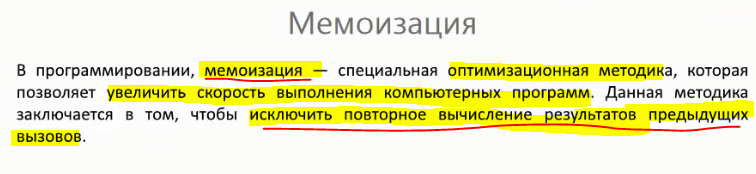
Создаем делегат и инициализируем его лямбда методом в котором в зависимости от переменной х выполняется то или иное действие (рекурсия!!!)

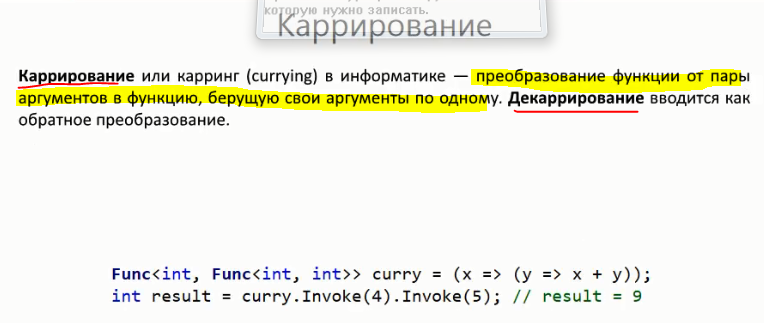


После В методе Memoize мы в первом параметре получаем метод делегат, а именно рекурсивные методы и в переменной func(x); всегда будет храниться эти методы

После делегат fib инициализируем новой лямбда выражением и так как в теле лямда func(x); есть ссылки на fib то он будет изначально вызывать метод синий, а если в словаре не находит то метод красный







Каррирование цепочка функций

Вызываем два метода на одной переменной за счет того что первый метод возвращает делегат

