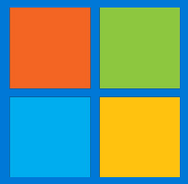


Microsoft

CIKLUM

EMPOWERING COLLABORATION

What's new in C# 7

Bezpalyi Kyrylo

.Net Developer @ Ciklum

MSP @ Microsoft

Agenda

- Roslyn
- C#

Roslyn

Roslyn

- C# and VB language engine for *all the things*!
  - All the IDEs and editors
  - All the linters and analysis tools
  - All the fixing and refactoring and code generation tools
  - All the scripting and all the REPLs

There should only need to be *one* code base in the world for understanding C#

Starting in early 2009

Luca Bolognese: C# and VB.NET Co-Evolution – The Twain Shall Meet

http://bit.ly/dotNetCoEvolution

C#

C# new features

Version	Key new features
C# 7	In progress
C# 6	String interpolation; Null-Conditional Operator; nameof; Expression body functions; Auto-Property improvements; using static
C# 5	async/await; Caller information
C# 4	dynamic; Named and optional arguments; Task Parallel Library
C# 3	LINQ; Lambda; Expression Trees; Anonymous types; Implicit typing (var)
C# 2	Generics; Anonymous methods; Nullable types
C# 1	Managed

C# techniques

Version	Techniques
C# 7	OOP, Generic programming, FP Techniques, Dynamic Typing, Async programming, Roslyn, Meta-programming?
C# 6	OOP, Generic programming, FP Techniques, Dynamic Typing, Async programming, Roslyn
C# 5	OOP, Generic programming, FP Techniques, Dynamic Typing, Async programming
C# 4	OOP, Generic programming, FP Techniques, Dynamic Typing
C# 3	OOP, Generic programming, FP Techniques
C# 2	OOP, Generic programming
C# 1	OOP

C# 7 features list

Strong interest:

- Local functions **– finished**
- Tuples **– in-progress**
- Pattern matching, part I **– in-progress**
- Ref locals and ref returns **– finished**
- replace/original (part of generators)

Some interest:

- Binary literals **– finished**
- Digit separators **– finished**
- out var **– finished**
- async main
- address of static method for unsafe interop code
- More expression bodies
- Throw expressions

https://github.com/dotnet/roslyn/issues/2136

Digit separators & Binary literals

Allow _ as separator

- `var d = 123_456;`
- `var x = 0xAB_CD_EF;`

Allow 0b as binary number representation:

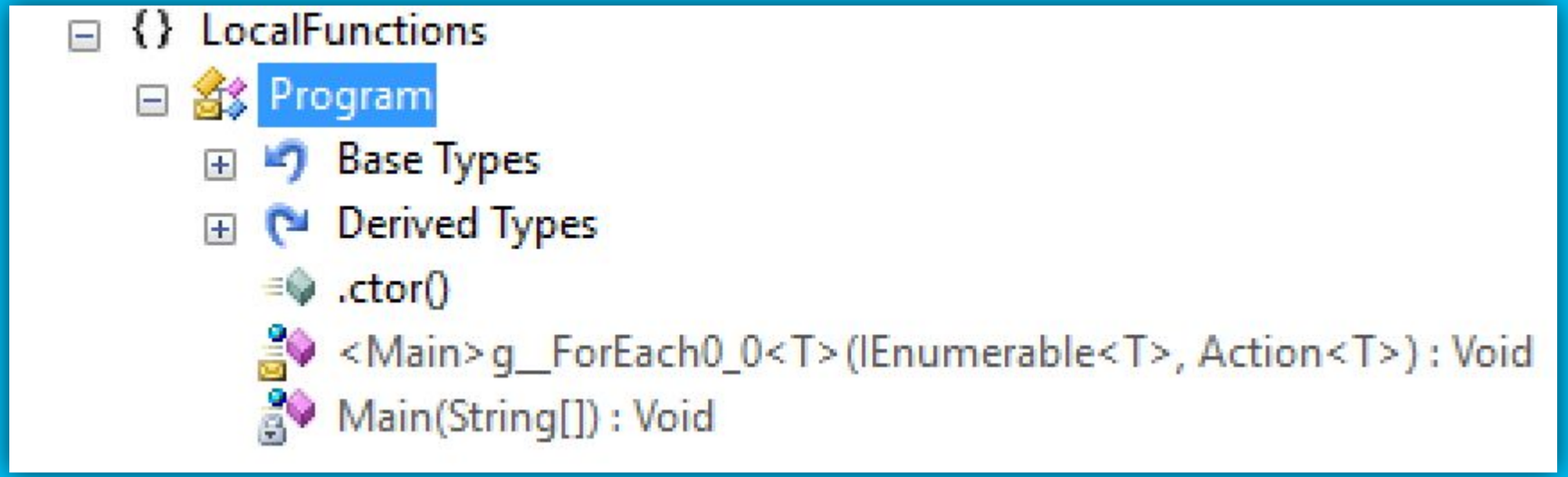
- `var b = 0b1010_1011_1100_1101_1110_1111;`

Local functions

```
static void Main(string[] args)
{
    void ForEach<T>(IEnumerable<T> source, Action<T> action)
    {
        foreach (var item in source)
        {
            action?.Invoke(item);
        }
    }

    var list = new[] { "First", "Second", "Third", "Forth", "Fifth" };
    ForEach(list, Console.WriteLine);
    Console.ReadKey();
}
```

Demo: local function



Out var

```
int x;

if(int.TryParse(Console.ReadLine(), out x))
{
   Console.WriteLine($"{nameof(x)} = {x}");
}
```

```
if(int.TryParse(Console.ReadLine(), out int x))
{
   Console.WriteLine($"{nameof(x)} = {x}");
}
```

For Preview 4 restriction: Out variables are scoped to the statement they are declared in

Demo: out var

```
private static void PrintCoordinates(Point p)
{
    int num;
    int num2;
    if (p.GetCoordinates(out num, out num2))
    {
        Console.WriteLine($"({num}, {num2})");
    }
}
```

Pattern matching

```
static void PrintInt(object o)
{
    if (o is int i || (o is string s && int.TryParse(s, out i)))
    {
        Console.WriteLine(i);
    }
}
```

Demo pattern matching

```
private static void PrintInt(object o)
{
    string str;
    int? nullable = o as int?;
    int valueOrDefault = nullable.GetValueOrDefault();
    if (nullable.HasValue || (((str = o as string) != null) && int.TryParse(str, out valueOrDefault)))
    {
        Console.WriteLine(valueOrDefault);
    }
}
```

Tuples

```
static void Main(string[] args)
{
    var data = new[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    var result = GetMinMax(data);
    Console.WriteLine($"Min: {result.Item1} Max:
{result.Item2}");
    Console.ReadKey();
}

static Tuple<T,T> GetMinMax<T>(IEnumerable<T> source) =>
     Tuple.Create(source.Min(), source.Max());
```

```
static void Main(string[] args)
{
    var data = new[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    var result = GetMinMax(data);
    Console.WriteLine($"Min: {result.Min} Max: {result.Max}");
    Console.ReadKey();
}

static (T Min, T Max) GetMinMax<T>(IEnumerable<T> source) =>
            (source.Min(), source.Max());
```

```
PM> Install-Package
System.ValueTuple -Pre
```

Tuples deconstruction (decomposition)

```
static void Main(string[] args)
{
    var data = new[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    var result = GetMinMax(data);
    Console.WriteLine($"Min: {result.Min} Max: {result.Max}");
    Console.ReadKey();
}

static (T Min, T Max) GetMinMax<T>(IEnumerable<T> source) =>
            (source.Min(), source.Max());
```

```
static void Main(string[] args)
{
    var data = new[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    var (min, max) = GetMinMax(data);
    Console.WriteLine($"Min: {min} Max: {max}");
    Console.ReadKey();
}

static (T Min, T Max) GetMinMax<T>(IEnumerable<T> source) =>
            (source.Min(), source.Max());
```

Demo tuples

```
internal class Program
{
    // Methods
    [return: TupleElementNames(new string[] { "Min", "Max" })]
    private static ValueTuple<T, T> GetMinMax<T>(IEnumerable<T> source) =>
        new ValueTuple<T, T>(source.Min<T>(), source.Max<T>());

    private static void Main(string[] args)
    {
        ValueTuple<int, int> minMax = GetMinMax<int>(new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 });
        int num = minMax.Item1;
        int num2 = minMax.Item2;
        Console.WriteLine($"Min: {num} Max: {num2}");
        Console.ReadKey();
    }
}
```

Ref returns and locals

```
public ref int Find(int number, int[] numbers)
{
    for (int i = 0; i < numbers.Length; i++)
    {
        if (numbers[i] == number)
        {
            return ref numbers[i]; // return the storage location, not the value
        }
    }
    throw new IndexOutOfRangeException($"{nameof(number)} not found");
}

void Main(string[] args)
{
    int[] array = { 1, 15, -39, 0, 7, 14, -12 };
    ref int place = ref Find(7, array); // aliases 7's place in the array
    place = 9; // replaces 7 with 9 in the array
    Console.WriteLine(array[4]); // prints 9
}
```

Demo ref return

```
public static unsafe ref int Find(int number, int[] numbers)
{
    for (int i = 0; i < numbers.Length; i++)
    {
        if (numbers[i] == number)
        {
            return (int) &(numbers[i]);
        }
    }
    throw new IndexOutOfRangeException($"{"number"} not found");
}

private static void Main(string[] args)
{
    int[] numbers = new int[] { 1, 15, -39, 0, 7, 14, -12 };
    Find(7, numbers) = 9;
    Console.WriteLine(numbers[4]);
}
```

More expression bodies

```
class Person
{
    private static ConcurrentDictionary<int, string> names = new ConcurrentDictionary<int,
string>();
    private int id = GetId();

    public Person(string name) => names.TryAdd(id, name);// constructors
    ~Person() => names.TryRemove(id, out **);                // destructors
    public string Name
    {
        get => names[id];                                    // getters
        set => names[id] = value;                // setters
        }
    }
}
```

async Main

```
static async Task DoWork()
{
    await ...
    await ...
}

static void Main()
{
    DoWork().GetAwaiter().GetResult();
}
```

```
static async Task<int> Main(string[] args)
{
    // User code goes here
}

static int $GeneratedMain(string[] args)
{
    return Main(args).GetAwaiter().GetResult();
}
```

Maybe: Records

```
class Person(string First, string Last);
```

```
class Person : IEquatable<Person>
{
    public string First { get; }
    public string Last { get; }

    public Person(string First, string Last) { this.First = First; this.Last = Last; }

    public bool Equals(Person other)
        => other != null && First == other.First && Last == other.Last;

    public override bool Equals(object obj) => obj is Person other ? Equals(other) : false;
    public override int GetHashCode() => GreatHashFunction(First, Last);
    …
}
```

Maybe: Creating immutable objects

```
var p1 = new Point { X = 3, Y = 7 };

var p2 = p1 with { X = -p1.X };
```

Resources

- Roslyn
  - https://github.com/dotnet/roslyn
- .NET blog
  - https://blogs.msdn.microsoft.com/dotnet
- VS blog
  - https://blogs.msdn.microsoft.com/visualstudio/
- CoreCLR
  - https://github.com/dotnet/coreclr
- Sergey Teplyakov blog
  - http://sergeyteplyakov.blogspot.com/

Вопросы?

Microsoft

CIKLUM
EMPOWERING COLLABORATION

Code: https://github.com/compilyator/CSharp7

Slides: https://goo.gl/4YGzHe