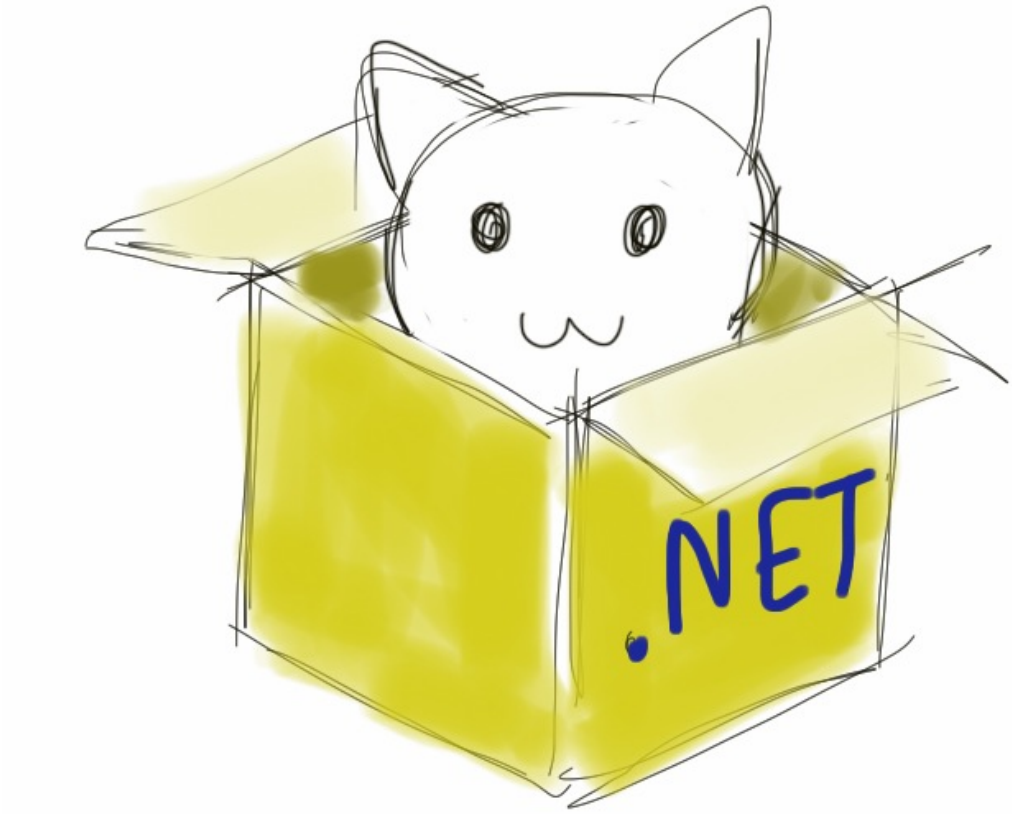
ProblemBook.NET



ProblemBook.NET»
© 2014, Andrey Akinshin <andrey.akinshin@gmail.com>
Edited by Ivan Pashchenko

This work is licensed under a "Attribution-NonCommercial-NoDerivatives 4.0 International" (CC BY-NC-ND 4.0). To view a copy of this license, visit:
http://creativecommons.org/licenses/by-nc-nd/4.0/.
Text version: v-2014-Dec-12
To view the source code and the actual version of the text, visit:
https://github.com/AndreyAkinshin/ProblemBook.NET

Contents

Introduction **4**

Problems **6**

1 OOP (Problems) . . . 6
2 LINQ (Problems) . . . 10
3 Mathematics (Problems) . . . 14
4 Value types (Problems) . . . 16
5 Strings (Problems) . . . 18
6 Multithreading (Problems) . . . 21

Solutions **23**

7 OOP (Solutions) . . . 23
8 LINQ (Solutions) . . . 25
9 Mathematics (Solutions) . . . 32
10 Value types (Solutions) . . . 34
11 Strings (Solutions) . . . 35
12 Multithreading (Solutions) . . . 38

Bibliography **39**

Introduction

This book is a collection of problems on the knowledge of the .NET platform and the C# programming language. To avoid misunderstandings, just want to say that the book *is not*:

- This book is not a universal way to check your knowledge platform .NET. If you are easily solved all problems, it does not mean that you is a wonderful .NET-programmer. And if you met a lot of new things for yourself, then it does not mean that you don't know .NET.
- This book is not a compilation of new, previously unseen problems. Many examples can be found in the literature, in questions on StackOverflow, in programmer's blogs. Just because they have become classics.
- This book is not focused on those who already feel Senior .NET Developer and want to learn a lot.

So what is this book? ProblemBook.NET is an attempt to gather in one place a variety of interesting practical problems on the knowledge platform. Problems are divided into chapters, so you can not read everything, but only questions of those areas that are of interest to you. In this book you will not find a deep philosophical questions such as "What is a class?" or "Why is polymorphism needed?". Most of the problems is a piece of C#-code for which you should to determine the result. Each question is provided with a solution (sometimes with a description of why .NET behaves this way).

It is recommended to treat the problem is not as a way to test your knowledge as well as the starting point for the discussion of various aspects of the platform. If you find something new for yourself, it's a great cause to learn .NET a little more detail. Try to play around with the code: modify it and learn how changes affect the result. Read the relevant literature. Think about how the new knowledge can be useful to you in your work.

ProblemBook.NET distributed electronically, the development is on GitHub. It should also be noted that at present the problem book is far from the final version. The book will be updated with new problems and old will be refined and improved.

Technical Details

Today, there are two popular platform implementation .NET: The original Microsoft .NET Framework (hereinafter - MS.NET) and Mono. It is known that the behavior of some code fragments may change when you change the implementation. Similarly, the behavior might depends on the CLR or processor architecture. If the result of the fragment of the code depends on the environment, it is likely that the answers will be given an explanation. However, it is *not guaranteed*. If one of the examples you have received an answer that is different from this in the book, please contact the author, that he corrected this mistake.

Examples of code in all tasks can be run from LINQPad (in C# Statements or C# Program, depending on the availability of the method `Main`), but you should import the following namespace (Query —> Query Properties —> Additional Namespace Imports):

```
System.Globalization
System.Runtime.InteropServices
System.Runtime.CompilerServices
```

Thanks

The author sincerely thanks the Chief Editor Ivan Pashchenko for useful discussions during the writing of this book.

The author evinces appreciation to Ekaterina Demidova for the kindly providing illustrations.

Problems

1 OOP (Problems)

1. What will the following code display?

```
void Foo(object a)
{
  Console.WriteLine("object");
}
void Foo(object a, object b)
{
  Console.WriteLine("object, object");
}
void Foo(params object[] args)
{
  Console.WriteLine("params object[]");
}
void Foo<T>(params T[] args)
{
  Console.WriteLine("params T[]");
}
class Bar { }
void Main()
{
  Foo();
  Foo(null);
  Foo(new Bar());
  Foo(new Bar(), new Bar());
  Foo(new Bar(), new object());
}
```

Solution

2. What will the following code display?

```
class Foo
{
  public virtual void Quux(int a)
  {
    Console.WriteLine("Foo.Quux(int)");
  }
}
class Bar : Foo
{
  public override void Quux(int a)
  {
    Console.WriteLine("Bar.Quux(int)");
  }
  public void Quux(object a)
  {
    Console.WriteLine("Bar.Quux(object)");
  }
}
class Baz : Bar
{
  public override void Quux(int a)
  {
    Console.WriteLine("Baz.Quux(int)");
  }
  public void Quux<T>(params T[] a)
  {
    Console.WriteLine("Baz.Quux(params T[])");
  }
}
void Main()
{
  new Bar().Quux(42);
  new Baz().Quux(42);
}
```

Solution

3. What will the following code display?

```
class Foo
{
  public Foo()
  {
    Quux();
  }
  public virtual void Quux()
  {
    Console.WriteLine("Foo.Quux()");
  }
}
class Bar : Foo
{
  protected string name;
  public Bar()
  {
    name = "Bar";
  }
  public override void Quux()
  {
    Console.WriteLine("Bar.Quux(),  " + name);
  }
  public void Quux(params object[] args)
  {
    Console.WriteLine("Bar.Quux(params object[])");
  }
}
class Baz : Bar
{
  public Baz()
  {
    name = "Baz";
    Quux();
    ((Foo)this).Quux();
  }
}
void Main()
{
  new Baz();
}
```

Solution

4. What will the following code display?

```
class Foo
{
  protected class Quux
  {
    public Quux()
    {
      Console.WriteLine("Foo.Quux()");
    }
  }
}
class Bar : Foo
{
  new class Quux
  {
    public Quux()
    {
      Console.WriteLine("Bar.Quux()");
    }
  }
}
class Baz : Bar
{
  public Baz()
  {
    new Quux();
  }
}
void Main()
{
  new Baz();
}
```

Solution

2 LINQ (Problems)

1. What will the following code display?

```
public static string GetString(string s)
{
  Console.WriteLine("GetString: " + s);
  return s;
}
public static IEnumerable<string> GetStringEnumerable()
{
  yield return GetString("Foo");
  yield return GetString("Bar");
}
public static string[] EnumerableToArray()
{
  var strings = GetStringEnumerable();
  foreach (var s in strings)
    Console.WriteLine("EnumerableToArray: " + s);
  return strings.ToArray();
}
void Main()
{
  EnumerableToArray();
}
```

Solution

2. What will the following code display?

```
IEnumerable<string> Foo()
{
  yield return "Bar";
  Console.WriteLine("Baz");
}
void Main()
{
  foreach (var str in Foo())
    Console.Write(str);
}
```

Solution

3. What will the following code display?

```
var actions = new List<Action>();
foreach (var i in Enumerable.Range(1, 3))
  actions.Add(() => Console.WriteLine(i));
foreach (var action in actions)
  action();
```

Solution

4. What will the following code display?

```
var list = new List<string> { "Foo", "Bar", "Baz" };
var query = list.Where(c => c.StartsWith("B"));
list.Remove("Bar");
Console.WriteLine(query.Count());
```

Solution

5. What will the following code display?

```
var list = new List<string> { "Foo", "Bar", "Baz" };
var startLetter = "F";
var query = list.Where(c => c.StartsWith(startLetter));
startLetter = "B";
query = query.Where(c => c.StartsWith(startLetter));
Console.WriteLine(query.Count());
```

Solution

6. What will the following code display?

```
int Inc(int x)
{
  Console.WriteLine("Inc: " + x);
  return x + 1;
}
void Main()
{
  var numbers = Enumerable.Range(0, 10);
  var query =
    (from number in numbers
     let number2 = Inc(number)
     where number2 % 2 == 0
     select number2).Take(2);
  foreach (var number in query)
    Console.WriteLine("Number: " + number);
}
```

Solution

7. At what point will happen `Exception`?

```
public static IEnumerable<int> GetSmallNumbers()
{
  throw new Exception();
  yield return 1;
  yield return 2;
}
void Main()
{
  var numbers = GetSmallNumbers();
  var evenNumbers = numbers.Select(n => n * 2);
  Console.WriteLine(evenNumbers.FirstOrDefault());
}
```

Solution

8. At what point will happen `Exception`?

```
public static IEnumerable<int> GetSmallNumbers()
{
  yield return 1;
  throw new Exception();
  yield return 2;
}
void Main()
{
  var numbers = GetSmallNumbers();
  var evenNumbers = numbers.Select(n => n * 2);
  Console.WriteLine(evenNumbers.FirstOrDefault());
}
```

Solution

9. What will the following code display?

```
public static IEnumerable<int> GetSmallNumbers()
{
  try
  {
    yield return 1;
    yield return 2;
  }
  finally
  {
    Console.WriteLine("Foo");
  }
}
void Main()
{
  Console.WriteLine(GetSmallNumbers().First());
}
```

Solution

3 Mathematics (Problems)

1. What will the following code display?

```
Console.WriteLine(
  "| Number | Round | Floor | Ceiling | Truncate | Format |");
foreach (var x in new[] { -2.9, -0.5, 0.3, 1.5, 2.5, 2.9 })
{
  Console.WriteLine(string.Format(CultureInfo.InvariantCulture,
    "| {0,6} | {1,5} | {2,5} | {3,7} | {4,8} | {0,6:N0} |",
    x, Math.Round(x), Math.Floor(x),
    Math.Ceiling(x), Math.Truncate(x)));
}
```

Solution

2. What will the following code display?

```
Action<int,int> print = (a, b) =>
  Console.WriteLine("{0,2} = {1,2} * {2,3}   + {3,3}  ",
                    a, b, a/b, a%b);
Console.WriteLine(" a =  b * (a/b) + (a%b)");
print(7, 3);
print(7, -3);
print(-7, 3);
print(-7, -3);
```

Solution

3. What will the following code display?

```
Console.WriteLine("0.1 + 0.2 {0} 0.3",
  0.1 + 0.2 == 0.3 ? "==" : "!=");
```

Solution

4. What will the following code display?

```
var zero = 0;
try
{
  Console.WriteLine(42 / 0.0);
  Console.WriteLine(42.0 / 0);
  Console.WriteLine(42 / zero);
}
catch (DivideByZeroException)
{
  Console.WriteLine("DivideByZeroException");
}
```

Solution

5. What will the following code display?

```
var maxInt32 = Int32.MaxValue;
var maxDouble = Double.MaxValue;
var maxDecimal = Decimal.MaxValue;
checked
{
  Console.Write("Checked   Int32   increased max: ");
  try { Console.WriteLine(maxInt32 + 42); }
  catch { Console.WriteLine("OverflowException"); }
  Console.Write("Checked   Double  increased max: ");
  try { Console.WriteLine(maxDouble + 42); }
  catch { Console.WriteLine("OverflowException"); }
  Console.Write("Checked   Decimal increased max: ");
  try { Console.WriteLine(maxDecimal + 42); }
  catch { Console.WriteLine("OverflowException"); }
}
unchecked
{
  Console.Write("Unchecked Int32   increased max: ");
  try { Console.WriteLine(maxInt32 + 42); }
  catch { Console.WriteLine("OverflowException"); }
  Console.Write("Unchecked Double  increased max: ");
  try { Console.WriteLine(maxDouble + 42); }
  catch { Console.WriteLine("OverflowException"); }
  Console.Write("Unchecked Decimal increased max: ");
  try { Console.WriteLine(maxDecimal + 42); }
  catch { Console.WriteLine("OverflowException"); }
}
```

Solution

6. What will the following code display?

```
int a = 0;
int Foo()
{
  a = a + 42;
  return 1;
}
void Main()
{
  a += Foo();
  Console.WriteLine(a);
}
```

Solution

4 Value types (Problems)

1. What will the following code display?

```
struct Foo
{
  int value;
  public override string ToString()
  {
    if (value == 2)
      return "Baz";
    return (value++ == 0) ? "Foo" : "Bar";
  }
}
void Main()
{
  var foo = new Foo();
  Console.WriteLine(foo);
  Console.WriteLine(foo);
  object bar = foo;
  object qux = foo;
  object baz = bar;
  Console.WriteLine(baz);
  Console.WriteLine(bar);
  Console.WriteLine(baz);
  Console.WriteLine(qux);
}
```

Solution

2. What will the following code display?

```
public struct Foo
{
  public int Value;
  public void Change(int newValue)
  {
    Value = newValue;
  }
}
public class Bar
{
  public Foo Foo { get; set; }
}
void Main()
{
  var bar = new Bar { Foo = new Foo() };
  bar.Foo.Change(5);
  Console.WriteLine(bar.Foo.Value);
}
```

Solution

3. What will the following code display?

```
var x = new
  {
    Items = new List<int> { 1, 2, 3 }.GetEnumerator()
  };
while (x.Items.MoveNext())
  Console.WriteLine(x.Items.Current);
```

Solution

4. What will the following code display?

```
public struct Foo
{
  public byte Byte1;
  public int Int1;
}
public struct Bar
{
  public byte Byte1;
  public byte Byte2;
  public byte Byte3;
  public byte Byte4;
  public int Int1;
}
void Main()
{
  Console.WriteLine(Marshal.SizeOf(typeof(Foo)));
  Console.WriteLine(Marshal.SizeOf(typeof(Bar)));
}
```

Solution

5 Strings (Problems)

1. What will the following code display?

```
Console.WriteLine(1 + 2 + "A");
Console.WriteLine(1 + "A" + 2);
Console.WriteLine("A" + 1 + 2);
```

Solution

2. What will the following code display?

```
Console.WriteLine(1 + 2 + 'A');
Console.WriteLine(1 + 'A' + 2);
Console.WriteLine('A' + 1 + 2);
```

Solution

3. What will the following code display?

```
try
{
  Console.WriteLine(((string)null + null + null) == null);
}
catch (Exception e)
{
  Console.WriteLine(e.GetType());
}
```

Solution

4. Are the following methods of comparison term `a` and`b` insensitive equivalent:

```
Console.WriteLine(
  string.Compare(a.ToUpper(), b.ToUpper()));
Console.WriteLine(
  string.Compare(a, b, StringComparison.OrdinalIgnoreCase));
```

Solution

5. Please, append the following code so that it displays the console output contained the string `Foo`, without redefining the standard method `Console.WriteLine`?

```
// ???
Console.WriteLine(-42);
```

Solution

6. Look to the following code:

```
Thread.CurrentThread.CurrentUICulture =
  CultureInfo.CreateSpecificCulture("ru-RU");
Thread.CurrentThread.CurrentCulture =
  CultureInfo.CreateSpecificCulture("en-US");
var dateTime = new DateTime(2014, 12, 31, 13, 1, 2);
Console.WriteLine(dateTime);
```

Which locale will be choose for date formatting?

Solution

7. Can the two lines coincide by the `String.CompareTo` method, but differ by the `String.Equals` method? How to contact them the `==`?

Solution

8. Please, append the following code so that it displays the console output the `aaaaa` string, without redefining the standard method `Console.WriteLine`?

```
// ???
Console.WriteLine("Hello");
```

Solution

9. What will the following code display?

```
var x = "AB";
var y = new StringBuilder().Append('A').Append('B').ToString();
var z = string.Intern(y);
Console.WriteLine(x == y);
Console.WriteLine(x == z);
Console.WriteLine((object)x == (object)y);
Console.WriteLine((object)x == (object)z);
```

Solution

10. Look to the following code:

```
[assembly: CompilationRelaxationsAttribute
  (CompilationRelaxations.NoStringInterning)]
```

Does it means, that the literal string in this assembly now will be interned?

Solution

11. If the mechanism of internment lines under this version of the CLR is disabled and in a literal line of code `"Hello"` occurs twice, how many times it will occur in the assembly metadata?

Solution

12. Keep in regular strings secret data (such as a password) is a bad way, since string is stored in memory in the clear, hackers can easily get to it. What are the standard tools to protect data?

Solution

13. Suppose we have an array of different strings. Can the sorting method from time to time give different results?

Solution

14. Can the number of text elements (which can be accessed via `TextElementEnumerator`) of string differ from the number of char-forming her character?

Solution

6 Multithreading (Problems)

1. What will the following code display?

```
[ThreadStatic]
static int Foo = 42;

void Main()
{
  var thread = new Thread(() => Console.WriteLine(Foo));
  thread.Start();
  thread.Join();
}
```

Solution

2. What will the following code display?

```
class Foo
{
  private readonly ReaderWriterLockSlim lockSlim;

  public Foo(ReaderWriterLockSlim lockSlim)
  {
    this.lockSlim = lockSlim;
    lockSlim.EnterReadLock();
  }

  ~Foo()
  {
    Console.WriteLine("~Foo: start");
    try
    {
      if (lockSlim != null)
        lockSlim.ExitReadLock();
    }
    catch (Exception e)
    {
      Console.WriteLine("Exception: " + e.GetType().Name);
    }
    Console.WriteLine("~Foo: finish");
  }
}
void Main()
{
  var foo = new Foo(new ReaderWriterLockSlim());
  GC.Collect();
  GC.WaitForPendingFinalizers();
}
```

Solution

Solutions

7 OOP (Solutions)

1. Problem "Oop:OverloadResolutionBasic"

   Answer

   ```
   params object[]
   params object[]
   params T[]
   params T[]
   object, object
   ```

   Explanation

   There are methods:

   - `void Foo(object a) -> object`
   - `void Foo(object a, object b) -> object, object`
   - `void Foo(params object[] args) -> params object[]`
   - `void Foo<T>(params T[] args) -> params T[]`

   Let's consider each call separately.

   - `Foo() -> params object[]`

   The `object` and `object, object` options is inaccessible because of number of arguments. The `params T[]` option is inaccessible because of the compiler can't resolve `T`. Thus, the right options is `params object[]`.

   - `Foo(null) -> params object[]`

   The `object, object` option is inaccessible because of number of arguments. The `params T[]` options is inaccessible because of the compiler can't resolve `T`. The `params object[]` option is acceptable in its `expanded` and `unexpanded` forms. In this case, we should choose the `expanded` form, i.e. the method signature will look like `Foo(object[] args)`. Now, we have the `object` and `object[]` options. The compiler will choose more specific option, i.e. `object[]` (or `params object[]`).

   - `Foo(new Bar()) -> params T[]`

The `object, object` option is inaccessible because of number of arguments. The `object` and `params object[]` options request additional implicitly conversion `Bar` to `object`. So, the `params T[]` (or `params Bar[]`) option is more preferable.

- `Foo(new Bar(), new Bar())` —> `params T[]`

The `object, object` option is inaccessible because of number of arguments. The `object` and `params object[]` options request additional implicitly conversion `Bar` to `object`. So, the `params T[]` (or `params Bar[]`) option is more preferable.

- `Foo(new Bar(), new object())` —> `object, object`

The `object` option is inaccessible because of number of arguments. There is exactly one option without the `params` keyword: `object, object`. It is more preferable.

Links

- "Overload resolution", "Method invocations", "Applicable function member" in MSDN
- "Overloading" in "C# in Depth"

2. Problem "Oop:OverloadResolutionOverride"

Answer

```
Bar.Quux(object)
Baz.Quux(params T[])
```

Explanation

There is a rule: if compiler found a suitable signature for a method call in the "current" class, compiler will not look to parents classes. In this problem, the `Bar` and `Baz` classes have own versions of the `Quux` method. Their signatures are suitable for the call argument list. Thus, they will be called; the overloaded `Quux` method of the base class will be ignored.

Links

- "Overloading" in "C# in Depth"

3. Problem "Oop:OverloadResolutionInheritance"

Answer

```
Bar.Quux(),
Bar.Quux(params object[])
Bar.Quux(),  Baz
```

Explanation

A schematic calls tree of the code:

```
Main();
  new Baz(); // Baz.ctor
      base.ctor(); // Bar.ctor
        base.ctor(); // Foo.ctor
          Quux(); // Bar.Quux() beacuse there is an overload
            Console.WriteLine("Bar.Quux(),  " + name);
            // name doesn't have value
        name = "Bar";
    name = "Baz";
    Quux();
      // Bar.Quux(params object[] args) because Bar have a suitable method
      Console.WriteLine("Bar.Quux(params object[])");
    ((Foo)this).Quux(); // Bar.Quux() beacuse there is an overload
      Console.WriteLine("Bar.Quux(),  " + name); // name == "Baz"
```

Links

- The "OverloadResolutionOverride" problem

4. Problem "Oop:InheritanceNestedClass"

Answer

```
Foo.Quux()
```

Explanation

The `Bar.Quux` class is declared in `private` scope, you can't use it from a child class. Thus, the call of the `Quux` method from the `Baz` class will be used the `Foo.Quux` class.

8 LINQ (Solutions)

1. Problem "Linq:EnumerableToArray"

Answer

```
GetString: Foo
EnumerableToArray: Foo
GetString: Bar
EnumerableToArray: Bar
GetString: Foo
GetString: Bar
```

Explanation

LINQ queries use deferred/lazy execution. It means, a LINQ query without a cast method like ToArray() or ToList() is not executed immediately. The execution will be deferred until we do not explicitly require the results. Thus, the line

```
var strings = GetStringEnumerable();
```

will not print anything to the console. Next, in the loop

```
foreach (var s in strings)
  Console.WriteLine("EnumerableToArray: " + s);
```

query execution will be performed. Moreover, first will be evaluated the first `yield` (print `GetString: Foo`), next the loop body will be evaluated for the first enumerable element (print `EnumerableToArray: Foo`). Next, the `foreach` loop will require the second enumerable element, the second `yield` will be evaluated (print `GetString: Bar`), the loop body will be evaluated for the second element (print `EnumerableToArray: Bar`).

Next, the following line will be executed:

```
return strings.ToArray();
```

Our LINQ query will be performed again. So, the lines `GetString: Foo` and `GetString: Bar` will be printed again.

2. Problem "Linq:LifeAfterYield"

Answer

```
BarBaz
```

Explanation

The code from the problem will be transformed in the following code (some part of the code specifically removed for better understanding):

```
private sealed class FooEnumerable :
    IEnumerable<string>, IEnumerator<string>
{
  private int state;
  public string Current { get; private set; }

  object IEnumerator.Current
  {
    get { return Current; }
  }

  public FooEnumerable(int state)
  {
    this.state = state;
  }
```

```
  public IEnumerator<string> GetEnumerator()
  {
    FooEnumerable fooEnumerable;
    if (state == -2)
    {
      state = 0;
      fooEnumerable = this;
    }
    else
      fooEnumerable = new FooEnumerable(0);
    return fooEnumerable;
  }

  IEnumerator IEnumerable.GetEnumerator()
  {
    return GetEnumerator();
  }

  bool IEnumerator.MoveNext()
  {
    switch (state)
    {
      case 0:
        Current = "Bar";
        state = 1;
        return true;
      case 1:
        state = -1;
        Console.WriteLine("Baz");
        break;
    }
    return false;
  }

  void IEnumerator.Reset()
  {
    throw new NotSupportedException();
  }

  void IDisposable.Dispose()
  {
  }
}

IEnumerable<string> Foo()
{
  return new FooEnumerable(-2);
}
```

```
void Main()
{
  var enumerator = Foo().GetEnumerator();
  while (enumerator.MoveNext())
    Console.Write(enumerator.Current);
}
```

3. Problem "Linq:ClosureAndForeach"

Answer

In old compiler versions: `3 3 3`.

In new compiler versions: `1 2 3`.

Explanation

In old compiler versions, the code from the problem will be transformed in the following code:

```
public void Run()
{
  var actions = new List<Action>();
  DisplayClass c1 = new DisplayClass();
  foreach (int i in Enumerable.Range(1, 3))
  {
    c1.i = i;
    list.Add(c1.Action);
  }
  foreach (Action action in list)
    action();
}

private sealed class DisplayClass
{
  public int i;

  public void Action()
  {
    Console.WriteLine(i);
  }
}
```

Thus, all elements from the list refer to same delegate. So, we will see 3 same values in the console, they will equal the last value of `i`.

Some breaking changes were in new compiler versions. The new version of the code:

```
public void Run()
{
  var actions = new List<Action>();
```

```
  foreach (int i in Enumerable.Range(1, 3))
  {
    DisplayClass c1 = new DisplayClass();
    c1.i = i;
    list.Add(c1.Action);
  }
  foreach (Action action in list)
    action();
}

private sealed class DisplayClass
{
  public int i;

  public void Action()
  {
    Console.WriteLine(i);
  }
}
```

Now, each element of the list refers to own delegate, all printed values will be different.

Examples:

```
Mono compiler 2.4.4                       : 3 3 3
Mono compiler 3.10.0                      : 1 2 3
Mono compiler 3.10.0        langversion=4 : 1 2 3
MS compiler 3.5.30729.7903                : 3 3 3
MS compiler 4.0.30319.1                   : 3 3 3
MS compiler 4.0.30319.33440               : 1 2 3
MS compiler 4.0.30319.33440 langversion=4 : 1 2 3
```

Links

- ItDepends.NET: ClosureAndForeach
- Eric Lippert: Closing over the loop variable considered harmful
- Eric Lippert: Closing over the loop variable, part two

4. Problem "Linq:QueryAfterRemove"

Answer

```
1
```

Explanation

When you call `list.Where(c => c.StartsWith("B"))`, query will only built, but not executed. The actual execution begins at the moment of the call `query.Count()`. By this time, the value of `list` will be `{ "Foo", "Baz" }`, and hence, there exists only one element that starts with the `'B'` letter.

5. Problem “Linq:ClosureAndVariable”

Answer

```
2
```

Explanation

The code is translated into the following form:

```
class DisplayClass
{
  public string startLetter;

  public bool Method1(string c)
  {
    return c.StartsWith(this.startLetter);
  }

  public bool Method2(string c)
  {
    return c.StartsWith(this.startLetter);
  }
}

void Main()
{
  DisplayClass displayClass = new DisplayClass();
  var list1 = new List<string> { "Foo", "Bar", "Baz" };
  var list2 = list1;
  displayClass.startLetter = "F";
  IEnumerable<string> source = list2.Where(displayClass.Method1);
  displayClass.startLetter = "B";
  Console.WriteLine(source.Where(displayClass.Method2).Count());
}
```

The execution of the LINQ queries will start only in the last line of code. As can be seen, the same helper class creates for both queries. First, the `list2.Where(displayClass.Method1)` query will be executed. It returns `{ "Bar", "Baz" }` because `displayClass.startLetter` at the time of execution is `"B"`. Next, the `source.Where(displayClass.Method2)` query will be executed. It also returns `{ "Bar", "Baz" }`. The number of elements in the result is two.

6. Problem “Linq:QueryWithInc”

Answer

```
Inc: 0
Inc: 1
Number: 2
```

```
Inc: 2
Inc: 3
Number: 4
```

Explanation

An imperative version of the code is as follows:

```
var numbers = new[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
var takenAmount = 0;
for (int i = 0; i < numbers.Length; i++)
{
  var number = numbers[i];
  Console.WriteLine("Inc: " + number);
  var number2 = number + 1;
  if (number2 % 2 == 0)
  {
    Console.WriteLine("Number: " + number2);
    takenAmount++;
    if (takenAmount == 2)
      break;
  }
}
```

7. Problem "Linq:ExceptionYieldYield"

Answer

Call of `evenNumbers.FirstOrDefault()`.

Explanation

The lines

```
var numbers = GetSmallNumbers();
var evenNumbers = numbers.Select(n => n * 2);
```

only build a query, but don't execute it. A logic of the `GetSmallNumbers()` method starts run at the first call of the `MoveNext()` method, which corresponds to the first call of the `MoveNext()` method. At this point, the `Exception` will happen.

8. Problem "Linq:YieldExceptionYield"

Answer

Exception is not going to happen.

Explanation

Indeed, the line

```
var numbers = GetSmallNumbers();
```

is only building a query, but does not execute it. The line

```
var evenNumbers = numbers.Select(n => n * 2);
```

is also only building another query without direct execution. Let's look to the last list of the `Main` method:

```
Console.WriteLine(evenNumbers.FirstOrDefault());
```

This call will get only the first element of the result enumerable (single call of `MoveNext()` and `Current` will be executed). Evaluation of the next elements will nor occur. Thus, the code will work without any exceptions.

9. Problem "Linq:TryYieldFinally"

Answer

```
Foo
1
```

Links

- Eric Lippert: Iterator Blocks, Part Five: Push vs Pull
- Dan Crevier: Yield and usings - your Dispose may not be called!
- StackOverflow: yield return with try catch, how can i solve it
- StackOverflow: Yield return from a try/catch block
- StackOverflow: Finally block may not be called when enumerating over yielding method
- StackOverflow: Yield return from a try/catch block

9 Mathematics (Solutions)

1. Problem "Math:Rounding1"

Answer

```
| Number | Round | Floor | Ceiling | Truncate | Format |
|   -2.9 |    -3 |    -3 |      -2 |       -2 |     -3 |
|   -0.5 |     0 |    -1 |       0 |        0 |     -1 |
|    0.3 |     0 |     0 |       1 |        0 |      0 |
|    1.5 |     2 |     1 |       2 |        1 |      2 |
|    2.5 |     2 |     2 |       3 |        2 |      3 |
|    2.9 |     3 |     2 |       3 |        2 |      3 |
```

Explanation

If the number is exactly halfway between two possibilities, the following rules will work:

- Math.Round rounds to the nearest even integer (by default).
- Math.Floor rounds down towards the negative infinity.
- Math.Ceiling rounds up towards the positive infinity.
- Math.Truncate rounds down or up towards zero.
- String.Format rounds towards the number away from zero.

Links

- Standard Numeric Format Strings
- Custom Numeric Format Strings

2. Problem "Math:Rounding2"

   Answer

```
 a =  b * (a/b) + (a%b)
 7 =  3 *   2   +   1
 7 = -3 *  -2   +   1
-7 =  3 *  -2   +  -1
-7 = -3 *   2   +  -1
```

3. Problem "Math:Eps"

   Answe

```
0.1 + 0.2 != 0.3
```

4. Problem "Math:DivideByZero"

   Answer

```
Infinity
Infinity
DivideByZeroException
```

5. Problem "Math:Overflow"

   Answer

```
Checked   Int32   increased max: OverflowException
Checked   Double  increased max: 1,79769313486232E+308
Checked   Decimal increased max: OverflowException
Unchecked Int32   increased max: -2147483607
Unchecked Double  increased max: 1,79769313486232E+308
Unchecked Decimal increased max: OverflowException
```

Overflow operations with `sbyte`, `byte`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`, `char` throw `OverflowException` depending on `checked`/`unchecked` context (ECMA-334, 11.1.5).

Overflow operations with `float`, `double` never throw `OverflowException` (ECMA-334, 11.1.6).

Overflow operations with `decimal` always throw `OverflowException` (ECMA-334, 11.1.7).

6. Problem "Math:AugmentedAssignment"

   Answer

   ```
   1
   ```

   Explanation

   The construction

   ```
   a += Foo();
   ```

   will be transformed to

   ```
   a = a + Foo();
   ```

10 Value types (Solutions)

1. Problem "ValueTypes:Boxing"

   Answer

   ```
   Foo
   Foo
   Foo
   Bar
   Baz
   Foo
   ```

2. Problem "ValueTypes:MutableProperty"

   Answer

   ```
   0
   ```

3. Problem "ValueTypes:Enumerator"

   Answer

   Loop hangs, zeros will be displayed.

4. Problem "ValueTypes:StructLayout"

   Answer

   ```
   8
   8
   ```

11 Strings (Solutions)

1. Problem "Strings:PlusString"

   Answer

   ```
   3A
   1A2
   A12
   ```

2. Problem "Strings:PlusChar"

   Answer

   ```
   68
   68
   68
   ```

3. Problem "Strings:StringPlusNull"

   Answer

   ```
   False
   ```

   Explanation

   A text fragment from C# Language Specification, the "7.8.4 Addition operator" section:

   "String concatenation:

   ```
   string operator +(string x, string y);
   string operator +(string x, object y);
   string operator +(object x, string y);
   ```

   These overloads of the binary + operator perform string concatenation. **If an operand of string concatenation is null, an empty string is substituted**. Otherwise, any non-string argument is converted to its string representation by invoking the virtual ToString method inherited from type object. If ToString returns null, an empty string is substituted."

   Thus:

   ```
   ((string)null + null + null) == ("" + null + null) ==
     (("" + null) + null) == (("" + "") + null) == ("" + null) ==
     ("" + "") == ""
   "" != null
   ```

   Links

   - Microsoft Reference Source

- Mono 3.10.0 Source

4. Problem “Strings:CaseInComparison”

   Answer

   No.

   Explanation

   It depends from current CultureInfo. An example:

```
var a = "i";
var b = "I";
Thread.CurrentThread.CurrentCulture =
  CultureInfo.CreateSpecificCulture("tr-TR");
Console.WriteLine(
  string.Compare(a.ToUpper(), b)); //'1'
Console.WriteLine(
  string.Compare(a, b, StringComparison.OrdinalIgnoreCase)); //'0'
```

   Links

   - The Turkey Test.

5. Problem “Strings:CorruptedNumber”

   Answer

   You can use custom `CultureInfo`:

```
var culture = (CultureInfo) CultureInfo.InvariantCulture.Clone();
culture.NumberFormat.NegativeSign = "Foo";
Thread.CurrentThread.CurrentCulture = culture;
Console.WriteLine(-42); // Displays "Foo42"
```

6. Problem “Strings:CurrentCulture”

   Answer

   .NET Runtime will be use `CurrentCulture`.

   Explanation

   Thus, the `en-US` locale will be used:

```
12/31/2014 1:01:02 PM
```

   The result in the `ru-RU` locale:

```
31.12.2014 13:01:02
```

7. Problem “Strings:CompareToVsEquals”

   Answer

   Yes.

   Explanation

   `CompareTo` executes according to CultureInfo, `Equals` — without.

   An example: the string `"ß"` (Eszett , ß0df) and `"ss"` will be equal by `CompareTo` in German CultreInfo.

   The == operator works like the `String.Equals` method.

8. Problem “Strings:CorruptedString”

   Answer

   You can use the internment mechanism:

```
var s = "Hello";
string.Intern(s);
unsafe
{
  fixed (char* c = s)
    for (int i = 0; i < s.Length; i++)
      c[i] = 'a';
}
Console.WriteLine("Hello"); // Displays: "aaaaa"
```

9. Problem “Strings:ExplicitlyInternment”

   Answer

```
True
True
False
True
```

10. Problem “Strings:NoStringInterning”

    Answer

    No.

    Links

    - MSDN: CompilationRelaxations Enumeration.

11. Problem “Strings:InternmentAndMetadata”

    Answer

    One time.

12. Problem “Strings:Secure”

    Answer

    You should use `SecureString`.

    Links

    - MSDN: SecureString Class

13. Problem “Strings:StableSorting”

    Answer

    Yes.

    Explanation

    The standard sorting .NET is unstable, which means that if we, say, sort rows insensitive, the strings `"AAA"` and `"aaa"` can accommodate in any order.

14. Problem “Strings:TextElementEnumerator”

    Answer

    Yes.

    Explanation

    FCL supports encoding using more than 16 bits: one text element can be determined by several `char`-symbols.

12 Multithreading (Solutions)

1. Problem “Multithreading:ThreadStaticVariable”

    Answer

    ```
    0
    ```

    Links

    - MSDN: ThreadStaticAttribute
    - StackOverflow: How does the ThreadStatic attribute work?

2. Problem “Multithreading:LockSlimInFinalizer”

    Answer

    ```
    ~Foo: start
    Exception: SynchronizationLockException
    ~Foo: finish
    ```

    Explanation

    A SynchronizationLockException will be thrown because the finalizer will be called from the GC thread.

Bibliography

Some useful books about .NET:

1. Standard ECMA-335: Common Language Infrastructure (CLI)
2. Standard ECMA-334: C# Language Specification
3. «CLR via C#» by Jeffrey Richter
4. «C# in Depth» by Jon Skeet
5. «Pro C# 2010 and the .NET 4 Platform» by Andrew Troelsen
6. «C# 4.0: The Complete Reference» by Herbert Schildt
7. «C# Unleashed» by Joseph Mayo
8. «Essential C# 5.0» by Mark Michaelis, Eric Lippert
9. «Pro .Net Performance» by Sasha Goldshtein, Dima Zurbalev, Ido Flatow
10. «Under the Hood of .NET Memory Management» by Chris Farrell, Nick Harrison
11. «Effective C#: 50 Specific Ways to Improve Your C#» by Bill Wagner
12. «Expert .NET 2.0 IL Assembler» by by Serge Lidin
13. «Advanced .NET Debugging» by Mario Hewardt, Patrick Dessud
14. «Essential .Net Volume 1: The Common Language Runtime» by Don Box