```
LinksPlatform's Platform.Numbers Class Library
./Arithmetic.cs
    #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
   namespace Platform. Numbers
        public static class Arithmetic
5
6
             public static T Add<T>(T x, T y) => Arithmetic<T>.Add(x, y);
public static T And<T>(T x, T y) => Arithmetic<T>.And(x, y);
public static T Increment<T>(T x) => Arithmetic<T>.Increment(x);
             public static T Subtract<T>(T x, T y) => Arithmetic<T>.Subtract(x, y);
             public static T Subtract<T>(Integer<T> x, Integer<T> y) => Arithmetic<T>.Subtract(x, y);
11
             public static T Decrement<T>(T x) => Arithmetic<T>.Decrement(x);
12
13
        }
14
    }
./ArithmeticExtensions.cs
    #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
    namespace Platform. Numbers
3
        public static class ArithmeticExtensions
5
6
             public static T Decrement<T>(this ref T x) where T : struct => x =
              → Arithmetic<T>.Decrement(x);
             public static T Increment<T>(this ref T x) where T : struct => x =
              → Arithmetic<T>.Increment(x);
        }
9
    }
10
/Arithmetic[T].cs
   using System;
   using Platform.Exceptions; using Platform.Reflection;
3
    using Platform.Reflection.Sigil;
    // ReSharper disable StaticFieldInGenericType
    #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
    namespace Platform. Numbers
    ₹
10
        public static class Arithmetic<T>
11
12
                                                 T, T> Add;
             public static readonly Func<T,</pre>
13
             public static readonly Func<T, T,</pre>
                                                     T> And;
14
             public static readonly Func<T, T> Increment;
public static readonly Func<T, T, T> Subtrac
public static readonly Func<T, T> Decrement;
15
16
                                                     T> Subtract;
17
18
             static Arithmetic()
19
                  Add = DelegateHelpers.Compile<Func<T, T, T>>(emiter =>
21
                  {
22
                       Ensure.Always.IsNumeric<T>();
                       emiter.LoadArguments(0, 1);
                       emiter.Add();
25
                       emiter.Return();
26
                  });
27
                  And = DelegateHelpers.Compile<Func<T, T, T>>(emiter =>
2.8
29
                       Ensure.Always.IsNumeric<T>();
                       emiter.LoadArguments(0, 1);
                       emiter.And();
32
                       emiter.Return();
33
                  });
                  Increment = DelegateHelpers.Compile<Func<T, T>>(emiter =>
35
36
                       Ensure.Always.IsNumeric<T>();
                       emiter.LoadArgument(0);
                       emiter.Increment(typeof(T));
39
                       emiter.Return();
40
                  });
                  Subtract = DelegateHelpers.Compile<Func<T, T, T>>(emiter =>
42
43
                       Ensure.Always.IsNumeric<T>();
44
                       emiter.LoadArguments(0, 1);
                       emiter.Subtract();
46
                       emiter.Return();
```

```
48
                Decrement = DelegateHelpers.Compile<Func<T, T>>(emiter =>
50
                     Ensure.Always.IsNumeric<T>();
51
                     emiter.LoadArgument(0);
                     emiter.Decrement(typeof(T));
53
                     emiter.Return();
54
                });
55
            }
        }
57
58
./Bit.cs
   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
   namespace Platform. Numbers
3
4
        public static class Bit
5
6
            public static long Count(long x)
                long n = 0;
                while (x != 0)
10
11
                    n++;
                    x &= x - 1;
13
                return n;
15
            }
16
17
            public static int GetLowestPosition(ulong value)
18
19
                if (value == 0)
20
                {
21
                    return -1;
22
23
                var position = 0;
24
                while ((value & 1UL) == 0)
25
26
                     value >>= 1;
27
                     ++position;
28
29
                return position;
            }
31
32
            public static T PartialWrite<T>(T target, T source, int shift, int limit) =>
33
               Bit<T>.PartialWrite(target, source, shift, limit);
34
            public static T PartialRead<T>(T target, int shift, int limit) =>
            → Bit<T>.PartialRead(target, shift, limit);
        }
   }
37
./BitExtensions.cs
   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
   namespace Platform.Numbers
3
4
        public static class BitwiseExtensions
5
6
            public static T PartialWrite<T>(this ref T target, T source, int shift, int limit) where
            T: struct => target = Bit<T>.PartialWrite(target, source, shift, limit);
            public static T PartialRead<T>(this T target, int shift, int limit) =>
             → Bit<T>.PartialRead(target, shift, limit);
        }
   }
10
/Bit[T].cs
   using System;
   using Platform.Exceptions; using Platform.Reflection;
   using Platform.Reflection.Sigil;
4
   // ReSharper disable StaticFieldInGenericType
   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
   namespace Platform. Numbers
10
   {
        public static class Bit<T>
```

```
public static readonly Func<T, T, int, int, T> PartialWrite;
public static readonly Func<T, int, int, T> PartialRead;
static Bit()
    PartialWrite = DelegateHelpers.Compile<Func<T, T, int, int, T>>(emiter =>
    {
        Ensure.Always.IsNumeric<T>();
        var constants = GetConstants();
        var bitsNumber = constants.Item1;
        var numberFilledWithOnes = constants.Item2;
        ushort shiftArgument = 2;
        ushort limitArgument = 3;
var checkLimit = emiter.DefineLabel();
        var calculateSourceMask = emiter.DefineLabel();
        // Check shift
        emiter.LoadArgument(shiftArgument);
        emiter.LoadConstant(0)
        emiter.BranchIfGreaterOrEqual(checkLimit); // Skip fix
        // Fix shift
        emiter.LoadConstant(bitsNumber);
        emiter.LoadArgument(shiftArgument);
        emiter.Add():
        emiter.StoreArgument(shiftArgument);
        emiter.MarkLabel(checkLimit);
        // Check limit
        emiter.LoadArgument(limitArgument);
        emiter.LoadConstant(0);
        emiter.BranchIfGreaterOrEqual(calculateSourceMask); // Skip fix
        // Fix limit
        emiter.LoadConstant(bitsNumber);
        emiter.LoadArgument(limitArgument);
        emiter.Add();
        emiter.StoreArgument(limitArgument);
        emiter.MarkLabel(calculateSourceMask);
        using (var sourceMask = emiter.DeclareLocal<T>())
        using (var targetMask = emiter.DeclareLocal<T>())
            emiter.LoadConstant(typeof(T), numberFilledWithOnes);
            emiter.LoadArgument(limitArgument);
            emiter.ShiftLeft();
            emiter.Not();
            emiter.LoadConstant(typeof(T), numberFilledWithOnes);
            emiter.And();
            emiter.StoreLocal(sourceMask);
            emiter.LoadLocal(sourceMask);
            emiter.LoadArgument(shiftArgument);
            emiter.ShiftLeft();
            emiter.Not();
            emiter.StoreLocal(targetMask);
            emiter.LoadArgument(0); // target
            emiter.LoadLocal(targetMask);
            emiter.And();
            emiter.LoadArgument(1); // source
            emiter.LoadLocal(sourceMask);
            emiter.And();
            emiter.LoadArgument(shiftArgument);
            emiter.ShiftLeft();
            emiter.Or();
        emiter.Return();
    });
    PartialRead = DelegateHelpers.Compile<Func<T, int, int, T>>(emiter =>
        Ensure.Always.IsNumeric<T>();
        var constants = GetConstants();
        var bitsNumber = constants.Item1;
        var numberFilledWithOnes = constants.Item2;
        ushort shiftArgument = 1;
        ushort limitArgument = 2;
        var checkLimit = emiter.DefineLabel();
        var calculateSourceMask = emiter.DefineLabel();
        // Check shift
        emiter.LoadArgument(shiftArgument);
        emiter.LoadConstant(0);
        emiter.BranchIfGreaterOrEqual(checkLimit); // Skip fix
        // Fix shift
```

12

13

15

16 17

18

19

21

23

25 26

27

28

31

32

34

35

37 38

39

40

41

42

44

45

46 47

48

49

51

52

53

55

56

5.9

60

62

63

65

66

67

69

70

72

73

75 76

77

79

80

81

82

83

85

86

89

```
emiter.LoadConstant(bitsNumber);
                     emiter.LoadArgument(shiftArgument);
                     emiter.Add();
92
                     emiter.StoreArgument(shiftArgument);
93
                     emiter.MarkLabel(checkLimit);
                     // Check limit
95
                     emiter.LoadArgument(limitArgument);
96
                     emiter.LoadConstant(0)
97
                     emiter.BranchIfGreaterOrEqual(calculateSourceMask); // Skip fix
                     // Fix limit
99
                     emiter.LoadConstant(bitsNumber);
100
                     emiter.LoadArgument(limitArgument);
101
                     emiter.Add();
102
                     emiter.StoreArgument(limitArgument);
103
                     emiter.MarkLabel(calculateSourceMask);
104
                            (var sourceMask = emiter.DeclareLocal<T>())
                     using (var targetMask = emiter.DeclareLocal<T>())
106
107
                          emiter.LoadConstant(typeof(T), numberFilledWithOnes);
108
                          emiter.LoadArgument(limitArgument); // limit
109
                          emiter.ShiftLeft();
110
                          emiter.Not();
111
                          emiter.LoadConstant(typeof(T), numberFilledWithOnes);
112
                          emiter.And():
113
                          emiter.StoreLocal(sourceMask);
114
                          emiter.LoadLocal(sourceMask);
                          emiter.LoadArgument(shiftArgument);
116
                          emiter.ShiftLeft();
117
                          emiter.StoreLocal(targetMask);
118
                          emiter.LoadArgument(0); // target
                          emiter.LoadLocal(targetMask);
120
                          emiter.And();
121
                          emiter.LoadArgument(shiftArgument);
122
                          emiter.ShiftRight();
123
124
                     emiter.Return();
125
                 });
126
             }
127
128
            private static Tuple<int, T> GetConstants()
129
130
131
                 var type = typeof(T);
                 if (type == typeof(ulong))
132
                 {
133
                     return new Tuple<int, T>(64, (T)(object)ulong.MaxValue);
134
                   (type == typeof(uint))
136
137
                     return new Tuple<int, T>(32, (T)(object)uint.MaxValue);
138
139
                 if (type == typeof(ushort))
140
                 {
141
                     return new Tuple<int, T>(16, (T)(object)ushort.MaxValue);
                 }
143
                 if (type == typeof(byte))
144
145
                     return new Tuple<int, T>(8, (T)(object)byte.MaxValue);
146
147
                 throw new NotSupportedException();
148
             }
149
        }
150
    }
151
./Integer.cs
    using System;
    using Platform.Converters;
    #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
    namespace Platform. Numbers
        public struct Integer : IEquatable<Integer>
 9
            public readonly ulong Value;
10
            public Integer(ulong value) => Value = value;
11
            public static implicit operator Integer(ulong integer) => new Integer(integer);
12
            public static implicit operator Integer(long integer) => To.UInt64(integer);
13
            public static implicit operator Integer(uint integer) => new Integer(integer);
```

```
public static implicit operator Integer(int integer) => To.UInt64(integer);
15
            public static implicit operator Integer(ushort integer) => new Integer(integer);
16
            public static implicit operator Integer(short integer) => To.UInt64(integer)
17
            public static implicit operator Integer(byte integer) => new Integer(integer);
18
            public static implicit operator Integer(sbyte integer) => To.UInt64(integer);
            public static implicit operator Integer(bool integer) => To.UInt64(integer);
20
            public static implicit operator ulong(Integer integer) => integer.Value;
2.1
            public static implicit operator long(Integer integer) => To.Int64(integer.Value);
22
            public static implicit operator uint(Integer integer) => To.UInt32(integer.Value);
23
            public static implicit operator int(Integer integer) => To.Int32(integer.Value);
24
            public static implicit operator ushort(Integer integer) => To.UInt16(integer.Value);
25
            public static implicit operator short(Integer integer) => To.Int16(integer.Value);
            public static implicit operator byte(Integer integer) => To.Byte(integer.Value);
27
            public static implicit operator sbyte(Integer integer) => To.SByte(integer.Value)
28
            public static implicit operator bool(Integer integer) => To.Boolean(integer.Value);
29
            public bool Equals(Integer other) => Value == other.Value;
            public override string ToString() => Value.ToString();
3.1
32
   }
./Integer[T].cs
   using
          System:
   using System Linq;
   using System.Reflection:
   using System.Collections.Generic;
   using Platform. Exceptions;
   using Platform.Reflection;
using Platform.Reflection.Sigil;
   using Platform.Converters;
   // ReSharper disable StaticFieldInGenericType
10
   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12
   namespace Platform.Numbers
14
   {
        public struct Integer<T> : IEquatable<Integer<T>>
15
16
            private static readonly EqualityComparer<T> _equalityComparer =
17

→ EqualityComparer<T>.Default;

            private static readonly Func<ulong, Integer<T>> _create;
18
19
            public static readonly T Zero;
20
                                      One:
21
            public static readonly
            public static readonly T Two;
22
23
            public readonly T Value;
24
26
            static Integer()
27
                _create = DelegateHelpers.Compile<Func<ulong, Integer<T>>>(emiter => {
28
29
                    if (typeof(T) != typeof(Integer))
30
                    {
31
                        Ensure.Always.CanBeNumeric<T>();
32
33
                    emiter.LoadArgument(0);
34
                    if (typeof(T) != typeof(ulong) && typeof(T) != typeof(Integer))
35
                         emiter.Call(typeof(To).GetTypeInfo().GetMethod(typeof(T).Name,
37
                         → Types<ulong>.Array.ToArray()));
38
                    if (CachedTypeInfo<T>.IsNullable)
39
40
                         emiter.NewObject(typeof(T), CachedTypeInfo<T>.UnderlyingType);
41
                       (typeof(T) == typeof(Integer))
44
                         emiter.NewObject(typeof(Integer), typeof(ulong));
45
46
                    emiter.NewObject(typeof(Integer<T>), typeof(T));
47
                    emiter.Return();
48
                });
49
50
                try
5.1
                    Zero = default;
52
                    One = Arithmetic.Increment(Zero);
53
                    Two = Arithmetic.Increment(One);
                catch (Exception exception)
56
```

```
exception. Ignore();
58
                }
            }
60
            public Integer(T value) => Value = value;
62
63
            public static implicit operator Integer(Integer<T> integer)
64
65
                if (typeof(T) == typeof(Integer))
66
                    return (Integer)(object)integer.Value;
68
69
70
                return Convert.ToUInt64(integer.Value);
            }
71
72
            public static implicit operator ulong(Integer<T> integer) => ((Integer)integer).Value;
73
74
            public static implicit operator T(Integer<T> integer) => integer.Value;
75
76
            public static implicit operator Integer<T>(T integer) => new Integer<T>(integer);
77
78
            public static implicit operator Integer<T>(ulong integer) => _create(integer);
79
80
            public static implicit operator Integer<T>(Integer integer) => _create(integer.Value);
81
82
            public static implicit operator Integer<T>(long integer) => To.UInt64(integer);
83
84
            public static implicit operator Integer<T>(uint integer) => new Integer(integer);
85
            public static implicit operator Integer<T>(int integer) => To.UInt64(integer);
87
88
            public static implicit operator Integer<T>(ushort integer) => new Integer(integer);
89
90
            public static implicit operator Integer<T>(short integer) => To.UInt64(integer);
91
92
            public static implicit operator Integer<T>(byte integer) => new Integer(integer);
93
94
            public static implicit operator Integer<T>(sbyte integer) => To.UInt64(integer);
96
            public static implicit operator Integer<T>(bool integer) => To.UInt64(integer);
98
            public static implicit operator long(Integer<T> integer) => To.Int64(integer);
99
100
            public static implicit operator uint(Integer<T> integer) => To.UInt32(integer);
101
102
            public static implicit operator int(Integer<T> integer) => To.Int32(integer);
103
104
            public static implicit operator ushort(Integer<T> integer) => To.UInt16(integer);
105
106
            public static implicit operator short(Integer<T> integer) => To.Int16(integer);
107
108
            public static implicit operator byte(Integer<T> integer) => To.Byte(integer);
109
110
            public static implicit operator sbyte(Integer<T> integer) => To.SByte(integer);
111
112
            public static implicit operator bool(Integer<T> integer) => To.Boolean(integer);
114
            public bool Equals(Integer<T> other) => _equalityComparer.Equals(Value, other.Value);
115
116
            public override string ToString() => Value.ToString();
117
        }
118
    }
119
./Math.cs
    #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
 2
    namespace Platform. Numbers
 3
 4
        /// <remarks>
 5
        /// Resizable array (FileMappedMemory) for values cache may be used. or cached oeis.org
        /// </remarks>
        public static class Math
 9
            /// <remarks>
10
            /// Source: https://oeis.org/A000142/list
11
            /// </remarks>
12
            private static readonly ulong[] _factorials =
13
14
                         6, 24, 120, 720, 5040, 40320, 362880, 3628800, 39916800
15
                479001600, 6227020800, 87178291200, 1307674368000, 20922789888000,
```

```
355687428096000, 6402373705728000, 121645100408832000, 2432902008176640000
17
             };
18
19
             /// <remarks>
             /// Source: https://oeis.org/A000108/list
21
             /// </remarks>
22
             private static readonly ulong[] _catalans =
23
24
                         2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 0, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 20420, 24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452
25
                  742900.
26
                  6564120420,
27
                  18367353072152, 69533550916004, 263747951750360, 1002242216651368, 3814986502092304
28
             };
29
30
             public static double Factorial(double n)
31
32
                  if (n <= 1)
33
                  {
34
                      return 1;
35
36
                  if (n < _factorials.Length)</pre>
37
                      return _factorials[(int)n];
39
40
                  return n * Factorial(n - 1);
41
             }
42
43
             public static double Catalan(double n)
44
45
                  if (n <= 1)
46
                  {
                      return 1:
48
49
                  if (n < _catalans.Length)</pre>
50
                  {
51
                      return _catalans[(int)n];
52
                  return Factorial(2 * n) / (Factorial(n + 1) * Factorial(n));
54
55
56
             public static bool IsPowerOfTwo(ulong x) => (x & x - 1) == 0;
57
             public static T Abs<T>(T x) => Math<T>.Abs(x);
59
60
             public static T Negate<T>(T x) => Math<T>.Negate(x);
61
62
    }
./MathExtensions.cs
   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
   namespace Platform. Numbers
3
4
        public static class MathExtensions
6
             public static T Abs<T>(this ref T x) where T : struct => x = Math<T>.Abs(x);
             public static T Negate<T>(this ref T x) where T : struct => x = Math<T>.Negate(x);
9
    }
10
/Math[T].cs
   using System;
using System.Reflection;
    using Platform. Exceptions;
    using Platform.Reflection;
    using Platform.Reflection.Sigil;
    // ReSharper disable StaticFieldInGenericType
    #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
    namespace Platform.Numbers
10
11
        public static class Math<T>
12
13
             public static readonly Func<T, T> Abs;
public static readonly Func<T, T> Negate;
14
15
16
             static Math()
18
                  Abs = DelegateHelpers.Compile<Func<T, T>>(emiter =>
19
```

```
{
20
                     Ensure.Always.IsNumeric<T>();
^{21}
                     emiter.LoadArgument(0);
^{22}
                     if (CachedTypeInfo<T>.IsSigned)
23
                          emiter.Call(typeof(System.Math).GetTypeInfo().GetMethod("Abs", new[] {
^{25}

    typeof(T) }));
^{26}
                     emiter.Return();
27
                 });
                 Negate = DelegateHelpers.Compile<Func<T, T>>(emiter =>
29
30
                     Ensure.Always.IsSigned<T>();
31
                     emiter.LoadArgument(0);
^{32}
                     emiter.Negate();
33
                     emiter.Return();
34
                 });
35
            }
36
        }
37
   }
38
```

Index ./Arithmetic.cs, 1 ./ArithmeticExtensions.cs, 1 ./Arithmetic[T].cs, 1 ./Bit.cs, 2 ./BitExtensions.cs, 2 ./Bit[T].cs, 2 ./Integer.cs, 4 ./Integer[T].cs, 5 ./Math.cs, 6 ./MathExtensions.cs, 7 ./Math[T].cs, 7