```
// Example 3 (file example3/Lib.cs) from C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Compile with:
// csc /target:module Mod.cs
// csc /target:library Lib.cs
// csc /addmodule:Mod.netmodule /reference:Lib.dll Prog.cs

using System;

public class LibClass {
  public static void Hello(Object name) {
    Console.WriteLine("Hello to " + name + " from Lib");
  }
}
```

```
// Example 3 (file example3/Mod.cs) from C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Compile with:
// csc /target:module Mod.cs
// csc /target:library Lib.cs
// csc /addmodule:Mod.netmodule /reference:Lib.dll Prog.cs

using System;

class ModClass {
  public static void Hello(String name) {
    Console.WriteLine("Hello to " + name + " from Mod");
  }
}
```

```
// Example 3 (file example3/Prog.cs) from C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Compile with:
// csc /target:module Mod.cs
// csc /target:library Lib.cs
// csc /addmodule:Mod.netmodule /reference:Lib.dll Prog.cs

using System;

class Class {
  public static void Main(String[] args) {
    if (args.Length != 1)
      Console.WriteLine("Usage: Prog <yourname>");
    else {
      Console.WriteLine("Hello," + args[0]);
      ModClass.Hello(args[0]);
      LibClass.Hello(args[0]);
    }
  }
}
```

```
// Example 1 from page 3 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class Sum {
  static void Main(String[] args) {
    int sum = 0;
    for (int i=0; i<args.Length; i++)
      sum += int.Parse(args[i]);
    Console.WriteLine("The sum is " + sum);
  }
}
```

```
// Example 4 from page 5 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class School {
  const int @class = 2004;
  const bool @public = true;
  String @delegate = "J. Smith ";

  public static int @double(int i) {
    return 2 * @i;
  }

  public static void Main(String[] args) {
    School school = new School();
    Console.WriteLine(school.@delegate.Trim() + " " + School.@class);
  }
}
```

```
// Example 5 from page 5 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

class Comment {
  // This is a one-line comment; it extends to the end of the line
  /* This is a delimited comment,
     extending over several lines
  */
  int /* This delimited comment extends over part of a line */ x = 117;
}
```

```
// Example 6 from page 5 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

class LayoutExample {                    // Class declaration
  int j;

  LayoutExample(int j) {
    this.j = j;                          // One-line body
  }

  int Sum(int b) {                       // Multi-line body
    if (j > 0) {                         // If statement
      return j + b;                      // Single statement
    } else if (j < 0) {                  // Nested if-else, block statement
      int res = -j + b;
      return res * 117;
    } else { // j == 0                   // Terminal else, block statement
      int sum = 0;
      for (int i=0; i<10; i++)           // For loop
        sum += (b - i) * (b - i);
      return sum;
    }
  }

  static void Main() {
  }
}
```

```
// Example 7 from page 7 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static void Main(String[] args) {
    int i1;
    Int32 i2;
    System.Int32 i3;
  }
}

class Dummy {
  public static void Main() {
  }
}
```

```
// Example 8 from page 9 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static void Main(String[] args) {
    Object o1 = new Object(), o2 = new Object(), o3 = o1;
    Console.WriteLine(o1.Equals(o3) + " " + o1.Equals(o2));       // True False
    Console.WriteLine(o1.GetHashCode() == o3.GetHashCode());      // True
    Console.WriteLine(o1.GetHashCode() == o2.GetHashCode());      // Usually False
    Console.WriteLine(o1.GetHashCode() + " " + o2.GetHashCode()); // Usually distinct
    Console.WriteLine(o1.GetType());                              // System.Object
    String s1 = "abc", s2 = "ABC", s3 = s1 + "";
    Console.WriteLine(s1.Equals(s3) + " " + s1.Equals(s2));       // True False
    Console.WriteLine(s1.GetHashCode() == s3.GetHashCode());      // True
    Console.WriteLine(s1.GetHashCode() == s2.GetHashCode());      // Usually False
    Console.WriteLine(s1.GetHashCode() + " " + s2.GetHashCode()); // Usually distinct
    Console.WriteLine(s1.GetType());                              // System.String
    Console.WriteLine(117.GetHashCode());                         // 117
    Console.WriteLine(5.GetType());                               // System.Int32
    Console.WriteLine(5.0.GetType());                             // System.Double
    int[] ia1 = { 7, 9, 13 }, ia2 = { 7, 9, 13 };
    Console.WriteLine(ia1.GetType());                             // System.Int32[]
    Console.WriteLine(ia1.Equals(ia2));                           // False
    Console.WriteLine(Object.ReferenceEquals(ia1,ia2));           // False
    Console.WriteLine(ia1.GetHashCode() == ia2.GetHashCode());    // Usually False
    int[,] ia3 = new int[6,7];
    Console.WriteLine(ia3.GetType());                             // System.Int32[,]
    int[][] ia4 = new int[6][];
    Console.WriteLine(ia4.GetType());                             // System.Int32[][]
  }
}
```

```
// Example 9 from page 11 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static void Main(String[] args) {
    double d = 2.9;
    Console.WriteLine((int)d);                    // ET double-->int; prints 2
    Console.WriteLine((int)(-d));                 // ET double-->int; prints -2
    uint seconds = (uint)(24 * 60 * 60);          // EB int-->uint
    double avgSecPerYear = 365.25 * seconds;      // I  uint-->double
    float f = seconds;                            // IL uint-->float
    long nationalDebt1 = 14349539503882;
    double perSecond = 45138.89;
    decimal perDay =                              // ED double-->decimal
      seconds * (decimal)perSecond;               // I  uint-->decimal
    double nd2 = nationalDebt1 + (double)perDay;  // ER decimal-->double
    long nd3 = (long)nd2;                         // ET double-->long
    float nd4 = (float)nd2;                       // ER double-->float
    Console.WriteLine(nationalDebt1);
    Console.WriteLine(nd2);
    Console.WriteLine(nd3);
    Console.WriteLine(nd4);
  }
}
```

```
// Example 10 from page 13 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

interface I1 { }
interface I2 : I1 { }
interface J { }
class B : I2 { }
class C : B, J { }
delegate void D(String s);

class MyTest {
  public static void Main(String[] args) {
    Object b1 = new B();            // Implicit B-->Object
    I2 b2 = new B();                // Implicit B-->I2
    B c1 = new C();                 // Implicit C-->B
    I1 b3 = b2;                     // Implicit I2-->B
    I1[] i2a1 = new I2[5];          // Implicit I2[]-->I1[]
    Array inta1 = new int[5];       // Implicit int[]-->Array
    Delegate d1 = new D(Print);     // Implicit D-->Delegate
    C n = null;                     // Implicit null type-->C

    B b4 = (B)b1;                   // Explicit Object-->B
    C c2 = (C)c1;                   // Explicit B-->C
    J b5 = (J)c1;                   // Explicit C-->J
    B b6 = (B)b2;                   // Explicit I2-->B
    I1 i2 = (I1)b2;                 // Explicit I2-->I1
    I2[] i2a2 = (I2[])i2a1;         // Explicit I1[]-->I2[]
    int[] inta2 = (int[])inta1;     // Explicit Array-->int[]
    D d2 = (D)d1;                   // Explicit Delegate-->D
  }

  public static void Print(String s) {
    Console.WriteLine(s);
    return;
  }
}
```

```
// Example 11 from page 13 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

interface I { void Print(); }
struct S : I {
  public int i;
  public S(int i) { this.i = i; }
  public void Print() { Console.WriteLine(i); }
}

class MyTest {
  public static void Main(String[] args) {
    int i = 7;
    Object o = i;                       // Implicit boxing int-->Object
    int j = 5 + (int)o;                 // Explicit unboxing Object-->int
    Console.WriteLine("o {0}", o);                          // 12
    Console.WriteLine("o is int:    {0}", o is int);     // True
    Console.WriteLine("o is double: {0}", o is double);  // False
    S s1 = new S(11);
    I s2 = s1;                          // Implicit boxing S-->I
    s1.i = 22;
    s1.Print();                         // 22
    s2.Print();                         // 11
    S s3 = (S)s2;                       // Explicit unboxing I-->S
    s3.Print();                         // 11
  }
}
```

```
// Example 12 from page 15 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class Scope {
  void M1(int x) {    // declaration of parameter x (#1); shadows x (#5)
    x = 7;            // x #1 in scope; legal, but no effect outside M1
  }                   //
  void M3() {         //
    int x;            // declaration of variable x (#2); shadows x (#5)
    x = 7;            // x #2 in scope
  }
  void M4() {         //
    x = 7;            // x #5 in scope
    // int x;         // would be ILLEGAL, giving a new meaning to x
  }
  void M5() {         //
    {                 //
      int x;          // declaration of variable x (#3); shadows x (#5)
      x = 7;          // x #3 in scope
    }                 //
    {                 //
      int x;          // declaration of variable x (#4); shadows x (#5)
      x = 7;          // x #4 in scope
    }                 //
  }
  public int x;       // declaration of field x (#5)

  public static void Main(String[] args) {
    Scope s = new Scope();
    s.x = 88;
    s.M1(8);
    s.M3();
    s.M4();
  }
}
```

```
// Example 13 from page 15 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static void Main(String[] args) {
    int x, y, z;
    if (args.Length == 0)
      x = y = 10;
    else
      x = args.Length;
    Console.WriteLine(x);           // x definitely assigned, y and z not (#1)
    y = x;
    for (int i=0; i<y; i++)         // x and y definitely assigned, z not (#2)
      z = i;
    // Console.WriteLine(z);        // z still not definitely assigned!   (#3)
  }
}
```

```
// Example 14 from page 17 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

class MyTest {

  class Phone {
    public readonly String name;
    public readonly int phone;
    public Phone(String name, int phone) {
      this.name = name;
      this.phone = phone;
    }
  }

  public static void Main(String[] args) {
    var x = 0.0;                          // Inferred type double
    var b = false;                        // Inferred type bool
    var ps = new List<int>();             // Inferred type List<int>
    ps.Add(2); ps.Add(3); ps.Add(5);
    Console.WriteLine(ps.GetType());
    Console.WriteLine(b);

    var d1 = 34;                      // Inferred type int
    int i1 = d1 * 2;                  //
    int i2 = (int)d1 * 2;             // Cast (int)d1 succeeds at compile-time
    // bool b1 = d1;                  // Rejected at compile-time
    // d1 = true;                     // Rejected at compile-time
    var p1 = new Phone("Kasper", 5170);
    String s1 = p1.name;              // Field access checked only at compile-time
    // int n1 = p1.age;               // Field access rejected at compile-time
    var p2 = new { name = "Kasper", phone = 5170 };
    String s2 = p2.name;              // Field access checked only at compile-time
    // int n2 = p2.age;               // Field access rejected at compile-time

    foreach (var p in ps)
      x += p;
    Console.WriteLine(x);
    for (var etor = ps.GetEnumerator(); etor.MoveNext(); )
      x += etor.Current;
    Console.WriteLine(x);
    using (var etor = ps.GetEnumerator())
      while (etor.MoveNext())
        x += etor.Current;
    Console.WriteLine(x);
  }
}
```

```
// Example 16 from page 17 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

class MyTest {
  private static readonly Random rnd = new Random();

  class Phone {
    public readonly String name;
    public readonly int phone;
    public Phone(String name, int phone) {
      this.name = name;
      this.phone = phone;
    }
  }

  public static void Main(String[] args) {
    // Basic rules of type dynamic
    {
      dynamic d1 = 34;
      int i1 = d1 * 2;                  // OK: cast (int)(d1*2) at run-time
      int i2 = (int)d1 * 2;             // OK: cast (int)d1 at run-time
      // bool b1 = d1;                  // Compiles OK; cast (bool)d1 throws at run-t
ime
      d1 = true;                        // OK
      bool b2 = d1;                     // OK: cast (bool)d1 succeeds at run-time
      dynamic p1 = new Phone("Kasper", 5170);
      String s1 = p1.name;              // Field access checked at run-time
      // int n1 = p1.age;               // Compiles OK; field access throws at run-ti
me
      dynamic p2 = new { name = "Kasper", phone = 5170 };
      String s2 = p2.name;              // Field access checked at run-time
      // int n2 = p2.age;               // Compiles OK; fields access throws at run-t
ime
    }

    // Dynamic operator resolution; run-time type determines meaning of "+" in Plus2(
)
    {
      Console.WriteLine(Plus2(int.MaxValue-1));             // -2147483648, due to i
nt overflow
      Console.WriteLine(Plus2((long)(int.MaxValue-1)));     //  2147483648, no long
overflow
      Console.WriteLine(Plus2(11.5));                       // 13.5
      Console.WriteLine(Plus2("Spar"));                    // Spar2
      // Console.WriteLine(Plus2(false));       // Compiles OK; throws RuntimeBinder
Exception
    }

    // Dynamic receiver; run-time type determines whether to call Length on array or
String
    {
      dynamic v;
      if (args.Length==0)      v = new int[] { 2, 3, 5, 7 };
      else                     v = "abc";
      int res = v.Length;
      Console.WriteLine(res);
    }

    // Dynamic overload resolution; run-time type of v determines which Process calle
d at (**)
    {
      dynamic v;
      if (args.Length==0)      v = 5;
      else if (args[0] == "1") v = "abc";
      else                     v = (Func<int,int>)(x => x*3);
      dynamic r = Process(v);                                    // (**)
      if (args.Length==0 || args[0] == "1")
        Console.WriteLine(r);
      else
        Console.WriteLine(r(11));
      dynamic s = "abc";
      Console.WriteLine(Process(s).StartsWith("abca"));
     }

      // Run-time type tests
    {
```

```
      Console.WriteLine(Square(5));
      Console.WriteLine(Square("abc"));
      Func<int,int> f = x => x*3;
      Console.WriteLine(Square(f)(11));
    }

    {
      // Types dynamic[], List<dynamic>, IEnumerable<dynamic>
      dynamic[] arr = new dynamic[] { 19, "Electric", (Func<int,int>)(n => n+2), 3.2, f
alse };
      int number = arr[0] * 5;
      String street = arr[1].ToUpper();
      int result = arr[2](number);
      Console.WriteLine(number + " " + street);
      double sum = 0;
      List<dynamic> list = new List<dynamic>(arr);
      IEnumerable<dynamic> xs = list;
      foreach (dynamic x in xs)
        if (x is int || x is double)
          sum += x;
      Console.WriteLine(sum);                                    // 22.2
    }

    // Dynamic and anonymous object expressions
    {
      dynamic v = new { x = 34, y = false };
      Console.WriteLine(v.x);
    }
  }

  // Run-time type of v determines meaning of "+": int+, long+, double+, String+, ...
  static dynamic Plus2(dynamic v) { return v + 2; }

  // Ordinary overloaded methods
  static int Process(int v) { return v * v; }

  static double Process(double v) { return v * v; }

  static String Process(String v) { return v + v; }

  static Func<int,int> Process(Func<int,int> v) { return x => v(v(x)); }

  // Explicit tests on run-time type.  Using "dynamic" rather than
  // "Object" here means that the compiler accepts the (v * v)
  // expression and the application of v to arguments, and that the
  // value returned by Square can be further processed: added to,
  // applied to arguments, and so on.
  static dynamic Square(dynamic v) {
    if (v is int || v is double)
      return v * v;
    else if (v is String)
      return v + v;
    else if (v is Func<int,int>)
      return (Func<int,int>)(x => v(v(x)));
    else
      throw new Exception("Don't know how to square " + v);
  }
}

class C : List<dynamic> { }
```

```
// Example 17 from page 19 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class StringEks {
  public static void Main() {
    Console.WriteLine("\u0041BC");              // ABC
    Console.WriteLine(@"\u0041BC");             // \u0041BC
    Console.WriteLine("Say \"Hello\"!");       // Say "Hello"!
    Console.WriteLine(@"Say ""Hello""!");      // Say "Hello"!
    String s1 = @"Line 1
  and Line 2";                                 // Newline allowed only in verbatim string
    String s2 = "Line 1\n  and Line 2";
    Console.WriteLine(s1 == s2);                // True
  }
}
```

```
// Example 18 from page 19 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class Example018 {
  public static void Main(String[] args) {
    if (args.Length != 1)
      Console.WriteLine("Usage: Example018 abc\n");
    else {
      String s1 = "abc", s2 = "ab" + "c", s3 = null;  // Compile-time constants
      String s4 = args[0];                             // Value given at runtime
      // Assume command line argument args[0] is "abc":
      Console.WriteLine(s1==s2);                                  // True
      Console.WriteLine((Object)s1==(Object)s2);                  // Probably True
      Console.WriteLine(s2==s4);                                  // True
      Console.WriteLine((Object)s2==(Object)s4);                  // False
      Console.WriteLine("{0} {1} {2}", s3==s1, s3!=s1, s3==s3); // False True True
      Console.WriteLine("{0} {1}", s1!="", s3!="");              // True True
    }
  }
}
```

```
// Example 19 from page 19 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class Example019 {
  static int eCount(String s) {
    int ecount = 0;
    for (int i=0; i<s.Length; i++)
      if (s[i] == 'e')
        ecount++;
    return ecount;
  }

  public static void Main(String[] args) {
    if (args.Length != 1)
      Console.WriteLine("Usage: Example019 <string>\n");
    else {
      Console.WriteLine("Number of e's is " + eCount(args[0]));
    }
  }
}
```

```
// Example 20 from page 19 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class StringConcatenate {
  public static void Main(String[] args) {
    String res = "";                                    // Inefficient
    for (int i=0; i<args.Length; i++)                   // Inefficient
      res += args[i];                                   // Inefficient
    Console.WriteLine(res);
  }
}
```

```
// Example 21 from page 19 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class StringEks {
  public static void Main() {
    // These statements print 35A and A1025 because (+) is left associative:
    Console.WriteLine(10 + 25 + "A");  // Same as (10 + 25) + "A"
    Console.WriteLine("A" + 10 + 25);  // Same as ("A" + 10) + 25
  }
}
```

```
// Example 22 from page 21 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class StringEks {
  static bool Sorted(String[] a) {
    for (int i=1; i<a.Length; i++)
      if (a[i-1].CompareTo(a[i]) > 0)
        return false;
    return true;
  }
}
```

```
// Example 23 from page 21 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class StringEks {
  public static void Main() {
    String text =
      @"C# is a class-based single-inheritance object-oriented programming
language designed for the Common Language
Runtime of Microsoft's .Net
platform, a managed execution environment with a
typesafe intermediate language and automatic memory management.  Thus
C# is similar to the Java programming language in many respects, but
it is different in almost all details.  In general, C# favors
programmer convenience over language simplicity.  It was designed by
Anders Hejlsberg, Scott Wiltamuth and Peter Golde from Microsoft
Corporation.";
    Console.WriteLine(Readability(text));
  }

  static double Readability(String text) {
    int wordCount = 0, longWordsCount = 0;
    String[] sentences = text.Split(new char[] {'.'});             // Split into sent
ences
    foreach (String sentence in sentences) {
      String[] words = sentence.Split(' ', ',');                   // Split into words
      // String[] words = sentence.Split(new char[] {' ', ','});   // Alternative
      wordCount += words.Length;
      foreach (String word in words) {
        if (word.Length > 6)
          longWordsCount++;
        else if (word.Length == 0)
          wordCount--;
      }
    }
    return (wordCount*1.0)/sentences.Length + (longWordsCount*100.0)/wordCount;
  }
}
```

```
// Example 24 from page 21 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class StringEks {
        public static void Main() {

Point p1 = new Point(10, 20), p2 = new Point(30, 40);
Console.WriteLine("p1 is " + p1);      // Prints: p1 is (10, 20)
Console.WriteLine("p2 is " + p2);      // Prints: p2 is (30, 40)
p2.Move(7, 7);
Console.WriteLine("p2 is " + p2);      // Prints: p2 is (37, 47)

        }
}

class Point {
  protected internal int x, y;

  public Point(int x, int y) { this.x = x; this.y = y; }

  public void Move(int dx, int dy) { x += dx; y += dy; }

  public override String ToString() { return "(" + x + "," + y + ")"; }
}
```

```
// Example 25 from page 23 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Prints:
// |     3D326|    250662|3D326|250662|

using System;

class MyTest {
  public static void Main(String[] args) {
    int i = 250662;
    String s = String.Format("|{0,10:X}|{1,10}|{2:X}|{3}|", i, i, i, i);
    Console.WriteLine(s);
  }
}
```

```
// Example 26 from page 23 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class ArraysExample {
  public static void Main() {
    // Roll a die, count frequencies
    Random rnd = new Random();          // Random number generator
    int[] freq = new int[6];            // All initialized to 0
    for (int i=0; i<1000; i++) {
      int die = rnd.Next(1, 7);         // Random integer in range 1..6
      freq[die-1] += 1;
    }
    for (int c=1; c<=6; c++)
      Console.WriteLine("{0} came up {1} times", c, freq[c-1]);
  }
}
```

```
// Example 27 from page 23 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class StringFormattingAlignment {
  public static void Main() {
    // Fill a 3x5 -matrix with random integers from [0,999]
    Random rnd = new Random();          // Random number generator
    int[,] m = new int[3,5];            // 3x5 matrix
    for (int i=0; i<m.GetLength(0); i++)
      for (int j=0; j<m.GetLength(1); j++)
        m[i,j] = rnd.Next(1000);        // Random integer in range 0..999

    // Print matrix
    for (int i=0; i<m.GetLength(0); i++)
      Console.WriteLine("{0,4} {1,4} {2,4} {3,4} {4,4}", m[i,0], m[i,1], m[i,2], m[i,3], m[i,
4]);
  }
}
```

```
// Example 28 from page 25 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// The output is in LaTeX format.
// Used to generate example ex-number-format-table

using System;
using System.Threading;
using System.Globalization;
using System.IO;

public class DateFormatting {
  public static void Main() {
    String[] formats = {"{0:D4}", "{0,7}", "{0:F0}", "{0:F2}", "{0,8:F3}", "{0:E4}", "{0,9:C}"}
;
    int[] numbers1 = { 0, 1, 145, -1 };
    double[] numbers2 =  { 2.5, -1.5, 330.8, 1234.516 };

    Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");

    StreamWriter fs = File.CreateText(Directory.GetCurrentDirectory().ToString() + @"
\number-formats.txt");

    fs.WriteLine(@"\begin{center}");
    fs.WriteLine(@"\begin{tabular}{r|lllllll}");
    fs.WriteLine(@"\hline\hline");
    fs.WriteLine(@"& \multicolumn{7}{c}{Format Specifications}\\");
    fs.WriteLine(@"\cline{2-8}\cline{2-8}");
    fs.Write(@"Number");

    foreach(String format in formats)
      fs.Write(@" & \verb|" + format + "|");
    fs.WriteLine(@"\\ \hline");

    foreach(int number in numbers1) {
      fs.Write(number);
      foreach(String format in formats)
        fs.Write(@" & \verb|" + String.Format(format, number) + "|");
      fs.WriteLine(@"\\");
    }

    foreach(double number in numbers2) {
      fs.Write(number);
      foreach(String format in formats) {
        try { fs.Write(@" & \verb|" + String.Format(format, number) + "|"); }
        catch (FormatException) { fs.Write(" &"); }
      }
      fs.WriteLine(@"\\");
    }

    fs.WriteLine(@"\hline\hline");
    fs.WriteLine(@"\end{tabular}");
    fs.WriteLine(@"\end{center}");

    fs.Close();
  }
}
```

```
// Example 29 from page 25 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// The output is in LaTeX format.
// Used to generate example ex-number-format-custom.

using System;
using System.Threading;
using System.Globalization;
using System.IO;

public class CustomNumberFormat {
  public static void Main() {
    double[] numbers = { 1230.1, 17, 0.15, 0, -26 };
    String[] formats = {"{0:000.0}", "{0:###.#}", "{0:##0.0}", "{0:#0E+0}", "{0:00##;'(neg)';'-'}"}
;
    int noFormats = formats.Length;

    Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");

    StreamWriter fs = File.CreateText(Directory.GetCurrentDirectory().ToString() + @"
\number-formats-custom.txt");

    fs.WriteLine(@"\begin{center}");
    fs.Write(@"\begin{tabular}{r|");
    for (int i=0; i<noFormats; i++)
      fs.Write("l");
    fs.WriteLine("}");
    fs.WriteLine(@"\hline\hline");
    fs.WriteLine(@"& \multicolumn{" + noFormats + @"}{c}{Format Specifications}\\");
    fs.WriteLine(@"\cline{2-" + (noFormats+1) + @"}\cline{2-" + (noFormats+1) + "}");
    fs.Write(@"Number");

    foreach(String format in formats)
      fs.Write(@" & \verb|" + format + "|");
    fs.WriteLine(@"\\ \hline");

    foreach(double number in numbers) {
      fs.Write(number);
      foreach(String format in formats)
        fs.Write(@" & \verb|" + String.Format(format, number) + "|");
      fs.WriteLine(@"\\");
    }

    fs.WriteLine(@"\hline\hline");
    fs.WriteLine(@"\end{tabular}");
    fs.WriteLine(@"\end{center}");

    fs.Close();
  }
}
```

```
// Example 30 from page 25 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Threading;         // Thread
using System.Globalization;     // CultureInfo

public class Tryformatting {

  public static void Main(String[] args) {
    CultureInfo ci;
    ci = new CultureInfo("en-US");       // USA
    // ci = new CultureInfo("fr-FR");    // France
    // ci = new CultureInfo("de-DE");    // Germany
    // ci = new CultureInfo("da-DK");    // Denmark
    Thread.CurrentThread.CurrentCulture = ci;
    Maketable();
  }

  static void Maketable() {
    DateTime now = DateTime.Now;
    String[] fmts = { "{0:F}", "{0:f}", "{0:G}", "{0:g}", "{0:s}",
                      "{0:u}", "{0:U}", "{0:R}", "{0:D}", "{0:Y}",
                      "{0:M}", "{0:T}", "{0:t}" };

    Console.WriteLine("{0,-12} {1}", "Format code", "Formatted date");
    for (int j=0; j<fmts.Length; j++) {
      Console.Write("{0,-12} ", fmts[j]);
      Console.WriteLine(String.Format(fmts[j], now));
    }
  }
}
```

```
// Example 31 from page 27 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Text;              // StringBuilder

class StringBuilderConcatenate {
  public static void Main(String[] args) {
    StringBuilder res = new StringBuilder();
    for (int i=0; i<args.Length; i++)
      res.Append(args[i]);
    Console.WriteLine(res.ToString());
  }
}
```

```
// Example 32 from page 27 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Text;              // StringBuilder

class Example032 {
  public static void Main(String[] args) {
    if (args.Length != 1)
      Console.WriteLine("Usage: Example032 <length>\n");
    else {
      Console.WriteLine("Timing character replacement in a string:");
      Random rnd = new Random();
      int length = int.Parse(args[0]);
      char[] cbuf = new char[length];
      for (int i=0; i<length; i++)
        cbuf[i] = (char)(65 + rnd.Next(26));
      String s = new String(cbuf);
      for (int i=0; i<10; i++) {
        Timer t = new Timer();
        String res = ReplaceCharString(s, 'A', "HA");
        Console.Write(t.Check() + " ");
      }
      Console.WriteLine();
    }
  }

  static String ReplaceCharString(String s, char c1, String s2) {
    StringBuilder res = new StringBuilder();
    for (int i=0; i<s.Length; i++)
      if (s[i] == c1)
        res.Append(s2);
      else
        res.Append(s[i]);
    return res.ToString();
  }

  private class Timer {
    private DateTime start;

    public Timer() {
      start = DateTime.Now;
    }

    public double Check() {
      TimeSpan dur = DateTime.Now - start;
      return dur.TotalSeconds;
    }
  }
}
```

```
// Example 33 from page 27 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Text;              // StringBuilder

class Example033 {
  public static void Main(String[] args) {
    if (args.Length != 1)
      Console.WriteLine("Usage: Example033 <length>\n");
    else {
      Console.WriteLine("Timing character replacement in a string:");
      Random rnd = new Random();
      int length = int.Parse(args[0]);
      char[] cbuf = new char[length];
      for (int i=0; i<length; i++)
        cbuf[i] = (char)(65 + rnd.Next(26));
      String s = new String(cbuf);
      for (int i=0; i<10; i++) {
        StringBuilder sb = new StringBuilder(s);
        Timer t = new Timer();
        ReplaceCharString(sb, 'A', "HA");
        Console.Write(t.Check() + " ");
      }
      Console.WriteLine();
    }
  }

  // In-place replacement in a StringBuffer; very inefficient and strange

  static void ReplaceCharString(StringBuilder sb, char c1, String s2) {
    int i = 0;                                  // Inefficient
    while (i < sb.Length) {                     // Inefficient
      if (sb[i] == c1) {                        // Inefficient
        sb.Remove(i, 1);                        // Inefficient
        sb.Insert(i, s2);                       // Inefficient
        i += s2.Length;                         // Inefficient
      } else                                    // Inefficient
        i += 1;                                 // Inefficient
    }                                           // Inefficient
  }

  private class Timer {
    private DateTime start;

    public Timer() {
      start = DateTime.Now;
    }

    public double Check() {
      TimeSpan dur = DateTime.Now - start;
      return dur.TotalSeconds;
    }
  }
}
```

```
// Example 34 from page 29 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class ArraysExample {
  public static void Main() {
    // Roll a die, count frequencies
    Random rnd = new Random();                  // Random number generator
    int[] freq = new int[6];                    // All elements initialized to 0
    for (int i=0; i<1000; i++) {
      int die = rnd.Next(1, 7);
      freq[die-1] += 1;
    }
    for (int c=1; c<=6; c++)
      Console.WriteLine(c + " came up " + freq[c-1] + " times");
  }
}
```

```
// Example 35 from page 29 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class ArrayCheckdate {
  public static void Main(String[] args) {
    Console.WriteLine("August 31 is legal: " + CheckDate(8, 31));
    Console.WriteLine("April 31 is legal: " + CheckDate(4, 31));
  }

  static readonly int[] days = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

  static bool CheckDate(int mth, int day)
  { return (mth >= 1) && (mth <= 12) && (day >= 1) && (day <= days[mth-1]); }
}
```

```
// Example 36 from page 29 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static void Main(String[] args) {
    Point[] a = new RedPoint[10];     // Length 10, element type RedPoint
    Point p1 = new Point(42, 117);    // Compile-time type Point, class Point
    RedPoint cp = new RedPoint(3,4);  // Compile-time type RedPoint, class RedPoint
    Point p2 = cp;                    // Compile-time type Point, class RedPoint
    a[0] = cp;                        // OK, RedPoint is subclass of RedPoint
    a[1] = p2;                        // OK, RedPoint is subclass of RedPoint
    a[2] = p1;                        // Runtime error: Point not subclass of RedPoin
t
  }
}

public class Point {
  protected internal int x, y;

  public Point(int x, int y) { this.x = x; this.y = y; }

  public void Move(int dx, int dy) { x += dx; y += dy; }

  public override String ToString() { return "(" + x + "," + y + ")"; }
}

class RedPoint : Point {
  private uint rgb;

  public RedPoint(int x, int y) : base(x, y) { this.rgb = 0xFF0000; }

  public override String ToString() {
    return base.ToString() + "@" + rgb;
  }
}
```

```
// Example 37 from page 29 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static void Main(String[] args) {
    Object[] a1 = new String[] { "a", "bc" };   // Legal:   array conversion
    Object[] a2 = new Object[] { 1, 2 };        // Legal:   conversion of 1, 2
    // Object[] a3 = new int[] { 1, 2 };        // Illegal: no array conversion
    // double[] a4 = new int[] { 1, 2 };        // Illegal: no array conversion
    foreach (String s in a1)
      Console.WriteLine(s);
    foreach (int i in a2)
      Console.WriteLine(i);
  }
}
```

```
// Example 38 from page 31 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class ArraysExample {
  public static void Main() {
    // Rectangular array creation
    double[,] r1 = { { 0.0, 0.1 }, { 1.0, 1.1 }, { 2.0, 2.1 } };
    double[,] r2 = new double[3,2];
    for (int i=0; i<3; i++)
      for (int j=0; j<2; j++)
        r2[i,j] = i + 0.1 * j;

    // Jagged array creation
    double[] row0 = { 0.0 }, row1 = { 1.0, 1.1 }, row2 = { 2.0, 2.1, 2.2 };
    double[][] t1 = { row0, row1, row2 };
    double[][] t2 = { new double[] {0.0},
                      new double[] {1.0, 1.1},
                      new double[] {2.0, 2.1, 2.2}};
    double[][] t3 = new double[3][];          // Create first dimension array
    for (int i=0; i<3; i++) {
      t3[i] = new double[i+1];                // Create second dimension arrays
      for (int j=0; j<=i; j++)
        t3[i][j] = i + 0.1 * j;
    }
    // double[][] t4 = new double[3][3];      // Illegal array creation
    Print(r1); Print(r2);
    Print(t1); Print(t2); Print(t3);
  }

  private static void Print(double[,] arr) {
    for (int i=0; i<arr.GetLength(0); i++) {
      for (int j=0; j<arr.GetLength(1); j++)
        Console.Write("{0:F1} ", arr[i,j]);
      Console.WriteLine();
    }
    Console.WriteLine();
  }

  private static void Print(double[][] arr) {
    for (int i=0; i<arr.Length; i++) {
      for (int j=0; j<arr[i].Length; j++)
        Console.Write("{0:F1} ", arr[i][j]);
      Console.WriteLine();
    }
    Console.WriteLine();
  }
}
```

```
// Example 39 from page 31 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static void Main(String[] args) {
    double[,][] rate = new double[10,12][];
    rate[0, 0] = new double[31];        // Jan 2000 has 31 days
    rate[0, 1] = new double[29];        // Feb 2000 has 29 days
    rate[0, 2] = new double[31];        // Mar 2000 has 31 days
    rate[0, 3] = new double[30];        // Apr 2000 has 30 days
    rate[0,11] = new double[31];        // Dec 2000 has 31 days
    rate[1, 0] = new double[31];        // Jan 2001 has 31 days
    rate[1, 1] = new double[28];        // Feb 2001 has 28 days
    rate[2, 1] = new double[28];        // Feb 2002 has 28 days
    rate[3, 1] = new double[28];        // Feb 2003 has 28 days
    rate[3,11] = new double[31];        // Dec 2003 has 31 days
    rate[4, 0] = new double[31];        // Jan 2004 has 31 days

    // USD per EUR daily Interbank exchange rates (www.oanda.com)
    rate[0, 1][27] = 0.9748;            // 28 Feb 2000
    rate[0, 1][28] = 0.9723;            // 29 Feb 2000
    rate[0, 2][ 0] = 0.9651;            //  1 Mar 2000
    rate[0,11][30] = 0.9421;            // 31 Dec 2000
    rate[1, 0][ 0] = 0.9421;            //  1 Jan 2001
    rate[1, 1][27] = 0.9180;            // 28 Feb 2001
    rate[2, 1][27] = 0.8641;            // 28 Feb 2002
    rate[3, 1][27] = 1.0759;            // 28 Feb 2003
    rate[3,11][ 1] = 1.1983;            //  1 Dec 2003
    rate[3,11][30] = 1.2557;            // 31 Dec 2003
    rate[4, 0][ 6] = 1.2741;            //  7 Jan 2004

    for (int y=0; y<rate.GetLength(0); y++)
      for (int m=0; m<rate.GetLength(1); m++)
        if (rate[y,m] != null)
          for (int d=0; d<rate[y,m].Length; d++)
            if (rate[y,m][d] != 0.0)
              Console.WriteLine("{0:D4}-{1:D2}-{2:D2}: {3:F4} $US/Euro",
                                y+2000, m+1, d+1, rate[y,m][d]);
  }
}
```

```
// Example 40 from page 33 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  static void ArrayInfo(String name, Array a) {
    Console.Write("{0} has length={1} rank={2} [", name, a.Length, a.Rank);
    for (int i=0, stop=a.Rank; i<stop; i++)
      Console.Write(" {0}", a.GetLength(i));
    Console.WriteLine(" ]");
  }

  public static void Main(String[] args) {
    double[,] r2 = new double[3,2];
    for (int i=0; i<3; i++)
      for (int j=0; j<2; j++)
        r2[i,j] = i + 0.1 * j;
    double[][] t2 = { new double[] {0.0},
                      new double[] {1.0, 1.1},
                      new double[] {2.0, 2.1, 2.2}};
    ArrayInfo("r2", r2);                              // length=6 rank=2 [ 3 2 ]
    ArrayInfo("t2", t2);                              // length=3 rank=1 [ 3 ]
    r2.SetValue(10.0, 1, 0);                           // Same as r2[1,0] = 10.0;
    r2.SetValue(21.0, 2, 1);                           // Same as r2[2,1] = 21.0;
    ((double[])t2.GetValue(1)).SetValue(10.0, 0);      // Same as t2[1][0] = 10.0;
    ((double[])t2.GetValue(2)).SetValue(21.0, 1);      // Same as t2[2][1] = 21.0;
    foreach (double d in r2)                           // 0 0.1 10 1.1 2.0 21
      Console.Write(d + " ");
    Console.WriteLine();
    foreach (double[] row in t2)                       // 0 10 1.1 2 21 2.2
      foreach (double d in row)
        Console.Write(d + " ");
    Console.WriteLine();
  }
}
```

```
// Example 41 from page 33 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  static String[] a = { "Armonk", "Chicago", "London", "Paris", "Seattle" };

  static void Search(String c) {
    int i = Array.BinarySearch(a, c);
    if (i >= 0)
      Console.WriteLine("{0} found in position {1}", c, i);
    else
      Console.WriteLine("{0} not found; belongs in position {1}", c, ~i);
  }

  public static void Main(String[] args) {
    Search("London");                    // found in position 2
    Search("Aachen");                    // belongs in position 0
    Search("Copenhagen");                // belongs in position 2
    Search("Washington");                // belongs in position 5
  }
}
```

```
// Example 42 from page 33 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static void Main(String[] args) {
    String[] a = { "Armonk", "Chicago", "Seattle", "London", "Paris" };
    Array.Reverse(a, 0, 3);
    Array.Reverse(a, 3, 2);
    Array.Reverse(a);
    foreach (String s in a)
      Console.Write(s + " ");
    Console.WriteLine();
  }
}
```

```
// Example 43 from page 35 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class Point {
  protected internal int x, y;

  public Point(int x, int y) {
    this.x = x; this.y = y;
  }

  public void Move(int dx, int dy) {
    x += dx; y += dy;
  }

  public override String ToString() {
    return "(" + x + "," + y + ")";
  }
}

class Dummy {
  public static void Main() { }
}
```

```
// Example 44 from page 35 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

public class APoint {
  private static List<APoint> allpoints = new List<APoint>();
  private int x, y;
  public APoint(int x, int y) {
    allpoints.Add(this); this.x = x; this.y = y;
  }
  public void Move(int dx, int dy) {
    x += dx; y += dy;
  }
  public override String ToString() {
    return "(" + x + "," + y + ")";
  }
  public int GetIndex() {
    return allpoints.IndexOf(this);
  }
  public static int GetSize() {
    return allpoints.Count;
  }
  public static APoint GetPoint(int i) {
    return allpoints[i];
  }
}

class Dummy {
  public static void Main() { }
}
```

```
// Example 45 from page 37 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public abstract class Vessel {
  private double contents;
  public abstract double Capacity();
  public void Fill(double amount) { contents = Math.Min(contents + amount, Capacity()
); }
  public double Contents { get { return contents; } }
}
public class Tank : Vessel {
  protected readonly double length, width, height;
  public Tank(double length, double width, double height)
  { this.length = length; this.width = width; this.height = height; }
  public override double Capacity() { return length * width * height; }
  public override String ToString()
  { return "tank (" + length + "," + width + "," + height + ")"; }
}
public class Cube : Tank {
  public Cube(double side) : base(side, side, side) {}
  public override String ToString() { return "cube (" + length + ")"; }
}
public class Barrel : Vessel {
  private readonly double radius, height;
  public Barrel(double radius, double height) { this.radius = radius; this.height = h
eight; }
  public override double Capacity() { return height * Math.PI * radius * radius; }
  public override String ToString() { return "barrel (" + radius + "," + height + ")"; }
}

public class UseVesselHierarchy {
  public static void Main(String[] args) {
    Vessel v1 = new Barrel(3, 10);
    Vessel v2 = new Tank(10, 20, 12);
    Vessel v3 = new Cube(4);
    Vessel[] vs = { v1, v2, v3 };
    v1.Fill(90); v1.Fill(10); v2.Fill(100); v3.Fill(80);
    double sum = 0;
    for (int i=0; i<vs.Length; i++)
      sum += vs[i].Capacity();
    Console.WriteLine("Total capacity is " + sum);
    sum = 0;
    for (int i=0; i<vs.Length; i++)
      sum += vs[i].Contents;
    Console.WriteLine("Total contents is " + sum);
    for (int i=0; i<vs.Length; i++)
      Console.WriteLine("vessel number " + i + ": " + vs[i]);
  }
}
```

```
// Example 47 from page 39 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public abstract class Vessel {
  private double contents;
  public abstract double Capacity();
  public void Fill(double amount) { contents = Math.Min(contents + amount, Capacity()
); }
  public double Contents { get { return contents; } }
}
public class Tank : Vessel {
  protected readonly double length, width, height;
  public Tank(double length, double width, double height)
  { this.length = length; this.width = width; this.height = height; }
  public override double Capacity() { return length * width * height; }
  public override String ToString()
  { return "tank (" + length + "," + width + "," + height + ")"; }
}
public class Cube : Tank {
  public Cube(double side) : base(side, side, side) {}
  public override String ToString() { return "cube (" + length + ")"; }
}
public class Barrel : Vessel {
  private readonly double radius, height;
  public Barrel(double radius, double height) { this.radius = radius; this.height = h
eight; }
  public override double Capacity() { return height * Math.PI * radius * radius; }
  public override String ToString() { return "barrel (" + radius + "," + height + ")"; }
}

public class UseVesselHierarchy {
  public static void Main(String[] args) {
    Vessel v1 = new Barrel(3, 10);
    Vessel v2 = new Tank(10, 20, 12);
    Vessel v3 = new Cube(4);
    Vessel[] vs = { v1, v2, v3 };
    v1.Fill(90); v1.Fill(10); v2.Fill(100); v3.Fill(80);
    double sum = 0;
    for (int i=0; i<vs.Length; i++)
      sum += vs[i].Capacity();
    Console.WriteLine("Total capacity is " + sum);
    sum = 0;
    for (int i=0; i<vs.Length; i++)
      sum += vs[i].Contents;
    Console.WriteLine("Total contents is " + sum);
    for (int i=0; i<vs.Length; i++)
      Console.WriteLine("vessel number " + i + ": " + vs[i]);
  }
}
```

```
// Example 48 from page 39 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class B {
  public int f;
  public B(int f) { this.f = f; }
  public void M1()         { Console.Write("B.M1 "); }   // Non-virtual instance meth
od
  public virtual void M2() { Console.Write("B.M2 "); }   // Virtual instance method
  public int FVal { get { return f; } }                   // Property
  public int this[int i] { get { return f+i; } }          // Indexer
}

class C : B {
  public new int f;
  public C(int f) : base(f/2) {
    this.f = f;
    Console.WriteLine("{0} {1} {2}", base.f, base.FVal, base[5]); // 11 11 16
    Console.WriteLine("{0} {1} {2}", f, FVal, this[5]);           // 22 22 27
  }
  public new void M1()      { base.M1(); Console.Write("C.M1 "); }
  public override void M2() { base.M2(); Console.Write("C.M2 "); }
  public new int FVal { get { return f; } }
  public new int this[int i] { get { return f+i; } }
}

class MyTest {
  public static void Main(String[] args) {
    C c = new C(22);
    c.M1(); c.M2();                                          // B.M1 C.M1 B.M2 C.M2
    Console.WriteLine();
  }
}
```

```
// Example 50 from page 41 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class B {                       // One instance field nf, one static field sf
  public int nf;
  public static int sf;
  public B(int i) { nf = i; sf = i+1; }
}
class C : B {                   // Two instance fields nf, one static field sf
  new public int nf;
  new public static int sf;
  public C(int i) : base(i+20) { nf = i; sf = i+2; }
}
class D : C {                   // Three instance fields nf
  new public int nf;
  public D(int i) : base(i+40) { nf = i; sf = i+4; }
}

class FieldAccessExample {
  public static void Main(String[] args) {
    C c1 = new C(100);                  // c1 has type C; object has class C
    B b1 = c1;                          // b1 has type B; object has class C
    Print(C.sf,  B.sf);                 // Prints 102 121
    Print(c1.nf, b1.nf);                // Prints 100 120
    C c2 = new C(200);                  // c2 has type C; object has class C
    B b2 = c2;                          // b2 has type B; object has class C
    Print(c2.nf, b2.nf);                // Prints 200 220
    Print(c1.nf, b1.nf);                // Prints 100 120
    D d3 = new D(300);                  // d3 has type D; object has class D
    C c3 = d3;                          // c3 has type C; object has class D
    B b3 = d3;                          // b3 has type B; object has class D
    Print(D.sf,  C.sf,  B.sf);          // Prints 304 304 361
    Print(d3.nf, c3.nf, b3.nf);         // Prints 300 340 360
  }

  static void Print(int x, int y) { Console.WriteLine(x+" "+y); }
  static void Print(int x, int y, int z) { Console.WriteLine(x+" "+y+" "+z); }
}
```

```
// Example 52 from page 43 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class Overloading {
  double M(int i) { return i; }
  bool M(bool b) { return !b; }
  static double M(int x, double y) { return x + y + 1; }
  static double M(double x, double y) { return x + y + 3; }

  public static void Main(String[] args) {
    Console.WriteLine(M(10, 20));               // Prints 31
    Console.WriteLine(M(10, 20.0));             // Prints 31
    Console.WriteLine(M(10.0, 20));             // Prints 33
    Console.WriteLine(M(10.0, 20.0));           // Prints 33
  }
}
```

```
// Example 54 from page 43 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static int Max(int a, double b) {
    Console.Write("Max(int, double): ");
    return a > b ? a : (int) b;
  }

  public static int Max(int a, int b, int c) {
    Console.Write("Max(int, int, int): ");
    a = a > b ? a : b;
    return a > c ? a : c;
  }

  public static int Max(int x0, params int[] xr) {
    Console.Write("Max(int, int[]): ");
    foreach (int i in xr)
      if (i > x0)
        x0 = i;
    return x0;
  }

  public static void Main(String[] args) {
    Console.WriteLine(Max(2, 1));             // Calls Max(int, int[])
    Console.WriteLine(Max(4));                // Calls Max(int, int[])
    Console.WriteLine(Max(5, 8, 7));          // Calls Max(int, int, int)
    Console.WriteLine(Max(8, 16, 10, 11));    // Calls Max(int, int[])
    int[] xr = { 13, 32, 15 };
    Console.WriteLine(Max(12, xr));           // Calls Max(int, int[])
    // Console.WriteLine(Max(16, ref xr[0])); // Illegal: no ref params
  }
}
```

```
// Example 55 from page 45 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Experiments with method modifiers (except for access modifiers)

using System;

abstract class A {
  public static void M1() { Console.Write("A.M1 "); }
  public void M2() { Console.Write("A.M2 "); }
  public virtual void M3() { Console.Write("A.M3 "); }
  public abstract void M4();
}

class B : A {
  public override void M4() { Console.Write("B.M4 "); }
}

class C : B {
  public new static void M1() { Console.Write("C.M1 "); }
  public new void M2() { Console.Write("C.M2 "); }
  public override void M3() { Console.Write("C.M3 "); }
}

abstract class D : C {
  public new abstract void M2();
  public new virtual void M3() { Console.Write("D.M3 "); }
  public abstract override void M4();
}

class E : D {
  public sealed override void M2() { Console.Write("E.M2 "); }
  public override void M3() { Console.Write("E.M3 "); }
  public override void M4() { Console.Write("E.M4 "); }
}

class MyTest {
  public static void Main(String[] args) {
    E ee = new E(); D de = ee; C ce = ee; B be = ee; A ae = ee;
    A ab = new B(); A ac = new C();
    A.M1(); B.M1(); C.M1(); D.M1(); E.M1();        // A.M1 A.M1 C.M1 C.M1 C.M1
    Console.WriteLine();
    ae.M2(); be.M2(); ce.M2(); de.M2(); ee.M2();   // A.M2 A.M2 C.M2 E.M2 E.M2
    Console.WriteLine();
    ae.M3(); be.M3(); ce.M3(); de.M3(); ee.M3();   // C.M3 C.M3 C.M3 E.M3 E.M3
    Console.WriteLine();
    ab.M2(); ac.M2(); ae.M2();                     // A.M2 A.M2 A.M2
    Console.WriteLine();
    ab.M3(); ac.M3(); ae.M3();                     // A.M3 C.M3 C.M3
    Console.WriteLine();
    ab.M4(); ac.M4(); ae.M4();                     // B.M4 B.M4 E.M4
    Console.WriteLine();
  }
}
```

```
// Example 56 from page 45 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class InitializerExample {
  static double[] ps = new double[6];
  static readonly Random rnd = new Random();

  static InitializerExample() {         // Static constructor
    double sum = 0;
    for (int i=0; i<ps.Length; i++)     // Fill with increasing random numbers
      ps[i] = sum += rnd.NextDouble();  // Random number 0 <= x < 1
    for (int i=0; i<ps.Length; i++)     // Scale so last ps element is 1.0
      ps[i] /= sum;
  }

  static int roll() {
    double p = rnd.NextDouble();
    int i = 0;
    while (p > ps[i])
      i++;
    return i+1;
  }

  public static void Main(String[] args) {
    for (int i=0; i<36; i++)
      Console.Write(roll() + " ");
    Console.WriteLine();
  }
}
```

```
// Example 57 from page 47 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;       // For IEnumerable<T>
using System.Text;                      // For StringBuilder

class MyTest {
  public static void Main(String[] args) {
    Console.WriteLine(DateTime.Today.IsoWeek());
    Console.WriteLine(new DateTime(2010, 8, 27).IsoWeek());
    String[] sarr = { "www", "itu", "dk" };
    Console.WriteLine(sarr.ConcatWith("."));
    Console.WriteLine(sarr.IsSorted());
    Console.WriteLine(new int[] { 2, 3, 5 }.IsSorted());
  }
}

public static class DateTimeExtensions {
  // ISO week number: Week 1 of a year contains its first Thursday
  public static int IsoWeek(this DateTime dt) {
    int yday = dt.DayOfYear-1, wday = IsoWeekDay(dt), y = dt.Year;
    const int THU = 3;
    int week = (yday - wday + THU + 7)/7;
    if (week == 0) {
      int prevyear = DateTime.IsLeapYear(y-1) ? 366 : 365;
      return (yday + prevyear - wday + THU + 7)/7;
    } else if (week == 53 && IsoWeekDay(new DateTime(y, 12, 31)) < THU) {
      return 1;
    } else {
      return week;
    }
  }

  // Auxiliary method: ISO weekdays: Mon=0, Tue=1, ..., Sun=6
  private static int IsoWeekDay(DateTime dt) {
    return ((int)dt.DayOfWeek + 6) % 7;
  }
}

public static class StringArrayExtensions {
  public static String ConcatWith(this String[] arr, String sep) {
    StringBuilder sb = new StringBuilder();
    if (arr.Length > 0)
      sb.Append(arr[0]);
    for (int i=1; i<arr.Length; i++)
      sb.Append(sep).Append(arr[i]);
    return sb.ToString();
  }
}

public static class EnumerableExtensions {
  public static bool IsSorted<T>(this IEnumerable<T> xs) where T : IComparable<T> {
    var etor = xs.GetEnumerator();
    if (etor.MoveNext()) {
      T prev = etor.Current;
      while (etor.MoveNext())
        if (prev.CompareTo(etor.Current) > 0)
          return false;
        else
          prev = etor.Current;
    }
    return true;
  }
}

// Extension method scope:

class My { }

namespace Outer {
  static class MyExtensions {
    public static void Extension1(this My my) { }
  }

  namespace Inner {
    static class MyExtensions {
      public static void Extension2(this My my) { }
    }
```

```
    class Try2C {
      public static void Try() {
        new My().Extension1(); // In scope here
        new My().Extension2(); // In scope here
      }
    }
  }

  class Try1C {
    public static void Try() {
      new My().Extension1();    // In scope here
      // new My().Extension2(); // Not in scope here
    }
  }
}
```

```
// Example 58 from page 47 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;       // For IEnumerable<T>
using System.Text;                      // For StringBuilder

class MyTest {
  public static void Main(String[] args) {
    Console.WriteLine(DateTime.Today.IsoWeek());
    Console.WriteLine(new DateTime(2010, 8, 27).IsoWeek());
    String[] sarr = { "www", "itu", "dk" };
    Console.WriteLine(sarr.ConcatWith("."));
    Console.WriteLine(sarr.IsSorted());
    Console.WriteLine(new int[] { 2, 3, 5 }.IsSorted());
  }
}

public static class DateTimeExtensions {
  // ISO week number: Week 1 of a year contains its first Thursday
  public static int IsoWeek(this DateTime dt) {
    int yday = dt.DayOfYear-1, wday = IsoWeekDay(dt), y = dt.Year;
    const int THU = 3;
    int week = (yday - wday + THU + 7)/7;
    if (week == 0) {
      int prevyear = DateTime.IsLeapYear(y-1) ? 366 : 365;
      return (yday + prevyear - wday + THU + 7)/7;
    } else if (week == 53 && IsoWeekDay(new DateTime(y, 12, 31)) < THU) {
      return 1;
    } else {
      return week;
    }
  }

  // Auxiliary method: ISO weekdays: Mon=0, Tue=1, ..., Sun=6
  private static int IsoWeekDay(DateTime dt) {
    return ((int)dt.DayOfWeek + 6) % 7;
  }
}

public static class StringArrayExtensions {
  public static String ConcatWith(this String[] arr, String sep) {
    StringBuilder sb = new StringBuilder();
    if (arr.Length > 0)
      sb.Append(arr[0]);
    for (int i=1; i<arr.Length; i++)
      sb.Append(sep).Append(arr[i]);
    return sb.ToString();
  }
}

public static class EnumerableExtensions {
  public static bool IsSorted<T>(this IEnumerable<T> xs) where T : IComparable<T> {
    var etor = xs.GetEnumerator();
    if (etor.MoveNext()) {
      T prev = etor.Current;
      while (etor.MoveNext())
        if (prev.CompareTo(etor.Current) > 0)
          return false;
        else
          prev = etor.Current;
    }
    return true;
  }
}

// Extension method scope:

class My { }

namespace Outer {
  static class MyExtensions {
    public static void Extension1(this My my) { }
  }

  namespace Inner {
    static class MyExtensions {
      public static void Extension2(this My my) { }
    }
```

```
    class Try2C {
      public static void Try() {
        new My().Extension1(); // In scope here
        new My().Extension2(); // In scope here
      }
    }
  }

  class Try1C {
    public static void Try() {
      new My().Extension1();    // In scope here
      // new My().Extension2(); // Not in scope here
    }
  }
}
```

```
// Example 59 from page 47 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;       // For IEnumerable<T>
using System.Text;                      // For StringBuilder

class MyTest {
  public static void Main(String[] args) {
    Console.WriteLine(DateTime.Today.IsoWeek());
    Console.WriteLine(new DateTime(2010, 8, 27).IsoWeek());
    String[] sarr = { "www", "itu", "dk" };
    Console.WriteLine(sarr.ConcatWith("."));
    Console.WriteLine(sarr.IsSorted());
    Console.WriteLine(new int[] { 2, 3, 5 }.IsSorted());
  }
}

public static class DateTimeExtensions {
  // ISO week number: Week 1 of a year contains its first Thursday
  public static int IsoWeek(this DateTime dt) {
    int yday = dt.DayOfYear-1, wday = IsoWeekDay(dt), y = dt.Year;
    const int THU = 3;
    int week = (yday - wday + THU + 7)/7;
    if (week == 0) {
      int prevyear = DateTime.IsLeapYear(y-1) ? 366 : 365;
      return (yday + prevyear - wday + THU + 7)/7;
    } else if (week == 53 && IsoWeekDay(new DateTime(y, 12, 31)) < THU) {
      return 1;
    } else {
      return week;
    }
  }

  // Auxiliary method: ISO weekdays: Mon=0, Tue=1, ..., Sun=6
  private static int IsoWeekDay(DateTime dt) {
    return ((int)dt.DayOfWeek + 6) % 7;
  }
}

public static class StringArrayExtensions {
  public static String ConcatWith(this String[] arr, String sep) {
    StringBuilder sb = new StringBuilder();
    if (arr.Length > 0)
      sb.Append(arr[0]);
    for (int i=1; i<arr.Length; i++)
      sb.Append(sep).Append(arr[i]);
    return sb.ToString();
  }
}

public static class EnumerableExtensions {
  public static bool IsSorted<T>(this IEnumerable<T> xs) where T : IComparable<T> {
    var etor = xs.GetEnumerator();
    if (etor.MoveNext()) {
      T prev = etor.Current;
      while (etor.MoveNext())
        if (prev.CompareTo(etor.Current) > 0)
          return false;
        else
          prev = etor.Current;
    }
    return true;
  }
}

// Extension method scope:

class My { }

namespace Outer {
  static class MyExtensions {
    public static void Extension1(this My my) { }
  }

  namespace Inner {
    static class MyExtensions {
      public static void Extension2(this My my) { }
    }
```

```
    class Try2C {
      public static void Try() {
        new My().Extension1(); // In scope here
        new My().Extension2(); // In scope here
      }
    }
  }

  class Try1C {
    public static void Try() {
      new My().Extension1();    // In scope here
      // new My().Extension2(); // Not in scope here
    }
  }
}
```

```
// Example 60 from page 47 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static void Main(String[] args) {
    Console.WriteLine(Convert("765"));
    Console.WriteLine(Convert("765", 10));
    Console.WriteLine(Convert("765", 8));
  }

  public static uint Convert(String s, uint radix = 10) {
    uint result = 0;
    foreach (char ch in s) {
      int d = Convert(ch);
      if (d < 0 || d >= radix)
        throw new ArgumentException("Illegal digit");
      result = result * radix + (uint)d;
    }
    return result;
  }

  private static int Convert(char ch) {
    if ('0' <= ch && ch <= '9')
      return ch - '0';
    else if ('A' <= ch && ch <= 'Z')
      return ch - 'A' + 10;
    else if ('a' <= ch && ch <= 'z')
      return ch - 'a' + 10;
    else
      return -1;
  }
}
```

```
// Example 61 from page 49 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class Test {
  public static void Main() { }
}

public class Point {
  protected internal int x, y;

  public Point(int x, int y)            // overloaded constructor
  { this.x = x; this.y = y; }

  public Point()                        // overloaded constructor
  { }

  public Point(Point p)                 // overloaded constructor
    : this(p.x, p.y) {}                 // calls the first constructor

  public void Move(int dx, int dy)
  { x += dx; y += dy; }

  public override String ToString()
  { return "(" + x + "," + y + ")"; }
}
```

```
// Example 63 from page 49 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class TestInit {
  public static void Main(String[] args) {
    new C();                    // Should print: 1 2 3 4 5 6 7
  }
}

class B {
  public readonly int fb1 = Print(3);
  public B(int k) { Print(5); }
  public readonly int fb2 = Print(4);
  public static int Print(int i) { Console.Write(i + " "); return i; }
}

class C : B {
  public readonly int fc1 = Print(1);
  public C() : this(0) { Print(7); }
  public C(int k) : base(k) { Print(6); }
  public readonly int fc2 = Print(2);
}
```

```
// Example 64 from page 51 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// The struct type of integer sequences

// Overloaded operators and indexers, enumerators

using System;                           // For Console, String
using System.Text;                      // For StringBuilder
using SC = System.Collections;          // For IEnumerator, IEnumerable
using System.Collections.Generic;       // For IEnumerator<T>

struct Seq : ISeq {
  private readonly int b, k, n;                      // Sequence b+k*[0..n-1]

  // Default constructor Seq() creates an empty sequence with n=0

  public Seq(int m, int n) : this(m, 1, n-m+1) { }   // Sequence [m..n]

  public Seq(int b, int k, int n) {
    this.b = b; this.k = k; this.n = n;
  }

  // Add b to sequence
  public static Seq operator +(int b, Seq seq) {
    return new Seq(seq.b+b, seq.k, seq.n);
  }

  // Add b to sequence
  public static Seq operator +(Seq seq, int b) {
    return new Seq(seq.b+b, seq.k, seq.n);
  }

  // Multiply all members of the sequence by k
  public static Seq operator *(int k, Seq seq) {
    return new Seq(seq.b*k, seq.k*k, seq.n);
  }

  // Multiply all members of the sequence by k
  public static Seq operator *(Seq seq, int k) {
    return new Seq(seq.b*k, seq.k*k, seq.n);
  }

  // Reverse the sequence
  public static Seq operator !(Seq seq) {
    return new Seq(seq.b+(seq.n-1)*seq.k, -seq.k, seq.n);
  }

  // Equality and inequality
  public static bool operator ==(Seq s1, Seq s2) {
    return s1.n==s2.n && (s1.n==0 || s1.b==s2.b && (s1.n==1 || s1.k==s2.k));
  }

  public static bool operator !=(Seq s1, Seq s2) { return !(s1==s2); }

  public override bool Equals(Object that) {
    return that is Seq && this==(Seq)that;
  }

  public override int GetHashCode() {
    return n==0 ? 0 : n==1 ? b : b^k^n;
  }

  // Get enumerator for the sequence
  public IEnumerator<int> GetEnumerator() {
    return new SeqEnumerator(this);
  }

  // Get enumerator for the sequence
  SC.IEnumerator SC.IEnumerable.GetEnumerator() {
    return GetEnumerator();
  }

  // An enumerator for a sequence, used in foreach statements
  private class SeqEnumerator : IEnumerator<int> {    // Static member class
    private readonly Seq seq;
    private int i;

    public SeqEnumerator(Seq seq) {
      this.seq = seq; Reset();
    }

    public int Current {                // For IEnumerator<int>
      get {
        if (0 <= i && i < seq.n)
          return seq.b + seq.k * i;
        else
          throw new InvalidOperationException();
      }
    }

    Object SC.IEnumerator.Current { get { return Current; } }

    public bool MoveNext() {            // For IEnumerator<int> and IEnumerator
      return ++i < seq.n;
    }

    public void Reset() {               // For IEnumerator
      i = -1;
    }

    public void Dispose() { }           // For IEnumerator<int>
  }

  public int Count {
    get { return n; }
  }

  public int this[int i] {
    get {
      if (0 <= i && i < n)
        return b + k * i;
      else
        throw new ArgumentOutOfRangeException("Seq indexer: " + i);
    }
  }

  public int[] this[params int[] ii] {
    get {
      int[] res = new int[ii.Length];
      for (int h=0; h<res.Length; h++)
        res[h] = this[ii[h]];
      return res;
    }
  }

  public void Print() {
    IEnumerator<int> etor = GetEnumerator();
    while (etor.MoveNext())
      Console.Write(etor.Current + " ");
  }

  public override String ToString() {
    StringBuilder sb = new StringBuilder();
    foreach (int i in this)
      sb.Append(i).Append(" ");
    return sb.ToString();
  }
}

class TestSeq {
  public static void Main(String[] args) {
    Seq s1 = new Seq(1, 3);             // 1 2 3
    Seq s2 = 2 * s1 + 5;                // 7 9 11
    Seq s3 = s2 * 3;                    // 21 27 33
    Seq s4 = !s3;                       // 33 27 21
    Console.WriteLine(s1);
    Console.WriteLine(s2);
    Console.WriteLine(s3);
    Console.WriteLine(s4);
    Console.WriteLine(s1==s2);                          // False
    Console.WriteLine(s3==!s4);                         // True
    Console.WriteLine(new Seq()==new Seq(5,7,0));       // True
    Console.WriteLine(new Seq(17,17)==new Seq(17,5,1)); // True
    s4.Print();                         // 33 27 21
    Console.WriteLine();                // 33 27 21
    for (int i=0, stop=s4.Count; i<stop; i++)
      Console.Write(s4[i] + " ");
```

```
    Console.WriteLine();
    int[] r = s4[2, 2, 1, 2, 0];
    for (int i=0, stop=r.Length; i<stop; i++)
      Console.Write(r[i] + " ");        // 21 21 27 21 33
    Console.WriteLine();
  }
}

interface ISeq : IEnumerable<int> {
  int Count { get; }
  int this[int i] { get; }
  int[] this[params int[] ii] { get; }
}
```

```
// Example 65 from page 51 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class B {
  protected static int bx = 10;
  private static int bz = 10;
}

class C : B {
  private static int cx = 11;
  public class D {
    private static int dx = bx + cx;  // Can access protected bx and private cx
    // private static int dz = bz;    // Cannot access private bz in base class
  }
  static void m() {
    // int z = D.dx;                  // Cannot access private dx in nested class
  }
}

class MyTest {
  public static void Main(String[] args) {
  }
}
```

```
// Example 66 from page 53 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

// A log of Strings that retains only the last SIZE logged Strings

public class Log {
  private const int SIZE = 5;
  private static int instanceCount = 0;
  private int count = 0;
  private String[] log = new String[SIZE];

  public Log() {
    instanceCount++;
  }

  // The number of Logs created

  public static int InstanceCount {
    get { return instanceCount; }
  }

  // Add a String to this Log

  public void Add(String msg) {
    log[count++ % SIZE] = msg;
  }

  // Property giving the number of strings inserted in this Log

  public int Count {
    get { return count; }
  }

  // The most recently logged string, if any

  public String Last {
    get { // Return the last log entry, or null if nothing logged yet
      return count==0 ? null : log[(count-1)%SIZE];
    }
    set { // Update the last log entry, or create one if nothing logged yet
      if (count==0)
        log[count++] = value;
      else
        log[(count-1)%SIZE] = value;
    }
  }

  // Return all log entries

  public String[] All {
    get {
      int size = Math.Min(count, SIZE);
      String[] res = new String[size];
      for (int i=0; i<size; i++)
        res[i] = log[(count-size+i) % SIZE];
      return res;
    }
  }
}

class TestLog {
  public static void Main(String[] args) {
    Log log1 = new Log(), log2 = new Log();
    Console.WriteLine("Number of logs = " + Log.InstanceCount);
    log1.Add("Alarm"); log1.Add("Shower"); log1.Add("Coffee");
    log1.Add("Bus"); log1.Add("Work"); log1.Add("Lunch");
    Console.WriteLine(log1.Last);
    log1.Last += " nap";
    Console.WriteLine(log1.Last);
    log1.Add("More work");
    Console.WriteLine("Logged entries = " + log1.Count);
    foreach (String s in log1.All)
      Console.WriteLine(s);
  }
}
```

```
// Example 67 from page 53 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;
using System.Text;

public class SparseMatrix {
  private readonly int rows;
  // A SparseMatrix has an array of lists of NonZeros, one for each column.
  // Invariant: In each list the nonzeros appear in increasing order of nz.i
  readonly List<NonZero>[] cols;

  // Create a sparse matrix from 2D array B which must be rectangular

  public SparseMatrix(double[][] B) {
    cols = new List<NonZero>[B.Length];
    rows = B.Length != 0 ? B[0].Length : 0;
    for (int j=0; j<B.Length; j++) {
      cols[j] = new List<NonZero>();
      for (int i=0; i<rows; i++)
        if (B[i][j] != 0.0)
          cols[j].Add(new NonZero(i, B[i][j]));
    }
  }

  // Create an all-zero rows-by-cols sparse matrix

  public SparseMatrix(int r, int c) {
    cols = new List<NonZero>[c];
    this.rows = r;
    for (int j=0; j<c; j++)
      cols[j] = new List<NonZero>();
  }

  // Properties to get the number of rows and columns of the matrix

  public int Rows {
    get { return rows; }
  }

  public int Cols {
    get { return cols.Length; }
  }

  // Indexer to get and set an element of the matrix

  public double this[int i, int j] {
    get {
      List<NonZero> colj = this[j];
      int k = 0;
      while (k < colj.Count && colj[k].i < i)
        k++;
      return k < colj.Count && colj[k].i == i ? colj[k].Mij : 0.0;
    }
    set {
      List<NonZero> colj = this[j];
      int k = 0;
      while (k < colj.Count && colj[k].i < i)
        k++;
      if (k < colj.Count && colj[k].i == i)
        colj[k].Mij = value;
      else if (value != 0.0)
        colj.Insert(k, new NonZero(i, value));
    }
  }

  // Indexer to get j'th column of matrix

  private List<NonZero> this[int j] {
    get { return cols[j]; }
  }

  // A pair of a row number i and a non-zero element B[i][-]

  private class NonZero {
    public readonly int i;
    public double Mij;
```

```
    public NonZero(int i, double Mij) {
      this.i = i; this.Mij = Mij;
    }

    public NonZero(NonZero nz) {
      this.i = nz.i; this.Mij = nz.Mij;
    }
  }

  public static SparseMatrix Add(SparseMatrix A, SparseMatrix B) {
    if (A.Rows == B.Rows && A.Cols == B.Cols) {
      int rRows = A.Rows, rCols = A.Cols;
      SparseMatrix R = new SparseMatrix(rRows, rCols);
      for (int j=0; j<rCols; j++) {
        List<NonZero> Aj = A[j], Bj = B[j], Rj = R[j];
        int ak = 0, bk = 0;
        while (ak<Aj.Count && bk<Bj.Count) {
          if (Aj[ak].i < Bj[bk].i)
            Rj.Add(new NonZero(Aj[ak++]));
          else if (Bj[bk].i < Aj[ak].i)
            Rj.Add(new NonZero(Bj[bk++]));
          else // Aj[ak].i==Bj[bk].i
            Rj.Add(new NonZero(Aj[ak].i, Aj[ak++].Mij+Bj[bk++].Mij));
        }
        while (ak<Aj.Count)
          Rj.Add(new NonZero(Aj[ak++]));
        while (bk<Bj.Count)
          Rj.Add(new NonZero(Bj[bk++]));
      }
      return R;
    } else
      throw new ApplicationException("SparseMatrix.Add: Matrix size misfit");
  }

  public override String ToString() {
    StringBuilder sb = new StringBuilder();
    for (int i=0; i<Rows; i++) {
      for (int j=0; j<Cols; j++)
        sb.AppendFormat("{0,6} ", this[i,j]);
      sb.Append("\n");
    }
    return sb.ToString();
  }
}

class TestSparseMatrix {
  public static void Main(String[] args) {
    SparseMatrix A = new SparseMatrix(4, 5), B = new SparseMatrix(4, 5);
    A[0,2] = 102; A[0,3] = 103; A[1,0] = 110; A[3,4] = 134;
    B[0,2] = 202; B[1,3] = 213; B[2,0] = 220; B[3,4] = 234;
    Console.WriteLine("A =\n{0}", A);
    Console.WriteLine("B =\n{0}", B);
    Console.WriteLine("A+B =\n{0}", SparseMatrix.Add(A,B));
  }
}
```

```
// Example 68 from page 55 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// The struct type of integer sequences

// Overloaded operators and indexers, enumerators

using System;                           // For Console, String
using System.Text;                      // For StringBuilder
using SC = System.Collections;          // For IEnumerator, IEnumerable
using System.Collections.Generic;       // For IEnumerator<T>

struct Seq : ISeq {
  private readonly int b, k, n;                     // Sequence b+k*[0..n-1]

  // Default constructor Seq() creates an empty sequence with n=0

  public Seq(int m, int n) : this(m, 1, n-m+1) { }   // Sequence [m..n]

  public Seq(int b, int k, int n) {
    this.b = b; this.k = k; this.n = n;
  }

  // Add b to sequence
  public static Seq operator +(int b, Seq seq) {
    return new Seq(seq.b+b, seq.k, seq.n);
  }

  // Add b to sequence
  public static Seq operator +(Seq seq, int b) {
    return new Seq(seq.b+b, seq.k, seq.n);
  }

  // Multiply all members of the sequence by k
  public static Seq operator *(int k, Seq seq) {
    return new Seq(seq.b*k, seq.k*k, seq.n);
  }

  // Multiply all members of the sequence by k
  public static Seq operator *(Seq seq, int k) {
    return new Seq(seq.b*k, seq.k*k, seq.n);
  }

  // Reverse the sequence
  public static Seq operator !(Seq seq) {
    return new Seq(seq.b+(seq.n-1)*seq.k, -seq.k, seq.n);
  }

  // Equality and inequality
  public static bool operator ==(Seq s1, Seq s2) {
    return s1.n==s2.n && (s1.n==0 || s1.b==s2.b && (s1.n==1 || s1.k==s2.k));
  }

  public static bool operator !=(Seq s1, Seq s2) { return !(s1==s2); }

  public override bool Equals(Object that) {
    return that is Seq && this==(Seq)that;
  }

  public override int GetHashCode() {
    return n==0 ? 0 : n==1 ? b : b^k^n;
  }

  // Get enumerator for the sequence
  public IEnumerator<int> GetEnumerator() {
    return new SeqEnumerator(this);
  }

  // Get enumerator for the sequence
  SC.IEnumerator SC.IEnumerable.GetEnumerator() {
    return GetEnumerator();
  }

  // An enumerator for a sequence, used in foreach statements
  private class SeqEnumerator : IEnumerator<int> {    // Static member class
    private readonly Seq seq;
    private int i;

    public SeqEnumerator(Seq seq) {
```

```
      this.seq = seq; Reset();
    }

    public int Current {                // For IEnumerator<int>
      get {
        if (0 <= i && i < seq.n)
          return seq.b + seq.k * i;
        else
          throw new InvalidOperationException();
      }
    }

    Object SC.IEnumerator.Current { get { return Current; } }

    public bool MoveNext() {            // For IEnumerator<int> and IEnumerator
      return ++i < seq.n;
    }

    public void Reset() {               // For IEnumerator
      i = -1;
    }

    public void Dispose() { }           // For IEnumerator<int>
  }

  public int Count {
    get { return n; }
  }

  public int this[int i] {
    get {
      if (0 <= i && i < n)
        return b + k * i;
      else
        throw new ArgumentOutOfRangeException("Seq indexer: " + i);
    }
  }

  public int[] this[params int[] ii] {
    get {
      int[] res = new int[ii.Length];
      for (int h=0; h<res.Length; h++)
        res[h] = this[ii[h]];
      return res;
    }
  }

  public void Print() {
    IEnumerator<int> etor = GetEnumerator();
    while (etor.MoveNext())
      Console.Write(etor.Current + " ");
  }

  public override String ToString() {
    StringBuilder sb = new StringBuilder();
    foreach (int i in this)
      sb.Append(i).Append(" ");
    return sb.ToString();
  }
}

class TestSeq {
  public static void Main(String[] args) {
    Seq s1 = new Seq(1, 3);             // 1 2 3
    Seq s2 = 2 * s1 + 5;                // 7 9 11
    Seq s3 = s2 * 3;                    // 21 27 33
    Seq s4 = !s3;                       // 33 27 21
    Console.WriteLine(s1);
    Console.WriteLine(s2);
    Console.WriteLine(s3);
    Console.WriteLine(s4);
    Console.WriteLine(s1==s2);                          // False
    Console.WriteLine(s3==!s4);                         // True
    Console.WriteLine(new Seq()==new Seq(5,7,0));       // True
    Console.WriteLine(new Seq(17,17)==new Seq(17,5,1)); // True
    s4.Print();                         // 33 27 21
    Console.WriteLine();                // 33 27 21
    for (int i=0, stop=s4.Count; i<stop; i++)
      Console.Write(s4[i] + " ");
    Console.WriteLine();
    int[] r = s4[2, 2, 1, 2, 0];
    for (int i=0, stop=r.Length; i<stop; i++)
      Console.Write(r[i] + " ");        // 21 21 27 21 33
    Console.WriteLine();
  }
}

interface ISeq : IEnumerable<int> {
  int Count { get; }
  int this[int i] { get; }
  int[] this[params int[] ii] { get; }
}
```

```
// Example 69 from page 57 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// The default argumentless constructor Frac(), which is unavoidable
// in a struct type, creates a Frac value that has d==0, violating the
// desirable invariant d!=0.  Fortunately, computing with such Frac
// values will throw DivideByZeroException, and conversion to double
// will produce a NaN.

using System;

struct Frac : IComparable {
  public readonly long n, d;       // NB: Meaningful only if d!=0

  public Frac(long n, long d) {
    long f = Gcd(n, d);
    this.n = n/f;
    this.d = d/f;
  }

  private static long Gcd(long m, long n) {
    while (m != 0)
      m = n % (n = m);
    return n;
  }

  public static Frac operator+(Frac r1, Frac r2) {
    return new Frac(r1.n*r2.d+r2.n*r1.d, r1.d*r2.d);
  }

  public static Frac operator*(Frac r1, Frac r2) {
    return new Frac(r1.n*r2.n, r1.d*r2.d);
  }

  // Both (or none) of the operators == and != must be defined:

  public static bool operator==(Frac r1, Frac r2) {
    return r1.n==r2.n && r1.d==r2.d;
  }

  public static bool operator!=(Frac r1, Frac r2) {
    return r1.n!=r2.n || r1.d!=r2.d;
  }

  // The preincrement and postincrement operator:

  public static Frac operator++(Frac r) {
    return r + 1;
  }

  // To implement the IComparable interface:

  public int CompareTo(Object that) {
    return ((double)this).CompareTo((double)(Frac)that);
  }

  // When == and != are defined, compatible methods Equals and
  // GetHashCode must be declared also:

  public override bool Equals(Object that) {
    return that is Frac && this == (Frac)that;
  }

  public override int GetHashCode() {
    return n.GetHashCode() ^ d.GetHashCode();
  }

  // Implicit conversion from int to Frac:

  public static implicit operator Frac(int n) {
    return new Frac(n, 1);
  }

  // Implicit conversion from long to Frac:

  public static implicit operator Frac(long n) {
    return new Frac(n, 1);
  }

  // Explicit conversion from Frac to long:

  public static explicit operator long(Frac r) {
    return r.n/r.d;
  }

  // Explicit conversion from Frac to float:

  public static explicit operator float(Frac r) {
    return ((float)r.n)/r.d;
  }

  // One cannot have an implicit conversion from Frac to double and at
  // the same time an implicit conversion from Frac to String; this
  // makes it impossible to decide which overload of WriteLine to use.

  public override String ToString() {
    if (d != 1)
      return n + "/" + d;
    else
      return n.ToString();
  }

  public bool IsZero {
    get { return n==0 && d!=0; }
  }
}

class TestFrac {
  public static void Main(String[] args) {
    Frac r1 = new Frac(6, 2), r2 = new Frac(5, 2);
    Console.WriteLine("r1={0} and r2={1}", r1, r2);
    Console.WriteLine((double)r2);      // Explicit conversion to double
    r2 = r2 * r2;                       // Overloaded multiplication
    Console.WriteLine("{0} {1} {2} {3} {4}", r2, ++r2, r2, r2++, r2);
    r2 = 0;                             // Implicit conversion from long
    for (int i=1; i<=10; i++) {
      r2 += new Frac(1, i);             // Overloaded += derived from overloaded +
      Console.WriteLine(r2 + " " + (r2 == new Frac(11, 6)));
    }
    Console.WriteLine("r2.IsZero is {0}", r2.IsZero);
    // Console.WriteLine(new Frac() + 1);
    // Console.WriteLine(new Frac() * new Frac(2, 3));
    Frac[] fs = { 5, new Frac(7, 8), 4, 2, new Frac(11, 3) };
    Array.Sort(fs);
    foreach (Frac f in fs)
      Console.WriteLine(f);
    // Using the user-defined conversions:
    Frac f1 = (byte)5;                          // Implicit int-->Frac
    Frac f2 = 1234567890123L;                   // Implicit long-->Frac
    int i1 = (int)f1;                           // Explicit Frac-->long
    double d2 = (double)f2;                     // Explicit Frac-->float
    Console.WriteLine(f1 + "==" + i1);
    Console.WriteLine("Note loss of precision:");
    Console.WriteLine(f2 + "==" + d2);
  }
}
```

```
// Example 70 from page 57 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Using events to pick up readings from a simulated thermometer.

using System;
using System.Threading;

delegate void Handler(double temperature);

class Thermometer {
  public event Handler Reading;
  private int temperature = 80;
  private static Random rnd = new Random();

  public Thermometer() {
    new Thread(new ThreadStart(Run)).Start();
  }

  private void Run() {
    for (;;) {                          // Forever simulate new readings
      temperature += rnd.Next(-5, 6);   // Random number in range -5..5
      if (Reading != null)              // If there are any handlers,
        Reading(temperature);           // call them with the new reading
      Thread.Sleep(rnd.Next(2000));
    }
  }
}

class MyTest {
  public static void Main(String[] args) {
    Thermometer t = new Thermometer();
    t.Reading += new Handler(PrintReading);
    t.Reading += new Handler(CountReading);
  }

  public static void PrintReading(double temperature) {
    Console.WriteLine(temperature);
  }

  public static void CountReading(double temperature) {
    if (++readCount % 10 == 0)
      Console.WriteLine("Now {0} readings", readCount);
  }

  private static int readCount = 0;
}
```

```
// Example 71 from page 59 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class Point {
  protected internal int x, y;

  public Point(int x, int y) { this.x = x; this.y = y; }
}

struct SPoint {
  public int x, y;

  public SPoint(int x, int y) { this.x = x; this.y = y; }
}

class MyTest {
  public static void M1() {
    Point p = new Point(11, 111), q = new Point(22, 222);
    p = q;
    p.x = 33;
    SPoint r = new SPoint(44, 444), s = new SPoint(55, 555);
    r = s;
    r.x = 66;
    int[] iarr1 = new int[4];
    int[] iarr2 = iarr1;
    iarr1[0] = 77;
    SPoint[] sarr = new SPoint[3];
    sarr[0].x = 88;
    Console.WriteLine("q.x={0} s.x={1} iarr2[0]={2}", p.x, s.x, iarr2[0]);
    M2(2);
  }

  public static void M2(int i) {
    Console.WriteLine(i);
    if (i > 0)
      M2(i-1);
  }

  public static void Main(String[] args) {
    M1();
  }
}
```

```
// Example 72 from page 63 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class ArithmeticOperators {
public static void Main() {
  int max = 2147483647;                         // = int.MaxValue
  int min = -2147483648;                        // = int.MinValue
  WriteLine(max+1);                             // Prints -2147483648
  WriteLine(min-1);                             // Prints  2147483647
  WriteLine(-min);                              // Prints -2147483648
  Write(   10/3); WriteLine(   10/(-3));        // Prints  3 -3
  Write((-10)/3); WriteLine((-10)/(-3));        // Writes -3  3
  Write(   10%3); WriteLine(   10%(-3));        // Prints  1  1
  Write((-10)%3); WriteLine((-10)%(-3));        // Prints -1 -1
}
static void Write(int i)   { Console.Write(i + " "); }
static void WriteLine(int i) { Console.WriteLine(i); }
}
```

```
// Example 73 from page 63 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class Checked {
  public static void Main() {
    char a = char.MaxValue;
    // Run-time overflow of conversion from int to char
    char b = (char)(a + 66);           // b = 'A'
    char c = checked((char)(a + 66));  // Throws OverflowException

    int max = int.MaxValue;
    // Run-time overflow of max+1:
    int j = max+1;                     // j = -2147483648
    int k = checked(max+1);            // Throws OverflowException
    int l = checked(Add(max,1));       // l = -2147483648

    // Compile-time constant overflow
    int m = int.MaxValue+1;            // Compile-time error!
    int n = unchecked(int.MaxValue+1); // n = -2147483648
    int p = checked(int.MaxValue+1);   // Compile-time error!

    Console.WriteLine(b); // 'A'
    Console.WriteLine(c); // 'A'
    Console.WriteLine(j); // -2147483648
    Console.WriteLine(l); // -2147483648
    Console.WriteLine(n); // -2147483648
  }

  static int Add(int i, int j) {
    return i+j;
  }
}
```

```
// Example 74 from page 63 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

public class Year {
  static bool LeapYear(int y) {
    return y % 4 == 0 && y % 100 != 0 || y % 400 == 0;
  }
}
```

```
// Example 75 from page 65 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class Bitwise {
  public static void Main() {
    int a = 0x3;                          // Bit pattern   0011
    int b = 0x5;                          // Bit pattern   0101
    WriteLine4(a);                        // Prints        0011
    WriteLine4(b);                        // Prints        0101
    WriteLine4(~a);                       // Prints        1100
    WriteLine4(~b);                       // Prints        1010
    WriteLine4(a & b);                    // Prints        0001
    WriteLine4(a ^ b);                    // Prints        0110
    WriteLine4(a | b);                    // Prints        0111
    Console.WriteLine(1 << 48);           // Prints           65536
    Console.WriteLine(1L << 48);          // Prints 281474976710656
    Console.WriteLine(1024 >> 40);        // Prints               4
    Console.WriteLine(1024L >> 40);       // Prints               0
    Console.WriteLine(1 << -2);           // Prints      1073741824
  }
  static void WriteLine4(int n) {
    for (int i=3; i>=0; i--)
      Console.Write(n >> i & 1);
    Console.WriteLine();
  }
}
```

```
// Example 76 from page 65 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

[Flags]              // Print enum combinations symbolically
public enum FileAccess {
  Read = 1 << 0,
  Write = 1 << 1
}

class MyTest {
  public static void Write(FileAccess access) {
    if (0 != (access & FileAccess.Write))
      Console.WriteLine("You have write permission");
  }

  public static void Main(String[] args) {
    FileAccess access = FileAccess.Read | FileAccess.Write;
    Console.WriteLine(access);                              // Prints: Read, Write
    Write(access);
  }
}
```

```
// Example 77 from page 67 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class Assignments {
  public static void Main() {
    double d;
    d = 12;               // legal:   implicit conversion from int to double
    byte b;
    b = 252 + 1;          // legal:   252 + 1 is a compile-time constant
    // b = 252 + 5;       // illegal: 252 + 5 is too large
    // b = b + 2;         // illegal: b + 2 has type int
    b = (byte)(b + 2);    // legal:   right-hand side has type byte
    b += 2;               // legal:   equivalent to b = (byte)(b + 2)
    // b += 257;          // illegal: b = 257 would be illegal
    Console.WriteLine("b = {0}", b);
  }
}
```

```
// Example 78 from page 67 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class CompoundAssignment {
  static double Multiply(double[] xs) {
    double prod = 1.0;
    for (int i=0; i<xs.Length; i++)
      prod *= xs[i];                          // equivalent to: prod = prod * xs[i]
    return prod;
  }

  public static void Main() {
    Console.WriteLine(Multiply(new double[] { 7.1, 6.3, 10.0 }));
  }
}
```

```
// Example 79 from page 67 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class Expressions3 {
  public static void Main(String[] args) {
    Console.WriteLine(Absolute(-12));
    Console.WriteLine(Absolute(12));
    Console.WriteLine(Absolute(0));
  }

  // Returns the absolute value of x (always non-negative)
  static double Absolute(double x)
  { return (x >= 0 ? x : -x); }
}
```

```
// Example 80 from page 67 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

class LongComparer : IComparer<long> {
  public int Compare(long v1, long v2) {
    return v1<v2 ? -1 : v1>v2 ? +1 : 0;
  }

  public static void Main(String[] args) {
    // Prints -1 -1 0 0 1 1 1
    LongComparer cmp = new LongComparer();
    Console.WriteLine(cmp.Compare(5L, 7L));
    Console.WriteLine(cmp.Compare(long.MinValue, long.MaxValue));
    Console.WriteLine(cmp.Compare(7L, 7L));
    Console.WriteLine(cmp.Compare(long.MinValue, long.MinValue));
    Console.WriteLine(cmp.Compare(7L, 5L));
    Console.WriteLine(cmp.Compare(0L, long.MinValue));
    Console.WriteLine(cmp.Compare(long.MaxValue, long.MinValue));
  }
}
```

```
// Example 81 from page 69 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

class MyTest {
  public static void Main(String[] args) {
    Point p1 = new Point { x = 10, y = 12 };
    Point p2 = new Point();                       // Equivalent to p1 initialization
    p2.x = 10;
    p2.y = 12;
    Point p3 = new Point(p1) { y = 17 };
    Point p4 = new Point(p1);                     // Equivalent to p3 initialization
    p4.y = 17;
    Console.WriteLine(p1);
    Console.WriteLine(p2);
    Console.WriteLine(p3);
    Console.WriteLine(p4);

    Rectangle r1 = new Rectangle { Ul = new Point { x = 10, y = 12 },
                                   Lr = new Point { x = 14, y = 20 } };
    Rectangle r2 = new Rectangle { Ul = { x = 10, y = 12 }, Lr = { x = 14, y = 20 } }
;
    Rectangle r3 = new Rectangle();
    r3.Ul.x = 10;
    r3.Ul.y = 12;
    r3.Lr.x = 14;
    r3.Lr.y = 20;
    Console.WriteLine(r1);
    Console.WriteLine(r2);
    Console.WriteLine(r3);
  }
}

public class Rectangle {
  public Point Ul { get; set; }          // Upper left corner
  public Point Lr { get; set; }          // Lower right corner
  public Rectangle() {
    Ul = new Point();
    Lr = new Point();
  }
  public override String ToString() {
    return "{Ul=" + Ul + ",Lr=" + Lr + "}";
  }
}

public class Point {
  protected internal int x, y;

  public Point(int x, int y)
  { this.x = x; this.y = y; }

  public Point()
  { }

  public Point(Point p)
    : this(p.x, p.y) {}

  public void Move(int dx, int dy)
  { x += dx; y += dy; }

  public override String ToString()
  { return "(" + x + "," + y + ")"; }
}
```

```
// Example 82 from page 69 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

class MyTest {
  public static void Main(String[] args) {
    Point p1 = new Point { x = 10, y = 12 };
    Point p2 = new Point();                     // Equivalent to p1 initialization
    p2.x = 10;
    p2.y = 12;
    Point p3 = new Point(p1) { y = 17 };
    Point p4 = new Point(p1);                   // Equivalent to p3 initialization
    p4.y = 17;
    Console.WriteLine(p1);
    Console.WriteLine(p2);
    Console.WriteLine(p3);
    Console.WriteLine(p4);

    Rectangle r1 = new Rectangle { Ul = new Point { x = 10, y = 12 },
                                   Lr = new Point { x = 14, y = 20 } };
    Rectangle r2 = new Rectangle { Ul = { x = 10, y = 12 }, Lr = { x = 14, y = 20 } }
;
    Rectangle r3 = new Rectangle();
    r3.Ul.x = 10;
    r3.Ul.y = 12;
    r3.Lr.x = 14;
    r3.Lr.y = 20;
    Console.WriteLine(r1);
    Console.WriteLine(r2);
    Console.WriteLine(r3);
  }
}

public class Rectangle {
  public Point Ul { get; set; }         // Upper left corner
  public Point Lr { get; set; }         // Lower right corner
  public Rectangle() {
    Ul = new Point();
    Lr = new Point();
  }
  public override String ToString() {
    return "{Ul=" + Ul + ",Lr=" + Lr + "}";
  }
}

public class Point {
  protected internal int x, y;

  public Point(int x, int y)
  { this.x = x; this.y = y; }

  public Point()
  { }

  public Point(Point p)
    : this(p.x, p.y) {}

  public void Move(int dx, int dy)
  { x += dx; y += dy; }

  public override String ToString()
  { return "(" + x + "," + y + ")"; }
}
```

```
// Example 83 from page 69 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

class MyTest {
  public static void Main(String[] args) {
    List<int> list1 = new List<int> { { 2 }, { 3 }, { 2+3 }, { 5+2 } };
    List<int> list2 = new List<int>();
    list2.Add(2); list2.Add(3); list2.Add(2+3); list2.Add(5+2);
    List<int> list3 = new List<int> { 2, 3, 2+3, 5+2 };
    int cinq, sept;
    List<int> list4 = new List<int> { 2, 3, { cinq = 2+3 }, { sept = cinq+2 } };
    Print(list1);
    Print(list2);
    Print(list3);
    Print(list4);

    Dictionary<int,String> numerals
      = new Dictionary<int,String> { { 1, "one" }, { 2, "two" }, { 5, "five" } };
    Console.WriteLine(numerals[2]);

    Polygon poly = new Polygon { { 1, 1 }, { 1, 4 }, { 4, 4  } };
  }

  private static void Print(List<int> xs) {
    foreach (int x in xs)
      Console.Write(x + " ");
    Console.WriteLine();
  }
}

public class Point {
  protected internal int x, y;

  public Point(int x, int y)            // overloaded constructor
  { this.x = x; this.y = y; }

  public Point()                        // overloaded constructor
  { }

  public Point(Point p)                 // overloaded constructor
    : this(p.x, p.y) {}                 // calls the first constructor

  public void Move(int dx, int dy)
  { x += dx; y += dy; }

  public override String ToString()
  { return "(" + x + "," + y + ")"; }
}

public class Polygon : List<Point> {
  public void Add(int x, int y) { base.Add(new Point { x = x, y = y }); }
}
```

```
// Example 84 from page 71 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// The default argumentless constructor Frac(), which is unavoidable
// in a struct type, creates a Frac value that has d==0, violating the
// desirable invariant d!=0.  Fortunately, computing with such Frac
// values will throw DivideByZeroException, and conversion to double
// will produce a NaN.

using System;

struct Frac : IComparable {
  public readonly long n, d;       // NB: Meaningful only if d!=0

  public Frac(long n, long d) {
    long f = Gcd(n, d);
    this.n = n/f;
    this.d = d/f;
  }

  private static long Gcd(long m, long n) {
    while (m != 0)
      m = n % (n = m);
    return n;
  }

  public static Frac operator+(Frac r1, Frac r2) {
    return new Frac(r1.n*r2.d+r2.n*r1.d, r1.d*r2.d);
  }

  public static Frac operator*(Frac r1, Frac r2) {
    return new Frac(r1.n*r2.n, r1.d*r2.d);
  }

  // Both (or none) of the operators == and != must be defined:

  public static bool operator==(Frac r1, Frac r2) {
    return r1.n==r2.n && r1.d==r2.d;
  }

  public static bool operator!=(Frac r1, Frac r2) {
    return r1.n!=r2.n || r1.d!=r2.d;
  }

  // The preincrement and postincrement operator:

  public static Frac operator++(Frac r) {
    return r + 1;
  }

  // To implement the IComparable interface:

  public int CompareTo(Object that) {
    return ((double)this).CompareTo((double)(Frac)that);
  }

  // When == and != are defined, compatible methods Equals and
  // GetHashCode must be declared also:

  public override bool Equals(Object that) {
    return that is Frac && this == (Frac)that;
  }

  public override int GetHashCode() {
    return n.GetHashCode() ^ d.GetHashCode();
  }

  // Implicit conversion from int to Frac:

  public static implicit operator Frac(int n) {
    return new Frac(n, 1);
  }

  // Implicit conversion from long to Frac:

  public static implicit operator Frac(long n) {
    return new Frac(n, 1);
  }

  // Explicit conversion from Frac to long:

  public static explicit operator long(Frac r) {
    return r.n/r.d;
  }

  // Explicit conversion from Frac to float:

  public static explicit operator float(Frac r) {
    return ((float)r.n)/r.d;
  }

  // One cannot have an implicit conversion from Frac to double and at
  // the same time an implicit conversion from Frac to String; this
  // makes it impossible to decide which overload of WriteLine to use.

  public override String ToString() {
    if (d != 1)
      return n + "/" + d;
    else
      return n.ToString();
  }

  public bool IsZero {
    get { return n==0 && d!=0; }
  }
}

class TestFrac {
  public static void Main(String[] args) {
    Frac r1 = new Frac(6, 2), r2 = new Frac(5, 2);
    Console.WriteLine("r1={0} and r2={1}", r1, r2);
    Console.WriteLine((double)r2);      // Explicit conversion to double
    r2 = r2 * r2;                       // Overloaded multiplication
    Console.WriteLine("{0} {1} {2} {3} {4}", r2, ++r2, r2, r2++, r2);
    r2 = 0;                             // Implicit conversion from long
    for (int i=1; i<=10; i++) {
      r2 += new Frac(1, i);             // Overloaded += derived from overloaded +
      Console.WriteLine(r2 + " " + (r2 == new Frac(11, 6)));
    }
    Console.WriteLine("r2.IsZero is {0}", r2.IsZero);
    // Console.WriteLine(new Frac() + 1);
    // Console.WriteLine(new Frac() * new Frac(2, 3));
    Frac[] fs = { 5, new Frac(7, 8), 4, 2, new Frac(11, 3) };
    Array.Sort(fs);
    foreach (Frac f in fs)
      Console.WriteLine(f);
    // Using the user-defined conversions:
    Frac f1 = (byte)5;                          // Implicit int-->Frac
    Frac f2 = 1234567890123L;                   // Implicit long-->Frac
    int i1 = (int)f1;                           // Explicit Frac-->long
    double d2 = (double)f2;                     // Explicit Frac-->float
    Console.WriteLine(f1 + "==" + i1);
    Console.WriteLine("Note loss of precision:");
    Console.WriteLine(f2 + "==" + d2);
  }
}
```

```
// Example 85 from page 71 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

interface I1 { }
interface I2 : I1 { }
class B : I2 { }
class C : B { }

class MyTest {
  public static void Main(String[] args) {
    Object n1 = new Exception(), n2 = "foo", n3 = null, n4 = 4711;
    Object n5 = new B(), n6 = new C();
    Object n7 = new C[10];
    Print("n1 is a String:    " + (n1 is String));     // False
    Print("n2 is a String:    " + (n2 is String));     // True
    Print("null is a String:  " + (n3 is String));     // False
    Print("4711 is an int:    " + (n4 is int));        // True
    Print("4711 is a long:    " + (n4 is long));       // False
    Print("4711 is a ValueType: " + (n4 is ValueType)); // True
    Print("n5 is an I1:       " + (n5 is I1));         // True
    Print("n5 is an I2:       " + (n5 is I2));         // True
    Print("n5 is a B:         " + (n5 is B));          // True
    Print("n5 is a C:         " + (n5 is C));          // False
    Print("n6 is a B:         " + (n6 is B));          // True
    Print("n6 is a C:         " + (n6 is C));          // True
    Print("n7 is an Array:    " + (n7 is Array));      // True
    Print("n7 is a B[]:       " + (n7 is B[]));        // True
    Print("n7 is a C[]:       " + (n7 is C[]));        // True
  }

  public static void Print(String s) {
    Console.WriteLine(s);
  }
}
```

```
// Example 86 from page 71 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static void Main(String[] args) {
    IsNearPoint(new Point(30, 20));
    IsNearPoint(new Point(4, 5));
    IsNearPoint("foo");
    IsNearPoint(null);
  }

  public static void IsNearPoint(Object o) {
    Point p = o as Point;
    if (p != null && p.x*p.x + p.y*p.y <= 100)
      Console.WriteLine(p + " is a Point near (0,0)");
    else
      Console.WriteLine(o + " is not a Point or not near (0,0)");
  }
}

public class Point {
  protected internal int x, y;

  public Point(int x, int y) { this.x = x; this.y = y; }

  public void Move(int dx, int dy) { x += dx; y += dy; }

  public override String ToString() { return "(" + x + "," + y + ")"; }
}
```

```
// Example 87 from page 73 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class B {                       // One instance field nf, one static field sf
  public int nf;
  public static int sf;
  public B(int i) { nf = i; sf = i+1; }
}
class C : B {                   // Two instance fields nf, one static field sf
  new public int nf;
  new public static int sf;
  public C(int i) : base(i+20) { nf = i; sf = i+2; }
}
class D : C {                   // Three instance fields nf
  new public int nf;
  public D(int i) : base(i+40) { nf = i; sf = i+4; }
}

class FieldAccessExample {
  public static void Main(String[] args) {
    C c1 = new C(100);                  // c1 has type C; object has class C
    B b1 = c1;                          // b1 has type B; object has class C
    Print(C.sf,  B.sf);                 // Prints 102 121
    Print(c1.nf, b1.nf);                // Prints 100 120
    C c2 = new C(200);                  // c2 has type C; object has class C
    B b2 = c2;                          // b2 has type B; object has class C
    Print(c2.nf, b2.nf);                // Prints 200 220
    Print(c1.nf, b1.nf);                // Prints 100 120
    D d3 = new D(300);                  // d3 has type D; object has class D
    C c3 = d3;                          // c3 has type C; object has class D
    B b3 = d3;                          // b3 has type B; object has class D
    Print(D.sf,  C.sf,  B.sf);          // Prints 304 304 361
    Print(d3.nf, c3.nf, b3.nf);         // Prints 300 340 360
  }

  static void Print(int x, int y) { Console.WriteLine(x+" "+y); }
  static void Print(int x, int y, int z) { Console.WriteLine(x+" "+y+" "+z); }
}
```

```
// Example 88 from page 73 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class Point {
  protected internal int x, y;

  public Point(int x, int y) {
    this.x = x; this.y = y;
  }

  public void Move(int dx, int dy) {
    x += dx; y += dy;
  }

  public override String ToString() {
    return "(" + x + "," + y + ")";
  }
}

class Dummy {
  public static void Main() { }
}
```

```
// Example 89 from page 73 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

public class APoint {
  private static List<APoint> allpoints = new List<APoint>();
  private int x, y;
  public APoint(int x, int y) {
    allpoints.Add(this); this.x = x; this.y = y;
  }
  public void Move(int dx, int dy) {
    x += dx; y += dy;
  }
  public override String ToString() {
    return "(" + x + "," + y + ")";
  }
  public int GetIndex() {
    return allpoints.IndexOf(this);
  }
  public static int GetSize() {
    return allpoints.Count;
  }
  public static APoint GetPoint(int i) {
    return allpoints[i];
  }
}

class Dummy {
  public static void Main() { }
}
```

```
// Example 90 from page 75 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class B {
  public static void M() { Console.WriteLine("B.M"); }
  private static void M(int i) { Console.WriteLine("B.M(int)"); }
}

class E {
  static void M() { Console.WriteLine("E.M"); }
  static void M(int i) { Console.WriteLine("E.M(int)"); }

  class C : B {
    public static void Main(String[] args) {
      M();
    }
  }
}

/*
   1. The call to M() in C will call B.M() because it is inherited by C.

   2. If B.M() is made private it is no longer inherited by C, and the
      call to M() in C will call E.M().

   3. If furthermore B.M(int) is made public, then the call to M() in
      C is no longer legal.
*/
```

```
// Example 91 from page 75 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

class TestAPoint {
  public static void Main(String[] args) {
    Console.WriteLine("Number of points created: " + APoint.GetSize());
    APoint p = new APoint(12, 123), q = new APoint(200, 10), r = new APoint(99, 12);
    APoint s = p;
    q = null;
    Console.WriteLine("Number of points created: " + APoint.GetSize());
    Console.WriteLine("r is point number " + r.GetIndex());
    for (int i=0; i<APoint.GetSize(); i++)
      Console.WriteLine("APoint number " + i + " is " + APoint.GetPoint(i));
  }
}

public class APoint {
  private static List<APoint> allpoints = new List<APoint>();
  private int x, y;
  public APoint(int x, int y) {
    allpoints.Add(this); this.x = x; this.y = y;
  }
  public void Move(int dx, int dy) {
    x += dx; y += dy;
  }
  public override String ToString() {
    return "(" + x + "," + y + ")";
  }
  public int GetIndex() {
    return allpoints.IndexOf(this);
  }
  public static int GetSize() {
    return allpoints.Count;
  }
  public static APoint GetPoint(int i) {
    return allpoints[i];
  }
}
```

```
// Example 92 from page 75 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class Overloading {
  double M(int i) { return i; }
  bool M(bool b) { return !b; }
  static double M(int x, double y) { return x + y + 1; }
  static double M(double x, double y) { return x + y + 3; }

  public static void Main(String[] args) {
    Console.WriteLine(M(10, 20));              // Prints 31
    Console.WriteLine(M(10, 20.0));            // Prints 31
    Console.WriteLine(M(10.0, 20));            // Prints 33
    Console.WriteLine(M(10.0, 20.0));          // Prints 33
  }
}
```

```
// Example 94 from page 77 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Passing an object by value is much the same as passing a struct by
// reference.

using System;

public class Point {
  protected internal int x, y;

  public Point(int x, int y) { this.x = x; this.y = y; }

  public void Move(int dx, int dy) { x += dx; y += dy; }

  public override String ToString() { return "(" + x + "," + y + ")"; }
}

public struct SPoint {
  internal int x, y;

  public SPoint(int x, int y) { this.x = x; this.y = y; }

  public void Move(int dx, int dy) { x += dx; y += dy; }

  public override String ToString() { return "(" + x + "," + y + ")"; }
}

class TestClassStruct {
  public static void Main(String[] args) {
    AssignPointClass();
    AssignPointStruct();
    PassClassStruct();
  }

  static void AssignPointClass() {
    Point p = new Point(11, 111);
    Point q = new Point(22, 222);
    Point r = new Point(33, 333);
    r = q;
    r.x = 44;
    Console.WriteLine("p = {0}, q = {1}, r = {2}", p, q, r);
  }

  static void AssignPointStruct() {
    SPoint p = new SPoint(11, 111);
    SPoint q = new SPoint(22, 222);
    SPoint r = new SPoint(33, 333);
    r = q;
    r.x = 44;
    Console.WriteLine("p = {0}, q = {1}, r = {2}", p, q, r);
  }

  static void PassClassStruct() {
    double d1 = 1.1, d2 = 2.2;
    int[] a1 = new int[4], a2 = new int[4];
    M(d1, ref d2, a1, ref a2);
    Console.WriteLine("d1 = {0}, d2 = {1}", d1, d2);
    Console.WriteLine("a1.Length = {0}, a1[0] = {1}", a1.Length, a1[0]);
    Console.WriteLine("a2.Length = {0}, a2[0] = {1}", a2.Length, a2[0]);

    Point pc1 = new Point(55, 555), pc2 = new Point(66, 666);
    SPoint ps1 = new SPoint(77, 777), ps2 = new SPoint(88, 888);
    M(pc1, ref pc2, ps1, ref ps2);
    Console.WriteLine("pc1 = {0}, pc2 = {1}", pc1, pc2);
    Console.WriteLine("ps1 = {0}, ps2 = {1}", ps1, ps2);
  }

  static void M(double dd1, ref double dd2, int[] aa1, ref int[] aa2) {
    dd1 = 3.3; dd2 = 4.4;
    aa1[0] = 17;
    aa2[0] = 18;
    aa2 = new int[3];
    aa1 = aa2;
  }

  static void M(Point ppc1, ref Point ppc2, SPoint pps1, ref SPoint pps2) {
    ppc1.x = 97;
    ppc2 = new Point(16, 17);
    ppc1 = ppc2;
    pps1.x = 98;
    pps1 = new SPoint(18, 19);
    pps2.x = 99;
  }
}
```

```
// Example 95 from page 77 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static int Max(int a, double b) {
    Console.Write("Max(int, double): ");
    return a > b ? a : (int) b;
  }

  public static int Max(int a, int b, int c) {
    Console.Write("Max(int, int, int): ");
    a = a > b ? a : b;
    return a > c ? a : c;
  }

  public static int Max(int x0, params int[] xr) {
    Console.Write("Max(int, int[]): ");
    foreach (int i in xr)
      if (i > x0)
        x0 = i;
    return x0;
  }

  public static void Main(String[] args) {
    Console.WriteLine(Max(2, 1));             // Calls Max(int, int[])
    Console.WriteLine(Max(4));                // Calls Max(int, int[])
    Console.WriteLine(Max(5, 8, 7));          // Calls Max(int, int, int)
    Console.WriteLine(Max(8, 16, 10, 11));    // Calls Max(int, int[])
    int[] xr = { 13, 32, 15 };
    Console.WriteLine(Max(12, xr));           // Calls Max(int, int[])
    // Console.WriteLine(Max(16, ref xr[0])); // Illegal: no ref params
  }
}
```

```
// Example 96 from page 79 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Passing an object by value is much the same as passing a struct by
// reference.

using System;

public class Point {
  protected internal int x, y;

  public Point(int x, int y) { this.x = x; this.y = y; }

  public void Move(int dx, int dy) { x += dx; y += dy; }

  public override String ToString() { return "(" + x + "," + y + ")"; }
}

public struct SPoint {
  internal int x, y;

  public SPoint(int x, int y) { this.x = x; this.y = y; }

  public void Move(int dx, int dy) { x += dx; y += dy; }

  public override String ToString() { return "(" + x + "," + y + ")"; }
}

class TestClassStruct {
  public static void Main(String[] args) {
    AssignPointClass();
    AssignPointStruct();
    PassClassStruct();
  }

  static void AssignPointClass() {
    Point p = new Point(11, 111);
    Point q = new Point(22, 222);
    Point r = new Point(33, 333);
    r = q;
    r.x = 44;
    Console.WriteLine("p = {0}, q = {1}, r = {2}", p, q, r);
  }

  static void AssignPointStruct() {
    SPoint p = new SPoint(11, 111);
    SPoint q = new SPoint(22, 222);
    SPoint r = new SPoint(33, 333);
    r = q;
    r.x = 44;
    Console.WriteLine("p = {0}, q = {1}, r = {2}", p, q, r);
  }

  static void PassClassStruct() {
    double d1 = 1.1, d2 = 2.2;
    int[] a1 = new int[4], a2 = new int[4];
    M(d1, ref d2, a1, ref a2);
    Console.WriteLine("d1 = {0}, d2 = {1}", d1, d2);
    Console.WriteLine("a1.Length = {0}, a1[0] = {1}", a1.Length, a1[0]);
    Console.WriteLine("a2.Length = {0}, a2[0] = {1}", a2.Length, a2[0]);

    Point pc1 = new Point(55, 555), pc2 = new Point(66, 666);
    SPoint ps1 = new SPoint(77, 777), ps2 = new SPoint(88, 888);
    M(pc1, ref pc2, ps1, ref ps2);
    Console.WriteLine("pc1 = {0}, pc2 = {1}", pc1, pc2);
    Console.WriteLine("ps1 = {0}, ps2 = {1}", ps1, ps2);
  }

  static void M(double dd1, ref double dd2, int[] aa1, ref int[] aa2) {
    dd1 = 3.3; dd2 = 4.4;
    aa1[0] = 17;
    aa2[0] = 18;
    aa2 = new int[3];
    aa1 = aa2;
  }

  static void M(Point ppc1, ref Point ppc2, SPoint pps1, ref SPoint pps2) {
    ppc1.x = 97;
    ppc2 = new Point(16, 17);
```

```
    ppc1 = ppc2;
    pps1.x = 98;
    pps1 = new SPoint(18, 19);
    pps2.x = 99;
  }
}
```

```
// Example 97 from page 81 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Experiments with method modifiers (except for access modifiers)

using System;

abstract class A {
  public static void M1() { Console.Write("A.M1 "); }
  public void M2() { Console.Write("A.M2 "); }
  public virtual void M3() { Console.Write("A.M3 "); }
  public abstract void M4();
}

class B : A {
  public override void M4() { Console.Write("B.M4 "); }
}

class C : B {
  public new static void M1() { Console.Write("C.M1 "); }
  public new void M2() { Console.Write("C.M2 "); }
  public override void M3() { Console.Write("C.M3 "); }
}

abstract class D : C {
  public new abstract void M2();
  public new virtual void M3() { Console.Write("D.M3 "); }
  public abstract override void M4();
}

class E : D {
  public sealed override void M2() { Console.Write("E.M2 "); }
  public override void M3() { Console.Write("E.M3 "); }
  public override void M4() { Console.Write("E.M4 "); }
}

class MyTest {
  public static void Main(String[] args) {
    E ee = new E(); D de = ee; C ce = ee; B be = ee; A ae = ee;
    A ab = new B(); A ac = new C();
    A.M1(); B.M1(); C.M1(); D.M1(); E.M1();        // A.M1 A.M1 C.M1 C.M1 C.M1
    Console.WriteLine();
    ae.M2(); be.M2(); ce.M2(); de.M2(); ee.M2();   // A.M2 A.M2 C.M2 E.M2 E.M2
    Console.WriteLine();
    ae.M3(); be.M3(); ce.M3(); de.M3(); ee.M3();   // C.M3 C.M3 C.M3 E.M3 E.M3
    Console.WriteLine();
    ab.M2(); ac.M2(); ae.M2();                     // A.M2 A.M2 A.M2
    Console.WriteLine();
    ab.M3(); ac.M3(); ae.M3();                     // A.M3 C.M3 C.M3
    Console.WriteLine();
    ab.M4(); ac.M4(); ae.M4();                     // B.M4 B.M4 E.M4
    Console.WriteLine();
  }
}
```

```
// Example 98 from page 81 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static void Main(String[] args) {
    D2 d2 = new D2();
    d2.M2();
  }
}

class D1 {
  public D1() { M2(); }
  public virtual void M1() { Console.WriteLine("D1.M1 "); }
  public virtual void M2() { Console.Write("D1.M2 "); M1(); }
}

class D2 : D1 {
  int f;
  public D2() { f = 7; }
  public override void M1() { Console.WriteLine("D2.M1:" + f); }
}
```

```
// Example 101 from page 83 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

class MyTest {
  private static readonly Random rnd = new Random();

  class Phone {
    public readonly String name;
    public readonly int phone;
    public Phone(String name, int phone) {
      this.name = name;
      this.phone = phone;
    }
  }

  public static void Main(String[] args) {
    // Basic rules of type dynamic
    {
      dynamic d1 = 34;
      int i1 = d1 * 2;                  // OK: cast (int)(d1*2) at run-time
      int i2 = (int)d1 * 2;             // OK: cast (int)d1 at run-time
      // bool b1 = d1;                  // Compiles OK; cast (bool)d1 throws at run-t
ime
      d1 = true;                        // OK
      bool b2 = d1;                     // OK: cast (bool)d1 succeeds at run-time
      dynamic p1 = new Phone("Kasper", 5170);
      String s1 = p1.name;              // Field access checked at run-time
      // int n1 = p1.age;               // Compiles OK; field access throws at run-ti
me
      dynamic p2 = new { name = "Kasper", phone = 5170 };
      String s2 = p2.name;              // Field access checked at run-time
      // int n2 = p2.age;               // Compiles OK; fields access throws at run-t
ime
    }

    // Dynamic operator resolution; run-time type determines meaning of "+" in Plus2(
)
    {
      Console.WriteLine(Plus2(int.MaxValue-1));             // -2147483648, due to i
nt overflow
      Console.WriteLine(Plus2((long)(int.MaxValue-1)));     //  2147483648, no long
overflow
      Console.WriteLine(Plus2(11.5));                       // 13.5
      Console.WriteLine(Plus2("Spar"));                    // Spar2
      // Console.WriteLine(Plus2(false));       // Compiles OK; throws RuntimeBinder
Exception
    }

    // Dynamic receiver; run-time type determines whether to call Length on array or
String
    {
      dynamic v;
      if (args.Length==0)      v = new int[] { 2, 3, 5, 7 };
      else                     v = "abc";
      int res = v.Length;
      Console.WriteLine(res);
    }

    // Dynamic overload resolution; run-time type of v determines which Process calle
d at (**)
    {
      dynamic v;
      if (args.Length==0)      v = 5;
      else if (args[0] == "1") v = "abc";
      else                     v = (Func<int,int>)(x => x*3);
      dynamic r = Process(v);                                    // (**)
      if (args.Length==0 || args[0] == "1")
        Console.WriteLine(r);
      else
        Console.WriteLine(r(11));
      dynamic s = "abc";
      Console.WriteLine(Process(s).StartsWith("abca"));
     }

      // Run-time type tests
    {
```

```
      Console.WriteLine(Square(5));
      Console.WriteLine(Square("abc"));
      Func<int,int> f = x => x*3;
      Console.WriteLine(Square(f)(11));
    }

    {
      // Types dynamic[], List<dynamic>, IEnumerable<dynamic>
      dynamic[] arr = new dynamic[] { 19, "Electric", (Func<int,int>)(n => n+2), 3.2, f
alse };
      int number = arr[0] * 5;
      String street = arr[1].ToUpper();
      int result = arr[2](number);
      Console.WriteLine(number + " " + street);
      double sum = 0;
      List<dynamic> list = new List<dynamic>(arr);
      IEnumerable<dynamic> xs = list;
      foreach (dynamic x in xs)
        if (x is int || x is double)
          sum += x;
      Console.WriteLine(sum);                                    // 22.2
    }

    // Dynamic and anonymous object expressions
    {
      dynamic v = new { x = 34, y = false };
      Console.WriteLine(v.x);
    }
  }

  // Run-time type of v determines meaning of "+": int+, long+, double+, String+, ...
  static dynamic Plus2(dynamic v) { return v + 2; }

  // Ordinary overloaded methods
  static int Process(int v) { return v * v; }

  static double Process(double v) { return v * v; }

  static String Process(String v) { return v + v; }

  static Func<int,int> Process(Func<int,int> v) { return x => v(v(x)); }

  // Explicit tests on run-time type.  Using "dynamic" rather than
  // "Object" here means that the compiler accepts the (v * v)
  // expression and the application of v to arguments, and that the
  // value returned by Square can be further processed: added to,
  // applied to arguments, and so on.
  static dynamic Square(dynamic v) {
    if (v is int || v is double)
      return v * v;
    else if (v is String)
      return v + v;
    else if (v is Func<int,int>)
      return (Func<int,int>)(x => v(v(x)));
    else
      throw new Exception("Don't know how to square " + v);
  }
}

class C : List<dynamic> { }
```

```
// Example 102 from page 83 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

class MyTest {
  private static readonly Random rnd = new Random();

  class Phone {
    public readonly String name;
    public readonly int phone;
    public Phone(String name, int phone) {
      this.name = name;
      this.phone = phone;
    }
  }

  public static void Main(String[] args) {
    // Basic rules of type dynamic
    {
      dynamic d1 = 34;
      int i1 = d1 * 2;                  // OK: cast (int)(d1*2) at run-time
      int i2 = (int)d1 * 2;             // OK: cast (int)d1 at run-time
      // bool b1 = d1;                  // Compiles OK; cast (bool)d1 throws at run-t
ime
      d1 = true;                        // OK
      bool b2 = d1;                     // OK: cast (bool)d1 succeeds at run-time
      dynamic p1 = new Phone("Kasper", 5170);
      String s1 = p1.name;              // Field access checked at run-time
      // int n1 = p1.age;               // Compiles OK; field access throws at run-ti
me
      dynamic p2 = new { name = "Kasper", phone = 5170 };
      String s2 = p2.name;              // Field access checked at run-time
      // int n2 = p2.age;               // Compiles OK; fields access throws at run-t
ime
    }

    // Dynamic operator resolution; run-time type determines meaning of "+" in Plus2(
)
    {
      Console.WriteLine(Plus2(int.MaxValue-1));             // -2147483648, due to i
nt overflow
      Console.WriteLine(Plus2((long)(int.MaxValue-1)));     //  2147483648, no long
overflow
      Console.WriteLine(Plus2(11.5));                       // 13.5
      Console.WriteLine(Plus2("Spar"));                    // Spar2
      // Console.WriteLine(Plus2(false));       // Compiles OK; throws RuntimeBinder
Exception
    }

    // Dynamic receiver; run-time type determines whether to call Length on array or
String
    {
      dynamic v;
      if (args.Length==0)      v = new int[] { 2, 3, 5, 7 };
      else                     v = "abc";
      int res = v.Length;
      Console.WriteLine(res);
    }

    // Dynamic overload resolution; run-time type of v determines which Process calle
d at (**)
    {
      dynamic v;
      if (args.Length==0)      v = 5;
      else if (args[0] == "1") v = "abc";
      else                     v = (Func<int,int>)(x => x*3);
      dynamic r = Process(v);                                    // (**)
      if (args.Length==0 || args[0] == "1")
        Console.WriteLine(r);
      else
        Console.WriteLine(r(11));
      dynamic s = "abc";
      Console.WriteLine(Process(s).StartsWith("abca"));
     }

      // Run-time type tests
    {
```

```
      Console.WriteLine(Square(5));
      Console.WriteLine(Square("abc"));
      Func<int,int> f = x => x*3;
      Console.WriteLine(Square(f)(11));
    }

    {
      // Types dynamic[], List<dynamic>, IEnumerable<dynamic>
      dynamic[] arr = new dynamic[] { 19, "Electric", (Func<int,int>)(n => n+2), 3.2, f
alse };
      int number = arr[0] * 5;
      String street = arr[1].ToUpper();
      int result = arr[2](number);
      Console.WriteLine(number + " " + street);
      double sum = 0;
      List<dynamic> list = new List<dynamic>(arr);
      IEnumerable<dynamic> xs = list;
      foreach (dynamic x in xs)
        if (x is int || x is double)
          sum += x;
      Console.WriteLine(sum);                                    // 22.2
    }

    // Dynamic and anonymous object expressions
    {
      dynamic v = new { x = 34, y = false };
      Console.WriteLine(v.x);
    }
  }

  // Run-time type of v determines meaning of "+": int+, long+, double+, String+, ...
  static dynamic Plus2(dynamic v) { return v + 2; }

  // Ordinary overloaded methods
  static int Process(int v) { return v * v; }

  static double Process(double v) { return v * v; }

  static String Process(String v) { return v + v; }

  static Func<int,int> Process(Func<int,int> v) { return x => v(v(x)); }

  // Explicit tests on run-time type.  Using "dynamic" rather than
  // "Object" here means that the compiler accepts the (v * v)
  // expression and the application of v to arguments, and that the
  // value returned by Square can be further processed: added to,
  // applied to arguments, and so on.
  static dynamic Square(dynamic v) {
    if (v is int || v is double)
      return v * v;
    else if (v is String)
      return v + v;
    else if (v is Func<int,int>)
      return (Func<int,int>)(x => v(v(x)));
    else
      throw new Exception("Don't know how to square " + v);
  }
}

class C : List<dynamic> { }
```

```
// Example 103 from page 85 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

// A log of Strings that retains only the last SIZE logged Strings

public class Log {
  private const int SIZE = 5;
  private static int instanceCount = 0;
  private int count = 0;
  private String[] log = new String[SIZE];

  public Log() {
    instanceCount++;
  }

  // The number of Logs created

  public static int InstanceCount {
    get { return instanceCount; }
  }

  // Add a String to this Log

  public void Add(String msg) {
    log[count++ % SIZE] = msg;
  }

  // Property giving the number of strings inserted in this Log

  public int Count {
    get { return count; }
  }

  // The most recently logged string, if any

  public String Last {
    get { // Return the last log entry, or null if nothing logged yet
      return count==0 ? null : log[(count-1)%SIZE];
    }
    set { // Update the last log entry, or create one if nothing logged yet
      if (count==0)
        log[count++] = value;
      else
        log[(count-1)%SIZE] = value;
    }
  }

  // Return all log entries

  public String[] All {
    get {
      int size = Math.Min(count, SIZE);
      String[] res = new String[size];
      for (int i=0; i<size; i++)
        res[i] = log[(count-size+i) % SIZE];
      return res;
    }
  }
}

class TestLog {
  public static void Main(String[] args) {
    Log log1 = new Log(), log2 = new Log();
    Console.WriteLine("Number of logs = " + Log.InstanceCount);
    log1.Add("Alarm"); log1.Add("Shower"); log1.Add("Coffee");
    log1.Add("Bus"); log1.Add("Work"); log1.Add("Lunch");
    Console.WriteLine(log1.Last);
    log1.Last += " nap";
    Console.WriteLine(log1.Last);
    log1.Add("More work");
    Console.WriteLine("Logged entries = " + log1.Count);
    foreach (String s in log1.All)
      Console.WriteLine(s);
  }
}
```

```
// Example 104 from page 85 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

abstract class A {
  public abstract String Cl { get; }
}

class B : A {
  public static String Name { get { return "B"; } }
  public String Ty { get { return "B"; } }
  public override String Cl { get { return "B"; } }
}

class C : B {
  public new static String Name { get { return "C"; } }
  public new String Ty { get { return "C:" + base.Ty; } }
  public override String Cl { get { return "C:" + base.Cl; } }
}

class TestProperty {
  public static void Main(String[] args) {
    B b1 = new B();
    C c2 = new C();
    B b2 = c2;
    Console.WriteLine("B.Name = {0}, C.Name = {1}", B.Name, C.Name);
    Console.WriteLine("b1.Ty = {0}, b2.Ty = {1}, c2.Ty = {2}", b1.Ty, b2.Ty, c2.Ty);
    Console.WriteLine("b1.Cl = {0}, b2.Cl = {1}, c2.Cl = {2}", b1.Cl, b2.Cl, c2.Cl);
  }
}
```

```
// Example 105 from page 87 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;
using System.Text;

public class SparseMatrix {
  private readonly int rows;
  // A SparseMatrix has an array of lists of NonZeros, one for each column.
  // Invariant: In each list the nonzeros appear in increasing order of nz.i
  readonly List<NonZero>[] cols;

  // Create a sparse matrix from 2D array B which must be rectangular

  public SparseMatrix(double[][] B) {
    cols = new List<NonZero>[B.Length];
    rows = B.Length != 0 ? B[0].Length : 0;
    for (int j=0; j<B.Length; j++) {
      cols[j] = new List<NonZero>();
      for (int i=0; i<rows; i++)
        if (B[i][j] != 0.0)
          cols[j].Add(new NonZero(i, B[i][j]));
    }
  }

  // Create an all-zero rows-by-cols sparse matrix

  public SparseMatrix(int r, int c) {
    cols = new List<NonZero>[c];
    this.rows = r;
    for (int j=0; j<c; j++)
      cols[j] = new List<NonZero>();
  }

  // Properties to get the number of rows and columns of the matrix

  public int Rows {
    get { return rows; }
  }

  public int Cols {
    get { return cols.Length; }
  }

  // Indexer to get and set an element of the matrix

  public double this[int i, int j] {
    get {
      List<NonZero> colj = this[j];
      int k = 0;
      while (k < colj.Count && colj[k].i < i)
        k++;
      return k < colj.Count && colj[k].i == i ? colj[k].Mij : 0.0;
    }
    set {
      List<NonZero> colj = this[j];
      int k = 0;
      while (k < colj.Count && colj[k].i < i)
        k++;
      if (k < colj.Count && colj[k].i == i)
        colj[k].Mij = value;
      else if (value != 0.0)
        colj.Insert(k, new NonZero(i, value));
    }
  }

  // Indexer to get j'th column of matrix

  private List<NonZero> this[int j] {
    get { return cols[j]; }
  }

  // A pair of a row number i and a non-zero element B[i][-]

  private class NonZero {
    public readonly int i;
    public double Mij;
```

```
    public NonZero(int i, double Mij) {
      this.i = i; this.Mij = Mij;
    }

    public NonZero(NonZero nz) {
      this.i = nz.i; this.Mij = nz.Mij;
    }
  }

  public static SparseMatrix Add(SparseMatrix A, SparseMatrix B) {
    if (A.Rows == B.Rows && A.Cols == B.Cols) {
      int rRows = A.Rows, rCols = A.Cols;
      SparseMatrix R = new SparseMatrix(rRows, rCols);
      for (int j=0; j<rCols; j++) {
        List<NonZero> Aj = A[j], Bj = B[j], Rj = R[j];
        int ak = 0, bk = 0;
        while (ak<Aj.Count && bk<Bj.Count) {
          if (Aj[ak].i < Bj[bk].i)
            Rj.Add(new NonZero(Aj[ak++]));
          else if (Bj[bk].i < Aj[ak].i)
            Rj.Add(new NonZero(Bj[bk++]));
          else // Aj[ak].i==Bj[bk].i
            Rj.Add(new NonZero(Aj[ak].i, Aj[ak++].Mij+Bj[bk++].Mij));
        }
        while (ak<Aj.Count)
          Rj.Add(new NonZero(Aj[ak++]));
        while (bk<Bj.Count)
          Rj.Add(new NonZero(Bj[bk++]));
      }
      return R;
    } else
      throw new ApplicationException("SparseMatrix.Add: Matrix size misfit");
  }

  public override String ToString() {
    StringBuilder sb = new StringBuilder();
    for (int i=0; i<Rows; i++) {
      for (int j=0; j<Cols; j++)
        sb.AppendFormat("{0,6} ", this[i,j]);
      sb.Append("\n");
    }
    return sb.ToString();
  }
}

class TestSparseMatrix {
  public static void Main(String[] args) {
    SparseMatrix A = new SparseMatrix(4, 5), B = new SparseMatrix(4, 5);
    A[0,2] = 102; A[0,3] = 103; A[1,0] = 110; A[3,4] = 134;
    B[0,2] = 202; B[1,3] = 213; B[2,0] = 220; B[3,4] = 234;
    Console.WriteLine("A =\n{0}", A);
    Console.WriteLine("B =\n{0}", B);
    Console.WriteLine("A+B =\n{0}", SparseMatrix.Add(A,B));
  }
}
```

```
// Example 106 from page 87 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using SC = System.Collections;

class MyTest {
  public static void Main(String[] args) {
    StringList ss = new StringList();
    ss.Add("Cop"); ss.Add("en"); ss.Add("cabana");
    ss[2] = "hagen";
    ss[0] += "en" + ss[2];
    Console.WriteLine("A total of {0} strings", ss.Count);
    String last = ss[2];                          // Correct type
    Console.WriteLine(ss["0"] + "/" + last);    // Prints: Copenhagen/hagen
  }
}

class StringList : SC.ArrayList {               // Needs: using SC = System.Collections
;
  public new String this[int i] {
    get { return (String)base[i]; }
    set { base[i] = value; }
  }
  public String this[String s] {
    get { return this[int.Parse(s)]; }
  }
}
```

```
// Example 107 from page 89 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class B { }
class C : B {
  private String s;
  public C(String s) { this.s = s; }
  public static explicit operator C(String s) { return new C(s + s); }
}

class MyTest {
  public static void Main(String[] args) {
    int i = 4711;
    long ll = (byte)i + (long)i;        // Simple type conversions
    String s = "ole";
    B b1 = new C("foo"), b2 = new B();
    C c1 = (C)b1;                       // Succeeds, b1 has class C
    C c2 = (C)b2;                       // Fails, b2 has class B
    C c3 = (C)s;                        // User-defined conversion String-->C
    Object o = (Object)s;               // Always succeeds
    C c4 = (C)(String)o;                // Succeeds, Object-->String-->C
    C c5 = (C)o;                        // Fails, no Object-->C conversion
    // Array arr = (Array)s;            // Rejected at compile-time
  }
}
```

```
// Example 108 from page 89 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

interface I { }
class B { }
class C : B, I { }
delegate int D(int i);
struct S : I { }
class G<T> {
  public static void WriteType() {
    Console.WriteLine(typeof(T));
  }
}

class MyTest {
  public static void Main(String[] args) {
    Console.WriteLine(typeof(String));              // System.String
    Console.WriteLine(typeof(int));                 // System.Int32  (int)
    Console.WriteLine(typeof(double));              // System.Double (double)
    Console.WriteLine(typeof(int[]));               // System.Int32[]
    Console.WriteLine(typeof(int[][]));             // System.Int32[][]
    Console.WriteLine(typeof(int[,]));              // System.Int32[,]
    Console.WriteLine(typeof(void));                // System.Void
    Console.WriteLine(typeof(B));                   // B
    Console.WriteLine(typeof(C));                   // C
    Console.WriteLine(typeof(I));                   // I
    Console.WriteLine(typeof(D));                   // D
    Console.WriteLine(typeof(S));                   // S
    Console.WriteLine(typeof(G<int>));              // G[System.Int32]
    Console.WriteLine(typeof(G<String>));           // G[System.String]
    G<int>.WriteType();                             // System.Int32
    Console.WriteLine(typeof(int)==typeof(Int32));  // True
  }
}
```

```
// Example 109 from page 91 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

delegate int D1(int x, ref int y, out int z);
delegate int D2(int x, ref int y);
delegate int D3(int x);
delegate void D4(int x, ref int y);
class Test {
  static D1 d11 = delegate(int x, ref int y, out int z) { z = y++; return x + y; };
  static D2 d21 = delegate(int x, ref int y) { y+=2; return x + y; };
  static D2 d22 = delegate { return 5; };
  public static D2 M(int mx) {
    if (mx < 6)
      return delegate(int x, ref int y) { y+=2; return x + y; };
    else
      return delegate { return mx; };
  }
  public static void Main(String[] args) {
    D2[] ds = { d21, d22, M(4), M(7), delegate { return 8; } };
    int y = 0;
    foreach (D2 d in ds)
      Console.WriteLine(d(2, ref y));   // Prints 4 5 6 7 8
    Console.WriteLine(y);               // Prints 4
  }
}
```

```
// Example 110 from page 91 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Threading;

class MyTest {
  public static void Main(String[] args) {
    int v = 0;
    (new Thread(new ThreadStart(delegate {
      Console.Write(v++);
      Thread.Sleep(0);
    }))).Start();
    (new Thread(new ThreadStart(delegate {
      Console.Write(v--);
      Thread.Sleep(0);
    }))).Start();
    Console.WriteLine();
    Console.WriteLine("\nv = {0}", v);
  }
}

public delegate void D();
```

```
// Example 111 from page 93 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;
using System.Linq;

class MyTest {
  public static void Main(String[] args) {
    String[] animals = { "cat", "elephant", "dog", "fox", "squirrel" };
    foreach (String animal in animals.Where(s => s.Length==3))
      Console.WriteLine(animal);
    Func<int,int> f1 = x => x * x;
    Func<double,double> f2 = x => x * x;
    Console.WriteLine(f1(5));
    Console.WriteLine(f2(0.5));
    Func<int,int,int> f3 = (x, y) => x+y;             // Uncurried, call as f3(11, 22
)
    Func<int,Func<int,int>> f4 = x => y => x+y;       // Curried, call as f4(11)(22)
    Console.WriteLine(f3(11, 22));
    Console.WriteLine(f4(11)(22));
    var f5 = f4(11);                                  // The function that adds 11
    Console.WriteLine(f5(22));
    // var bad1 = (int x) => x * x;                   // Illegal, cannot infer type
    // Object bad2 = (int x) => x * x;                // Illegal, not delegate type
    // dynamic bad3 = (int x) => x * x;               // Illegal, not delegate type
    // var bads = new [] { (int x) => x * x };        // Illegal, no best type
    Func<int,double> fib1 = null;
    fib1 = n => n < 2 ? 1 : fib1(n-1) + fib1(n-2);
    for (int i=0; i<39; i++)
      Console.Write(fib1(i) + " ");
    Console.WriteLine();
    Func<int,double> fib2 = Recursive<int,double>(fib => n => n < 2 ? 1 : fib(n-1) +
fib(n-2));
    for (int i=0; i<39; i++)
      Console.Write(fib2(i) + " ");
    Console.WriteLine();
    Func<int,double> fib3 = RecursiveMemoize<int,double>(fib => n => n < 2 ? 1 : fib(
n-1) + fib(n-2));
    for (int i=0; i<39; i++)
      Console.Write(fib3(i) + " ");
    Console.WriteLine();
  }

  public static Func<A,R> Recursive<A,R>(Func<Func<A,R>,Func<A,R>> protoF) {
    Func<A,R> f = null;
    return f = protoF(x => f(x));
  }

  public static Func<A,R> RecursiveMemoize<A,R>(Func<Func<A,R>,Func<A,R>> protoF) whe
re A : IEquatable<A> {
    var memoTable = new Dictionary<A,R>();
    Func<A,R> f = null;
    return f = protoF(x => memoTable.ContainsKey(x) ? memoTable[x] : memoTable[x] = f
(x));
  }

  public static Func<A,C> Compose<A,B,C>(Func<B,C> f, Func<A,B> g) {
    return x => f(g(x));
  }

  public static Func<A,Func<B,C>> Curry<A,B,C>(Func<A,B,C> f) {
    return x => y => f(x,y);
  }

  public static Func<A,B,C> UnCurry<A,B,C>(Func<A,Func<B,C>> f) {
    return (x, y) => f(x)(y);
  }
}
```

```
// Example 112 from page 93 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static void Main(String[] args) {
    double z = 3.14;
    var p1 = new { x = 13, y = "foo" };         // Fields: x, y
    var p2 = new { x = 42, p1.y, z };           // Fields: x, y, z
    var p3 = new { };                           // Fields: none
    int sum = p1.x + p2.x + (int)p2.z + p2.y.Length;
    Console.WriteLine(sum);
    Console.WriteLine(p1);
    Console.WriteLine(p2);
    Console.WriteLine(p3);
  }
}
```

```
// Example 113 from page 93 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static void Main(String[] args) {
    var ds = new [] { 2, 3, 5.0, 7 };                              // Type double[]
    Console.WriteLine(ds.GetType());

    var r1 = new [,] { { 0.0, 0.1 }, { 1.0, 1.1 }, { 2.0, 2.1 } };  // Type double[,]
    Console.WriteLine(r1.GetType());

    var arr = new [] { new { n = 22, r = "XXII" }, new { n = 5, r = "V" } };
    int sum = arr[0].n + arr[1].n;
    Console.WriteLine(sum);
  }
}
```

```
// Example 114 from page 95 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// All the ways a statement may terminate: normally, throw exception,
// return, etc.

using System;

class Example114 {
  public static void Main(String[] args) {
    if (args.Length != 1)
      Console.WriteLine("Usage: Example114 <integer 1−7>\n");
    else
      Statement(int.Parse(args[0]));
  }

  public static void Statement(int choice) {
    bool again = true;
    while (Again(again)) {
      again = !again;
      if (choice == 1)                    // Terminate normally
        Console.WriteLine("Choice 1");
      else if (choice == 2)               // Throw exception
        throw new Exception();
      else if (choice == 3)               // Return from method
        return;
      else if (choice == 4)               // Break out of loop
        break;
      else if (choice == 5)               // Continue at loop test
        continue;
      else if (choice == 6)               // Jump out of loop
        goto end;
      else                                // Loop forever
        while (true) { }
      Console.WriteLine("At end of loop");
    }
    Console.WriteLine("After loop");
  end:
    Console.WriteLine("At end of method");
  }

  private static bool Again(bool again) {
    Console.WriteLine("Loop test");
    return again;
  }
}
```

```
// Example 116 from page 95 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class VariableDeclExample {
  public static void Main(String[] args) {
    int a;
    const int year = 365, week = 7, weekMax = year / week + 1;
    Console.WriteLine(weekMax);
    int x, y = year, z = 3, ratio = z/y;
    const double pi = 3.141592653589;
    bool found = false;
    var stillLooking = true;
    a = x = y;
    if (!found || stillLooking)
      Console.WriteLine(a + x + y + z + ratio + pi);
  }
}
```

```
// Example 118 from page 97 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class If1 {
  public static void Main(String[] args) {
    Console.WriteLine(Absolute(-12));
    Console.WriteLine(Absolute(12));
    Console.WriteLine(Absolute(0));
  }

  static double Absolute(double x) {
    if (x >= 0)
      return x;
    else
      return -x;
  }
}
```

```
// Example 119 from page 97 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class Example119 {
  public static void Main(String[] args) {
    if (args.Length != 1)
      Console.WriteLine("Usage: Example119 <age>\n");
    else
      Console.WriteLine(AgeGroup(int.Parse(args[0])));
  }

  static String AgeGroup(int age) {
    if (age <= 12)      return "child";
    else if (age <= 19) return "teenager";
    else if (age <= 45) return "young";
    else if (age <= 60) return "middle-age";
    else                return "old";
  }
}
```

```
// Example 120 from page 97 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class FindingCountry {
  public static void Main(String[] args) {
    Console.WriteLine("44 is " + FindCountry(44));
  }

  static String FindCountry(int prefix) {
    switch (prefix) {
    default:  return "Unknown";
    case 1:   return "North America";
    case 44:  return "Great Britain";
    case 45:  return "Denmark";
    case 299: return "Greenland";
    case 46:  return "Sweden";
    case 7:   return "Russia";
    case 972: return "Israel";
    }
  }
}
```

```
// Example 121 from page 99 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class LoopExample1 {
  public static void Main(String[] args) {
    for (int i=1; i<=4; i++) {                  // Output:
      for (int j=1; j<=i; j++)                  // *
        Console.Write("*");                     // **
      Console.WriteLine();                      // ***
    }                                           // ****
  }
}
```

```
// Example 122 from page 99 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Reversing an array of strings

using System;

class MyTest {
  public static void Main(String[] args) {
    Reverse(args);
    Console.WriteLine("Reversed input:");
    Console.WriteLine("--------------------");
    foreach (String s in args)
      Console.WriteLine(s);
    Console.WriteLine("--------------------");
  }

  public static void Reverse(Object[] arr) {
    for (int s=0, t=arr.Length-1; s<t; s++, t--) {
      Object tmp = arr[s]; arr[s] = arr[t]; arr[t] = tmp;
    }
  }
}
```

```
// Example 123 from page 99 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Text;

class ForeachArray {
  public static void Main(String[] args) {
    String[] arr =  { "foo", "bar", "", "baz", "" };
    Console.WriteLine(ConcatenateBracketed(arr));
  }

  // Using foreach to iterate over an array

  static String ConcatenateBracketed(String[] arr) {
    StringBuilder sb = new StringBuilder();
    foreach (String s in arr)
      sb.Append(s).Append(s);
    return sb.ToString();
  }
}
```

```
// Example 124 from page 99 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Text;
using System.Collections;

class ForeachExpanded {
  public static void Main(String[] args) {
    String[] arr =  { "foo", "bar", "", "baz", "" };
    Console.WriteLine(ConcatenateBracketed(arr));
  }

  // Using an explicit enumerator instead of foreach

  static String ConcatenateBracketed(String[] arr) {
    StringBuilder sb = new StringBuilder();
    IEnumerator enm = arr.GetEnumerator();
    try {
      while (enm.MoveNext()) {
        String s = (String)enm.Current;
        sb.Append(s).Append(s);
      }
    } finally {
      Console.WriteLine("(now in finally block)");
      IDisposable disp = enm as System.IDisposable;
      if (disp != null)
        disp.Dispose();
    }
    return sb.ToString();
  }
}
```

```
// Example 125 from page 101 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class WdaynoWhile {
  public static void Main(String[] args) {
    Console.WriteLine("Thursday is " + WeekDayNo1("Thursday"));
  }

  static int WeekDayNo1(String wday) {
    int i=0;
    while (i < wdays.Length && wday != wdays[i])
      i++;
    // Now i >= wdays.Length or wday == wdays[i]
    if (i < wdays.Length) return i+1;
    else                  return -1;           // Here used to mean 'not found'
  }

  static readonly String[] wdays =
  { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday" };
}
```

```
// Example 126 from page 101 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class Example126 {
  public static void Main(String[] args) {
    if (args.Length != 1)
      Console.WriteLine("Usage: Example126 <string>\n");
    else {
      String q = args[0];
      Console.WriteLine(q + " substring of hjsdfk: " + Substring1(q, "hjsdfk"));
    }
  }

  // Decide whether query is a substring of target (using for and while);
  // recommended

  static bool Substring1(String query, String target) {
    for (int j=0, n=target.Length-query.Length; j<=n; j++) {
      int k=0, m=query.Length;
      while (k<m && target[j+k] == query[k])
        k++;
      // Now k>=m (and target[j..]==query[0..m-1]) or target[j+k] != query[k]
      if (k>=m)
        return true;
    }
    return false;
  }
}
```

```
// Example 127 from page 101 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class LoopExample3 {
  public static void Main(String[] args) {
    Console.WriteLine("Infinite loop!  Stop it by pressing ctrl−C\n\n");
    int i=0;
    while (i<10);
      i++;
  }
}
```

```
// Example 128 from page 101 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class LoopExample4 {
  public static void Main(String[] args) {
    Console.WriteLine("Counting sum of eyes until 5 or 6 comes up (10000 dice).");
    int[] wait = new int[1000];
    for (int i=0; i<10000; i++)
      wait[WaitSum()]++;
    Console.WriteLine("sum: frequency");
    for (int w=5; w<20; w++)
      Console.WriteLine(w + ": " + wait[w]);
  }

  private static readonly Random rnd = new Random();

  // Roll a die and compute sum until five or six comes up
  static int WaitSum() {
    int sum = 0, eyes;
    do {
      eyes = 1 + rnd.Next(6);
      sum += eyes;
    } while (eyes < 5);
    return sum;
  }
}
```

```
// Example 129 from page 103 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static void Main(String[] args) {
    Print("Hello!");
  }

  public static void Print(String s) {
    Console.WriteLine(s);
    return;
  }
}
```

```
// Example 130 from page 103 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class LoopExample5 {
  public static void Main(String[] args) {
    Console.WriteLine("Thursday is " + WeekDayNo3("Thursday"));
  }

  static int WeekDayNo3(String wday) {
    for (int i=0; i < wdays.Length; i++)
      if (wday.Equals(wdays[i]))
        return i+1;
    return -1;                                  // Here used to mean 'not found'
  }

  static readonly String[] wdays =
  { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday" };
}
```

```
// Example 131 from page 103 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using SC = System.Collections;

class BreakForeach {
  public static void Main(String[] args) {
    String[] arr =  { "foo", "", "bar", "baz", "" };
    SearchNonBlank1(arr);
    SearchNonBlank2(arr);
  }

  // Using break to exit the loop as soon as an empty string is found

  static void SearchNonBlank1(String[] arr) {
    bool found = false;
    foreach (String s in arr)
      if (s == "") {
        found = true;
        break;
      }
    Console.WriteLine(found);
  }

  // A solution with while instead of foreach and break is more cumbersome.

  // Note that method GetEnumerator on an array type returns a
  // non-generic IEnumerator, and that the cast to String in
  // ((String)enm.Current == "") is necessary to obtain a string
  // comparison; otherwise it will be a reference comparison.

  static void SearchNonBlank2(String[] arr) {
    bool found = false;
    SC.IEnumerator enm = arr.GetEnumerator();
    while (!found && enm.MoveNext())
      found = (String)enm.Current == "";
    Console.WriteLine(found);
  }
}
```

```
// Example 132 from page 103 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class ContinueForeach {
  public static void Main(String[] args) {
    String[] arr =  { "foo", "", "bar", "baz", "" };
    PrintNonBlank3(arr);
  }

  // Using continue to skip empty strings when printing

  static void PrintNonBlank3(String[] arr) {
    Console.WriteLine("---------");
    foreach (String s in arr) {
      if (s == "")
        continue;
      Console.WriteLine(s);
    }
    Console.WriteLine("---------");
  }
}
```

```
// Example 133 from page 105 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class Example133 {
  public static void Main(String[] args) {
    if (args.Length != 1)
      Console.WriteLine("Usage: Example133 <string>\n");
    else {
      String q = args[0];
      Console.WriteLine(q + " substring of hjsdfk: " + Substring1(q, "hjsdfk"));
    }
  }

  // Decide whether query is a substring of target (using goto)

  static bool Substring1(String query, String target) {
    for (int j=0, n=target.Length-query.Length; j<=n; j++) {
      for (int k=0, m=query.Length; k<m; k++)
        if (target[j+k] != query[k])
          goto nextPos;
      return true;
    nextPos: { }               // Label on empty statement
    }
    return false;
  }
}
```

```
// Example 134 from page 105 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class ExceptionExamples {
  public static void Main(String[] args) {
    try {
      Console.WriteLine(args[0] + " is weekday number " + WeekDayNo4(args[0]));
    } catch (WeekdayException x) {
      Console.WriteLine("Weekday problem: " + x);
    } catch (Exception x) {
      Console.WriteLine("Other problem: " + x);
    }
  }

  // Behaves the same as wdayno3, but throws Exception instead of
  // returning bogus weekday number:
  static int WeekDayNo4(String wday) {
    for (int i=0; i < wdays.Length; i++)
      if (wday.Equals(wdays[i]))
        return i+1;
    throw new WeekdayException(wday);
  }

  static readonly String[] wdays =
  { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday" };
}

class WeekdayException : ApplicationException {
  public WeekdayException(String wday) : base("Illegal weekday: " + wday) {
  }
}
```

```
// Example 135 from page 105 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// A finite state machine recognizing the regular expression (a|b)*abb
// from Aho, Sethi, Ullman: Compilers, Principles, Techniques, and
// Tools. Addison-Wes;ey 1986 page 136.

using System;

class Example135 {
  public static void Main(String[] args) {
    if (args.Length != 1)
      Console.WriteLine("Usage: Example135 <string>\n");
    else
      Console.WriteLine(Match(args[0]) ? "Success" : "Failure");
  }

  public static bool Match(String str) {
    int stop = str.Length, i = 0;
  state1:
    if (i==stop) return false;
    switch (str[i++]) {
    case 'a': goto state2;
    case 'b': goto state1;
    default: return false;
    }
  state2:
    if (i==stop) return false;
    switch (str[i++]) {
    case 'a': goto state2;
    case 'b': goto state3;
    default: return false;
    }
  state3:
    if (i==stop) return false;
    switch (str[i++]) {
    case 'a': goto state2;
    case 'b': goto state4;
    default: return false;
    }
  state4:
    if (i==stop) return true;
    switch (str[i++]) {
    case 'a': goto state2;
    case 'b': goto state1;
    default: return false;
    }
  }
}
```

```
// Example 136 from page 107 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class ExceptionExamples {
  public static void Main(String[] args) {
    try {
      Console.WriteLine(args[0] + " is weekday number " + WeekDayNo4(args[0]));
    } catch (WeekdayException x) {
      Console.WriteLine("Weekday problem: " + x);
    } catch (Exception x) {
      Console.WriteLine("Other problem: " + x);
    }
  }

  // Behaves the same as wdayno3, but throws Exception instead of
  // returning bogus weekday number:
  static int WeekDayNo4(String wday) {
    for (int i=0; i < wdays.Length; i++)
      if (wday.Equals(wdays[i]))
        return i+1;
    throw new WeekdayException(wday);
  }

  static readonly String[] wdays =
  { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday" };
}

class WeekdayException : ApplicationException {
  public WeekdayException(String wday) : base("Illegal weekday: " + wday) {
  }
}
```

```
// Example 137 from page 107 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.IO;                // StreamReader, TextReader

class TryFinally {
  public static void Main(String[] args) {
    double[] xs = ReadRecord("foo");
    for (int i=0; i<xs.Length; i++)
      Console.WriteLine(xs[i]);
  }

  static double[] ReadRecord(String filename) {
    TextReader reader = new StreamReader(filename);
    double[] res = new double[3];
    try {
      res[0] = double.Parse(reader.ReadLine());
      res[1] = double.Parse(reader.ReadLine());
      res[2] = double.Parse(reader.ReadLine());
    } finally {
      reader.Close();
    }
    return res;
  }
}
```

```
// Example 138 from page 109 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class CheckedUncheckedStatements {
  public static void Main() {
    String big = "9999999999";              // 9999999999 > int.MaxValue

    checked {
      Console.WriteLine(int.MaxValue + 1);   // Compile-time error
      Console.WriteLine(int.MinValue - 1);   // Compile-time error
      Console.WriteLine((uint)(0-1));        // Compile-time error
      int i = int.Parse("9999999999");       // Throws OverflowException
    }
    unchecked {
      Console.WriteLine(int.MaxValue + 1);   // -2147483648 (wrap-around)
      Console.WriteLine(int.MinValue - 1);   // 2147483647  (wrap-around)
      Console.WriteLine((uint)(0-1));        // 4294967295  (wrap-around)
      int i = int.Parse("9999999999");       // Throws OverflowException
    }
  }
}
```

```
// Example 139 from page 109 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.IO;                // StreamReader, TextReader

class TestUsing {
  public static void Main(String[] args) {
    double[] xs = ReadRecord("foo");
    for (int i=0; i<xs.Length; i++)
      Console.WriteLine(xs[i]);
  }

  static double[] ReadRecord(String filename) {
    using (TextReader reader = new StreamReader(filename)) {
      double[] res = new double[3];
      res[0] = double.Parse(reader.ReadLine());
      res[1] = double.Parse(reader.ReadLine());
      res[2] = double.Parse(reader.ReadLine());
      return res;
    }
  }
}
```

```
// Example 140 from page 111 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Overloaded operators

using System;                           // For Console, String
using System.Text;                      // For StringBuilder
using SC = System.Collections;          // For IEnumerator, IEnumerable
using System.Collections.Generic;       // For IEnumerator<T>

struct Seq : ISeq {
  private readonly int b, k, n;                     // Sequence b+k*[0..n-1]

  // Default constructor Seq() creates an empty sequence with n=0

  public Seq(int m, int n) : this(m, 1, n-m+1) { }   // Sequence [m..n]

  public Seq(int b, int k, int n) {
    this.b = b; this.k = k; this.n = n;
  }

  // Add b to sequence
  public static Seq operator +(int b, Seq seq) {
    return new Seq(seq.b+b, seq.k, seq.n);
  }

  // Add b to sequence
  public static Seq operator +(Seq seq, int b) {
    return new Seq(seq.b+b, seq.k, seq.n);
  }

  // Multiply all members of the sequence by k
  public static Seq operator *(int k, Seq seq) {
    return new Seq(seq.b*k, seq.k*k, seq.n);
  }

  // Multiply all members of the sequence by k
  public static Seq operator *(Seq seq, int k) {
    return new Seq(seq.b*k, seq.k*k, seq.n);
  }

  // Reverse the sequence
  public static Seq operator !(Seq seq) {
    return new Seq(seq.b+(seq.n-1)*seq.k, -seq.k, seq.n);
  }

  // Equality and inequality
  public static bool operator ==(Seq s1, Seq s2) {
    return s1.n==s2.n && (s1.n==0 || s1.b==s2.b && (s1.n==1 || s1.k==s2.k));
  }

  public static bool operator !=(Seq s1, Seq s2) {
    return !(s1==s2);
  }

  public override bool Equals(Object that) {
    return that is Seq && this==(Seq)that;
  }

  public override int GetHashCode() {
    return n==0 ? 0 : n==1 ? b : b^k^n;
  }

  // Get enumerator for the sequence
  public IEnumerator<int> GetEnumerator() {
    for (int i=0; i<n; i++) {
      yield return b + k * i;
    }
  }

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

  public int Count {
    get { return n; }
  }

  public int this[int i] {
```

```
    get {
      if (0 <= i && i < n)
        return b + k * i;
      else
        throw new ArgumentOutOfRangeException("Seq indexer: " + i);
    }
  }

  public void Print() {
    IEnumerator<int> etor = GetEnumerator();
    while (etor.MoveNext())
      Console.Write(etor.Current + " ");
  }

  public override String ToString() {
    StringBuilder sb = new StringBuilder();
    foreach (int i in this)
      sb.Append(i).Append(" ");
    return sb.ToString();
  }
}

class TestSeq {
  public static void Main(String[] args) {
    Seq s1 = new Seq(1, 3);             // 1 2 3
    Seq s2 = 2 * s1 + 5;                // 7 9 11
    Seq s3 = s2 * 3;                    // 21 27 33
    Seq s4 = !s3;                       // 33 27 21
    Console.WriteLine(s1);
    Console.WriteLine(s2);
    Console.WriteLine(s3);
    Console.WriteLine(s4);
    s4.Print();
    Console.WriteLine();
    for (int i=0, stop=s4.Count; i<stop; i++)
      Console.Write(s4[i] + " ");
    Console.WriteLine();
  }
}

interface ISeq : IEnumerable<int> {
  int Count { get; }
  int this[int i] { get; }
}
```

```
// Example 141 from page 111 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

public class Example141 {
  public static void Main(String[] args) {
    if (args.Length != 1)
      Console.WriteLine("Usage: Example141 <queencount>\n");
    else {
      int n = int.Parse(args[0]);
      foreach (int[] sol in Queens(n-1, n)) {
        foreach (int r in sol)
          Console.Write("{0} ", r);
        Console.WriteLine();
      }
    }
  }

  // A result from the ienumerable produced by Queens(w, n) is an int
  // array whose columns 0..w contain a partial solution to the
  // n-queens problem: w+1 queens have been safely placed in the w+1
  // first columns.  It follows that a result of Queens(n-1, n) is a
  // solution to the n-queens problem.

  public static IEnumerable<int[]> Queens(int w, int n) {
    if (w < 0)
      yield return new int[n];
    else
      foreach (int[] sol in Queens(w-1, n))
        for (int r=1; r<=n; r++) {
          for (int c=0; c<w; c++)
            if (sol[c] == r || sol[c]+(w-c) == r || sol[c]-(w-c) == r)
              goto fail;
          sol[w] = r;
          yield return sol;
          fail: { }
        }
  }
}
```

```
// Example 142 from page 113 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public struct SPoint {
  internal int x, y;

  public SPoint(int x, int y) { this.x = x; this.y = y; }

  public SPoint Move(int dx, int dy) { x += dx; y += dy; return this; }

  public override String ToString() { return "(" + x + "," + y + ")"; }
}

class Dummy {
  public static void Main() { }
}
```

```
// Example 143 from page 113 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public struct SPoint {
  internal int x, y;

  public SPoint(int x, int y) { this.x = x; this.y = y; }

  public void Move(int dx, int dy) { x += dx; y += dy; }

  public override String ToString() { return "(" + x + "," + y + ")"; }
}

class MyTest {
  public static void Main(String[] args) {
    SPoint p = new SPoint(11, 22);             // Create a struct value in p
    SPoint[] arr = { p, p };                   // Two more copies of p
    arr[0].x = 33;
    Console.WriteLine(arr[0] + " " + arr[1]);  // Prints (33, 22) (11, 22)
    Object o = p;                              // Another copy of p, in heap
    p.x = 44;
    Console.WriteLine(p + " " + o);            // Prints (44, 22) (11, 22)
    Console.WriteLine(o is SPoint);            // Prints True
    Console.WriteLine(o is int);               // Prints False
  }

  public static void TryReverse() {
    SPoint[] arr = new SPoint[5];
    for (int i=0; i<arr.Length; i++)
      arr[i] = new SPoint(i*11, i*22);
    // Reverse((ValueType[])arr); // Cannot convert SPoint[] to ValueType[]
    Console.WriteLine("Reversed input:");
    Console.WriteLine("--------------------");
    foreach (SPoint q in arr)
      Console.WriteLine(q);
    Console.WriteLine("--------------------");
  }

  public static void Reverse(ValueType[] arr) {
    for (int s=0, t=arr.Length-1; s<t; s++, t--) {
      ValueType tmp = arr[s]; arr[s] = arr[t]; arr[t] = tmp;
    }
  }
}
```

```
// Example 144 from page 113 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public struct SPoint {
  internal int x, y;

  public SPoint(int x, int y) { this.x = x; this.y = y; }

  public SPoint Move(int dx, int dy) { return this = new SPoint(x+dx, y+dy); }

  public override String ToString() { return "(" + x + "," + y + ")"; }
}

class MyTest {
  static readonly SPoint q = new SPoint(33, 44);

  public static void Main(String[] args) {
    SPoint p = new SPoint(11, 22);
    Console.WriteLine("p={0}", p);           // Now p = (11, 22)
    p.Move(9,8);
    Console.WriteLine("p={0}", p);           // Now p = (20, 30)
    p.Move(5,5).Move(6,6);
    Console.WriteLine("p={0}", p);           // Now p = (25, 35) not (31, 41)
    q.Move(5,5);
    Console.WriteLine("q={0}", q);           // Now q = (33, 44) not (38, 49)
  }
}
```

```
// Example 145 from page 115 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// The default argumentless constructor Frac(), which is unavoidable
// in a struct type, creates a Frac value that has d==0, violating the
// desirable invariant d!=0.  Fortunately, computing with such Frac
// values will throw DivideByZeroException, and conversion to double
// will produce a NaN.

using System;

struct Frac : IComparable {
  public readonly long n, d;       // NB: Meaningful only if d!=0

  public Frac(long n, long d) {
    long f = Gcd(n, d);
    this.n = n/f;
    this.d = d/f;
  }

  private static long Gcd(long m, long n) {
    while (m != 0)
      m = n % (n = m);
    return n;
  }

  public static Frac operator+(Frac r1, Frac r2) {
    return new Frac(r1.n*r2.d+r2.n*r1.d, r1.d*r2.d);
  }

  public static Frac operator*(Frac r1, Frac r2) {
    return new Frac(r1.n*r2.n, r1.d*r2.d);
  }

  // Both (or none) of the operators == and != must be defined:

  public static bool operator==(Frac r1, Frac r2) {
    return r1.n==r2.n && r1.d==r2.d;
  }

  public static bool operator!=(Frac r1, Frac r2) {
    return r1.n!=r2.n || r1.d!=r2.d;
  }

  // The preincrement and postincrement operator:

  public static Frac operator++(Frac r) {
    return r + 1;
  }

  // To implement the IComparable interface:

  public int CompareTo(Object that) {
    return ((double)this).CompareTo((double)(Frac)that);
  }

  // When == and != are defined, compatible methods Equals and
  // GetHashCode must be declared also:

  public override bool Equals(Object that) {
    return that is Frac && this == (Frac)that;
  }

  public override int GetHashCode() {
    return n.GetHashCode() ^ d.GetHashCode();
  }

  // Implicit conversion from int to Frac:

  public static implicit operator Frac(int n) {
    return new Frac(n, 1);
  }

  // Implicit conversion from long to Frac:

  public static implicit operator Frac(long n) {
    return new Frac(n, 1);
  }
```

```
  // Explicit conversion from Frac to long:

  public static explicit operator long(Frac r) {
    return r.n/r.d;
  }

  // Explicit conversion from Frac to float:

  public static explicit operator float(Frac r) {
    return ((float)r.n)/r.d;
  }

  // One cannot have an implicit conversion from Frac to double and at
  // the same time an implicit conversion from Frac to String; this
  // makes it impossible to decide which overload of WriteLine to use.

  public override String ToString() {
    if (d != 1)
      return n + "/" + d;
    else
      return n.ToString();
  }

  public bool IsZero {
    get { return n==0 && d!=0; }
  }
}

class TestFrac {
  public static void Main(String[] args) {
    Frac r1 = new Frac(6, 2), r2 = new Frac(5, 2);
    Console.WriteLine("r1={0} and r2={1}", r1, r2);
    Console.WriteLine((double)r2);      // Explicit conversion to double
    r2 = r2 * r2;                       // Overloaded multiplication
    Console.WriteLine("{0} {1} {2} {3} {4}", r2, ++r2, r2, r2++, r2);
    r2 = 0;                             // Implicit conversion from long
    for (int i=1; i<=10; i++) {
      r2 += new Frac(1, i);             // Overloaded += derived from overloaded +
      Console.WriteLine(r2 + " " + (r2 == new Frac(11, 6)));
    }
    Console.WriteLine("r2.IsZero is {0}", r2.IsZero);
    // Console.WriteLine(new Frac() + 1);
    // Console.WriteLine(new Frac() * new Frac(2, 3));
    Frac[] fs = { 5, new Frac(7, 8), 4, 2, new Frac(11, 3) };
    Array.Sort(fs);
    foreach (Frac f in fs)
      Console.WriteLine(f);
    // Using the user-defined conversions:
    Frac f1 = (byte)5;                      // Implicit int-->Frac
    Frac f2 = 1234567890123L;               // Implicit long-->Frac
    int i1 = (int)f1;                       // Explicit Frac-->long
    double d2 = (double)f2;                 // Explicit Frac-->float
    Console.WriteLine(f1 + "==" + i1);
    Console.WriteLine("Note loss of precision:");
    Console.WriteLine(f2 + "==" + d2);
  }
}
```

```
// Example 146 from page 117 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Windows.Forms;
using System.Drawing;           // Color, Graphics, SolidBrush, Pen, ...

class UseColored : System.Windows.Forms.Form {
  private static IColoredDrawable[] cs;

  static void PrintColors(IColored[] cs) {
    for (int i=0; i<cs.Length; i++)
      Console.WriteLine(cs[i].GetColor);
  }

  static void Draw(Graphics g, IColoredDrawable[] cs) {
    for (int i=0; i<cs.Length; i++) {
      Console.WriteLine(cs[i].GetColor);
      cs[i].Draw(g);
    }
  }

  public static void Main(String[] args) {
    cs = new IColoredDrawable[]
          { new ColoredDrawablePoint(3, 4, Color.Red),
            new ColoredRectangle(50, 100, 60, 110, Color.Green) };
    PrintColors(cs);
    Application.Run(new UseColored());
  }

  protected override void OnPaint(PaintEventArgs e) {
    Graphics g = e.Graphics;
    Draw(g, cs);
  }
}

public class Point {
  protected internal int x, y;

  public Point(int x, int y) { this.x = x; this.y = y; }

  public void Move(int dx, int dy) { x += dx; y += dy; }

  public override String ToString() { return "(" + x + "," + y + ")"; }
}

interface IColored { Color GetColor { get; } }
interface IDrawable { void Draw(Graphics g); }
interface IColoredDrawable : IColored, IDrawable {}

class ColoredPoint : Point, IColored {
  protected Color c;
  public ColoredPoint(int x, int y, Color c) : base(x, y) { this.c = c; }
  public Color GetColor { get { return c; } }
}

class ColoredDrawablePoint : ColoredPoint, IColoredDrawable {
  public ColoredDrawablePoint(int x, int y, Color c) : base(x, y, c) { }
  public void Draw(Graphics g) {
    g.FillRectangle(new SolidBrush(c), x, y, 2, 2);
  }
}

class ColoredRectangle : IColoredDrawable {
  private int x1, x2, y1, y2;   // (x1, y1) upper left, (x2, y2) lower right
  protected Color c;

  public ColoredRectangle(int x1, int y1, int x2, int y2, Color c)
  { this.x1 = x1; this.y1 = y1; this.x2 = x2; this.y2 = y2; this.c = c; }
  public Color GetColor { get { return c; } }
  public void Draw(Graphics g) {
    g.DrawRectangle(new Pen(c), x1, y1, x2, y2);
  }
}
```

```
// Example 147 from page 117 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Windows.Forms;
using System.Drawing;           // Color, Graphics, SolidBrush, Pen, ...

class UseColored : System.Windows.Forms.Form {
  private static IColoredDrawable[] cs;

  static void PrintColors(IColored[] cs) {
    for (int i=0; i<cs.Length; i++)
      Console.WriteLine(cs[i].GetColor);
  }

  static void Draw(Graphics g, IColoredDrawable[] cs) {
    for (int i=0; i<cs.Length; i++) {
      Console.WriteLine(cs[i].GetColor);
      cs[i].Draw(g);
    }
  }

  public static void Main(String[] args) {
    cs = new IColoredDrawable[]
         { new ColoredDrawablePoint(3, 4, Color.Red),
           new ColoredRectangle(50, 100, 60, 110, Color.Green) };
    PrintColors(cs);
    Application.Run(new UseColored());
  }

  protected override void OnPaint(PaintEventArgs e) {
    Graphics g = e.Graphics;
    Draw(g, cs);
  }
}

public class Point {
  protected internal int x, y;

  public Point(int x, int y) { this.x = x; this.y = y; }

  public void Move(int dx, int dy) { x += dx; y += dy; }

  public override String ToString() { return "(" + x + "," + y + ")"; }
}

interface IColored { Color GetColor { get; } }
interface IDrawable { void Draw(Graphics g); }
interface IColoredDrawable : IColored, IDrawable {}

class ColoredPoint : Point, IColored {
  protected Color c;
  public ColoredPoint(int x, int y, Color c) : base(x, y) { this.c = c; }
  public Color GetColor { get { return c; } }
}

class ColoredDrawablePoint : ColoredPoint, IColoredDrawable {
  public ColoredDrawablePoint(int x, int y, Color c) : base(x, y, c) { }
  public void Draw(Graphics g) {
    g.FillRectangle(new SolidBrush(c), x, y, 2, 2);
  }
}

class ColoredRectangle : IColoredDrawable {
  private int x1, x2, y1, y2;   // (x1, y1) upper left, (x2, y2) lower right
  protected Color c;

  public ColoredRectangle(int x1, int y1, int x2, int y2, Color c)
  { this.x1 = x1; this.y1 = y1; this.x2 = x2; this.y2 = y2; this.c = c; }
  public Color GetColor { get { return c; } }
  public void Draw(Graphics g) {
    g.DrawRectangle(new Pen(c), x1, y1, x2, y2);
  }
}
```

```
// Example 148 from page 117 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Windows.Forms;
using System.Drawing;           // Color, Graphics, SolidBrush, Pen, ...

class UseColored : System.Windows.Forms.Form {
  private static IColoredDrawable[] cs;

  static void PrintColors(IColored[] cs) {
    for (int i=0; i<cs.Length; i++)
      Console.WriteLine(cs[i].GetColor);
  }

  static void Draw(Graphics g, IColoredDrawable[] cs) {
    for (int i=0; i<cs.Length; i++) {
      Console.WriteLine(cs[i].GetColor);
      cs[i].Draw(g);
    }
  }

  public static void Main(String[] args) {
    cs = new IColoredDrawable[]
         { new ColoredDrawablePoint(3, 4, Color.Red),
           new ColoredRectangle(50, 100, 60, 110, Color.Green) };
    PrintColors(cs);
    Application.Run(new UseColored());
  }

  protected override void OnPaint(PaintEventArgs e) {
    Graphics g = e.Graphics;
    Draw(g, cs);
  }
}

public class Point {
  protected internal int x, y;

  public Point(int x, int y) { this.x = x; this.y = y; }

  public void Move(int dx, int dy) { x += dx; y += dy; }

  public override String ToString() { return "(" + x + "," + y + ")"; }
}

interface IColored { Color GetColor { get; } }
interface IDrawable { void Draw(Graphics g); }
interface IColoredDrawable : IColored, IDrawable {}

class ColoredPoint : Point, IColored {
  protected Color c;
  public ColoredPoint(int x, int y, Color c) : base(x, y) { this.c = c; }
  public Color GetColor { get { return c; } }
}

class ColoredDrawablePoint : ColoredPoint, IColoredDrawable {
  public ColoredDrawablePoint(int x, int y, Color c) : base(x, y, c) { }
  public void Draw(Graphics g) {
    g.FillRectangle(new SolidBrush(c), x, y, 2, 2);
  }
}

class ColoredRectangle : IColoredDrawable {
  private int x1, x2, y1, y2;   // (x1, y1) upper left, (x2, y2) lower right
  protected Color c;

  public ColoredRectangle(int x1, int y1, int x2, int y2, Color c)
  { this.x1 = x1; this.y1 = y1; this.x2 = x2; this.y2 = y2; this.c = c; }
  public Color GetColor { get { return c; } }
  public void Draw(Graphics g) {
    g.DrawRectangle(new Pen(c), x1, y1, x2, y2);
  }
}
```

```
// Example 149 from page 119 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// The struct type of integer sequences

// Overloaded operators and indexers, enumerators

using System;                           // For Console, String
using System.Text;                      // For StringBuilder
using SC = System.Collections;          // For IEnumerator, IEnumerable
using System.Collections.Generic;       // For IEnumerator<T>

struct Seq : ISeq {
  private readonly int b, k, n;                      // Sequence b+k*[0..n-1]

  // Default constructor Seq() creates an empty sequence with n=0

  public Seq(int m, int n) : this(m, 1, n-m+1) { }   // Sequence [m..n]

  public Seq(int b, int k, int n) {
    this.b = b; this.k = k; this.n = n;
  }

  // Add b to sequence
  public static Seq operator +(int b, Seq seq) {
    return new Seq(seq.b+b, seq.k, seq.n);
  }

  // Add b to sequence
  public static Seq operator +(Seq seq, int b) {
    return new Seq(seq.b+b, seq.k, seq.n);
  }

  // Multiply all members of the sequence by k
  public static Seq operator *(int k, Seq seq) {
    return new Seq(seq.b*k, seq.k*k, seq.n);
  }

  // Multiply all members of the sequence by k
  public static Seq operator *(Seq seq, int k) {
    return new Seq(seq.b*k, seq.k*k, seq.n);
  }

  // Reverse the sequence
  public static Seq operator !(Seq seq) {
    return new Seq(seq.b+(seq.n-1)*seq.k, -seq.k, seq.n);
  }

  // Equality and inequality
  public static bool operator ==(Seq s1, Seq s2) {
    return s1.n==s2.n && (s1.n==0 || s1.b==s2.b && (s1.n==1 || s1.k==s2.k));
  }

  public static bool operator !=(Seq s1, Seq s2) { return !(s1==s2); }

  public override bool Equals(Object that) {
    return that is Seq && this==(Seq)that;
  }

  public override int GetHashCode() {
    return n==0 ? 0 : n==1 ? b : b^k^n;
  }

  // Get enumerator for the sequence
  public IEnumerator<int> GetEnumerator() {
    return new SeqEnumerator(this);
  }

  // Get enumerator for the sequence
  SC.IEnumerator SC.IEnumerable.GetEnumerator() {
    return GetEnumerator();
  }

  // An enumerator for a sequence, used in foreach statements
  private class SeqEnumerator : IEnumerator<int> {    // Static member class
    private readonly Seq seq;
    private int i;

    public SeqEnumerator(Seq seq) {
      this.seq = seq; Reset();
    }

    public int Current {                // For IEnumerator<int>
      get {
        if (0 <= i && i < seq.n)
          return seq.b + seq.k * i;
        else
          throw new InvalidOperationException();
      }
    }

    Object SC.IEnumerator.Current { get { return Current; } }

    public bool MoveNext() {            // For IEnumerator<int> and IEnumerator
      return ++i < seq.n;
    }

    public void Reset() {               // For IEnumerator
      i = -1;
    }

    public void Dispose() { }           // For IEnumerator<int>
  }

  public int Count {
    get { return n; }
  }

  public int this[int i] {
    get {
      if (0 <= i && i < n)
        return b + k * i;
      else
        throw new ArgumentOutOfRangeException("Seq indexer: " + i);
    }
  }

  public int[] this[params int[] ii] {
    get {
      int[] res = new int[ii.Length];
      for (int h=0; h<res.Length; h++)
        res[h] = this[ii[h]];
      return res;
    }
  }

  public void Print() {
    IEnumerator<int> etor = GetEnumerator();
    while (etor.MoveNext())
      Console.Write(etor.Current + " ");
  }

  public override String ToString() {
    StringBuilder sb = new StringBuilder();
    foreach (int i in this)
      sb.Append(i).Append(" ");
    return sb.ToString();
  }
}

class TestSeq {
  public static void Main(String[] args) {
    Seq s1 = new Seq(1, 3);             // 1 2 3
    Seq s2 = 2 * s1 + 5;                // 7 9 11
    Seq s3 = s2 * 3;                    // 21 27 33
    Seq s4 = !s3;                       // 33 27 21
    Console.WriteLine(s1);
    Console.WriteLine(s2);
    Console.WriteLine(s3);
    Console.WriteLine(s4);
    Console.WriteLine(s1==s2);                          // False
    Console.WriteLine(s3==!s4);                         // True
    Console.WriteLine(new Seq()==new Seq(5,7,0));       // True
    Console.WriteLine(new Seq(17,17)==new Seq(17,5,1)); // True
    s4.Print();                         // 33 27 21
    Console.WriteLine();                // 33 27 21
    for (int i=0, stop=s4.Count; i<stop; i++)
      Console.Write(s4[i] + " ");
```

```
    Console.WriteLine();
    int[] r = s4[2, 2, 1, 2, 0];
    for (int i=0, stop=r.Length; i<stop; i++)
      Console.Write(r[i] + " ");        // 21 21 27 21 33
    Console.WriteLine();
  }
}

interface ISeq : IEnumerable<int> {
  int Count { get; }
  int this[int i] { get; }
  int[] this[params int[] ii] { get; }
}
```

```
// Example 150 from page 119 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// This example should be illegal (by C# Language Specification
// 13.4.1) if the I1 interface were removed from class C's list of
// interfaces, but both MS csc 1.1, 2.0 alpha and 2.0 March 2004 CTP,
// and Mono mcs 0.25, 0.28 and 0.91 accept it.

using System;

interface I1 {
  void M0();
}
interface I2 : I1 {
  new void M0();
  int M1();
}
interface I3 : I1 {
  void M1();
  int P { get; }
  int this[int i] { get; }
}
class C : I1, I2, I3 {
  public void M0() { Console.Write("C.M0 "); }
  void I1.M0() { Console.Write("C:I1.M0 "); }
  void I2.M0() { Console.Write("C:I2.M0 "); }
  int  I2.M1() { Console.Write("C:I2.M1 "); return 1; }
  void I3.M1() { Console.Write("C:I3.M1 "); }
  int I3.P { get { return 11; } }
  int I3.this[int i] { get { return i+((I3)this).P; } }
  // void I3.M0() { }                    // Illegal: M0 not explicitly in I3
}
class D : C { }

class MyTest {
  public static void Main(String[] args) {
    C c = new C();
    // C.M0 C:I1.M0 C:I2.M0 C:I2.M1 C:I3.M1
    c.M0(); ((I1)c).M0(); ((I2)c).M0(); ((I2)c).M1(); ((I3)c).M1();
    Console.WriteLine();
    D d = new D();
    // C.M0 C:I1.M0 C:I2.M0 C:I2.M1 C:I3.M1
    d.M0(); ((I1)d).M0(); ((I2)d).M0(); ((I2)d).M1(); ((I3)d).M1();
    Console.WriteLine();
  }
}
```

```
// Example 151 from page 121 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Using enums in calendrical calculations

using System;

public enum Day {
  Mon, Tue, Wed, Thu, Fri, Sat, Sun
}

public enum Month {
  Jan=1, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
}

public class Date {
  readonly int yy /* 0-9999 */, dd /* 1-31 */;
  readonly Month mm;

  public Date(int yy, Month mm, int dd) {
    if (Ok(yy, mm, dd)) {
      this.yy = yy; this.mm = mm; this.dd = dd;
    } else
      throw new Exception("Illegal date ("+yy+","+mm+","+dd+")");
  }

  public static bool LeapYear(int y) {
    return y % 4 == 0 && y % 100 != 0 || y % 400 == 0;
  }

  public bool LeapYear() {
    return LeapYear(yy);
  }

  public static int MonthDays(int y, Month m) {
    switch (m) {
    case Month.Apr: case Month.Jun: case Month.Sep: case Month.Nov:
      return 30;
    case Month.Feb:
      return LeapYear(y) ? 29 : 28;
    default:
      return 31;
    }
  }

  public int MonthDays() {
    return MonthDays(yy, mm);
  }

  public static int YearDays(int y) {
    return LeapYear(y) ? 366 : 365;
  }

  public int YearDays() {
    return YearDays(yy);
  }

  public static bool Ok(int y, Month m, int d) {
    return 1 <= d && d <= MonthDays(y, m);
  }

  // ISO week numbers: the week is from Monday to Sunday.  Week 1 is
  // the first week having a Thursday.

  public static int WeekNumber(int y, Month m, int d) {
    int yday = DayInYear(y, m, d);
    int wday = (int)Weekday(y, m, d);
    int week = (yday - wday + 10)/7;
    if (week == 0)
      return (yday + YearDays(y-1) - wday + 10)/7;
    else
      return week;
  }

  public int WeekNumber() {
    return WeekNumber(yy, mm, dd);
  }

  // Translated from Emacs's calendar.el:

  // Reingold: Number of the day within the year:

  public static int DayInYear(int y, Month m, int d) {
    int monthno = (int)m - 1;
    int monthadjust =
      monthno > 1 ? (27 + 4 * monthno) / 10 - (LeapYear(y) ? 1 : 0) : 0;
    return d - 1 + 31 * monthno - monthadjust;
  }

  public int DayInYear() {
    return DayInYear(yy, mm, dd);
  }

  // Reingold: Find the number of days elapsed from the (imagined)
  // Gregorian date Sunday, December 31, 1 BC to the given date.

  public static int ToDaynumber(int y, Month m, int d) {
    int prioryears = y - 1;
    return
      DayInYear(y, m, d)
      + 1 + 365 * prioryears
      + prioryears / 4 - prioryears / 100 + prioryears / 400;
  }

  public int ToDaynumber() {
    return ToDaynumber(yy, mm, dd);
  }

  // Reingold et al: from absolute day number to year, month, date:

  public static Date FromDaynumber(int n) {
    int d0 = n - 1;
    int n400 = d0 / 146097;
    int d1 = d0 % 146097;
    int n100 = d1 / 36524;
    int d2 = d1 % 36524;
    int n4 = d2 / 1461;
    int d3 = d2 % 1461;
    int n1 = d3 / 365;
    int d = 1 + d3 % 365;
    int y = 400 * n400 + 100 * n100 + n4 * 4 + n1 + 1;
    if (n100 == 4 || n1 == 4) {
      return new Date(y-1, Month.Dec, 31);
    } else {
      Month m = Month.Jan;
      int mdays;
      while ((mdays = MonthDays(y, m)) < d) {
        d -= mdays;
        m++;
      }
      return new Date(y, m, d);
    }
  }

  // Day of the week: 0=Mon, 1=Tue, ..., 6=Sun

  public static Day Weekday(int y, Month m, int d) {
    return (Day)((ToDaynumber(y, m, d)+6) % 7);
  }

  public Day Weekday() {
    return Weekday(yy, mm, dd);
  }

  public override String ToString() { // ISO format such as 2003-05-31
    return String.Format("{0:D4}-{1:D2}-{2:D2}", yy, (int)mm, dd);
  }
}

class Example151 {
  public static void Main(String[] args) {
    if (args.Length != 3)
      Console.WriteLine("Usage: Example151 yyyy mm dd\n");
    else {
      Date d = new Date(int.Parse(args[0]),
                        (Month)int.Parse(args[1]),
                        int.Parse(args[2]));
      Console.WriteLine(d + " is " + d.Weekday() + " in week " + d.WeekNumber());
    }
```

```
  }
}
```

```
// Example 152 from page 121 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static void Main(String[] args) {
    Console.WriteLine(Color.Red + " " + (uint)Color.Red);
  }
}
public enum Color : uint {
  Red = 0xFF0000, Green = 0x00FF00, Blue = 0x0000FF
}
```

```
// Example 153 from page 123 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Quicksort using a delegate to compare elements

using System;

class DelegateQuicksort {
  public static void Main(String[] args) {
    Object[] ia = { 5, 7, 3, 9, 12, 45, 4, 8 };
    Qsort(ia, IntCompare, 0, ia.Length-1);
    foreach (int i in ia)
      Console.Write("{0} ", i);
    Console.WriteLine();
    String[] sa = { "New York", "Rome", "Dublin", "Riyadh", "Tokyo" };
    Qsort(sa, StringReverseCompare, 0, sa.Length-1);
    foreach (String s in sa)
      Console.Write("{0} ", s);
    Console.WriteLine();
    String[] sa2 = { "New York", "Rome", "Dublin", "Riyadh", "Tokyo" };
    Qsort(sa2, (v1, v2) => String.Compare((String)v2, (String)v1), 0, sa2.Length-1);
    foreach (String s in sa2)
      Console.Write("{0} ", s);
    Console.WriteLine();
  }

  // Quicksort: sorts arr[a..b] using delegate cmp to compare elements

  private static void Qsort(Object[] arr, DComparer cmp, int a, int b) {
    if (a < b) {
      int i = a, j = b;
      Object x = arr[(i+j) / 2];
      do {
        while (cmp(arr[i], x) < 0) i++;         // Call delegate cmp
        while (cmp(x, arr[j]) < 0) j--;         // Call delegate cmp
        if (i <= j) {
          Object tmp = arr[i]; arr[i] = arr[j]; arr[j] = tmp;
          i++; j--;
        }
      } while (i <= j);
      Qsort(arr, cmp, a, j);
      Qsort(arr, cmp, i, b);
    }
  }

  // The DComparer delegate type

  public delegate int DComparer(Object v1, Object v2);

  // Comparison methods for int and String

  static int IntCompare(Object v1, Object v2) {
    int i1 = (int)v1, i2 = (int)v2;
    return i1 < i2 ? -1 : i1 > i2 ? +1 : 0;
  }

  static int StringReverseCompare(Object v1, Object v2) {
    return String.Compare((String)v2, (String)v1);
  }
}
```

```
// Example 154 from page 123 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class TestDelegate {
  public static void Main(String[] args) {
    TestDelegate o = new TestDelegate();
    D dlg1 = o.M1, dlg2 = M2, dlg3 = dlg1 + dlg2;
    dlg3 += dlg3;
    int y = 0;
    Console.WriteLine(dlg3(ref y));     // Prints: M1/1 M2/2 M1/3 M2/4 4
    dlg3 -= o.M1;
    Console.WriteLine(dlg3(ref y));     // Prints: M1/5 M2/6 M2/7 7
  }

  public delegate int D(ref int x);

  int M1(ref int x) { x++; Console.Write("M1/{0} ", x); return x; }
  static int M2(ref int x) { x++; Console.Write("M2/{0} ", x); return x; }
}
```

```
// Example 155 from page 125 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

class MyTest {
  private static readonly Random rnd = new Random();

  class Phone {
    public readonly String name;
    public readonly int phone;
    public Phone(String name, int phone) {
      this.name = name;
      this.phone = phone;
    }
  }

  public static void Main(String[] args) {
    // Basic rules of type dynamic
    {
      dynamic d1 = 34;
      int i1 = d1 * 2;                  // OK: cast (int)(d1*2) at run-time
      int i2 = (int)d1 * 2;             // OK: cast (int)d1 at run-time
      // bool b1 = d1;                  // Compiles OK; cast (bool)d1 throws at run-t
ime
      d1 = true;                        // OK
      bool b2 = d1;                     // OK: cast (bool)d1 succeeds at run-time
      dynamic p1 = new Phone("Kasper", 5170);
      String s1 = p1.name;              // Field access checked at run-time
      // int n1 = p1.age;               // Compiles OK; field access throws at run-ti
me
      dynamic p2 = new { name = "Kasper", phone = 5170 };
      String s2 = p2.name;              // Field access checked at run-time
      // int n2 = p2.age;               // Compiles OK; fields access throws at run-t
ime
    }

    // Dynamic operator resolution; run-time type determines meaning of "+" in Plus2(
)
    {
      Console.WriteLine(Plus2(int.MaxValue-1));             // -2147483648, due to i
nt overflow
      Console.WriteLine(Plus2((long)(int.MaxValue-1)));     //  2147483648, no long
overflow
      Console.WriteLine(Plus2(11.5));                       // 13.5
      Console.WriteLine(Plus2("Spar"));                    // Spar2
      // Console.WriteLine(Plus2(false));       // Compiles OK; throws RuntimeBinder
Exception
    }

    // Dynamic receiver; run-time type determines whether to call Length on array or
String
    {
      dynamic v;
      if (args.Length==0)      v = new int[] { 2, 3, 5, 7 };
      else                     v = "abc";
      int res = v.Length;
      Console.WriteLine(res);
    }

    // Dynamic overload resolution; run-time type of v determines which Process calle
d at (**)
    {
      dynamic v;
      if (args.Length==0)      v = 5;
      else if (args[0] == "1") v = "abc";
      else                     v = (Func<int,int>)(x => x*3);
      dynamic r = Process(v);                                   // (**)
      if (args.Length==0 || args[0] == "1")
        Console.WriteLine(r);
      else
        Console.WriteLine(r(11));
      dynamic s = "abc";
      Console.WriteLine(Process(s).StartsWith("abca"));
     }

      // Run-time type tests
    {
      Console.WriteLine(Square(5));
      Console.WriteLine(Square("abc"));
      Func<int,int> f = x => x*3;
      Console.WriteLine(Square(f)(11));
    }

    {
      // Types dynamic[], List<dynamic>, IEnumerable<dynamic>
      dynamic[] arr = new dynamic[] { 19, "Electric", (Func<int,int>)(n => n+2), 3.2, f
alse };
      int number = arr[0] * 5;
      String street = arr[1].ToUpper();
      int result = arr[2](number);
      Console.WriteLine(number + " " + street);
      double sum = 0;
      List<dynamic> list = new List<dynamic>(arr);
      IEnumerable<dynamic> xs = list;
      foreach (dynamic x in xs)
        if (x is int || x is double)
          sum += x;
      Console.WriteLine(sum);                                   // 22.2
    }

    // Dynamic and anonymous object expressions
    {
      dynamic v = new { x = 34, y = false };
      Console.WriteLine(v.x);
    }
  }

  // Run-time type of v determines meaning of "+": int+, long+, double+, String+, ...
  static dynamic Plus2(dynamic v) { return v + 2; }

  // Ordinary overloaded methods
  static int Process(int v) { return v * v; }

  static double Process(double v) { return v * v; }

  static String Process(String v) { return v + v; }

  static Func<int,int> Process(Func<int,int> v) { return x => v(v(x)); }

  // Explicit tests on run-time type.  Using "dynamic" rather than
  // "Object" here means that the compiler accepts the (v * v)
  // expression and the application of v to arguments, and that the
  // value returned by Square can be further processed: added to,
  // applied to arguments, and so on.
  static dynamic Square(dynamic v) {
    if (v is int || v is double)
      return v * v;
    else if (v is String)
      return v + v;
    else if (v is Func<int,int>)
      return (Func<int,int>)(x => v(v(x)));
    else
      throw new Exception("Don't know how to square " + v);
  }
}

class C : List<dynamic> { }
```

```
// Example 156 from page 125 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

class MyTest {
  private static readonly Random rnd = new Random();

  class Phone {
    public readonly String name;
    public readonly int phone;
    public Phone(String name, int phone) {
      this.name = name;
      this.phone = phone;
    }
  }

  public static void Main(String[] args) {
    // Basic rules of type dynamic
    {
      dynamic d1 = 34;
      int i1 = d1 * 2;                  // OK: cast (int)(d1*2) at run-time
      int i2 = (int)d1 * 2;             // OK: cast (int)d1 at run-time
      // bool b1 = d1;                  // Compiles OK; cast (bool)d1 throws at run-t
ime
      d1 = true;                        // OK
      bool b2 = d1;                     // OK: cast (bool)d1 succeeds at run-time
      dynamic p1 = new Phone("Kasper", 5170);
      String s1 = p1.name;              // Field access checked at run-time
      // int n1 = p1.age;               // Compiles OK; field access throws at run-ti
me
      dynamic p2 = new { name = "Kasper", phone = 5170 };
      String s2 = p2.name;              // Field access checked at run-time
      // int n2 = p2.age;               // Compiles OK; fields access throws at run-t
ime
    }

    // Dynamic operator resolution; run-time type determines meaning of "+" in Plus2(
)
    {
      Console.WriteLine(Plus2(int.MaxValue-1));             // -2147483648, due to i
nt overflow
      Console.WriteLine(Plus2((long)(int.MaxValue-1)));     //  2147483648, no long
overflow
      Console.WriteLine(Plus2(11.5));                       // 13.5
      Console.WriteLine(Plus2("Spar"));                    // Spar2
      // Console.WriteLine(Plus2(false));       // Compiles OK; throws RuntimeBinder
Exception
    }

    // Dynamic receiver; run-time type determines whether to call Length on array or
String
    {
      dynamic v;
      if (args.Length==0)      v = new int[] { 2, 3, 5, 7 };
      else                     v = "abc";
      int res = v.Length;
      Console.WriteLine(res);
    }

    // Dynamic overload resolution; run-time type of v determines which Process calle
d at (**)
    {
      dynamic v;
      if (args.Length==0)      v = 5;
      else if (args[0] == "1") v = "abc";
      else                     v = (Func<int,int>)(x => x*3);
      dynamic r = Process(v);                                   // (**)
      if (args.Length==0 || args[0] == "1")
        Console.WriteLine(r);
      else
        Console.WriteLine(r(11));
      dynamic s = "abc";
      Console.WriteLine(Process(s).StartsWith("abca"));
     }

      // Run-time type tests
    {
      Console.WriteLine(Square(5));
      Console.WriteLine(Square("abc"));
      Func<int,int> f = x => x*3;
      Console.WriteLine(Square(f)(11));
    }

    {
      // Types dynamic[], List<dynamic>, IEnumerable<dynamic>
      dynamic[] arr = new dynamic[] { 19, "Electric", (Func<int,int>)(n => n+2), 3.2, f
alse };
      int number = arr[0] * 5;
      String street = arr[1].ToUpper();
      int result = arr[2](number);
      Console.WriteLine(number + " " + street);
      double sum = 0;
      List<dynamic> list = new List<dynamic>(arr);
      IEnumerable<dynamic> xs = list;
      foreach (dynamic x in xs)
        if (x is int || x is double)
          sum += x;
      Console.WriteLine(sum);                                   // 22.2
    }

    // Dynamic and anonymous object expressions
    {
      dynamic v = new { x = 34, y = false };
      Console.WriteLine(v.x);
    }
  }

  // Run-time type of v determines meaning of "+": int+, long+, double+, String+, ...
  static dynamic Plus2(dynamic v) { return v + 2; }

  // Ordinary overloaded methods
  static int Process(int v) { return v * v; }

  static double Process(double v) { return v * v; }

  static String Process(String v) { return v + v; }

  static Func<int,int> Process(Func<int,int> v) { return x => v(v(x)); }

  // Explicit tests on run-time type.  Using "dynamic" rather than
  // "Object" here means that the compiler accepts the (v * v)
  // expression and the application of v to arguments, and that the
  // value returned by Square can be further processed: added to,
  // applied to arguments, and so on.
  static dynamic Square(dynamic v) {
    if (v is int || v is double)
      return v * v;
    else if (v is String)
      return v + v;
    else if (v is Func<int,int>)
      return (Func<int,int>)(x => v(v(x)));
    else
      throw new Exception("Don't know how to square " + v);
  }
}

class C : List<dynamic> { }
```

```
// Example 157 from page 125 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

class MyTest {
  private static readonly Random rnd = new Random();

  class Phone {
    public readonly String name;
    public readonly int phone;
    public Phone(String name, int phone) {
      this.name = name;
      this.phone = phone;
    }
  }

  public static void Main(String[] args) {
    // Basic rules of type dynamic
    {
      dynamic d1 = 34;
      int i1 = d1 * 2;                  // OK: cast (int)(d1*2) at run-time
      int i2 = (int)d1 * 2;             // OK: cast (int)d1 at run-time
      // bool b1 = d1;                  // Compiles OK; cast (bool)d1 throws at run-t
ime
      d1 = true;                        // OK
      bool b2 = d1;                     // OK: cast (bool)d1 succeeds at run-time
      dynamic p1 = new Phone("Kasper", 5170);
      String s1 = p1.name;              // Field access checked at run-time
      // int n1 = p1.age;               // Compiles OK; field access throws at run-ti
me
      dynamic p2 = new { name = "Kasper", phone = 5170 };
      String s2 = p2.name;              // Field access checked at run-time
      // int n2 = p2.age;               // Compiles OK; fields access throws at run-t
ime
    }

    // Dynamic operator resolution; run-time type determines meaning of "+" in Plus2(
)
    {
      Console.WriteLine(Plus2(int.MaxValue-1));             // -2147483648, due to i
nt overflow
      Console.WriteLine(Plus2((long)(int.MaxValue-1)));     //  2147483648, no long
overflow
      Console.WriteLine(Plus2(11.5));                       // 13.5
      Console.WriteLine(Plus2("Spar"));                    // Spar2
      // Console.WriteLine(Plus2(false));       // Compiles OK; throws RuntimeBinder
Exception
    }

    // Dynamic receiver; run-time type determines whether to call Length on array or
String
    {
      dynamic v;
      if (args.Length==0)      v = new int[] { 2, 3, 5, 7 };
      else                     v = "abc";
      int res = v.Length;
      Console.WriteLine(res);
    }

    // Dynamic overload resolution; run-time type of v determines which Process calle
d at (**)
    {
      dynamic v;
      if (args.Length==0)      v = 5;
      else if (args[0] == "1") v = "abc";
      else                     v = (Func<int,int>)(x => x*3);
      dynamic r = Process(v);                                   // (**)
      if (args.Length==0 || args[0] == "1")
        Console.WriteLine(r);
      else
        Console.WriteLine(r(11));
      dynamic s = "abc";
      Console.WriteLine(Process(s).StartsWith("abca"));
     }

      // Run-time type tests
    {
      Console.WriteLine(Square(5));
      Console.WriteLine(Square("abc"));
      Func<int,int> f = x => x*3;
      Console.WriteLine(Square(f)(11));
    }

    {
      // Types dynamic[], List<dynamic>, IEnumerable<dynamic>
      dynamic[] arr = new dynamic[] { 19, "Electric", (Func<int,int>)(n => n+2), 3.2, f
alse };
      int number = arr[0] * 5;
      String street = arr[1].ToUpper();
      int result = arr[2](number);
      Console.WriteLine(number + " " + street);
      double sum = 0;
      List<dynamic> list = new List<dynamic>(arr);
      IEnumerable<dynamic> xs = list;
      foreach (dynamic x in xs)
        if (x is int || x is double)
          sum += x;
      Console.WriteLine(sum);                                   // 22.2
    }

    // Dynamic and anonymous object expressions
    {
      dynamic v = new { x = 34, y = false };
      Console.WriteLine(v.x);
    }
  }

  // Run-time type of v determines meaning of "+": int+, long+, double+, String+, ...
  static dynamic Plus2(dynamic v) { return v + 2; }

  // Ordinary overloaded methods
  static int Process(int v) { return v * v; }

  static double Process(double v) { return v * v; }

  static String Process(String v) { return v + v; }

  static Func<int,int> Process(Func<int,int> v) { return x => v(v(x)); }

  // Explicit tests on run-time type.  Using "dynamic" rather than
  // "Object" here means that the compiler accepts the (v * v)
  // expression and the application of v to arguments, and that the
  // value returned by Square can be further processed: added to,
  // applied to arguments, and so on.
  static dynamic Square(dynamic v) {
    if (v is int || v is double)
      return v * v;
    else if (v is String)
      return v + v;
    else if (v is Func<int,int>)
      return (Func<int,int>)(x => v(v(x)));
    else
      throw new Exception("Don't know how to square " + v);
  }
}

class C : List<dynamic> { }
```

```
// Example 158 from page 127 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static int? Sqrt(int? x) {
    if (x.HasValue && x.Value >= 0)
      return (int)(Math.Sqrt(x.Value));
    else
      return null;
  }

  public static void Main(String[] args) {
    // Prints :2:::
    Console.WriteLine(":{0}:{1}:{2}:", Sqrt(5), Sqrt(null), Sqrt(-5));
  }
}
```

```
// Example 159 from page 127 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static void Main(String[] args) {
    Console.WriteLine("In this example null prints as blank or []\n");
    int? i1 = 11, i2 = 22, i3 = null, i4 = i1+i2, i5 = i1+i3;
    // Values: 11 22 null 33 null
    Console.WriteLine("[{0}][{1}][{2}][{3}][{4}]", i1, i2, i3, i4, i5);
    int i6 = (int)i1;                          // Legal
    // int i7 = (int)i5;                       // Legal but fails at run-time
    // int i8 = i1;                            // Illegal

    Object o1 = i1, o3 = i3;                   // Boxing of int? gives boxed int
    Console.WriteLine(o1.GetType());           // System.Int32
    int? ii1 = (int?)o1, ii3 = (int?)o3;       // Unboxing of boxed int gives int?
    Console.WriteLine("[{0}][{1}]", ii1, ii3); // [11] [null]

    int?[] iarr = { i1, i2, i3, i4, i5 };
    i2 += i1;
    i2 += i4;
    Console.WriteLine("i2 = {0}", i2);          // 66 = 11+22+33

    int sum = 0;
    for (int i=0; i<iarr.Length; i++)
      sum += iarr[i] != null ? iarr[i].Value : 0;
      //     sum += iarr[i] ?? 0;
    Console.WriteLine("sum = {0}", sum);        // 66 = 11+22+33

    for (int i=0; i<iarr.Length; i++)
      if (iarr[i] > 11)
        Console.Write("[{0}]", iarr[i]);       // 22 33
    Console.WriteLine();

    for (int i=0; i<iarr.Length; i++)
      if (iarr[i] != i1)
        Console.Write("[{0}]", iarr[i]);       // 22 null 33 null
    Console.WriteLine();
    Console.WriteLine();
    int?[] ivals = { null, 2, 5 };
    Console.WriteLine("{0,6} {1,6} {2,6} {3,6} {4,-6} {5,-6} {6,-6} {7,-6}",
                      "x", "y", "x+y", "x-y", "x<y", "x>=y", "x==y", "x!=y");
    Console.WriteLine();
    foreach (int? x in ivals)
      foreach (int? y in ivals)
        Console.WriteLine("{0,6} {1,6} {2,6} {3,6} {4,-6} {5,-6} {6,-6} {7,-6}",
                          x, y, x+y, x-y, x<y, x>=y, x==y, x!=y);
  }
}
```

```
// Example 160 from page 127 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MyTest {
  public static void Main(String[] args) {
    Console.WriteLine("In this example null prints as blank or []\n");
    bool? b1 = null, b2 = false, b3 = true;
    bool? b4 = b1^b2, b5 = b1&b2, b6 = b1|b2;                    // null false null
    Console.WriteLine("[{0}][{1}][{2}]", b4, b5, b6);
    bool? b7 = b1^b3, b8 = b1&b3, b9 = b1|b3;                    // null null true
    Console.WriteLine("[{0}][{1}][{2}]", b7, b8, b9);

    Console.WriteLine();
    bool?[] bvals = new bool?[] { null, false, true };
    Console.WriteLine("{0,-6} {1,-6} {2,-6} {3,-6} {4,-6}",
                      "x", "y", "x&y", "x|y", "x^y");
    foreach (bool? x in bvals)
      foreach (bool? y in bvals)
        Console.WriteLine("{0,-6} {1,-6} {2,-6} {3,-6} {4,-6}",
                          x, y, x&y, x|y, x^y);
    Console.WriteLine();
    Console.WriteLine("{0,-6} {1,-6}", "x", "!x");
    foreach (bool? x in bvals)
      Console.WriteLine("{0,-6} {1,-6}", x, !x);
  }
}
```

```
// Example 161 from page 129 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class WeekdayException : ApplicationException {
  public WeekdayException(String wday) : base("Illegal weekday: " + wday) {
  }
}
```

```
// Example 162 from page 129 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// To exercise all paths through the try-catch-finally statement in
// method M, run this program with each of these arguments:
// 101 102 201 202 301 302 411 412 421 422 431 432
// like this:
//    Example162 101
//    Example162 102
//    etc

using System;

class Example162 {
  public static void Main(String[] args) {
    if (args.Length != 1)
      Console.WriteLine("Usage: Example162 <integer>\n");
    else
      Console.WriteLine(M(int.Parse(args[0])));
  }

  static String M(int a) {
    try {
      Console.Write("try ... ");
      if (a/100 == 2) return "returned from try";
      if (a/100 == 3) throw new Exception("thrown by try");
      if (a/100 == 4) throw new ApplicationException("thrown by try");
    } catch (ApplicationException) {
      Console.Write("catch ... ");
      if (a/10%10 == 2) return "returned from catch";
      if (a/10%10 == 3) throw new Exception("thrown by catch");
    } finally {
      Console.WriteLine("finally");
      // return "foo"; // Would be illegal
      if (a%10 == 2) throw new Exception("thrown by finally");
    }
    return "terminated normally with " + a;
  }
}
```

```
// Example 163 from page 131 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Threading;

class ThreadDemo {
  private static int i;

  public static void Main() {
    Thread u = new Thread(new ThreadStart(Run));
    u.Start();
    Console.WriteLine("Repeatedly press Enter to get the current value of i:");
    for (;;) {
      Console.ReadLine();                          // Wait for keyboard input
      Console.WriteLine(i);
    }
  }

  private static void Run() {
    for (;;) {                                     // Forever
      i++;                                         //   increment i
      Thread.Sleep(0);                             //   yield to other thread
    }
  }
}
```

```
// Example 164 from page 133 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// 1. Run this program to see that mutual exclusion works: the program
//    alternately prints a dash (-) and a slash (/) forever.
// 2. Then comment out lock(mutex) as in
//      /* lock (mutex) */ {
//    and compile and run the program again.  Now the strict alternation
//    between dash (-) and slash (/) in the output will break.

using System;
using System.Threading;

class Printer {
  static readonly Object mutex = new Object();
  public static void Run() {
    for (;;) {
      lock (mutex) {
        Console.Write("-");
        Util.Pause(100,300);
        Console.Write("/");
      }
      Util.Pause(200);
} } }

class TestPrinter {
  public static void Main(String[] args) {
    Console.WriteLine("Observe concurrent threads.  Use ctrl-C to stop.\n");
    new Thread(new ThreadStart(Printer.Run)).Start();
    new Thread(new ThreadStart(Printer.Run)).Start();
  }
}

// Pseudo-random numbers and sleeping threads

class Util {
  private static readonly Random rnd = new Random();

  public static void Pause(int length) {
    Thread.Sleep(length);
  }

  public static void Pause(int a, int b) {
    Pause(Random(a, b));
  }

  public static int Random(int a, int b) {
    return rnd.Next(a, b);
  }
}
```

```
// Example 165 from page 133 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// 1. Run this program to see that mutual exclusion works: the sum
//    of the bank accounts' balances forever remains 30.
// 2. Then comment out lock(this) as in
//      /* lock (this) */ {
//    and compile and run the program again.  Now the sum of the balances
//    will deviate from 30 because the bank clerks occasionally overwrite
//    each others' updates.

using System;
using System.Threading;

class Bank {
  private int account1 = 10, account2 = 20;
  public void Transfer(int amount) {
    lock (this) {
      int new1 = account1 - amount;
      Util.Pause(10);
      account1 = new1; account2 = account2 + amount;
      Console.WriteLine("Sum is " + (account1+account2));
    }
} }

class Clerk {
  private Bank bank;
  public Clerk(Bank bank) {
    this.bank = bank;
  }

  public void Run() {
    for (;;) {                                  // Forever
      bank.Transfer(Util.Random(-10, 10));      //   transfer money
      Util.Pause(200, 300);                     //   then take a break
} } }

class TestBank {
  public static void Main(String[] args) {
    Bank bank = new Bank();
    Clerk clerk1 = new Clerk(bank), clerk2 = new Clerk(bank);
    new Thread(new ThreadStart(clerk1.Run)).Start();
    new Thread(new ThreadStart(clerk2.Run)).Start();
  }
}

// Pseudo-random numbers and sleeping threads

class Util {
  private static readonly Random rnd = new Random();

  public static void Pause(int length) {
    Thread.Sleep(length);
  }

  public static void Pause(int a, int b) {
    Pause(Random(a, b));
  }

  public static int Random(int a, int b) {
    return rnd.Next(a, b);
  }
}
```

```
// Example 166 from page 135 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Threading;

// In general, a while-loop and not an if-statement is needed around
// the Wait operation, in case there are several producers or
// consumers, and somebody may execute Pulse or PulseAll on the buffer
// object.

class Buffer {
  private int contents;
  private bool empty = true;
  public int Get() {
    lock (this) {
      while (empty)
        Monitor.Wait(this);
      empty = true;
      Monitor.PulseAll(this);
      return contents;
  } }
  public void Put(int v) {
    lock (this) {
      while (!empty)
        Monitor.Wait(this);
      empty = false;
      contents = v;
      Monitor.PulseAll(this);
  } }
}

class TestBuffer {
  static readonly Buffer buf = new Buffer();

  public static void Main(String[] args) {
    new Thread(new ThreadStart(producer)).Start();
    new Thread(new ThreadStart(consumer)).Start();
  }

  private static void producer() {
    for (int i=1; true; i++) {
      buf.Put(i);
      Util.Pause(10, 100);
    }
  }

  private static void consumer() {
    for (;;)
      Console.WriteLine("Consumed " + buf.Get());
  }
}

// Pseudo-random numbers and sleeping threads

class Util {
  private static readonly Random rnd = new Random();

  public static void Pause(int length) {
    Thread.Sleep(length);
  }

  public static void Pause(int a, int b) {
    Pause(Random(a, b));
  }

  public static int Random(int a, int b) {
    return rnd.Next(a, b);
  }
}
```

```
// Example 167 from page 135 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Threading;

class Person  {
  String name, fst, snd;

  public Person(String name, String fst, String snd) {
    this.name = name; this.fst = fst; this.snd = snd;
    new Thread(new ThreadStart(Run)).Start();
  }

  public void Run() {
    lock (fst) {
      Console.WriteLine(name + " got " + fst);
      Thread.Sleep(0);          // yield to other threads
      lock (snd)
        { Console.WriteLine(name + " got " + snd); }
      Console.WriteLine(name + " released " + snd);
    }
    Console.WriteLine(name + " released " + fst);
  }
}

class TestDeadlock {
  public static void Main(String[] args) {
    String left  = "left shoe", right = "right shoe";
    new Person("groucho", left, right);
    new Person("harpo", right, left);
  }
}
```

```
// Example 168 from page 137 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Diagnostics;       // For Stopwatch
using System.Threading.Tasks;   // For Parallel

class MyTest {
  public static void Main(String[] args) {
    if (args.Length != 3)
      Console.WriteLine("Usage: MatrixMultiply <aRows> <aCols> <bCols>\n");
    else {
      int
        aRows = int.Parse(args[0]),
        aCols = int.Parse(args[1]),
        bCols = int.Parse(args[2]);
      double[,]
        A = RandomMatrix(aRows, aCols),
        B = RandomMatrix(aCols, bCols),
        R = new double[aRows, bCols];
      {
        Console.WriteLine("Sequential matrix multiplication");
        Stopwatch timer = new Stopwatch();
        timer.Reset();
        timer.Start();
        int count = 10000;
        for (int i=0; i<count; i++)
          Multiply(R, A, B);
        timer.Stop();
        double time = timer.ElapsedMilliseconds * 1E0;
        Console.WriteLine("{0} ms/multiplication", time);
      }
      {
        Console.WriteLine("Parallel matrix multiplication");
        Stopwatch timer = new Stopwatch();
        timer.Reset();
        timer.Start();
        int count = 10000;
        for (int i=0; i<count; i++)
          MultiplyParallel(R, A, B);
        timer.Stop();
        double time = timer.ElapsedMilliseconds * 1E0;
        Console.WriteLine("{0} ms/multiplication", time);
      }
    }
  }

  public static void Multiply(double[,] R, double[,] A, double[,] B) {
    int
      aRows = A.GetLength(0),
      aCols = A.GetLength(1),
      bRows = B.GetLength(0),
      bCols = B.GetLength(1),
      rRows = R.GetLength(0),
      rCols = R.GetLength(1);
    if (aCols==bRows && rRows==aRows && rCols==bCols) {
      for (int r=0; r<rRows; r++)
        for (int c=0; c<rCols; c++) {
          double sum = 0.0;
          for (int k=0; k<aCols; k++)
            sum += A[r,k]*B[k,c];
          R[r,c] = sum;
        }
    }
  }

  public static void MultiplyParallel(double[,] R, double[,] A, double[,] B) {
    int
      aRows = A.GetLength(0),
      aCols = A.GetLength(1),
      bRows = B.GetLength(0),
      bCols = B.GetLength(1),
      rRows = R.GetLength(0),
      rCols = R.GetLength(1);
    if (aCols==bRows && rRows==aRows && rCols==bCols) {
      Parallel.For(0, rRows, r =>
        {
          for (int c=0; c<rCols; c++) {
            double sum = 0.0;
            for (int k=0; k<aCols; k++)
              sum += A[r,k]*B[k,c];
            R[r,c] = sum;
          }
        });
    }
  }

  private static readonly Random rnd = new Random(117);

  public static double[,] RandomMatrix(int rows, int cols) {
    double[,] res = new double[rows, cols];
    for (int r=0; r<rows; r++)
      for (int c=0; c<cols; c++)
        res[r,c] = rnd.NextDouble();
    return res;
  }

  public static void PrintMatrix(double[,] M) {
    Console.WriteLine();
    for (int i=0; i<M.GetLength(0); i++) {
      for (int j=0; j<M.GetLength(1); j++)
        Console.Write("{0}", M[i,j]);
      Console.WriteLine();
    }
  }
}
```

```
// Example 169 from page 137 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Threading.Tasks;           // Parallel
using System.Diagnostics;               // Stopwatch

class MyTest {
  public static void Main(String[] args) {
    {
      Stopwatch stopwatch = new Stopwatch();
      stopwatch.Reset();
      stopwatch.Start();
      Console.WriteLine("Computing SlowFib(40) * 3 + SlowFib(43) = {0}", SequentialSlowFib());
      stopwatch.Stop();
      Console.WriteLine("Sequential: {0} ms", stopwatch.ElapsedMilliseconds);
    }
    {
      Stopwatch stopwatch = new Stopwatch();
      stopwatch.Reset();
      stopwatch.Start();
      Console.WriteLine("Computing SlowFib(40) * 3 + SlowFib(43) = {0}", ParallelSlowFib());
      stopwatch.Stop();
      Console.WriteLine("Parallel:  {0} ms", stopwatch.ElapsedMilliseconds);
    }
  }

  public static double SequentialSlowFib() {
    double fib40 = SlowFib(40);
    double fib43 = SlowFib(43);
    double result = fib40 * 3 + fib43;
    return result;
  }


  public static double ParallelSlowFib() {
    double fib40 = 0.0, fib43 = 0.0;    // Definite assignment rules require initiali
zation
    Parallel.Invoke(delegate { fib40 = SlowFib(40); },
                    delegate { fib43 = SlowFib(43); });
    double result = fib40 * 3 + fib43;
    return result;
  }

  public static double SlowFib(int n) {
    if (n < 2)
      return 1;
    else
      return SlowFib(n-1) + SlowFib(n-2);
  }
}
```

```
// Example 170 from page 137 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;       // IList, List, IEnumerable
using System.Net;                       // WebClient
using System.Linq;                      // from ... select ... syntax
using System.Threading.Tasks;           // Parallel
using System.Text;                      // ASCIIEncoding

class MyTest {
  // Entrez E-utilities at the US National Center for Biotechnology Information:
  static readonly String server = "http://www.ncbi.nlm.nih.gov/entrez/eutils/";

  public static void Main(String[] args) {
    ShowResult(NcbiProtein("P01308"));
    ShowResult(NcbiProteinParallel("P01308", "P01315", "P01317"));
    ShowResult(NcbiProteinParallel2("P01308", "P01315", "P01317"));
  }

  private static void ShowResult(String s) {
    Console.WriteLine("\n----------------------------------------------------------------");
    Console.WriteLine(s);
  }

  private static void ShowResult(IEnumerable<String> ss) {
    Console.WriteLine("\n----------------------------------------------------------------");
    foreach (var s in ss)
      Console.WriteLine(s);
  }

  public static String NcbiEntrez(String query) {
    byte[] bytes = new WebClient().DownloadData(new Uri(server + query));
    return ASCIIEncoding.ASCII.GetString(bytes);
  }

  public static String NcbiProtein(String id) {
    return NcbiEntrez("efetch.fcgi?rettype=fasta&retmode=text&db=protein&id=" + id);
  }

  public static String[] NcbiProteinParallel(params String[] ids) {
    String[] results = new String[ids.Length];
    Parallel.For(0, ids.Length, i => { results[i] = NcbiProtein(ids[i]); });
    return results;
  }

  public static String[] NcbiProteinParallel2(params String[] ids) {
    IList<String> results = new List<String>();
    Parallel.For(0, ids.Length,
                 i => { String res = NcbiProtein(ids[i]);
                        lock (results) results.Add(res); });
    return results.ToArray();
  }
}
```

```
// Example 171 from page 139 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Threading.Tasks;           // Task, Task<T>

class MyTest {
  public static void Main(String[] args) {
    const int n = 43;
    Console.Write("Computing SlowFib({0}) = ", n);
    double result = SlowFib(n);
    Console.WriteLine(result);
    Console.Write("Computing SlowFibTask({0}) = ", n);
    Task<double> task = SlowFibTask(n);         // Returns a Running task
    Console.Write("[task is running] ");
    Console.WriteLine(task.Result);             // Blocks until task completes
  }

  public static double SlowFib(int n) {
    if (n < 2)
      return 1;
    else
      return SlowFib(n-1) + SlowFib(n-2);
  }

  public static Task<double> SlowFibTask(int n) {
    return TaskEx.Run(() => SlowFib(n));
  }
}
```

```
// Example 172 from page 139 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;       // IList, List, IEnumerable
using System.Net;                       // WebClient
using System.Linq;                      // from ... select ... syntax
using System.Threading.Tasks;           // Task<T>
using System.Xml;                       // XmlDocument, XmlNode
using System.Text;                      // ASCIIEncoding

class MyTest {
  // Entrez E-utilities at the US National Center for Biotechnology Information:
  static readonly String server = "http://www.ncbi.nlm.nih.gov/entrez/eutils/";

  public static void Main(String[] args) {
    ShowResult(NcbiProteinTask("P01308").Result);
    ShowResult(NcbiProteinParallelTasks("P01308", "P01315", "P01317").Result);
    // ShowResult(NcbiProteinAsync("P01308").Result);
    // ShowResult(NcbiProteinParallelAsync("P01308", "P01315", "P01317").Result);
    // ShowResult(NcbiSomeProteinAsync("P01308", "P01315", "P01317").Result);
    // ShowResult(NcbiPubmedAsync("molin+s[au]").Result);
    // ShowResult(NcbiPubmedParallelAsync("molin+s[au]", "ingmer+h[au]").Result);
  }

  private static void ShowResult(String s) {
    Console.WriteLine("\n----------------------------------------------------------------------");
    Console.WriteLine(s);
  }

  private static void ShowResult(IEnumerable<String> ss) {
    Console.WriteLine("\n----------------------------------------------------------------------");
    foreach (var s in ss)
      Console.WriteLine(s);
  }

  public static Task<String> NcbiEntrezTask(String query) {
    return new WebClient().DownloadDataTaskAsync(new Uri(server + query))
           .ContinueWith((Task<byte[]> task) =>
                         ASCIIEncoding.ASCII.GetString(task.Result));
  }

  public static Task<String> NcbiProteinTask(String id) {
    return NcbiEntrezTask("efetch.fcgi?rettype=fasta&retmode=text&db=protein&id=" + id);
  }

  public static Task<String[]> NcbiProteinParallelTasks(params String[] ids) {
    IEnumerable<Task<String>> tasks = from id in ids select NcbiProteinTask(id);
    return TaskEx.WhenAll(tasks);
  }

  public static async Task<String> NcbiEntrezAsync(String query) {
    Console.WriteLine(">>>" + query + ">>>");
    byte[] bytes = await new WebClient().DownloadDataTaskAsync(new Uri(server + query
));
    Console.WriteLine("<<<" + query + "<<<");
    return ASCIIEncoding.ASCII.GetString(bytes);
  }

  public static async Task<String> NcbiProteinAsync(String id) {
    return await NcbiEntrezAsync("efetch.fcgi?rettype=fasta&retmode=text&db=protein&id=" + id);
  }

  public static async Task<String[]> NcbiProteinParallelAsync(params String[] ids) {
    IEnumerable<Task<String>> tasks = from id in ids select NcbiProteinAsync(id);
    return await TaskEx.WhenAll(tasks);
  }

  public static async Task<String> NcbiSomeProteinAsync(params String[] ids) {
    IEnumerable<Task<String>> tasks = from id in ids select NcbiProteinAsync(id);
    return await TaskEx.WhenAny(tasks).Result;
  }

  public static async Task<String> NcbiPubmedAsync(String term) {
    String search = String.Format("esearch.fcgi?db=Pubmed&retmax=1&usehistory=y&term={0}", term
);
    XmlDocument xml = new XmlDocument();
    xml.LoadXml(await NcbiEntrezAsync(search));
    XmlNode node = xml["eSearchResult"];
```

```
    String fetch = String.Format("retmax=3&db=Pubmed&query_key={0}&WebEnv={1}",
                                 node["QueryKey"].InnerText, node["WebEnv"].InnerText)
;
    return await NcbiEntrezAsync("efetch.fcgi?rettype=abstract&retmode=text&" + fetch);
  }

  public static async Task<String[]> NcbiPubmedParallelAsync(params String[] terms) {
    IEnumerable<Task<String>> tasks = from term in terms select NcbiPubmedAsync(term)
;
    return await TaskEx.WhenAll(tasks);
  }
}
```

```
// Example 173 from page 139 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Threading.Tasks;

class TestGui {
  public static void Main(String[] args) {
    {
      int n = 37;
      Console.WriteLine(SlowFibTimeout1Task(n++).Result);
      Console.WriteLine(SlowFibTimeout1Task(n++).Result);
      Console.WriteLine(SlowFibTimeout1Task(n++).Result);
      Console.WriteLine(SlowFibTimeout1Task(n++).Result);
      Console.WriteLine(SlowFibTimeout1Task(n++).Result);
    }
    {
      int n = 37;
      Console.WriteLine(SlowFibTimeout2Task(n++).Result);
      Console.WriteLine(SlowFibTimeout2Task(n++).Result);
      Console.WriteLine(SlowFibTimeout2Task(n++).Result);
      Console.WriteLine(SlowFibTimeout2Task(n++).Result);
      Console.WriteLine(SlowFibTimeout2Task(n++).Result);
    }
  }

  public static double SlowFib(int n) {
    if (n < 2)
      return 1;
    else
      return SlowFib(n-1) + SlowFib(n-2);
  }

  public static Task<double> SlowFibTask(int n) {
    return TaskEx.Run(() => SlowFib(n));
  }

  // These two versions of SlowFibTimeout are equivalent:

  public static Task<double> SlowFibTimeout1Task(int n) {
    Task<double> slowFibTask = SlowFibTask(n);
    return TaskEx.WhenAny(slowFibTask, TaskEx.Delay(1000))
                 .ContinueWith<double>((Task<Task> task) =>
                                       task.Result == slowFibTask ? slowFibTask.Resul
t : -1);
  }

  public static async Task<double> SlowFibTimeout2Task(int n) {
    Task<double> slowFibTask = SlowFibTask(n);
    Task completed = await TaskEx.WhenAny(slowFibTask, TaskEx.Delay(1000));
    return completed == slowFibTask ? slowFibTask.Result : -1;
  }
}
```

```
// Example 174 from page 141 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;                   // OperationCanceledException
using System.Threading.Tasks;   // Task, Task<T>
using System.Threading;         // CancellationToken, CancellationTokenSource

class MyTest {
  public static void Main(String[] args) {
    Console.WriteLine("\n-- Cancellation with acknowledgement ----------");
    {
      CancellationTokenSource cts = new CancellationTokenSource();
      CancellationToken token = cts.Token;
      Task task = TaskEx.Run(() => ComputeTaskWithAcknowledgement(token), token);
      Thread.Sleep(0);  // Allow task to be scheduled
      Console.WriteLine(task.Status);   // Running
      cts.Cancel();
      Thread.Sleep(0);
      Console.WriteLine(task.Status);   // Canceled
      try {
        task.Wait();    // Throws AggregateException containing TaskCanceledException
      } catch (Exception exn) {
        Console.WriteLine("Caught " + exn);
      }
      Console.WriteLine(task.Status);   // Canceled
    }
    Console.WriteLine("\n-- Cancellation without acknowledgement ----------");
    {
      CancellationTokenSource cts = new CancellationTokenSource();
      CancellationToken token = cts.Token;
      Task task = TaskEx.Run(() => ComputeTaskWithoutAcknowledgement(token), token);
      Thread.Sleep(0);
      Console.WriteLine(task.Status);   // Running
      cts.Cancel();
      Console.WriteLine(task.Status);   // Running
      task.Wait();
      Console.WriteLine(task.Status);   // RanToCompletion
    }
    Console.WriteLine("\n-- Cancellation before Start ----------------------");
    {
      // Cancel before running
      CancellationTokenSource cts = new CancellationTokenSource();
      CancellationToken token = cts.Token;
      Task task = new Task(delegate { }, token);
      Console.WriteLine(task.Status);  // Created
      cts.Cancel();
      Console.WriteLine(task.Status);   // Canceled
      try {
        task.Start();   // Throws InvalidOperationException
      } catch (Exception exn) {
        Console.WriteLine("Caught " + exn);
      }
      Console.WriteLine(task.Status);   // Canceled
    }
    Console.WriteLine("\n-- Completing before cancellation ----------------");
    {
      CancellationTokenSource cts = new CancellationTokenSource();
      CancellationToken token = cts.Token;
      Task task = new Task(delegate { }, token);
      Console.WriteLine(task.Status);  // Created
      task.Start();
      Thread.Sleep(0);  // Allow task to be scheduled
      Console.WriteLine(task.Status);   // RanToCompletion
      cts.Cancel();
      Console.WriteLine(task.Status);   // RanToCompletion
    }
  }

  public static void ComputeTaskWithAcknowledgement(CancellationToken token) {
    for (int i=0; i<100000000; i++)
      token.ThrowIfCancellationRequested();
  }

  public static void ComputeTaskWithoutAcknowledgement(CancellationToken token) {
    for (int i=0; i<1000000000; i++)
      { /* do nothing */ }
  }
}
```

```
// Example 175 from page 141 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Threading.Tasks;   // Task, Task<T>

class MyTest {
  public static void Main(String[] args) {
    Console.WriteLine("\n-- Faulting a Task with an exception ----------");
    {
      Task task = new Task(delegate { throw new Exception("died"); });
      Console.WriteLine(task.Status);   // Created
      task.Start();
      Console.WriteLine(task.Status);   // Faulted
      try {
        task.Wait();     // Would throw AggregateException containing Exception("died
")
      } catch (Exception exn) {
        Console.WriteLine("Caught " + exn);
      }
      Console.WriteLine(task.Status);   // Faulted
    }
    Console.WriteLine("\n-- Faulting a Task<T> with an exception ----------");
    {
      Task<int> task = new Task<int>(delegate { throw new Exception("died"); });
      Console.WriteLine(task.Status);   // Created
      task.Start();
      Console.WriteLine(task.Status);   // Faulted
      try {
        int res = task.Result; // Throws AggregateException with inner Exception("die
d")
      } catch (Exception exn) {
        Console.WriteLine("Caught " + exn);
      }
      Console.WriteLine(task.Status);   // Faulted
    }
    Console.WriteLine("\n----------------------------------------------------");
  }
}
```

```
// Example 176 from page 143 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;       // IList, List, IEnumerable
using System.Net;                       // WebClient
using System.Linq;                      // from ... select ... syntax
using System.Threading.Tasks;           // Task<T>
using System.Xml;                       // XmlDocument, XmlNode
using System.Text;                      // ASCIIEncoding

class MyTest {
  // Entrez E-utilities at the US National Center for Biotechnology Information:
  static readonly String server = "http://www.ncbi.nlm.nih.gov/entrez/eutils/";

  public static void Main(String[] args) {
    ShowResult(NcbiProteinTask("P01308").Result);
    ShowResult(NcbiProteinParallelTasks("P01308", "P01315", "P01317").Result);
    // ShowResult(NcbiProteinAsync("P01308").Result);
    // ShowResult(NcbiProteinParallelAsync("P01308", "P01315", "P01317").Result);
    // ShowResult(NcbiSomeProteinAsync("P01308", "P01315", "P01317").Result);
    // ShowResult(NcbiPubmedAsync("molin+s[au]").Result);
    // ShowResult(NcbiPubmedParallelAsync("molin+s[au]", "ingmer+h[au]").Result);
  }

  private static void ShowResult(String s) {
    Console.WriteLine("\n-------------------------------------------------------------");
    Console.WriteLine(s);
  }

  private static void ShowResult(IEnumerable<String> ss) {
    Console.WriteLine("\n-------------------------------------------------------------");
    foreach (var s in ss)
      Console.WriteLine(s);
  }

  public static Task<String> NcbiEntrezTask(String query) {
    return new WebClient().DownloadDataTaskAsync(new Uri(server + query))
           .ContinueWith((Task<byte[]> task) =>
                         ASCIIEncoding.ASCII.GetString(task.Result));
  }

  public static Task<String> NcbiProteinTask(String id) {
    return NcbiEntrezTask("efetch.fcgi?rettype=fasta&retmode=text&db=protein&id=" + id);
  }

  public static Task<String[]> NcbiProteinParallelTasks(params String[] ids) {
    IEnumerable<Task<String>> tasks = from id in ids select NcbiProteinTask(id);
    return TaskEx.WhenAll(tasks);
  }

  public static async Task<String> NcbiEntrezAsync(String query) {
    Console.WriteLine(">>>" + query + ">>>");
    byte[] bytes = await new WebClient().DownloadDataTaskAsync(new Uri(server + query
));
    Console.WriteLine("<<<" + query + "<<<");
    return ASCIIEncoding.ASCII.GetString(bytes);
  }

  public static async Task<String> NcbiProteinAsync(String id) {
    return await NcbiEntrezAsync("efetch.fcgi?rettype=fasta&retmode=text&db=protein&id=" + id);
  }

  public static async Task<String[]> NcbiProteinParallelAsync(params String[] ids) {
    IEnumerable<Task<String>> tasks = from id in ids select NcbiProteinAsync(id);
    return await TaskEx.WhenAll(tasks);
  }

  public static async Task<String> NcbiSomeProteinAsync(params String[] ids) {
    IEnumerable<Task<String>> tasks = from id in ids select NcbiProteinAsync(id);
    return await TaskEx.WhenAny(tasks).Result;
  }

  public static async Task<String> NcbiPubmedAsync(String term) {
    String search = String.Format("esearch.fcgi?db=Pubmed&retmax=1&usehistory=y&term={0}", term
);
    XmlDocument xml = new XmlDocument();
    xml.LoadXml(await NcbiEntrezAsync(search));
    XmlNode node = xml["eSearchResult"];
    String fetch = String.Format("retmax=3&db=Pubmed&query_key={0}&WebEnv={1}",
                                 node["QueryKey"].InnerText, node["WebEnv"].InnerText)
;
    return await NcbiEntrezAsync("efetch.fcgi?rettype=abstract&retmode=text&" + fetch);
  }

  public static async Task<String[]> NcbiPubmedParallelAsync(params String[] terms) {
    IEnumerable<Task<String>> tasks = from term in terms select NcbiPubmedAsync(term)
;
    return await TaskEx.WhenAll(tasks);
  }
}
```

```
// Example 177 from page 143 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;       // IList, List, IEnumerable
using System.Net;                       // WebClient
using System.Linq;                      // from ... select ... syntax
using System.Threading.Tasks;           // Task<T>
using System.Xml;                       // XmlDocument, XmlNode
using System.Text;                      // ASCIIEncoding

class MyTest {
  // Entrez E-utilities at the US National Center for Biotechnology Information:
  static readonly String server = "http://www.ncbi.nlm.nih.gov/entrez/eutils/";

  public static void Main(String[] args) {
    ShowResult(NcbiProteinTask("P01308").Result);
    ShowResult(NcbiProteinParallelTasks("P01308", "P01315", "P01317").Result);
    // ShowResult(NcbiProteinAsync("P01308").Result);
    // ShowResult(NcbiProteinParallelAsync("P01308", "P01315", "P01317").Result);
    // ShowResult(NcbiSomeProteinAsync("P01308", "P01315", "P01317").Result);
    // ShowResult(NcbiPubmedAsync("molin+s[au]").Result);
    // ShowResult(NcbiPubmedParallelAsync("molin+s[au]", "ingmer+h[au]").Result);
  }

  private static void ShowResult(String s) {
    Console.WriteLine("\n-------------------------------------------------------------");
    Console.WriteLine(s);
  }

  private static void ShowResult(IEnumerable<String> ss) {
    Console.WriteLine("\n-------------------------------------------------------------");
    foreach (var s in ss)
      Console.WriteLine(s);
  }

  public static Task<String> NcbiEntrezTask(String query) {
    return new WebClient().DownloadDataTaskAsync(new Uri(server + query))
           .ContinueWith((Task<byte[]> task) =>
                         ASCIIEncoding.ASCII.GetString(task.Result));
  }

  public static Task<String> NcbiProteinTask(String id) {
    return NcbiEntrezTask("efetch.fcgi?rettype=fasta&retmode=text&db=protein&id=" + id);
  }

  public static Task<String[]> NcbiProteinParallelTasks(params String[] ids) {
    IEnumerable<Task<String>> tasks = from id in ids select NcbiProteinTask(id);
    return TaskEx.WhenAll(tasks);
  }

  public static async Task<String> NcbiEntrezAsync(String query) {
    Console.WriteLine(">>>" + query + ">>>");
    byte[] bytes = await new WebClient().DownloadDataTaskAsync(new Uri(server + query
));
    Console.WriteLine("<<<" + query + "<<<");
    return ASCIIEncoding.ASCII.GetString(bytes);
  }

  public static async Task<String> NcbiProteinAsync(String id) {
    return await NcbiEntrezAsync("efetch.fcgi?rettype=fasta&retmode=text&db=protein&id=" + id);
  }

  public static async Task<String[]> NcbiProteinParallelAsync(params String[] ids) {
    IEnumerable<Task<String>> tasks = from id in ids select NcbiProteinAsync(id);
    return await TaskEx.WhenAll(tasks);
  }

  public static async Task<String> NcbiSomeProteinAsync(params String[] ids) {
    IEnumerable<Task<String>> tasks = from id in ids select NcbiProteinAsync(id);
    return await TaskEx.WhenAny(tasks).Result;
  }

  public static async Task<String> NcbiPubmedAsync(String term) {
    String search = String.Format("esearch.fcgi?db=Pubmed&retmax=1&usehistory=y&term={0}", term
);
    XmlDocument xml = new XmlDocument();
    xml.LoadXml(await NcbiEntrezAsync(search));
    XmlNode node = xml["eSearchResult"];
    String fetch = String.Format("retmax=3&db=Pubmed&query_key={0}&WebEnv={1}",
                                 node["QueryKey"].InnerText, node["WebEnv"].InnerText)
;
    return await NcbiEntrezAsync("efetch.fcgi?rettype=abstract&retmode=text&" + fetch);
  }

  public static async Task<String[]> NcbiPubmedParallelAsync(params String[] terms) {
    IEnumerable<Task<String>> tasks = from term in terms select NcbiPubmedAsync(term)
;
    return await TaskEx.WhenAll(tasks);
  }
}
```

```
// Example 178 from page 143 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;       // IList, List, IEnumerable
using System.Net;                       // WebClient
using System.Linq;                      // from ... select ... syntax
using System.Threading.Tasks;           // Task<T>
using System.Xml;                       // XmlDocument, XmlNode
using System.Text;                      // ASCIIEncoding

class MyTest {
  // Entrez E-utilities at the US National Center for Biotechnology Information:
  static readonly String server = "http://www.ncbi.nlm.nih.gov/entrez/eutils/";

  public static void Main(String[] args) {
    ShowResult(NcbiProteinTask("P01308").Result);
    ShowResult(NcbiProteinParallelTasks("P01308", "P01315", "P01317").Result);
    // ShowResult(NcbiProteinAsync("P01308").Result);
    // ShowResult(NcbiProteinParallelAsync("P01308", "P01315", "P01317").Result);
    // ShowResult(NcbiSomeProteinAsync("P01308", "P01315", "P01317").Result);
    // ShowResult(NcbiPubmedAsync("molin+s[au]").Result);
    // ShowResult(NcbiPubmedParallelAsync("molin+s[au]", "ingmer+h[au]").Result);
  }

  private static void ShowResult(String s) {
    Console.WriteLine("\n-------------------------------------------------------------");
    Console.WriteLine(s);
  }

  private static void ShowResult(IEnumerable<String> ss) {
    Console.WriteLine("\n-------------------------------------------------------------");
    foreach (var s in ss)
      Console.WriteLine(s);
  }

  public static Task<String> NcbiEntrezTask(String query) {
    return new WebClient().DownloadDataTaskAsync(new Uri(server + query))
           .ContinueWith((Task<byte[]> task) =>
                         ASCIIEncoding.ASCII.GetString(task.Result));
  }

  public static Task<String> NcbiProteinTask(String id) {
    return NcbiEntrezTask("efetch.fcgi?rettype=fasta&retmode=text&db=protein&id=" + id);
  }

  public static Task<String[]> NcbiProteinParallelTasks(params String[] ids) {
    IEnumerable<Task<String>> tasks = from id in ids select NcbiProteinTask(id);
    return TaskEx.WhenAll(tasks);
  }

  public static async Task<String> NcbiEntrezAsync(String query) {
    Console.WriteLine(">>>" + query + ">>>");
    byte[] bytes = await new WebClient().DownloadDataTaskAsync(new Uri(server + query
));
    Console.WriteLine("<<<" + query + "<<<");
    return ASCIIEncoding.ASCII.GetString(bytes);
  }

  public static async Task<String> NcbiProteinAsync(String id) {
    return await NcbiEntrezAsync("efetch.fcgi?rettype=fasta&retmode=text&db=protein&id=" + id);
  }

  public static async Task<String[]> NcbiProteinParallelAsync(params String[] ids) {
    IEnumerable<Task<String>> tasks = from id in ids select NcbiProteinAsync(id);
    return await TaskEx.WhenAll(tasks);
  }

  public static async Task<String> NcbiSomeProteinAsync(params String[] ids) {
    IEnumerable<Task<String>> tasks = from id in ids select NcbiProteinAsync(id);
    return await TaskEx.WhenAny(tasks).Result;
  }

  public static async Task<String> NcbiPubmedAsync(String term) {
    String search = String.Format("esearch.fcgi?db=Pubmed&retmax=1&usehistory=y&term={0}", term
);
    XmlDocument xml = new XmlDocument();
    xml.LoadXml(await NcbiEntrezAsync(search));
    XmlNode node = xml["eSearchResult"];
    String fetch = String.Format("retmax=3&db=Pubmed&query_key={0}&WebEnv={1}",
                                 node["QueryKey"].InnerText, node["WebEnv"].InnerText)
;
    return await NcbiEntrezAsync("efetch.fcgi?rettype=abstract&retmode=text&" + fetch);
  }

  public static async Task<String[]> NcbiPubmedParallelAsync(params String[] terms) {
    IEnumerable<Task<String>> tasks = from term in terms select NcbiPubmedAsync(term)
;
    return await TaskEx.WhenAll(tasks);
  }
}
```

```
// Example 179 from page 143 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Threading.Tasks;

class TestGui {
  public static void Main(String[] args) {
    {
      int n = 37;
      Console.WriteLine(SlowFibTimeout1Task(n++).Result);
      Console.WriteLine(SlowFibTimeout1Task(n++).Result);
      Console.WriteLine(SlowFibTimeout1Task(n++).Result);
      Console.WriteLine(SlowFibTimeout1Task(n++).Result);
      Console.WriteLine(SlowFibTimeout1Task(n++).Result);
    }
    {
      int n = 37;
      Console.WriteLine(SlowFibTimeout2Task(n++).Result);
      Console.WriteLine(SlowFibTimeout2Task(n++).Result);
      Console.WriteLine(SlowFibTimeout2Task(n++).Result);
      Console.WriteLine(SlowFibTimeout2Task(n++).Result);
      Console.WriteLine(SlowFibTimeout2Task(n++).Result);
    }
  }

  public static double SlowFib(int n) {
    if (n < 2)
      return 1;
    else
      return SlowFib(n-1) + SlowFib(n-2);
  }

  public static Task<double> SlowFibTask(int n) {
    return TaskEx.Run(() => SlowFib(n));
  }

  // These two versions of SlowFibTimeout are equivalent:

  public static Task<double> SlowFibTimeout1Task(int n) {
    Task<double> slowFibTask = SlowFibTask(n);
    return TaskEx.WhenAny(slowFibTask, TaskEx.Delay(1000))
                 .ContinueWith<double>((Task<Task> task) =>
                                       task.Result == slowFibTask ? slowFibTask.Resul
t : -1);
  }

  public static async Task<double> SlowFibTimeout2Task(int n) {
    Task<double> slowFibTask = SlowFibTask(n);
    Task completed = await TaskEx.WhenAny(slowFibTask, TaskEx.Delay(1000));
    return completed == slowFibTask ? slowFibTask.Result : -1;
  }
}
```

```
// Example 180 from page 145 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MathFactorial {
  public static void Main(String[] args) {
    for (int i=0; i<=100; i++)
      Console.WriteLine(i + "!=" + Fact(i));
  }

  static double Fact(int n) {
    double res = 0.0;
    for (int i=1; i<=n; i++)
      res += Math.Log(i);
    return Math.Exp(res);
  }
}
```

```
// Example 181 from page 145 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MathGaussian {
  public static void Main(String[] args) {
    PrintGaussians(100);
  }

  // From http://www.taygeta.com/random/gaussian.html 2001-09-21:
  // The most basic form of the transformation looks like:
  //               y1 = sqrt( - 2 ln(x1) ) cos( 2 pi x2 )
  //               y2 = sqrt( - 2 ln(x1) ) sin( 2 pi x2 )

  // We start with two independent random numbers, x1 and x2, which
  // come from a uniform distribution (in the range from 0 to 1). Then
  // apply the above transformations to get two new independent random
  // numbers which have a Gaussian distribution with zero mean and a
  // standard deviation of one.

  static void PrintGaussians(int n) {
    Random rnd = new Random();
    for (int i=0; i<n; i+=2) {
      double x1 = rnd.NextDouble(), x2 = rnd.NextDouble();
      Print(Math.Sqrt(-2 * Math.Log(x1)) * Math.Cos(2 * Math.PI * x2));
      Print(Math.Sqrt(-2 * Math.Log(x1)) * Math.Sin(2 * Math.PI * x2));
    }
  }

  static void Print(double d) {
    Console.WriteLine(d);
  }
}
```

```
// Example 182 from page 145 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

class MathSpecial {
  public static void Main(String[] args) {
    MathTest();
  }

  static void MathTest() {
    Print("Illegal arguments, NaN results:");
    Print(Math.Sqrt(-1));                 // NaN
    Print(Math.Log(-1));                  // NaN
    Print(Math.Pow(-1, 2.5));             // NaN
    Print(Math.Acos(1.1));                // NaN
    Print("Infinite results:");
    Print(Math.Log(0));                   // -Infinity
    Print(Math.Pow(0, -1));               // Infinity
    Print(Math.Exp(1000.0));              // Infinity (overflow)
    Print("Infinite arguments:");
    double infinity = Double.PositiveInfinity;
    Print(Math.Sqrt(infinity));           // Infinity
    Print(Math.Log(infinity));            // Infinity
    Print(Math.Exp(-infinity));           // 0
    Print(Math.Pow(infinity, 0.5));       // Infinity
    Print(Math.Pow(0.5, infinity));       // 0
    Print(Math.Pow(0.5, -infinity));      // Infinity
    Print(Math.Pow(2, infinity));         // Infinity
    Print(Math.Pow(2, -infinity));        // 0
    Print("Special cases:");
    Print(Math.Pow(0, 0));                // 1.0
    Print(Math.Pow(infinity, 0));         // 1.0
    Print(Math.Pow(-infinity, 0));        // 1.0
    Print(Math.Pow(-infinity, 0.5));      // Infinity
    Print(Math.Pow(1, infinity));         // NaN
    Print(Math.Pow(1, -infinity));        // NaN
    // For all (x, y) except (0.0, 0.0):
    // sign(Cos(Atan2(y, x))) == sign(x) && sign(Sin(Atan2(y, x))) == sign(y)
    for (double x=-100; x<=100; x+=0.125) {
      for (double y=-100; y<=100; y+=0.125) {
        double r = Math.Atan2(y, x);
        if (!(sign(Math.Cos(r))==sign(x) && sign(Math.Sin(r))==sign(y)))
          Print("x = " + x + "; y = " + y);
      }
    }
  }

  // The built-in Math.Sign method cannot be used because Sin and
  // Cos are inexact

  static int sign(double x) {
    double tolerance = 1E-14;
    if (x < -tolerance)
      return -1;
    else if (x > +tolerance)
      return +1;
    else
      return 0;
  }

  static void Print(String d) {
    Console.WriteLine(d);
  }

  static void Print(double d) {
    Console.WriteLine(d);
  }

  static void Print(long d) {
    Console.WriteLine(d);
  }
}
```

```
// Example 183 from page 147 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.IO;

class BasicIOExample {
  public static void Main() {
    TextReader r = Console.In;
    int count = 0;
    String s = r.ReadLine();
    while (s != null && !s.Equals("")) {
      count++;
      s = r.ReadLine();
    }
    Console.WriteLine("You entered " + count + " nonempty lines");
  }
}
```

```
// Example 184 from page 149 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;  // BinaryFormatter

public class IOExample {
  public static void Main() {

    // Write numbers and words on file "f.txt" in human-readable form:
    TextWriter twr = new StreamWriter(new FileStream("f.txt", FileMode.Create));
    twr.Write(4711); twr.Write(' '); twr.Write("cool"); twr.Close();

    // Write primitive values to a binary file "p.dat":
    BinaryWriter bwr = new BinaryWriter(new FileStream("p.dat", FileMode.Create));
    bwr.Write(4711); bwr.Write(' '); bwr.Write("cool"); bwr.Close();

    // Read primitive values from binary file "p.dat":
    BinaryReader brd = new BinaryReader(new FileStream("p.dat", FileMode.Open));
    Console.WriteLine(brd.ReadInt32() + "|" + brd.ReadChar() + "|" + brd.ReadString())
;

    // Write an object or array to binary file "o.dat":
    FileStream fs1 = new FileStream("o.dat", FileMode.Create);
    BinaryFormatter bf = new BinaryFormatter();
    bf.Serialize(fs1, new int[] { 2, 3, 5, 7, 11 }); fs1.Close();

    // Read objects or arrays from binary file "o.dat":
    FileStream fs2 = new FileStream("o.dat", FileMode.Open);
    int[] ia = (int[]) bf.Deserialize(fs2);
    Console.WriteLine("{0} {1} {2} {3} {4}", ia[0], ia[1], ia[2], ia[3], ia[4]); fs2.Close
();

    // Read and write parts of file "raf.dat" in arbitrary order:
    FileStream fs = new FileStream("raf.dat", FileMode.OpenOrCreate, FileAccess.ReadWri
te);
    BinaryWriter bw = new BinaryWriter(fs);
    bw.Write(3.1415); bw.Write(42);
    fs.Seek(0, SeekOrigin.Begin);
    BinaryReader br = new BinaryReader(fs);
    Console.WriteLine("{0} {1}", br.ReadDouble(), br.ReadInt32());

    // Read from a String as if it were a text file:
    TextReader tr = new StringReader("abc");
    Console.WriteLine("abc: " + (char)tr.Read() + (char)tr.Read() + (char)tr.Read());

    // Write to a StringBuffer as if it were a text file:
    TextWriter tw = new StringWriter();
    tw.Write('d'); tw.Write('e'); tw.Write('f');
    Console.WriteLine(tw.ToString());

    // Write characters to standard output and standard error:
    Console.Out.WriteLine("std output"); Console.Error.WriteLine("std error");

    // Read characters from standard input (the keyboard):
    Console.WriteLine("Type some characters and press Enter: ");
    TextReader intext = Console.In;
    String response = intext.ReadLine();
    Console.WriteLine("You typed: '{0}'", response);

    // Read a character from standard input (the keyboard):
    Console.Write("Type one character and press Enter: ");
    char c = (char)Console.In.Read();
    Console.WriteLine("First character of your input is: " + c);
  }
}
```

```
// Example 185 from page 151 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Read text file one line at a time, parse one number from each line,
// and compute the sum of these numbers.  NB: double.Parse respects
// the current culture, so one must use culture en-US or similar to
// parse a number whose decimal point is a period (.).

using System;
using System.IO;                // StreamReader, TextReader
using System.Threading;         // Thread
using System.Globalization;     // CultureInfo

class MyTest {
  public static void Main(String[] args) {
    Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
    // = new CultureInfo("fr-FR");      // France
    // = new CultureInfo("de-DE");      // Germany
    // = new CultureInfo("da-DK");      // Denmark
    double sum = 0.0;
    TextReader rd = new StreamReader("foo");
    String line;
    while (null != (line = rd.ReadLine()))
      sum += double.Parse(line);
    rd.Close();
    Console.WriteLine("The sum is {0}", sum);
  }
}
```

```
// Example 186 from page 151 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.IO;                // StringReader, TextReader
using System.Text;              // StringBuilder

class MyTest {
  public static void Main(String[] args) {
    if (args.Length == 1)
      Tokenize(new StringReader(args[0]));
    else
      Tokenize(new StringReader("(6 + abc2) * 3343"));
  }

  public static void Tokenize(TextReader rd) {
    while (rd.Peek() != -1) {
      if (Char.IsWhiteSpace((char)rd.Peek()))           // Whitespace, skip
        rd.Read();
      else if (Char.IsDigit((char)rd.Peek())) {         // Number
        int val = rd.Read() - '0';
        while (Char.IsDigit((char)rd.Peek()))
          val = 10 * val + rd.Read() - '0';
        Console.WriteLine(new Int(val));
      } else if (Char.IsLetter((char)rd.Peek())) {      // Identifier
        StringBuilder id = new StringBuilder().Append((char)rd.Read());
        while (Char.IsLetterOrDigit((char)rd.Peek()))
          id.Append((char)rd.Read());
        Console.WriteLine(new Id(id.ToString()));
      } else
        switch (rd.Peek()) {
        case '+': case '-': case '*': case '/':         // Operator
          Console.WriteLine(new Op((char)rd.Read())); break;
        case '(': case ')':                             // Separator
          Console.WriteLine(new Sep((char)rd.Read())); break;
        default:                                         // Illegal token
          throw new ApplicationException("Illegal character '"+(char)rd.Peek()+"'");
        }
    }
  }
}

// Classes to represent tokens: identifiers, numbers, operators, delimiters

abstract class Token { }

class Int : Token {
  public readonly int i;

  public Int(int i) {
    this.i = i;
  }

  public override String ToString() {
    return String.Format("int:{0}", i);
  }
}

class Id : Token {
  public readonly String id;

  public Id(String id) {
    this.id = id;
  }

  public override String ToString() {
    return String.Format("id:{0}", id);
  }
}

class Op : Token {
  public readonly char op;

  public Op(char op) {
    this.op = op;
  }

  public override String ToString() {
    return String.Format("op:{0}", op);
```

```
  }
}

class Sep : Token {
  public readonly char sep;

  public Sep(char sep) {
    this.sep = sep;
  }

  public override String ToString() {
    return String.Format("sep:{0}", sep);
  }
}
```

```
// Example 187 from page 153 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.IO;                // StreamWriter, Textwriter

public class TextWriterExample {
  public static void Main() {
    TextWriter tw = new StreamWriter("dice.txt");
    Random rnd = new Random();
    for (int i=1; i<=1000; i++) {
      int die = (int)(1 + 6 * rnd.NextDouble());
      tw.Write(die); tw.Write(' ');
      if (i % 20 == 0) tw.WriteLine();
    }
    tw.Close();                 // Without this, the output file may be empty
  }
}
```

```
// Example 188 from page 153 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.IO;                // FileStream, StreamWriter, Textwriter

public class TextWriterExample {
  public static void Main() {
    TextWriter tw = new StreamWriter(new FileStream("temperature.html", FileMode.Create))
;
    tw.WriteLine("<table border><tr><th>Fahrenheit<th>Celsius</tr>");
    for (double f=100; f<=400; f+=10) {
      double c = 5 * (f - 32) / 9;
      tw.WriteLine("<tr align=right><td>{0:#0}<td>{1:0.0}", f, c);
    }
    tw.WriteLine("</table>");
    tw.Close();                 // Without this, the output file may be empty
  }
}
```

```
// Example 190 from page 155 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.IO;

public class BinaryIOExample {
  public static void Main() {
    BinaryWriter bw =
      new BinaryWriter(new FileStream("tmp1.dat", FileMode.Create));
    WriteData(bw); bw.Close();
    BinaryReader br =
      new BinaryReader(new FileStream("tmp1.dat", FileMode.Open));
    ReadData(br);
  }

  static void WriteData(BinaryWriter bw) {
    bw.Write(true);                          // Write 1 byte
    bw.Write((byte)120);                     // Write 1 byte
    bw.Write('A');                           // Write 1 byte (UTF-8)
    bw.Write("foo");                         // Write 1+3 bytes (UTF-8)
    bw.Write("Rhône");                       // Write 1+6 bytes (UTF-8)
    bw.Write(300.1);                         // Write 8 bytes
    bw.Write(300.2F);                        // Write 4 bytes
    bw.Write(1234);                          // Write 4 bytes
    bw.Write(12345L);                        // Write 8 bytes
    bw.Write((short)32000);                  // Write 2 bytes
    bw.Write((sbyte)-1);                     // Write 1 byte
    bw.Write((short)-1);                     // Write 2 bytes
  }

  static void ReadData(BinaryReader br) {
    Console.Write(      br.ReadBoolean());   // Read 1 byte
    Console.Write(" " + br.ReadByte());      // Read 1 byte
    Console.Write(" " + br.ReadChar());      // Read 1 byte
    Console.Write(" " + br.ReadString());    // Read 1+3 bytes
    Console.Write(" " + br.ReadString());    // Read 1+6 bytes
    Console.Write(" " + br.ReadDouble());    // Read 8 bytes
    Console.Write(" " + br.ReadSingle());    // Read 4 bytes
    Console.Write(" " + br.ReadInt32());     // Read 4 bytes
    Console.Write(" " + br.ReadInt64());     // Read 8 bytes
    Console.Write(" " + br.ReadInt16());     // Read 2 bytes
    Console.Write(" " + br.ReadSByte());     // Read 1 byte
    Console.Write(" " + br.ReadUInt16());    // Read 2 bytes
    Console.WriteLine();
  }
}
```

```
// Example 192 from page 157 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;
using System.IO;

public class RandomAccessFileExample {
  public static void Main() {
    String[] dna = { "TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT",
                     "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC",
                     "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
                     "GGGGGGGGGGGGGGGGGGGGGGGG" };
    WriteStrings("dna.dat", dna);
  }

  static void WriteStrings(String filename, String[] dna) {
    FileStream raf = new FileStream(filename, FileMode.Create);
    BinaryWriter sw = new BinaryWriter(raf);
    raf.SetLength(0);                                  // Truncate the file
    List<long> offsettable = new List<long>();
    foreach (String s in dna) {
      offsettable.Add(raf.Position);                   // Store string offset
      sw.Write(s);                                     // Write string
    }

    foreach (long v in offsettable) {                  // Write string offsets
      Console.WriteLine(v);
      sw.Write(v);
    }
    sw.Write(offsettable.Count);                       // Write string count
    sw.Close();
  }
}
```

```
// Example 193 from page 157 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.IO;

public class RandomAccessFileExample {
  public static void Main() {
    for (int i=0; i<4; i++)
      Console.WriteLine(ReadOneString("dna.dat", i));
  }

  static String ReadOneString(String filename, int i) {
    const int IntSize = 4, LongSize = 8;
    FileStream raf = new FileStream(filename, FileMode.Open);
    raf.Seek(raf.Length - IntSize, SeekOrigin.Begin);
    BinaryReader br = new BinaryReader(raf);
    int N = br.ReadInt32();
    raf.Seek(raf.Length - IntSize - LongSize * N + LongSize * i, SeekOrigin.Begin);
    long si = br.ReadInt64();
    raf.Seek(si, SeekOrigin.Begin);
    String s = br.ReadString();
    br.Close();
    return s;
  }
}
```

```
// Example 194 from page 161 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.IO;        // Directory, DirectoryInfo, FileInfo

public class DirectoryHierarchyExample {
  public static void ShowDir(int indent, DirectoryInfo dir) {
    Indent(indent); Console.WriteLine(dir.Name);
    DirectoryInfo[] subdirs = dir.GetDirectories();
    foreach (DirectoryInfo d in subdirs)
      ShowDir(indent+4, d);
    FileInfo[] files = dir.GetFiles();
    foreach (FileInfo file in files) {
      Indent(indent); Console.WriteLine(file.Name);
    }
  }

  public static void Indent(int indent) {
    for (int i=0; i<indent; i++)
      Console.Write('-');
  }

  public static void Main() {
    DirectoryInfo dir = new DirectoryInfo(Directory.GetCurrentDirectory());
    ShowDir(0, dir);
  }
}
```

```
// Example 195 from page 161 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.IO;        // FileInfo, StreamReader

class MyTest {
  public static void Main(String[] args) {
    FileInfo fi1 = new FileInfo("example3\\Prog.cs.old" ); // Windows, Relative
    Console.WriteLine(fi1.Extension);                         // Extension is ".old"
    FileInfo fi2 = new FileInfo("c:tmp\\foo");              // Windows, Volume+relative
    Console.WriteLine(fi2.Extension);                         // Extension is ""
    FileInfo fi3 = new FileInfo("c:\\tmp\\foo");           // Windows, Volume+absolute
    FileInfo fi4 = new FileInfo("example3/Prog.cs");       // Unix, Relative
    Console.WriteLine(fi4.Name);                              // Prog.cs
    Console.WriteLine(fi4.FullName);                          // C:\tmp\example3\Prog.cs
    FileInfo fi5 = new FileInfo("/etc/passwd");             // Unix, Absolute
    Console.WriteLine("--- Printing contents of {0} ---", fi4.Name);
    StreamReader sr = fi4.OpenText();
    String line;
    while ((line = sr.ReadLine()) != null)
      Console.WriteLine(line);
    sr.Close();
  }
}
```

```
// Example 196 from page 163 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.IO;           // BinaryReader, BinaryWriter
using System.Net;          // AddressFamily, Dns, IPAddress, ProtocolType, ...
using System.Net.Sockets;  // NetworkStream, Socket, SocketType,

class Example196 {
  const int PortNo = 2357;

  public static void Main(String[] args) {
    bool server = (args.Length == 1 && args[0] == "server");
    bool client = (args.Length == 2 && args[0] == "client");
    if (server) {              // Server: accept questions about primality
      Socket serversocket =
        new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
      serversocket.Bind(new IPEndPoint(IPAddress.Any, PortNo));
      serversocket.Listen(10);  // Max queue = 10 connections.
      for (;;) {               // For ever, accept connections
        NetworkStream s = new NetworkStream(serversocket.Accept());
        BinaryReader input  = new BinaryReader(s);
        BinaryWriter output = new BinaryWriter(s);
        int number = input.ReadInt32();
        output.Write(IsPrime(number));
        input.Close(); output.Close();
      }
    } else if (client) {       // Client: ask questions about primality
      IPAddress ipa = Dns.GetHostEntry(args[1]).AddressList[0];
      for (int i=1; i<100; i++) {
        Socket clientsocket =
          new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp)
;
        clientsocket.Connect(new IPEndPoint(ipa, PortNo));
        NetworkStream n = new NetworkStream(clientsocket);
        BinaryWriter output = new BinaryWriter(n);
        BinaryReader input = new BinaryReader(n);
        output.Write(i);
        if (input.ReadBoolean())
          Console.Write(i + " ");
        output.Close(); input.Close();
      }
    } else {                   // Neither server nor client
      Console.WriteLine("Start two copies of this program, possibly on different machines:");
      Console.WriteLine("  Example196 server");
      Console.WriteLine("  Example196 client <serverhostname>");
      Console.WriteLine("Use 'Example196 client localhost' if the");
      Console.WriteLine("client and server run on the same machine.");
      Console.WriteLine("You may start several clients all talking to the same server.");
    }
  }

  static bool IsPrime(int p) {
    if (p == 2)
      return true;
    if (p == 1 || p % 2 == 0)
      return false;
    for (int q=3; q*q<=p; q+=2)
      if (p % q == 0)
        return false;
    return true;
  }
}
```

```
// Example 197 from page 165 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Object Arraylist: no compile-time type check

using System;
using SC = System.Collections;

class MyTest {
  public static void Main(String[] args) {
    SC.ArrayList cool = new SC.ArrayList();    // Needs: using SC = System.Collection
s;
    cool.Add(new Person("Kristen"));
    cool.Add(new Person("Bjarne"));
    cool.Add(new Exception("Larry"));          // Wrong, but no compiletime check
    cool.Add(new Person("Anders"));
    Person p = (Person)cool[2];                // Compiles OK, but throws at runtime
  }
}

class Person {
  private static int counter = 0;
  private readonly String name;
  private readonly int serialNumber;

  public Person(String name) {
    this.name = name;
    this.serialNumber = counter++;
  }
}
```

```
// Example 198 from page 165 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Generic Arraylist: compile-time type check, no run-time checks needed

using System;
using System.Collections.Generic;

class MyTest {
  public static void Main(String[] args) {
    List<Person> cool = new List<Person>();
    cool.Add(new Person("Kristen"));
    cool.Add(new Person("Bjarne"));
    //    cool.Add(new Exception("Larry"));   // Wrong, detected at compile-time
    cool.Add(new Person("Anders"));
    Person p = (Person)cool[2];         // No run-time check needed
  }
}

class Person {
  private static int counter = 0;
  private readonly String name;
  private readonly int serialNumber;

  public Person(String name) {
    this.name = name;
    this.serialNumber = counter++;
  }
}
```

```
// Example 199 from page 165 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class Log<T> {
  private const int SIZE = 5;
  public static int InstanceCount { get; private set; }
  public int Count { get; private set; }
  private T[] log = new T[SIZE];
  public Log() { InstanceCount++; }
  public void Add(T msg) { log[Count++ % SIZE] = msg; }
  public T Last {
    get { // Return the last log entry, or null if nothing logged yet
      return Count==0 ? default(T) : log[(Count-1)%SIZE];
    }
    set { // Update the last log entry, or create one if nothing logged yet
      if (Count==0)
        log[Count++] = value;
      else
        log[(Count-1)%SIZE] = value;
    }
  }
  public T[] All {
    get {
      int size = Math.Min(Count, SIZE);
      T[] res = new T[size];
      for (int i=0; i<size; i++)
        res[i] = log[(Count-size+i) % SIZE];
      return res;
    }
  }
}

class TestLog {
  class MyTest {
    public static void Main(String[] args) {
      Log<String> log1 = new Log<String>();
      log1.Add("Reboot");
      log1.Add("Coffee");
      Log<DateTime> log2 = new Log<DateTime>();
      log2.Add(DateTime.Now);
      log2.Add(DateTime.Now.AddHours(1));
      DateTime[] dts = log2.All;
      // Printing both logs:
      foreach (String s in log1.All)
        Console.Write("{0} ", s);
      Console.WriteLine();
      foreach (DateTime dt in dts)
        Console.Write("{0} ", dt);
      Console.WriteLine();
      TestPairLog();
    }

    public static void TestPairLog() {
      Log<Pair<DateTime,String>> log = new Log<Pair<DateTime,String>>();
      log.Add(new Pair<DateTime,String>(DateTime.Now, "Tea leaves"));
      log.Add(new Pair<DateTime,String>(DateTime.Now.AddMinutes(2), "Hot water"));
      log.Add(new Pair<DateTime,String>(DateTime.Now.AddMinutes(7), "Ready"));
      Pair<DateTime,String>[] allMsgs = log.All;
      foreach (Pair<DateTime,String> p in allMsgs)
        Console.WriteLine("At {0}: {1}", p.Fst, p.Snd);
    }
  }
}

public struct Pair<T,U> {
  public readonly T Fst;
  public readonly U Snd;
  public Pair(T fst, U snd) {
    this.Fst = fst;
    this.Snd = snd;
  }
}
```

```
// Example 200 from page 167 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// A generic LinkedList class

using System;
using System.IO;                        // TextWriter
using System.Collections.Generic;       // IEnumerable<T>, IEnumerator<T>
using SC = System.Collections;          // IEnumerable, IEnumerator


public interface IMyList<T> : IEnumerable<T>, IEquatable<IMyList<T>> {
  int Count { get; }                    // Number of elements
  T this[int i] { get; set; }           // Get or set element at index i
  void Add(T item);                     // Add element at end
  void Insert(int i, T item);           // Insert element at index i
  void RemoveAt(int i);                 // Remove element at index i
  IMyList<U> Map<U>(Func<T,U> f);       // Map f over all elements
}

public class LinkedList<T> : IMyList<T> {
  protected int size;                // Number of elements in the list
  protected Node first, last;        // Invariant: first==null iff last==null

  protected class Node {
    public Node prev, next;
    public T item;

    public Node(T item) {
      this.item = item;
    }

    public Node(T item, Node prev, Node next) {
      this.item = item; this.prev = prev; this.next = next;
    }
  }

  public LinkedList() {
    first = last = null;
    size = 0;
  }

  public LinkedList(params T[] arr) : this() {
    foreach (T x in arr)
      Add(x);
  }

  public int Count {
    get { return size; }
  }

  public T this[int i] {
    get { return get(i).item; }
    set { get(i).item = value; }
  }

  private Node get(int n) {
    if (n < 0 || n >= size)
      throw new IndexOutOfRangeException();
    else if (n < size/2) {              // Closer to front
      Node node = first;
      for (int i=0; i<n; i++)
        node = node.next;
      return node;
    } else {                            // Closer to end
      Node node = last;
      for (int i=size-1; i>n; i--)
        node = node.prev;
      return node;
    }
  }

  public void Add(T item) {
    Insert(size, item);
  }

  public void Insert(int i, T item) {
    if (i == 0) {
      if (first == null) // and thus last == null
        first = last = new Node(item);
      else {
        Node tmp = new Node(item, null, first);
        first.prev = tmp;
        first = tmp;
      }
      size++;
    } else if (i == size) {
      if (last == null) // and thus first = null
        first = last = new Node(item);
      else {
        Node tmp = new Node(item, last, null);
        last.next = tmp;
        last = tmp;
      }
      size++;
    } else {
      Node node = get(i);
      // assert node.prev != null;
      Node newnode = new Node(item, node.prev, node);
      node.prev.next = newnode;
      node.prev = newnode;
      size++;
    }
  }

  public void RemoveAt(int i) {
    Node node = get(i);
    if (node.prev == null)
      first = node.next;
    else
      node.prev.next = node.next;
    if (node.next == null)
      last = node.prev;
    else
      node.next.prev = node.prev;
    size--;
  }

  public override bool Equals(Object that) {
    return Equals(that as IMyList<T>);
  }

  public bool Equals(IMyList<T> that) {
    if (this == that)
      return true;
    if (that == null || this.Count != that.Count)
      return false;
    Node thisnode = this.first;
    IEnumerator<T> thatenm = that.GetEnumerator();
    while (thisnode != null) {
      if (!thatenm.MoveNext())
        throw new ApplicationException("Impossible: LinkedList<T>.Equals");
      // assert MoveNext() was true (because of the above size test)
      if (!thisnode.item.Equals(thatenm.Current))
        return false;
      thisnode = thisnode.next;
    }
    // assert !MoveNext(); // because of the size test
    return true;
  }

  public override int GetHashCode() {
    int hash = 0;
    foreach (T x in this)
      hash ^= x.GetHashCode();
    return hash;
  }

  public static explicit operator LinkedList<T>(T[] arr) {
    return new LinkedList<T>(arr);
  }

  public static LinkedList<T> operator +(LinkedList<T> xs1, LinkedList<T> xs2) {
    LinkedList<T> res = new LinkedList<T>();
    foreach (T x in xs1)
      res.Add(x);
    foreach (T x in xs2)
      res.Add(x);
```

```
    return res;
  }

  public IMyList<U> Map<U>(Func<T,U> f) {
    LinkedList<U> res = new LinkedList<U>();
    foreach (T x in this)
      res.Add(f(x));
    return res;
  }

  public IEnumerator<T> GetEnumerator() {
    return new LinkedListEnumerator(this);
  }

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

  private class LinkedListEnumerator : IEnumerator<T> {
    T curr;                    // The enumerator's current element
    bool valid;                // Is the current element valid?
    Node next;                 // Node holding the next element, or null

    public LinkedListEnumerator(LinkedList<T> lst) {
      next = lst.first; valid = false;
    }

    public T Current {
      get {
        if (valid)
          return curr;
        else
          throw new InvalidOperationException();
      }
    }

    public bool MoveNext() {
      if (next != null)  {
        curr = next.item; next = next.next; valid = true;
      } else
        valid = false;
      return valid;
    }

    public void Dispose() {
      curr = default(T);
      next = null; valid = false;
    }

    Object SC.IEnumerator.Current {
      get { return Current; }
    }

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

class SortedList<T> : LinkedList<T> where T : IComparable<T> {
  // Sorted insertion
  public void Insert(T x) {
    Node node = first;
    while (node != null && x.CompareTo(node.item) > 0)
      node = node.next;
    if (node == null)          // x > all elements; insert at end
      Add(x);
    else {                     // x <= node.item; insert before node
      Node newnode = new Node(x);
      if (node.prev == null)   // insert as first element
        first = newnode;
      else
        node.prev.next = newnode;
      newnode.next = node;
      newnode.prev = node.prev;
      node.prev = newnode;
    }
  }
}
```

```
interface IPrintable {
  void Print(TextWriter fs);
}
class PrintableLinkedList<T> : LinkedList<T>, IPrintable where T : IPrintable {
  public void Print(TextWriter fs) {
    bool firstElement = true;
    foreach (T x in this) {
      x.Print(fs);
      if (firstElement)
        firstElement = false;
      else
        fs.Write(",");
    }
  }
}

class MyString : IComparable<MyString> {
  private readonly String s;
  public MyString(String s) {
    this.s = s;
  }
  public int CompareTo(MyString that) {
    return String.Compare(that.Value, s);       // Reverse ordering
  }
  public bool Equals(MyString that) {
    return String.Equals(that.Value, s);
  }
  public String Value {
    get { return s; }
  }
}

class MyTest {
  public static void Main(String[] args) {
    LinkedList<double> dLst = new LinkedList<double>(7.0, 9.0, 13.0, 0.0);
    foreach (double d in dLst)
      Console.Write("{0} ", d);
    Console.WriteLine();
    IMyList<int> iLst = dLst.Map<int>(Math.Sign);
    foreach (int i in iLst)
      Console.Write("{0} ", i);
    Console.WriteLine();
    IMyList<String> sLst1 =
      dLst.Map<String>(delegate(double d) { return "s" + d; });
    foreach (String s in sLst1)
      Console.Write("{0} ", s);
    Console.WriteLine();
    IMyList<String> sLst2 = dLst.Map<String>(d => "s" + d);
    foreach (String s in sLst2)
      Console.Write("{0} ", s);
    Console.WriteLine();
    // Testing SortedList<MyString>
    SortedList<MyString> sortedLst = new SortedList<MyString>();
    sortedLst.Insert(new MyString("New York"));
    sortedLst.Insert(new MyString("Rome"));
    sortedLst.Insert(new MyString("Dublin"));
    sortedLst.Insert(new MyString("Riyadh"));
    sortedLst.Insert(new MyString("Tokyo"));
    foreach (MyString s in sortedLst)
      Console.Write("{0} ", s.Value);
    Console.WriteLine();
    // MyList equality
    Console.WriteLine(dLst.Equals(dLst));       // True
    Console.WriteLine(dLst.Equals(sLst1));      // False
    Console.WriteLine(sLst1.Equals(sLst2));     // True
    Console.WriteLine(sLst1.Equals(null));      // False
  }
}
```

```
// Example 201 from page 167 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Drawing;                           // Color

class MyTest {
  public static void Main(String[] args) {
    Point<String> p1 = new Point<String>(5, 117, "home"),
      p2 = new Point<String>(2, 3, "work");
    Point<double> p3 = new Point<double>(10, 100, 3.1415);
    ColorPoint<String,uint> p4 =
      new ColorPoint<String,uint>(20, 30, "foo", 0x0000FF);
    ColorPoint<String,Color> p5 =
      new ColorPoint<String,Color>(40, 50, "bar", Color.Blue);
    IMovable[] movables = { p1, p2, p3, p4, p5 };
    Point<String>[] stringpoints = { p1, p4, p5 };
  }
}

interface IMovable {
  void Move(int dx, int dy);
}

class Point<Label> : IMovable {
  protected internal int x, y;
  private Label lab;

  public Point(int x, int y, Label lab) {
    this.x = x; this.y = y; this.lab = lab;
  }

  public void Move(int dx, int dy) {
    x += dx; y += dy;
  }

  public Label Lab {
    get { return Lab; }
  }
}

class ColorPoint<Label, Color> : Point<Label> {
  private Color c;

  public ColorPoint(int x, int y, Label lab, Color c) : base(x, y, lab) {
    this.c = c;
  }
}
```

```
// Example 202 from page 169 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// A generic LinkedList class

using System;
using System.IO;                          // TextWriter
using System.Collections.Generic;         // IEnumerable<T>, IEnumerator<T>
using SC = System.Collections;            // IEnumerable, IEnumerator


public interface IMyList<T> : IEnumerable<T>, IEquatable<IMyList<T>> {
  int Count { get; }                      // Number of elements
  T this[int i] { get; set; }             // Get or set element at index i
  void Add(T item);                       // Add element at end
  void Insert(int i, T item);             // Insert element at index i
  void RemoveAt(int i);                   // Remove element at index i
  IMyList<U> Map<U>(Func<T,U> f);         // Map f over all elements
}

public class LinkedList<T> : IMyList<T> {
  protected int size;                 // Number of elements in the list
  protected Node first, last;         // Invariant: first==null iff last==null

  protected class Node {
    public Node prev, next;
    public T item;

    public Node(T item) {
      this.item = item;
    }

    public Node(T item, Node prev, Node next) {
      this.item = item; this.prev = prev; this.next = next;
    }
  }

  public LinkedList() {
    first = last = null;
    size = 0;
  }

  public LinkedList(params T[] arr) : this() {
    foreach (T x in arr)
      Add(x);
  }

  public int Count {
    get { return size; }
  }

  public T this[int i] {
    get { return get(i).item; }
    set { get(i).item = value; }
  }

  private Node get(int n) {
    if (n < 0 || n >= size)
      throw new IndexOutOfRangeException();
    else if (n < size/2) {                // Closer to front
      Node node = first;
      for (int i=0; i<n; i++)
        node = node.next;
      return node;
    } else {                              // Closer to end
      Node node = last;
      for (int i=size-1; i>n; i--)
        node = node.prev;
      return node;
    }
  }

  public void Add(T item) {
    Insert(size, item);
  }

  public void Insert(int i, T item) {
    if (i == 0) {
      if (first == null) // and thus last == null
```

```
      first = last = new Node(item);
    else {
      Node tmp = new Node(item, null, first);
      first.prev = tmp;
      first = tmp;
    }
    size++;
  } else if (i == size) {
    if (last == null) // and thus first = null
      first = last = new Node(item);
    else {
      Node tmp = new Node(item, last, null);
      last.next = tmp;
      last = tmp;
    }
    size++;
  } else {
    Node node = get(i);
    // assert node.prev != null;
    Node newnode = new Node(item, node.prev, node);
    node.prev.next = newnode;
    node.prev = newnode;
    size++;
  }
}

public void RemoveAt(int i) {
  Node node = get(i);
  if (node.prev == null)
    first = node.next;
  else
    node.prev.next = node.next;
  if (node.next == null)
    last = node.prev;
  else
    node.next.prev = node.prev;
  size--;
}

public override bool Equals(Object that) {
  return Equals(that as IMyList<T>);
}

public bool Equals(IMyList<T> that) {
  if (this == that)
    return true;
  if (that == null || this.Count != that.Count)
    return false;
  Node thisnode = this.first;
  IEnumerator<T> thatenm = that.GetEnumerator();
  while (thisnode != null) {
    if (!thatenm.MoveNext())
      throw new ApplicationException("Impossible: LinkedList<T>.Equals");
    // assert MoveNext() was true (because of the above size test)
    if (!thisnode.item.Equals(thatenm.Current))
      return false;
    thisnode = thisnode.next;
  }
  // assert !MoveNext(); // because of the size test
  return true;
}

public override int GetHashCode() {
  int hash = 0;
  foreach (T x in this)
    hash ^= x.GetHashCode();
  return hash;
}

public static explicit operator LinkedList<T>(T[] arr) {
  return new LinkedList<T>(arr);
}

public static LinkedList<T> operator +(LinkedList<T> xs1, LinkedList<T> xs2) {
  LinkedList<T> res = new LinkedList<T>();
  foreach (T x in xs1)
    res.Add(x);
  foreach (T x in xs2)
    res.Add(x);
  return res;
}

public IMyList<U> Map<U>(Func<T,U> f) {
  LinkedList<U> res = new LinkedList<U>();
  foreach (T x in this)
    res.Add(f(x));
  return res;
}

public IEnumerator<T> GetEnumerator() {
  return new LinkedListEnumerator(this);
}

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

private class LinkedListEnumerator : IEnumerator<T> {
  T curr;                    // The enumerator's current element
  bool valid;                // Is the current element valid?
  Node next;                 // Node holding the next element, or null

  public LinkedListEnumerator(LinkedList<T> lst) {
    next = lst.first; valid = false;
  }

  public T Current {
    get {
      if (valid)
        return curr;
      else
        throw new InvalidOperationException();
    }
  }

  public bool MoveNext() {
    if (next != null)  {
      curr = next.item; next = next.next; valid = true;
    } else
      valid = false;
    return valid;
  }

  public void Dispose() {
    curr = default(T);
    next = null; valid = false;
  }

  Object SC.IEnumerator.Current {
    get { return Current; }
  }

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

class SortedList<T> : LinkedList<T> where T : IComparable<T> {
  // Sorted insertion
  public void Insert(T x) {
    Node node = first;
    while (node != null && x.CompareTo(node.item) > 0)
      node = node.next;
    if (node == null)          // x > all elements; insert at end
      Add(x);
    else {                     // x <= node.item; insert before node
      Node newnode = new Node(x);
      if (node.prev == null)   // insert as first element
        first = newnode;
      else
        node.prev.next = newnode;
      newnode.next = node;
      newnode.prev = node.prev;
      node.prev = newnode;
    }
  }
}
```

```
interface IPrintable {
  void Print(TextWriter fs);
}
class PrintableLinkedList<T> : LinkedList<T>, IPrintable where T : IPrintable {
  public void Print(TextWriter fs) {
    bool firstElement = true;
    foreach (T x in this) {
      x.Print(fs);
      if (firstElement)
        firstElement = false;
      else
        fs.Write(",");
    }
  }
}

class MyString : IComparable<MyString> {
  private readonly String s;
  public MyString(String s) {
    this.s = s;
  }
  public int CompareTo(MyString that) {
    return String.Compare(that.Value, s);       // Reverse ordering
  }
  public bool Equals(MyString that) {
    return String.Equals(that.Value, s);
  }
  public String Value {
    get { return s; }
  }
}

class MyTest {
  public static void Main(String[] args) {
    LinkedList<double> dLst = new LinkedList<double>(7.0, 9.0, 13.0, 0.0);
    foreach (double d in dLst)
      Console.Write("{0} ", d);
    Console.WriteLine();
    IMyList<int> iLst = dLst.Map<int>(Math.Sign);
    foreach (int i in iLst)
      Console.Write("{0} ", i);
    Console.WriteLine();
    IMyList<String> sLst1 =
      dLst.Map<String>(delegate(double d) { return "s" + d; });
    foreach (String s in sLst1)
      Console.Write("{0} ", s);
    Console.WriteLine();
    IMyList<String> sLst2 = dLst.Map<String>(d => "s" + d);
    foreach (String s in sLst2)
      Console.Write("{0} ", s);
    Console.WriteLine();
    // Testing SortedList<MyString>
    SortedList<MyString> sortedLst = new SortedList<MyString>();
    sortedLst.Insert(new MyString("New York"));
    sortedLst.Insert(new MyString("Rome"));
    sortedLst.Insert(new MyString("Dublin"));
    sortedLst.Insert(new MyString("Riyadh"));
    sortedLst.Insert(new MyString("Tokyo"));
    foreach (MyString s in sortedLst)
      Console.Write("{0} ", s.Value);
    Console.WriteLine();
    // MyList equality
    Console.WriteLine(dLst.Equals(dLst));       // True
    Console.WriteLine(dLst.Equals(sLst1));      // False
    Console.WriteLine(sLst1.Equals(sLst2));     // True
    Console.WriteLine(sLst1.Equals(null));      // False
  }
}
```

```
// Example 203 from page 169 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

// A constraint may involve type parameters
// A type may have multiple constraints

struct ComparablePair<T,U> : IComparable<ComparablePair<T,U>>
  where T : IComparable<T>
  where U : IComparable<U> {
  public readonly T Fst;
  public readonly U Snd;

  public ComparablePair(T fst, U snd) {
    Fst = fst; Snd = snd;
  }

  // Lexicographic ordering
  public int CompareTo(ComparablePair<T,U> that) {
    int firstCmp = this.Fst.CompareTo(that.Fst);
    return firstCmp != 0 ? firstCmp : this.Snd.CompareTo(that.Snd);
  }

  public override String ToString() {
    return "(" + Fst + "," + Snd + ")";
  }
}

// Sorting soccer world champions by country and year

class MyTest {
  public static void Main(String[] args) {
    List<ComparablePair<String,int>> lst
      = new List<ComparablePair<String,int>>();
    lst.Add(new ComparablePair<String,int>("Brazil", 2002));
    lst.Add(new ComparablePair<String,int>("Italy", 1982));
    lst.Add(new ComparablePair<String,int>("Argentina", 1978 ));
    lst.Add(new ComparablePair<String,int>("Argentina", 1986 ));
    lst.Add(new ComparablePair<String,int>("Germany", 1990));
    lst.Add(new ComparablePair<String,int>("Brazil", 1994));
    lst.Add(new ComparablePair<String,int>("France", 1998));
    lst.Sort();
    foreach (ComparablePair<String,int> pair in lst)
      Console.WriteLine(pair);
  }
}
```

```
// Example 204 from page 169 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

// Class constraint -- permits use of null

class C1<T> where T : class {
  T f = null;                           // Legal: T is a reference type
}

// class C2<T> {
//   T f = null;                        // Illegal: T could be a value type
// }


// Struct constraint -- permits use of U in U?, a nullable type

class D1<U> where U : struct {
  U? f;                                 // Legal: U is a non-nullable value type
}

// class D2<U> {
//   U? f;                              // Illegal: U could be a reference type
// }

class MyTest {
  public static void Main(String[] args) {
  }
}
```

```
// Example 205 from page 171 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// A generic LinkedList class

using System;
using System.IO;                        // TextWriter
using System.Collections.Generic;       // IEnumerable<T>, IEnumerator<T>
using SC = System.Collections;          // IEnumerable, IEnumerator


public interface IMyList<T> : IEnumerable<T>, IEquatable<IMyList<T>> {
  int Count { get; }                    // Number of elements
  T this[int i] { get; set; }           // Get or set element at index i
  void Add(T item);                     // Add element at end
  void Insert(int i, T item);           // Insert element at index i
  void RemoveAt(int i);                 // Remove element at index i
  IMyList<U> Map<U>(Func<T,U> f);       // Map f over all elements
}

public class LinkedList<T> : IMyList<T> {
  protected int size;                   // Number of elements in the list
  protected Node first, last;           // Invariant: first==null iff last==null

  protected class Node {
    public Node prev, next;
    public T item;

    public Node(T item) {
      this.item = item;
    }

    public Node(T item, Node prev, Node next) {
      this.item = item; this.prev = prev; this.next = next;
    }
  }

  public LinkedList() {
    first = last = null;
    size = 0;
  }

  public LinkedList(params T[] arr) : this() {
    foreach (T x in arr)
      Add(x);
  }

  public int Count {
    get { return size; }
  }

  public T this[int i] {
    get { return get(i).item; }
    set { get(i).item = value; }
  }

  private Node get(int n) {
    if (n < 0 || n >= size)
      throw new IndexOutOfRangeException();
    else if (n < size/2) {              // Closer to front
      Node node = first;
      for (int i=0; i<n; i++)
        node = node.next;
      return node;
    } else {                            // Closer to end
      Node node = last;
      for (int i=size-1; i>n; i--)
        node = node.prev;
      return node;
    }
  }

  public void Add(T item) {
    Insert(size, item);
  }

  public void Insert(int i, T item) {
    if (i == 0) {
      if (first == null) // and thus last == null
```

```
      first = last = new Node(item);
    else {
      Node tmp = new Node(item, null, first);
      first.prev = tmp;
      first = tmp;
    }
    size++;
  } else if (i == size) {
    if (last == null) // and thus first = null
      first = last = new Node(item);
    else {
      Node tmp = new Node(item, last, null);
      last.next = tmp;
      last = tmp;
    }
    size++;
  } else {
    Node node = get(i);
    // assert node.prev != null;
    Node newnode = new Node(item, node.prev, node);
    node.prev.next = newnode;
    node.prev = newnode;
    size++;
  }
}

public void RemoveAt(int i) {
  Node node = get(i);
  if (node.prev == null)
    first = node.next;
  else
    node.prev.next = node.next;
  if (node.next == null)
    last = node.prev;
  else
    node.next.prev = node.prev;
  size--;
}

public override bool Equals(Object that) {
  return Equals(that as IMyList<T>);
}

public bool Equals(IMyList<T> that) {
  if (this == that)
    return true;
  if (that == null || this.Count != that.Count)
    return false;
  Node thisnode = this.first;
  IEnumerator<T> thatenm = that.GetEnumerator();
  while (thisnode != null) {
    if (!thatenm.MoveNext())
      throw new ApplicationException("Impossible: LinkedList<T>.Equals");
    // assert MoveNext() was true (because of the above size test)
    if (!thisnode.item.Equals(thatenm.Current))
      return false;
    thisnode = thisnode.next;
  }
  // assert !MoveNext(); // because of the size test
  return true;
}

public override int GetHashCode() {
  int hash = 0;
  foreach (T x in this)
    hash ^= x.GetHashCode();
  return hash;
}

public static explicit operator LinkedList<T>(T[] arr) {
  return new LinkedList<T>(arr);
}

public static LinkedList<T> operator +(LinkedList<T> xs1, LinkedList<T> xs2) {
  LinkedList<T> res = new LinkedList<T>();
  foreach (T x in xs1)
    res.Add(x);
  foreach (T x in xs2)
    res.Add(x);
  return res;
}

public IMyList<U> Map<U>(Func<T,U> f) {
  LinkedList<U> res = new LinkedList<U>();
  foreach (T x in this)
    res.Add(f(x));
  return res;
}

public IEnumerator<T> GetEnumerator() {
  return new LinkedListEnumerator(this);
}

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

private class LinkedListEnumerator : IEnumerator<T> {
  T curr;                    // The enumerator's current element
  bool valid;                // Is the current element valid?
  Node next;                 // Node holding the next element, or null

  public LinkedListEnumerator(LinkedList<T> lst) {
    next = lst.first; valid = false;
  }

  public T Current {
    get {
      if (valid)
        return curr;
      else
        throw new InvalidOperationException();
    }
  }

  public bool MoveNext() {
    if (next != null)  {
      curr = next.item; next = next.next; valid = true;
    } else
      valid = false;
    return valid;
  }

  public void Dispose() {
    curr = default(T);
    next = null; valid = false;
  }

  Object SC.IEnumerator.Current {
    get { return Current; }
  }

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

class SortedList<T> : LinkedList<T> where T : IComparable<T> {
  // Sorted insertion
  public void Insert(T x) {
    Node node = first;
    while (node != null && x.CompareTo(node.item) > 0)
      node = node.next;
    if (node == null)          // x > all elements; insert at end
      Add(x);
    else {                     // x <= node.item; insert before node
      Node newnode = new Node(x);
      if (node.prev == null)   // insert as first element
        first = newnode;
      else
        node.prev.next = newnode;
      newnode.next = node;
      newnode.prev = node.prev;
      node.prev = newnode;
    }
  }
}
```

```
interface IPrintable {
  void Print(TextWriter fs);
}
class PrintableLinkedList<T> : LinkedList<T>, IPrintable where T : IPrintable {
  public void Print(TextWriter fs) {
    bool firstElement = true;
    foreach (T x in this) {
      x.Print(fs);
      if (firstElement)
        firstElement = false;
      else
        fs.Write(",");
    }
  }
}

class MyString : IComparable<MyString> {
  private readonly String s;
  public MyString(String s) {
    this.s = s;
  }
  public int CompareTo(MyString that) {
    return String.Compare(that.Value, s);       // Reverse ordering
  }
  public bool Equals(MyString that) {
    return String.Equals(that.Value, s);
  }
  public String Value {
    get { return s; }
  }
}

class MyTest {
  public static void Main(String[] args) {
    LinkedList<double> dLst = new LinkedList<double>(7.0, 9.0, 13.0, 0.0);
    foreach (double d in dLst)
      Console.Write("{0} ", d);
    Console.WriteLine();
    IMyList<int> iLst = dLst.Map<int>(Math.Sign);
    foreach (int i in iLst)
      Console.Write("{0} ", i);
    Console.WriteLine();
    IMyList<String> sLst1 =
      dLst.Map<String>(delegate(double d) { return "s" + d; });
    foreach (String s in sLst1)
      Console.Write("{0} ", s);
    Console.WriteLine();
    IMyList<String> sLst2 = dLst.Map<String>(d => "s" + d);
    foreach (String s in sLst2)
      Console.Write("{0} ", s);
    Console.WriteLine();
    // Testing SortedList<MyString>
    SortedList<MyString> sortedLst = new SortedList<MyString>();
    sortedLst.Insert(new MyString("New York"));
    sortedLst.Insert(new MyString("Rome"));
    sortedLst.Insert(new MyString("Dublin"));
    sortedLst.Insert(new MyString("Riyadh"));
    sortedLst.Insert(new MyString("Tokyo"));
    foreach (MyString s in sortedLst)
      Console.Write("{0} ", s.Value);
    Console.WriteLine();
    // MyList equality
    Console.WriteLine(dLst.Equals(dLst));       // True
    Console.WriteLine(dLst.Equals(sLst1));      // False
    Console.WriteLine(sLst1.Equals(sLst2));     // True
    Console.WriteLine(sLst1.Equals(null));      // False
  }
}
```

```
// Example 206 from page 171 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

// A type implements AddMul<A,R> if one can add an A to it, giving an R:

interface AddMul<A,R> {
  R Add(A e);                   // Addition with A, giving R
  R Mul(A e);                   // Multiplication with A, giving R
}

// Polynomials over E, Polynomial<E>:

// The base type E of the polynomial must support addition,
// multiplication and zero (via the nullary constructor).  That's what
// the type parameter constraint on E says.

// In return, one can add an E or a polynomial over E to a polynomial
// over E.  Similarly, a polynomial over E can be multiplied by an E
// or by a polynomial over E.  That's what the interface clauses say.

class Polynomial<E> : AddMul<E,Polynomial<E>>,
                      AddMul<Polynomial<E>,Polynomial<E>>
  where E : AddMul<E,E>, new() {
  // cs contains coefficients of x^0, x^1, ...; absent coefficients are zero.
  // Invariant: cs != null && cs.Length >= 0; cs.Length==0 represents zero.
  private readonly E[] cs;

  public Polynomial() {
    this.cs = new E[0];
  }

  public Polynomial(E[] cs) {
    this.cs = cs;
  }

  public Polynomial<E> Add(Polynomial<E> that) {
    int newlen = Math.Max(this.cs.Length, that.cs.Length);
    int minlen = Math.Min(this.cs.Length, that.cs.Length);
    E[] newcs = new E[newlen];
    if (this.cs.Length <= that.cs.Length) {
      for (int i=0; i<minlen; i++)
        newcs[i] = this.cs[i].Add(that.cs[i]);
      for (int i=minlen; i<newlen; i++)
        newcs[i] = that.cs[i];
    } else {
      for (int i=0; i<minlen; i++)
        newcs[i] = this.cs[i].Add(that.cs[i]);
      for (int i=minlen; i<newlen; i++)
        newcs[i] = this.cs[i];
    }
    return new Polynomial<E>(newcs);
  }

  public Polynomial<E> Add(E that) {
    return this.Add(new Polynomial<E>(new E[] { that }));
  }

  public Polynomial<E> Mul(E that) {
    E[] newcs = new E[cs.Length];
    for (int i=0; i<cs.Length; i++)
      newcs[i] = that.Mul(cs[i]);
    return new Polynomial<E>(newcs);
  }

  public Polynomial<E> Mul(Polynomial<E> that) {
    int newlen = Math.Max(1, this.cs.Length + that.cs.Length - 1);
    E[] newcs = new E[newlen];
    for (int i=0; i<newlen; i++) {
      E sum = new E();                  // Permitted by constraint E : new()
      int start = Math.Max(0, i-that.cs.Length+1);
      int stop  = Math.Min(i, this.cs.Length-1);
      for (int j=start; j<=stop; j++) {
        // assert 0<=j && j<this.cs.Length && 0<=i-j && i-j<that.cs.Length;
        sum = sum.Add(this.cs[j].Mul(that.cs[i-j]));
      }
      newcs[i] = sum;
    }
```

```
    return new Polynomial<E>(newcs);
  }

  public E Eval(E x) {
    E res = new E();                          // Permitted by constraint E : new()
    for (int j=cs.Length-1; j>=0; j--)
      res = res.Mul(x).Add(cs[j]);
    return res;
  }
}

struct Int : AddMul<Int,Int> {
  private readonly int i;
  public Int(int i) {
    this.i = i;
  }
  public Int Add(Int that) {
    return new Int(this.i + that.i);
  }
  public Int Mul(Int that) {
    return new Int(this.i * that.i);
  }
  public override String ToString() {
    return i.ToString();
  }
}

class TestPolynomial {
  public static void Main(String[] args) {
    // The integer polynomial 2 + 5x + x^2
    Polynomial<Int> ip =
      new Polynomial<Int>(new Int[] { new Int(2), new Int(5), new Int(1) });
    Console.WriteLine(ip.Eval(new Int(10)));            // 152
    Console.WriteLine(ip.Add(ip).Eval(new Int(10)));    // 304 = 152 + 152
    Console.WriteLine(ip.Mul(ip).Eval(new Int(10)));    // 23104 = 152 * 152
  }
}
```

```
// Example 207 from page 173 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Generic quicksort in functional style (the most efficient one)

using System;
using SC = System.Collections;                  // IComparer
using System.Collections.Generic;               // IComparer<T>

class GenericFunQuicksort {
  public static void Main(String[] args) {
    int[] ia = { 5, 7, 3, 9, 12, 45, 4, 8 };
    Qsort<int>(ia, new IntComparer(), 0, ia.Length-1);
    foreach (int i in ia)
      Console.Write("{0} ", i);
    Console.WriteLine();
    String[] sa = { "New York", "Rome", "Dublin", "Riyadh", "Tokyo" };
    Qsort<String>(sa, new StringReverseComparer(), 0, sa.Length-1);
    foreach (String s in sa)
      Console.Write("{0} ", s);
    Console.WriteLine();
  }

  // Generic functional-style quicksort: sorts arr[a..b]

  private static void Qsort<T>(T[] arr, IComparer<T> cmp, int a, int b) {
    if (a < b) {
      int i = a, j = b;
      T x = arr[(i+j) / 2];
      do {
        while (cmp.Compare(arr[i], x) < 0) i++;
        while (cmp.Compare(x, arr[j]) < 0) j--;
        if (i <= j) {
          T tmp = arr[i]; arr[i] = arr[j]; arr[j] = tmp;
          i++; j--;
        }
      } while (i <= j);
      Qsort<T>(arr, cmp, a, j);
      Qsort<T>(arr, cmp, i, b);
    }
  }
}

// Comparers for int and String

public class IntComparer : SC.IComparer, IComparer<int> {
  public int Compare(Object o1, Object o2) {
    return Compare((int)o1, (int)o2);
  }
  public int Compare(int v1, int v2) {
    return v1 < v2 ? -1 : v1 > v2 ? +1 : 0;
  }
  public bool Equals(int v1, int v2) {
    return v1 == v2;
  }
  public int GetHashCode(int v) {
    return v;
  }
}

public class StringReverseComparer : IComparer<string> {
  public int Compare(String v1, String v2) {
    return String.Compare(v2, v1);
  }
  public bool Equals(String v1, String v2) {
    return String.Equals(v2, v1);
  }
  public int GetHashCode(String v) {
    return v.GetHashCode();
  }
}
```

```
// Example 208 from page 173 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Generic quicksort in object-oriented style

using System;
using System.Collections.Generic;               // IComparable<T>

class GenericObjQuicksort {
  public static void Main(String[] args) {
    MyString[] sa = { new MyString("New York"), new MyString("Rome"),
                      new MyString("Dublin"), new MyString("Riyadh"),
                      new MyString("Tokyo") };
    Qsort<MyString>(sa, 0, sa.Length-1);
    foreach (MyString s in sa)
      Console.Write("{0} ", s.Value);
    Console.WriteLine();
  }

  // Generic object-oriented style quicksort: sorts arr[a..b]

  private static void Qsort<T>(T[] arr, int a, int b)
    where T : IComparable<T> {
    if (a < b) {
      int i = a, j = b;
      T x = arr[(i+j) / 2];
      do {
        while (arr[i].CompareTo(x) < 0) i++;
        while (x.CompareTo(arr[j]) < 0) j--;
        if (i <= j) {
          T tmp = arr[i]; arr[i] = arr[j]; arr[j] = tmp;
          i++; j--;
        }
      } while (i <= j);
      Qsort<T>(arr, a, j);
      Qsort<T>(arr, i, b);
    }
  }
}

class MyString : IComparable<MyString> {
  private readonly String s;
  public MyString(String s) {
    this.s = s;
  }
  public int CompareTo(MyString that) {
    return String.Compare(that.Value, s);       // Reverse ordering
  }
  public String Value {
    get { return s; }
  }
}
```

```
// Example 209 from page 173 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// A generic LinkedList class

using System;
using System.IO;                            // TextWriter
using System.Collections.Generic;           // IEnumerable<T>, IEnumerator<T>
using SC = System.Collections;              // IEnumerable, IEnumerator


public interface IMyList<T> : IEnumerable<T>, IEquatable<IMyList<T>> {
  int Count { get; }                        // Number of elements
  T this[int i] { get; set; }               // Get or set element at index i
  void Add(T item);                         // Add element at end
  void Insert(int i, T item);               // Insert element at index i
  void RemoveAt(int i);                     // Remove element at index i
  IMyList<U> Map<U>(Func<T,U> f);           // Map f over all elements
}

public class LinkedList<T> : IMyList<T> {
  protected int size;                   // Number of elements in the list
  protected Node first, last;           // Invariant: first==null iff last==null

  protected class Node {
    public Node prev, next;
    public T item;

    public Node(T item) {
      this.item = item;
    }

    public Node(T item, Node prev, Node next) {
      this.item = item; this.prev = prev; this.next = next;
    }
  }

  public LinkedList() {
    first = last = null;
    size = 0;
  }

  public LinkedList(params T[] arr) : this() {
    foreach (T x in arr)
      Add(x);
  }

  public int Count {
    get { return size; }
  }

  public T this[int i] {
    get { return get(i).item; }
    set { get(i).item = value; }
  }

  private Node get(int n) {
    if (n < 0 || n >= size)
      throw new IndexOutOfRangeException();
    else if (n < size/2) {              // Closer to front
      Node node = first;
      for (int i=0; i<n; i++)
        node = node.next;
      return node;
    } else {                            // Closer to end
      Node node = last;
      for (int i=size-1; i>n; i--)
        node = node.prev;
      return node;
    }
  }

  public void Add(T item) {
    Insert(size, item);
  }

  public void Insert(int i, T item) {
    if (i == 0) {
      if (first == null) // and thus last == null
```

```
      first = last = new Node(item);
    else {
      Node tmp = new Node(item, null, first);
      first.prev = tmp;
      first = tmp;
    }
    size++;
  } else if (i == size) {
    if (last == null) // and thus first = null
      first = last = new Node(item);
    else {
      Node tmp = new Node(item, last, null);
      last.next = tmp;
      last = tmp;
    }
    size++;
  } else {
    Node node = get(i);
    // assert node.prev != null;
    Node newnode = new Node(item, node.prev, node);
    node.prev.next = newnode;
    node.prev = newnode;
    size++;
  }
}

public void RemoveAt(int i) {
  Node node = get(i);
  if (node.prev == null)
    first = node.next;
  else
    node.prev.next = node.next;
  if (node.next == null)
    last = node.prev;
  else
    node.next.prev = node.prev;
  size--;
}

public override bool Equals(Object that) {
  return Equals(that as IMyList<T>);
}

public bool Equals(IMyList<T> that) {
  if (this == that)
    return true;
  if (that == null || this.Count != that.Count)
    return false;
  Node thisnode = this.first;
  IEnumerator<T> thatenm = that.GetEnumerator();
  while (thisnode != null) {
    if (!thatenm.MoveNext())
      throw new ApplicationException("Impossible: LinkedList<T>.Equals");
    // assert MoveNext() was true (because of the above size test)
    if (!thisnode.item.Equals(thatenm.Current))
      return false;
    thisnode = thisnode.next;
  }
  // assert !MoveNext(); // because of the size test
  return true;
}

public override int GetHashCode() {
  int hash = 0;
  foreach (T x in this)
    hash ^= x.GetHashCode();
  return hash;
}

public static explicit operator LinkedList<T>(T[] arr) {
  return new LinkedList<T>(arr);
}

public static LinkedList<T> operator +(LinkedList<T> xs1, LinkedList<T> xs2) {
  LinkedList<T> res = new LinkedList<T>();
  foreach (T x in xs1)
    res.Add(x);
  foreach (T x in xs2)
    res.Add(x);
  return res;
}

public IMyList<U> Map<U>(Func<T,U> f) {
  LinkedList<U> res = new LinkedList<U>();
  foreach (T x in this)
    res.Add(f(x));
  return res;
}

public IEnumerator<T> GetEnumerator() {
  return new LinkedListEnumerator(this);
}

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

private class LinkedListEnumerator : IEnumerator<T> {
  T curr;                    // The enumerator's current element
  bool valid;                // Is the current element valid?
  Node next;                 // Node holding the next element, or null

  public LinkedListEnumerator(LinkedList<T> lst) {
    next = lst.first; valid = false;
  }

  public T Current {
    get {
      if (valid)
        return curr;
      else
        throw new InvalidOperationException();
    }
  }

  public bool MoveNext() {
    if (next != null)  {
      curr = next.item; next = next.next; valid = true;
    } else
      valid = false;
    return valid;
  }

  public void Dispose() {
    curr = default(T);
    next = null; valid = false;
  }

  Object SC.IEnumerator.Current {
    get { return Current; }
  }

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

class SortedList<T> : LinkedList<T> where T : IComparable<T> {
  // Sorted insertion
  public void Insert(T x) {
    Node node = first;
    while (node != null && x.CompareTo(node.item) > 0)
      node = node.next;
    if (node == null)          // x > all elements; insert at end
      Add(x);
    else {                     // x <= node.item; insert before node
      Node newnode = new Node(x);
      if (node.prev == null)   // insert as first element
        first = newnode;
      else
        node.prev.next = newnode;
      newnode.next = node;
      newnode.prev = node.prev;
      node.prev = newnode;
    }
  }
}
```

```
interface IPrintable {
  void Print(TextWriter fs);
}
class PrintableLinkedList<T> : LinkedList<T>, IPrintable where T : IPrintable {
  public void Print(TextWriter fs) {
    bool firstElement = true;
    foreach (T x in this) {
      x.Print(fs);
      if (firstElement)
        firstElement = false;
      else
        fs.Write(",");
    }
  }
}

class MyString : IComparable<MyString> {
  private readonly String s;
  public MyString(String s) {
    this.s = s;
  }
  public int CompareTo(MyString that) {
    return String.Compare(that.Value, s);       // Reverse ordering
  }
  public bool Equals(MyString that) {
    return String.Equals(that.Value, s);
  }
  public String Value {
    get { return s; }
  }
}

class MyTest {
  public static void Main(String[] args) {
    LinkedList<double> dLst = new LinkedList<double>(7.0, 9.0, 13.0, 0.0);
    foreach (double d in dLst)
      Console.Write("{0} ", d);
    Console.WriteLine();
    IMyList<int> iLst = dLst.Map<int>(Math.Sign);
    foreach (int i in iLst)
      Console.Write("{0} ", i);
    Console.WriteLine();
    IMyList<String> sLst1 =
      dLst.Map<String>(delegate(double d) { return "s" + d; });
    foreach (String s in sLst1)
      Console.Write("{0} ", s);
    Console.WriteLine();
    IMyList<String> sLst2 = dLst.Map<String>(d => "s" + d);
    foreach (String s in sLst2)
      Console.Write("{0} ", s);
    Console.WriteLine();
    // Testing SortedList<MyString>
    SortedList<MyString> sortedLst = new SortedList<MyString>();
    sortedLst.Insert(new MyString("New York"));
    sortedLst.Insert(new MyString("Rome"));
    sortedLst.Insert(new MyString("Dublin"));
    sortedLst.Insert(new MyString("Riyadh"));
    sortedLst.Insert(new MyString("Tokyo"));
    foreach (MyString s in sortedLst)
      Console.Write("{0} ", s.Value);
    Console.WriteLine();
    // MyList equality
    Console.WriteLine(dLst.Equals(dLst));       // True
    Console.WriteLine(dLst.Equals(sLst1));      // False
    Console.WriteLine(sLst1.Equals(sLst2));     // True
    Console.WriteLine(sLst1.Equals(null));      // False
  }
}
```

```
// Example 210 from page 175 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Quicksort using a generic delegate to compare elements

using System;

// The DComparer delegate type

public delegate int DComparer<T>(T v1, T v2);

class DelegateQuicksort {
  public static void Main(String[] args) {
    int[] ia = { 5, 7, 3, 9, 12, 45, 4, 8 };
    DComparer<int> intCmp = IntCompare;
    Qsort<int>(ia, intCmp, 0, ia.Length-1);
    foreach (int i in ia)
      Console.Write("{0} ", i);
    Console.WriteLine();
    String[] sa = { "New York", "Rome", "Dublin", "Riyadh", "Tokyo" };
    DComparer<String> strCmp = String.Compare;
    Qsort<String>(sa, strCmp, 0, sa.Length-1);
    foreach (String s in sa)
      Console.Write("{0} ", s);
    Console.WriteLine();
  }

  // Quicksort: sorts arr[a..b] using delegate cmp to compare elements

  private static void Qsort<T>(T[] arr, DComparer<T> cmp, int a, int b) {
    if (a < b) {
      int i = a, j = b;
      T x = arr[(i+j) / 2];
      do {
        while (cmp(arr[i], x) < 0) i++;         // Call delegate cmp
        while (cmp(x, arr[j]) < 0) j--;         // Call delegate cmp
        if (i <= j) {
          T tmp = arr[i]; arr[i] = arr[j]; arr[j] = tmp;
          i++; j--;
        }
      } while (i <= j);
      Qsort<T>(arr, cmp, a, j);
      Qsort<T>(arr, cmp, i, b);
    }
  }

  // Type-safe comparison method for int

  static int IntCompare(int i1, int i2) {
    return i1 < i2 ? -1 : i1 > i2 ? +1 : 0;
  }
}
```

```
// Example 211 from page 175 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Quicksort using a generic delegate to compare elements

using System;

// The DComparer delegate type

public delegate int DComparer<T>(T v1, T v2);

class DelegateQuicksort {
  public static void Main(String[] args) {
    int[] ia = { 5, 7, 3, 9, 12, 45, 4, 8 };
    DComparer<int> intCmp = IntCompare;
    Qsort<int>(ia, intCmp, 0, ia.Length-1);
    foreach (int i in ia)
      Console.Write("{0} ", i);
    Console.WriteLine();
    String[] sa = { "New York", "Rome", "Dublin", "Riyadh", "Tokyo" };
    DComparer<String> strCmp = String.Compare;
    Qsort<String>(sa, strCmp, 0, sa.Length-1);
    foreach (String s in sa)
      Console.Write("{0} ", s);
    Console.WriteLine();
  }

  // Quicksort: sorts arr[a..b] using delegate cmp to compare elements

  private static void Qsort<T>(T[] arr, DComparer<T> cmp, int a, int b) {
    if (a < b) {
      int i = a, j = b;
      T x = arr[(i+j) / 2];
      do {
        while (cmp(arr[i], x) < 0) i++;         // Call delegate cmp
        while (cmp(x, arr[j]) < 0) j--;         // Call delegate cmp
        if (i <= j) {
          T tmp = arr[i]; arr[i] = arr[j]; arr[j] = tmp;
          i++; j--;
        }
      } while (i <= j);
      Qsort<T>(arr, cmp, a, j);
      Qsort<T>(arr, cmp, i, b);
    }
  }

  // Type-safe comparison method for int

  static int IntCompare(int i1, int i2) {
    return i1 < i2 ? -1 : i1 > i2 ? +1 : 0;
  }
}
```

```
// Example 213 from page 175 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// A generic LinkedList class

using System;
using System.IO;                          // TextWriter
using System.Collections.Generic;         // IEnumerable<T>, IEnumerator<T>
using SC = System.Collections;            // IEnumerable, IEnumerator


public interface IMyList<T> : IEnumerable<T>, IEquatable<IMyList<T>> {
  int Count { get; }                      // Number of elements
  T this[int i] { get; set; }             // Get or set element at index i
  void Add(T item);                       // Add element at end
  void Insert(int i, T item);             // Insert element at index i
  void RemoveAt(int i);                   // Remove element at index i
  IMyList<U> Map<U>(Func<T,U> f);         // Map f over all elements
}

public class LinkedList<T> : IMyList<T> {
  protected int size;                 // Number of elements in the list
  protected Node first, last;         // Invariant: first==null iff last==null

  protected class Node {
    public Node prev, next;
    public T item;

    public Node(T item) {
      this.item = item;
    }

    public Node(T item, Node prev, Node next) {
      this.item = item; this.prev = prev; this.next = next;
    }
  }

  public LinkedList() {
    first = last = null;
    size = 0;
  }

  public LinkedList(params T[] arr) : this() {
    foreach (T x in arr)
      Add(x);
  }

  public int Count {
    get { return size; }
  }

  public T this[int i] {
    get { return get(i).item; }
    set { get(i).item = value; }
  }

  private Node get(int n) {
    if (n < 0 || n >= size)
      throw new IndexOutOfRangeException();
    else if (n < size/2) {                // Closer to front
      Node node = first;
      for (int i=0; i<n; i++)
        node = node.next;
      return node;
    } else {                              // Closer to end
      Node node = last;
      for (int i=size-1; i>n; i--)
        node = node.prev;
      return node;
    }
  }

  public void Add(T item) {
    Insert(size, item);
  }

  public void Insert(int i, T item) {
    if (i == 0) {
      if (first == null) // and thus last == null
```

```
      first = last = new Node(item);
    else {
      Node tmp = new Node(item, null, first);
      first.prev = tmp;
      first = tmp;
    }
    size++;
  } else if (i == size) {
    if (last == null) // and thus first = null
      first = last = new Node(item);
    else {
      Node tmp = new Node(item, last, null);
      last.next = tmp;
      last = tmp;
    }
    size++;
  } else {
    Node node = get(i);
    // assert node.prev != null;
    Node newnode = new Node(item, node.prev, node);
    node.prev.next = newnode;
    node.prev = newnode;
    size++;
  }
}

public void RemoveAt(int i) {
  Node node = get(i);
  if (node.prev == null)
    first = node.next;
  else
    node.prev.next = node.next;
  if (node.next == null)
    last = node.prev;
  else
    node.next.prev = node.prev;
  size--;
}

public override bool Equals(Object that) {
  return Equals(that as IMyList<T>);
}

public bool Equals(IMyList<T> that) {
  if (this == that)
    return true;
  if (that == null || this.Count != that.Count)
    return false;
  Node thisnode = this.first;
  IEnumerator<T> thatenm = that.GetEnumerator();
  while (thisnode != null) {
    if (!thatenm.MoveNext())
      throw new ApplicationException("Impossible: LinkedList<T>.Equals");
    // assert MoveNext() was true (because of the above size test)
    if (!thisnode.item.Equals(thatenm.Current))
      return false;
    thisnode = thisnode.next;
  }
  // assert !MoveNext(); // because of the size test
  return true;
}

public override int GetHashCode() {
  int hash = 0;
  foreach (T x in this)
    hash ^= x.GetHashCode();
  return hash;
}

public static explicit operator LinkedList<T>(T[] arr) {
  return new LinkedList<T>(arr);
}

public static LinkedList<T> operator +(LinkedList<T> xs1, LinkedList<T> xs2) {
  LinkedList<T> res = new LinkedList<T>();
  foreach (T x in xs1)
    res.Add(x);
  foreach (T x in xs2)
    res.Add(x);
  return res;
}

public IMyList<U> Map<U>(Func<T,U> f) {
  LinkedList<U> res = new LinkedList<U>();
  foreach (T x in this)
    res.Add(f(x));
  return res;
}

public IEnumerator<T> GetEnumerator() {
  return new LinkedListEnumerator(this);
}

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

private class LinkedListEnumerator : IEnumerator<T> {
  T curr;                    // The enumerator's current element
  bool valid;                // Is the current element valid?
  Node next;                 // Node holding the next element, or null

  public LinkedListEnumerator(LinkedList<T> lst) {
    next = lst.first; valid = false;
  }

  public T Current {
    get {
      if (valid)
        return curr;
      else
        throw new InvalidOperationException();
    }
  }

  public bool MoveNext() {
    if (next != null)  {
      curr = next.item; next = next.next; valid = true;
    } else
      valid = false;
    return valid;
  }

  public void Dispose() {
    curr = default(T);
    next = null; valid = false;
  }

  Object SC.IEnumerator.Current {
    get { return Current; }
  }

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

class SortedList<T> : LinkedList<T> where T : IComparable<T> {
  // Sorted insertion
  public void Insert(T x) {
    Node node = first;
    while (node != null && x.CompareTo(node.item) > 0)
      node = node.next;
    if (node == null)          // x > all elements; insert at end
      Add(x);
    else {                     // x <= node.item; insert before node
      Node newnode = new Node(x);
      if (node.prev == null)   // insert as first element
        first = newnode;
      else
        node.prev.next = newnode;
      newnode.next = node;
      newnode.prev = node.prev;
      node.prev = newnode;
    }
  }
}
```

```
interface IPrintable {
  void Print(TextWriter fs);
}
class PrintableLinkedList<T> : LinkedList<T>, IPrintable where T : IPrintable {
  public void Print(TextWriter fs) {
    bool firstElement = true;
    foreach (T x in this) {
      x.Print(fs);
      if (firstElement)
        firstElement = false;
      else
        fs.Write(",");
    }
  }
}

class MyString : IComparable<MyString> {
  private readonly String s;
  public MyString(String s) {
    this.s = s;
  }
  public int CompareTo(MyString that) {
    return String.Compare(that.Value, s);       // Reverse ordering
  }
  public bool Equals(MyString that) {
    return String.Equals(that.Value, s);
  }
  public String Value {
    get { return s; }
  }
}

class MyTest {
  public static void Main(String[] args) {
    LinkedList<double> dLst = new LinkedList<double>(7.0, 9.0, 13.0, 0.0);
    foreach (double d in dLst)
      Console.Write("{0} ", d);
    Console.WriteLine();
    IMyList<int> iLst = dLst.Map<int>(Math.Sign);
    foreach (int i in iLst)
      Console.Write("{0} ", i);
    Console.WriteLine();
    IMyList<String> sLst1 =
      dLst.Map<String>(delegate(double d) { return "s" + d; });
    foreach (String s in sLst1)
      Console.Write("{0} ", s);
    Console.WriteLine();
    IMyList<String> sLst2 = dLst.Map<String>(d => "s" + d);
    foreach (String s in sLst2)
      Console.Write("{0} ", s);
    Console.WriteLine();
    // Testing SortedList<MyString>
    SortedList<MyString> sortedLst = new SortedList<MyString>();
    sortedLst.Insert(new MyString("New York"));
    sortedLst.Insert(new MyString("Rome"));
    sortedLst.Insert(new MyString("Dublin"));
    sortedLst.Insert(new MyString("Riyadh"));
    sortedLst.Insert(new MyString("Tokyo"));
    foreach (MyString s in sortedLst)
      Console.Write("{0} ", s.Value);
    Console.WriteLine();
    // MyList equality
    Console.WriteLine(dLst.Equals(dLst));       // True
    Console.WriteLine(dLst.Equals(sLst1));      // False
    Console.WriteLine(sLst1.Equals(sLst2));     // True
    Console.WriteLine(sLst1.Equals(null));      // False
  }
}
```

```
// Example 214 from page 177 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class Log<T> {
  private const int SIZE = 5;
  public static int InstanceCount { get; private set; }
  public int Count { get; private set; }
  private T[] log = new T[SIZE];
  public Log() { InstanceCount++; }
  public void Add(T msg) { log[Count++ % SIZE] = msg; }
  public T Last {
    get { // Return the last log entry, or null if nothing logged yet
      return Count==0 ? default(T) : log[(Count-1)%SIZE];
    }
    set { // Update the last log entry, or create one if nothing logged yet
      if (Count==0)
        log[Count++] = value;
      else
        log[(Count-1)%SIZE] = value;
    }
  }
  public T[] All {
    get {
      int size = Math.Min(Count, SIZE);
      T[] res = new T[size];
      for (int i=0; i<size; i++)
        res[i] = log[(Count-size+i) % SIZE];
      return res;
    }
  }
}

class TestLog {
  class MyTest {
    public static void Main(String[] args) {
      Log<String> log1 = new Log<String>();
      log1.Add("Reboot");
      log1.Add("Coffee");
      Log<DateTime> log2 = new Log<DateTime>();
      log2.Add(DateTime.Now);
      log2.Add(DateTime.Now.AddHours(1));
      DateTime[] dts = log2.All;
      // Printing both logs:
      foreach (String s in log1.All)
        Console.Write("{0} ", s);
      Console.WriteLine();
      foreach (DateTime dt in dts)
        Console.Write("{0} ", dt);
      Console.WriteLine();
      TestPairLog();
    }

    public static void TestPairLog() {
      Log<Pair<DateTime,String>> log = new Log<Pair<DateTime,String>>();
      log.Add(new Pair<DateTime,String>(DateTime.Now, "Tea leaves"));
      log.Add(new Pair<DateTime,String>(DateTime.Now.AddMinutes(2), "Hot water"));
      log.Add(new Pair<DateTime,String>(DateTime.Now.AddMinutes(7), "Ready"));
      Pair<DateTime,String>[] allMsgs = log.All;
      foreach (Pair<DateTime,String> p in allMsgs)
        Console.WriteLine("At {0}: {1}", p.Fst, p.Snd);
    }
  }
}

public struct Pair<T,U> {
  public readonly T Fst;
  public readonly U Snd;
  public Pair(T fst, U snd) {
    this.Fst = fst;
    this.Snd = snd;
  }
}
```

```
// Example 215 from page 177 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

public class Log<T> {
  private const int SIZE = 5;
  public static int InstanceCount { get; private set; }
  public int Count { get; private set; }
  private T[] log = new T[SIZE];
  public Log() { InstanceCount++; }
  public void Add(T msg) { log[Count++ % SIZE] = msg; }
  public T Last {
    get { // Return the last log entry, or null if nothing logged yet
      return Count==0 ? default(T) : log[(Count-1)%SIZE];
    }
    set { // Update the last log entry, or create one if nothing logged yet
      if (Count==0)
        log[Count++] = value;
      else
        log[(Count-1)%SIZE] = value;
    }
  }
  public T[] All {
    get {
      int size = Math.Min(Count, SIZE);
      T[] res = new T[size];
      for (int i=0; i<size; i++)
        res[i] = log[(Count-size+i) % SIZE];
      return res;
    }
  }
}

class TestLog {
  class MyTest {
    public static void Main(String[] args) {
      Log<String> log1 = new Log<String>();
      log1.Add("Reboot");
      log1.Add("Coffee");
      Log<DateTime> log2 = new Log<DateTime>();
      log2.Add(DateTime.Now);
      log2.Add(DateTime.Now.AddHours(1));
      DateTime[] dts = log2.All;
      // Printing both logs:
      foreach (String s in log1.All)
        Console.Write("{0} ", s);
      Console.WriteLine();
      foreach (DateTime dt in dts)
        Console.Write("{0} ", dt);
      Console.WriteLine();
      TestPairLog();
    }

    public static void TestPairLog() {
      Log<Pair<DateTime,String>> log = new Log<Pair<DateTime,String>>();
      log.Add(new Pair<DateTime,String>(DateTime.Now, "Tea leaves"));
      log.Add(new Pair<DateTime,String>(DateTime.Now.AddMinutes(2), "Hot water"));
      log.Add(new Pair<DateTime,String>(DateTime.Now.AddMinutes(7), "Ready"));
      Pair<DateTime,String>[] allMsgs = log.All;
      foreach (Pair<DateTime,String> p in allMsgs)
        Console.WriteLine("At {0}: {1}", p.Fst, p.Snd);
    }
  }
}

public struct Pair<T,U> {
  public readonly T Fst;
  public readonly U Snd;
  public Pair(T fst, U snd) {
    this.Fst = fst;
    this.Snd = snd;
  }
}
```

```
// Example 216 from page 177 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

// Fun<R,A> is a function with result type R and argument type A

public delegate R Fun<R,A>(A x);

// A struct of type Option<T> represents a value of value type T that
// may be absent.

public struct Option<T> {
  public readonly bool HasValue;
  private readonly T value;
  // Rejected by compiler, but why?
  // public static readonly Option<T> None = new Option<T>();

  public Option(T value) { this.HasValue = true; this.value = value; }

  public T Value {
    get {
      if (HasValue) return value;
      else throw new InvalidOperationException("No value");
    }
  }

  public static implicit operator Option<T>(T value) {
    return new Option<T>(value);
  }

  public static explicit operator T(Option<T> option) {
    return option.Value;
  }

  public Option<U> Apply<U>(Fun<U,T> fun) {
    if (HasValue)
      return new Option<U>(fun(value));
    else
      return new Option<U>();
  }

  public override String ToString() {
    return HasValue ? value.ToString() : "[No value]";
  }
}

class MyTest {
  public static Option<double> Sqrt(double x) {
    return x >= 0.0 ? new Option<double>(Math.Sqrt(x)) : new Option<double>();
  }

  public static void Main(String[] args) {
    Option<double> res1 = Sqrt(5.0);
    Option<double> res2 = Sqrt(-5.0);
    Console.WriteLine("res1={0} and res2={1}", res1, res2);
    double res3 = (double)res1;         // Explicit Option<double> --> double
    res2 = 17.0;                        // Implicit double --> Option<double>
    Console.WriteLine("res3={0} and res2={1}", res3, res2);
    res1 = res1.Apply(new Fun<double,double>(Math.Log));
    Console.WriteLine("res1={0}", res1);
  }
}
```

```
// Example 217 from page 179 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;       // IEnumerable<T>, IComparer<T>
using System.Drawing;                   // Color

// delegate R Foo<in R, out A>(A x);    // Invalid: R is an output type, A is an inpu
t type

delegate R Foo<out R, in A>(A x);  // OK

class MyTest {
  public static void Main(String[] args) {
    // Interface examples:
    Student[] students = { new Student("Anders"), new Student("Kasper"), new Student("V
incens") };
    IEnumerable<Student> ss = students;
    PrintPersons(ss);
    PrintPersons(students);
    bool sorted = Sorted(new PersonComparer(), students);
    // Delegate type examples:
    Func<Person,ColoredPoint> pc = (Person p) => new ColoredPoint(2, p.name.Length, C
olor.Red);
    Func<Student,Point> sp = pc;
    Func<Func<Student,Point>,int> fspi
      = (Func<Student,Point> fsp) => fsp(new Student("Lise")).y;
    Func<Func<Person,ColoredPoint>,int> fpci = fspi;
    Console.WriteLine(sp(new Student("Morten")));
    Console.WriteLine(fpci(pc));
  }

  static void PrintPersons(IEnumerable<Person> ps) {
    foreach (Person p in ps)
      Console.WriteLine(p.name);
  }

  static bool Sorted(IComparer<Student> cmp, Student[] a) {
    for (int i=1; i<a.Length; i++)
      if (cmp.Compare(a[i-1],a[i]) > 0)
        return false;
    return true;
  }
}

class Person {
  public readonly String name;
  public Person(String name) {
    this.name = name;
  }
}

class Student : Person {
  public Student(String name) : base(name) { }
}

class PersonComparer : IComparer<Person> {
  public int Compare(Person p1, Person p2) {
    return p1.name.CompareTo(p2.name);
  }
}

public class Point {
  protected internal int x, y;
  public Point(int x, int y) { this.x = x; this.y = y; }
  public void Move(int dx, int dy) { x += dx; y += dy; }
  public override String ToString() { return "(" + x + "," + y + ")"; }
}

class ColoredPoint : Point {
  protected Color c;
  public ColoredPoint(int x, int y, Color c) : base(x, y) { this.c = c; }
  public Color GetColor { get { return c; } }
}

namespace VarianceDemo {
  interface IEnumerator<out T> { T Current { get; } }
  interface IEnumerable<out T> { IEnumerator<T> GetEnumerator(); }
  interface IComparer<in T> { int Compare(T v1, T v2); }
  interface IComparable<in T> { int CompareTo(T v); }
  interface IEqualityComparer<in T> { bool Equals(T v1, T v2); int GetHashCode(T v); }
  interface IEquatable<in T> { bool Equals(T v); }
  delegate R Func<out R>();
  delegate R Func<in A1,out R>(A1 x1);
  delegate R Func<in A1,in A2,out R>(A1 x1, A2 x2);
  delegate void Action<out R>();
  delegate void Action<in A1,out R>(A1 x1);
  delegate void Action<in A1,in A2,out R>(A1 x1, A2 x2);
}
```

```
// Example 218 from page 179 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;       // IEnumerable<T>, IComparer<T>
using System.Drawing;                   // Color

// delegate R Foo<in R, out A>(A x);    // Invalid: R is an output type, A is an inpu
t type

delegate R Foo<out R, in A>(A x);  // OK

class MyTest {
  public static void Main(String[] args) {
    // Interface examples:
    Student[] students = { new Student("Anders"), new Student("Kasper"), new Student("V
incens") };
    IEnumerable<Student> ss = students;
    PrintPersons(ss);
    PrintPersons(students);
    bool sorted = Sorted(new PersonComparer(), students);
    // Delegate type examples:
    Func<Person,ColoredPoint> pc = (Person p) => new ColoredPoint(2, p.name.Length, C
olor.Red);
    Func<Student,Point> sp = pc;
    Func<Func<Student,Point>,int> fspi
      = (Func<Student,Point> fsp) => fsp(new Student("Lise")).y;
    Func<Func<Person,ColoredPoint>,int> fpci = fspi;
    Console.WriteLine(sp(new Student("Morten")));
    Console.WriteLine(fpci(pc));
  }

  static void PrintPersons(IEnumerable<Person> ps) {
    foreach (Person p in ps)
      Console.WriteLine(p.name);
  }

  static bool Sorted(IComparer<Student> cmp, Student[] a) {
    for (int i=1; i<a.Length; i++)
      if (cmp.Compare(a[i-1],a[i]) > 0)
        return false;
    return true;
  }
}

class Person {
  public readonly String name;
  public Person(String name) {
    this.name = name;
  }
}

class Student : Person {
  public Student(String name) : base(name) { }
}

class PersonComparer : IComparer<Person> {
  public int Compare(Person p1, Person p2) {
    return p1.name.CompareTo(p2.name);
  }
}

public class Point {
  protected internal int x, y;
  public Point(int x, int y) { this.x = x; this.y = y; }
  public void Move(int dx, int dy) { x += dx; y += dy; }
  public override String ToString() { return "(" + x + "," + y + ")"; }
}

class ColoredPoint : Point {
  protected Color c;
  public ColoredPoint(int x, int y, Color c) : base(x, y) { this.c = c; }
  public Color GetColor { get { return c; } }
}

namespace VarianceDemo {
  interface IEnumerator<out T> { T Current { get; } }
  interface IEnumerable<out T> { IEnumerator<T> GetEnumerator(); }
  interface IComparer<in T> { int Compare(T v1, T v2); }
  interface IComparable<in T> { int CompareTo(T v); }
  interface IEqualityComparer<in T> { bool Equals(T v1, T v2); int GetHashCode(T v); }
  interface IEquatable<in T> { bool Equals(T v); }
  delegate R Func<out R>();
  delegate R Func<in A1,out R>(A1 x1);
  delegate R Func<in A1,in A2,out R>(A1 x1, A2 x2);
  delegate void Action<out R>();
  delegate void Action<in A1,out R>(A1 x1);
  delegate void Action<in A1,in A2,out R>(A1 x1, A2 x2);
}
```

```
// Example 219 from page 181 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;                       // Console
using System.Collections.Generic;  // IList, IDictionary, List, Dictionary, ...

class TestCollections {
  public static void Main(String[] args) {
    IList<bool> list1 = new List<bool>();
    list1.Add(true); list1.Add(false); list1.Add(true); list1.Add(false);
    Print(list1);                   // Must print: true false true false
    bool b1 = list1[3];             // false
    Console.WriteLine(b1);
    IDictionary<String, int> dict1 = new Dictionary<String, int>();
    dict1.Add("Sweden", 46); dict1.Add("Germany", 49); dict1.Add("Japan", 81);
    Print(dict1.Keys);              // May print:  Japan Sweden Germany
    Print(dict1.Values);            // May print:  81 46 49
    int i1 = dict1["Japan"];       // 81
    Console.WriteLine(i1);
    Print(dict1);                   // Print key/value pairs in some order
    IDictionary<String, int> dict2 = new SortedDictionary<String, int>();
    dict2.Add("Sweden", 46); dict2.Add("Germany", 49); dict2.Add("Japan", 81);
    Print(dict2.Keys);              // Must print: Germany Japan Sweden
    Print(dict2.Values);            // Must print: 49 81 46
    Print(dict2);                   // Print key/value pairs in sorted key order
  }

  public static void Print<T>(ICollection<T> coll) {
    foreach (T x in coll)
      Console.Write("{0} ", x);
    Console.WriteLine();
  }

  public static void Print<K,V>(IDictionary<K,V> dict) {
    foreach (KeyValuePair<K,V> entry in dict)
      Console.WriteLine("{0} --> {1}", entry.Key, entry.Value);
    Console.WriteLine();
  }
}
```

```
// Example 220 from page 183 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;                       // Console
using System.Collections.Generic;  // IList, IDictionary, List, Dictionary, ...

class TestCollections {
  public static void Main(String[] args) {
    IList<bool> list1 = new List<bool>();
    list1.Add(true); list1.Add(false); list1.Add(true); list1.Add(false);
    Print(list1);                   // Must print: true false true false
    bool b1 = list1[3];             // false
    Console.WriteLine(b1);
    IDictionary<String, int> dict1 = new Dictionary<String, int>();
    dict1.Add("Sweden", 46); dict1.Add("Germany", 49); dict1.Add("Japan", 81);
    Print(dict1.Keys);              // May print:  Japan Sweden Germany
    Print(dict1.Values);            // May print:  81 46 49
    int i1 = dict1["Japan"];       // 81
    Console.WriteLine(i1);
    Print(dict1);                   // Print key/value pairs in some order
    IDictionary<String, int> dict2 = new SortedDictionary<String, int>();
    dict2.Add("Sweden", 46); dict2.Add("Germany", 49); dict2.Add("Japan", 81);
    Print(dict2.Keys);              // Must print: Germany Japan Sweden
    Print(dict2.Values);            // Must print: 49 81 46
    Print(dict2);                   // Print key/value pairs in sorted key order
  }

  public static void Print<T>(ICollection<T> coll) {
    foreach (T x in coll)
      Console.Write("{0} ", x);
    Console.WriteLine();
  }

  public static void Print<K,V>(IDictionary<K,V> dict) {
    foreach (KeyValuePair<K,V> entry in dict)
      Console.WriteLine("{0} --> {1}", entry.Key, entry.Value);
    Console.WriteLine();
  }
}
```

```
// Example 221 from page 183 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;                       // Console
using System.Collections.Generic;  // IList, IDictionary, List, Dictionary, ...

class TestCollections {
  public static void Main(String[] args) {
    IList<bool> list1 = new List<bool>();
    list1.Add(true); list1.Add(false); list1.Add(true); list1.Add(false);
    Print(list1);                   // Must print: true false true false
    bool b1 = list1[3];             // false
    Console.WriteLine(b1);
    IDictionary<String, int> dict1 = new Dictionary<String, int>();
    dict1.Add("Sweden", 46); dict1.Add("Germany", 49); dict1.Add("Japan", 81);
    Print(dict1.Keys);              // May print:  Japan Sweden Germany
    Print(dict1.Values);            // May print:  81 46 49
    int i1 = dict1["Japan"];       // 81
    Console.WriteLine(i1);
    Print(dict1);                   // Print key/value pairs in some order
    IDictionary<String, int> dict2 = new SortedDictionary<String, int>();
    dict2.Add("Sweden", 46); dict2.Add("Germany", 49); dict2.Add("Japan", 81);
    Print(dict2.Keys);              // Must print: Germany Japan Sweden
    Print(dict2.Values);            // Must print: 49 81 46
    Print(dict2);                   // Print key/value pairs in sorted key order
  }

  public static void Print<T>(ICollection<T> coll) {
    foreach (T x in coll)
      Console.Write("{0} ", x);
    Console.WriteLine();
  }

  public static void Print<K,V>(IDictionary<K,V> dict) {
    foreach (KeyValuePair<K,V> entry in dict)
      Console.WriteLine("{0} --> {1}", entry.Key, entry.Value);
    Console.WriteLine();
  }
}
```

```
// Example 222 from page 183 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// A generic LinkedList class

using System;
using System.IO;                        // TextWriter
using System.Collections.Generic;       // IEnumerable<T>, IEnumerator<T>
using SC = System.Collections;          // IEnumerable, IEnumerator


public interface IMyList<T> : IEnumerable<T>, IEquatable<IMyList<T>> {
  int Count { get; }                    // Number of elements
  T this[int i] { get; set; }           // Get or set element at index i
  void Add(T item);                     // Add element at end
  void Insert(int i, T item);           // Insert element at index i
  void RemoveAt(int i);                 // Remove element at index i
  IMyList<U> Map<U>(Func<T,U> f);       // Map f over all elements
}

public class LinkedList<T> : IMyList<T> {
  protected int size;                // Number of elements in the list
  protected Node first, last;        // Invariant: first==null iff last==null

  protected class Node {
    public Node prev, next;
    public T item;

    public Node(T item) {
      this.item = item;
    }

    public Node(T item, Node prev, Node next) {
      this.item = item; this.prev = prev; this.next = next;
    }
  }

  public LinkedList() {
    first = last = null;
    size = 0;
  }

  public LinkedList(params T[] arr) : this() {
    foreach (T x in arr)
      Add(x);
  }

  public int Count {
    get { return size; }
  }

  public T this[int i] {
    get { return get(i).item; }
    set { get(i).item = value; }
  }

  private Node get(int n) {
    if (n < 0 || n >= size)
      throw new IndexOutOfRangeException();
    else if (n < size/2) {              // Closer to front
      Node node = first;
      for (int i=0; i<n; i++)
        node = node.next;
      return node;
    } else {                            // Closer to end
      Node node = last;
      for (int i=size-1; i>n; i--)
        node = node.prev;
      return node;
    }
  }

  public void Add(T item) {
    Insert(size, item);
  }

  public void Insert(int i, T item) {
    if (i == 0) {
      if (first == null) // and thus last == null
```

```
      first = last = new Node(item);
    else {
      Node tmp = new Node(item, null, first);
      first.prev = tmp;
      first = tmp;
    }
    size++;
  } else if (i == size) {
    if (last == null) // and thus first = null
      first = last = new Node(item);
    else {
      Node tmp = new Node(item, last, null);
      last.next = tmp;
      last = tmp;
    }
    size++;
  } else {
    Node node = get(i);
    // assert node.prev != null;
    Node newnode = new Node(item, node.prev, node);
    node.prev.next = newnode;
    node.prev = newnode;
    size++;
  }
}

public void RemoveAt(int i) {
  Node node = get(i);
  if (node.prev == null)
    first = node.next;
  else
    node.prev.next = node.next;
  if (node.next == null)
    last = node.prev;
  else
    node.next.prev = node.prev;
  size--;
}

public override bool Equals(Object that) {
  return Equals(that as IMyList<T>);
}

public bool Equals(IMyList<T> that) {
  if (this == that)
    return true;
  if (that == null || this.Count != that.Count)
    return false;
  Node thisnode = this.first;
  IEnumerator<T> thatenm = that.GetEnumerator();
  while (thisnode != null) {
    if (!thatenm.MoveNext())
      throw new ApplicationException("Impossible: LinkedList<T>.Equals");
    // assert MoveNext() was true (because of the above size test)
    if (!thisnode.item.Equals(thatenm.Current))
      return false;
    thisnode = thisnode.next;
  }
  // assert !MoveNext(); // because of the size test
  return true;
}

public override int GetHashCode() {
  int hash = 0;
  foreach (T x in this)
    hash ^= x.GetHashCode();
  return hash;
}

public static explicit operator LinkedList<T>(T[] arr) {
  return new LinkedList<T>(arr);
}

public static LinkedList<T> operator +(LinkedList<T> xs1, LinkedList<T> xs2) {
  LinkedList<T> res = new LinkedList<T>();
  foreach (T x in xs1)
    res.Add(x);
  foreach (T x in xs2)
    res.Add(x);
  return res;
}

public IMyList<U> Map<U>(Func<T,U> f) {
  LinkedList<U> res = new LinkedList<U>();
  foreach (T x in this)
    res.Add(f(x));
  return res;
}

public IEnumerator<T> GetEnumerator() {
  return new LinkedListEnumerator(this);
}

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

private class LinkedListEnumerator : IEnumerator<T> {
  T curr;                    // The enumerator's current element
  bool valid;                // Is the current element valid?
  Node next;                 // Node holding the next element, or null

  public LinkedListEnumerator(LinkedList<T> lst) {
    next = lst.first; valid = false;
  }

  public T Current {
    get {
      if (valid)
        return curr;
      else
        throw new InvalidOperationException();
    }
  }

  public bool MoveNext() {
    if (next != null)  {
      curr = next.item; next = next.next; valid = true;
    } else
      valid = false;
    return valid;
  }

  public void Dispose() {
    curr = default(T);
    next = null; valid = false;
  }

  Object SC.IEnumerator.Current {
    get { return Current; }
  }

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

class SortedList<T> : LinkedList<T> where T : IComparable<T> {
  // Sorted insertion
  public void Insert(T x) {
    Node node = first;
    while (node != null && x.CompareTo(node.item) > 0)
      node = node.next;
    if (node == null)          // x > all elements; insert at end
      Add(x);
    else {                     // x <= node.item; insert before node
      Node newnode = new Node(x);
      if (node.prev == null)   // insert as first element
        first = newnode;
      else
        node.prev.next = newnode;
      newnode.next = node;
      newnode.prev = node.prev;
      node.prev = newnode;
    }
  }
}
```

```
interface IPrintable {
  void Print(TextWriter fs);
}
class PrintableLinkedList<T> : LinkedList<T>, IPrintable where T : IPrintable {
  public void Print(TextWriter fs) {
    bool firstElement = true;
    foreach (T x in this) {
      x.Print(fs);
      if (firstElement)
        firstElement = false;
      else
        fs.Write(",");
    }
  }
}

class MyString : IComparable<MyString> {
  private readonly String s;
  public MyString(String s) {
    this.s = s;
  }
  public int CompareTo(MyString that) {
    return String.Compare(that.Value, s);       // Reverse ordering
  }
  public bool Equals(MyString that) {
    return String.Equals(that.Value, s);
  }
  public String Value {
    get { return s; }
  }
}

class MyTest {
  public static void Main(String[] args) {
    LinkedList<double> dLst = new LinkedList<double>(7.0, 9.0, 13.0, 0.0);
    foreach (double d in dLst)
      Console.Write("{0} ", d);
    Console.WriteLine();
    IMyList<int> iLst = dLst.Map<int>(Math.Sign);
    foreach (int i in iLst)
      Console.Write("{0} ", i);
    Console.WriteLine();
    IMyList<String> sLst1 =
      dLst.Map<String>(delegate(double d) { return "s" + d; });
    foreach (String s in sLst1)
      Console.Write("{0} ", s);
    Console.WriteLine();
    IMyList<String> sLst2 = dLst.Map<String>(d => "s" + d);
    foreach (String s in sLst2)
      Console.Write("{0} ", s);
    Console.WriteLine();
    // Testing SortedList<MyString>
    SortedList<MyString> sortedLst = new SortedList<MyString>();
    sortedLst.Insert(new MyString("New York"));
    sortedLst.Insert(new MyString("Rome"));
    sortedLst.Insert(new MyString("Dublin"));
    sortedLst.Insert(new MyString("Riyadh"));
    sortedLst.Insert(new MyString("Tokyo"));
    foreach (MyString s in sortedLst)
      Console.Write("{0} ", s.Value);
    Console.WriteLine();
    // MyList equality
    Console.WriteLine(dLst.Equals(dLst));       // True
    Console.WriteLine(dLst.Equals(sLst1));      // False
    Console.WriteLine(sLst1.Equals(sLst2));     // True
    Console.WriteLine(sLst1.Equals(null));      // False
  }
}
```

```
// Example 223 from page 185 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;                           // IComparable, IComparable<T>, IEquatable<T>
using System.Collections.Generic;       // IDictionary<K,V>, SortedDictionary<K,V>

public class Time : IComparable, IComparable<Time>, IEquatable<Time> {
  private readonly int hh, mm;                  // 24-hour clock

  public Time(int hh, int mm) { this.hh = hh; this.mm = mm; }

  // Return neg if before that; return pos if after that; return zero if same
  public int CompareTo(Object that) {           // For IComparable
    return CompareTo((Time)that);
  }

  public int CompareTo(Time that) {             // For IComparable<T>
    return hh != that.hh ? hh - that.hh : mm - that.mm;
  }

  public bool Equals(Time that) {               // For IEquatable<T>
    return hh == that.hh && mm == that.mm;
  }

  public override String ToString() {
    return String.Format("{0:00}:{1:00}", hh, mm);
  }
}

class TestDatebook {
  public static void Main(String[] args) {
    IDictionary<Time,String> datebook = new SortedDictionary<Time,String>();
    datebook.Add(new Time(12, 30), "Lunch");
    datebook.Add(new Time(15, 30), "Afternoon coffee break");
    datebook.Add(new Time( 9,  0), "Lecture");
    datebook.Add(new Time(13, 15), "Board meeting");
    foreach (KeyValuePair<Time,String> entry in datebook)
      Console.WriteLine(entry.Key + " " + entry.Value);
  }
}
```

```
// Example 224 from page 185 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using SC = System.Collections;                  // IComparer
using System.Collections.Generic;               // IComparer<T>, IEqualityComparer<T>

public struct IntPair {
  public readonly int Fst, Snd;
  public IntPair(int fst, int snd) {
    this.Fst = fst; this.Snd = snd;
  }
  public override String ToString() {
    return String.Format("({0},{1})", Fst, Snd);
  }
}

public class IntPairComparer : SC.IComparer, IComparer<IntPair>, IEqualityComparer<Int
Pair>  {
  public int Compare(Object o1, Object o2) {    // For SC.IComparer
    return Compare((IntPair)o1, (IntPair)o2);
  }
  public int Compare(IntPair v1, IntPair v2) {  // For IComparer<T>
    return v1.Fst<v2.Fst ? -1 : v1.Fst>v2.Fst ? +1
         : v1.Snd<v2.Snd ? -1 : v1.Snd>v2.Snd ? +1 : 0;
  }
  public bool Equals(IntPair v1, IntPair v2) {  // For IEqualityComparer<T>
    return v1.Fst==v2.Fst && v1.Snd==v2.Snd;
  }
  public int GetHashCode(IntPair v) {           // For IEqualityComparer<T>
    return v.Fst ^ v.Snd;
  }
}

class MyTest {
  public static void Main(String[] args) {
    IntPair[] ips =
      { new IntPair(15, 15), new IntPair(12, 30), new IntPair(15, 30) };
    Dictionary<IntPair, String> dict0 = new Dictionary<IntPair, String>();
    foreach (IntPair ip in ips)
      dict0.Add(ip, "meeting");
    foreach (KeyValuePair<IntPair, String> entry in dict0)
      Console.WriteLine("{0}: {1}", entry.Key, entry.Value);
    Dictionary<IntPair, String> dict1 =
      new Dictionary<IntPair, String>(dict0, new IntPairComparer());
  }
}
```

```
// Example 225 from page 187 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// File index: read a text file, build and print a list of all words
// and the line numbers (possibly with duplicates) at which they occur.

using System;                            // Console
using System.Collections.Generic;        // Dictionary, List
using System.IO;                         // StreamReader, TextReader
using System.Text.RegularExpressions;    // Regex

class Example225 {
  static void Main(String[] args) {
    if (args.Length != 1)
      Console.WriteLine("Usage: Example225 <filename>\n");
    else {
      IDictionary<String, List<int>> index = IndexFile(args[0]);
      PrintIndex(index);
    }
  }

  static IDictionary<String, List<int>> IndexFile(String filename) {
    IDictionary<String, List<int>> index = new Dictionary<String, List<int>>();
    Regex delim = new Regex("[^a-zA-Z0-9]+");
    TextReader rd = new StreamReader(filename);
    int lineno = 0;
    String line;
    while (null != (line = rd.ReadLine())) {
      String[] res = delim.Split(line);
      lineno++;
      foreach (String s in res)
        if (s != "") {
          if (!index.ContainsKey(s))
            index[s] = new List<int>();
          index[s].Add(lineno);
        }
    }
    rd.Close();
    return index;
  }

  static void PrintIndex(IDictionary<String, List<int>> index) {
    List<String> words = new List<String>(index.Keys);
    words.Sort();
    foreach (String word in words) {
      Console.Write("{0}:", word);
      foreach (int ln in index[word])
        Console.Write("{0} ", ln);
      Console.WriteLine();
    }
  }
}
```

```
// Example 226 from page 187 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Representing sets of ints using a HashSet.
// sestoft@itu.dk 2001, 2011-08-05

// In contrast to the .NET 4.0 HashSet<T> implementation, the
// GetHashCode for this set representation is based on the set's
// elements, so Set<Set<int>> and Dictionary<Set<int>>,V> work as one
// would expect from mathematics.  We cache the hash code between
// calls to GetHashCode() to avoid needlessly recomputing it, and of
// course invalidate it when the collection is modified by Add, Clear
// or Remove.

// Note that we use composition (private field HashSet<T> inner).  It
// does not work to use subclassing (class Set<T> : HashSet<T>)
// because the Add and Remove methods on HashSet<T> are non-virtual,
// so the validity of the cached hash code can be undermined by
// calling ((HashSet<T>)set).Add(item).  Dubious .NET 4.0 design,
// probably to save some nanoseconds.

// Computing the intersection closure to illustrate a worklist algorithm.

using System;                           // Console
using System.Text;                      // StringBuilder
using SC = System.Collections;          // IEnumerable, IEnumerator
using System.Collections.Generic;       // Dictionary<K,V>, IEnumerable<T>

class Set<T> : IEquatable<Set<T>>, ICollection<T> where T : IEquatable<T> {
  private readonly HashSet<T> inner = new HashSet<T>();
  private int? cachedHash = null;   // Cached hash code is valid if non-null

  public Set() { }

  public Set(T x) : this() {
    Add(x);
  }

  public Set(IEnumerable<T> coll) : this() {
    foreach (T x in coll)
      Add(x);
  }

  public bool Contains(T x) {
    return inner.Contains(x);
  }

  public void Add(T x) {
    if (!Contains(x)) {
      inner.Add(x);
      cachedHash = null;
    }
  }

  public bool Remove(T x) {
    bool removed = inner.Remove(x);
    if (removed)
      cachedHash = null;
    return removed;
  }

  public IEnumerator<T> GetEnumerator() {
    return inner.GetEnumerator();
  }

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

  public int Count {
    get { return inner.Count; }
  }

  public void CopyTo(T[] arr, int i) {
    inner.CopyTo(arr, i);
  }

  public void Clear() {
    inner.Clear();
    cachedHash = null;
  }

  public bool IsReadOnly {
    get { return false; }
  }

  // Is this set a subset of that?
  public bool IsSubsetOf(Set<T> that) {
    foreach (T x in this)
      if (!that.Contains(x))
        return false;
    return true;
  }

  // Create new set as intersection of this and that
  public Set<T> Intersection(Set<T> that) {
    Set<T> res = new Set<T>();
    foreach (T x in this)
      if (that.Contains(x))
        res.Add(x);
    return res;
  }

  // Create new set as union of this and that
  public Set<T> Union(Set<T> that) {
    Set<T> res = new Set<T>(this);
    foreach (T x in that)
      res.Add(x);
    return res;
  }

  // Create new set as difference between this and that
  public Set<T> Difference(Set<T> that) {
    Set<T> res = new Set<T>();
    foreach (T x in this)
      if (!that.Contains(x))
        res.Add(x);
    return res;
  }

  // Create new set as symmetric difference between this and that
  public Set<T> SymmetricDifference(Set<T> that) {
    Set<T> res = new Set<T>();
    foreach (T x in this)
      if (!that.Contains(x))
        res.Add(x);
    foreach (T x in that)
      if (!this.Contains(x))
        res.Add(x);
    return res;
  }

  // Compute hash code based on set contents, and cache it
  public override int GetHashCode() {
    if (!cachedHash.HasValue) {
      int res = 0;
      foreach (T x in this)
        res ^= x.GetHashCode();
      cachedHash = res;
    }
    return cachedHash.Value;
  }

  public bool Equals(Set<T> that) {
    return that != null && that.Count == this.Count && that.IsSubsetOf(this);
  }

  public override String ToString() {
    StringBuilder res = new StringBuilder();
    res.Append("{");
    bool first = true;
    foreach (T x in this) {
      if (!first)
        res.Append(",");
      res.Append(x);
      first = false;
    }
    res.Append("}");
```

```
    return res.ToString();
  }
}

class IntersectionClosure {
  public static void Main(String[] args) {
    Set<Set<int>> SS = new Set<Set<int>>();
    SS.Add(new Set<int>(new int[] { 2, 3 }));
    SS.Add(new Set<int>(new int[] { 1, 3 }));
    SS.Add(new Set<int>(new int[] { 1, 2 }));
    Console.WriteLine("SS = " + SS);
    Set<Set<int>> TT = IntersectionClose(SS);
    Console.WriteLine("TT = " + TT);
  }

  // Given a set SS of sets of Integers, compute its intersection
  // closure, that is, the least set TT such that SS is a subset of TT
  // and such that for any two sets t1 and t2 in TT, their
  // intersection is also in TT.

  // For instance, if SS is {{2,3}, {1,3}, {1,2}},
  // then TT is {{2,3}, {1,3}, {1,2}, {3}, {2}, {1}, {}}.

  // Both the argument and the result is a Set<Set<int>>

  static Set<Set<T>> IntersectionClose<T>(Set<Set<T>> ss) where T : IEquatable<T> {
    Queue<Set<T>> worklist = new Queue<Set<T>>(ss);
    Set<Set<T>> tt = new Set<Set<T>>();
    while (worklist.Count != 0) {
      Set<T> s = worklist.Dequeue();
      foreach (Set<T> t in tt) {
        Set<T> ts = t.Intersection(s);
        if (!tt.Contains(ts))
          worklist.Enqueue(ts);
      }
      tt.Add(s);
    }
    return tt;
  }
}
```

```
// Example 227 from page 189 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

class MyTest {
  public static void Main(String[] args) {
    List<int> lst = new List<int>();
    lst.Add(7); lst.Add(9); lst.Add(13); lst.Add(7);
    Print(lst);                          // 7 9 13 7
    int i1 = lst[2];                     // 13
    Console.WriteLine(i1);
    int i2 = lst.IndexOf(7);             // 0
    Console.WriteLine(i2);
    int i3 = lst.IndexOf(12);            // -1
    Console.WriteLine(i3);
    lst.Remove(8); Print(lst);           // 7 9 13 7
    lst.Remove(7); Print(lst);           // 9 13 7
    lst.Insert(3, 88); Print(lst);       // 9 13 7 88
    int count = 100000;
    Console.WriteLine("Adding elements at end of list (fast) ...");
    for (int i=0; i<count; i++) {
      lst.Add(i);
      if (i % 5000 == 0)
        Console.Write("{0} ", i);
    }
    Console.WriteLine();
    lst.Clear();
    Console.WriteLine("Adding elements at head of list (slow) ...");
    for (int i=0; i<count; i++) {
      lst.Insert(0, i);
      if (i % 5000 == 0)
        Console.Write("{0} ", i);
    }
    Console.WriteLine();
  }

  public static void Print<T>(ICollection<T> coll) {
    foreach (T x in coll)
      Console.Write("{0} ", x);
    Console.WriteLine();
  }
}
```

```
// Example 228 from page 189 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// This example requires:

//  * MySQL-ODBC driver 3.51
//  * A MySQL database server on host ellemose with a database
//    test containing a table Message declared like this:
//       CREATE TABLE Message
//              (name VARCHAR(80),
//               msg VARCHAR(200),
//               severity INT);
//  * Due to the default security restrictions it cannot be run from a
//    network drive.

using System;
using System.Data.Odbc;           // OdbcConnection OdbcCommand OdbcDataReader
using System.Collections.Generic; // List<T>

class Example228 {
  static Record[] GetMessages(OdbcConnection conn) {
    String query = "SELECT name, msg, severity FROM Message ORDER BY name";
    OdbcCommand cmd = new OdbcCommand(query, conn);
    OdbcDataReader r = cmd.ExecuteReader();
    List<Record> results = new List<Record>();
    while (r.Read())
      results.Add(new Record(r.GetString(0), r.GetString(1), r.GetInt32(2)));
    r.Close();
    return results.ToArray();
  }

  struct Record {
    public readonly String name, msg;
    public readonly int severity;
    public Record(String name, String msg, int severity) {
      this.name = name; this.msg = msg; this.severity = severity;
    }
    public override String ToString() {
      return String.Format("{0}: {1} ({2})", name, msg, severity);
    }
  }

  public static void Main(String[] args) {
    if (args.Length != 1)
      Console.WriteLine("Usage: Example228 <password>\n");
    else {
      String setup =
        "DRIVER={MySQL ODBC 3.51 Driver};"
        + "SERVER=sql.dina.kvl.dk;"
        + "DATABASE=test;"
        + "UID=sestoft;"
        + "PASSWORD=" + args[0] + ";";
      using (OdbcConnection conn = new OdbcConnection(setup)) {
        conn.Open();
        Record[] results = GetMessages(conn);
        foreach (Record rec in results)
          Console.WriteLine(rec);
      }
    }
  }
}
```

```
// Example 229 from page 191 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;

class MyTest {
  public static void Main(String[] args) {
    Dictionary<String, int> dict = new Dictionary<String, int>();
    dict.Add("Sweden", 46); dict.Add("Germany", 49);
    dict["Japan"] = 81;                          // New entry, no exception thrown
    Print(dict.Keys);                             // Japan Sweden Germany
    Console.WriteLine(dict.Count);                // 3
    // Console.WriteLine(dict["Greece"]);         // ArgumentException
    // dict.Add("Germany", 49);                   // ArgumentException
    bool b1 = dict.Remove("Greece");             // False (but no exception)
    Console.WriteLine(b1);                        //
    bool b2 = dict.Remove("Japan");              // True
    Console.WriteLine(b2);                        //
    Print(dict.Keys);                             // Sweden Germany
    bool b3 = dict.ContainsKey("Germany");       // True
    Console.WriteLine(b3);                        //
    dict["Sweden"] = 45;                          // No exception
    Console.WriteLine(dict["Sweden"]);            // 45
  }

  public static void Print<T>(ICollection<T> coll) {
    foreach (T x in coll)
      Console.Write("{0} ", x);
    Console.WriteLine();
  }

  public static void Print<K,V>(IDictionary<K,V> dict) {
    foreach (KeyValuePair<K,V> entry in dict)
      Console.WriteLine("{0} --> {1}", entry.Key, entry.Value);
    Console.WriteLine();
  }
}
```

```
// Example 230 from page 191 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;
using System.Collections.Generic;
using System.Diagnostics;

class Example230 {
  static readonly String[] keywordarray =
    { "abstract", "as", "base", "bool", "break", "byte", "case", "catch",
      "char", "checked", "class", "const", "continue", "decimal", "default",
      "delegate", "do", "double", "else", "enum", "event", "explicit",
      "extern", "false", "finally", "fixed", "float", "for", "foreach",
      "goto", "if", "implicit", "in", "int", "interface", "internal", "is",
      "lock", "long", "namespace", "new", "null", "object", "operator",
      "out", "override", "params", "private", "protected", "public",
      "readonly", "ref", "return", "sbyte", "sealed", "short", "sizeof",
      "stackalloc", "static", "string", "struct", "switch", "this", "throw",
      "true", "try", "typeof", "uint", "ulong", "unchecked", "unsafe",
      "ushort", "using", "virtual", "void", "volatile", "while" };

  static readonly ISet<String> keywords = new HashSet<String>();

  static Example230() {
    foreach (String keyword in keywordarray)
      keywords.Add(keyword);
  }

  static bool IsKeyword1(String id) {
    return keywords.Contains(id);
  }

  static bool IsKeyword2(String id) {
    return Array.BinarySearch(keywordarray, id) >= 0;
  }

  public static void Main(String[] args) {
    if (args.Length != 2)
      Console.WriteLine("Usage: Example230 <iterations> <word>\n");
    else {
      int count = int.Parse(args[0]);
      String id = args[1];
      for (int i=0; i<keywordarray.Length; i++)
        if (IsKeyword1(keywordarray[i]) != IsKeyword2(keywordarray[i]))
          Console.WriteLine("Error at i = " + i);
      if (IsKeyword1(id) != IsKeyword2(id))
        Console.WriteLine("Error at id = " + id);

      Console.Write("HashSet.Contains ");
      Stopwatch sw = new Stopwatch();
      sw.Start();
      for (int i=0; i<count; i++)
        IsKeyword1(id);
      Console.WriteLine("{0} ms", sw.ElapsedMilliseconds);

      Console.Write("Array.BinarySearch ");
      sw.Reset();
      sw.Start();
      for (int i=0; i<count; i++)
        IsKeyword2(id);
      Console.WriteLine("{0} ms", sw.ElapsedMilliseconds);
    }
  }
}
```

```
// Example 231 from page 191 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// RegExp -> NFA -> DFA -> Graph in Generic C#
// 2001-10-23, 2003-09-03, 2011-08-04

// This file contains, in order:
//   * A class Nfa for representing an NFA (a nondeterministic finite
//     automaton), and for converting it to a DFA (a deterministic
//     finite automaton).  Most complexity is in this class.
//   * A class Dfa for representing a DFA, a deterministic finite
//     automaton, and for writing a dot input file representing the DFA.
//   * Classes for representing regular expressions, and for building an
//     NFA from a regular expression
//   * A test class that creates an NFA, a DFA, and a dot input file
//     for a number of small regular expressions.  The DFAs are
//     not minimized.

using System;
using System.Text;
using SC = System.Collections;
using System.Collections.Generic;
using System.IO;

// A set, with element-based hash codes, built upon HashSet<T>

class Set<T> : IEquatable<Set<T>>, ICollection<T> where T : IEquatable<T> {
  private readonly HashSet<T> inner = new HashSet<T>();
  private int? cachedHash = null;   // Cached hash code is valid if non-null

  public Set() { }

  public Set(T x) : this() {
    Add(x);
  }

  public Set(IEnumerable<T> coll) : this() {
    foreach (T x in coll)
      Add(x);
  }

  public bool Contains(T x) {
    return inner.Contains(x);
  }

  public void Add(T x) {
    if (!Contains(x)) {
      inner.Add(x);
      cachedHash = null;
    }
  }

  public bool Remove(T x) {
    bool removed = inner.Remove(x);
    if (removed)
      cachedHash = null;
    return removed;
  }

  public IEnumerator<T> GetEnumerator() {
    return inner.GetEnumerator();
  }

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

  public int Count {
    get { return inner.Count; }
  }

  public void CopyTo(T[] arr, int i) {
    inner.CopyTo(arr, i);
  }

  public void Clear() {
    inner.Clear();
    cachedHash = null;
  }
```

```
  public bool IsReadOnly {
    get { return false; }
  }

  // Is this set a subset of that?
  public bool IsSubsetOf(Set<T> that) {
    foreach (T x in this)
      if (!that.Contains(x))
        return false;
    return true;
  }

  // Create new set as intersection of this and that
  public Set<T> Intersection(Set<T> that) {
    Set<T> res = new Set<T>();
    foreach (T x in this)
      if (that.Contains(x))
        res.Add(x);
    return res;
  }

  // Create new set as union of this and that
  public Set<T> Union(Set<T> that) {
    Set<T> res = new Set<T>(this);
    foreach (T x in that)
      res.Add(x);
    return res;
  }

  // Create new set as difference between this and that
  public Set<T> Difference(Set<T> that) {
    Set<T> res = new Set<T>();
    foreach (T x in this)
      if (!that.Contains(x))
        res.Add(x);
    return res;
  }

  // Create new set as symmetric difference between this and that
  public Set<T> SymmetricDifference(Set<T> that) {
    Set<T> res = new Set<T>();
    foreach (T x in this)
      if (!that.Contains(x))
        res.Add(x);
    foreach (T x in that)
      if (!this.Contains(x))
        res.Add(x);
    return res;
  }

  // Compute hash code based on set contents, and cache it
  public override int GetHashCode() {
    if (!cachedHash.HasValue) {
      int res = 0;
      foreach (T x in this)
        res ^= x.GetHashCode();
      cachedHash = res;
    }
    return cachedHash.Value;
  }

  public bool Equals(Set<T> that) {
    return that != null && that.Count == this.Count && that.IsSubsetOf(this);
  }

  public override String ToString() {
    StringBuilder res = new StringBuilder();
    res.Append("{");
    bool first = true;
    foreach (T x in this) {
      if (!first)
        res.Append(",");
      res.Append(x);
      first = false;
    }
    res.Append("}");
    return res.ToString();
  }
}

// ------------------------------------------------------------------------

// Regular expressions, NFAs, DFAs, and dot graphs
// sestoft@itu.dk
// Java 2001-07-10 * C# 2001-10-22 * Generic C# 2001-10-23, 2003-09-03

// We
//  use Queue<int> and Queue<Set<int>> for worklists
//  use Set<int> for pre-DFA states
//  use List<Transition> for NFA transition relations
//  use Dictionary<Set<int>, Dictionary<String, Set<int>>>
//  use Dictionary<int, Dictionary<String, int>> for DFA transition relations
//  and we need to use Set<int> because it has a proper element-based
//      GetHashCode(); the .NET 4.0 HashSet<T> and SortedSet<T> do not.

/* Class Nfa and conversion from NFA to DFA ---------------------------

  A nondeterministic finite automaton (NFA) is represented as a
  Map from state number (int) to a List of Transitions, a
  Transition being a pair of a label lab (a String, null meaning
  epsilon) and a target state (an int).

  A DFA is created from an NFA in two steps:

    (1) Construct a DFA whose each of whose states is composite,
        namely a set of NFA states (Set of int).  This is done by
        methods CompositeDfaTrans and EpsilonClose.

    (2) Replace composite states (Set of int) by simple states
        (int).  This is done by methods Rename and MkRenamer.

  Method CompositeDfaTrans works as follows:

    Create the epsilon-closure S0 (a Set of ints) of the start
    state s0, and put it in a worklist (a Queue).  Create an
    empty DFA transition relation, which is a Map from a
    composite state (an epsilon-closed Set of ints) to a Map
    from a label (a non-null String) to a composite state.

    Repeatedly choose a composite state S from the worklist.  If it is
    not already in the keyset of the DFA transition relation, compute
    for every non-epsilon label lab the set T of states reachable by
    that label from some state s in S.  Compute the epsilon-closure
    Tclose of every such state T and put it on the worklist.  Then add
    the transition S -lab-> Tclose to the DFA transition relation, for
    every lab.

  Method EpsilonClose works as follows:

    Given a set S of states.  Put the states of S in a worklist.
    Repeatedly choose a state s from the worklist, and consider all
    epsilon-transitions s -eps-> s' from s.  If s' is in S already,
    then do nothing; otherwise add s' to S and the worklist.  When the
    worklist is empty, S is epsilon-closed; return S.

  Method MkRenamer works as follows:

    Given a Map from Set of int to something, create an
    injective Map from Set of int to int, by choosing a fresh
    int for every value of the map.

  Method Rename works as follows:

    Given a Map from Set of int to Map from String to Set of
    int, use the result of MkRenamer to replace all Sets of ints
    by ints.

*/


class Nfa {
  private readonly int startState;
  private readonly int exitState;    // This is the unique accept state
  private readonly IDictionary<int,List<Transition>> trans;

  public Nfa(int startState, int exitState) {
    this.startState = startState; this.exitState = exitState;
```

```
    trans = new Dictionary<int,List<Transition>>();
    if (!startState.Equals(exitState))
      trans.Add(exitState, new List<Transition>());
  }

  public int Start { get { return startState; } }

  public int Exit { get { return exitState; } }

  public IDictionary<int, List<Transition>> Trans {
    get { return trans; }
  }

  public void AddTrans(int s1, String lab, int s2) {
    List<Transition> s1Trans;
    if (trans.ContainsKey(s1))
      s1Trans = trans[s1];
    else {
      s1Trans = new List<Transition>();
      trans.Add(s1, s1Trans);
    }
    s1Trans.Add(new Transition(lab, s2));
  }

  public void AddTrans(KeyValuePair<int, List<Transition>> tr) {
    // Assumption: if tr is in trans, it maps to an empty list (end state)
    trans.Remove(tr.Key);
    trans.Add(tr.Key, tr.Value);
  }

  public override String ToString() {
    return "NFA start=" + startState + " exit=" + exitState;
  }

  // Construct the transition relation of a composite-state DFA
  // from an NFA with start state s0 and transition relation
  // trans (a Map from int to List of Transition).  The start
  // state of the constructed DFA is the epsilon closure of s0,
  // and its transition relation is a Map from a composite state
  // (a Set of ints) to a Map from label (a String) to a
  // composite state (a Set of ints).

  static IDictionary<Set<int>, IDictionary<String, Set<int>>>
    CompositeDfaTrans(int s0, IDictionary<int, List<Transition>> trans) {
    Set<int> S0 = EpsilonClose(new Set<int>(s0), trans);
    Queue<Set<int>> worklist = new Queue<Set<int>>();
    worklist.Enqueue(S0);
    // The transition relation of the DFA
    IDictionary<Set<int>, IDictionary<String, Set<int>>> res =
      new Dictionary<Set<int>, IDictionary<String, Set<int>>>();
    while (worklist.Count != 0) {
      Set<int> S = worklist.Dequeue();
      if (!res.ContainsKey(S)) {
        // The S -lab-> T transition relation being constructed for a given S
        IDictionary<String, Set<int>> STrans =
          new Dictionary<String, Set<int>>();
        // For all s in S, consider all transitions s -lab-> t
        foreach (int s in S) {
          // For all non-epsilon transitions s -lab-> t, add t to T
          foreach (Transition tr in trans[s]) {
            if (tr.lab != null) {       // Already a transition on lab
              Set<int> toState;
              if (STrans.ContainsKey(tr.lab))
                toState = STrans[tr.lab];
              else {                    // No transitions on lab yet
                toState = new Set<int>();
                STrans.Add(tr.lab, toState);
              }
              toState.Add(tr.target);
            }
          }
        }
        // Epsilon-close all T such that S -lab-> T, and put on worklist
        Dictionary<String, Set<int>> STransClosed =
          new Dictionary<String, Set<int> >();
        foreach (KeyValuePair<String, Set<int>> entry in STrans) {
          Set<int> Tclose = EpsilonClose(entry.Value, trans);
          STransClosed.Add(entry.Key, Tclose);
          worklist.Enqueue(Tclose);
        }
        res.Add(S, STransClosed);
      }
    }
    return res;
  }

  // Compute epsilon-closure of state set S in transition relation trans.

  static Set<int>
    EpsilonClose(Set<int> S, IDictionary<int, List<Transition>> trans) {
    // The worklist initially contains all S members
    Queue<int> worklist = new Queue<int>(S);
    Set<int> res = new Set<int>(S);
    while (worklist.Count != 0) {
      int s = worklist.Dequeue();
      foreach (Transition tr in trans[s]) {
        if (tr.lab == null && !res.Contains(tr.target)) {
          res.Add(tr.target);
          worklist.Enqueue(tr.target);
        }
      }
    }
    return res;
  }

  // Compute a renamer, which is a Map from Set of int to int

  static IDictionary<Set<int>, int> MkRenamer(ICollection<Set<int>> states) {
    IDictionary<Set<int>, int> renamer = new Dictionary<Set<int>, int>();
    int count = 0;
    foreach (Set<int> k in states)
      renamer.Add(k, count++);
    return renamer;
  }

  // Using a renamer (a Map from Set of int to int), replace
  // composite (Set of int) states with simple (int) states in
  // the transition relation trans, which is assumed to be a Map
  // from Set of int to Map from String to Set of int.  The
  // result is a Map from int to Map from String to int.

  static IDictionary<int, IDictionary<String, int>>
    Rename(IDictionary<Set<int>, int> renamer,
           IDictionary<Set<int>, IDictionary<String, Set<int>>> trans) {
    IDictionary<int, IDictionary<String, int>> newtrans =
      new Dictionary<int, IDictionary<String, int>>();
    foreach (KeyValuePair<Set<int>, IDictionary<String, Set<int>>> entry
             in trans) {
      Set<int> k = entry.Key;
      IDictionary<String, int> newktrans = new Dictionary<String, int>();
      foreach (KeyValuePair<String, Set<int>> tr in entry.Value)
        newktrans.Add(tr.Key, renamer[tr.Value]);
      newtrans.Add(renamer[k], newktrans);
    }
    return newtrans;
  }

  static Set<int> AcceptStates(ICollection<Set<int>> states,
                               IDictionary<Set<int>, int> renamer,
                               int exit) {
    Set<int> acceptStates = new Set<int>();
    foreach (Set<int> state in states)
      if (state.Contains(exit))
        acceptStates.Add(renamer[state]);
    return acceptStates;
  }

  public Dfa ToDfa() {
    IDictionary<Set<int>, IDictionary<String, Set<int>>>
      cDfaTrans = CompositeDfaTrans(startState, trans);
    Set<int> cDfaStart = EpsilonClose(new Set<int>(startState), trans);
    ICollection<Set<int>> cDfaStates = cDfaTrans.Keys;
    IDictionary<Set<int>, int> renamer = MkRenamer(cDfaStates);
    IDictionary<int, IDictionary<String, int>> simpleDfaTrans =
      Rename(renamer, cDfaTrans);
    int simpleDfaStart = renamer[cDfaStart];
    Set<int> simpleDfaAccept = AcceptStates(cDfaStates, renamer, exitState);
    return new Dfa(simpleDfaStart, simpleDfaAccept, simpleDfaTrans);
```

```
  }

  // Nested class for creating distinctly named states when constructing NFAs

  public class NameSource {
    private static int nextName = 0;

    public int next() {
      return nextName++;
    }
  }
}

// Class Transition, a transition from one state to another ----------

  public class Transition {
    public String lab;
    public int target;

    public Transition(String lab, int target) {
      this.lab = lab; this.target = target;
    }

    public override String ToString() {
      return "-" + lab + "-> " + target;
    }
  }

// Class Dfa, deterministic finite automata --------------------------

/*
  A deterministic finite automaton (DFA) is represented as a Map
  from state number (int) to a Map from label (a String,
  non-null) to a target state (an int).
*/

class Dfa {
  private readonly int startState;
  private readonly Set<int> acceptStates;
  private readonly IDictionary<int, IDictionary<String,int>> trans;

  public Dfa(int startState, Set<int> acceptStates,
             IDictionary<int, IDictionary<String,int>> trans) {
    this.startState = startState;
    this.acceptStates = acceptStates;
    this.trans = trans;
  }

  public int Start { get { return startState; } }

  public Set<int> Accept { get { return acceptStates; } }

  public IDictionary<int, IDictionary<String,int>> Trans {
    get { return trans; }
  }

  public override String ToString() {
    return "DFA start=" + startState + "\naccept=" + acceptStates;
  }

  // Write an input file for the dot program.  You can find dot at
  // http://www.research.att.com/sw/tools/graphviz/

  public void WriteDot(String filename) {
    TextWriter wr =
      new StreamWriter(new FileStream(filename, FileMode.Create,
                                      FileAccess.Write));
    wr.WriteLine("// Format this file as a Postscript file with ");
    wr.WriteLine("//   dot " + filename + " -Tps -o out.ps\n");
    wr.WriteLine("digraph dfa {");
    wr.WriteLine("size=\"11,8.25\";");
    wr.WriteLine("rotate=90;");
    wr.WriteLine("rankdir=LR;");
    wr.WriteLine("n999999 [style=invis];");     // Invisible start node
    wr.WriteLine("n999999 -> n" + startState); // Edge into start state

    // Accept states are double circles
    foreach (int state in trans.Keys)
      if (acceptStates.Contains(state))
        wr.WriteLine("n" + state + " [peripheries=2];");

    // The transitions
    foreach (KeyValuePair<int, IDictionary<String, int>> entry in trans) {
      int s1 = entry.Key;
      foreach (KeyValuePair<String, int> s1Trans in entry.Value) {
        String lab = s1Trans.Key;
        int s2 = s1Trans.Value;
        wr.WriteLine("n" + s1 + " -> n" + s2 + " [label=\"" + lab + "\"];");
      }
    }
    wr.WriteLine("}");
    wr.Close();
  }
}

// Regular expressions -----------------------------------------------
//
// Abstract syntax of regular expressions
//    r ::= A | r1 r2 | (r1|r2) | r*
//

abstract class Regex {
  abstract public Nfa MkNfa(Nfa.NameSource names);
}

class Eps : Regex {
  // The resulting nfa0 has form s0s -eps-> s0e

  public override Nfa MkNfa(Nfa.NameSource names) {
    int s0s = names.next();
    int s0e = names.next();
    Nfa nfa0 = new Nfa(s0s, s0e);
    nfa0.AddTrans(s0s, null, s0e);
    return nfa0;
  }
}

class Sym : Regex {
  String sym;

  public Sym(String sym) {
    this.sym = sym;
  }

  // The resulting nfa0 has form s0s -sym-> s0e

  public override Nfa MkNfa(Nfa.NameSource names) {
    int s0s = names.next();
    int s0e = names.next();
    Nfa nfa0 = new Nfa(s0s, s0e);
    nfa0.AddTrans(s0s, sym, s0e);
    return nfa0;
  }
}

class Seq : Regex {
  Regex r1, r2;

  public Seq(Regex r1, Regex r2) {
    this.r1 = r1; this.r2 = r2;
  }

  // If   nfa1 has form s1s ----> s1e
  // and  nfa2 has form s2s ----> s2e
  // then nfa0 has form s1s ----> s1e -eps-> s2s ----> s2e

  public override Nfa MkNfa(Nfa.NameSource names) {
    Nfa nfa1 = r1.MkNfa(names);
    Nfa nfa2 = r2.MkNfa(names);
    Nfa nfa0 = new Nfa(nfa1.Start, nfa2.Exit);
    foreach (KeyValuePair<int, List<Transition>> entry in nfa1.Trans)
      nfa0.AddTrans(entry);
    foreach (KeyValuePair<int, List<Transition>> entry in nfa2.Trans)
      nfa0.AddTrans(entry);
    nfa0.AddTrans(nfa1.Exit, null, nfa2.Start);
    return nfa0;
  }
}
```

```
class Alt : Regex {
  Regex r1, r2;

  public Alt(Regex r1, Regex r2) {
    this.r1 = r1; this.r2 = r2;
  }

  // If   nfa1 has form s1s ----> s1e
  // and  nfa2 has form s2s ----> s2e
  // then nfa0 has form s0s -eps-> s1s ----> s1e -eps-> s0e
  //                    s0s -eps-> s2s ----> s2e -eps-> s0e

  public override Nfa MkNfa(Nfa.NameSource names) {
    Nfa nfa1 = r1.MkNfa(names);
    Nfa nfa2 = r2.MkNfa(names);
    int s0s = names.next();
    int s0e = names.next();
    Nfa nfa0 = new Nfa(s0s, s0e);
    foreach (KeyValuePair<int, List<Transition>> entry in nfa1.Trans)
      nfa0.AddTrans(entry);
    foreach (KeyValuePair<int, List<Transition>> entry in nfa2.Trans)
      nfa0.AddTrans(entry);
    nfa0.AddTrans(s0s, null, nfa1.Start);
    nfa0.AddTrans(s0s, null, nfa2.Start);
    nfa0.AddTrans(nfa1.Exit, null, s0e);
    nfa0.AddTrans(nfa2.Exit, null, s0e);
    return nfa0;
  }
}

class Star : Regex {
  Regex r;

  public Star(Regex r) {
    this.r = r;
  }

  // If   nfa1 has form s1s ----> s1e
  // then nfa0 has form s0s ----> s0s
  //                    s0s -eps-> s1s
  //                    s1e -eps-> s0s

  public override Nfa MkNfa(Nfa.NameSource names) {
    Nfa nfa1 = r.MkNfa(names);
    int s0s = names.next();
    Nfa nfa0 = new Nfa(s0s, s0s);
    foreach (KeyValuePair<int, List<Transition>> entry in nfa1.Trans)
      nfa0.AddTrans(entry);
    nfa0.AddTrans(s0s, null, nfa1.Start);
    nfa0.AddTrans(nfa1.Exit, null, s0s);
    return nfa0;
  }
}

// Trying the RE->NFA->DFA translation on three regular expressions

class TestNFA {
  public static void Main(String[] args) {
    Regex a = new Sym("A");
    Regex b = new Sym("B");
    Regex c = new Sym("C");
    Regex abStar = new Star(new Alt(a, b));
    Regex bb = new Seq(b, b);
    Regex r = new Seq(abStar, new Seq(a, b));
    // The regular expression (a|b)*ab
    BuildAndShow("dfa1.dot", r);
    // The regular expression ((a|b)*ab)*
    BuildAndShow("dfa2.dot", new Star(r));
    // The regular expression ((a|b)*ab)((a|b)*ab)
    BuildAndShow("dfa3.dot", new Seq(r, r));
    // The regular expression (a|b)*abb, from ASU 1986 p 136
    BuildAndShow("dfa4.dot", new Seq(abStar, new Seq(a, bb)));
    // SML reals: sign?((digit+(\.digit+)?))([eE]sign?digit+)?
    Regex d = new Sym("digit");
    Regex dPlus = new Seq(d, new Star(d));
    Regex s = new Sym("sign");
    Regex sOpt = new Alt(s, new Eps());
    Regex dot = new Sym(".");
    Regex dotDigOpt = new Alt(new Eps(), new Seq(dot, dPlus));
    Regex mant = new Seq(sOpt, new Seq(dPlus, dotDigOpt));
    Regex e = new Sym("e");
    Regex exp = new Alt(new Eps(), new Seq(e, new Seq(sOpt, dPlus)));
    Regex smlReal = new Seq(mant, exp);
    BuildAndShow("dfa5.dot", smlReal);
  }

  public static void BuildAndShow(String filename, Regex r) {
    Nfa nfa = r.MkNfa(new Nfa.NameSource());
    Console.WriteLine(nfa);
    Console.WriteLine("---");
    Dfa dfa = nfa.ToDfa();
    Console.WriteLine(dfa);
    Console.WriteLine("Writing DFA graph to file " + filename);
    dfa.WriteDot(filename);
    Console.WriteLine();
  }
}
```

```
// Example 232 from page 193 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Representing sets of ints using a HashSet.
// sestoft@itu.dk 2001, 2011-08-05

// In contrast to the .NET 4.0 HashSet<T> implementation, the
// GetHashCode for this set representation is based on the set's
// elements, so Set<Set<int>> and Dictionary<Set<int>>,V> work as one
// would expect from mathematics.  We cache the hash code between
// calls to GetHashCode() to avoid needlessly recomputing it, and of
// course invalidate it when the collection is modified by Add, Clear
// or Remove.

// Note that we use composition (private field HashSet<T> inner).  It
// does not work to use subclassing (class Set<T> : HashSet<T>)
// because the Add and Remove methods on HashSet<T> are non-virtual,
// so the validity of the cached hash code can be undermined by
// calling ((HashSet<T>)set).Add(item).  Dubious .NET 4.0 design,
// probably to save some nanoseconds.

// Computing the intersection closure to illustrate a worklist algorithm.

using System;                           // Console
using System.Text;                      // StringBuilder
using SC = System.Collections;          // IEnumerable, IEnumerator
using System.Collections.Generic;       // Dictionary<K,V>, IEnumerable<T>

class Set<T> : IEquatable<Set<T>>, ICollection<T> where T : IEquatable<T> {
  private readonly HashSet<T> inner = new HashSet<T>();
  private int? cachedHash = null;   // Cached hash code is valid if non-null

  public Set() { }

  public Set(T x) : this() {
    Add(x);
  }

  public Set(IEnumerable<T> coll) : this() {
    foreach (T x in coll)
      Add(x);
  }

  public bool Contains(T x) {
    return inner.Contains(x);
  }

  public void Add(T x) {
    if (!Contains(x)) {
      inner.Add(x);
      cachedHash = null;
    }
  }

  public bool Remove(T x) {
    bool removed = inner.Remove(x);
    if (removed)
      cachedHash = null;
    return removed;
  }

  public IEnumerator<T> GetEnumerator() {
    return inner.GetEnumerator();
  }

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

  public int Count {
    get { return inner.Count; }
  }

  public void CopyTo(T[] arr, int i) {
    inner.CopyTo(arr, i);
  }

  public void Clear() {
    inner.Clear();
    cachedHash = null;
  }

  public bool IsReadOnly {
    get { return false; }
  }

  // Is this set a subset of that?
  public bool IsSubsetOf(Set<T> that) {
    foreach (T x in this)
      if (!that.Contains(x))
        return false;
    return true;
  }

  // Create new set as intersection of this and that
  public Set<T> Intersection(Set<T> that) {
    Set<T> res = new Set<T>();
    foreach (T x in this)
      if (that.Contains(x))
        res.Add(x);
    return res;
  }

  // Create new set as union of this and that
  public Set<T> Union(Set<T> that) {
    Set<T> res = new Set<T>(this);
    foreach (T x in that)
      res.Add(x);
    return res;
  }

  // Create new set as difference between this and that
  public Set<T> Difference(Set<T> that) {
    Set<T> res = new Set<T>();
    foreach (T x in this)
      if (!that.Contains(x))
        res.Add(x);
    return res;
  }

  // Create new set as symmetric difference between this and that
  public Set<T> SymmetricDifference(Set<T> that) {
    Set<T> res = new Set<T>();
    foreach (T x in this)
      if (!that.Contains(x))
        res.Add(x);
    foreach (T x in that)
      if (!this.Contains(x))
        res.Add(x);
    return res;
  }

  // Compute hash code based on set contents, and cache it
  public override int GetHashCode() {
    if (!cachedHash.HasValue) {
      int res = 0;
      foreach (T x in this)
        res ^= x.GetHashCode();
      cachedHash = res;
    }
    return cachedHash.Value;
  }

  public bool Equals(Set<T> that) {
    return that != null && that.Count == this.Count && that.IsSubsetOf(this);
  }

  public override String ToString() {
    StringBuilder res = new StringBuilder();
    res.Append("{");
    bool first = true;
    foreach (T x in this) {
      if (!first)
        res.Append(",");
      res.Append(x);
      first = false;
    }
    res.Append("}");
```

```
    return res.ToString();
  }
}

class IntersectionClosure {
  public static void Main(String[] args) {
    Set<Set<int>> SS = new Set<Set<int>>();
    SS.Add(new Set<int>(new int[] { 2, 3 }));
    SS.Add(new Set<int>(new int[] { 1, 3 }));
    SS.Add(new Set<int>(new int[] { 1, 2 }));
    Console.WriteLine("SS = " + SS);
    Set<Set<int>> TT = IntersectionClose(SS);
    Console.WriteLine("TT = " + TT);
  }

  // Given a set SS of sets of Integers, compute its intersection
  // closure, that is, the least set TT such that SS is a subset of TT
  // and such that for any two sets t1 and t2 in TT, their
  // intersection is also in TT.

  // For instance, if SS is {{2,3}, {1,3}, {1,2}},
  // then TT is {{2,3}, {1,3}, {1,2}, {3}, {2}, {1}, {}}.

  // Both the argument and the result is a Set<Set<int>>

  static Set<Set<T>> IntersectionClose<T>(Set<Set<T>> ss) where T : IEquatable<T> {
    Queue<Set<T>> worklist = new Queue<Set<T>>(ss);
    Set<Set<T>> tt = new Set<Set<T>>();
    while (worklist.Count != 0) {
      Set<T> s = worklist.Dequeue();
      foreach (Set<T> t in tt) {
        Set<T> ts = t.Intersection(s);
        if (!tt.Contains(ts))
          worklist.Enqueue(ts);
      }
      tt.Add(s);
    }
    return tt;
  }
}
```

```
// Example 233 from page 193 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Representing and traversing a directed graph with unlabelled edges.

using System;                           // Console
using System.Collections.Generic;       // Dictionary, Queue, Stack

// Graph nodes labelled with a T value

public class Node<T> {
  private readonly T label;
  private Node<T>[] neighbors;

  public Node(T label) : this(label, new Node<T>[] { }) { }

  public Node(T label, Node<T>[] neighbors) {
    this.label = label;
    this.neighbors = neighbors;
  }

  public Node<T>[] Neighbors {
    get { return neighbors; }
    set { neighbors = value; }
  }

  // Visit all nodes reachable from root.
  // The Dictionary is used as a set of nodes; only the key matters,
  // and the value false associated with every key is ignored.
  // Using Queue (and Enqueue, Dequeue) gives breadth-first traversal
  // Using Stack (and Push, Pop) gives depth-first traversal

  public void VisitBreadthFirst() {
    Dictionary<Node<T>,bool> visited = new Dictionary<Node<T>,bool>();
    Queue<Node<T>> worklist = new Queue<Node<T>>();
    visited.Add(this, false);
    worklist.Enqueue(this);
    // Invariant: every node in the worklist is also in the visited set
    while (worklist.Count != 0) {
      Node<T> node = worklist.Dequeue();
      Console.Write("{0} ", node.label);
      foreach (Node<T> neighbor in node.Neighbors)
        if (!visited.ContainsKey(neighbor)) {
          visited.Add(neighbor, false);
          worklist.Enqueue(neighbor);
        }
    }
    Console.WriteLine();
  }

  public void VisitDepthFirst() {
    Dictionary<Node<T>,bool> visited = new Dictionary<Node<T>,bool>();
    Stack<Node<T>> worklist = new Stack<Node<T>>();
    visited.Add(this, false);
    worklist.Push(this);
    // Invariant: every node in the worklist is also in the visited set
    while (worklist.Count != 0) {
      Node<T> node = worklist.Pop();
      Console.Write("{0} ", node.label);
      foreach (Node<T> neighbor in node.Neighbors)
        if (!visited.ContainsKey(neighbor)) {
          visited.Add(neighbor, false);
          worklist.Push(neighbor);
        }
    }
    Console.WriteLine();
  }
}

class TestGraphs {
  public static void Main(String[] args) {
    Node<int>
      leaf4 = new Node<int>(4), leaf5 = new Node<int>(5),
      leaf6 = new Node<int>(6), leaf7 = new Node<int>(7);
    Node<int>
      node2 = new Node<int>(2, new Node<int>[] { leaf4, leaf5 }),
      node3 = new Node<int>(3, new Node<int>[] { leaf6, leaf7 });
    Node<int> tree = new Node<int>(1, new Node<int>[] { node2, node3 });
    tree.VisitBreadthFirst();                           // 1 2 3 4 5 6 7
```

```
    tree.VisitDepthFirst();                              // 1 3 7 6 2 5 4
    Node<String>
      v1 = new Node<String>("one"),
      v2 = new Node<String>("two"),
      v3 = new Node<String>("three"),
      v4 = new Node<String>("four");
    v1.Neighbors = new Node<String>[] { v1, v2, v3 };
    v2.Neighbors = new Node<String>[] { v4 };
    v3.Neighbors = new Node<String>[] { v4 };
    v4.Neighbors = new Node<String>[] { v2 };
    v1.VisitBreadthFirst();                              // one two three four
    v1.VisitDepthFirst();                                // one three four two
  }
}
```

```
// Example 247 from page 211 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Compile with
//
//    csc /d:DEBUG Example247.cs

using System;
using SC = System.Collections;          // IEnumerator
using System.Collections.Generic;       // IEnumerable<T>, Queue<T>
using System.Diagnostics;               // Debug
using System.IO;                        // TextReader, TextWriter
using System.Text;                      // StringBuilder
using System.Text.RegularExpressions;   // Regex

class Example247 {
  public static void Main(String[] args) {
    if (args.Length != 2)
      Console.WriteLine("Usage: Example247 <textfile> <linewidth>\n");
    else {
      IEnumerator<String> words =
        new ReaderEnumerator(new StreamReader(args[0]));
      int lineWidth = int.Parse(args[1]);
      Format(words, lineWidth, Console.Out);
    }
  }

  // This method formats a sequence of words (Strings obtained from
  // an enumerator) into lines of text, padding the inter-word
  // gaps with extra spaces to obtain lines of length lineWidth, and
  // thus a straight right margin.
  //
  // There are the following exceptions:
  //  * if a word is longer than lineWidth, it is put on a line by
  //    itself (producing a line longer than lineWidth)
  //  * a single word may appear on a line by itself (producing a line
  //     shorter than lineWidth) if adding the next word to the line
  //     would make the line longer than lineWidth
  //  * the last line of the output is not padded with extra spaces.
  //
  // The algorithm for padding with extra spaces ensures that the
  // spaces are evenly distributed over inter-word gaps, using modulo
  // arithmetics.  An Assert method call asserts that the resulting
  // output line has the correct length unless the line contains only
  // a single word or is the last line of the output.

  public static void Format(IEnumerator<String> words, int lineWidth,
                            TextWriter tw) {
    lineWidth = Math.Max(0, lineWidth);
    WordList curLine = new WordList();
    bool moreWords = words.MoveNext();
    while (moreWords) {
      while (moreWords && curLine.Length < lineWidth) {
        String word = words.Current;
        if (word != null && word != "")
          curLine.AddLast(word);
        moreWords = words.MoveNext();
      }
      int wordCount = curLine.Count;
      if (wordCount > 0) {
        int extraSpaces = lineWidth - curLine.Length;
        if (wordCount > 1 && extraSpaces < 0) { // last word goes on next line
          int lastWordLength = curLine.GetLast.Length;
          extraSpaces += 1 + lastWordLength;
          wordCount -= 1;
        } else if (!moreWords)                  // last line, do not pad
          extraSpaces = 0;
        // Pad inter-word space with evenly distributed extra blanks
        int holes = wordCount - 1;
        int spaces = holes/2;
        StringBuilder sb = new StringBuilder();
        sb.Append(curLine.RemoveFirst());
        for (int i=1; i<wordCount; i++) {
          spaces += extraSpaces;
          appendBlanks(sb, 1 + spaces / holes);
          spaces %= holes;
          sb.Append(curLine.RemoveFirst());
        }
        String res = sb.ToString();
```

```
        Debug.Assert(res.Length==lineWidth || wordCount==1 || !moreWords);
        tw.WriteLine(res);
      }
    }
    tw.Flush();
  }

  private static void appendBlanks(StringBuilder sb, int count) {
    for (int i=0; i<count; i++)
      sb.Append(' ');
  }
}

// A word list with a fast length method, and invariant assertions

class WordList {
  private Queue<String> strings = new Queue<String>();
  // Invariant: length equals word lengths plus inter-word spaces
  private int length = -1;
  private String lastAdded = null;

  public int Length { get { return length; } }

  public int Count { get { return strings.Count; } }

  public void AddLast(String s) {
    lastAdded = s;
    strings.Enqueue(s);
    length += 1 + s.Length;
    Debug.Assert(length == computeLength() + strings.Count - 1);
  }

  public String RemoveFirst() {
    String res = strings.Dequeue();
    length -= 1 + res.Length;
    Debug.Assert(length == computeLength() + strings.Count - 1);
    return res;
  }

  public String GetLast {
    get { return lastAdded; }
  }

  private int computeLength() {  // For checking the invariant only
    int sum = 0;
    foreach (String s in strings)
      sum += s.Length;
    return sum;
  }
}

// A String-producing IEnumerator, created from a TextReader

class ReaderEnumerator : IEnumerator<String> {
  private static Regex delim = new Regex("[ \\t]+");
  private TextReader rd;
  private String[] thisLine = null;
  private int available = 0;

  public ReaderEnumerator(TextReader rd) {
    this.rd = rd;
  }

  public bool MoveNext() {
    available--;
    // If necessary, try to find some non-blank words
    String line;
    while (rd != null && available <= 0 && null != (line = rd.ReadLine())) {
      thisLine = delim.Split(line);
      available = thisLine.Length;
    }
    return available >= 1;
  }

  public String Current {
    get {
      if (available >= 1)
        return thisLine[thisLine.Length-available];
      else
        throw new InvalidOperationException();
    }
  }

  public void Dispose() {
    if (rd != null) {
      rd.Close();
      available = 0;
    }
    rd = null;
  }

  Object SC.IEnumerator.Current {
    get { return Current; }
  }

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

```
// Example 248 from page 211 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Namespaces and using

// Compile with
//    MS .Net:  csc /target:library Example201.cs
//    Mono:     mcs /target:library Example201.cs

using System;

namespace N1 {
  public class C11 { N3.C31 c31; }      // N1 depends on N3
  namespace N2 {
    public class C121 { }
  }
}
class C1 { }                            // Default accessibility: internal
namespace N1 {
  public struct S13 { }
}
namespace N1.N2 {
  internal class C122 { }
}

namespace N3 {
  class C31 { N1.C11 c11; }             // N3 depends on N1
}

class MyTest {
  public static void Main(String[] args) {
    N1.C11 c11;
    N1.N2.C121 c121;
    C1 c1;
    N1.S13 c13;
    N1.N2.C122 c122;
  }
}
```

```
// Example 249 from page 211 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Namespaces and using

// You must compile the example declaring N1 and N1.N2 to a .dll first.

// Then compile with
//    MS .Net: csc /reference:Example201.dll Example202.cs
//    Mono:    mcs /reference:Example201.dll Example202.cs

using System;
using N1;
using N1.N2;                    // using N2; does not suffice here

class MyTest {
  public static void Main(String[] args) {
    C11 c11;
    C121 c121;
    // C1 c1;                   // Inaccessible: internal to above example
    S13 c13;
    // C122 c122;               // Inaccessible: internal to above example
  }
}
```

```
// Example 250 from page 213 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

partial interface I {
  void M2(C.S n);
}

sealed partial class C : I {
  public void M1(S n) {
    if (n.x > 0)
      M2(n.Decr());
  }

  public partial struct S {
    public S(int x) { this.x = x; }
  }

  public static void Main() {
    C c = new C();
    c.M1(new S(5));
  }
}
```

```
// Example 251 from page 213 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

using System;

partial interface I {
  void M1(C.S n);
}

public partial class C {
  public partial struct S {
    public int x;
    public S Decr() { x--; return this; }
  }

  public void M2(S n) {
    Console.WriteLine("n.x={0} ", n.x);
    M1(n);
  }
}
```

```
// Example 252 from page 215 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Compile with
//
//    csc /d:DEBUG Example252.cs

using System;
using System.Diagnostics;

class Example252 {
  public static void Main(String[] args) {
    if (args.Length != 1)
      Console.WriteLine("Usage: Example252 <integer>\n");
    else {
      int x = int.Parse(args[0]);
      Console.WriteLine("Integer square root of " + x + " is " + Sqrt(x));
    }
  }

  // Modified for C# from C code on Paul Hsieh's square root page

  static int Sqrt(int x) {  // Algorithm by Borgerding, Hsieh, Ulery
    if (x < 0)
      throw new ArgumentOutOfRangeException("sqrt: negative argument");
    int temp, y = 0, b = 0x8000, bshft = 15, v = x;;
    do {
      if (v >= (temp = (y<<1)+b << bshft--)) {
        y += b; v -= temp;
      }
    } while ((b >>= 1) > 0);
    Debug.Assert((long)y * y <= x && (long)(y+1)*(y+1) > x);
    return y;
  }
}
```

```
// Example 253 from page 215 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Compile with
//
//    csc /d:DEBUG Example253.cs

using System;
using SC = System.Collections;          // IEnumerator
using System.Collections.Generic;       // IEnumerable<T>, Queue<T>
using System.Diagnostics;               // Debug
using System.IO;                        // TextReader, TextWriter
using System.Text;                      // StringBuilder
using System.Text.RegularExpressions;   // Regex

class Example253 {
  public static void Main(String[] args) {
    if (args.Length != 2)
      Console.WriteLine("Usage: Example253 <textfile> <linewidth>\n");
    else {
      IEnumerator<String> words =
        new ReaderEnumerator(new StreamReader(args[0]));
      int lineWidth = int.Parse(args[1]);
      Format(words, lineWidth, Console.Out);
    }
  }

  // This method formats a sequence of words (Strings obtained from
  // an enumerator) into lines of text, padding the inter-word
  // gaps with extra spaces to obtain lines of length lineWidth, and
  // thus a straight right margin.
  //
  // There are the following exceptions:
  //  * if a word is longer than lineWidth, it is put on a line by
  //    itself (producing a line longer than lineWidth)
  //  * a single word may appear on a line by itself (producing a line
  //     shorter than lineWidth) if adding the next word to the line
  //     would make the line longer than lineWidth
  //  * the last line of the output is not padded with extra spaces.
  //
  // The algorithm for padding with extra spaces ensures that the
  // spaces are evenly distributed over inter-word gaps, using modulo
  // arithmetics.  An Assert method call asserts that the resulting
  // output line has the correct length unless the line contains only
  // a single word or is the last line of the output.

  public static void Format(IEnumerator<String> words, int lineWidth,
                            TextWriter tw) {
    lineWidth = Math.Max(0, lineWidth);
    WordList curLine = new WordList();
    bool moreWords = words.MoveNext();
    while (moreWords) {
      while (moreWords && curLine.Length < lineWidth) {
        String word = words.Current;
        if (word != null && word != "")
          curLine.AddLast(word);
        moreWords = words.MoveNext();
      }
      int wordCount = curLine.Count;
      if (wordCount > 0) {
        int extraSpaces = lineWidth - curLine.Length;
        if (wordCount > 1 && extraSpaces < 0) { // last word goes on next line
          int lastWordLength = curLine.GetLast.Length;
          extraSpaces += 1 + lastWordLength;
          wordCount -= 1;
        } else if (!moreWords)                  // last line, do not pad
          extraSpaces = 0;
        // Pad inter-word space with evenly distributed extra blanks
        int holes = wordCount - 1;
        int spaces = holes/2;
        StringBuilder sb = new StringBuilder();
        sb.Append(curLine.RemoveFirst());
        for (int i=1; i<wordCount; i++) {
          spaces += extraSpaces;
          appendBlanks(sb, 1 + spaces / holes);
          spaces %= holes;
          sb.Append(curLine.RemoveFirst());
        }
        String res = sb.ToString();
```

```
        Debug.Assert(res.Length==lineWidth || wordCount==1 || !moreWords);
        tw.WriteLine(res);
      }
    }
    tw.Flush();
  }

  private static void appendBlanks(StringBuilder sb, int count) {
    for (int i=0; i<count; i++)
      sb.Append(' ');
  }
}

// A word list with a fast length method, and invariant assertions

class WordList {
  private Queue<String> strings = new Queue<String>();
  // Invariant: length equals word lengths plus inter-word spaces
  private int length = -1;
  private String lastAdded = null;

  public int Length { get { return length; } }

  public int Count { get { return strings.Count; } }

  public void AddLast(String s) {
    lastAdded = s;
    strings.Enqueue(s);
    length += 1 + s.Length;
    Debug.Assert(length == computeLength() + strings.Count - 1);
  }

  public String RemoveFirst() {
    String res = strings.Dequeue();
    length -= 1 + res.Length;
    Debug.Assert(length == computeLength() + strings.Count - 1);
    return res;
  }

  public String GetLast {
    get { return lastAdded; }
  }

  private int computeLength() {  // For checking the invariant only
    int sum = 0;
    foreach (String s in strings)
      sum += s.Length;
    return sum;
  }
}

// A String-producing IEnumerator, created from a TextReader

class ReaderEnumerator : IEnumerator<String> {
  private static Regex delim = new Regex("[ \\t]+");
  private TextReader rd;
  private String[] thisLine = null;
  private int available = 0;

  public ReaderEnumerator(TextReader rd) {
    this.rd = rd;
  }

  public bool MoveNext() {
    available--;
    // If necessary, try to find some non-blank words
    String line;
    while (rd != null && available <= 0 && null != (line = rd.ReadLine())) {
      thisLine = delim.Split(line);
      available = thisLine.Length;
    }
    return available >= 1;
  }

  public String Current {
    get {
      if (available >= 1)
        return thisLine[thisLine.Length-available];
      else
        throw new InvalidOperationException();
    }
  }

  public void Dispose() {
    if (rd != null) {
      rd.Close();
      available = 0;
    }
    rd = null;
  }

  Object SC.IEnumerator.Current {
    get { return Current; }
  }

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

```
// Example 254 from page 217 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Serialize a data structure to a file "objects".  Strangely,
// SoapFormatter and

using System;
using System.IO;                                    // File, FileMode, Stream
using System.Runtime.Serialization;                 // IFormatter
using System.Runtime.Serialization.Formatters.Soap;    // SoapFormatter
using System.Runtime.Serialization.Formatters.Binary;  // BinaryFormatter

[Serializable()]
class SC { public int ci; }

[Serializable()]
class SO {
  public int i;
  public SC c;
 [NonSerialized()] public String s;

  public SO(int i, SC c) { this.i = i; this.c = c; s = i.ToString(); }
  public void CPrint() {
    Console.WriteLine("i{0}c{1}({2})", i, c.ci, s); }
}

class SerializeUnshared {
  public static void Main(String[] args) {
    IFormatter fmtr = new SoapFormatter();
    // IFormatter fmtr = new BinaryFormatter();        // Alternative
    if (!File.Exists("objects")) {
      Console.WriteLine("Creating objects and writing them to file:");
      SC c = new SC();
      SO o1 = new SO(1, c), o2 = new SO(2, c);
      Console.WriteLine("The SC object is shared between o1 and o2:");
      o1.c.ci = 3; o2.c.ci = 4;                 // Update the shared c twice
      o1.CPrint(); o2.CPrint();                 // Prints i1c4 i2c4
      // Open file and serialize objects to it:
      Stream strm = File.Open("objects", FileMode.Create);
      fmtr.Serialize(strm, o1); fmtr.Serialize(strm, o2);
      strm.Close();
      Console.WriteLine("\nRun the example again to read objects from file");
    } else {
      Console.WriteLine("Reading objects from file (unshared c):");
      Stream strm = File.Open("objects", FileMode.Open);
      SO o1i = (SO)(fmtr.Deserialize(strm)), o2i = (SO)(fmtr.Deserialize(strm));
      strm.Close();
      o1i.CPrint(); o2i.CPrint();               // Prints i1c4() i2c4()
      Console.WriteLine("The sharing of the SC object is lost:");
      o1i.c.ci = 5; o2i.c.ci = 6;               // Update two different c's
      o1i.CPrint(); o2i.CPrint();               // Prints i1c5() i2c6()
      File.Delete("objects");
    }
    Console.WriteLine();
  }
}
```

```
// Example 255 from page 217 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// Attribute arguments are evaluated at compiletime; they must be
// constant expressions of a limited repertoire of types.

// Attribute constructors are executed at runtime, by applying them to
// the pre-evaluated argument values.  This happens at every call to
// GetCustomAttributes (in MS .Net 2.0 as well as Mono 1.0).

using System;                   // Attribute, AttributeUsage, AttributeTargets
using System.Reflection;        // MemberInfo

public enum Month {
  Jan=1, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
}

[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method,
                AllowMultiple = true)]
class AuthorAttribute : Attribute {
  public readonly String name;
  public readonly Month mm;

  public AuthorAttribute(String name, Month mm) {
    this.name = name; this.mm = mm;
    Console.WriteLine("Creating AuthorAttribute: {0}", this);
  }

  public override String ToString() {
    return String.Format("{0} ({1})", name, mm);
  }
}

class TestAttributes {
 [Author("Donald", Month.May)]
  public void MyMethod1() { }

 [Author("Andrzej", Month.Jul)]
 [Author("Andreas", Month.Mar)]
  public void MyMethod2() { }

  public static void Main(String[] args) {
    Type ty = typeof(TestAttributes);
    foreach (MemberInfo mif in ty.GetMembers()) {
      if (mif.Name.StartsWith("MyMethod")) {
        Console.WriteLine("\nGetting attributes of {0}:", mif.Name);
        Object[] attrs = mif.GetCustomAttributes(false);
        Console.WriteLine("\nThe attributes of {0} are:", mif.Name);
        foreach (Attribute attr in attrs)
          Console.Write("{0} ", attr);
        Console.WriteLine();
        Console.WriteLine("\nGetting attributes of {0} again:", mif.Name);
        mif.GetCustomAttributes(false);
      }
    }
  }
}
```

```
// Example 256 from page 219 of C# Precisely, 2nd ed. (MIT Press 2012)
// Authors: Peter Sestoft (sestoft@itu.dk) and Henrik I. Hansen

// This is based to a large extent on the generic LinkedList example.

using System;
using System.IO;                        // TextWriter
using System.Collections.Generic;       // IEnumerable<T>, IEnumerator<T>
using SC = System.Collections;          // IEnumerable, IEnumerator
using System.Linq;                      // For Linq query syntax

public interface IMyList<T> : IEnumerable<T>, IEquatable<IMyList<T>> {
  int Count { get; }                    // Number of elements
  T this[int i] { get; set; }           // Get or set element at index i
  void Add(T item);                     // Add element at end
  void Insert(int i, T item);           // Insert element at index i
  void RemoveAt(int i);                 // Remove element at index i
  IMyList<U> Map<U>(Func<T,U> f);       // Map f over all elements
  void Apply(Action<T> act);            // Apply act to all elements
}

public class LinkedList<T> : IMyList<T> {
  protected int size;                   // Number of elements in the list
  protected Node first, last;           // Invariant: first==null iff last==null

  protected class Node {                // Static member class
    public Node prev, next;
    public T item;

    public Node(T item) {
      this.item = item;
    }

    public Node(T item, Node prev, Node next) {
      this.item = item; this.prev = prev; this.next = next;
    }
  }

  public LinkedList() {
    first = last = null;
    size = 0;
  }

  public int Count { get { return size; } }     // Property with get accessor

  public T this[int i] {                        // Indexer with get and set accessors
    get { return get(i).item; }
    set { get(i).item = value; }
  }

  private Node get(int n) {
    if (n < 0 || n >= size)
      throw new IndexOutOfRangeException();
    else if (n < size/2) {              // Closer to front
      Node node = first;
      for (int i=0; i<n; i++)
        node = node.next;
      return node;
    } else {                            // Closer to end
      Node node = last;
      for (int i=size-1; i>n; i--)
        node = node.prev;
      return node;
    }
  }

  public void Add(T item) {                     // Enables collection initializer
    Insert(size, item);
  }

  public void Insert(int i, T item) {
    if (i == 0) {
      if (first == null) // and thus last == null
        first = last = new Node(item);
      else {
        Node tmp = new Node(item, null, first);
        first.prev = tmp;
        first = tmp;
      }
      size++;
    } else if (i == size) {
      if (last == null) // and thus first = null
        first = last = new Node(item);
      else {
        Node tmp = new Node(item, last, null);
        last.next = tmp;
        last = tmp;
      }
      size++;
    } else {
      Node node = get(i);
      // assert node.prev != null;
      Node newnode = new Node(item, node.prev, node);
      node.prev.next = newnode;
      node.prev = newnode;
      size++;
    }
  }

  public void RemoveAt(int i) {
    Node node = get(i);
    if (node.prev == null)
      first = node.next;
    else
      node.prev.next = node.next;
    if (node.next == null)
      last = node.prev;
    else
      node.next.prev = node.prev;
    size--;
  }

  public override bool Equals(Object that) {
    return Equals(that as IMyList<T>);                  // Exact runtime type test
  }

  public bool Equals(IMyList<T> that) {
    if (this == that)
      return true;
    if (that == null || this.Count != that.Count)
      return false;
    Node thisnode = this.first;
    IEnumerator<T> thatenm = that.GetEnumerator();
    while (thisnode != null) {
      if (!thatenm.MoveNext())
        throw new ApplicationException("Impossible: LinkedList<T>.Equals");
      // assert MoveNext() was true (because of the above size test)
      if (!thisnode.item.Equals(thatenm.Current))
        return false;
      thisnode = thisnode.next;
    }
    // assert !MoveNext(); // because of the size test
    return true;
  }

  public override int GetHashCode() {
    int hash = 0;
    foreach (T x in this)
      hash ^= x.GetHashCode();
    return hash;
  }

  public IEnumerator<T> GetEnumerator() {               // IEnumerable<T> via iterato
r block
    for (Node curr=first; curr!=null; curr=curr.next)
      yield return curr.item;
  }

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

  // Explicit conversion from array of T
  // (Note: <T> is part of the target type name, not not a method type parameter)

  public static explicit operator LinkedList<T>(T[] arr) {
    var res = new LinkedList<T>();
    foreach (T x in arr)
```

```
      res.Add(x);
    return res;
  }

  // Overloaded operator

  public static LinkedList<T> operator +(LinkedList<T> xs1, LinkedList<T> xs2) {
    var res = new LinkedList<T>();
    foreach (T x in xs1)
      res.Add(x);
    foreach (T x in xs2)
      res.Add(x);
    return res;
  }

  // Methods with Func and Action arguments

  public IMyList<U> Map<U>(Func<T,U> f) {
    var res = new LinkedList<U>();
    foreach (T x in this)
      res.Add(f(x));
    return res;
  }

  public static LinkedList<T> Tabulate(Func<int,T> f, int from, int to) {
    var res = new LinkedList<T>();
    for (int i=from; i<to; i++)
      res.Add(f(i));
    return res;
  }

  public void Apply(Action<T> act) {                // Taking delegate argument
    foreach (T x in this)
      act(x);
  }
}

public class TestLinkedList {
  static void Main(String[] args) {
    LinkedList<int> xs = new LinkedList<int> { 0, 2, 4, 6, 8 };                // (1
)
    Console.WriteLine(xs.Count + " " + xs[2]);                                 // (2
)
    xs[2] = 102;                                                               // (3
)
    foreach (int k in xs)
      Console.WriteLine(k);
    LinkedList<int> ys = (LinkedList<int>)(new int[] { 1, 2, 3, 4, 5 });       // (4
)
    LinkedList<int> zs = xs + ys;                                              // (5
)
    zs.Apply(delegate(int x) { Console.Write(x + " "); });                     // (6
)
    Console.WriteLine();
    var vs = LinkedList<double>.Tabulate(x => 1.0/x, 1, 5);                    // (7
)
    foreach (var g in from z in zs group z by z/10)                            // (8
)
      Console.WriteLine("{0} to {1}: {2} items", g.Key*10, g.Key*10+9, g.Count());
    LinkedList<dynamic> ds = new LinkedList<dynamic>();                        // (9
)
    ds.Add(5); ds.Add(0.25); ds.Add(true); ds.Add("foo");                      // (9
)
    double d = ds[1];                                                          // (9
)
    Console.WriteLine(ds[2] ? ds[3].Length : false);                           // (9
)
    Console.WriteLine(xs.Equals(xs));
    Console.WriteLine(xs.Equals(ys));
  }
}
```