**Quantifier Operations**

int [] numbers = { 1, 2, 3, 4, 5};

Console.WriteLine("Are all numbers > 0? " +

numbers.All(x => x > 0));

Console.WriteLine("Are all numbers odd? " +

numbers.All(x => x%2 == 1));

// once the iterator finds a match, it doesn't check other elements

Console.WriteLine("Any number less than two? " + numbers.Any(x => x < 2));

// also, Any() is a way to check whether a collection is empty

Console.WriteLine(new int[]{/\*42\*/}.Any());

Console.WriteLine("Contains 5? " + numbers.Contains(5)); // not a lambda here!

Console.WriteLine("Number of odd elements: " + numbers.Count(x => x % 2 == 1));

Console.WriteLine("Total no. of elements: " + numbers.Count()); // beware!

**Partition Data**

var numbers = new[]{ 3, 3, 2, 2, 1, 1, 2, 2, 3, 3 };

Console.WriteLine(numbers.Skip(2).Take(6));

Console.WriteLine(new int[]{}.Take(2));

Console.WriteLine(numbers.SkipWhile(i => i == 3));

Console.WriteLine(numbers.TakeWhile(i => i > 1));

**Exercise 3**

public static int LengthOfPositive(IEnumerable<int> input)

{

var skipStartingNegatives = input.SkipWhile(i => i < 0);

var getNextPositives = skipStartingNegatives.TakeWhile(i => i > 0);

return getNextPositives.Count();

}

**int cnt = Exercise3.LengthOfPositive(new int[] { -3, -1, 3, 7, 1, -3, 7 });**

**Equality**

var arr1 = new[]{1,2,3};

var arr2 = new[]{1,2,3};

// LINQ

Console.WriteLine(arr1.SequenceEqual(arr2));

var list1 = new List<int>{1,2,3};

Console.WriteLine(arr1.SequenceEqual(list1));

**Element Operations**

// first: returns first element satisfying a predicate or throws

var numbers = new List<int>{1,2,3};

Console.WriteLine(numbers.First());

Console.WriteLine(numbers.First(x => x > 2)); // try 10

Console.WriteLine(numbers.FirstOrDefault(x => x > 10)); // string - null

// same for last value

Console.WriteLine(numbers.Last());

Console.WriteLine(numbers.Last(x => x < 3));

// single: ensures that there's only one value, otherwise throws

// throws because non-singular

//Console.WriteLine(numbers.Single());

// also throws

//Console.WriteLine(numbers.SingleOrDefault());

// doesn't throw only if sequence is empty

Console.WriteLine("Empty array: " + new int[]{}.SingleOrDefault());

Console.WriteLine("Item at position 1: " + numbers.ElementAt(1));

Console.WriteLine("Item at position 4: " + numbers.ElementAtOrDefault(4));

**Concatenation**

var integralTypes = new[]{ typeof(int), typeof(short) };

var fpTypes = new[]{ typeof(float), typeof(double) };

Console.WriteLine(

integralTypes.Concat(fpTypes));

void Main()

{

var integralTypes = new[]{ typeof(int), typeof(short) };

var fpTypes = new[]{ typeof(float), typeof(double) };

Console.WriteLine(

integralTypes.Concat(fpTypes)

.Prepend(typeof(byte))

);

}

// Define other methods and classes here

static class ExtensionMethods

{

public static IEnumerable<T> Prepend<T>(

this IEnumerable<T> values, T value)

{

yield return value;

foreach (var item in values) {

yield return item;

}

}

}

**Aggregation Operations**

var numbers = Enumerable.Range(1,10);

Console.WriteLine("We have " + numbers.Count() + " elements");

//Console.WriteLine(numbers);

// 1 2 3 4 5 ...

// 1 2 -> 3

// 3 3 -> 6

Console.WriteLine("Sum = " +

numbers.Sum());

// 1 1 -> 1

// 1 2 -> 2

// 2 3 -> 6

Console.WriteLine("Sum = " +

numbers.Aggregate((p,q) => p + q));

Console.WriteLine("product = " +

numbers.Aggregate(1, (p,q) => p \* q));

Console.WriteLine("Average = " +

numbers.Average());

var words = new[] { "one", "two", "three" };

Console.WriteLine(words.Aggregate("hello", (p,x) => p + "," + x));

// Rectangle.Union(r1, r2)

var rectangles = new[] {

new Rectangle(0,0,20,20),

new Rectangle(20,20,60,60),

new Rectangle(80,80,20,20)

};

Console.WriteLine(rectangles.Aggregate(Rectangle.Union));

**Exercise4**

public static int Poly(int x, IEnumerable<int> coeffs)

{

int seed = 0;

var su = coeffs.Reverse().Aggregate(seed, (p, q) => p + q \* (int)Math.Pow(x,seed++) );

return su;

}