Odpovědi pište na zvláštní odpovědní list s vaším jménem a fotografií. Pokud budete odevzdávat více než jeden list s řešením, tak se na 2. a další listy nezapomeňte podepsat. Do zápatí všech listů vždy napište i/N (kde i je číslo listu, N je celkový počet odevzdaných listů).

### Otázka č. 1

Předpokládejte následující kód:

```
public static IEnumerable<int> Range(
   int from, int to
) {
   if (to < from) {
      throw new ArgumentException(
        $"'{nameof(from)}' has to be less"
        + $" or equal than '{nameof(to)}'."
      );
   }
   for (int i = from; i <= to; i++) {
      yield return i;
}  }
```

Byla by výše uvedená metoda vhodnou implementací standardní metody `Enumerable.Range`? Detailně vysvětlete proč. Pokud odpovíte ne, tak napište vhodnější implementaci.

**[1,5 bodu]**

---

### Otázka č. 2

Naprogramujte v jazyce C# thread-safe třídu `LfStack<T>` implementující datovou strukturu lock-free zásobníku jako jednosměrně vázaný spojový seznam (ve své implementaci nevyužívejte žádné standardní .NET kolekce). Předpokládejte standardní .NET memory model.

**[1,5 bod]**

---

Za předpokladu vaší implementace třídy `LfStack<T>` je následující metoda thread-safe? Vysvětlete.

```
public void Move<T>(LfStack<T> s, LfStack<T> d) {
   d.Push(s.Pop());
}
```

**[1 bod]**

---

### Otázka č. 3

Předpokládejte následující program:

```
public class Prg3 {
   public static void Main() {
      var actions = new List<Action>();

      for (int a = 1; a <= 2; a++) {
         int b = 1;
         for (int c = 1; c <= 2; c++) {
            int d = 1;
            actions.Add(() => Console.WriteLine(
               $"{++a}, {++b}, {++c}, {++d}"
            ));
      }  }

      foreach (var a in actions) a();
} }
```

Rozhodněte, co přesně program po spuštění vypíše na standardní výstup, a detailně vysvětlete proč.

**[1,5 bodu]**

---

### Otázka č. 4

Předpokládejte následující program:

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

public class X {
   public List<int> Where(Predicate<int> p) {
      var list = new List<int>();
      for (int i = 0; i < 10; i++) {
         if (p(i)) list.Add(i);
      }
      return list;
   }
}

public class Prg4 {
   public static void Main() {
      var x = new X();
      var odd = from b in (
                   from a in x
                   where a % 2 != 0
                   select a
                ).AsParallel()
                where b > 5
                select b * 2;

      foreach (var b in odd) {
         Console.WriteLine(b);
      }
   }
}
```

Rozhodněte, jaký bude typ proměnné `odd`. Popište datovou strukturu, která bude v proměnné `odd` „uložena“, resp. na kterou proměnná `odd` ukazuje, je-li referenčního typu.

**[1 bod]**

---

Pokud bychom celý `foreach` cyklus v metodě `Main` nahradili níže uvedeným kódem, změnilo by se nějak chování programu? Vysvětlete proč, resp. případně jak.

```
odd.ForAll(b => Console.WriteLine(b));
```

**[1 bod]**

---

### Otázka č. 5

Metadata assembly `Library.dll` se v nástroji `ildasm` zobrazí následujícím způsobem:


Dále lze zjistit, že implementace metody `A.m` je v CIL (MSIL) assembleru následující:

```
.method public hidebysig static int32
m() cil managed
{
   .maxstack 8
   ldsfld     int32 A::v
   ldc.i4.1
   add
   dup
   stsfld     int32 A::v
   ret
}
```

Zapište v C# deklaraci a tělo takové metody `A.m` (která se bude chovat stejně jako výše uvedený kód).
**[1 bod]**

---

### Otázka č. 6

Předpokládejte, že chceme navrhnout třídu `SuperSortedSet` s následujícím prototypem:

```
class SuperSortedSet<T> : IEnumerable<T>
```

tak, abychom ji mohli použít v následujícím kontextu (třída reprezentuje prostý seznam hodnot setříděný dle klíče předaného jako parametr konstruktoru):

```
class Person {
   public string FirstName { get; set; }
   public string LastName { get; set; }

   public override string ToString() {
      return
         $"[{nameof(FirstName)} = {FirstName},"
         + $"[{nameof(LastName)} = {LastName}]";
  }
}

public class Prg6 {
   public static void Main() {
      var personSet = new SuperSortedSet<Person>(
         p => p.LastName
      );

      personSet.Add(new Person {
         FirstName = "a", LastName = "y" });
      personSet.Add(new Person {
         FirstName = "c", LastName = "x" });
      personSet.Add(new Person {
         FirstName = "b", LastName = "z" });
      personSet.Add(new Person {
         FirstName = "d", LastName = "w" });


      var q1 = personSet.OrderBy(
               p => p.LastName).Take(3);
      var q2 = personSet.OrderBy(
               p => p.FirstName).Take(3);

      foreach (var p in q1) Console.WriteLine(p);
      foreach (var p in q2) Console.WriteLine(p);
   }
}
```

Naším cílem je, aby v případě 1. LINQ dotazu v `q1` třída `SuperSortedSet` „sama poznala“, že vyžadujeme třízení dle stejného klíče a seznam již znovu nepřetřiďovala. Napište minimalistickou implementaci třídy `SuperSortedSet` tak, aby šla přeložit minimálně ve výše uvedeném kontextu.
**[1,5 bodu]**