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

Rozhodněte a zdůvodněte, co níže uvedený program, který se bez chyb přeloží, vypíše na výstup:

```
class Prg1 {
   static void Log(
      object s, UnhandledExceptionEventArgs a) {
      Console.Write("-un-");
   }

   public static void Main() {
      AppDomain.CurrentDomain.UnhandledException
         += Log;
      try {
         Console.Write("-pre-");
         throw new Exception();
         Console.Write("-post-");
      } finally {
         Console.Write("-fin-");
} } }
```

[1 bod]

Společná část pro otázky označené X

Předpokládejte, že programujeme hru, ve které bude ústředním motivem vyrábění a objevování nový věcí. Nyní se soustředíme na část, která načítá a zpracovává dva základní druhy receptů (druh `Item` a druh `Eureka`) načítaných z následujícího formátu (kde např. `1 Iron * Fire` znamená jedna ingredience typu `Iron` a nekonečno ingrediencí typu `Fire`):

```
Item IronIngot
Ingredients 1 Iron * Fire
Locked No
Output 3
---
Eureka Wheel
Ingredients 1 Plank 5 Stick
Locked No
Unlocks Cart Mill
---
```

Otázka č. 2 (X)

V naší aplikaci reprezentujeme množství nějaké ingredience v receptu pomocí následující třídy:

```
class Quantity {
   public string ItemName;
   public bool IsInfinite;
   public int Count;

   public override string ToString() {
      return (IsInfinite ? "*" : Count.ToString())
             + " " + ItemName;
} }
```

(A) Upravte třídu `Quantity` tak, aby byla immutable.
[1 bod]

(B) Upravte implementaci `Quantity` tak, aby standardní .NET typy správně chápaly dvě různé instance `Quantity` jako ekvivalentní, pokud mají stejné jméno `ItemName` a stejný počet.
[1 bod]

Otázka č. 3 (X)

Jako součást naší aplikace jsme převzali následující třídu popisující různé druhy receptů (momentálně „Item“ a „Eureka“ rozlišené obsahem datové položky `IsEureka`):

```
class Recipe {
   public bool IsEureka;
   public List<Quantity> Ingredients
      = new List<Quantity>();
   public bool IsLocked;
   public Quantity Output;
   public List<string> Unlocks
      = new List<string>();

   public override string ToString() {
      var result = $"Ing {Ingredients.Stringify()}"
                 + $" Locked {IsLocked} ";
      if (IsEureka) { result += "Unlocks "
                              + Unlocks.Stringify();
      } else { result += "Output " + Output; }
      return result;
} }
```

(A) Výše uvedený návrh se nám ale nelíbí, protože se nám zdá nedostatečně udržovatelný a rozšiřitelný do budoucna, až budeme přidávat další druhy receptů, které budou mít dle svého druhu další specifické vlastnosti. Navrhněte jak co nejlépe uvedený kód refaktorovat, a vysvětlete, proč je vaše řešení vhodnější.
[1 bod]

(B) Napište a vysvětlete, jak by bylo možné v uvedeném kódu zařídit uvedenou syntaktickou možnost zavolání metod `Ingredients.Stringify()` a `Unlocks.Stringify()`, přestože třída `List<T>` žádnou takovou metodu nemá. Bylo by možné stejným způsobem dosáhnout reimplementace standardní metody `.ToString()`, kterou již `List<T>` samozřejmě má?
[1 bod]

(C) Napište implementaci výše uvedené metody `.Stringify()` pro typ `List<T>` tak, aby vracela textový řetězec ve stejném formátu jako je ve vstupním souboru. Tedy pro `Ingredients.Stringify()` bude výstup např.:

```
1 Iron * Fire
```

Pro `Unlocks.Stringify()` bude výstup např.:

```
Cart Mill
```

Předpokládejte, že v programu tuto metodu budeme na instancích `List<T>` volat velmi často s relativně velkým počtem prvků. Je váš kód jediná alternativa, jak takovou implementaci napsat? Pokud ne, tak vysvětlete, proč je zrovna vaše implementace nejvhodnější.
[1 bod]

Otázka č. 4

Předpokládejte níže uvedený program, který se bez chyb přeloží.

```
using C = System.Console;

class W4 {
   public virtual void m(double t) { C.Write(1); }
}

class X4 : W4 {
   public virtual void m<T>(T t) { C.Write(2); }
}

class Y4 : X4 {
   public override void m<T>(T t) { C.Write(3); }
   public void m(float t) { C.Write(4); }
}

class Z4 : Y4 {
   public sealed override void m<T>(T t) {
      C.Write(5);
   }
   public override void m(double t) {
      C.Write(6);
   }
}

class Prg4 {
   public static void Main() {
      X4 x = new Z4();
      x.m(3.14);
   }
}
```

(A) Co vypíše uvedený program na standardní výstup? Detailně vysvětlete proč.
[1 bod]

(B) Vysvětlete, jaký je význam kombinace klíčových slov `sealed override`, a ukažte a vysvětlete nějaký praktický příklad toho, kde je použití `sealed override` vhodné. Vysvětlete proč.
[1 bod]

Otázka č. 5

Následující program se bez chyb přeloží. Co po spuštění vypíše na standardní výstup? Detailně vysvětlete proč.

```
class A<T> {
    public class B : A<long> {
        public void f() {
            Console.WriteLine(
                typeof(T).ToString());
        }
        public class C : A<long>.B { }
}   }

class Prg5 { static void Main() {
    var c = new A<float>.B.C();
    c.f();
}   }
```

[1 bod]

Otázka č. 6

Předpokládejte následující program, který se bez chyb přeloží:

```
using C = System.Console;

interface IW6 {
   void m1(); void m2(); void m3();
}

abstract class X6 {
   public void m1() { C.WriteLine(11); }
   public abstract void m2();
   public virtual void m3() { C.WriteLine(13); }
}

class Y6 : X6, IW6 {
   public virtual void m1() { C.WriteLine(21); }
   public override void m2() { C.WriteLine(22); }
}

class Z6 : Y6 {
   public override void m1() { C.WriteLine(31); }
   public virtual void m2() { C.WriteLine(32); }
   public override void m3() { C.WriteLine(33); }
}

class A6 : Z6 {
   public void m3() { C.WriteLine(43); }
}

class Prg6 {
   public static void Main() {
      IW6 iw = new A6();
      iw.m1();
      iw.m2();
      iw.m3();
   }
}
```

Detailně vysvětlete, co a proč vypíše po svém spuštění.
[1 bod]