```
using System.Numerics;

namespace ExamPrep;

/*
interface I1
{
    public int X { get; }
}

class A1 : I1
{
    public virtual int X { get; set; } = 10;
}

class B1 : A1
{
    public override int X => 20;
    public int Special => base.X + 1;
}

class C1 : B1
{
    public override int X { get; set; }
}
*/
/*
class A6
{
    public int X { get; }

    public A6()
    {
        X = f();
    }

    public virtual int f() => 4;
}

class B6 : A6
{
    private int _y = 10;
    public override int f() => _y + 1000;
}

class C6 : B6
{
    public new virtual int f() => 20;
}

class D6 : C6
{
    public sealed override int f() => 2;
}
class Program
```

```
{
    static void Main(string[] args)
    {
        var d = new D6();
        Console.WriteLine(m1(d));
        Console.WriteLine(d.X);
    }

    public static int m1(A6 a) => m2((C6)a);

    public static int m2(C6 c) => c.f();
}
*/

/*
interface I1
{
    public char f();
}

class A1 : I1
{
    public virtual char f() => 'A';
}

class B1 : A1
{
    public override char f() => 'B';
}

class C1 : B1
{
    public virtual char f() => (char)(base.f() + 10);
}

class D1 : C1, I1 {}

class E1 : D1
{
    public new char f() => (char)(base.f() + 5);
}

class Program
{
    static void Main(string[] args)
    {
        E1? a1 = new E1();
        Console.WriteLine(a1.f());

        I1? i1 = a1;
        Console.WriteLine(i1.f());

        Console.WriteLine(a1 is B1);
        a1 = null;
        Console.WriteLine(a1 is A1);
```

```
    }

}
*/
/*
interface I5
{
    public int X { get; set; }
}

struct S(int a, int b, int c) : I5
{
    public int X { get; set; } = 5;
    public int Y { get; } = 5;

    public int f()
    {
        return a + b + c;
    }
}

class Program
{
    private static void Update(I5 i)
    {
        i.X *= 10;
    }
    static void Main(string[] args)
    {
        S s1 = new S(11, 12, 13);
        I5 i1 = s1; // boxed, so it has the normal overhead + only
        Update(i1);
        Console.WriteLine(s1.X);
        Console.WriteLine(i1.X);

        S[] sa = new S[3];
        Console.WriteLine(sa[0].X);


    }
}
*/
/*
class Program
{
    enum  E
    {
        One = 1,
        Four = 4,
        Five
    }
    static void Main(string[] args)
    {
        E e1 = E.One;
        Console.WriteLine(e1);
```

```
        e1++;
        Console.WriteLine((e1));

        E e = E.Five;
        Console.WriteLine((int)e);
    }
}
*/
/*
class Prg8
{
    public static void Main()
    {
        var items = new List<int>();
        m1(items);
        Console.WriteLine(items.Count);

        m2(ref items);
        Console.WriteLine(items.Count);
    }

    static void m1(List<int> ints)  // the reference in items is copied to the variable ints
    {
        ints.Add(6);
        ints = null!;
    }

    static void m2(ref List<int> ints)  // modifies the original variable itself
    {
        ints.Add(7);
        ints = null!;
    }
}
*/
/*
interface I1
{
    int f();
}

class A1
{
    public int f()
    {
        return 4;
    }
}

class B1 : A1, I1
{
    public int f(int x) => x * 10;
}

class C1 : B1
{
```

```
    public new int f()
    {
        return 8;
    }
}

class Program
{
    static void Main()
    {
        I1 i1 = new C1();
        Console.WriteLine(i1.f());

        A1 a1 = (A1)i1;
        Console.WriteLine(a1.f());

    }
}
*/
/*
struct S3
{
    public int X;
    public int Y => X + 10;

    public void QuadrupleX()
    {
        X *= 4;
    }
}

class Program
{
    static void Main()
    {
        S3 s1 = new S3 { X = 10 };
        s1.QuadrupleX();
        Console.WriteLine(s1.Y);

        // Nullable<T>.Value is a property, hence it returns a copy of the struct and does not modify the
        // original one. The output will be 10.
        S3? s2 = new S3() { X = 10 };
        if (s2 is not null)
        {
            s2.Value.QuadrupleX();
            Console.WriteLine(s2.Value.X);
        }
    }
}
*/
/*
interface I1
{
    int f();
}
```

```
class A1
{
    public int f()
    {
        return 1;
    }
}

class B1 : A1, I1
{
    public int f(int x) => x * 2;
}

class C1 : B1
{
    public new int f()
    {
        return 3;
    }
}

class Program
{
    static void Main()
    {
        I1 i1 = new C1();
        Console.WriteLine(i1.f());

        A1 a1 = (A1)i1;
        Console.WriteLine(a1.f());
    }
}
*/
/*
class Program
{
    enum DayOfWeek {
        Monday, Tuesday, Wednesday, Thursday,
        Friday, Saturday, Sunday
    };

    static void Main()
    {
        var day = DayOfWeek.Sunday;
        day++;
        Console.WriteLine(day);
    }
}
*/
/*
interface I6
{
    public char f();
}
```

```
class A6
{
    public virtual char f() => ’A’;
}

class B6 : A6, I6   // !!! Interface looks at the signature, not implementation
{
    public override char f() => ’B’;
}

class C6 : B6
{
    public virtual char f() => ’C’;
}

class D6 : C6
{
    public override char f() =>
        (char)(base.f() + 1);   // base forcefully calls the father object’s method (think about endless
recursion)
}

class E6 : D6
{
    public override char f() => (char)(base.f() + 1);
}

class Program
{
    static void Main()
    {
        C6 c6 = new E6();
        Console.WriteLine(c6.f());

        I6 i6 = new C6();
        Console.WriteLine(i6.f());
    }
}
*/
/*
class Prg6 {
    public ref int m(ref int a) {
        var x = 5;
        ref int r1 = ref x;
        return ref m(ref r1);
    }
}
*/
/*
public class X
{
    public void m(int i)
    {
        Console.WriteLine("X.f(int)");
```

```
    }
}

public class Y : X
{
    public void m(float f)
    {
        Console.WriteLine("Y.f(float)");
    }

    public void Test()
    {
        int i = 1;
        m(i);   // Y.m(float) is called because int fits well into the new method.
    }
}

class Prg6
{
    static void Main(string[] args)
    {

        Y y = new Y();
        y.Test();
    }
}
*/
/*
abstract class A
{
    public virtual void m() => Console.Write("A");
}

class B : A
{
    public virtual void m(int i = 42)
        => Console.Write($"B-{i}");
}

class C : B
{
    public override void m() => Console.Write("C1");

    public virtual void m(int j = 333)
        => Console.Write($"C2-{j}");
}

class D : C
{
    public void m() => Console.Write("D");
}

class Prg7
{
    static void Main()
```

```
    {
        var x = new C();
        x.m();
    }
}
*/

interface I7
{
    public char f();
}

class A7 : I7
{
    public virtual char f() => 'A';
}

class B7 : A7
{
    public override char f() => 'B';
}

class C7 : B7
{
    public virtual char f() => (char)(base.f() + 5);
}

class D7 : C7
{
    public override char f() => 'D';
}

class E7 : D7
{
    public override char f() => (char) (base.f() + 10);
}

class Program
{
    static void Main()
    {
        C7? c7 = new E7();
        Console.WriteLine(c7.f());

        I7 i7 = new D7();
        Console.WriteLine(i7.f());
    }
}

/*
class X1
{
    public int Value
    {
        get
```

```
        {
            Console.WriteLine("X");
            return 10;
        }
    }
}

class Program
{
    static void Main()
    {
        var x1 = new X1();
        if (x1.Value > 7 && x1.Value < 25 && x1.Value != 8)
        {
            Console.WriteLine("OK1");
        }

        Console.WriteLine("+");

        if (x1.Value is > 7 and < 25 and not 8)
        {
            Console.WriteLine("OK2");
        }
    }
}
*/
/*
struct S
{
    private int x = 7;
    private int y = 8;

    public S() { }
}

class Program
{
    static void Main()
    {
        S[] sField = new S[3];  // gets initialized to zeroes

    }
}
*/
/*
class Program
{
    public static void Main(string[] args)
    {
        var before = GC.GetAllocatedBytesForCurrentThread();
        var r1 = Calc([1, 2, 3, 4], 0);
        var after = GC.GetAllocatedBytesForCurrentThread();
        Console.WriteLine($"Allocated {after - before} B.");

        Console.WriteLine(r1);
```

```
        var r2 = Calc(null, 0);
        Console.WriteLine(r2);
    }

    static int Calc(int[] a, int b) => a switch
    {
        [] => b,
        [var x, .. var y] when x % 2 == 0
            => Calc(y, x + b),
        [_, .. var z] => Calc(z, b)
    };
}
*/
/*
class Prgram
{
    static void Main()
    {
        try
        {
            try
            {
                throw new ArgumentException();
            }
            finally
            {
                Console.WriteLine("A");
                try
                {
                    try
                    {
                        throw new IndexOutOfRangeException();
                    }
                    finally
                    {
                        Console.WriteLine("B");
                        try
                        {
                            try
                            {
                                throw new NotSupportedException();
                            }
                            finally
                            {
                                Console.WriteLine("C");
                            }
                        }
                        catch (Exception ex)
                        {
                            Console.WriteLine($"{ex.GetType()}");
                            throw;
                        }

                        Console.WriteLine("D");
                    }
```

```
            }
            catch (Exception ex)
            {
                Console.WriteLine($"{ex.GetType()}");
            }

            Console.WriteLine("E");
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"{ex.GetType()}");
    }
  }
}
*/
```