```
public static class RepeatClass
{
    public static IEnumerable<T> Repeat<T>(this IEnumerable<T> s, int
         multiplicity, T forbidden){
        IEnumerable<T> Repeat_Aux()
        {
            using (var it = s.GetEnumerator())
            {
                while (it.MoveNext())
                {
                    if (Equals(it.Current,forbidden))
                     throw new ArgumentException("forbidden trovato");
                    for (var i = 0; i < multiplicity; i++)
                      yield return it.Current;
                }
            }
        }
        if (multiplicity <= 0)
               throw new ArgumentOutOfRangeException(nameof(multiplicity));
        return Repeat_Aux();
    }
}

public class RepeatTest
{
    [Test]
    public void NormalBehaviour() =>
        Assert.That(new[] { 8, 11, 35 }.Repeat(2, 122), Is.EqualTo(
               new[] { 8, 8, 11, 11, 35, 35 }));

    [Test]
    public void MultiplicityThrowErr() =>
        Assert.That(() => new[] { 3.8, 24.31, 3.675 }.Repeat(0, -4.67),
               Throws.TypeOf<ArgumentOutOfRangeException>());

    [TestCase(3,7)]
    [TestCase(3, 0)]
    [TestCase(0,27)]
    public void BadGuyThrowErr(int position, int badguy)
    {
        IEnumerable<int> Infinite(int p, int bg)
        {
            var rand = new Random();
            int i = 0;
            int num = 0;
            while (true)
            {
                i++;
                while ((num = rand.Next()) == badguy)
                    ;
                if (i == position) num = badguy;
                yield return num;
            }
        }

        if (position <= 0)
               Assert.Inconclusive("position non strettamente positiva");
        var ris = Infinite(position, badguy).Repeat(3, badguy)
               .Take(3 * position + 1);
        Assert.Multiple(() =>
        {
            Assert.That(ris.Take(3 * position), Is.Not.Null);
            Assert.That(() => ris.ToList(), Throws.TypeOf<ArgumentException>());
        });
```