# How to remove item from list in C#?

Asked 7 years, 6 months ago Active 2 months ago Viewed 472k times

153

22

I have a list stored in resultlist as follows:

```
var resultlist = results.ToList();
```

It looks something like this:

```
ID FirstName  LastName
-- ---------  --------
1  Bill       Smith
2  John       Wilson
3  Doug       Berg
```

How do I remove ID 2 from the list?

c# list

edited Nov 30 '18 at 6:58 Cœur 22.8k 10 130 188

asked Apr 4 '12 at 20:44 Nate Pet 16.2k 105 234 374

## 8 Answers

308

`List<T>` has two methods you can use.

RemoveAt(int index) can be used if you know the index of the item. For example:

```
resultlist.RemoveAt(1);
```

```
var itemToRemove = resultlist.Single(r => r.Id == 2);
resultList.Remove(itemToRemove);
```

When you are not sure the item really exists you can use SingleOrDefault. `SingleOrDefault` will return `null` if there is no item ( `Single` will throw an exception when it can't find the item). Both will throw when there is a duplicate value (two items with the same `id` ).

```
var itemToRemove = resultlist.SingleOrDefault(r => r.Id == 2);
if (itemToRemove != null)
    resultList.Remove(itemToRemove);
```

edited Aug 28 '13 at 6:12

answered Apr 4 '12 at 20:47

Wouter de Kort

**32k** 8 61 94

4 well, than maybe `var itemsToRemove = resultlist.Where(r => r.Id == 2); foreach (var itemToRemove in ItemsToRemove) resultList.Remove(itemToRemove);` – Vlad Apr 4 '12 at 20:51

1 Shouldn't this be `resultlist.Items.RemoveAt(1);` ? – DreamTeK Oct 17 '16 at 8:57

39

```
resultList = results.Where(x=>x.Id != 2).ToList();
```

There's a little Linq helper I like that's easy to implement and can make queries with "where not" conditions a little easier to read:

```
public static IEnumerable<T> ExceptWhere<T>(this IEnumerable<T> source, Predicate<T>
predicate)
{
    return source.Where(x=>!predicate(x));
}

//usage in above situation
resultList = results.ExceptWhere(x=>x.Id == 2).ToList();
```

edited Apr 4 '12 at 20:52

answered Apr 4 '12 at 20:45

KeithS

1 Another similar approach (that uses a predicate) is to use `List.FindIndex` / `List.RemoteAt` (which has the "nice" or "not so nice" feature of being a mutating operation). – user166390 Apr 4 '12 at 20:50

True, but be careful about saying that List's operation *is* mutating. List uses an array behind the scenes, and it can recreate its array with a smaller or larger capacity when it thinks that's necessary. *Usually*, removal is an in-place mutation of the existing array. – KeithS Apr 4 '12 at 21:09

This isnt thread safe, and for its simplicity you can just use SingleOrDefault, it doesnt need to be contained in a static method – user1043000 Aug 27 '13 at 20:08

Nobody said it was thread-safe (and whether it is depends on what the threads are supposed to be doing; it may in fact be preferable to give a different in-memory construct to a worker thread versus letting them all work on one concurrent collection), and the OP wants all records *except* the one matching the predicate, so SingleOrDefault would in fact return exactly what they *don't* want. The "static method" is in fact an extension method, like most of Linq, and it works whenever what you don't want (one element or many) is easier to define than what you do. – KeithS Aug 27 '13 at 22:44

30

### Short answer:

**Remove (from list `results`)**

`results.RemoveAll(r => r.ID == 2);` will remove the item with **ID 2** in `results` (in place).

**Filter (without removing from original list `results`):**

`var filtered = result.Where(f => f.ID != 2);` returns all items except the one with **ID 2**

### Detailed answer:

I think `.RemoveAll()` is very flexible, because you can have a list of item IDs which you want to remove - please regard the following example.

If you have:

```
class myClass {
    public int ID; public string FirstName; public string LastName;
}
```

and assigned some values to `results` as follows:

```
var results=new List<myClass> {
    new myClass() { ID=1, FirstName="Bill", LastName="Smith" },
```

```
    new myClass()  { ID=4, FirstName="Bill", LastName="Wilson" },
};
```

Then you can define a list of IDs to remove:

```
var removeList = new List<int>() { 2, 3 };
```

And simply use this to remove them:

```
results.RemoveAll(r => removeList.Any(a => a==r.ID));
```

It will **remove the items 2 and 3** and keep the items 1 and 4 - as specified by the `removeList`. **Note** that this happens in place, so there is no additional assigment required.

Of course, you can also use it on single items like:

```
results.RemoveAll(r => r.ID==4);
```

where it will remove Bill with ID 4 in our example.

DotNetFiddle: Run the demo

edited Apr 27 '18 at 7:45

answered Feb 15 '16 at 14:50

Matt
**16.8k** 7 77 120

---

5

There is another approach. It uses `List.FindIndex` and `List.RemoveAt`.

While I would *probably* use the solution presented by KeithS (just the simple `Where` / `ToList`) this approach differs in that it *mutates* the original list object. This can be a good (or a bad) "feature" depending upon expectations.

In any case, the `FindIndex` (coupled with a guard) ensures the `RemoveAt` will be correct if there are gaps in the IDs or the ordering is wrong, etc, and using `RemoveAt` (vs `Remove`) avoids a *second* O(n) search through the list.

```
var list = new List<int> { 1, 3, 2 };
var index = list.FindIndex(i => i == 2); // like Where/Single
if (index >= 0) {   // ensure item found
    list.RemoveAt(index);
}
list.Dump();        // results -> 1, 3
```

Happy coding.

edited Apr 4 '12 at 21:05

answered Apr 4 '12 at 21:00
user166390

---

3

You don't specify what kind of list, but the generic List can use either the `RemoveAt(index)` method, or the `Remove(obj)` method:

```
// Remove(obj)
var item = resultList.Single(x => x.Id == 2);
resultList.Remove(item);

// RemoveAt(index)
resultList.RemoveAt(1);
```

answered Apr 4 '12 at 20:49

mgnoonan
**5,842** 5 20 25

---

3

More simplified:

```
resultList.Remove(resultList.Single(x => x.Id == 2));
```

there is no needing to create a new var object.

answered May 6 '16 at 20:08

0

... or just `resultlist.RemoveAt(1)` if you know exactly the index.

answered Apr 4 '12 at 20:46

Vlad

**30.9k** 5 62 163

3 only if it's sorted by `ID` . – zzzzBov Apr 4 '12 at 20:47

0

```
{
    class Program
    {
        public static List<Product> list;
        static void Main(string[] args)
        {

            list = new List<Product>() { new Product() { ProductId=1, Name="Nike
12N0",Brand="Nike",Price=12000,Quantity=50},
                new Product() { ProductId =2, Name = "Puma 560K", Brand = "Puma", Price
= 120000, Quantity = 55 },
                new Product() { ProductId=3, Name="WoodLand
V2",Brand="WoodLand",Price=21020,Quantity=25},
                new Product() { ProductId=4, Name="Adidas
S52",Brand="Adidas",Price=20000,Quantity=35},
                new Product() { ProductId=5, Name="Rebook
SPEED2O",Brand="Rebook",Price=1200,Quantity=15}};


            Console.WriteLine("Enter ProductID to remove");
            int uno = Convert.ToInt32(Console.ReadLine());
            var itemToRemove = list.Find(r => r.ProductId == uno);
            if (itemToRemove != null)
                list.Remove(itemToRemove);
            Console.WriteLine($"{itemToRemove.ProductId}{itemToRemove.Name}
{itemToRemove.Brand}{itemToRemove.Price}{ itemToRemove.Quantity}");
            Console.WriteLine("------------sucessfully Removed---------------");

            var query2 = from x in list select x;
            foreach (var item in query2)
```

```
                Console.WriteLine($"{item.ProductId}{item.Name}{item.Brand}{item.Price}{
item.Quantity}");
            }

        }

    }
}
```

answered Jul 24 at 6:24

Prabhakaran M

**1**