**Collections**

**Write a method to copy items from one arraylist or array into other without using any .NET framework method like CopyTo() and counter.**

class Program

{

static void Main(string[] args)

{

List<string> toList;

List<string> fromList = new List<string>();

fromList.Add("One");

fromList.Add("Two");

fromList.Add("Three");

CopyTo(fromList, out toList);

foreach (var item in toList)

{

Console.WriteLine(item);

}

Console.ReadLine();

}

/// <summary>

/// Copies the contents

/// </summary>

/// <param name="fromList">List from which </param>

/// <param name="toList"></param>

private static void CopyTo(List<string> fromList, out List<string> toList)

{

toList = new List<string>();

//Like C# provides indexers to index elements within objects, C# also provides us with Enumerators to

//allow us enumerate through the elements within the objects. We need to use an Interface IEnumerator

//to enumerate through the elements of the object.

IEnumerator<string> enumeratorObj = fromList.GetEnumerator();

while (enumeratorObj.MoveNext())

{

toList.Add(enumeratorObj.Current);

}

}

}

**Why Array Index starts from Zero?**

1. The index of array, which is of the form a[i], is converted by the compiler in the form [a+i]. So, the index of first element is zero because [a+0] will give 'a' & the first array element can be accessed. Due to this, we can also access the array elements as 'i[a]' instead of 'a[i]', & this will not produce an error. But its compiler specific to start an array index with 1 or zero, as in pascal the array index starts with '1'.

2. This boils down to the concept of Binary digits. Take an array size of 64 for example. We start from 0 and end at 63. We require 6 bits. But, if we were to start from 1 and end at 64, we would require 7 bits to store the same number, thus increasing the storage size.

**What is the difference between Array and ArrayList?**

* **Array** is: a datatype, thatcan be used by calling indexes. during runtime, one cannot really change the size of the array, unless you use the method of copying the array and getting rid of the old one.  
  In .NET, the Visual Studio makes use of a special class to store the data. Because of this, the performance is actually quite fast. This is also because in an array, you need to specify the size and thus, the data is stored one after the other.

Examples:

* + int[ ] myNumbers= new int[5];
  + myNumbers[0] = 16;
* **ArrayList** is: a datatype collection. In order to fill an ArrayList, one can use the .Add property. ArrayLists are very dynamic in the sense that when you add and/or remove items from it, the performace stays the same.   
  The internal structure of an ArrayList is an array.

Examples:

* + ArrayList myArray = new ArrayList();
  + myArray .Add(“Steph”);
  + string str = myArray [0];

Most of the time, we tend to choose array lists rather than arrays since we have no idea how big it is going to turn out. Arrays are ideal when you know how many items you are going to put in it. Whenever possible, it is recommended to use arrays as this drastically improves the performance.

**Differentiate HashTable and Dictionary in C#.NET.**

|  |  |
| --- | --- |
| **HashTable** | **Dictionary** |
| A C# Hashtable stores items faster than a C# Dictionary, which sacrifices speed for the sake of order | The Dictionary data structure stores items slower than Hashtable. |
| Values are then stored in order according to their key's HashCode. Meaning that the order in which items are added to a C# Hashtable is not preserved. | The Dictionary data structure *does* keep items in the same order |
| Type unsafe | Type safe since it is a generic type. |
| Hashtable uses rehashing for collision resolution (when a collision occurs, tries another hash function to map the key to a bucket). | Dictionary relies on chaining (maintaining a list of items for each hash table bucket) to resolve collisions. |

(For those Java programmers, a Dictionary is more or less a TreeMap and a Hashtable is a HashMap).

**Differentiate HashSet<T> and List<T> in C#.NET.**

|  |  |
| --- | --- |
| **HashSet** | **List** |
| It does not allow duplicate values. For example:  HashSet<int> mySet = new HashSet<int>();  mySet.Add(3);  mySet.Add(5);  mySet.Add(3);  mySet.Add(10);  List<int> myListFromSet = mySet.ToList<int>();  int myInt = myListFromSet[2];  If mySet were a regular List data structure, the index 2 should return the value 3 (count it out). But if you run the example you will see that myInt actually returns the value 10. This is because the HashSet **C# data structure** ignored the duplicate addition of the value 3.  You might wonder what is the point of this. After all, you could achieve the same behavior with a List data structure. Something like:  if (!myList.Contains(element))  myList.Add(element);  The result is indeed the same. But what is not apparent is the speed at which this happens. When an element is added to a HashSet, internally the same thing happens: the data structure makes sure the element doesn't already exist. However a HashSet is not a simple array, it is specifically designed to allow fast *search times* which dramatically improves the performace of checking whether a new element is a duplicate or not. |  |
|  |  |
|  |  |
|  |  |

**When you use a List, it doesn't matter if you use the ForEach method of the generic list or use a normal foreach or does it? Sometimes it makes a difference!**

**Using the code**

The ForEach method of the List<T> (not IList<T>) executes an operation for every object which is stored in the list. Normally it contains code to either read or modify every object which is in the list or to do something with list itself for every object.

**Modify the object itself**

The following sample with a ForEach method loops over all stored Points in the collection. It substracts 10 from the x coordinate of the point. At the end the Points will be printed to the console.

List<Point> points = new List<Point>(){ new Point(14, 10), new Point(19, 10) };  
  
items.ForEach(point => point.X = point.X - 10);  
  
foreach (Point point in points)  
{  
 Console.WriteLine(point);  
}

The output in the console is in this case {X=14, Y=10} and {X=19, Y=10}. I expected that X is 4 and 9, so what's wrong? If you put the same logic into a normal foreach statement the compiler throws the following error: "Cannot modify members of 'point' because it is a 'foreach iteration variable'". If we define our own type, the code does what it should do!

public class MyPoint  
{  
 public MyPoint(int x, int y){ X = x; Y = y; }  
 public int X{ get; set; }  
 public int Y{ get; set; }  
}  
  
List<MyPoint> points = new List<MyPoint>(){ new MyPoint(14, 10), new MyPoint(19, 10) };  
  
items.ForEach(point => point.X = point.X - 10);  
  
foreach (MyPoint point in points)  
{  
 Console.WriteLine(point);  
}

The difference is, that Point is a value type, a struct, and MyPoint is a reference type. So in the case where Point is used, a copy of the object is passed to the method, not the object itself. So if the action, which is passed into the ForEach method, changes the copy, but it won't affect the original object.

**Modify the collection**

When you use a normal foreach statement, you can't add or remove items while iterating over the collection. But with List.ForEach you can, so the following code can be executed without any errors. Which result do you expect?

public class Integer  
{  
 public int Value { get; set; }  
 public Integer(int value) { Value = value; }  
}  
  
public void Sample()  
{  
 List<Integer> items = new List<Integer>()   
 {   
 new Integer(14),   
 new Integer(0),   
 new Integer(19)   
 };  
  
 items.ForEach(item =>  
 {  
 if (item.Value == 0)  
 {  
 items.Remove(item);  
 }  
 item.Value = item.Value - 10;  
 });  
  
 foreach (Integer item in items)  
 {  
 Console.WriteLine(item.Value);  
 }  
}

The result which is shown in the console is 4 and 19. So this is a good example that not all what you can do, you also should do! The result should be 4 and 9! It seems that internally a for loop is is used, which iterates backward over the collection.

**Conclusion**

So List<T>.ForEach allows several things which is blocked in a foreach loop. These things aren't allowed for a good reason. So if you want to store objects of value types, like int, long, double, bool or even string,  in a generic List, you shouldn't use the ForEach method if you want to avoid problems. A good solution is use a for loop and access the data over the indexer of the collection. Also removing items in the ForEach method is a thing which should be avoided also when it is possible.

.............

long Sum(List<int> intList)  
{  
  long result = 0;  
  intList.ForEach(delegate(int i) { result += i; });  
  result result;  
}

Or, the lambda expression equivalent:

long Sum(List<int> intList)  
{  
  long result = 0;  
  intList.ForEach(i => result += i);  
  return result;  
}

Using List<T>.ForEach results in only one method call per iteration: whatever Action<T> delegate that you supply. This will be called with a *callvirt* IL instruction but two *calls* should be slower than one *callvirt*. So, my expectation is that List<T>.ForEach will actually be faster.

**How do you remove duplicate elements from an array or list?**

If you're going the route of "just don't add duplicates", then checking "List.Contains" before adding an item *works*, but its O(n^2) where n is the number strings you want to add. Its no different from your current solution using two nested loops.

You'll have better luck using a hashset to store items you've already added, but since you're using .NET 2.0, a Dictionary can substitute for a hash set:

static List<T> RemoveDuplicates<T>(List<T> input)   
{   
    List<T> result = new List<T>(input.Count);   
    Dictionary<T, object> hashSet = new Dictionary<T, object>();   
    foreach (T s in input)   
    {   
        if (!hashSet.ContainsKey(s))   
        {   
            result.Add(s);   
            hashSet.Add(s, null);   
        }   
    }   
    return result;   
}

This runs in O(n) and uses O(2n) space, it will generally work very well for up to 100K items. Actual performance depends on the average length of the strings -- if you really need to maximum performance, you can exploit some more powerful data structures like tries make inserts even faster.

Simply initialize a HashSet with a List of the same type in .NET 3.5:

var noDupes = new HashSet<T>(withDupes);

If you don't care about the order you can just shove the items into a HashSet, if you *do* want to maintain the order you can do something like this:

var unique = new List<T>();   
var hs = new HashSet<T>();   
foreach (T t in list)   
    if (hs.Add(t))   
        unique.Add(t);

Or the Linq way:

var hs = new HashSet<T>();   
list.All( x =>  hs.Add(x) );

**Suppose you have a string say "Hello .NET world". Tell how to reverse the entire line i.e., "world .NET Hello" with and without a single collection object?**

**Consider the example:**

**List<int> a = new List<int>();**

**for(int i = 0; i<10; i++)**

**{**

**a.Items.Add(i);**

**}**

**for(int i = 0; i<10; i++)**

**{**

**a.Items.Remove(i);**

**}**

**What is the problem with this code?**

**What is the problem in below code? How to solve the problem?**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**int position = 1;**

**List<int> list = new List<int>();**

**for (int i = 0; i < 4; i++)**

**{**

**list.Add(i);**

**}**

**for (int i = 0; i < 4; i++)**

**{**

**//remove elements from even positions**

**list.RemoveAt(position);**

**position++;**

**}**

**}**

**}**

Throws System.ArgumentOutOfRangeException. Index was out of range. Must be non-negative and less than the size of the collection.

Parameter name: index

To correct the problem use

for (int i = 0; i < list.Count; i++)

{

//remove elements from even positions

list.RemoveAt(position);

position++;

}

**Can you tell how to design below method? Use same method to display contents of stack, queue, arraylist**

**void M1(...) //should be allowed to pass any collection objects like ArrayList, List, Stack, Queue, etc**

**{**

**//Print each elements of collection**

**}**

void M1(ICollection coll) //need to pass ICollection type

{

foreach (var item in coll)

{

Console.WriteLine(item);

}

}

**Suppose you have elements in an array of elements with values 0 and 1 say int [] intArr = {0,1,0,1,0,1,1}; How do we sort the elements with a single for loop and without using any other collection or array?**

#include <stdio.h>   
   
int main()    
{   
    int i;   
    int count;   
    int N = 18;   
    int arr[] = {1,1,0,1,0,0,0,1,0,1,0,1,0,1,0,1,0,1};   
   
    /\* Sum up all elements \*/   
    i = 0;   
    count = 0;   
    while (i < N) count += arr[i++];   
   
    /\* Overwrite the array \*/   
    i = 0;   
    count = N - count;   
    while (i < count) arr[i++] = 0;   
    while (i < N) arr[i++] = 1;   
   
    /\* Print result \*/   
    for (i = 0; i < N; i++) printf("%d ",arr[i]);   
}

**Verify the flag bit in position 3 from MSB in a byte.**

**int IsThirdBitSet(byte in)**

**{**

**}**

int IsThirdBitSet(byte in)

{

byte THIRDBIT = 4; // 4 = 00000100 i.e. third bit is set   
 return in & THIRDBIT; // Returns 1 if the third bit is set, 0 otherwise   
}

**What would happen if you add an item to list object while iterating through the items in for loop and printing the value? Is it going to print the added item or throw exception while iterating?**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**int position = 1;**

**List<int> list = new List<int>();**

**for (int i = 0; i < 4; i++)**

**{**

**list.Add(i);**

**}**

**for (int i = 0; i < 4; i++)**

**{**

**//add an element**

**list.Add(position);**

**position++;**

**}**

**}**

**}**

**See the following code:**

|  |  |  |
| --- | --- | --- |
| **01** | **List<int> list = new List<int>();** | |
| **02** |  |

|  |  |  |
| --- | --- | --- |
| **03** | **for (int i = 1; i < 10; i++)** | |
| **04** | **{** |

|  |  |  |
| --- | --- | --- |
| **05** | **list.Add(i);** | |
| **06** | **}** |

|  |  |
| --- | --- |
| **07** |  |
| **08** | **foreach (int i in list)** | |

|  |  |
| --- | --- |
| **09** | **{** |
| **10** | **list.Remove(i);** | |

|  |  |
| --- | --- |
| **11** | **}** |

**Do you see anything wrong in this piece of code?**

Execute this code and you will get a System.InvalidOperationException with the message *“Collection was modified; enumeration operation may not execute.”*

The first time foreach loop executes, it deletes the first item and in the second iteration, this nasty exception is thrown. Reason? You deleted the first item and the list has changed. If not for the InvalidOperationException , you could have got a IndexOutOfRange exception while traversing the list.

So, how does one delete some items in the list while iterating through the list? I faced this problem in my project today and I couldn’t come up with an elegant solution. I did come up with a hack. I identify all items that need to be deleted and mark them with a flag (I need one iteration here). In another iteration, I create a new list and copy all those items which are not marked.

Sorry, I couldn’t come up with something better. If you have a solution or a better fix, please do let me know. Please note that if I want to remove all items, I can use list.Clear(). I am talking about a scenario where I want to delete only a select few items.

Update:

I read somewhere that if you use for loop instead of foreach, this problem does not arise. True, you will not get an exception, but you might not delete the items that you actually want to delete. Let’s see what happens in our case.

|  |  |  |
| --- | --- | --- |
| List<int> list = new List<int>(); | | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | for (int i = 1; i < 10; i++) | |
| 04 | { |

|  |  |  |
| --- | --- | --- |
| 05 | list.Add(i); | |
| 06 | } |

|  |  |
| --- | --- |
| 07 |  |
| 08 | for (int i = 0; i < list.Count; i++) | |

|  |  |
| --- | --- |
| 09 | { |
| 10 | int remove = list[i]; | |

|  |  |  |
| --- | --- | --- |
| 11 | list.Remove(remove); | |
| 12 | } |

You will delete items 1, 3, 5, 7 and 9. After the for loop exits, your list will still have 2, 4, 6 and 8. Surprised?

Let’s see why this is so.

**First iteration:** i = 0, remove = 1. Deleted item = 1 Once you delete this item, the list has changed. So, the first element in the list is now 2.  
**Second iteration:** i = 1, remove = 3. Deleted item = 3  
**Third iteration:** i = 2, remove = 5. Deleted item = 5

and so on….

So, replacing foreach with a for is not a solution. To solve the problem, iterate from back of the list or cache the count up front before any changes:

for (int i = list.Count-1; i >=0; i--)

{

/// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/// Choose one

/// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/// Remove positionally:

// list.RemoveAt(i);

/// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/// Remove by value

// list.Remove(i);

/// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

list.RemoveAt(i);

}

int count = list.Count;

for (int i = 0; i < count; ++i)

{

};

**How to implement cache using weak reference and dictionary?**

A weak reference allows the garbage collector to collect an object while still allowing an application to access the object. If you need the object, you can still obtain a strong reference to it and prevent it from being collected. For more information about how to use short and long weak references.

The following example demonstrates how you can use weak references to maintain a cache of objects as a resource for an application. The cache is constructed using an [IDictionary<TKey, TValue>](http://msdn.microsoft.com/en-us/library/s4ys34ea.aspx) of WeakReference objects keyed by an index value. The [Target](http://msdn.microsoft.com/en-us/library/system.weakreference.target.aspx) property for the WeakReference objects is an object in a byte array that represents data.

The example randomly accesses objects in the cache. If an object is reclaimed for garbage collection, a new data object is regenerated; otherwise, the object is available to access because of the weak reference.

using System;

using System.Collections.Generic;

public class Program

{

public static void Main()

{

// Create the cache.

int cacheSize = 50;

Random r = new Random();

Cache c = new Cache(cacheSize);

string DataName = "";

// Randomly access objects in the cache.

for (int i = 0; i < c.Count; i++)

{

int index = r.Next(c.Count);

// Access the object by

// getting a property value.

DataName = c[index].Name;

}

// Show results.

double regenPercent = c.RegenerationCount \* 100 / c.Count;

Console.WriteLine("Cache size: {0}, Regenerated: {1}%", c.Count.ToString(), regenPercent.ToString());

}

}

public class Cache

{

// Dictionary to contain the cache.

static Dictionary<int, WeakReference> \_cache;

// Track the number of times an

// object is regenerated.

int regenCount = 0;

public Cache(int count)

{

\_cache = new Dictionary<int, WeakReference>();

// Add data objects with a

// short weak reference to the cache.

for (int i = 0; i < count; i++)

{

\_cache.Add(i, new WeakReference(new Data(i), false));

}

}

// Returns the number of items in the cache.

public int Count

{

get

{

return \_cache.Count;

}

}

// Returns the number of times an

// object had to be regenerated.

public int RegenerationCount

{

get

{

return regenCount;

}

}

// Accesses a data object from the cache.

// If the object was reclaimed for garbage collection,

// create a new data object at that index location.

public Data this[int index]

{

get

{

// Obtain an instance of a data

// object from the cache of

// of weak reference objects.

Data d = \_cache[index].Target as Data;

if (d == null)

{

// Object was reclaimed, so generate a new one.

Console.WriteLine("Regenerate object at {0}: Yes", index.ToString());

d = new Data(index);

regenCount++;

}

else

{

// Object was obtained with the weak reference.

Console.WriteLine("Regenerate object at {0}: No", index.ToString());

}

return d;

}

}

}

// This class creates byte arrays to simulate data.

public class Data

{

private byte[] \_data;

private string \_name;

public Data(int size)

{

\_data = new byte[size \* 1024];

\_name = size.ToString();

}

// Simple property.

public string Name

{

get

{

return \_name;

}

}

}

// Example of the last lines of the output:

//

// ...

// Regenerate object at 36: Yes

// Regenerate object at 8: Yes

// Regenerate object at 21: Yes

// Regenerate object at 4: Yes

// Regenerate object at 38: No

// Regenerate object at 7: Yes

// Regenerate object at 2: Yes

// Regenerate object at 43: Yes

// Regenerate object at 38: No

// Cache size: 50, Regenerated: 94%

//

**Implement your own Hashtable using arrays.**

class HashtableEx

{

private List<object> keys = new List<object>();

private List<object> values = new List<object>();

public object this[object key]

{

get

{

int index = keys.IndexOf(key);

if (index == -1) return null;

else return values[index];

}

set

{

if (keys.Contains(key))

{

int index = keys.IndexOf(key);

values[index] = value;

}

else

{

keys.Add(key);

values.Add(value);

}

}

}

public object Get(object key)

{

int index = keys.IndexOf(key);

return values[index];

}

public void Add(object key, object value)

{

keys.Add(key);

values.Add(value);

}

public void Remove(object key)

{

int index = keys.IndexOf(key);

keys.RemoveAt(index);

values.RemoveAt(index);

}

public void Clear()

{

keys = new List<object>();

values = new List<object>();

}

}

**With this usage code:**

class Program

{

static void Main(string[] args)

{

HashtableEx ht = new HashtableEx();

ht["hello"] = "world";

Console.WriteLine("Value of \"hello\": {0}", ht["hello"]);

//Value of "hello": world

Console.WriteLine("Type of \"hello\": {0}", ht["hello"].GetType());

// Type of "hello": System.String

Console.ReadKey();

}

}

**Implement your own ArrayList using array or without using collections.**

ArrayList is similar to Object Array but provides the feature of dynamic space allocation when the number of objects in the list grows. In Object Array we need to provide the size at the time of initialization but that is not required for ArrayList. Actually when you initialize ArrayList, it automatically assigns its capacity to 10.

This is just the basic implementation of ArrayList using Array and to understand how it’s implemented.

public class MyArrayList

{

private static readonly int SIZE\_FACTOR = 5;

private object[] data;

private int index;

private int size;

public MyArrayList()

{

this.data = new object[SIZE\_FACTOR];

this.size = SIZE\_FACTOR;

}

public void Add(object obj)

{

Console.WriteLine("Index:" + this.index + " Size:" + this.size + " Data size:" + this.data.Length);

if (this.index == this.size - 1)

IncreaseSizeAndReallocate();

data[this.index] = obj;

this.index++;

}

/// <summary>

/// Increases the size of data[]

/// </summary>

private void IncreaseSizeAndReallocate()

{

this.size = this.size + SIZE\_FACTOR;

object[] newData = new object[this.size];

for (int i = 0; i < data.Length; i++)

{

newData[i] = data[i];

}

this.data = newData;

Console.WriteLine("\n\*\*\*Index:" + this.index + " Size:" + this.size + " Data size:" + this.data.Length);

}

public object GetItem(int indexOfItemToBeObtained)

{

if (indexOfItemToBeObtained > this.index - 1)

throw new IndexOutOfRangeException("ArrayIndexOutOfBound");

if (indexOfItemToBeObtained < 0)

throw new Exception("Negative Value");

return this.data[indexOfItemToBeObtained];

}

public void RemoveAt(int indexOfItemToBeRemoved)

{

if (indexOfItemToBeRemoved > this.index - 1)

throw new IndexOutOfRangeException("ArrayIndexOutOfBound");

if (indexOfItemToBeRemoved < 0)

throw new Exception("Negative Value");

Console.WriteLine("Object getting removed: " + this.data[indexOfItemToBeRemoved]);

for (int x = indexOfItemToBeRemoved; x < this.data.Length - 1; x++)

{

data[x] = data[x + 1];

}

this.index--;

}

public static void Main(String[] args)

{

MyArrayList mal = new MyArrayList();

mal.Add("0");

mal.Add("1");

mal.Add("2");

mal.Add("3");

mal.Add("4");

mal.Add("5");

mal.Add("6");

mal.Add("7");

mal.Add("8");

mal.Add("9");

mal.RemoveAt(5);

Console.WriteLine(mal.GetItem(7));

Console.ReadLine();

}

}

Index:0 Size:5 Data size:5

Index:1 Size:5 Data size:5

Index:2 Size:5 Data size:5

Index:3 Size:5 Data size:5

Index:4 Size:5 Data size:5

\*\*\*Index:4 Size:10 Data size:10

Index:5 Size:10 Data size:10

Index:6 Size:10 Data size:10

Index:7 Size:10 Data size:10

Index:8 Size:10 Data size:10

Index:9 Size:10 Data size:10

\*\*\*Index:9 Size:15 Data size:15

Object getting removed: 5

8

**Implement your own Queue using array.**

public class MyQueue

{

public static readonly int DEFAULT\_SIZE = 5;

private object[] data;

private int index;

public MyQueue()

{

data = new object[DEFAULT\_SIZE];

}

public bool IsEmpty()

{

return index == 0;

}

public void Enqueue(object obj)

{

if (index == DEFAULT\_SIZE - 1)

throw new Exception("Queue is full. Dequeue some objects");

this.data[index] = obj;

this.index++;

}

public object Dequeue()

{

if (IsEmpty())

throw new Exception("Queue is empty");

object obj = this.data[0];

for (int i = 0; i < this.index - 1; i++)

{

data[i] = data[i + 1];

}

this.index--;

return obj;

}

public static void Main(String[] args)

{

MyQueue queue = new MyQueue();

queue.Enqueue("1");

Console.WriteLine(queue.Dequeue()); //1

queue.Enqueue("2");

queue.Enqueue("3");

queue.Enqueue("4");

Console.WriteLine(queue.Dequeue()); //2

queue.Enqueue("5");

queue.Enqueue("6");

Console.ReadLine();

}

}

**Implement your own Stack using array.**

public class MyStack<T>

{

private int capacity;

private T[] stack;

private int top;

public MyStack(int maxElements)

{

capacity = maxElements;

stack = new T[capacity];

//initialize top with -1

}

public int push(T element)

{

//Check Overflow

if (top == capacity - 1)

{

// return -1 if over flow is there

return -1;

}

else

{

// insert element into stack

top = top + 1;

stack[top] = element;

}

return 0;

}

public T pop()

{

T removedElement;

T temp = default(T);

//check Underflow

if (!(top <= 0))

{

removedElement = stack[top];

top = top - 1;

return removedElement;

}

return temp;

}

public T peep(int position)

{

T temp = default(T);

//check if Position is Valid or not

if (position < capacity && position >= 0)

{

return stack[position];

}

return temp;

}

public T[] GetAllStackElements()

{

T[] Elements = new T[top + 1];

Array.Copy(stack, 0, Elements, 0, top + 1);

return Elements;

}

}

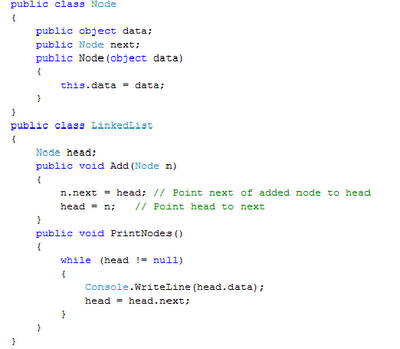
**Implement your own LinkedList object.**

Linked list is the most basic data structure taught in the colleges.Though it looks simple,the kind of complex problems we can solve through it is just amazing.In languages like C,C++ its very easy to understand though pointers.Unfortunately or fortunately in C#,we don't need pointers to learn about linked list though internally,its all pointers all the way to the memory addresses.In this post I will try to explain how a link list can be implemented in C#.

Ok,enough of technical jargon.Lets come straight to the point.

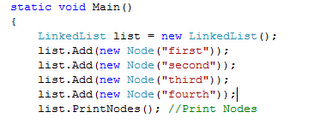
Definition : A linked list is a simple chain of nodes where a node points to next node and so on till the next node doesn't points to anything.

Now what is a node? A node is a simple object which contains some data and a reference to next node.Lets code both our node and linked list class.

**[](http://3.bp.blogspot.com/_YogLU8KL35I/TJRkJ2dx2FI/AAAAAAAAD7o/vrg2No-WEzs/s1600/ff.png)**

The Node class is self explanatory.  
As we can see the Linked List class just contains a head node and all the nodes are added to this node.A little bit of explanation on Add function.It takes a parameter of Node type which we are adding to the head.

The main point to note down here is that we are not adding after the head,rather we are adding before the head.If we realise, we find that it is based on stack based adding which will become more evident when we print the nodes.

**[](http://2.bp.blogspot.com/_YogLU8KL35I/TJueDD2wjcI/AAAAAAAAD8o/nKIeItio27A/s1600/Untitled.png)**

Lets write the client code.It is a simple console application which uses these 2 classes.

Following is the output :

fourth

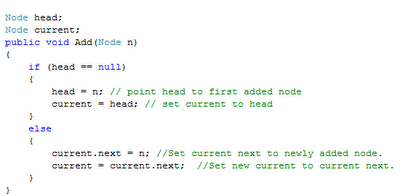
third

second

first

And we are not surprised why this is the output.

Now let’s modify our linkedlist class so that we always add a node after the head. The modified Add function is:

**[](http://3.bp.blogspot.com/_YogLU8KL35I/TJugjXRMjRI/AAAAAAAAD9A/0ZZjdRUDYic/s1600/Untitled.png)**

This time the output is :

first

second

third

fourth

This is just the beginning of linked lists and in future we will see more practical examples of linked lists.This is the beauty of C# that internal memory addressing is hidden from us and all we have to know is how reference and value types work in C#. .NET runtime do the all work of referencing and dereferencing for us.

**How do you design a type that may be placed in a Hashtable/Dictionary collection? Explain with an Employee class. Employee class has name, address and empid as properties.**

Dictionaries represent a sophisticated data structure that allows you to access an element based on a key. Dictionaries are also known as hash tables or maps. The main feature of dictionaries is fast lookup based on keys. You can also add and remove items freely, a bit like a List<T>, but without the performance overhead of having to shift subsequent items in memory.

[Figure](http://www.roque-patrick.com/windows/final/bbl0092.html#ch1) below shows a simplified representation of a dictionary. Here employee-ids such as B4711 are the keys added to the dictionary. The key is transformed into a hash. With the hash a number is created to associate an index with the values. The index then contains a link to the value. The figure is simplified because it is possible that a single index entry can be associated with multiple values, and the index can be stored as a tree.

[](http://www.roque-patrick.com/windows/final/images/fig350_01_0.jpg)

The .NET Framework offers several dictionary classes. The main class you can use is Dictionary<TKey, TValue>. This class offers nearly the same properties and methods as SortedList<TKey, TValue> discussed earlier; that’s why they are not repeated here.

Key Type

A type that is used as a key in the dictionary must override the method GetHashCode() of the Object class. Whenever a dictionary class needs to work out where an item should be located, it calls the GetHashCode() method. The int that is returned by GetHashCode() is used by the dictionary to calculate an index where to place the element. We don’t go into this part of the algorithm. What you should know is that it involves prime numbers, so the capacity of a dictionary is a prime number.

The implementation of GetHashCode() must satisfy these requirements:

* The same object should always return the same value.
* Different objects can return the same value.
* It should execute as quickly as possible; it must be inexpensive to compute.
* It must not throw exceptions.
* It should use at least one instance field.
* The hash code value should be evenly distributed across the entire range of numbers that an int can store.
* At best, the hash code should not change during the lifetime of the object.

|  |  |  |
| --- | --- | --- |
|  | Important | A good performance of the dictionary is based on a good implementation of the method GetHashCode(). |

What’s the reason for having hash code values evenly distributed across the range of integers? If two keys return hashes that give the same index, the dictionary class needs to start looking for the nearest available free location to store the second item - and will have to do some searching in order to retrieve this item later on. This is obviously going to hurt performance, and clearly, if lots of your keys are tending to give the same indexes for where they should be stored, this kind of clash becomes more likely. However, because of the way that Microsoft’s part of the algorithm works, this risk is minimized when the calculated hash values are evenly distributed between int.MinValue and int.MaxValue.

Besides having an implementation of GetHashCode(), the key type also must implement the IEquality.Equals() method or override the Equals() method from the Object class. Because different key objects may return the same hash code, the method Equals() is used by the dictionary comparing keys. The dictionary examines if two keys A and B are equal; it invokes A.Equals(B). This means that you must ensure that the following is always true:

|  |  |  |
| --- | --- | --- |
|  | Important | If A.Equals(B) is true, then A.GetHashCode() and B.GetHashCode() must always return the same hash code. |

This probably seems a fairly subtle point, but it is crucial. If you contrived some way of overriding these methods so that the preceding statement was not always true, a dictionary that uses instances of this class as its keys would simply not work properly. Instead, you’d find funny things happening. For example, you might place an object in the dictionary and then discover that you could never retrieve it, or you might try to retrieve an entry and have the wrong entry returned.

|  |  |  |
| --- | --- | --- |
|  | Tip | For this reason, the C# compiler will display a compilation warning if you supply an override for Equals() but don’t supply an override for GetHashCode(). |

For System.Object this condition is true, because Equals() simply compares references, and GetHashCode() actually returns a hash that is based solely on the address of the object. This means that hash tables based on a key that doesn’t override these methods will work correctly. However, the problem with this way of doing things is that keys are regarded as equal only if they are the same object. That means that when you place an object in the dictionary, you then have to hang onto the reference to the key. You can’t simply instantiate another key object later with the same value. If you don’t override Equals() and GetHashCode(), the type is not very convenient to use in a dictionary.

Incidentally, System.String implements the interface IEquatable and overloads GetHashCode() appropriately. Equals() provides value comparison, and GetHashCode() returns a hash based on the value of the string. Strings can be used conveniently as keys in dictionaries.

Number types such as Int32 also implement the interface IEquatable and overload GetHashCode(). However, the hash code returned by these types simply maps to the value. If the number you would like to use as a key is not itself distributed around the possible values of an integer, using integers as keys doesn’t fulfill the rule of evenly distributing key values to get the best performance. Int32 is not meant to be used in a dictionary.

If you need to use a key type that does not implement IEquatable and override GetHashCode according to the key values you store in the dictionary, you can create a comparer implementing the interface IEqualityComparer<T>. IEqualityComparer<T> defines the methods GetHashCode() and Equals() with an argument of the object passed, so you can offer an implementation different from the object type itself. An overload of the Dictionary<TKey, TValue> constructor allows passing an object implementing IEqualityComparer<T>. If such an object is assigned to the dictionary, this class is used to generate the hash codes and compare the keys.

Dictionary Example

The dictionary example is a program that sets up a dictionary of employees. The dictionary is indexed by EmployeeId objects, and each item stored in the dictionary is an Employee object that stores details of an employee.

The class EmployeeId is implemented to define a key to be used in a dictionary. The members of the class are a prefix character and a number for the employee. Both of these variables are read-only and can only be initialized in the constructor. A key within the dictionary shouldn’t change, and this way that is guaranteed. The fields are filled within the constructor. The ToString() method is overloaded to get a string representation of the employee ID. As required for a key type, EmployeeId implements the interface IEquatable and overloads the method GetHashCode().

[Serializable]

public class EmployeeId : IEquatable<EmployeeId>

{

private readonly char prefix;

private readonly int number;

public EmployeeId(string id)

{

if (id == null) throw new ArgumentNullException("id");

prefix = (id.ToUpper())[0];

int numLength = id.Length - 1;

number = int.Parse(id.Substring(1, numLength > 6 ? 6 : numLength));

}

public override string ToString()

{

return prefix.ToString() + string.Format("{0,6:000000}", number);

}

public override int GetHashCode()

{

return (number ^ number << 16) \* 0x15051505;

}

public bool Equals(EmployeeId other)

{

if (other == null) return false;

return (prefix == other.prefix && number == other.number);

}

}

The Equals() method that is defined by the IEquatable<T> interface compares the values of two EmployeeId objects and returns true if the both values are the same. Instead of implementing the Equals() method from the IEquatable<T> interface, you can also override the Equals() method from the Object class:

public bool Equals(EmployeeId other)

{

if (other == null) return false;

return (prefix == other.prefix && number == other.number);

}

With the number variable, a value from 1 to around 190000 is expected for the employees. This doesn’t fill the range of an integer. The algorithm used by GetHashCode() shifts the number 16 bits to the left, then does a xor with the original number, and finally multiplies the result by the hex value 15051505. The hash code is fairly distributed across the range of an integer.

public override int GetHashCode()

{

return (number ^ number << 16) \* 0x15051505;

}

|  |  |  |
| --- | --- | --- |
|  | Tip | On the Internet you can find a lot of more complex algorithms that have a better distribution across the integer range. You can also use the GetHashCode() method of a string to return a hash. |

The Employee class is a simple entity class containing the name, salary, and ID of the employee. The constructor initializes all values, and the method ToString() returns a string representation of an instance. The implementation of ToString() uses the StringBuilder object to create the string representation for performance reasons.

[Serializable]

public class Employee

{

private string name;

private decimal salary;

private readonly EmployeeId id;

public Employee(EmployeeId id, string name, decimal salary)

{

this.id = id;

this.name = name;

this.salary = salary;

}

public override string ToString()

{

StringBuilder sb = new StringBuilder(100);

sb.AppendFormat("{0}: {1, -20} {2:C}", id.ToString(), name, salary);

return sb.ToString();

}

}

In the Main() method of the sample application, a new Dictionary<TKey, TValue> instance is created, where the key is of type EmployeeId and the value is of type Employee. The constructor allocates a capacity of 31 elements. Remember, the capacity based on prime numbers. However, when you assign a value that is not a prime number, you don’t need to worry. The Dictionary<TKey, TValue> class itself takes the next prime number that follows the integer passed to the constructor to allocate the capacity. The employee objects and IDs are created and added to the dictionary with the Add() method. Instead of using the Add() method, you can also use the indexer to add keys and values to the dictionary, as shown with the employees Jeff and Casey:

static void Main()

{

Dictionary<EmployeeId, Employee> employees =

new Dictionary<EmployeeId, Employee>(31);

EmployeeId idJimmie = new EmployeeId("B4711");

Employee jimmie = new Employee(idJimmie, "Jimmie Johnson", 8909140.00m);

employees.Add(idJimmie, jimmie);

Console.WriteLine(jimmie);

EmployeeId idTony = new EmployeeId("B12836");

Employee tony = new Employee(idTony, "Tony Stewart", 7285280.00m);

employees.Add(idTony, tony);

Console.WriteLine(tony);

EmployeeId idMatt = new EmployeeId("N34434");

Employee matt = new Employee(idMatt, "Matt Kenseth", 6608920.00m);

employees.Add(idMatt, matt);

Console.WriteLine(matt);

EmployeeId idJeff = new EmployeeId("J127");

Employee jeff = new Employee(idJeff, "Jeff Gordon", 5975870.00m);

employees[idJeff] = jeff;

Console.WriteLine(jeff);

EmployeeId idCasey = new EmployeeId("K100223");

Employee casey = new Employee(idCasey, "Casey Mears", 5413340.00m);

employees[idCasey] = casey;

Console.WriteLine(casey);

After the entries are added to the dictionary, inside a while loop employees are read from the dictionary. The user is asked to enter an employee number to store the number in the variable userInput. The user can exit the application by entering X. If the key is in the dictionary, it is examined with the TryGetValue() method of the Dictionary<TKey, TValue> class. TryGetValue() returns true if the key is found and false otherwise. If the value is found, the value associated with the key is stored in the employee variable. This value is written to the console.

|  |  |  |
| --- | --- | --- |
|  | Tip | You can also use an indexer of the Dictionary<TKey, TValue> class instead of TryGetValue() to access a value stored in the dictionary. However, if the key is not found, the indexer throws an exception of type KeyNotFoundException. |

while (true)

{

Console.Write("Enter employee id (X to exit)> ");

string userInput = Console.ReadLine();

userInput = userInput.ToUpper();

if (userInput == "X") break;

EmployeeId id = new EmployeeId(userInput);

Employee employee;

if (!employees.TryGetValue(id, out employee))

{

Console.WriteLine("Employee with id {0} does not exist",

id);

}

else

{

Console.WriteLine(employee);

}

}

}

Running the application produces the following output:

Enter employee ID (format:A999999, X to exit)> J127

J000127: Jeff Gordon $5,975,870.00

Enter employee ID (format:A999999, X to exit)> N34434

N034434: Matt Kenseth $6,608,920.00

Enter employee ID (format:A999999, X to exit)> X

**What are the differences between stack and heap data structure?**

The following table summarizes the differences between the Heap and the Stack:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Memory** | **Contents** | **Item Order** | **Item Lifetime** | **Item Removal** | **Timing** |
| Stack | value types, stack frames | sequential (LIFO) | scope | pop | deterministic |
| Heap | objects | random | reference count | Garbage Collection | nondeterministic |

[**The Stack**](http://en.csharp-online.net/Glossary:Definition_-_Stack)

The [stack](http://en.csharp-online.net/Glossary:Definition_-_Stack) is a data structure in memory used for storing items in a last-in, first-out manner (LIFO). The stack contains both local variables and the call stack. In C#, [value types](http://en.csharp-online.net/Glossary:Definition_-_Value_type) are stored on the stack.

Local variables

Local variables are variables that are declared inside a method. They can be of either value type or reference type. Value type variables are stored directly on the stack. A reference type variable is actually a numerical address pointing to a location on the heap where the object is stored.

Call stack

Every C# application begins with a Main method. In turn, it calls other methods which call still other methods. When a method is called, it is added to the top of the stack. And, when a method returns to its caller, it is removed from the stack. Execution continues with the calling method.

Scope

Scope defines the lifetime of a variable. The block of code in which a variable is declared is its scope. When the block in which a variable is declared begins execution, the variable's lifetime begins when it is pushed onto the stack as part of the call stack. When execution leaves the block in which the variable was declared, the call stack and variables are popped off the stack. The variable's lifetime has ended.

[**The Heap**](http://en.csharp-online.net/Glossary:Definition_-_Heap)

The [heap](http://en.csharp-online.net/Glossary:Definition_-_Heap) or managed heap is a data structure in memory where all objects—reference types—are stored. When an object is instantiated, the object is stored on the heap as a block of data containing its data members. Then, the memory address of the block is stored in a reference variable. Future references to the object are through the reference variable.

In C#, the new operator is used to instantiate a class and return the reference to the object on the heap.

MyClass myObject = new MyClass (); *// declare and instantiate*

string bestSite = "C# Online.NET"; *// declare and initialize*

It is also possible for objects to contain references to other objects:

*// Create an array on the heap*

string[] words = new string [10];

*// words is a reference that exists on the stack as a 32 bit*

*// pointer to a location on the heap*

*// It refers to an object on the heap that contains 10 values*

*// Since string is a reference type, each of the array items*

*// is actually a reference to its own string object on the heap*

*// Therefore, the words array object has 10 string reference*

*// each of 4 bytes = 40 bytes*

Assert.IsTrue (sizeof (words) == 10 \* 4);

*// Also, each string reference is initialized to null*

Assert.IsTrue (words[0] == null);

**Show how the stack looks like when below code is executed.**

**public class A**

**{**

**void Print(int x) {Console.Write(x);}**

**}**

**public class MainClass**

**{**

**static void Main()**

**{**

**A a = new A();**

**a.Print(5);**

**}**

**}**

**I have two classes ClassA and ClassB both having a reference to a singleton object ClassHelper. My question is how should i dispose the singleton object once im done using both the ClassA and ClassB. Or When and how the instance of a singleton object will be disposed?**

public ClassA  
{  
 CHelper obj;  
  
 public ClassA()  
 {  
 obj = obj.GetInstance("Initialise");  
 obj.CallFuncA();  
 }  
}  
  
On the same lines  
public ClassB  
{  
 CHelper obj;  
  
 public ClassB()  
 {  
 obj = obj.GetInstance("Initialise");  
 obj.CallFuncB();  
 }  
}  
  
where   
  
CHelper  
{  
 private static sm\_CHelper;  
  
 public static CHelper GetInstance(string strInitialise)  
 {  
 if(sm\_CHelper == null)  
 {  
 sm\_CHelper = new CHelper(strInitialise);  
 }  
 }  
  
 private CHelper(string strInitialise)  
 {  
 //do something here   
 }  
  
 public CallFuncA()  
 {  
 // do something here  
 }  
 public CallFuncB()  
 {  
 // do something here  
 }  
}

Singleton does not address the issue of deleting the single object. In languages that provide memory management (for example, languages based on the .NET Framework), only the **Singleton** class could cause the instance to be de-allocated because it holds a private reference to the instance. In languages, such as C++, other classes could delete the object instance, but doing so would lead to a dangling reference inside the **Singleton** class.

"A singleton object's lifetime is the lifetime of the application containing it".

If you are talking about the pattern singleton then you should not dispose it.... if your not referring to the singleton pattern then you could try to use the destructor to run your dispose logic.

It is rarely the case that a singleton would implement IDisposable, since the interface will almost never be consumed (unless you want a big 'using' statement around your whole app :P).

Why static class can't inherit another static class?

The main features of a static class are:

* They only contain static members.
* They cannot be instantiated.
* They are sealed.
* They cannot contain Instance Constructors (C# Programming Guide).

So, inheriting from a non-static class violates the first feature of static classes on this list by introducing non-static members to your static class.

Citation from [here](http://connect.microsoft.com/VisualStudio/feedback/ViewFeedback.aspx?FeedbackID=97955):

This is actually by design. There seems to be no good reason to inherit a static class. It has public static members that you can always access via the class name itself. **The only reasons I have seen for inheriting static stuff have been bad ones, such as saving a couple of characters of typing.**

There may be reason to consider mechanisms to bring static members directly into scope (and we will in fact consider this after the Orcas product cycle), but static class inheritance is not the way to go: It is the wrong mechanism to use, and works only for static members that happen to reside in a static class.

(Mads Torgersen, C# Language PM)

Other opinions from [channel9](http://channel9.msdn.com/forums/TechOff/250972-Static-Inheritance/)

**Inheritance in .NET works only on instance base.** Static methods are defined on the type level not on the instance level. That is why overriding doesn't work with static methods/properties/events...

Static methods are only held once in memory. There is no virtual table etc. that is created for them.

If you invoke an instance method in .NET, you always give it the current instance. This is hidden by the .NET runtime, but it happens. **Each instance method has as first argument a pointer (reference) to the object that the method is run on. This doesn't happen with static methods** (as they are defined on type level). How should the compiler decide to select the method to invoke?

(littleguru)

And as a valuable idea, littleguru has a partial "workaround" for this issue: the [Singleton](http://www.dofactory.com/Patterns/PatternSingleton.aspx) pattern.

How do you optimize below logger class? Is it a correct class design for logging purpose.

Class Logger

{

Public void Log(string dataToBeLogged)

{

//log into a file

}

}

What are the various sorting algorithms? Explain their complexities.

You've an assembly in your working directory and also same assembly in GAC? Which assembly would the project use? The name, version, etc of both assemblies are same. How do you make sure that your application uses the assembly from you local directory and not GAC? or How can I force .NET to use a local copy of an assembly that's in the GAC? Or, How to get .NET to look for assemblies in local folders first instead of GAC?

<http://stackoverflow.com/questions/267693/how-can-i-force-net-to-use-a-local-copy-of-an-assembly-thats-in-the-gac>

Make sure the GAC Assembly and local Assembly have different version numbers (not a bad idea to let your build number, at least, auto-increment by wild-carding your AssemblyVersion in AssemblyInfo: [assembly: AssemblyVersion("1.0.0.\*")] ). Then, redirect your assembly binding using your app's config:

* <http://msdn.microsoft.com/en-us/library/2fc472t2(VS.80).aspx>
* [http://msdn.microsoft.com/en-us/library/433ysdt1(VS.80).aspx.](http://msdn.microsoft.com/en-us/library/433ysdt1(VS.80).aspx)

In your case, you won't need the "appliesTo" attribute of the assemblyBinding config. You just need something like:

<runtime>

<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">

<dependentAssembly>

<assemblyIdentity name="YourAssembly" publicKeyToken="AAAAAAAAAAAA" culture="neutral"/>

<bindingRedirect oldVersion="0.0.0.0-5.2.1.0" newVersion="5.0.8.1"/>

</dependentAssembly>

</assemblyBinding>

</runtime>

If they have the same version number the answer is you can't. If you attempt to load an assembly that has the same full assembly name (name, version, key) as a GAC'd assembly the CLR will pick the GAC'd assembly every single time. However, you can set the DEVPATH to force load an assembly, see [link text](http://msdn.microsoft.com/en-us/library/cskzh7h6.aspx). This doesn't answer this question since it only meant for development use and even then not really recommended as it doesn't mirror production usage.

What is the advantage of anonymous method? Or explain the power of anonymous method or Why anonymous method is introduced in .NET framework?

By using anonymous methods, you reduce the coding overhead in instantiating delegates by eliminating the need to create a separate method.

For example, specifying a code block in the place of a delegate can be useful in a situation when having to create a method might seem an unnecessary overhead. A good example would be when launching a new thread. This class creates a thread and also contains the code that the thread executes, without the need for creating an additional method for the delegate.

void StartThread()

{

System.Threading.Thread t1 = new System.Threading.Thread

(delegate()

{

System.Console.Write("Hello, ");

System.Console.WriteLine("World!");

});

t1.Start();

}

How many threads that one can create in thread pool?

Discuss thread safety of static members.

You’ve 10 million records to show in a data grid. These 10 million records are fetched using a background thread. How do you efficiently show in the grid? Use paging/data virtualization technique.

## Why Use Multiple Threads in the User Interface?

When you think of multithreading, some tasks immediately come to mind, such as data calculations, database queries, and input gathering. These activities, most often thought of as "background" tasks, do not directly involve the user interface or window management. Does this mean that there are no places that you can incorporate threads in your user interface? Certainly not. In general, if there are places in your application that do not need to be serialized, you can add threads. For instance, it does not make sense to distribute input to threads because all input is implicitly serialized by USER. It also does not make sense to distribute output to threads because output devices, such as a printer, are inherently single-threaded.

### Where?

Let's say you have an application that creates one window that displays current stock quotes and another window that allows you to enter requests for buying or selling stock. Within your application, you have two discrete tasks to complete that do not depend upon one another for information: (1) Displaying current stock quotes and (2) Placing a buy or sell order. These tasks do not affect each other in any way. One way to improve the performance of your application would be to create a thread for each window, thereby creating a more modular application.

Another situation in which multiple threads would benefit the user interface would be in an application in which one window displays a spinning cube and another window receives input from the user on how fast to spin the cube, the coordinates of the cube, and its angle. (Since we're just dreaming this up, let's make it three-dimensional.) You can fairly cleanly break down the tasks each window needs to complete—one window displays the spinning cube while the other window is gathering input. The big difference between this example and the previous one is that here you need to have one thread communicate some information (the new speed, coordinates, or angle) to the other thread. The first thread can continue to display the cube while the second thread is gathering new input. In other words, you use one thread for input and one for output.

### Where Not?

Because I like to be as evenhanded as possible, let me give you a couple of examples of situations in which multiple threads would be a detriment to the user interface. Let's say you have an application that relies upon getting some type of input before it can continue processing, such as a dialog box that accepts logon information. Would it be a good idea to create a thread to gather that input? Probably not. If you are creating a thread to accomplish something that needs to be synchronous, you are making things needlessly complex for almost no benefit, plus there are costs to adding threads, which I will discuss in the next section.

Another poor multithreading candidate is creating a thread for a nonreentrant window or dialog box. If this window has some global data, and if another thread calls it more than once, your global data can become corrupted. You can work around this problem by ensuring that you keep your data on a per-thread basis rather than a per-process basis. For example, let's say you have a dialog box that you use to open a file, but the data you keep in the thread for the pointer to file or the filename is kept in global data, rather than per-thread data. In this case, if the dialog box is called twice, the pointer and the filename will become corrupted and, when you try to access the file via the pointer in the first thread, you will be accessing the wrong file. This is not the functionality you want. Bear in mind that this situation is not specifically a USER issue—it is a general Windows NT multithreading issue. In any Windows NT-based application, you need to take care with data that you use on a per-thread basis.

What are the suggested theoretical limits of each generation?

250Kb for Gen 0, 2Mb for Gen 1 and 10Mb for Gen 2. The thresholds are automatically adjusted dynamically according to actual program allocations. If very little memory is being freed in Gen 0 and survives to Gen 1, the threshold is expanded and likewise it is maintained.

Why GC in .NET uses a **three-generation approach**?

What happens internally when you define a delegate type?

What are the differences between weak reference, weak event handlers, weak objects?

How do you design a component that uses two components with different versions but same assembly name? Or how do you solve the problem of “Found conflicts between different versions of the same dependent assembly”?

Write a recursive function that calculates multiplication of a and b.

Write an API to count number of words in a string. Also, find out the number of distinct words.

How do you delete a node from a singly linked list.

Implement your linked list object in C#.

In the UI thread, you want to get return type of method but the method should execute in a different thread? How can we achieve it?

Implement your own Backgroundworker like the Backgroundworker object in .NET.

You've a 1 million data in a collection object. How do you find the sum - using a thread vs not using a

thread? Justify your answer.

Give a real use of operator overloading of = operator. Hint: Deep cloning of object

Scenario based design pattern question. Strategy pattern, Abstract Factory

WCF:

**What are the advantages of hosting WCF Services in IIS as compared to self-hosting?**

**Which are the various programming approaches for WCF?**

**What is one-way operation?**

**Can you explain duplex contracts in WCF?**

**How can we host a service on two different protocols on a single server?**

**How can we use MSMQ bindings in WCF?**

**Can you explain transactions in WCF?**

**What different transaction isolation levels provided in WCF?**

**Can we do transactions using MSMQ?**

**Can we have two-way communications in MSMQ?**

**What are Volatile queues?**

**What are Dead letter queues?**

**What is a poison message?**

**What are the important principles of SOA (Service oriented Architecture)?**

WCF is based on SOA. All big companies are playing big bets on SOA. So how can Microsoft remain behind? So in order to implement SOA architecture easily you need to use WCF.  
SOA is based on four important concepts:-

**• Boundaries are well defined**

In SOA, everything is formalized. The client who is consuming the service does not need to know how the implementation of the service is done. If you look at some old methodologies of communication like DCOM. Any changes at server level the client also has to change. Therefore, the server and client implementation was so much bound that changes need to be done at all places. In SOA, the rule is if you do enhancement you do not need to change anything at the client. SOA based application only understands that there is an end point, contract, and bindings.

**• Services evolve**

Change is the law of nature and services will evolve. In SOA, services can be versioned and you can host those services in new ends. For instance, you have a service called as “Search Tickets (Ticket Number) “which gives details based on Ticket Number and its exposed on end point “ep1”. Tomorrow you want make your Search Tickets service more useful by also providing an extra option of allowing him to search by passenger name. Therefore, you just declare a new end “ep2” with service “Search Tickets (Ticket Number, Passenger Name)”. So the client who is consuming the service at end ep1 continues and at the other end, we have evolved our service by adding new ends ep2.

**• Services share only schemas and contracts**

Services use Schemas to represent data and contracts to understand behavior. They do not use language dependent types or classes in order to understand data and behavior. XML is used to define schemas and contracts. Due to this, there is not heavy coupling between environments.

**• Service compatibility is policy based**

Policy describes the capabilities of the system. Depending on policies, the service can degrade to match the service for the client. For instance your service needs to be hosted for two types of client one which uses Remoting as the communication methodology while other client uses DCOM. An ideal SOA service can cater to both of them according to there communication policies.

**How many ServiceContracts can a WCF Service have? Specifically, since a ServiceContract is an Attribute to an Interface, how many interfaces can I code into one WCF web service? Is it a one-to-one? Does it make sense to separate the contracts across multiple web services?**

WCF services can have multiple endpoints, each of which can implement a different service contract.

For example, you could have a service declared as follows:

[ServiceBehavior(Namespace = "DemoService")]

public class DemoService : IDemoService, IDoNothingService

Which would have configuration along these lines:

<service name="DemoService" behaviorConfiguration="Debugging">

<host>

<baseAddresses>

<add baseAddress = "http://localhost/DemoService.svc" />

</baseAddresses>

</host>

<endpoint

address =""

binding="customBinding"

bindingConfiguration="InsecureCustom"

bindingNamespace="http://schemas.com/Demo" contract="IDemoService"/>

<endpoint

address =""

binding="customBinding"

bindingConfiguration="InsecureCustom"

bindingNamespace="http://schemas.com/Demo" contract="IDoNothingService"/>

</service>

Hope that helps, but if you were after the theoretical maximum interfaces you can have for a service I suspect its some crazily large multiple of 2

**Can a single WCF service offer multiple interfaces, and if so how would you express this in** *app.config***? I mean one services offering several Interfaces on one endpoint.**

First you need to be clear what a service is. Do you mean a single endpoint, or multiple endpoints in the same host?

Assuming you mean a single endpoint, then yes, but with a little work. An endpoint can only implement a single interface; so what you need to do is combine all the interfaces you want to implement into a single interface

public interface IMyInterface : IInterface1, IInterface2

and then implement them all inside your implementation class. What you cannot do is have multiple interfaces and multiple implementations magically merge into a single endpoint.

**If we've a series of numbers say from 1 to 100 and one number is missing. How do you find the number?**

**Suppose you have a base class A. B and C are derived from class A. Class B also related with class A with a has-a (aggregation) relationship.**

**Class A has virtual method say print(). This Print() method is overridden in class B and C? Show this in UML class diagram. Write the skeleton code**

**in C# showing the implementation of this class diagram.**

**Suppose you have a base class A. B and C are derived from class A. Class A has virtual method say print(). This Print() method is overridden in**

**class B and C? Explain how runtime polymorhism exists with virtual table (vtable).**

**Suppose you get a runtime error in constructor of class A. How do you inform consumer of class A that error has occurred? Explain the advantages of**

**each one. What should be the most optimised solution/best practice?**

**Write a program for calculating the factorial of a number n? Also, solve this problem using recursive technique so that solution is optimized one.**

**What is the difference between Mutex and Semaphore? What would you use to synchronize when you have 1. two threads and 2. Four threads?**

**What is a critical section?**

**What are the different hash code algortithms that ship with .NET? How to hash a text file?**

**Write code for below requirement.**

**"You have class A. A has to notify another class B every 1 sec about the time change. B has to get the changed time from A and then print the time."**

**What is the difference between API, system call and function/method?**

**Write a function static string Space(string s) {...} that returns the string with added space between each character. ex. "FRED" should be obtained as**

**"F R E D".**

**1. Tell me about yourself**

**2. Tell briefly about your projects. Explain in pen and paper.**

**5. Which namespace is used for SQL objects?**

**6. How to establish a database connection?**

**7. What is ADO.NET? Tell which control in .NET to be used for displaying 10000 records/rows of data? How to display them efficiently such a**

**huge number of records?**

**8. What is remoting? In which situation is it used? Which feature is introduced in .NET 3.5 to serve the purpose?**

**11.What is a collection? Have you used custom collection? Suppose you have a custom collection say a collection of person objects. You want to sort**

**the collection with person age. How to achieve that? Also, suppose you want to loop through the collection using for each loop? How to achieve**

**this in a custom collection?**

**How do you unit test a private method? What are the different ways?**

**googStatic/instance valriables vs local variables**

**SQL query practise them**

**Nullable types**

**out and ref parameter, params keyword, etc.**

**operator overloading**

**What are static variables. explain static keyword.**

**multithreading with static variables.**

**Runtime polymorphism with interfaces and overriding.**

**why string is immutable in .net?**

**www.WiZiQ.com/classes/**

**•What does int? mean? Explain the relationship with Nullable.**

**Hard questions:**

**•Explain the following snippet of code: 'from x in collection select new { x.Foo }'. What is the compiler doing? What is the CLR executing?**

**•Explain the "yield" keyword. What is the compiler doing internally?**

**•Async pattern**

**What is the need for inner class?**

**What is parametric polymorphism?**

**2.How would you define an Anonymous Delegates and methods?**

**3.How does an Enumerator (IEnumerable & IEnumerator) work ? How does an Iterator work in C# 2.0**

**Memory leaks in .NET (Observer pattern) and the Dispose pattern. How do you find and solve performance problems?**

**Why is catch(Exception) almost always a bad idea?**

**?**

**Explain what is a delegate and show an example of its use.**

**What is the difference between a BeginInvoke and an Invoke?**

**What are the uses of delegates in c#? Ans: Avoids cross thread exception, Helps event handling**

**How would you define a new custom event?**

**Explain different ways of calling methods asynchronously.**

**Can attributes be placed on specific parameters to a method? Why is this useful?**

**What is the difference between typeof(foo) and myFoo.GetType()?**

**Explain what’s happening in the first constructor: public class c{ public c(string a) : this() {;}; public c() {;} } How is this construct useful?**

**ADO.net and SQL server**

**=======================**

**What is transaction? Why and when do we need it?**

**What is the difference between stored procedure and function?**

**difference between datasource and database?**

**How do we deploy database in the same form from developer to production(client) side? ---mdf file and ldf file**

**performance tuning, locking, indexing**

**Written Test(.Net) - Goldman sachs**

**I dont remember the exact questions but i can give u brief idea what were the questions like.**

**I have given the test as the .Net guy.**

**As they are using JAVA for the rest of the project, but they are using .Net for Front end (Web , windows application).**

**Few concepts or questions or toppics**

**-As the reuirement was for UI, front end so all the questions were revolving around that.**

**-Dataset**

**-DataAdaptor**

**-Fill()**

**-Outproc(SQL Server , state server)**

**-DebugListner**

**-Trace**

**-Application run mode (define- Debug/Trace)**

**-Gridview (ButtonField,CommandField)**

**-Viewstate**

**-Property to get a column from Dataset.**

**-Prepare well on Threading**

**-Prepare well on Delegates**

**-Design Patterns**

**-BinaryTextWriter**

**-How to get the root node of a dataset.**

**Write an algortithm to convert a number into word. For example: 12345 should be displayed as**

**"Twelve thousand three hundred forty five". Design the class and show it in UML class daigram and write the code.**

**What is the difference between class and object. Explain with real example. Is a telephone receiver set an object? If so, why?**

**How can we determine the mouse coordinates in C#.NET without using mouse events?**

**What is the difference bewteen component and control? Classify panel, textbox, dataSource as component and control.**

**Tell the inheritance hieracrchy of Form class.**

**What is GDI?How it works? Where does it paint?**

**Puzzle:**

**Explain product/software life cycle.**

**Functional and nonfunctional requirements.**

**MVP/MVC, Design principles, UML relationships, Design patterns, Cohesion, coupling, dependency inversion, etc**

**High level design with component diagram.**

**What are the basic principles/guidelines of software architecture?**

**Draw the sequence diagram for intercation with timer object.**

**How the CLR Creates Runtime Objects?**

**How do I obtain return types of multi cast delegate?**

**What is the use of fixed statement?**

The fixed statement sets a pointer to a managed variable and “pins” that variable during the execution of statement.

Without fixed, pointers to managed variables would be of little use since garbage collection could relocate the variables unpredictably. (In fact, the C# compiler will not allow you to set a pointer to a managed variable except in a fixed statement.)

Eg:Class A

{

public int i;

}

A objA = new A; // A is a .net managed type

fixed(int \*pt = &objA.i) // use fixed while using pointers with managed

// variables

{

\*pt=45; // in this block use the pointer the way u want

}

**Is there an equivalent of exit() or quiting a C#.NET application?**

Yes, you can use System.Environment.Exit(int exitCode) to exit the application or Application.Exit() if it’s a Windows Forms app.

**What optimization does the C# compiler perform when you use the /optimize+compiler option?**

The following is a response from a developer on the C# compiler team: We get rid of unused locals (i.e., locals that are never read, even if assigned). We get rid of unreachable code. We get rid of try-catch with an empty try. We get rid of try-finally with an empty try. We get rid of try-finally with an empty finally. We optimize branches over branches: gotoif A, lab1 goto lab2: lab1: turns into: gotoif !A, lab2 lab1: We optimize branches to ret, branches to next instruction, and branches to branches.

**59. Does C# support multiple inheritance?**

No, use interfaces instead.

**60. Is goto statement supported in C#?How about Java?**

Gotos are supported in C# to the fullest. In Java goto is a reserved keyword that provides absolutely no functionality.

**61. What happens when you encounter a continue statement inside for loop?**

The code for the rest of the loop is ignored, the control is transferred back to the beginning of the loop.

**General Questions**

1. **Does C# support multiple-inheritance?**   
   No.


5. **Explain the three services model commonly known as a three-tier application.**Presentation (UI), Business (logic and underlying code) and Data (from storage or other sources).
6. **Is it possible to inline assembly or IL in C# code?**

- No.

1. **Is it possible to have different access modifiers on the get/set methods of a property?** - No. The access modifier on a property applies to both its get and set accessors. What you need to do if you want them to be different is make the property read-only (by only providing a get accessor) and create a private/internal set method that is separate from the property.
2. **Is it possible to have a static indexer in C#?** - No. Static indexers are not allowed in C#.
3. **I was trying to use an “out int” parameter in one of my functions. How should I declare the variable that I am passing to it?** - You should declare the variable as an int, but when you pass it in you must specify it as ‘out’, like the following: int i; foo(out i); where foo is declared as follows: [return-type] foo(out int o) { }

**XML Documentation Questions**

1. **Is XML case-sensitive?**Yes.
2. **What’s the difference between // comments, /\* \*/ comments and /// comments?**Single-line comments, multi-line comments, and XML documentation comments.
3. **How do you generate documentation from the C# file commented properly with a command-line compiler?**Compile it with the /doc switch.

**Debugging and Testing Questions**

**What debugging tools come with the .NET SDK?**  
1.   CorDBG – command-line debugger.  To use CorDbg, you must compile the original C# file using the /debug switch.   
2.   DbgCLR – graphic debugger.  Visual Studio .NET uses the DbgCLR. 

**What does assert() method do?**In debug compilation, assert takes in a Boolean condition as a parameter, and shows the error dialog if the condition is false.  The program proceeds without any interruption if the condition is true. 

**Why do we use the Debug class?**

To produce messages that helps you to monitor the program execution sequence, to detect malfunctions, or to provide performance measurement information.

**Why do we use the Trace class?**

To instrument release builds. Instrumentation allows you to monitor the health of your application running in real-life settings. Tracing helps you isolate problems and fix them without disturbing a running system.

**What’s the difference between the Debug class and Trace class?**Documentation looks the same.  Use Debug class for debug builds, use Trace class for both debug and release builds. 

|  |  |
| --- | --- |
| **Debug** | **Trace** |
| If you use methods in the **Debug** class to print debugging information and check your logic with assertions, you can make your code more robust without impacting the performance and code size of your shipping product. | You can use the properties and methods in the **Trace** class to instrument release builds. Instrumentation allows you to monitor the health of your application running in real-life settings. Tracing helps you isolate problems and fix them without disturbing a running system. |
| In Visual Studio 2005 projects, creating a debug build enables **Debug**. [Debug](http://msdn.microsoft.com/en-us/library/system.diagnostics.debug(VS.71).aspx) is disabled in release builds by default, so no executable code is generated for **Debug** methods. | In Visual Studio 2005 projects, [Trace](http://msdn.microsoft.com/en-us/library/system.diagnostics.trace(VS.80).aspx) is enabled by default for both release and debug builds. Therefore, code is generated for all **Trace** methods in both release and debug builds. This allows an end user to turn on tracing to help identify the problem without the program having to be recompiled. |
| To enable debugging in C#, add the **/d:DEBUG** flag to the compiler command line when you compile your code, or add **#define DEBUG** to the top of your file | To enable debugging in C#, add the **/d:TRACE** flag to the compiler command line when you compile your code, or you can add **#define TRACE** to the top of your file. |
|  |  |
|  |  |

**Where does the Debug class produce output?**

By default, the messages that the **Debug** class produces appear in the Output window of the Visual Studio Integrated Development Environment (IDE).

**Can we redirect output from the Debug class to destinations other than the Output window?**

Yes. You can direct output from the **Debug** class to destinations other than the Output window. The **Debug** class has a collection named **Listeners** that includes **Listener** objects.  
  
Each **Listener** object monitors **Debug** output and directs the output to a specified target.   
  
Each **Listener** in the **Listener** collection receives any output that the **Debug** class generates. Use the **TextWriterTraceListener** class to define **Listener** objects. You can specify the target for a **TextWriterTraceListener** class through its constructor.   
  
Some possible output targets include the following:

* The Console window by using the **System.Console.Out** property.
* A text (.txt) file by using the **System.IO.File.CreateText("FileName.txt")** statement.

After you create a **TextWriterTraceListener** object, you must add the object to the **Debug.Listeners** collection to receive Debug output.

**Why are there five tracing levels in System.Diagnostics.TraceSwitcher?**The tracing dumps can be quite verbose.  For applications that are constantly running you run the risk of overloading the machine and the hard drive.  Five levels range from None to Verbose, allowing you to fine-tune the tracing activities. 

**Where is the output of TextWriterTraceListener redirected?**   
To the Console or a text file depending on the parameter passed to the constructor. 

**What is the output of following code when configuration type is Debug and Release?**

using System.Diagnostics;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

TextWriterTraceListener listener = new TextWriterTraceListener(System.IO.File.CreateText(@"C:\Data\Visual Studio 2008\Projects\ConsoleApplication1\ConsoleApplication1\TextFile1.txt"));

Debug.Listeners.Add(listener);

Debug.WriteLine("Start of debugging!!!");

Debug.Indent();

Debug.Flush();

Trace.WriteLine("-----Trace output starting-----");

Trace.Listeners.Add(listener);

Trace.WriteLine("-----Trace output ending------");

Debug.Unindent();

Debug.WriteLine("End of debugging!!!");

Trace.Flush();

}

}

}

Output – Debug mode

Start of debugging!!!

-----Trace output starting-----

-----Trace output ending------

-----Trace output ending------

End of debugging!!!

End of debugging!!!

Output – Release mode

-----Trace output starting-----

-----Trace output ending------

-----Trace output ending------

**What are three test cases you should go through in unit testing?**1.       Positive test cases (correct data, correct output).  
2.       Negative test cases (broken or missing data, proper handling).  
3.       Exception test cases (exceptions are thrown and caught properly). 

**Can you change the value of a variable while debugging a C# application?**Yes.  If you are debugging via Visual Studio.NET, just go to Immediate window.

**What is the difference between debug and release mode in .net?**

It is not well documented, here's what I know.  The compiler emits an instance of the System.Diagnostics.DebuggableAttribute.  In the debug version, the IsJitOptimizerEnabled property is True, in the release version it is False.  You can see this attribute in the assembly manifest with ildasm.exe  
  
The JIT compiler uses this attribute to disable optimizations that would make debugging difficult.  The ones that move code around like loop-invariant hoisting.  In selected cases, this can make a big difference in performance.  Not usually though.  
  
Mapping breakpoints to execution addresses is the job of the debugger.  It uses the .pdb file and info generated by the JIT compiler that provides the IL instruction to code address mapping.  If you would write your own debugger, you'd use ICorDebugCode::GetILToNativeMapping().

***C# struct/class Differences***

|  |  |  |
| --- | --- | --- |
| Class vs. Struct | | |
| **Feature** | **Class** | **Struct** |
| Events are locked? | Events declared in a class have their += and -= access automatically locked via a lock(this) to make them thread safe (static events are locked on the typeof the class | Events declared in a struct do *not* have their += and -= access automatically locked. A lock(this) for a struct would not work since you can only lock on a reference type expression. |
| Exist on stack or heap? | Reference type local instances are allocated on the heap. | Value type local instances are allocated on the stack. |
| Can cause garbage collection? | Creating a reference type instance can cause garbage collection. | Creating a struct instance cannot cause a garbage collection (unless the constructor directly or indirectly creates a reference type instance) |
| Always have a default constructor? | class DefaultConstructor  {  static void Eg()  {  Direct yes = new Direct(); *// always compiles ok*  InDirect maybe = new InDirect(); *// compiles if c'tor exists and is accessible*  *//...*  }  }  A class might not be since all its constructors could be private.  class NonInstantiable  {  private NonInstantiable() *// ok*  {  }  } | A struct *always* has a built-in public default constructor.  This means that a struct is *always* instantiable.  struct Direct  {  private Direct() *// compile-time error*  {  }  } |
| Default construction triggers static constructor? | class NotTriggered  {  static void Main()  {  Direct local = new Direct();  }  } | A structs static constructor is *not* triggered by calling the structs default constructor. It is for a class.  struct Direct  {  static Direct()  {  Console.WriteLine("This is not written");  }  } |
| Can be null? | class Nullness  {  static void Eg(Direct s, Indirect c)  {  if (s == null) ... *// compile-time error*  if (c == null) ... *// compiles ok*  }  } | A struct instance cannot be null. |
| Use with the as operator? | class Fragment  {  static void Eg(Direct s, Indirect c)  {  Direct no = s as Direct; *// compile-time error*  InDirect yes = c as InDirect; *// compiles ok*  *//...*  }  } | A struct type cannot be the right hand side operand of the as operator. |
| Can be locked? | class LockStatement  {  static void Eg(Direct s, InDirect c)  {  lock(s) { ... } *// compile-time error*  lock(c) { ... } *// compiles ok*  }  } | A struct type expression cannot be the operand of a lock statement. |
| Can have a destructor? | class InDirect  {  ~InDirect() {} *// compiles ok*  }  And the CIL for ~Indirect() looks like this:  .method family hidebysig virtual instance void  Finalize() cil managed  {  *// ...*  } *// end of method Indirect::Finalize* | A struct cannot have a destructor. A destructor is just an override of object.Finalize in disguise, and structs, being value types, are not subject to garabge collection.  struct Direct  {  ~Direct() {} *// compile-time error*  } |
| Inheritance differences? | * It is not implicitly sealed. * It can be abstract. * An abstract class with no explicit base class can call : base() in it’s constructor. * It can extend another class. * It can declare protected members (eg fields, nested types). * It can declare abstract function members * It can declare virtual function members. * It can declare sealed function members * It can declare override function members. | * A struct is implicitly sealed * A struct can't be abstract * A struct can't call : base() in its constructor * a struct can't extend another class * a struct can't declare protected members (eg fields, nested types) * a struct can't declare abstract function members. * a struct can't declare virtual function members. * a struct can't declare sealed function members. * a struct can't declare override function members, a class can. The one exception to this rule is that a struct can override the virtual methods of System.Object, viz, Equals(), and GetHashCode(), and ToString(). |

**Meaning of this?**

In a class, *this* is classified as a value, and thus cannot appear on the left hand side of an assignment, or be used as a ref/out parameter. For example:

class Indirect

{

*//...*

public void Method(Indirect that)

{

RefParameter(ref this); *// compile-time error*

OutParameter(out this); *// compile-time error*

this = that; *// compile-time error*

}

*//...*

}

In a struct, *this* is classified as an out parameter in a constructor and as a ref parameter in all other function members. Thus it is possible to modify the entire structure by assigning to this or passing this as a ref/out parameter. For example:

struct Direct

{

*//...*

public void Reassign(Direct that)

{

RefParameter(ref this); *// compiles ok*

OutParameter(out this); *// compiles ok*

this = that; *// compiles ok*

}

*//...*

}

Furthermore, you can reassign a whole struct even when the struct contains readonly fields!

struct Direct

{

public Direct(int value)

{

Field = value;

}

public void Reassign(Direct that)

{

RefParameter(ref this); *// compiles ok*

OutParameter(out this); *// compiles ok*

this = that; *// compiles ok*

}

public readonly int Field;

}

class Show

{

static void Main()

{

Direct s = new Direct(42);

Console.WriteLine(s.Field); *// writes 42*

s.Reassign(new Direct(24));

Console.WriteLine(s.Field); *// writes 24*

}

}

Note however that when you call a method on a readonly value-type field, the method call is made on a *copy* of the field.

struct Direct

{

*// as above*

}

class Caller

{

public void Method()

{

Console.WriteLine(d.Field); *// writes 42*

d.Reassign(new Direct(24));

Console.WriteLine(d.Field); *// writes 42!*

}

private readonly Direct d = new Direct(42);

}

class Show

{

static void Main()

{

Caller c = new Caller();

c.Method();

}

}

**Default field layout?**

The default [StructLayout] attribute (which lives in the System.Runtime.InteropServices namespace) for a struct is LayoutKind.Sequential whereas the default StructLayout for a class is LayoutKind.Auto. (And yes, despite its name you can tag a class with the StructLayout attribute.) In other words the CIL for this:

public struct Direct

{

*//...*

}

looks like this:

.class public sequential ansi sealed beforefieldinit Direct

extends [mscorlib]System.ValueType

{

*//...*

}

whereas the CIL for this:

public sealed class InDirect

{

*//...*

}

looks like this:

.class public auto ansi sealed beforefieldinit Indirect

extends [mscorlib]System.Object

{

*//...*

}

**Can be a volatile field?**

You can't declare a user-defined struct type as a volatile field but you can declare a user-defined class type as a volatile field.

class Bad

{

private volatile Direct field; *// compile-time error*

}

class Good

{

private volatile Indirect field; *// compiles ok*

}

**Can have synchronized methods?**

You can't use the [MethodImpl(MethodImplOptions.Synchronized)] attribute on methods of a struct type (if you call the method you get a runtime TypeLoadException) whereas you can use the [MethodImpl(MethodImplOptions.Synchronized)] attribute on methods of a class type.

using System.Runtime.CompilerServices;

class Indirect

{

[MethodImpl(MethodImplOptions.Synchronized)] *// compiles and runs ok*

public void Method()

{

*//...*

}

}

struct Direct

{

[MethodImpl(MethodImplOptions.Synchronized)] *// compiles ok, runtime TypeLoadException*

public void Method()

{

*//...*

}

}

**Can be pointed to?**

Clause 25.2 of the C# standard defines an unmanaged type as any type that isn't a reference type and doesn't contain reference-type fields at any level of nesting. That is, one of the following:

* Any simple value type (11.1.3, eg byte, int, long, double, bool, etc).
* Any enum type.
* Any pointer type.
* Any user-defined struct-type that contains fields of unmanaged types only.

You can *never* take the address of a instance of a type that is not unmanaged (a fixed variable 25.3).

class Bad

{

static void Main()

{

Indirect variable = new Indirect();

unsafe

{

fixed(Indirect \* ptr = &variable) *// compile-time error*

{

*//...*

}

}

}

}

If you want to fix an unmanaged instance you have to do so by fixing it through an unmanaged field. For example:

class Indirect

{

public int fixHandle;

}

class Bad

{

static void Main()

{

Indirect variable = new Indirect();

unsafe

{

fixed(int \* ptr = &variable.fixHandle) *// compiles ok*

{

*//...*

}

}

}

}

In contrast, you can (nearly) always take the address of an unmanaged instance.

struct Direct

{

*// no reference fields at any level of nesting*

}

class SimpleCase

{

static void Main()

{

Direct variable = new Direct();

unsafe

{

Direct \* ptr = &variable; *// compiles ok*

*//...*

}

}

}

However, you have to take the address inside a fixed statement if the variable is moveable (subject to relocation by the garbage collector, see 25.3 and example above). Also, you can never take the address of a volatile field.

So, in summary, you can *never* create a pointer to a class type but you sometimes create a pointer to a struct type.

**Can be stackalloc'd?**

You can only use stackalloc on unmanaged types. Hence you can never use stackalloc on class types. For example:

class Indirect

{

*//...*

}

class Bad

{

static void Main()

{

unsafe

{

Indirect \* array = stackalloc Indirect[42]; *// compile-time error*

*//...*

}

}

}

Where as you can use stackalloc on struct types that are unmanaged. For example:

struct Direct

{

*// no reference fields at any level of nesting*

}

class Good

{

static void Main()

{

unsafe

{

Direct \* array = stackalloc Direct[42]; *// compiles ok*

*//...*

}

}

}

**Can be sizeof'd?**

You can only use sizeof on unmanaged types. Hence you can never use sizeof on class types. For example:

class Indirect

{

*//...*

}

class Bad

{

static void Main()

{

unsafe

{

int size = sizeof(Indirect); *// compile-time error*

*//...*

}

}

}

Where as you can use sizeof on struct types that are unmanaged. For example:

struct Direct

{

*// no reference fields at any level of nesting*

}

class Good

{

static void Main()

{

unsafe

{

int size = sizeof(Direct); *// compiles ok*

*//...*

}

}

}

**How to initialize fields?**

The fields of a class have a default initialization to zero/false/null. The fields of a struct have no default value.

struct Direct

{

public int Field;

}

class Indirect

{

public Indirect()

{

}

*//...*

public int Field;

}

class Defaults

{

static void Main()

{

Direct s;

Console.WriteLine(s.Field); *// compile-time error*

Indirect c = new Indirect();

Console.WriteLine(c.Field); *// compiles ok*

}

}

You can initialize fields in a class at their point of declaration. For example:

class Indirect

{

*//...*

private int field = 42;

}

You can't do this for fields in a struct. For example:

struct Direct

{

*//...*

private int field = 42; *// compile-time error*

}

Fields in a struct *have* to be initialized in a constructor. For example:

struct Direct

{

public Direct(int value)

{

field = value;

}

*//...*

private int field; *// compiles ok*

}

Also, the definite assignment rules of a struct are tracked on an individual field basis. This means you can bypass initialization and "assign" the fields of a struct one a time. For example:

struct Direct

{

public int X, Y;

}

class Example

{

static void Main()

{

Direct d;

d.X = 42;

Console.WriteLine(d.X); *// compiles ok*

Console.WriteLine(d.Y); *// compile-time error*

}

}

**Equals behavior?**

classes inherit Object.Equals which implements identity equality whereas structs inherit ValueType.Equals which implements value equality.

using System.Diagnostics;

struct Direct

{

public Direct(int value)

{

field = value;

}

private int field;

}

class Indirect

{

public Indirect(int value)

{

field = value;

}

private int field;

}

class EqualsBehavior

{

static void Main()

{

Direct s1 = new Direct(42);

Direct s2 = new Direct(42);

Indirect c1 = new Indirect(42);

Indirect c2 = new Indirect(42);

bool structEquality = s1.Equals(s2);

bool classIdentity = !c1.Equals(c2);

Debug.Assert(structEquality);

Debug.Assert(classIdentity);

}

}

Overriding Equals for structs should be faster because it can avoid reflection and boxing.

struct Direct

{

public Direct(int value)

{

field = value;

}

public override bool Equals(object other)

{

return other is Direct && Equals((Direct)other);

}

public static bool operator==(Direct lhs, Direct rhs)

{

return lhs.Equals(rhs);

}

*//...*

private bool Equals(Direct other)

{

return field == other.field;

}

private int field;

}

**6.1 How do I do a case-insensitive string comparison?**

Use the String.Compare function. Its third parameter is a boolean which specifies whether case should be ignored or not.

"fred" == "Fred" // false

System.String.Compare( "fred", "Fred", true ) // true

**6.2 Does C# support a variable number of arguments?**

Yes, using the params keyword. The arguments are specified as a list of arguments of a specific type, e.g. int. For ultimate flexibility, the type can be object. The standard example of a method which uses this approach is System.Console.WriteLine().

**What types of object can I throw as exceptions?**

Only instances of the System.Exception classes, or classes derived from System.Exception. This is in sharp contrast with C++ where instances of almost any type can be thrown.

**What are anonymous methods?**

Answer:

In versions of C# previous to 2.0, the only way to declare a delegate was to use named methods. C# 2.0 introduces anonymous methods.

Creating anonymous methods is essentially a way to pass a code block as a delegate parameter. For example:

// Create a delegate instance

delegate void Del(int x);

// Instantiate the delegate using an anonymous method

Del d = delegate(int k) { /\* ... \*/ };

By using anonymous methods, you reduce the coding overhead in instantiating delegates by eliminating the need to create a separate method.

For example, specifying a code block in the place of a delegate can be useful in a situation when having to create a method might seem an unnecessary overhead.

**What is Partial Class?**

Answer:

Partial classes give you the ability to split a single class into more than one source code (.cs) file.

**Is it possible to assign null value to an integer variable in c#?**

Answer:

Yes.

You can define nullable integer variable using following notation

DataTime? dt = null;

int? i = null;

bool? b = null;

**Is it possible to inherit multiple classes in c#?**

Answer:

No.

C# does not support multiple inheritances but you can achieve multi-level inheritance in c#.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Attributes**

**What are attributes?**

Attributes provide a powerful method of associating declarative information with C# code (types, methods, fields, properties). Once associated with a program entity, the attribute can be queried at run time and used in any number of ways.

If you have used C++, you are probably familiar with declarations that contain keywords, such as public and private, that provide additional information about class members. These keywords further define the behavior of class members by describing their accessibility to other classes. Because compilers are explicitly designed to recognize predefined keywords, you do not traditionally have the opportunity to create your own. The common language runtime, however, allows you to add keyword-like descriptive declarations, called attributes, to annotate programming elements such as types, fields, methods, and properties.

When you compile your code for the runtime, it is converted into Microsoft intermediate language (MSIL) and placed inside a portable executable (PE) file along with metadata generated by the compiler. Attributes allow you to place extra descriptive information into metadata that can be extracted using runtime reflection services.

**Why use attributes?**

The .NET Framework uses attributes for a variety of reasons and to address a number of issues.

* Attributes describe how to serialize data, specify characteristics that are used to enforce security, and
* Attributes can limit optimizations by the just-in-time (JIT) compiler so the code remains easy to debug.
* Attributes can also record the name of a file or the author of code, or control the visibility of controls and members during forms development.

**How do you apply an attribute to an element of your code?**

Use the following process to apply an attribute to an element of your code.

1. Define a new attribute or use an existing attribute by importing its namespace from the .NET Framework.
2. Initialize the attribute directly preceding the element that you want described, calling the attribute's constructor with the desired flag or information.

The attribute is emitted into metadata when you compile your code and is available to the common language runtime and any custom tool or application through the runtime reflection services.

By convention, all attribute names end with Attribute. However, several languages that target the runtime, such as Visual Basic and C#, do not require you to specify the full name of an attribute. For example, if you want to initialize [System.ObsoleteAttribute](http://msdn.microsoft.com/en-us/library/system.obsoleteattribute(v=VS.71).aspx), you only need to reference it as **Obsolete**.

The following code example shows how to declare **System.ObsoleteAttribute**, which marks code as obsolete. The string "Will be removed in next version" is passed to the attribute. This attribute causes a compiler warning that displays the passed string when code that the attribute describes is called.

using System;

public class MainApp

{

public static void Main()

{

//This generates a compile-time warning.

int MyInt = Add(2,2);

}

//Specify attributes between square brackets in C#.

//This attribute is applied only to the Add method.

[Obsolete("Will be removed in next version")]

public static int Add( int a, int b)

{

return (a+b);

}

}

**How do you apply an attribute at assembly level?**

If you want to apply an attribute at the assembly level, use the **Assembly** keyword. The following code shows the **AssemblyNameAttribute** applied at the assembly level.

using System.Reflection;

[assembly:AssemblyName("MyAssembly")]

When this attribute is applied, the string "MyAssembly" is placed in the assembly manifest in the metadata portion of the file. You can view the attribute either by using the [MSIL Disassembler (Ildasm.exe)](http://msdn.microsoft.com/en-us/library/f7dy01k1(v=VS.71).aspx) or by creating a custom program to retrieve the attribute.

**Reflection**

## What is .NET Reflection?

*Reflection is the ability to read metadata at runtime. .NET Framework’s Reflection API allows you to fetch Type (Assembly) information at runtime programmatically.* We can also achieve the late binding by using .NET Reflection. At runtime, Reflection mechanism uses the PE file to read the information about the assembly. Reflection enables you to use code that was not available at compile time.

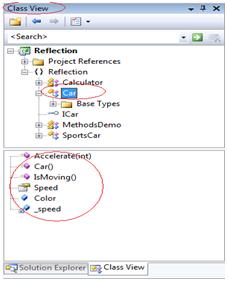
Reflection generally begins with a call to a method present on every object in the .NET framework: GetType. The GetType method is a member of the System.Object class, and the method returns an instance of System.Type. System.Type is the primary gateway to metadata. System.Type is actually derived from another important class for reflection: the MemeberInfo class from the System.Reflection namespace. MemberInfo is a base class for many other classes who describe the properties and methods of an object, including FieldInfo, MethodInfo, ConstructorInfo, ParameterInfo, and EventInfo among others. As you might suspect from thier names, you can use these classes to inspect different aspects of an object at runtime.

**What are the uses of reflection?**

.NET Reflection allows application to collect information about itself and also manipulate on itself. It can be used effectively

* To find all the types in an assembly and/or dynamically invoke methods in an assembly. This includes information about the type, properties, methods and events of an object.
* To invoke private methods of a type.
* To dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object and invoke its methods or access its fields and properties.
* To access attribute information using reflection.

Using reflection, you can get any kind of information which you can see in class viewer, for example information of methods, properties, field’s, and event’s of an object.



**Which namespace and class plays important role in reflection?**

System.Reflection namespace and System.Type class plays very important role in .NET Reflection, these two works together and allow you to reflect over many other aspects of a type.