

CODE SAMPLE: ITERATORS

```
//This example is based on:  
// http://msdn2.microsoft.com/en-us/library/0x6a29h6(VS.80).aspx  
  
using System;  
using System.Collections;  
using System.Collections.Generic;  
using System.Text;  
  
namespace Generic1  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            // int is the type argument  
            GenericList<int> list = new GenericList<int>();  
  
            for (int x = 0; x < 10; x++)  
            {  
                list.AddHead(x);  
            }  
  
            foreach (int i in list)  
            {  
                System.Console.Write(i + " ");  
            }  
            System.Console.WriteLine("\nDone\n");  
  
            // string is the type argument  
            GenericList<string> slist = new GenericList<string>();  
  
            for (int x = 0; x < 10; x++)  
            {  
                slist.AddHead(x.ToString());  
            }  
  
            foreach (string str in slist)  
            {  
                System.Console.Write(str + " ");  
            }  
            System.Console.WriteLine("\nDone");  
        }  
    }  
  
    // type parameter T in angle brackets  
    public class GenericList<T> : IEnumerable<T>  
    {  
        // The nested class is also generic on T since it uses T  
        private class Node  
        {  
            private Node next;  
            // T as private member data type  
            private T data;  
  
            // T used in non-generic constructor  
            public Node(T t)
```

```

    {
        next = null;
        data = t;
    }

    public Node Next
    {
        get { return next; }
        set { next = value; }
    }

    // T as return type of property
    public T Data
    {
        get { return data; }
        set { data = value; }
    }
}

private Node head;

// constructor
public GenericList()
{
    head = null;
}

// T as method parameter type:
public void AddHead(T t)
{
    Node n = new Node(t);
    n.Next = head;
    head = n;
}

//This is needed if supporting IEnumerable<T> since it derives from IEnumerable.
//It doesn't get called by the foreach loops.

System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
{
    return GetEnumerator();
}

public IEnumerator<T> GetEnumerator()
//Use IEnumerator<T> if supporting IEnumerable<T>
{
    Node current = head;

    while (current != null)
    {
        yield return current.Data;
        current = current.Next;
    }
}
}
}

```