

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ  
СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Практикум №1

з курсу «Сучасні технології розробки WEB-застосунків на платформі  
Microsoft.NET»

на тему: «Узагальнені типи (Generic) з підтримкою подій. Колекції»

Викладач:  
Бардін В.

Виконав:  
Хільчук А.В.  
студент 3 курсу  
групи ІП-14 ФІОТ

Київ-2023

## Практична робота №1

**Тема:** Узагальнені типи (Generic) з підтримкою подій. Колекції

**Завдання:**

1. Розробити клас власної узагальненої колекції, використовуючи стандартні інтерфейси колекцій із бібліотек System.Collections та System.Collections.Generic. Стандартні колекції при розробці власної не застосовувати. Для колекції передбачити методи внесення даних будь-якого типу, видалення, пошуку та ін. (відповідно до типу колекції).
2. Додати до класу власної узагальненої колекції підтримку подій та обробку виключних ситуацій.
3. Опис класу колекції та всіх необхідних для роботи з колекцією типів зберегти у динамічній бібліотеці.
4. Створити консольний додаток, в якому продемонструвати використання розробленої власної колекції, підписку на події колекції.

5	Динамічний масив	Див. List<T>	Збереження даних за допомогою динамічно зв'язаного списку
---	------------------	--------------	---

**Виконання:**

*Код програми:*

CustomList.cs:

```
using System;

using System.Collections;

using System.Collections.Generic;

namespace MyList

{

    public class CustomList<T> : IList<T>

    {

        private ListNode<T> _start;

        private ListNode<T> _end;

        private int _version;
```

```
private int _count;  
public int Count => _count;  
public bool IsReadOnly => false;
```

```
public event EventHandler<T> ItemAdded;  
public event EventHandler<T> ItemInserted;  
public event EventHandler<T> ItemRemoved;  
public event EventHandler<int> ItemSet;  
public event EventHandler Cleared;
```

```
public CustomList()  
{  
    _count = 0;  
    _start = null;  
    _end = null;  
    _version = 0;  
}
```

```
public void Clear()  
{  
    _end = null;  
    _start = null;  
    _count = 0;  
    _version++;  
    Cleared?.Invoke(this, EventArgs.Empty);  
}
```

```

public int IndexOf(T item)
{
    int i = 0;
    foreach (var a in this)
    {
        if (a.Equals(item)) return i;
        i++;
    }
    return -1;
}

public bool Contains(T item)
{
    foreach (var a in this)
    {
        if (a.Equals(item)) return true;
    }
    return false;
}

public void CopyTo(T[] array, int arrayIndex)
{
    if (array is null) throw new Exception("Array is null");
    if (array.Length - arrayIndex < Count) throw new Exception("Array
doesn't have enough space");

    int i = 0;
    foreach (var a in this)

```

```

        {
            array[arrayIndex + i] = a;
            i++;
        }
    }
    public void Add(T item)
    {
        AppendItem(item);

        ItemAdded?.Invoke(this, item);
    }

    public void Insert(int index, T item)
    {
        if (index < 0 || index > _count) throw new
ArgumentOutOfRangeException("Index out of range");

        if (Count == 0 || index == _count)
        {
            AppendItem(item);

            ItemInserted?.Invoke(this, item);
            return;
        }

        _count++;
        _version++;
    }

```

```

    if (index == 0)
    {
        var tmp = new ListNode<T>(item);
        tmp.Next = _start;
        _start = tmp;
        ItemInserted?.Invoke(this, item);

        return;
    }
    var prvs = _start;
    var curr = prvs.Next;
    for (int i = 1; i < _count - 1; i++)
    {
        if (i == index)
        {
            var tmp = new ListNode<T>(item);
            tmp.Next = curr;
            prvs.Next = tmp;
            ItemInserted?.Invoke(this, item);

            return;
        }
        prvs = curr;
        curr = curr.Next;
    }
}

```

```
public bool Remove(T item)
{
    if (Count == 0) return false;

    if (_start.Value.Equals(item))
    {
        _start = _start.Next;

        _count--;
        _version++;
        ItemRemoved?.Invoke(this, item);
        return true;
    }

    var prvs = _start;
    var curr = _start.Next;
    while (curr != _end)
    {
        if (curr.Value.Equals(item))
        {
            prvs.Next = curr.Next;

            _count--;
            _version++;
            ItemRemoved?.Invoke(this, item);
            return true;
        }
    }
}
```

```

    }

    prvs = curr;
    curr = curr.Next;
}

if (_end.Value.Equals(item))
{
    _end = prvs;
    _end.Next = null;

    _count--;
    _version++;
    ItemRemoved?.Invoke(this, item);
    return true;
}

return false;
}

public void RemoveAt(int index)
{
    if (index >= Count || index < 0) throw new
ArgumentOutOfRangeException("Argument was out of range");

    _count--;
    _version++;

```



```

if (index == 0)
{
    var tmp = _start.Value;
    _start = _start.Next;
    ItemRemoved?.Invoke(this, tmp);
    return;
}

var prvs = _start;
var curr = prvs.Next;
for (int i = 1; i < _count; i++)
{
    if (i == index)
    {
        var tmp = curr.Value;
        prvs.Next = curr.Next;
        ItemRemoved?.Invoke(this, tmp);
        return;
    }
    prvs = curr;
    curr = curr.Next;
}
if (index == _count)
{
    var tmp = _end.Value;

    _end = prvs;

```

```

        _end.Next = null;
        ItemRemoved?.Invoke(this, tmp);

    }
}

public T this[int index]
{
    get
    {
        if (Count > 0 && index >= 0 && Count > index)
        {
            var tmp = _start;
            for (int i = 0; i < index; i++)
            {
                tmp = tmp.Next;
            }
            return tmp.Value;
        }
        else { throw new IndexOutOfRangeException("Index was out of
range"); }
    }
    set
    {
        if (Count > 0 && index >= 0 && Count > index)
        {
            var tmp = _start;
            for (int i = 0; i < index; i++)

```

```

        {
            tmp = tmp.Next;
        }
        tmp.Value = value;
        _version++;
        ItemSet?.Invoke(this, index);
    }
    else { throw new IndexOutOfRangeException("Index was out of
range"); }
    }
}

```

```

private void AppendItem(T item)
{
    if (_end is null)
    {
        _start = new ListNode<T>(item);
        _end = _start;
    }
    else
    {
        _end.Next = new ListNode<T>(item);
        _end = _end.Next;
    }
    _count++;
    _version++;
}

public IEnumerator<T> GetEnumerator()

```

```

    {
        return new MyEnumerator(this);
    }
IEnumerator IEnumerable.GetEnumerator()
{
    return GetEnumerator();
}

internal class MyEnumerator : IEnumerator<T>
{
    private readonly CustomList<T> _myCollection;
    private readonly int _versionSnapshot;

    private T _current { get; set; }
    public T Current => _current;
    object IEnumerator.Current => _current;

    private int _indexer;

    public MyEnumerator(CustomList<T> myCollection)
    {
        _myCollection = myCollection ?? throw new
ArgumentNullException("The collection is null");

        _indexer = 0;
        _versionSnapshot = myCollection._version;
    }

```

```
        _current = _myCollection.Count > 0 ? _myCollection[_indexer] :  
default(T);  
    }
```

```
public void Dispose()  
{  
}
```

```
public bool MoveNext()  
{  
    if (_versionSnapshot != _myCollection._version) throw new  
Exception("The collection has been modified");
```

```
    if (_indexer >= _myCollection.Count)  
    {  
        Reset();  
        return false;  
    }  
    _current = _myCollection[_indexer];  
    _indexer++;  
    return true;  
}
```

```
public void Reset()  
{  
    _indexer = 0;  
    _current = _myCollection?.Count > 0 ? _myCollection[_indexer] :  
default(T);
```

```

    }
}

private class ListNode<T>
{
    public T Value { get; set; }
    public ListNode<T> Next { get; set; } = null;
    public ListNode(T value)
    {
        Value = value;
    }
}
}

```

Program.cs:

```

using System;
using System.Collections;
using System.Collections.Generic;
using MyList;

namespace CollectionTest
{
    class Program
    {
        static void Main(string[] args)
        {
            var coll = new CustomList<int>() { 1, 2, 3 };

```

```
Console.WriteLine("Initial state of list:");
```

```
foreach (var a in coll)
```

```
{
```

```
    Console.WriteLine(a);
```

```
}
```

```
coll.Add(4);
```

```
Console.WriteLine("\nAddeed 4.");
```

```
Console.WriteLine("State of list:");
```

```
foreach (var a in coll)
```

```
{
```

```
    Console.WriteLine(a);
```

```
}
```

```
Console.WriteLine("Value at [1]: {0}",coll[1]);
```

```
Console.WriteLine("Index of 3:{0}",coll.IndexOf(3));
```

```
coll[1] = 7;
```

```
Console.WriteLine("\nChanged [1] element into 7");
```

```
foreach (var a in coll)
```

```
{
```

```
    Console.WriteLine(a);
```

```
}
```

```
Console.WriteLine("\nRemoved element at [2]:");
```

```
coll.RemoveAt(2);
```

```
foreach (var a in coll)
```

```
{  
    Console.WriteLine(a);  
}
```

```
coll.Remove(7);  
Console.WriteLine("\nRemoved 7:");  
foreach (var a in coll)  
{  
    Console.WriteLine(a);  
}
```

```
coll.Insert(2, 12);  
Console.WriteLine("\nInserted 12 at [2]");  
foreach (var a in coll)  
{  
    Console.WriteLine(a);  
}
```

```
Console.WriteLine("\nContains 12? {0}", coll.Contains(12)?  
"Yes":"No");
```

```
int[] numbers = new int[5];  
numbers[0] = 999999;  
coll.CopyTo(numbers, 1);  
Console.WriteLine("Side array after CopyTo (one index offset, first one  
of orig array is 999999):");  
foreach(var a in numbers)  
{
```



```

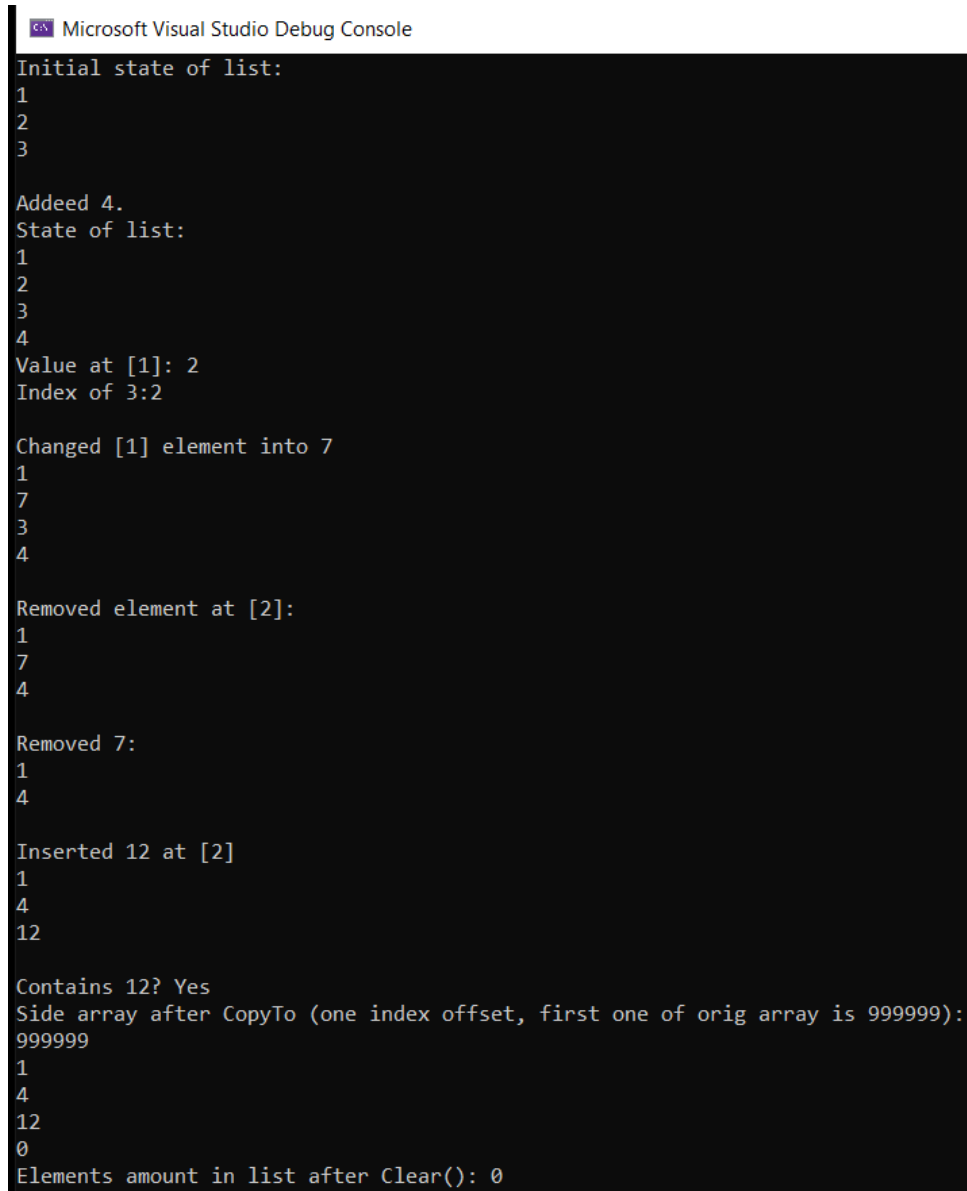
        Console.WriteLine(a);
    }

    coll.Clear();

    Console.WriteLine("Elements amount in list after Clear():
{0}",coll.Count);
}
}
}

```

*Результати тестування:*



```

Microsoft Visual Studio Debug Console
Initial state of list:
1
2
3

Addeed 4.
State of list:
1
2
3
4
Value at [1]: 2
Index of 3:2

Changed [1] element into 7
1
7
3
4

Removed element at [2]:
1
7
4

Removed 7:
1
4

Inserted 12 at [2]
1
4
12

Contains 12? Yes
Side array after CopyTo (one index offset, first one of orig array is 999999):
999999
1
4
12
0
Elements amount in list after Clear(): 0

```

## **Висновок**

Отож, у ході виконання лабораторної роботи було створено власну колекцію узагальненого типу за інтерфейсом `List<T>` за допомогою динамічно зв'язного списку. У рамках лабораторної роботи було реалізовано вищевказаний інтерфейс, власний енумератор, а також клас вузла для зв'язного списку. Урешті-решт, реалізовану колекцію було збережено в бібліотеці та протестовано в консольному додатку. Набуто практичних навичок реалізації колекцій узагальнених типів, зкоерма списків, та енумераторів.