Міністерство освіти і науки України КПІ ім. Ігоря Сікорського

Факультет Інформатики та Обчислювальної Техніки

ЗВІТ

до лабораторної роботи №1 з модуля

«Сучасні технології розробки WEB-застосувань на платформі Microsoft.NET»

Перевірив:
Викладач кафедри ІСТ
ФІОТ
Бардін В.

Виконав:
Поліщук Владислав
гр. ІК-13

2023

Узагальнені типи (Generic) з підтримкою подій. Колекції

Мета лабораторної роботи – навчитися проектувати та реалізовувати узагальнені типи, а також типи з підтримкою подій.

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.

Варіант	Опис узагальненої колекції	Функціонал	Реалізація
4	Дек (черга з двома кінцями)	Див. Queue<T>	Збереження даних за допомогою динамічно зв'язаного списку

Код програми

```
namespace MyDeq
{
    public class MyDeq<T> : IEnumerable<T>, ICollection
    {
        private T[] _array;
        private int _head;
        private int _tail;
        private int _size;

        private const int _defaultCapacity = 10;

        public MyDeq()
        {
            _array = Array.Empty<T>();
        }

        public MyDeq(int capacity)
        {
            if (capacity < 0)
                throw new ArgumentOutOfRangeException(nameof(capacity));
            if (capacity < 10)
            {
                _array = new T[_defaultCapacity];
            }
            _array = new T[capacity];
        }
```

```
public event Action<T> AddEvent;
public event Action<T> RemoveEvent;
public event Action ClearEvent;

protected void OnAddEvent(T e) => AddEvent?.Invoke(e);
protected void OnRemoveEvent(T e) => RemoveEvent?.Invoke(e);
protected void OnClearEvent() => ClearEvent?.Invoke();

public int Count
{
    get { return _size; }
}

bool ICollection.IsSynchronized
{
    get { return false; }
}

object ICollection.SyncRoot => this;

public void EnqueueItemAtStart(T item)
{
    if (_size == _array.Length)
    {
        ResizeQueue();
    }

    if(_size != 0)
        MoveNextLeft(ref _head);

    _array[_head] = item;

    OnAddEvent(item);

    _size++;
}

public T DequeueItemFromStart()
{
    int head = _head;
    T[] array = _array;

    if (_size == 0)
    {
        throw new InvalidOperationException("Deq is empty");
    }

    T removed = array[head];

    array[head] = default!;

    MoveNextRight(ref _head);

    _size--;

    OnRemoveEvent(removed);

    return removed;
}
public void EnqueueItemAtEnd(T item)
{
    if (_size == _array.Length)
    {
        ResizeQueue();
    }
```

```
    if (_size != 0)
        MoveNextRight(ref _tail);

    _array[_tail] = item;

    OnAddEvent(item);

    _size++;
}
public T DequeueItemFromEnd()
{
    int tail = _tail;
    T[] array = _array;

    if (_size == 0)
    {
        throw new InvalidOperationException("Deq is empty");
    }

    T removed = array[tail];

    array[tail] = default!;

    MoveNextLeft(ref _tail);

    _size--;

    OnRemoveEvent(removed);

    return removed;
}
public T PeekItemFromStart()
{
    if (_size == 0)
    {
        throw new InvalidOperationException("Deq is empty");
    }

    return _array[_head];
}
public T PeekItemFromEnd()
{
    if (_size == 0)
    {
        throw new InvalidOperationException("Deq is empty");
    }

    return _array[_tail];
}

private void MoveNextRight(ref int index)
{
    int tmp = index + 1;
    if (tmp == _array.Length)
    {
        tmp = 0;
    }
    index = tmp;
}

private void MoveNextLeft(ref int index)
{
    int tmp = index - 1;
    if (tmp == -1)
    {
        tmp = _array.Length - 1;
    }
```

```
    index = tmp;
}

private void ResizeQueue()
{
    int newcapacity = 2 * _array.Length;

    if(Array.MaxLength == _array.Length)
        throw new OverflowException("Deq is overflow");

    if((uint)newcapacity > Array.MaxLength)
        newcapacity = Array.MaxLength;

    T[] newarray = new T[newcapacity];

    CopyTo(newarray, 0);

    _array = newarray;
    _head = 0;

    _tail = _size - 1;
}

//public IEnumerator<T> GetEnumerator()
//{
//    return new Iterator(this);
//}

public IEnumerator<T> GetEnumerator()
{
    if (_head < _tail)
    {
        for (int i = 0; i < _size; i++)
        {
            yield return _array[i];
        }
    }
    else
    {
        for (int i = _head; i < _array.Length; i++)
        {
            yield return _array[i];
        }

        for (int i = 0; i < _tail + 1; i++)
        {
            yield return _array[i];
        }
    }
}

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

public void Clear()
{
    if (_size != 0)
    {
        int numToClear = _size;

        if (numToClear == 0)
            return;

        int firstPart = Math.Min(_array.Length - _head, numToClear);
```

```
                Array.Clear(_array, _head,  firstPart);

                numToClear -= firstPart;

                if (numToClear > 0)
                {
                    Array.Clear(_array, 0, numToClear);
                }

                _size = 0;
            }

            _head = 0;
            _tail = 0;

            OnClearEvent();
        }

        public void CopyTo(T[]? array, int arrayIndex)
        {
            if (array == null)
            {
                throw new ArgumentNullException(nameof(array));
            }

            if (arrayIndex < 0 || arrayIndex > array.Length)
            {
                throw new ArgumentOutOfRangeException(nameof(arrayIndex));
            }

            if (array.Length - arrayIndex < _size)
            {
                throw new ArgumentException("Invalid length of the array");
            }

            int numToCopy = _size;

            if (numToCopy == 0)
                return;

            int firstPart = Math.Min(_array.Length - _head, numToCopy);

            Array.Copy(_array, _head, array, arrayIndex, firstPart);

            numToCopy -= firstPart;

            if (numToCopy > 0)
            {
                Array.Copy(_array, 0, array, arrayIndex + _array.Length - _head,
numToCopy);
            }
        }

        public bool Contains(T item)
        {
            if (_size == 0)
            {
                return false;
            }

            if (_head < _tail)
            {
                return Array.IndexOf(_array, item, _head, _size) >= 0;
            }

            return
                Array.IndexOf(_array, item, _head, _array.Length - _head) >= 0 ||
```

```
                Array.IndexOf(_array, item, 0, _tail + 1) >= 0;
        }

        void ICollection.CopyTo(Array array, int index)
        {
            CopyTo(array as T[], index);
        }
        private class Iterator : IEnumerator
        {
            private readonly MyDeq<T> _myDeq;
            private T? _current;
            private int _index;

            public Iterator(MyDeq<T> myDeq)
            {
                _myDeq = myDeq;
                _index = -1;
                _current = default;
            }

            public bool MoveNext()
            {
                if (_index == -2)
                    return false;

                _index++;

                if (_index == _myDeq._size)
                {
                    _index = -2;
                    _current = default;
                    return false;
                }

                int capacity = _myDeq._array.Length;

                int arrayIndex = _myDeq._head + _index;

                if (arrayIndex >= capacity)
                {
                    arrayIndex -= capacity;
                }

                _current = _myDeq._array[arrayIndex];
                return true;
            }
            public T Current
            {
                get
                {
                    if (_current == null)
                        throw new InvalidOperationException("Enumeration has not been
started ot it is already finished");
                    return _current;
                }
            }

            object? IEnumerator.Current
            {
                get { return Current; }
            }

            void IEnumerator.Reset()
            {

                _index = -1;
                _current = default;
```

```
            }
        }
    }
}
```

Результати роботи програми

Вставка та видалення елементів з деку

```
a.EnqueueItemAtStart(20);
a.EnqueueItemAtStart(100);

a.EnqueueItemAtEnd(1);
a.EnqueueItemAtEnd(2);
a.EnqueueItemAtEnd(3);
a.EnqueueItemAtEnd(4);
a.EnqueueItemAtEnd(5);
a.EnqueueItemAtStart(6);
a.EnqueueItemAtEnd(7);
a.EnqueueItemAtEnd(8);
a.EnqueueItemAtEnd(9);
a.EnqueueItemAtStart(10);
a.EnqueueItemAtStart(11);
a.EnqueueItemAtStart(12);
a.EnqueueItemAtStart(13);
```

```
a.DequeueItemFromEnd();
a.DequeueItemFromStart();
a.DequeueItemFromEnd();
a.DequeueItemFromStart();
```

Робота подій

```
D:\КПІ\3 курс\.Net 4 variant\Labs1-2\Labs\bi
Element was added to deq
Element was added to deq
Element was added to deq
Element was added to deq
Element was added to deq
Element was added to deq
Element was added to deq
Element was added to deq
Element was added to deq
Element was added to deq
Element was added to deq
Element was added to deq
Element was added to deq
Element was added to deq
Element was added to deq
Element was removed from deq
Element was removed from deq
Element was removed from deq
Element was removed from deq
11
```

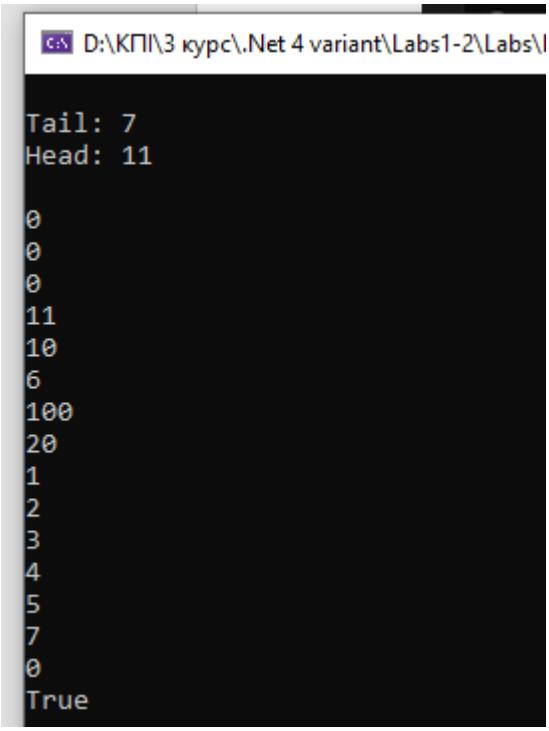
Робота циклу foreach

```
11
10
6
100
20
1
2
3
4
5
7
```

Робота методів Peek… та CopyTo

```
45
46      var tail = a.PeekItemFromEnd();
47      Console.WriteLine("Tail: " + tail);
48
49      var head = a.PeekItemFromStart();
50      Console.WriteLine("Head: " + head);
51
```

```
54      var arr = new int[15];
55
56      a.CopyTo(arr, 3);
57
58      foreach (var item in arr)
59      {
60          Console.WriteLine(item);
61      }
```



LINQ працює

```
var h = a.Any();
Console.WriteLine(h);
```