HomeWork 2.1

```
//HomeWork 2.1
public SingleLinkedList()
{
    head = new SingleLinkedNode<T>();
}

public SingleLinkedList(SingleLinkedList<T> a) : this()
{
    SingleLinkedNode<T> p = head;
    foreach(T item in a)//为了能使用foreach函数，要同时实现GetEnumerator()函数
    {
        SingleLinkedNode<T> new_item = new SingleLinkedNode<T>(item);
        p.Next = new_item;
        p = p.Next;
    }
}

public IEnumerator<T> GetEnumerator()
{
    SingleLinkedNode<T> p = head;
    while (p.Next != null)
    {
        yield return p.Next.Item;
        p = p.Next;
    }
}

public T GetNodeValue(int i)
{
     if ((i < 0) || (i >= Count))
    {
        throw new IndexOutOfRangeException("Index is out of range " + this.GetType());
    }
    else
    {
        SingleLinkedNode<T> p = head.Next;
        while (i > 0)
        {
            p = p.Next;
            i--;
        }
        return p.Item;
    }
}

public static int Sum()
{
    int sum = 0;
    SingleLinkedNode<int> p = head;
    while(p.Next != null)
    {
        sum += p.Next.Item;
        p = p.Next;
    }
    return sum;
```

```
}

public bool Contain(T item)
{
	bool isContain = false;
	SingleLinkedNode<T> p = head;
	while ((p.Next != null) && (!isContain))//一旦找到就可以提前退出循环
	{
		isContain = item.Equals(p.Next.Item);
		p = p.Next;
	}
	return isContain;
}

public void Remove(T item)
{
	SingleLinkedNode<T> p = head;
	SingleLinkedNode<T> q = head.Next;
	while (q.Next != null)
	{
		if (item.Equals(q.Item))
		{
			p.Next = q.Next;
			break;
		}
		p = p.Next;
		q = q.Next;
	}
}

public void AddRange(SingleLinkedList<T> list)
{
	SingleLinkedNode<T> p = head;
	while (p.Next != null)
	{
		p = p.Next;
	}
	p.Next = list.Head.Next;
}
```

HomeWork 2.2

```
//HomeWork 2.2
//SequencedList类
public override string ToString()
{
	 StringBuilder s = new StringBuilder();
	for (int i = 0; i < items.Length; i++)
	{
		s.Append(items[i]);
	}
	return s.ToString();
```

```
}
//SingleLinkedList类
public override string ToString()
{
	StringBuilder s = new StringBuilder();
	SingleLinkedNode<T> p = head;
	while (p.Next != null)
	{
		s.Append(p.Next);
		p = p.Next;
	}
	return s.ToString();
}
```

HomeWork 2.3

```
//HomeWork 2.3
public DoubleLinkedList(DoubleLinkedList<T> a) : this()
{
	DoubleLinkedNode<T> p = head;
	foreach (T item in a)
	{
		DoubleLinkedNode<T> new_item = new DoubleLinkedNode<T>(item);
		p.Next = new_item;
		new_item.Front = p;
		p = p.Next;
	}
}

public IEnumerator<T> GetEnumerator()
{
	DoubleLinkedNode<T> p = head;
	while (p.Next != null)
	{
		yield return p.Next.Item;
		p = p.Next;
	}
}
public void Remove(T item)
{
	DoubleLinkedNode<T> p = head;
	while(p.Next != null)
	{
		if (item.Equals(p.Item))
		{
			p.Front.Next = p.Next;
			p.Next.Front = p.Front;
			break;
		}
		p = p.Next;
	}
}

public int IndexOf(T item)
{
	int index = 0;
	DoubleLinkedNode<T> p = head;
```

```
        while (p.Next != null)
        {
            if (item.Equals(p.Next.Item))//一旦找到匹配的就立刻退出函数
            {
                return index;
            }
            else
            {
                index++;
                p = p.Next;
            }
        }
        return -1;
}

public override string ToString()
{
        StringBuilder s = new StringBuilder();
        DoubleLinkedNode<T> p = head;
        while (p.Next != null)
        {
            s.Append(p.Next);
            p = p.Next;
        }
        return s.ToString();
}

//由于链表本身是泛型的，无法直接进行排序，故此处将函数写成类的静态函数，并且假设链表存储的数据均为int类型
public static void InsertSort(DoubleLinkedList<int> list,int num)
{
        //先对链表进行排序
        Sort(list);
        DoubleLinkedNode<int> p = list.Head.Next;
        DoubleLinkedNode<int> item = new DoubleLinkedNode<int>(num);
        //排序后的链表进行插入操作
        while (p != null)
        {
            if((p.Item < num) && (p.Next.Item > num))
            {
                item.Front = p;
                item.Next = p.Next;
                p.Next.Front = item;
                p.Next = item;
            }
            p = p.Next;
        }
}

public static void Sort(DoubleLinkedList<int> list)
{
        //冒泡排序
        for (int i = 0; i < list.Count; i++)
        {
            for (int j = i; j < list.Count; j++)
            {
                //由于链表实现了索引器，所以可以直接像数组一样访问元素
                if (list[j] < list[i])
```

```
                {
                    //为避免代码过长，此处用一个函数进行链表的交换
                    Exchange(list, j, i);
                }
            }
        }
    }

    public static void Exchange(DoubleLinkedList<int> list, int small, int big)
    {
        int cnt = 0;
        int num_small = list[small], num_big = list[big];
        DoubleLinkedNode<int> p = list.Head;
        while (p.Next != null)
        {
            p = p.Next;
            if (cnt == big)
            {
                p.Item = num_small;
            }
            else if (cnt == small)
            {
                p.Item = num_big;
                break;
            }
            cnt++;
        }
    }
```

Homework 2.4

```
//HomeWork 2.4
//不含头结点的单项链表类
public class SingleLinkedList1<T>
{
    SingleLinkedNode<T> first;

    public SingleLinkedNode<T> First
    {
        get
        {
            return first;
        }
    }
    public SingleLinkedList1()
    {
        first = null;
    }

    public SingleLinkedList1(SingleLinkedNode<T> item) : this()//使用this()告诉电脑调用无参数构造函数
    {
        first = item;
    }

    public SingleLinkedList1(T[] itemArray) : this()//使用一个数组创建线性表
    {
```

```
        SingleLinkedNode<T> rear, q;
        first = new SingleLinkedNode<T>(itemArray[0]);
        rear = first;//指向链表尾结点
        for (int i = 1; i < itemArray.Length; i++)
        {
            q = new SingleLinkedNode<T>(itemArray[i]);//新建一个结点
            rear.Next = q;
            rear = q;
        }
    }

    //只读属性，获取链表的长度
    public virtual int Count
    {
        get
        {
            int n = 0;
            SingleLinkedNode<T> p = first;
            while (p != null)
            {
                n++;
                p = p.Next;
            }
            return n;
        }
    }

    public virtual bool Empty
    {
        get
        {
            return first == null;
        }
    }

    //索引器
    public virtual T this[int i]
    {
        get
        {
            if ((i < 0) || (i >= Count))
            {
                throw new IndexOutOfRangeException("Index is out of range " + this.GetType());
            }
            else
            {
                SingleLinkedNode<T> p = first;
                while (i > 0)
                {
                    p = p.Next;
                    i--;
                }
                return p.Item;
            }
        }
    }

    public void Add(T item)
```

```
    {
        SingleLinkedNode<T> new_item = new SingleLinkedNode<T>(item);
        SingleLinkedNode<T> p = first;
        if (first == null)
        {
            first = new_item;
        }
        else
        {
            while (p.Next != null)
            {
                p = p.Next;
            }
            p.Next = new_item;
        }
    }

    public void AddRange(T[] itemArray)
    {
        SingleLinkedNode<T> p = first;
        if (first == null)
        {
            first = new SingleLinkedNode<T>(itemArray[0]);
            for (int i = 1; i < itemArray.Length; i++)
            {
                SingleLinkedNode<T> new_item = new SingleLinkedNode<T>(itemArray[i]);
                p.Next = new_item;
                p = p.Next;
            }
        }
        else
        {
            while (p.Next != null)
            {
                p = p.Next;
            }
            for (int i = 0; i < itemArray.Length; i++)
            {
                SingleLinkedNode<T> new_item = new SingleLinkedNode<T>(itemArray[i]);
                p.Next = new_item;
                p = p.Next;
            }
        }
    }

    public void Insert(int index, T item)
    {
        if ((index < 0) || (index >= Count))
        {
            throw new IndexOutOfRangeException("Index out of range!!!");
        }
        else
        {
            if (index == 0)
            {
                SingleLinkedNode<T> new_item = new SingleLinkedNode<T>(item);
                new_item.Next = first;
                first = new_item;
            }
```

```
            else
            {
                SingleLinkedNode<T> new_item = new SingleLinkedNode<T>(item);
                SingleLinkedNode<T> p = first;
                SingleLinkedNode<T> q = first.Next;
                for (int i = 1; i < index; i++)
                {
                    p = p.Next;
                    q = q.Next;
                }
                p.Next = new_item;
                new_item.Next = q;
            }
        }
    }

    public void InsertRange(int index, T[] itemArray)
    {
        if ((index < 0) || (index >= Count))
        {
            throw new IndexOutOfRangeException("Index out of range!!!");
        }
        else
        {
            if(index == 0)
            {
                //保存原来的第一个元素
                SingleLinkedNode<T> p = first;
                first = new SingleLinkedNode<T>(itemArray[0]);
                //新链表的第一个元素
                SingleLinkedNode<T> q = first;
                for (int i = 1; i < itemArray.Length; i++)
                {
                    //新的数组有多少个元素就必须新建多少个对象
                    SingleLinkedNode<T> new_item = new SingleLinkedNode<T>(itemArray[i]);
                    q.Next = new_item;
                    q = q.Next;
                }
                q.Next = p;
            }
            else
            {
                SingleLinkedNode<T> p = first;
                SingleLinkedNode<T> q = first.Next;
                for (int i = 1; i < index; i++)
                {
                    p = p.Next;
                    q = q.Next;
                }
                for (int i = 0; i < itemArray.Length; i++)
                {
                    //新的数组有多少个元素就必须新建多少个对象
                    SingleLinkedNode<T> new_item = new SingleLinkedNode<T>(itemArray[i]);
                    p.Next = new_item;
                    p = p.Next;
                }
                p.Next = q;
            }
```

```
        }
    }

    public void Remove(T item)
    {
        SingleLinkedNode<T> p = first;
        SingleLinkedNode<T> q = first.Next;
        if (item.Equals(first.Item))
        {
            first = first.Next;
        }
        else
        {
            while (q != null)
            {
                if (item.Equals(q.Item))
                {
                    p.Next = q.Next;
                    break;
                }
                p = p.Next;
                q = q.Next;
            }
        }
    }

    public void RemoveAt(int index)
    {
        if((index < 0) || (index >= Count))
        {
            throw new IndexOutOfRangeException("Index out of range!!!");
        }
        else
        {
            SingleLinkedNode<T> p = first;
            SingleLinkedNode<T> q = first.Next;
            if (index == 0)
            {
                first = first.Next;
            }
            else
            {
                for (int i = 0; i < (index - 1); i++)
                {
                    p = p.Next;
                    q = q.Next;
                }
                p.Next = q.Next;
            }
        }
    }

    public void RemoveRange(int index, int cnt)
    {
        if ((index < 0) || (index >= Count))
        {
            throw new IndexOutOfRangeException("Index out of range!!!");
        }
        else
```

```
        {
            SingleLinkedNode<T> p = first;
            SingleLinkedNode<T> q;
            //定位到待删除范围最前面的元素的前一个元素
            for (int i = 0; i < (index - 1); i++)
            {
                p = p.Next;
            }
            q = p;
            //定位到待删除范围最后面的元素的后一个元素
            for (int i = 0; i < cnt; i++)
            {
                q = q.Next;
            }
            p.Next = q.Next;
        }
    }

    public void Clear()
    {
        first = null;
    }

    public bool Contain(T item)
    {
        bool isContain = false;
        SingleLinkedNode<T> p = first;
        while ((p != null) && (!isContain))//一旦找到就可以提前退出循环
        {
            isContain = item.Equals(p.Item);
            p = p.Next;
        }
        return isContain;
    }

    public int IndexOf(T item)
    {
        int index = 0;
        SingleLinkedNode<T> p = first;
        while (p != null)
        {
            if (item.Equals(p.Item))//一旦找到匹配的就立刻退出函数
            {
                return index;
            }
            else
            {
                index++;
                p = p.Next;
            }
        }
        return -1;
    }

    public T[] ToArray()
    {
        T[] array = new T[Count];
        SingleLinkedNode<T> p = first;
```

```
            int i = 0;
            while (p != null)
            {
                array[i] = p.Item;
                p = p.Next;
                i++;
            }
            return array;
        }

        public void Show()
        {
            SingleLinkedNode<T> p = first;
            while (p != null)
            {
                p.Show();
            }
        }

        public override string ToString()
        {
            StringBuilder s = new StringBuilder();
            SingleLinkedNode<T> p = first;
            while (p != null)
            {
                s.Append(p);
            }
            return s.ToString();
        }

        public IEnumerator<T> GetEnumerator()
        {
            SingleLinkedNode<T> p = first;
            while (p != null)
            {
                yield return p.Item;
                p = p.Next;
            }
        }
}
//该类对应的测试代码
SingleLinkedList1<int> list_int = new SingleLinkedList1<int>();

list_int.Add(100);
list_int.Add(150);
list_int.Add(200);
int[] test_array1 = { 1, 2, 3 };
list_int.AddRange(test_array1);

list_int.Insert(1, 120);
int[] test_array2 = { 4, 5, 6 };
list_int.InsertRange(2, test_array2);

list_int.Remove(150);
list_int.RemoveAt(3);
list_int.RemoveRange(2, 2);

int[] array_int = list_int.ToArray();
```

```
Console.WriteLine("This is an my list({0} items):",list_int.Count);
foreach(var item in list_int)
{
	Console.Write(item + "    ");
}
Console.WriteLine();

Console.WriteLine("This is an my list to array:");
foreach(var item in array_int)
{
	Console.Write(item + "    ");
}
Console.WriteLine();

if (list_int.Contain(110))
{
	Console.WriteLine("The list contains 110!!!");
}
else
{
	Console.WriteLine("The list doesn't contain 110!!!");
}
Console.WriteLine("Clear the whole list.....");
list_int.Clear();
if (list_int.Empty)
{
	Console.WriteLine("The list is empty!!!");
}
else
{
	Console.WriteLine("The list is not empty!!!");
}
Console.WriteLine();
Console.WriteLine();
```