|  |  |
| --- | --- |
|  | <summary> |
|  | C# indexer with out-of-bounds index |
|  | </summary> |
|  | <param name="index"></param> |
|  | <returns></returns> |
|  | public T this[int index] |
|  | { |
|  | get |
|  | { |
|  | if (index < count && index >= 0) |
|  | { |
|  | return items[index]; |
|  | } |
|  | else |
|  | { |
|  | throw new ArgumentOutOfRangeException(); |
|  | } |
|  | } |
|  | set |
|  | { |
|  | items[index] = value; |
|  | } |
|  | } |
|  |  |
|  | <summary> |
|  | A read-only Count property returning the count of the number of elements |
|  | </summary> |
|  | public int Count |
|  | { |
|  | get |
|  | { |
|  | return count; |
|  | } |
|  | } |
|  |  |
|  | <summary> |
|  | Capacity property returning the size of a private array. |
|  | </summary> |
|  | public int Capacity |
|  | { |
|  | get |
|  | { |
|  | return capacity; |
|  | } |
|  | } |
|  |  |
|  | <summary> |
|  | Add Method |
|  | </summary> |
|  | <param name="item"></param> |
|  | public void Add(T item) |
|  | { |
|  | items[count] = item; |
|  | count++; |
|  | if (count == capacity) |
|  | { |
|  | capacity = capacity \* 2; |
|  | T[] tempArray = new T[capacity]; |
|  | for (int i = 0; i < count; i++) |
|  | { |
|  | tempArray[i] = items[i]; |
|  | } |
|  | items = tempArray; |
|  | } |
|  | } |
|  |  |
|  | <summary> |
|  | Remove Method |
|  | </summary> |
|  | <param name="item"></param> |
|  | public void Remove(T item) |
|  | { |
|  | T[] tempArray = new T[capacity]; |
|  | bool hasFound = false; |
|  | for (int i = 0; i < count; i++) |
|  | { |
|  | if (items[i].Equals(item) && hasFound == false) |
|  | { |
|  | hasFound = true; |
|  | count--; |
|  | } |
|  |  |
|  | if (hasFound == true) |
|  | { |
|  |  |
|  | tempArray[i] = items[i + 1]; |
|  | } |
|  | else |
|  | { |
|  | tempArray[i] = items[i]; |
|  | } |
|  | } |
|  | items = tempArray; |
|  | } |
|  |  |
|  | <summary> |
|  | ToString() method converting the contents of the custom list to a string. |
|  | </summary> |
|  | <returns></returns> |
|  | public override string ToString() |
|  | { |
|  |  |
|  | string value = ""; |
|  |  |
|  | for (int i = 0; i < count; i++) |
|  | { |
|  | value += items[i].ToString(); |
|  | } |
|  | return value; |
|  | } |
|  | <summary> |
|  | Overloading the + operator adding two instances together. |
|  | </summary> |
|  | <param name="list1"></param> |
|  | <param name="list2"></param> |
|  | <returns></returns> |
|  | public static CustomList<T> operator +(CustomList<T> list1, CustomList<T> list2) |
|  | { |
|  | CustomList<T> list = new CustomList<T>(); |
|  |  |
|  | for (int i = 0; i < list1.count; i++) |
|  | { |
|  | list.Add(list1[i]); |
|  | } |
|  | for (int i = 0; i < list2.count; i++) |
|  | { |
|  | list.Add(list2[i]); |
|  | } |
|  | return list; |
|  | } |
|  |  |
|  | <summary> |
|  | Overloading the – operator subtracting one instance from another instance. |
|  | </summary> |
|  | <param name="list1"></param> |
|  | <param name="list2"></param> |
|  | <returns></returns> |
|  | public static CustomList<T> operator -(CustomList<T> list1, CustomList<T> list2) |
|  | { |
|  | CustomList<T> list = new CustomList<T>(); |
|  | list = list1 + list2; |
|  |  |
|  | for (int i = 0; i < list1.count; i++) |
|  | { |
|  | list.Remove(list1[i]); |
|  | } |
|  | for (int i = 0; i < list2.count; i++) |
|  | { |
|  | list.Remove(list2[i]); |
|  | } |
|  | return list; |
|  | } |
|  | <summary> |
|  | Zipping two list class instances together in a zipper |
|  | </summary> |
|  | <param name="odd"></param> |
|  | <param name="even"></param> |
|  | <returns></returns> |
|  | public static CustomList<T> Zip(CustomList<T>odd, CustomList<T>even) |
|  | { |
|  | int i = 0; |
|  | CustomList<T> zipList = new CustomList<T>(); |
|  |  |
|  | do |
|  | { |
|  | if (i + 1 <= odd.count) |
|  | { |
|  | zipList.Add(odd[i]); |
|  | } |
|  | if (i + 1 <= even.count) |
|  | { |
|  | zipList.Add(even[i]); |
|  | } |
|  | i++; |
|  | } |
|  | while ((i + 1 <= odd.count) || (i + 1 <= even.count)); |
|  |  |
|  | return zipList; |
|  | } |
|  |  |
|  | <summary> |
|  | Sorting Algorithm: Bubble Sort |
|  | </summary> |
|  | <returns></returns> |
|  | public CustomList<T> Sort() |
|  | { |
|  | int i; |
|  | int j; |
|  |  |
|  | for (j = count-1; j > 0; j--) |
|  | { |
|  | for (i = 0; i < j; i++) |
|  | { |
|  | if (items[i].CompareTo(items[i+1]) > 0) |
|  | { |
|  | exchange(this, i, i + 1); |
|  | } |
|  | } |
|  | } |
|  | return this; |
|  | } |
|  | public void exchange(CustomList<T> l1, int m, int n) |
|  | { |
|  | T temporary; |
|  | temporary = l1[m]; |
|  | l1[m] = l1[n]; |
|  | l1[n] = temporary; |
|  | } |