2024-11-26

# Custom Implementation of Lists

A “custom” implementation of list can be found [in this project](https:/princomp.github.io/code/projects/CList.zip).

﻿using System; // This is required for the exception.  
  
public class CList<T>  
{  
 // A CList is … a Cell.  
 private Cell first;  
  
 // By default, a CList contains only an empty cell.  
 public CList()  
 {  
 first = null;  
 }  
  
 // A Cell is itself two things:  
 // - An element of data (of type T),  
 // - Another cell, containing the next element of data.  
 // We implement this using automatic properties:  
 private class Cell  
 {  
 public T Data { get; set; }  
 public Cell Next { get; set; }  
 public Cell(T dataP, Cell nextP)  
 {  
 Data = dataP;  
 Next = nextP;  
 }  
 }  
  
 // A method to add a cell at the beginning   
 // of the CList (to the left).  
 // We call it AddF for "Add First".  
  
 public void AddF(T dataP)  
 {  
 first = new Cell(dataP, first);  
 }  
  
 // A method to add a cell at the end   
 // of the CList (to the right).  
 // We call it AddL for "Add Last".  
  
 public void AddL(T dataP)  
 {  
 if (first == null)  
 AddF(dataP);  
 else  
 {  
 Cell current = first;  
 while (current.Next != null)  
 // As long as the current Cell has a neighbour…   
 {  
 current = current.Next;  
 // We move the current cell to this neighbour.  
 }  
 // When we are done, we can insert the cell.  
 current.Next = new Cell(dataP, null);  
 }  
 }  
  
  
 // We will actually frequently test if  
 // a CList is empty, so we might  
 // as well introduce a method for that:  
  
 public bool IsEmpty()  
 {  
 return (first == null);  
 }  
  
 // Accessor for the size of the CList.  
 public int Size  
 {  
 get  
 {  
 int size;  
 if (IsEmpty())  
 {  
 size = 0;  
 }  
 else  
 {  
 size = 1;  
 Cell current = first;  
 while (current.Next != null)  
 // As long as the current Cell has a neighbour…   
 {  
 current = current.Next;  
 // We move the current cell to this neighbour.  
 size++;  
 }  
 }  
 return size;  
 }  
 }  
  
 // We can implement a ToString method  
 // "the usual way", using a loop   
 // similar to the one in AddL:  
 // (But we make it very fancy, as   
 // if we were drawing an array).  
  
 public override string ToString()  
 {  
 string returned = "";  
 for (int i = 0; i < Size; i++)  
 {  
 returned += "————";  
 }  
 returned += "\n| ";  
 Cell current = first;  
 while (current != null)  
 {  
 returned += $"{current.Data} | ";  
 current = current.Next;  
 }  
 returned += "\n";  
 for (int i = 0; i < Size; i++)  
 {  
 returned += "————";  
 }  
 return returned;  
 }  
  
 // Method to obtain the nth element if it exists.  
 public T Access(int index)  
 {  
 if (index >= Size)  
 {  
 throw new IndexOutOfRangeException();  
 }  
 else // Some IDE will flag this "else" as redundant.  
 {  
 int counter = 0;  
 Cell current = first;  
 while (counter < index)  
 {  
 current = current.Next;  
 counter++;  
 }  
 return current.Data;  
 }  
 }  
  
 /\*  
 \* We can write four methods to   
 \* remove elements from a CList.  
 \* - One that clears it entirely,  
 \* - One that removes the first cell,  
 \* - One that removes the last cell,  
 \* - One that removes the nth cell, if it exists,   
 \*/  
  
 public void Clear()  
 {  
 first = null;  
 }  
  
 public void RemoveF()  
 {  
 if (!IsEmpty()) first = first.Next;  
 }  
  
 public void RemoveL()  
 {  
 if (!IsEmpty())  
 {  
 if (first.Next == null)  
 {  
 RemoveF();  
 }  
 else  
 {  
 Cell current = first;  
 while (current.Next != null && current.Next.Next != null)  
 {  
 current = current.Next;  
 }  
  
 current.Next = null;  
 }  
 }  
 }  
  
 // Method to remove the nth element if it exists.  
 public void RemoveI(int index)  
 {  
 if (index > Size)  
 {  
 throw new IndexOutOfRangeException();  
 }  
 else // Some IDE will flag this "else" as redundant.  
 {  
 int counter = 0;  
 Cell current = first;  
 while (counter < index - 1)  
 {  
 current = current.Next;  
 counter++;  
 }  
 current.Next = current.Next.Next;  
 }  
 }  
  
 // Method to obtain the largest  
 // number of consecutive values  
 // dataP.  
  
 public int CountSuccessive(T dataP)  
 {  
 int cCount = 0;  
 int mCount = 0;  
 Cell cCell = first;  
 while (cCell != null)  
 {  
 if (cCell.Data.Equals(dataP))  
 {  
 cCount++;  
 }  
 else  
 {  
 if (cCount > mCount) { mCount = cCount; }  
 cCount = 0;  
  
 }  
 cCell = cCell.Next;  
 }  
 if (cCount > mCount) { mCount = cCount; }  
 return mCount;  
 }  
  
 // Method to remove at a particular index  
 public void RemoveAt(int index)  
 {  
 if (index >= 0 && index < Size)  
 {  
 if (index == 0)  
 RemoveF();  
 else if (index == (Size - 1))  
 RemoveL();  
 else  
 {  
 Cell cCell = first;  
 for (int i = 0; i < index - 1; i++)  
 {  
 cCell = cCell.Next;  
 }  
 cCell.Next = cCell.Next.Next;  
 }  
 }  
 else  
 throw new ArgumentOutOfRangeException();  
 }  
  
 // Method to reverse a list  
 public void Reverse()  
 {  
 Cell cCell = first;  
 Cell previous = null;  
 Cell next;  
 while (cCell != null)  
 {  
 next = cCell.Next;  
 cCell.Next = previous;  
 previous = cCell;  
 cCell = next;  
 }  
 first = previous;  
 }  
  
 // Method to look for a specific value (recursively)  
 public bool Find(T dataP)  
 {  
 return Find(first, dataP);  
 }  
  
 private bool Find (Cell cCell, T dataP)  
 {  
 if (cCell == null) return false;  
 else if (cCell.Data.Equals(dataP)) return true;  
 else if (cCell.Next == null) return false;  
 else return Find(cCell.Next, dataP);  
 }  
  
 // Method to obtain the last index  
 // of dataP.  
 public int LastIndexOf(T dataP)  
 {  
 int index = 0, lastIndex = -1;  
 Cell cCell = first;  
 while (cCell != null)  
 {  
 if (cCell.Data.Equals(dataP))  
 {  
 lastIndex = index;  
 }  
 index++;  
 cCell = cCell.Next;  
 }  
 return lastIndex;  
 }  
 // Recursive method to obtain the   
 // frequency of dataP  
 public double Frequency(T dataP)  
 {  
 if (Size == 0) throw new ArgumentNullException("The list is empty.");  
 else return Count(dataP, first) / (double)Size;  
 }  
 private int Count(T dataP, Cell pTmp)  
 {  
 if (pTmp == null)  
 return 0;  
 else if (pTmp.Data.Equals(dataP))  
 return 1 + Count(dataP, pTmp.Next);  
 else  
 return 0 + Count(dataP, pTmp.Next);  
 }  
  
}

[*(Download this code)*](https:/princomp.github.io/code/projects/CList.zip)