|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ | | | | | | | | | | | | |
| Федеральное государственное автономное образовательное учреждение  высшего образования | | | | | | | | | | | | |
| **Дальневосточный федеральный университет** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **Департамент программной инженерии и искусственного интеллекта** | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| **О Т Ч Е Т** | | | | | | | | | | | | |
| по лабораторной работе №1.1  дисциплина «Фундаментальные структуры данных и алгоритмы» | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Студент гр. Б9124-09.03.04прогин | | | |
|  |  |  | | И.Л. Барсуков | |
|  | | | | | | |  |  | (подпись) | |  | |
|  | | | | | | | | | | | | |
|  | | | | | | |  |  | Преподаватель | | |  |
|  | | | | | | |  |  | А. А. Шулятьев | | | |
|  | | | |  |  | |  |  |  |  | И.Л. Барсуков | |
|  | | | |  |  | |  |  | (подпись) |  | (И.О. Фамилия) | |
|  |  |  |  | | |  |  |  |  | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
|  | | | | | | | | | | | | |
| г. Владивосток | | | | | | | | | | | | |
| 2025 | | | | | | | | | | | | |

**1 Неформальная постановка задачи**

Реализовать пакет подпрограмм для работы с динамическим двусвязным кольцевым списком, в котором элементы – целые числа.

Основные операции:

1. Инициализация

2. Освобождение памяти

3. Добавление элемента упорядоченный по убыванию

4. Удаление всех вхождений заданного по значению элемента

5. Удаление перед каждым вхождением заданного

6. Поиск заданного элемента по значению

7. Печать

8. Операция объединение двух списком

**2 Описание типа + спецификация подпрограмм + тесты**

**Классы**

**class Node**

Вспомогательный внутренний класс для представления узлов двухсвязного кольцевого списка.

**Публичные Поля**  
**int Data** — значение узла.

**Node Next** — ссылка на следующий узел списка.

**Node Previous** — ссылка на предыдущий узел списка.

**class CircleList**

Основной класс, реализующий динамический двусвязный кольцевой список целых чисел, упорядоченный по убыванию.

**Свойства**

**Node? Head** – ссылка на первый узел списка

**bool Empty**

**Входные данные:** нет.

**Выходные данные:** булево значение, обозначающее пустой ли список.

Возвращает **true**, если список пустой, иначе **false**.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Пустой список | Пустой список | true |
| Непустой список | Непустой список | false |

**Публичные Конструкторы**

**CircleList()**

Создаёт пустой кольцевой двусвязный упорядоченный по убыванию список.

**Входные данные:** нет

**Выходные данные:** пустой двусвязный кольцевой список

**CircleList(int[] data)**

Создаёт кольцевой двусвязный список с элементами из массива **data**, упорядоченными по убыванию.

**Входные данные:** массив целых чисел.

**Выходные данные:** двусвязный кольцевой упорядоченный по убыванию список

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Инициализация с данными | data = [5, 6] | cписок = [6, 5] |

**Публичные методы**

**void Clear()**

Очищает список, удаляя все элементы.

**Входные данные:** двусвязный кольцевой список.

**Выходные данные:** изменённый двусвязный кольцевой список.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Очистка непустого списка | cписок = [6, 5] | Пустой список |

**void Dispose()**

Вызывает **Clear()** для освобождения ресурсов.

**Входные данные:** двусвязный кольцевой список.

**Выходные данные:** изменённый двусвязный кольцевой список.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Вызов Dispose | cписок = [6, 5] | Пустой список |

**void Add(int data)**

Добавляет элемент в список в отсортированном порядке по убыванию.

**Входные данные:** двусвязный кольцевой список

**Выходные данные:** изменённый двусвязный кольцевой список.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Добавление в пустой список | Пустой список  data = 10 | |  | | --- | |  |  |  | | --- | | [10] | |
| Добавление в список с одним элементом, который меньше добавляемого | [15] data = 10 | |  | | --- | | [15, 10] | |
| Добавление в список с одним элементом, который больше добавляемого | [15] data = 20 | |  | | --- | | [20, 15] | |
| Добавление большего элемента | [30, 20, 10]  data = 40 | |  | | --- | |  |   [40, 30, 20, 10] |
| Добавление меньшего элемента | [30, 20, 10]  data = 5 | [30, 20, 10, 5] |
| Добавление в середину | [30, 20, 10]  data = 25 | [30, 25, 20, 10] |
| Добавление дубликата | [30, 20, 10]  data = 20 | [30, 20, 20, 10] |

**bool Search(int data, out Node? foundNode)**

Ищет элемент со значением **data**. Возвращает **true** и узел первого найденного элемента через переменную **foundNode**; иначе **false** и **foundNode = null**.

**Входные данные:** двусвязный кольцевой список, целочисленный элемент для поиска.

**Выходные данные:** булево значение, найденный узел элемента.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Поиск единственного элемента, который существует в списке | cписок = [50, 49]  data = 50 | true, узел со значением 50 |
| Поиск элемента, который повторяется несколько раз в списке | cписок = [1, 0, 0]  data = 0 | true, узел со значением 0 |
| Поиск элемента в пустом списке | Пустой список  data = 0 | false, node = null |
| Поиск элемента, не существующего в списке | [50]  data = 51 | false, node = null |

**void Print()**

Выводит содержимое списка в консоль. Если список пустой, выводит "List is empty".

**Входные данные:** двусвязный кольцевой список.

**Выходные данные:** текстовые сообщения.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Печать непустого списка | cписок = [2, 1] | 2  1 |
| Печать пустого списка | Пустой список | «List is empty» |

**void RemoveAll(int data)**

Удаляет все элементы со значением data.

**Входные данные:** двусвязный кольцевой список,целочисленный элемент на удаление.

**Выходные данные:** изменённый двусвязный кольцевой список.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Удаление граничных элементов списка | cписок = [5, 5, 3, 1, 1]  data = 1  data = 5 | [3] |
| Удаление единственных элементов в списке | cписок = [4, 4]  data = 4 | Пустой список |
| Удаление элементов, не являющиеся границами списка | cписок = [4, 4, 3, 3, 1, 1]  data = 3 | [4, 4, 1, 1] |
| Удаление в пустом списке | Пустой список  data = 1 | Пустой список |

**void RemoveAllBefore(int data)**

Удаляет все элементы перед каждым элементом со значением **data**.

**Входные данные:** двусвязный кольцевой список,элемент на удаление.

**Выходные данные:** изменённый двусвязный кольцевой список.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Удаление в пустом списке | Пустой список  data = 1 | Пустой список |
| Удаление элемента, который стоит в середине списка | cписок = [3, 2, 1]  data = 1 | [3, 1] |
| Удаление первого элемента | cписок = [3, 2, 1]  data = 2 | [2, 1] |
| Удаление последнего элемента | cписок = [3, 2, 1]  data = 3 | [3, 2] |
| Удаление элементов, повторяющихся по значению | cписок = [3, 2, 2, 1]  data = 2 | [2, 1] |

**static CircleList Union(CircleList first, CircleList second)**

Возвращает новый список, содержащий все элементы из первого и второго списка.

**Входные данные:** первый список, второй список.

**Выходные данные:** список с объединением.

|  |  |  |
| --- | --- | --- |
| **Описание тестовой ситуации** | **Входные данные** | **Выходные данные** |
| Объединение двух списков | first = [15, 10, 5]  second = [20, 15, 5] | [20, 15, 15, 10, 5, 5] |
| Объединение пустых списков | first = []  second = [] | Пустой список |
| Объединение пустого и не пустого списка | first = [3, 2, 1]  second = [] | [3, 2, 1] |
| Объединение списков, где все элементы разные | first = [10, 8, 5]  second = [20, 15, 12] | [20, 15, 12, 10, 8, 5] |
| Объединение списков с дубликатами | first = [10, 8, 8, 5]  second = [8, 8, 5] | [10, 8, 8, 8, 8, 5, 5] |

**Приватные методы**

**void AddNode(int data, Node? previous)**

Внутренний метод. Добавляет новый узел в список.

**Входные данные:** двусвязный кольцевой список, элемент на добавление, узел после которого встанет новый узел.

**Выходные данные:** изменённый двусвязный кольцевой список.

**void RemoveNode(Node node)**

Внутренний метод. Удаляет узел в списке.

**Входные данные:** двусвязный кольцевой список, узел на удаление.

**Выходные данные:** изменённый двусвязный кольцевой список.

**3 Текст программы**

using System.Runtime.CompilerServices;  
  
namespace Lab1;  
  
public class Node  
{  
 public Node(int data, Node next, Node previous)  
 {  
 Data = data;  
 Next = next;  
 Previous = previous;  
 }  
  
 public int Data { get; }   
 public Node Next { get; internal set; }  
 public Node Previous { get; internal set; }  
}  
  
public class CircleList : IDisposable  
{  
 public bool Empty  
 {  
 [MethodImpl(MethodImplOptions.AggressiveInlining)]  
 get => Head is null;  
 }  
   
 public Node? Head { get; private set; }  
  
 public static CircleList operator |(CircleList first, CircleList second)  
 {  
 return Union(first, second);  
 }  
   
 public static CircleList Union(CircleList first, CircleList second)  
 {  
 var result = new CircleList();  
  
 if (first.Empty && second.Empty)  
 return result;  
   
 if (first.Empty)  
 {  
 var node = second.Head!;  
 do  
 {  
 result.Add(node.Data);  
 node = node.Next;  
 } while (node != second.Head);  
  
 return result;  
 }  
   
 if (second.Empty)  
 {  
 var node = first.Head!;  
 do  
 {  
 result.Add(node.Data);  
 node = node.Next;  
 } while (node != first.Head);  
   
 return result;  
 }  
  
 var firstFinished = false;  
 var secondFinished = false;  
 var node1 = first.Head;  
 var node2 = second.Head;  
   
 while (!firstFinished && !secondFinished)  
 {  
 if (node1!.Data > node2!.Data)  
 {  
 result.Add(node1.Data);  
 node1 = node1.Next;  
 if (node1 == first.Head) firstFinished = true;  
 continue;  
 }  
 if (node1.Data < node2.Data)  
 {  
 result.Add(node2.Data);  
 node2 = node2.Next;  
 if (node2 == second.Head) secondFinished = true;  
 continue;  
 }  
   
 result.Add(node1.Data);  
 result.Add(node2.Data);  
 node1 = node1.Next;  
 node2 = node2.Next;  
 if (node1 == first.Head) firstFinished = true;  
 if (node2 == second.Head) secondFinished = true;  
 }  
   
 while (!firstFinished)  
 {  
 result.Add(node1!.Data);  
 node1 = node1.Next;  
 if (node1 == first.Head) firstFinished = true;  
 }  
   
 while (!secondFinished)  
 {  
 result.Add(node2!.Data);  
 node2 = node2.Next;  
 if (node2 == second.Head) secondFinished = true;  
 }  
  
 return result;  
 }  
   
 public CircleList() {}  
   
 public CircleList(int[] data)  
 {  
 foreach (var value in data)  
 Add(value);  
 }  
  
 public void Add(int data)  
 {  
 if (Empty)  
 {  
 AddNode(data, null);  
 return;  
 }  
   
 if (data <= Head!.Previous.Data)  
 {  
 AddNode(data, Head.Previous);  
 return;  
 }  
  
 if (data >= Head!.Data)  
 {  
 AddNode(data, Head.Previous);  
 return;  
 }  
   
 var current = Head!;  
 while (current.Next != Head && current.Next.Data > data)  
 current = current.Next;  
   
 AddNode(data, current);  
 }  
  
 public void RemoveAll(int data)  
 {  
 if (Empty) return;  
  
 var current = Head!;  
 do  
 {  
 var next = current.Next;  
 if (current.Data == data)  
 RemoveNode(current);  
   
 current = next;  
 } while (current != Head);  
   
 if (Head.Data == data)  
 RemoveNode(Head);  
 }  
  
 public void RemoveAllBefore(int data)  
 {  
 if (Empty) return;  
  
 var current = Head!;  
 do  
 {  
 if (current.Data == data)  
 RemoveNode(current.Previous);  
   
 current = current.Next;  
 } while (current != Head);  
 }  
  
 public bool Search(int data, out Node? foundNode)  
 {  
 foundNode = null;  
 if (Empty)  
 return false;  
  
 var current = Head;  
 do  
 {  
 if (current!.Data == data) {  
 foundNode = current;  
 return true;  
 }  
  
 current = current.Next;  
  
 } while (current != Head);  
   
 foundNode = null;  
 return false;  
 }  
  
 public void Print()  
 {  
 if (Empty)  
 {  
 Console.WriteLine("List is empty");  
 return;  
 }  
  
 var current = Head!;  
  
 do  
 {  
 Console.WriteLine(current.Data);  
 current = current.Next;  
 } while (current != Head);  
 }  
   
 public void Clear()  
 {  
 if (Empty)  
 return;  
  
 var current = Head!;  
 do  
 {  
 var next = current.Next;  
 current.Next = null!;  
 current.Previous = null!;  
 current = next;  
 } while (current != Head);  
  
 Head = null;  
 }  
   
 public void Dispose()  
 {  
 Clear();  
 }  
   
 private void AddNode(int data, Node? previous)  
 {  
 Node node;  
   
 if (previous is null)  
 {  
 node = new Node(data, null!, null!);  
  
 node.Next = node;  
 node.Previous = node;  
 Head = node;  
 return;  
 }  
  
 node = new Node(data, previous.Next, previous);  
   
 if (data >= Head!.Data && previous.Next == Head)  
 Head = node;  
  
 previous.Next.Previous = node;  
 previous.Next = node;  
 }  
   
 private void RemoveNode(Node node)  
 {  
 if (node.Next == node)  
 {  
 Head = null;  
 return;  
 }  
  
 if (node == Head)  
 Head = Head.Next;  
  
 node.Previous.Next = node.Next;  
 node.Next.Previous = node.Previous;  
  
 node.Next = null!;  
 node.Previous = null!;  
 }  
}