|  |  |
| --- | --- |
| **Российский университет транспорта (МИИТ) Институт транспортной техники и систем управления Кафедра «Управление и защита информации»** | |
| **Отчет по практическому заданию по теме «Структуры данных» по дисциплине «Системы управления базами данных»** | |
|  | Выполнил:  Студент группы ТКИ-442  Перский М. А. |
|  | Проверил:  Доцент кафедры УиЗИ к.т.н., с.н.с  Васильева М. А. |
| Москва 2024 | |

* **Условие задачи**

Односвязный список

Определить класс «ListNode», включающий следующую информацию:

* Значение элемента списка
* Указатель на следующий элемент списка;

В программе предусмотреть:

* создание списка;
* вывод на экран;
* удаление элемента по ключу или значению;
* вставка элемента по ключу или в конец списка ;
* поиск данных элемента по ключу;
* поиск минимального и максимального элемента.
* проверка наличия элемента в списке
* совмещение двух разных списков в один
* **Текст программы на языке С#**
* **../list/program.cs**

using System;

using System.Collections.Generic;

public class ListNode

{

public int val;

public ListNode next;

public ListNode(int val = 0, ListNode next = null)

{

this.val = val;

this.next = next;

}

~ListNode()

{

}

public ListNode Add(int value)

{

ListNode dummyHead = this;

while (dummyHead.next != null)

{

dummyHead = dummyHead.next;

}

dummyHead.next = new ListNode(value);

return this;

}

public ListNode Delete(int value) // delete all ListNode with val = value

{

ListNode dummyHead = new ListNode(-1);

dummyHead.next = this;

ListNode current = dummyHead;

while (current.next != null && current != null)

{

if (current.next.val == value)

current.next = current.next.next;

else

current = current.next;

}

return dummyHead.next;

}

public ListNode Remove(int pos) // remove ListNode by key, pos считается с 0

{

ListNode dummyHead = new ListNode(-1);

dummyHead.next = this;

ListNode current = dummyHead;

for (int i = 0; i < pos; i++)

{

current = current.next;

}

current.next = current.next.next;

return dummyHead.next;

}

public ListNode Insert(int value, int start\_pos) //start\_pos считается с 0

{

ListNode dummyHead = new ListNode(-1);

dummyHead.next = this;

ListNode current = dummyHead;

for (int i = -1; i < start\_pos; i++)

{

current = current.next;

}

ListNode newNode = new ListNode(value);

newNode.next = current.next;

current.next = newNode;

return dummyHead.next;

}

public int SearchValueByPosition(int pos) //pos считается с 0

{

ListNode dummyHead = new ListNode(-1);

int i = -1;

dummyHead.next = this;

ListNode current = dummyHead;

while (i != pos)

{

i++;

current = current.next;

}

return current.val;

}

public bool IsValueInListNode(int val)

{

ListNode dummyHead = this;

while (dummyHead != null)

{

if (dummyHead.val == val)

return true;

dummyHead = dummyHead.next;

}

return false;

}

public ListNode MergeTwoLists(ListNode list2)

{

ListNode dummyHead = new ListNode(-1);

ListNode list1 = this;

ListNode current = dummyHead;

while (list1 != null && list2 != null)

{

if (list1.val <= list2.val)

{

current.next = list1;

list1 = list1.next;

}

else

{

current.next = list2;

list2 = list2.next;

}

current = current.next;

}

if (list1 != null)

{

current.next = list1;

}

else

{

current.next = list2;

}

return dummyHead.next;

}

public void PrintListNode()

{

ListNode dummyHead = this;

while (dummyHead != null)

{

Console.Write(String.Format("{0} ", dummyHead.val));

dummyHead = dummyHead.next;

}

}

public int GetMax()

{

int max = Int32.MinValue;

ListNode dummyHead = this;

while (dummyHead != null)

{

if (dummyHead.val > max)

max = dummyHead.val;

dummyHead = dummyHead.next;

}

return max;

}

public int GetMin()

{

int min = Int32.MaxValue;

ListNode dummyHead = this;

while (dummyHead != null)

{

if (dummyHead.val < min)

min = dummyHead.val;

dummyHead = dummyHead.next;

}

return min;

}

public ListNode SortList()

{

ListNode dummyHead = new ListNode();

ListNode head = this;

ListNode dummydummyHead = dummyHead;

List<int> list = new List<int>();

while (head != null)

{

list.Add(head.val);

head = head.next;

}

list.Sort();

for (int i = 0; i < list.Count; i++)

{

dummydummyHead.next = new ListNode(list[i]);

dummydummyHead = dummydummyHead.next;

}

return dummyHead.next;

}

public List<int> ReturnListNodeAsList()

{

ListNode dummy = this;

List<int> result\_list = new List<int>();

while (dummy != null)

{

result\_list.Add(dummy.val);

dummy = dummy.next;

}

return result\_list;

}

public ListNode ReturnListAsNode(List<int> input\_list)

{

ListNode list = new ListNode();

ListNode previous\_node = null;

for (int i = input\_list.Count - 1; i>= 0; i--)

{

list = new ListNode(input\_list[i]);

list.next = previous\_node;

previous\_node = list;

}

return list;

}

}

* **../listTests/UnitTest\_list.cs**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using System.Collections.Generic;

namespace listTests

{

[TestClass]

public class UnitTest\_list

{

[TestMethod]

public void ReturnListNodeAsListTest()

{

ListNode test\_listNode = new ListNode().ReturnListAsNode(new List<int>() { 1, 2, 3, 5, 8 });

List<int> expected\_result\_list = new List<int>() { 1, 2, 3, 5, 8 };

CollectionAssert.AreEqual(test\_listNode.ReturnListNodeAsList(), expected\_result\_list);

}

[TestMethod]

public void AddElementTest()

{

ListNode test\_listNode = new ListNode().ReturnListAsNode(new List<int>() { 1, 2, 3, 5, 8 });

int value = 13;

List<int> expected\_result\_list = new List<int>() { 1, 2, 3, 5, 8, 13 };

test\_listNode = test\_listNode.Add(value);

CollectionAssert.AreEqual(test\_listNode.ReturnListNodeAsList(), expected\_result\_list);

}

[TestMethod]

public void RemoveElementsByValueTest()

{

ListNode test\_listNode = new ListNode().ReturnListAsNode(new List<int>() { 1, 5, 2, 3, 5, 8, 5 });

int value = 5;

List<int> expected\_result\_list = new List<int>() { 1, 2, 3, 8 };

test\_listNode = test\_listNode.Delete(value);

CollectionAssert.AreEqual(test\_listNode.ReturnListNodeAsList(), expected\_result\_list);

}

[TestMethod]

public void RemoveElementsByPositionTest()

{

ListNode test\_listNode = new ListNode().ReturnListAsNode(new List<int>() { 1, 2, 3, 5, 8, 5 });

int pos = 5;

List<int> expected\_result\_list = new List<int>() { 1, 2, 3, 5, 8 };

test\_listNode = test\_listNode.Remove(pos);

test\_listNode.PrintListNode();

CollectionAssert.AreEqual(test\_listNode.ReturnListNodeAsList(), expected\_result\_list);

}

[TestMethod]

public void InsertElementTest()

{

ListNode test\_listNode = new ListNode().ReturnListAsNode(new List<int>() { 1, 2, 3, 8 });

int value = 5; int start\_pos = 2;

List<int> expected\_result\_list = new List<int>() { 1, 2, 3, 5, 8 };

test\_listNode = test\_listNode.Insert(value, start\_pos);

CollectionAssert.AreEqual(test\_listNode.ReturnListNodeAsList(), expected\_result\_list);

}

[TestMethod]

public void SearchValbyPosTest()

{

ListNode test\_listNode = new ListNode().ReturnListAsNode(new List<int>() { 1, 2, 3, 5, 8 });

int pos = 2; int expected\_result = 3;

int res = test\_listNode.SearchValueByPosition(pos);

Assert.AreEqual(res, expected\_result);

}

[TestMethod]

public void ValInListNodeTest()

{

ListNode test\_listNode = new ListNode().ReturnListAsNode(new List<int>() { 1, 2, 3, 5, 8 });

int val = 8; bool expected\_result = true;

bool res = test\_listNode.IsValueInListNode(val);

Assert.AreEqual(res, expected\_result);

}

[TestMethod]

public void ValNotInListNodeTest()

{

ListNode test\_listNode = new ListNode().ReturnListAsNode(new List<int>() { 1, 2, 3, 5, 8 });

int val = 6; bool expected\_result = false;

bool res = test\_listNode.IsValueInListNode(val);

Assert.AreEqual(res, expected\_result);

}

[TestMethod]

public void GetMinTest()

{

ListNode test\_listNode = new ListNode().ReturnListAsNode(new List<int>() { 1, 2, 3, 5, 8 });

int expected\_result = 1;

int res = test\_listNode.GetMin();

Assert.AreEqual(res, expected\_result);

}

[TestMethod]

public void GetMaxTest()

{

ListNode test\_listNode = new ListNode().ReturnListAsNode(new List<int>() { 1, 2, 3, 5, 8 });

int expected\_result = 8;

int res = test\_listNode.GetMax();

Assert.AreEqual(res, expected\_result);

}

[TestMethod]

public void MergeListsTest()

{

ListNode test\_listNode = new ListNode().ReturnListAsNode(new List<int>() { 1, 2, 3, 5 });

ListNode merge\_listNode = new ListNode().ReturnListAsNode(new List<int>() { 8, 13, 21, 34 });

List<int> expected\_result\_list = new List<int>() { 1, 2, 3, 5, 8, 13, 21, 34 };

test\_listNode = test\_listNode.MergeTwoLists(merge\_listNode);

CollectionAssert.AreEqual(test\_listNode.ReturnListNodeAsList(), expected\_result\_list);

}

[TestMethod]

public void SortListTest()

{

ListNode test\_listNode = new ListNode().ReturnListAsNode(new List<int>() {5, 1, 3, 2});

List<int> expected\_result\_list = new List<int>() { 1, 2, 3, 5 };

test\_listNode = test\_listNode.SortList();

CollectionAssert.AreEqual(test\_listNode.ReturnListNodeAsList(), expected\_result\_list);

}

}

}

* **Результат работы программы**



Результат запуска тестов в MS Visual Studio 2022

* **UML диаграмма класса ListNode**

