**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной Техники**

**ОТЧЕТ**

**по лабораторной работе №1**

**по дисциплине «Программирование в среде .NET»**

**ТЕМА: РЕАЛИЗАЦИЯ БАЗОВЫХ АЛГОРИТМОВ СРЕДСТВАМИ ЯЗЫКА С#**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 6307 |  | Пякшина О.С. |
| Преподаватель |  | Пешехонов К.А. |

Санкт-Петербург

2020

**Цель работы**

Реализовать несколько базовых алгоритмов в среде .NET с помощью языка C#.

**Задание**

1) Реализовать связный список: создание, удаление, добавление произвольных элементов, реверс списка - без использования стандартных коллекций/LINQ (только IEnumerable);

2) Реализовать бинарное дерево: заполнение, поиск, удаление элемента - без использования стандартных деревьев;

3) Реализовать сортировку вставками - без .OrderBy().

**Код программы**

Проект GenericLibrary

BinaryNode.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace GenericLibrary

{

public enum Side

{

Left,

Right

}

public class BinaryNode<T> where T : IComparable

{

public BinaryNode(T data)

{

Data = data;

}

public T Data { get; set; }

public BinaryNode<T> LeftNode { get; set; }

public BinaryNode<T> RightNode { get; set; }

public BinaryNode<T> ParentNode { get; set; }

public Side? NodeSide =>

ParentNode == null

? (Side?)null

: ParentNode.LeftNode == this

? Side.Left

: Side.Right;

public override string ToString() => Data.ToString();

}

}

BinaryTree.cs

using System;

using System.Collections.Generic;

namespace GenericLibrary

{

public class BinaryTree<T> where T : IComparable

{

public BinaryNode<T> RootNode { get; set; }

public BinaryNode<T> Add(BinaryNode<T> node, BinaryNode<T> currentNode = null)

{

if (RootNode == null)

{

node.ParentNode = null;

return RootNode = node;

}

currentNode = currentNode ?? RootNode;

node.ParentNode = currentNode;

int result;

return (result = node.Data.CompareTo(currentNode.Data)) == 0

? currentNode

: result < 0

? currentNode.LeftNode == null

? (currentNode.LeftNode = node)

: Add(node, currentNode.LeftNode)

: currentNode.RightNode == null

? (currentNode.RightNode = node)

: Add(node, currentNode.RightNode);

}

public BinaryNode<T> Add(T data)

{

return Add(new BinaryNode<T>(data));

}

public BinaryNode<T> FindNode(T data, BinaryNode<T> startWithNode = null)

{

startWithNode = startWithNode ?? RootNode;

int result = data.CompareTo(startWithNode.Data);

return (result == 0)

? startWithNode

: result < 0

? startWithNode.LeftNode == null

? null

: FindNode(data, startWithNode.LeftNode)

: startWithNode.RightNode == null

? null

: FindNode(data, startWithNode.RightNode);

}

public void Remove(BinaryNode<T> node)

{

if (node == null)

{

return;

}

var currentNodeSide = node.NodeSide;

//если у узла нет подузлов -> удаляем

if (node.LeftNode == null && node.RightNode == null)

{

if (currentNodeSide == Side.Left)

{

node.ParentNode.LeftNode = null;

}

else

{

node.ParentNode.RightNode = null;

}

}

//если нет левого, то правый ставим на место удаляемого

else if (node.LeftNode == null)

{

if (currentNodeSide == Side.Left)

{

node.ParentNode.LeftNode = node.RightNode;

}

else

{

node.ParentNode.RightNode = node.RightNode;

}

node.RightNode.ParentNode = node.ParentNode;

}

//если нет правого, то левый ставим на место удаляемого

else if (node.RightNode == null)

{

if (currentNodeSide == Side.Left)

{

node.ParentNode.LeftNode = node.LeftNode;

}

else

{

node.ParentNode.RightNode = node.LeftNode;

}

node.LeftNode.ParentNode = node.ParentNode;

}

//если оба дочерних присутствуют, то правый ставится на место удаляемого,

//а левый на место правого

else

{

switch (currentNodeSide)

{

case Side.Left:

node.ParentNode.LeftNode = node.RightNode;

node.RightNode.ParentNode = node.ParentNode;

Add(node.LeftNode, node.RightNode);

break;

case Side.Right:

node.ParentNode.RightNode = node.RightNode;

node.RightNode.ParentNode = node.ParentNode;

Add(node.LeftNode, node.RightNode);

break;

default:

var bufLeft = node.LeftNode;

var bufRightLeft = node.RightNode.LeftNode;

var bufRightRight = node.RightNode.RightNode;

node.Data = node.RightNode.Data;

node.RightNode = bufRightRight;

node.LeftNode = bufRightLeft;

Add(bufLeft, node);

break;

}

}

}

public void Remove(T data)

{

var foundNode = FindNode(data);

Remove(foundNode);

}

public void PrintTree()

{

PrintTree(RootNode);

}

private void PrintTree(BinaryNode<T> startNode, string indent = "", Side? side = null)

{

if (startNode != null)

{

var nodeSide = side == null ? "+" : side == Side.Left ? "L" : "R";

Console.WriteLine($"{indent} [{nodeSide}]- {startNode.Data}");

indent += new string(' ', 3);

//рекурсивный вызов для левой и правой веток

PrintTree(startNode.LeftNode, indent, Side.Left);

PrintTree(startNode.RightNode, indent, Side.Right);

}

}

}

}

MyList.cs

using System;

using System.Collections.Generic;

using System.Collections;

namespace GenericLibrary

{

public class MyList<T> : IEnumerable<T> // односвязный список

{

Node<T> head;

Node<T> tail;

int count;

public void Add(T data)

{

Node<T> node = new Node<T>(data);

if (head == null)

head = node;

else

tail.Next = node;

tail = node;

count++;

}

public MyList<T> Reverse()

{

Node<T> current = head;

MyList<T> newList = new MyList<T>();

while (current != null)

{

newList.AppendFirst(current.getData());

current = current.Next;

}

return newList;

}

public bool Remove(T data)

{

Node<T> current = head;

Node<T> previous = null;

while (current != null)

{

if (current.Data.Equals(data))

{

if (previous != null)

{

previous.Next = current.Next;

if (current.Next == null)

tail = previous;

}

else

{

head = head.Next;

if (head == null)

tail = null;

}

count--;

return true;

}

previous = current;

current = current.Next;

}

return false;

}

public int Count { get { return count; } }

public bool IsEmpty { get { return count == 0; } }

public void Clear()

{

head = null;

tail = null;

count = 0;

}

public bool Contains(T data)

{

Node<T> current = head;

while (current != null)

{

if (current.Data.Equals(data))

return true;

current = current.Next;

}

return false;

}

public void AppendFirst(T data)

{

Node<T> node = new Node<T>(data);

node.Next = head;

head = node;

if (count == 0)

tail = head;

count++;

}

IEnumerator IEnumerable.GetEnumerator()

{

return ((IEnumerable)this).GetEnumerator();

}

IEnumerator<T> IEnumerable<T>.GetEnumerator()

{

Node<T> current = head;

while (current != null)

{

yield return current.Data;

current = current.Next;

}

}

}

}

Node.cs

namespace GenericLibrary

{

public class Node<T>

{

public Node(T data)

{

Data = data;

}

public T Data { get; set; }

public Node<T> Next { get; set; }

public T getData()

{

return Data;

}

}

}

InsertionSort.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GenericLibrary

{

class InsertionSort

{

public static int[] Sort(int[] array)

{

int[] result = new int[array.Length];

for (int i = 0; i < array.Length; i++)

{

int j = i;

while (j > 0 && result[j - 1] > array[i])

{

result[j] = result[j - 1];

j--;

}

result[j] = array[i];

}

return result;

}

}

}

Generic.Program.cs

using System;

using System.Collections;

using System.Collections.Generic;

using GenericLibrary;

namespace Generic

{

class Program

{

//метод обмена элементов

static void Swap(ref int e1, ref int e2)

{

var temp = e1;

e1 = e2;

e2 = temp;

}

//сортировка вставками

static int[] InsertionSort(int[] array)

{

for (var i = 1; i < array.Length; i++)

{

var key = array[i];

var j = i;

while ((j > 1) && (array[j - 1] > key))

{

Swap(ref array[j - 1], ref array[j]);

j--;

}

array[j] = key;

}

return array;

}

static void Main(string[] args)

{

MyList<string> stringList = new MyList<string>();

MyList<int> intList = new MyList<int>();

int[] sortArr = { 1, 5, 2, 6, 9, 7 };

Console.WriteLine("MY INT LIST:");

intList.Add(1);

intList.Add(2);

intList.Add(3);

Console.WriteLine("Add 1, 2, 3 to intList:");

foreach (var item in intList)

{

Console.Write(item + " ");

}

Console.WriteLine();

intList.Remove(2);

Console.WriteLine("Delete 2 from intList:");

foreach (var item in intList)

{

Console.Write(item + " ");

}

Console.WriteLine();

bool f = intList.Contains(3);

Console.WriteLine(f == true ? "List contain 3" : "List doesn't contain 3");

Console.WriteLine("Add 4 first in intList:");

intList.AppendFirst(4);

foreach (var item in intList)

{

Console.Write(item + " ");

}

Console.WriteLine();

Console.WriteLine("Reverse:");

foreach (var item in intList.Reverse())

{

Console.Write(item + " ");

}

Console.WriteLine();

intList.Clear();

Console.ReadKey();

Console.Clear();

Console.WriteLine("MY STRING LIST:");

stringList.Add("AA");

stringList.Add("BB");

stringList.Add("CC");

stringList.Add("DD");

Console.WriteLine("Add AA, BB, CC, DD to stringList:");

foreach (var item in stringList)

{

Console.Write(item + " ");

}

Console.WriteLine();

Console.WriteLine("Delete AA from stringList:");

stringList.Remove("AA");

foreach (var item in stringList)

{

Console.Write(item + " ");

}

Console.WriteLine();

bool isPresent = stringList.Contains("CC");

Console.WriteLine(f == true ? "List contain CC" : "List doesn't contain CC");

Console.WriteLine("Add WW first in intList:");

stringList.AppendFirst("WW");

foreach (var item in stringList)

{

Console.Write(item + " ");

}

Console.WriteLine();

Console.WriteLine("Reverse:");

foreach (var item in stringList.Reverse())

{

Console.Write(item + " ");

}

Console.WriteLine();

stringList.Clear();

Console.ReadKey();

Console.Clear();

Console.WriteLine("BINARY TREE:");

var binaryTree = new BinaryTree<int>();

binaryTree.Add(5);

binaryTree.Add(6);

binaryTree.Add(8);

binaryTree.Add(11);

binaryTree.Add(2);

binaryTree.Add(6);

binaryTree.Add(8);

binaryTree.Add(4);

binaryTree.Add(12);

binaryTree.PrintTree();

Console.WriteLine(new string('-', 40));

binaryTree.Remove(5);

binaryTree.PrintTree();

Console.WriteLine(new string('-', 40));

binaryTree.Remove(8);

binaryTree.PrintTree();

Console.WriteLine(new string('-', 40));

Console.WriteLine("Try to find node 10");

if (binaryTree.FindNode(11) == null)

{

Console.WriteLine("Node isn't exist");

}

else

{

Console.WriteLine("Node is found");

}

Console.ReadKey();

Console.Clear();

Console.WriteLine("SORT:");

Console.WriteLine("Before: \n1 5 2 6 9 7");

Console.WriteLine("After: ");

foreach (var item in InsertionSort(sortArr))

{

Console.Write(item + " ");

}

Console.ReadKey();

}

}

}

**Выводы**

В процессе выполнения лабораторной работы были реализованы несколько стандартных структур данных средствами языка C# (связный список, бинарное дерево). Были получены навыки работы в среде .NET на языке C#.