Пермский филиал федерального государственного автономного образовательного учреждения высшего образования   
«Национальный исследовательский университет   
«Высшая школа экономики»

*Факультет экономики, менеджмента и бизнес-информатики*

Шешин Николай Андреевич

**Отчет по лабораторной работе №12**

студента образовательной программы «Программная инженерия»

по направлению подготовки *09.03.04 Программная инженерия*

Руководитель

доцент кафедры

информационных

технологий в бизнесе

к.т.н. О.Л. Викентьева

Пермь, 2020 год

**Постановка задачи**

### 2.1. Задание 1.

1. Сформировать однонаправленный список, в информационное поле записать объекты из иерархии классов лабораторной работы №10.
2. Распечатать полученный список.
3. Выполнить обработку списка в соответствии с заданием.
4. Распечатать полученный список.
5. Удалить список из памяти.
6. Сформировать двунаправленный список, в информационное поле записать объекты из иерархии классов лабораторной работы №10.
7. Распечатать полученный список.
8. Выполнить обработку списка в соответствии с заданием.
9. Распечатать полученный список.
10. Удалить список из памяти.
11. Сформировать идеально сбалансированное бинарное дерево, в информационное поле записать объекты из иерархии классов лабораторной работы №10.
12. Распечатать полученное дерево.
13. Выполнить обработку дерева в соответствии с заданием, вывести полученный результат.
14. Преобразовать идеально сбалансированное дерево в дерево поиска.
15. Распечатать полученное дерево.
16. Удалить дерево из памяти.

### 2.2. Задание 2

Реализовать обобщенную коллекцию, указанную в варианте. Для этого:

1. Реализовать конструкторы:

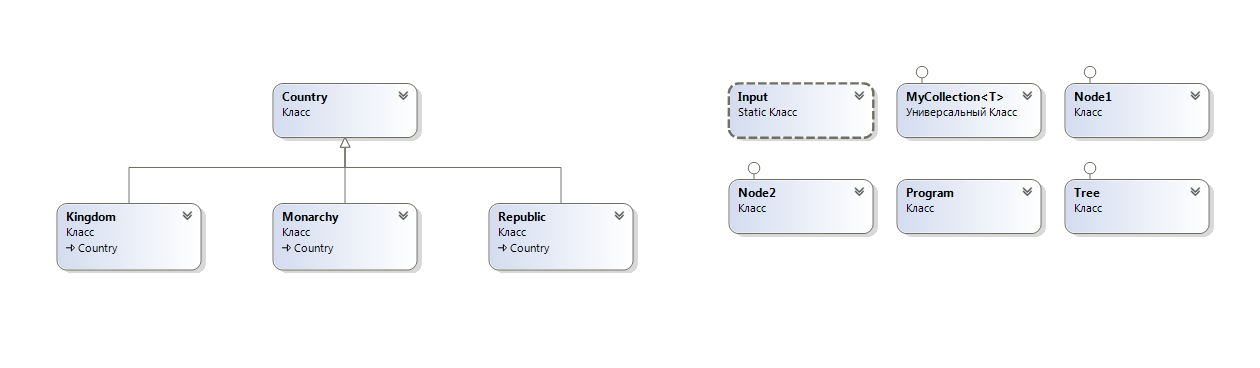
* public MyCollection() - предназначен для создания пустой коллекции.
* public MyCollection (int capacity*)* - создает пустую коллекцию с начальной емкостью, заданной параметром capacity.
* public MyCollection (MyCollection c) - служит для создания коллекции, которая инициализируется элементами и емкостью коллекции, заданной параметром с.

1. Для всех коллекций реализовать:
   * свойство Count, позволяющее получить количество элементов в коллекции;
   * методы для добавления одного или нескольких элементов в коллекцию;
   * методы для удаления одного или нескольких элементов из коллекции (кроме деревьев);[[1]](#footnote-1)
   * метод для поиска элемента по значению;
   * метод для клонирования коллекции;
   * метод для поверхностного копирования;
   * метод для удаления коллекции из памяти.
2. Для всех коллекций реализовать интерфейсы IEnumerable и IEnumerator.
3. Написать демонстрационную программу, в которой создаются коллекции, и демонстрируется работа всех реализованных методов, в том числе, перебор коллекции циклом foreach.

При работе с коллекцией использовать объекты из иерархии классов, разработанной в работе №10.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  варианта | Однонаправленный список | Двунаправленный список | Бинарное дерево | Коллекция |
| 3 | Удалить из списка первый элемент с четным информационным полем. | Добавить в список элемент после элемента с заданным информационным полем. | Найти количество листьев в дереве. | Двунаправленный список |

**Диаграмма классов**



**Методы, реализующие формирование коллекции (конструкторы), выполнение задания.**

Однонаправленный список – формирование и задание по варианту

**public** **Node1**()

{

Data = **default**(Country);

Next = **null**;

}

**public** **Node1**(Country data)

{

Data = data;

Next = **null**;

}

**static** Node1 **MakeNode**(Country data)

{

Node1 node = **new** Node1(data);

**return** node;

}

**public** **static** Node1 **MakeList**(**int** size)

{

Country data = **new** Country();

Node1 first = MakeNode(data);

**for** (**int** i = **1**; i < size; i++)

{

data = **new** Country();

Node1 node = MakeNode(data);

node.Next = first;

first = node;

}

**return** first;

}

**public** Node1 **Delete**(Node1 node)

{

**if** (node.Next == **null**)

**return** node;

Node1 newNode = **new** Node1();

**if** (node.Next.Data.GetPopulation() % **2** == **0**)

{

newNode.Next = Delete(node.Next.Next);

}

**else**

{

newNode.Next = Delete(node.Next);

}

**if** (node.Data.GetPopulation() % **2** == **0**)

newNode = newNode.Next;

**else**

newNode.Data = node.Data;

**return** newNode;

}

Двунаправленный список – формирование и задание по варианту

**public** **Node2**()

{

Data = **default**(Country);

Next = **null**;

Pred = **null**;

}

**public** **Node2**(Country data)

{

Data = data;

Next = **null**;

Pred = **null**;

}

**public** **void** **MakeNode**(**int** size)

{

**if** (size == **0**)

**return**;

Data = **new** Country();

Next = **new** Node2();

Next.Pred = **this**;

Next.MakeNode(size - **1**);

}

**public** **void** **Add**(Country dataToFind, Country afterData)

{

**if** (**this**.Next == **null**)

{

Console.WriteLine("Не найдено!");

**return**;

}

**if** (**this**.Data.GetName() == dataToFind.GetName() && **this**.Data.GetPopulation() == dataToFind.GetPopulation())

{

Node2 node = **new** Node2(afterData);

node.Next = **this**.Next;

node.Pred = **this**;

**this**.Next = node;

**return**;

}

**this**.Next.Add(dataToFind, afterData);

}

Дерево – формирование и задание по варианту

**public** **Tree**(Country data)

{

Data = data;

Left = **null**;

Right = **null**;

}

**public** **Tree**()

{

Data = **default**(Country);

Left = **null**;

Right = **null**;

}

**public** **void** **Add**(Country data)

{

**if** (data.GetPopulation() < Data.GetPopulation())

{

**if** (Left == **null**)

{

Left = **new** Tree(data);

**return**;

}

**else**

{

Left.Add(data);

}

}

**else**

{

**if** (Right == **null**)

{

Right = **new** Tree(data);

**return**;

}

**else**

{

Right.Add(data);

}

}

}

**public** **void** **MakeTree**(**int** size)

{

**if** (**this**.Data == **null**)

{

Data = **new** Country();

size--;

}

**for** (**int** i = **0**; i < size; i++)

{

Add(**new** Country());

}

}

**public** **void** **Print**()

{

Data.Print();

**if** (Left != **null**)

{

Left.Print();

}

**if** (Right != **null**)

{

Right.Print();

}

}

**private** **void** **TreeToMassive**(**ref** List<Country> countries)

{

countries.Add(**this**.Data);

**if** (Left != **null**)

{

Left.TreeToMassive(**ref** countries);

}

**if** (Right != **null**)

{

Right.TreeToMassive(**ref** countries);

}

}

**private** **bool** **AddToLevel**(Country country, **int** level)

{

**if** (level == **1**)

{

**if** (Left == **null**)

{

Left = **new** Tree(country);

**return** **true**;

}

**if** (Right == **null**)

{

Right = **new** Tree(country);

**return** **true**;

}

**return** **false**;

}

**if** (Left.AddToLevel(country, level - **1**))

{

**return** **true**;

}

**else** **if** (Right.AddToLevel(country, level - **1**))

{

**return** **true**;

}

**else**

{

**return** **false**;

}

}

**public** **static** Tree **IdealTreeFromDefault**(Tree tree)

{

List<Country> listedTree = **new** List<Country>();

tree.TreeToMassive(**ref** listedTree);

Tree newTree = **new** Tree();

newTree.Data = listedTree[**0**];

**for** (**int** i = **2**; i <= listedTree.Count; i++)

{

newTree.AddToLevel(listedTree[i - **1**], (**int**)Math.Log2(i));

}

**return** newTree;

}

**public** **static** Tree **DefaultTreeFromIdeal**(Tree tree)

{

List<Country> listedTree = **new** List<Country>();

tree.TreeToMassive(**ref** listedTree);

Tree newTree = **new** Tree();

newTree.Data = listedTree[**0**];

**for** (**int** i = **1**; i < listedTree.Count; i++)

{

newTree.Add(listedTree[i]);

}

**return** newTree;

}

**public** **int** **LeafsCount**()

{

**if** (Left == **null** && Right == **null**)

{

**return** **1**;

}

**if** (Left == **null**)

**return** Right.LeafsCount();

**if** (Right == **null**)

**return** Left.LeafsCount();

**return** Left.LeafsCount() + Right.LeafsCount();

}

Коллекция двунаправленного списка – формирование и доп. методы по заданию

**public** **MyCollection**(T data = **default**(T))

{

Count = **1**;

Data = data;

Next = **null**;

Previous = **null**;

}

**public** **MyCollection**(**int** capacity)

{

Count = capacity;

**if** (capacity == **0**)

{

**return**;

}

Data = **default**(T);

Next = **new** MyCollection<T>(capacity - **1**);

Next.Previous = **this**;

}

**public** **MyCollection**(MyCollection<T> myCollection)

{

Count = myCollection.Count;

**if** (myCollection == **null**)

{

**return**;

}

Data = myCollection.Data;

Next = **new** MyCollection<T>(myCollection.Next);

Next.Previous = **this**;

}

**public** **void** **Add**(T data)

{

**if** (Count == **1**)

{

Next = **new** MyCollection<T>();

Next.Data = data;

Next.Previous = **this**;

Count++;

**return**;

}

Count++;

Next.Add(data);

}

**public** **void** **Add**(MyCollection<T> myCollection)

{

**if** (Count == **1**)

{

Next = myCollection;

Count += myCollection.Count;

**return**;

}

Count += myCollection.Count;

Next.Add(myCollection);

}

**public** **static** **bool** **Remove**(**ref** MyCollection<T> myCollection, T data)

{

**if** (myCollection.Data.Equals(data))

{

**if** (myCollection.Previous == **null**)

{

myCollection = myCollection.Next;

**if** (myCollection != **null**)

myCollection.Previous = **null**;

**return** **true**;

}

**if** (myCollection.Next == **null**)

{

myCollection.Previous.Next = **null**;

**return** **true**;

}

//MyCollection<T> temp = myCollection.Previous;

myCollection.Next.Previous = myCollection.Previous;

myCollection.Previous.Next = myCollection.Next;

**return** **true**;

}

**if** (myCollection.Count == **1**)

{

**return** **false**;

}

**if** (MyCollection<T>.Remove(**ref** myCollection.Next, data))

{

myCollection.Count--;

**return** **true**;

}

**else**

{

**return** **false**;

}

}

**public** **static** MyCollection<T> Remove(MyCollection<T> myCollection, MyCollection<T> toDelete)

{

**while**(toDelete != **null**)

{

MyCollection<T>.Remove(**ref** myCollection, toDelete.Data);

toDelete = toDelete.Next;

}

**return** myCollection;

}

**public** **int** **Contains**(T data, **int** index = **0**)

{

**if** (data.Equals(Data))

{

**return** index;

}

**if** (Count == **1**)

{

**return** -**1**;

}

**return** Next.Contains(data, index + **1**);

}

**Методы реализующие нумератор.**

**public** T Current

{

**get**

{

**return** current.Data;

}

}

**object** IEnumerator.Current => **throw** **new** NotImplementedException();

**public** **void** **Reset**()

{

current = **this**;

position = -**1**;

}

**public** **bool** **MoveNext**()

{

**if** (current.Next == **null**)

{

Reset();

**return** **false**;

}

**else**

{

**if** (position > -**1**)

{

current = current.Next;

}

position++;

**return** **true**;

}

}

**public** IEnumerator<T> GetEnumerator()

{

**return** **this**;

}

IEnumerator IEnumerable.GetEnumerator()

{

**return** **GetEnumerator**();

}

**Тестирование по черному ящику**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Таблица ЧЯ (Ввод страны)** | | | | | | | | | | | | | | | | |
| № | Проверяемая ситуация | Т1 | Т2 | | Т3 | | | Т4 | Т5 | | | Т6 | Т7 | Т8 | Т9 | Т10 |
| 1 | **Название** | | | | | | | | | | | | | | | |
| 1.1 | Слишком большое название | + | |  | |  |  | | |  |  | |  |  |  |  |
| 1.2 | Достаточной длины название |  | | + | |  |  | | |  |  | |  |  |  |  |
| 2 | **Популяция** | | | | | | | | | | | | | | | |
| 2.1 | Неположительное число |  | |  | |  | + | | |  |  | |  |  |  |  |
| 2.2 | Нецелое число |  | |  | |  | + | | |  |  | |  |  |  |  |
| 2.3 | Не число |  | |  | |  |  | | | + |  | |  |  |  |  |
| 2.4 | Положительное число |  | |  | |  |  | | |  | + | | + | + |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Таблица тестов** | | | | |
| **Тест** | **Вход** | **Ожидаемый результат** | **Реальный результат** | **Примечания** |
| **Название** | | | | |
| 1 | Р...Р (n = 1e100) | Сообщение об ошибке | Сообщение об ошибке |  |
| 2 | Россия | Название = Россия | Название = Россия |  |
| **Ввод Популяции** | | | | |
| 4 | -1,37 | Сообщение об ошибке | Сообщение об ошибке |  |
| 5 | а | Сообщение об ошибке | Сообщение об ошибке |  |
| 6 | 3 | Популяция = 3 | Популяция = 3 |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Таблица ЧЯ (однонаправленный список)** | | | | | | | | | | | | | | | | |
| № | Проверяемая ситуация | Т1 | Т2 | | Т3 | | | Т4 | | Т5 | | Т6 | Т7 | Т8 | Т9 | Т10 |
| 1 | **Длина** | | | | | | | | | | | | | | | |
| 1.1 | Неположительное число | + | |  | |  |  | |  | |  | |  |  |  |  |
| 1.2 | Нецелое число | + | |  | |  |  | |  | |  | |  |  |  |  |
| 1.3 | Не число |  | | + | |  |  | |  | |  | |  |  |  |  |
| 1.4 | Положительное целое число |  | |  | | + |  | |  | |  | |  |  |  |  |
| 2 | **Удаление первого четного элемента (популяция)** | | | | | | | | | | | | | | | |
| 2.1 | Четный элемент есть |  | |  | |  | + | |  | |  | |  |  |  |  |
| 2.2 | Четного элемента нет |  | |  | |  |  | | + | |  | |  |  |  |  |
| 3 | **Вывод** | | | | | | | | | | | | | | | |
| 3.1 | Список пуст |  | |  | |  |  | |  | | + | |  |  |  |  |
| 3.2 | Список непуст |  | |  | |  |  | |  | |  | | + |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Таблица тестов** | | | | |
| **Тест** | **Вход** | **Ожидаемый результат** | **Реальный результат** | **Примечания** |
| **Размеры случайного набора** | | | | |
| 1 | -1,37 | Сообщение об ошибке | Сообщение об ошибке |  |
| 2 | а | Сообщение об ошибке | Сообщение об ошибке |  |
| 3 | 3 | Список размера 3 | Список размера 3 |  |
| **Удаление первого четного элемента (популяция, p)** | | | | |
| 4 | p1 = 2, p2 = 3 | p1 = 3 | p1 = 3 |  |
| 5 | p1 = 3 | Не найдено | Не найдено |  |
| **Вывод** | | | | |
| 6 | N = 0 | Список пуст | Список пуст |  |
| 7 | N = 2 | Список размера 2 | Список размера 2 |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Таблица ЧЯ (двунаправленный список)** | | | | | | | | | | | | | | | | |
| № | Проверяемая ситуация | Т1 | Т2 | | Т3 | | | Т4 | | Т5 | | Т6 | Т7 | Т8 | Т9 | Т10 |
| 1 | **Длина** | | | | | | | | | | | | | | | |
| 1.1 | Неположительное число | + | |  | |  |  | |  | |  | |  |  |  |  |
| 1.2 | Нецелое число | + | |  | |  |  | |  | |  | |  |  |  |  |
| 1.3 | Не число |  | | + | |  |  | |  | |  | |  |  |  |  |
| 1.4 | Положительное целое число |  | |  | | + |  | |  | |  | |  |  |  |  |
| 2 | **Добавление элемента после определенного элемента** | | | | | | | | | | | | | | | |
| 2.1 | Определенный элемент есть |  | |  | |  | + | |  | |  | |  |  |  |  |
| 2.2 | Определенный элемента нет |  | |  | |  |  | | + | |  | |  |  |  |  |
| 3 | **Вывод** | | | | | | | | | | | | | | | |
| 3.1 | Список пуст |  | |  | |  |  | |  | | + | |  |  |  |  |
| 3.2 | Список непуст |  | |  | |  |  | |  | |  | | + |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Таблица тестов** | | | | |
| **Тест** | **Вход** | **Ожидаемый результат** | **Реальный результат** | **Примечания** |
| **Размеры случайного набора** | | | | |
| 1 | -1,37 | Сообщение об ошибке | Сообщение об ошибке |  |
| 2 | а | Сообщение об ошибке | Сообщение об ошибке |  |
| 3 | 3 | Список размера 3 | Список размера 3 |  |
| **Добавление элемента после определенного элемента** | | | | |
| 4 | Список = (Россия, 100), (США, 200)  Элемент1 = (Англия, 300)  Элемент2 = (США, 200) | Список = (Россия, 100), (США, 200), (Англия, 300) | Список = (Россия, 100), (США, 200), (Англия, 300) |  |
| 5 | Список = (Россия, 100), (США, 200)  Элемент1 = (Англия, 300)  Элемент2 = (США, 250) | Список = (Россия, 100), (США, 200) | Список = (Россия, 100), (США, 200) |  |
| **Вывод** | | | | |
| 6 | N = 0 | Список пуст | Список пуст |  |
| 7 | N = 2 | Список размера 2 | Список размера 2 |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Таблица ЧЯ (дерево)** | | | | | | | | | | | | | | | | |
| № | Проверяемая ситуация | Т1 | Т2 | | Т3 | | | Т4 | | Т5 | | Т6 | Т7 | Т8 | Т9 | Т10 |
| 1 | **Длина** | | | | | | | | | | | | | | | |
| 1.1 | Неположительное число | + | |  | |  |  | |  | |  | |  |  |  |  |
| 1.2 | Нецелое число | + | |  | |  |  | |  | |  | |  |  |  |  |
| 1.3 | Не число |  | | + | |  |  | |  | |  | |  |  |  |  |
| 1.4 | Положительное целое число |  | |  | | + |  | |  | |  | |  |  |  |  |
| 2 | **Найти кол-во листьев** | | | | | | | | | | | | | | | |
| 2.1 | Дерево пусто |  | |  | |  |  | | + | |  | |  |  |  |  |
| 2.2 | Дерево непусто |  | |  | |  | + | |  | |  | |  |  |  |  |
| 3 | **Вывод** | | | | | | | | | | | | | | | |
| 3.1 | Список пуст |  | |  | |  |  | |  | | + | |  |  |  |  |
| 3.2 | Список непуст |  | |  | |  |  | |  | |  | | + |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Таблица тестов** | | | | |
| **Тест** | **Вход** | **Ожидаемый результат** | **Реальный результат** | **Примечания** |
| **Размеры случайного набора** | | | | |
| 1 | -1,37 | Сообщение об ошибке | Сообщение об ошибке |  |
| 2 | а | Сообщение об ошибке | Сообщение об ошибке |  |
| 3 | 3 | Список размера 3 | Список размера 3 |  |
| **Найти кол-во листьев** | | | | |
| 4 | Дерево с 3 листьями | 3 | 3 |  |
| 5 | Пустое дерево | 0 | 0 |  |
| **Вывод** | | | | |
| 6 | N = 0 | Список пуст | Список пуст |  |
| 7 | N = 2 | Список размера 2 | Список размера 2 |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Таблица ЧЯ (Коллекция)** | | | | | | | | | | | | | | | | |
| № | Проверяемая ситуация | Т1 | Т2 | | Т3 | | | Т4 | | Т5 | | Т6 | Т7 | Т8 | Т9 | Т10 |
| 1 | **Длина** | | | | | | | | | | | | | | | |
| 1.1 | Неположительное число | + | |  | |  |  | |  | |  | |  |  |  |  |
| 1.2 | Нецелое число | + | |  | |  |  | |  | |  | |  |  |  |  |
| 1.3 | Не число |  | | + | |  |  | |  | |  | |  |  |  |  |
| 1.4 | Положительное целое число |  | |  | | + |  | |  | |  | |  |  |  |  |
| 2 | **Удаление элемента** | | | | | | | | | | | | | | | |
| 2.1 | Элемент есть |  | |  | |  |  | | + | |  | |  |  |  |  |
| 2.2 | Элемента нет |  | |  | |  | + | |  | |  | |  |  |  |  |
| 3 | **Вывод** | | | | | | | | | | | | | | | |
| 3.1 | Список пуст |  | |  | |  |  | |  | | + | |  |  |  |  |
| 3.2 | Список непуст |  | |  | |  |  | |  | |  | | + |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Таблица тестов** | | | | |
| **Тест** | **Вход** | **Ожидаемый результат** | **Реальный результат** | **Примечания** |
| **Размеры случайного набора** | | | | |
| 1 | -1,37 | Сообщение об ошибке | Сообщение об ошибке |  |
| 2 | а | Сообщение об ошибке | Сообщение об ошибке |  |
| 3 | 3 | Список размера 3 | Список размера 3 |  |
| **Найти кол-во листьев** | | | | |
| 4 | Элемент есть | Объект удален | Объект удален |  |
| 5 | Элемента нет | Объект не найден | Объект не найден |  |
| **Вывод** | | | | |
| 6 | N = 0 | Список пуст | Список пуст |  |
| 7 | N = 2 | Список размера 2 | Список размера 2 |  |

1. [↑](#footnote-ref-1)