|  |
| --- |
| Пермский филиал федерального государственного автономного образовательного учреждения высшего образования  «Национальный исследовательский университет  «Высшая школа экономики»  *Факультет экономики, менеджмента и бизнес-информатики* |
|  |
| Никитин Андрей Владимирович  **Отчет по практической работе №12**  **Вариант 6**  студента образовательной программы «Программная инженерия»  по направлению подготовки *09.03.04 Программная инженерия*   |  |  | | --- | --- | |  | Руководитель  О. Л. Викентьева | |

Пермь, 2020 год

**Задание**

### 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.

### Задание 2

Реализовать обобщенную коллекцию, указанную в варианте. Для этого:

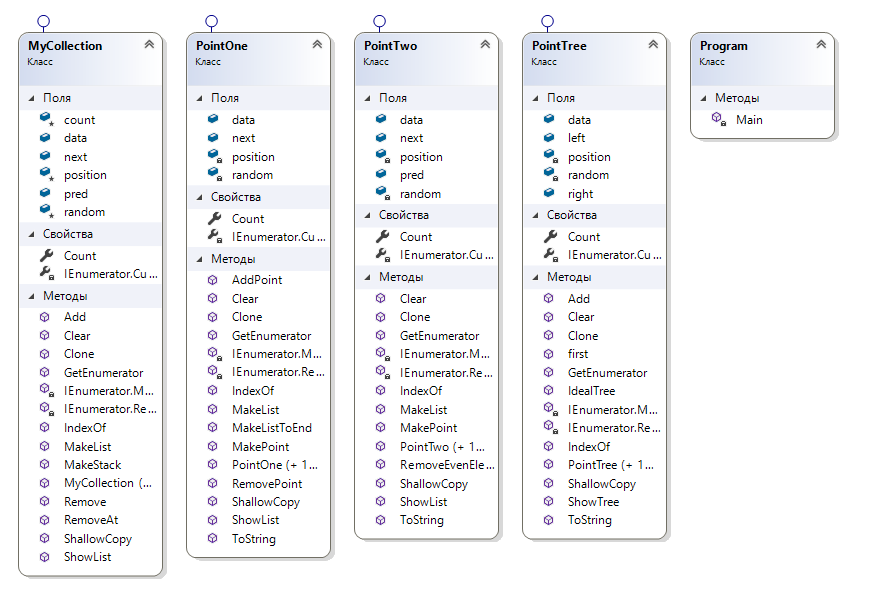
1. Реализовать конструкторы:

* 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.

**Диаграмма классов:**

****

public class **PointOne** : ICloneable, IEnumerable, IEnumerator

{

private static Random random = new Random();

private static int position = -1;

public Place data; // Информационное поле

public PointOne next; // Адресное поле

public int Count

{

get

{

if (data == null)

{

throw new Exception("Collection is empty");

}

int count = 1;

PointOne tmp = this;

while (next != null)

{

count++;

tmp = tmp.next;

}

return count;

}

}

object IEnumerator.Current

{

get

{

if (position == -1)

throw new InvalidOperationException();

return position;

}

}

// Конструктор класса без параметров

public PointOne()

{

data = null;

next = null;

}

// Конструктор класса c параметрами

public PointOne(Place place)

{

position = 0;

data = place;

next = null;

}

// Поверхостное копирование

public PointOne ShallowCopy()

{

return (PointOne)MemberwiseClone();

}

// Метод для поиска элемента по значению

public int IndexOf(Place place)

{

int index = 0;

bool check = false;

PointOne point = (PointOne) this.Clone();

while (point.next != null && !check)

{

if(point.data == place)

{

check = true;

}

index++;

point = point.next;

}

if (!check)

{

return -1;

}

return index;

}

// Клонирование

public object Clone()

{

return this.MemberwiseClone();

}

// Полное удаление коллекции

public void Clear()

{

data = null;

next = null;

}

// Создание элемента одноправленного списка

public static PointOne MakePoint(Place place)

{

PointOne point = new PointOne(place);

return point;

}

// Добавление в начало однонаправленного списка

public static PointOne MakeList(int size)

{

int typeclass = random.Next(1, 4);

PointOne beg = new PointOne();

switch (typeclass)

{

case 1:

{

Place place = new Place();

place.RandomCreate();

Console.WriteLine("The element {0} is adding...", place.ToString());

beg = MakePoint(place);

}

break;

case 2:

{

Region region = new Region();

region.RandomCreate();

Console.WriteLine("The element {0} is adding...", region.ToString());

beg = MakePoint(region);

}

break;

case 3:

{

City city = new City();

city.RandomCreate();

Console.WriteLine("The element {0} is adding...", city.ToString());

beg = MakePoint(city);

}

break;

case 4:

{

Megapolis megapolis = new Megapolis();

megapolis.RandomCreate();

Console.WriteLine("The element {0} is adding...", megapolis.ToString());

beg = MakePoint(megapolis);

}

break;

}

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

{

typeclass = random.Next(1, 4);

PointOne point = new PointOne();

switch (typeclass)

{

case 1:

{

Place place = new Place();

place.RandomCreate();

Console.WriteLine("The element {0} is adding...", place.ToString());

point = MakePoint(place);

}

break;

case 2:

{

Region region = new Region();

region.RandomCreate();

Console.WriteLine("The element {0} is adding...", region.ToString());

point = MakePoint(region);

}

break;

case 3:

{

City city = new City();

city.RandomCreate();

Console.WriteLine("The element {0} is adding...", city.ToString());

point = MakePoint(city);

}

break;

case 4:

{

Megapolis megapolis = new Megapolis();

megapolis.RandomCreate();

Console.WriteLine("The element {0} is adding...", megapolis.ToString());

point = MakePoint(megapolis);

}

break;

}

point.next = beg;

beg = point;

}

return beg;

}

// Добавление в конец однонаправленного списка

public static PointOne MakeListToEnd(int size)

{

int typeclass = random.Next(1, 4);

PointOne beg = new PointOne();

switch (typeclass)

{

case 1:

{

Place place = new Place();

place.RandomCreate();

Console.WriteLine("The element {0} is adding...", place.ToString());

beg = MakePoint(place);

}

break;

case 2:

{

Region region = new Region();

region.RandomCreate();

Console.WriteLine("The element {0} is adding...", region.ToString());

beg = MakePoint(region);

}

break;

case 3:

{

City city = new City();

city.RandomCreate();

Console.WriteLine("The element {0} is adding...", city.ToString());

beg = MakePoint(city);

}

break;

case 4:

{

Megapolis megapolis = new Megapolis();

megapolis.RandomCreate();

Console.WriteLine("The element {0} is adding...", megapolis.ToString());

beg = MakePoint(megapolis);

}

break;

}

PointOne r = beg;

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

{

typeclass = random.Next(1, 4);

PointOne point = new PointOne();

switch (typeclass)

{

case 1:

{

Place place = new Place();

place.RandomCreate();

Console.WriteLine("The element {0} is adding...", place.ToString());

point = MakePoint(place);

}

break;

case 2:

{

Region region = new Region();

region.RandomCreate();

Console.WriteLine("The element {0} is adding...", region.ToString());

point = MakePoint(region);

}

break;

case 3:

{

City city = new City();

city.RandomCreate();

Console.WriteLine("The element {0} is adding...", city.ToString());

point = MakePoint(city);

}

break;

case 4:

{

Megapolis megapolis = new Megapolis();

megapolis.RandomCreate();

Console.WriteLine("The element {0} is adding...", megapolis.ToString());

point = MakePoint(megapolis);

}

break;

}

r.next = point;

r = point;

}

return beg;

}

// Добавление элемента в одноправленный список

public static PointOne AddPoint(PointOne beg, int number)

{

int typeclass = random.Next(1, 4);

PointOne point = new PointOne();

switch (typeclass)

{

case 1:

{

Place place = new Place();

place.RandomCreate();

Console.WriteLine("The element {0} is adding...", place.ToString());

point = MakePoint(place);

}

break;

case 2:

{

Region region = new Region();

region.RandomCreate();

Console.WriteLine("The element {0} is adding...", region.ToString());

point = MakePoint(region);

}

break;

case 3:

{

City city = new City();

city.RandomCreate();

Console.WriteLine("The element {0} is adding...", city.ToString());

point = MakePoint(city);

}

break;

case 4:

{

Megapolis megapolis = new Megapolis();

megapolis.RandomCreate();

Console.WriteLine("The element {0} is adding...", megapolis.ToString());

point = MakePoint(megapolis);

}

break;

}

if (beg == null)

{

switch (typeclass)

{

case 1:

{

Place place = new Place();

place.RandomCreate();

beg = MakePoint(place);

}

break;

case 2:

{

Region region = new Region();

region.RandomCreate();

beg = MakePoint(region);

}

break;

case 3:

{

City city = new City();

city.RandomCreate();

beg = MakePoint(city);

}

break;

case 4:

{

Megapolis megapolis = new Megapolis();

megapolis.RandomCreate();

beg = MakePoint(megapolis);

}

break;

}

return beg;

}

if (number == 1)

{

point.next = beg;

beg = point;

return beg;

}

// Вспом. переменная для прохода по списку

PointOne p = beg;

// Идем по списку до нужного элемента

for (int i = 1; i < number - 1 && p != null; i++)

p = p.next;

if (p == null) //элемент не найден

{

Console.WriteLine("Error! The size of List less than Number");

return beg;

}

// Lобавляем новый элемент

point.next = p.next;

p.next = point;

return beg;

}

// Удаление элемента из одноправленного списка

public static PointOne RemovePoint(PointOne beg, int number)

{

if (beg == null) // Пустой список

{

Console.WriteLine("Error! The List is empty");

return null;

}

if (number == 1) // Удаляем первый элемент

{

beg = beg.next;

return beg;

}

PointOne p = beg;

// Ищем элемент для удаления и встаем на предыдущий

for (int i = 1; i < number - 1 && p != null; i++)

p = p.next;

if (p == null) // Если элемент не найден

{

Console.WriteLine("Error! The size of List less than Number");

return beg;

}

// исключаем элемент из списка

p.next = p.next.next;

return p;

}

// Печать одноправленного списка

public static void ShowList(PointOne beg)

{

//проверка наличия элементов в списке

if (beg == null)

{

Console.WriteLine("The List is empty");

return;

}

PointOne p = beg;

while (p != null)

{

Console.Write(p.ToString());

p = p.next;//переход к следующему элементу

}

Console.WriteLine();

}

// Вывод информационного поля

public override string ToString()

{

Console.WriteLine();

data.Show();

return "\n";

}

public IEnumerator GetEnumerator()

{

return GetEnumerator();

}

bool IEnumerator.MoveNext()

{

if (data != null && next != null)

{

position++;

return true;

}

else

{

return false;

}

}

void IEnumerator.Reset()

{

if (data != null)

{

position = 0;

}

else

{

position = -1;

}

}

}

public class PointTwo : ICloneable, IEnumerable, IEnumerator

{

private static Random random = new Random();

private static int position = -1;

public Place data; // Информационное поле

public PointTwo next, // Адрес следующего элемента

pred; // Адрес предыдущего элемента

public int Count

{

get

{

if (data == null)

{

throw new Exception("Collection is empty");

}

PointTwo point = (PointTwo)this.Clone();

while(point.pred != null)

{

point = point.pred;

}

int count = 1;

while(point.next != null)

{

count++;

point = point.next;

}

return count;

}

}

object IEnumerator.Current

{

get

{

if (position == -1)

throw new InvalidOperationException();

return position;

}

}

// Конструктор без параметров

public PointTwo()

{

data = null;

next = null;

pred = null;

}

// Конструктор с параметром

public PointTwo(Place place)

{

position = 0;

data = place;

next = null;

pred = null;

}

// Поверхостное копирование

public PointTwo ShallowCopy()

{

return (PointTwo)MemberwiseClone();

}

// Метод для поиска элемента по значению

public int IndexOf(Place place)

{

int index = 0;

bool check = false;

PointTwo point = (PointTwo)this.Clone();

while (point.pred != null)

{

point = point.pred;

}

while (point.next != null && !check)

{

if (point.data == place)

{

check = true;

}

index++;

point = point.next;

}

if (!check)

{

return -1;

}

return index;

}

// Клонирование

public object Clone()

{

return this.MemberwiseClone();

}

// Полное удаление коллекции

public void Clear()

{

data = null;

next = null;

pred = null;

}

// Создание элемента двунаправленного списка

public static PointTwo MakePoint(Place place)

{

PointTwo point = new PointTwo(place);

return point;

}

// Добавление элементов в начало двунаправленного списка

public static PointTwo MakeList(int size) //добавление в начало

{

int typeclass = random.Next(1, 4);

PointTwo beg = new PointTwo();

switch (typeclass)

{

case 1:

{

Place place = new Place();

place.RandomCreate();

Console.WriteLine("The element {0} is adding...", place.ToString());

beg = MakePoint(place);

}

break;

case 2:

{

Region region = new Region();

region.RandomCreate();

Console.WriteLine("The element {0} is adding...", region.ToString());

beg = MakePoint(region);

}

break;

case 3:

{

City city = new City();

city.RandomCreate();

Console.WriteLine("The element {0} is adding...", city.ToString());

beg = MakePoint(city);

}

break;

case 4:

{

Megapolis megapolis = new Megapolis();

megapolis.RandomCreate();

Console.WriteLine("The element {0} is adding...", megapolis.ToString());

beg = MakePoint(megapolis);

}

break;

}

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

{

typeclass = random.Next(1, 4);

PointTwo point = new PointTwo();

switch (typeclass)

{

case 1:

{

Place place = new Place();

place.RandomCreate();

Console.WriteLine("The element {0} is adding...", place.ToString());

point = MakePoint(place);

}

break;

case 2:

{

Region region = new Region();

region.RandomCreate();

Console.WriteLine("The element {0} is adding...", region.ToString());

point = MakePoint(region);

}

break;

case 3:

{

City city = new City();

city.RandomCreate();

Console.WriteLine("The element {0} is adding...", city.ToString());

point = MakePoint(city);

}

break;

case 4:

{

Megapolis megapolis = new Megapolis();

megapolis.RandomCreate();

Console.WriteLine("The element {0} is adding...", megapolis.ToString());

point = MakePoint(megapolis);

}

break;

}

point.next = beg;

beg.pred = point;

beg = point;

}

return beg;

}

// Печать двунаправленного списка

public static void ShowList(PointTwo beg)

{

// проверка наличия элементов в списке

if (beg == null)

{

Console.WriteLine("The List is empty");

return;

}

PointTwo p = beg;

// Идем к левому краю списка

while(p.pred != null)

{

p = p.pred;

}

while (p != null)

{

Console.Write(p.ToString());

p = p.next; //переход к следующему элементу

}

Console.WriteLine();

}

// УДаление элементов с четными номерами

public static PointTwo RemoveEvenElems(PointTwo beg)

{

if (beg == null) // Пустой список

{

Console.WriteLine("Error! The List is empty");

return null;

}

while (beg.pred != null)

{

beg = beg.pred;

}

int count = 1;

while (beg.next != null)

{

beg = beg.next;

count++;

}

if (count % 2 == 1)

{

beg = beg.pred;

beg.next = null;

count--;

}

PointTwo p = beg;

for (int i = 0; i < count / 2 - 1; i++)

{

p = p.pred.pred;

p.next = p.next.next;

}

p.pred = null;

return p;

}

// Печать информационного поля

public override string ToString()

{

return data.ToString() + " ";

}

public IEnumerator GetEnumerator()

{

return GetEnumerator();

}

bool IEnumerator.MoveNext()

{

if (data != null && next != null)

{

position++;

return true;

}

else

{

return false;

}

}

void IEnumerator.Reset()

{

if (data != null)

{

position = 0;

}

else

{

position = -1;

}

}

}

public class PointTree : ICloneable, IEnumerable, IEnumerator

{

private static Random random = new Random();

private static int position = -1;

object IEnumerator.Current

{

get

{

if (position == -1)

throw new InvalidOperationException();

return position;

}

}

public Place data; // Информационное поле

public PointTree left, // Aдрес левого поддерева

right; // Aдрес правого поддерева

public int Count

{

get

{

int count = 0;

PointTree point = (PointTree)this.Clone();

if (point != null)

count = 1 + left.Count + right.Count;

return count;

}

}

// Конструктор без параметров

public PointTree()

{

data = null;

left = null;

right = null;

}

// Конструктор с параметрам

public PointTree(Place place)

{

data = place;

left = null;

right = null;

}

// Поверхостное копирование

public PointTree ShallowCopy()

{

return (PointTree)MemberwiseClone();

}

// Метод для поиска элемента по значению

public bool IndexOf(Place place)

{

bool index = false;

PointTree point = (PointTree)this.Clone();

if (point.data == place)

{

return true;

}

if (point.left != null)

{

index = point.left.IndexOf(place);

}

if (!index && point.right != null)

{

index = point.right.IndexOf(place);

}

return index;

}

// Клонирование

public object Clone()

{

return this.MemberwiseClone();

}

// Полное удаление коллекции

public void Clear()

{

data = null;

left = null;

right = null;

}

// Рекурсивная функция для печати дерева по уровням с обходом слева направо

public static void ShowTree(PointTree p, int l)

{

if (p != null)

{

ShowTree(p.left, l + 3);//переход к левому поддереву

//формирование отступа

for (int i = 0; i < l; i++) Console.Write(" ");

Console.WriteLine(p.ToString());//печать узла

ShowTree(p.right, l + 3);//переход к правому поддереву

}

}

// Формирование первого элемента дерева

public static PointTree first(Place place)

{

PointTree p = new PointTree(place);

return p;

}

// Добавление элемента d в дерево поиска

public static PointTree Add(PointTree root, Place place)

{

PointTree p = root; // корень дерева

PointTree r = null;

// Флаг для проверки существования элемента d в дереве

bool ok = false;

while (p != null && !ok)

{

r = p;

// элемент уже существует

if (place == p.data)

{

ok = true;

}

else

{

if (place < p.data)

{

// Пойти в левое поддерево

p = p.left;

}

else

{

// Пойти в правое поддерево

p = p.right;

}

}

}

// Если найдено, то не добавляем

if (ok) return p;

// создаем узел с выделением памяти

PointTree NewPoint = new PointTree(place);

// Если place < r.key, то добавляем его в левое поддерево

if (place < r.data)

{

r.left = NewPoint;

}

// если place >r.key, то добавляем его в правое поддерево

else

{

r.right = NewPoint;

}

return r;

}

// Построение идеально сбалансированного дерева

public static PointTree IdealTree(int size, PointTree p)

{

PointTree r;

int nl, nr;

if (size == 0)

{

p = null;

return p;

}

nl = size / 2;

nr = size - nl - 1;

r = new PointTree();

int typeclass = random.Next(1, 4);

switch (typeclass)

{

case 1:

{

Place place = new Place();

place.RandomCreate();

Console.WriteLine("The element {0} is adding...", place.ToString());

r = first(place);

}

break;

case 2:

{

Region region = new Region();

region.RandomCreate();

Console.WriteLine("The element {0} is adding...", region.ToString());

r = first(region);

}

break;

case 3:

{

City city = new City();

city.RandomCreate();

Console.WriteLine("The element {0} is adding...", city.ToString());

r = first(city);

}

break;

case 4:

{

Megapolis megapolis = new Megapolis();

megapolis.RandomCreate();

Console.WriteLine("The element {0} is adding...", megapolis.ToString());

r = first(megapolis);

}

break;

}

r.left = IdealTree(nl, r.left);

r.right = IdealTree(nr, r.right);

return r;

}

// Печать информационного поля

public override string ToString()

{

return data.PlaceName + " ";

}

public IEnumerator GetEnumerator()

{

return GetEnumerator();

}

bool IEnumerator.MoveNext()

{

if (data != null || left != null || right != null)

{

position++;

return true;

}

else

{

return false;

}

}

void IEnumerator.Reset()

{

if (data != null)

{

position = 0;

}

else

{

position = -1;

}

}

}

**Класс MyCollection:**

public class MyCollection: ICloneable, IEnumerable, IEnumerator

{

private protected static Random random = new Random();

private protected static int count = 0;

private protected static int position = -1;

object IEnumerator.Current

{

get

{

if (position == -1)

throw new InvalidOperationException();

return position;

}

}

public Place data;

public MyCollection next;

public MyCollection pred;

public int Count

{

get { return count; }

}

public MyCollection()

{

data = null;

next = null;

pred = null;

}

public MyCollection(Place place)

{

count = 1;

data = place;

next = null;

pred = null;

}

public MyCollection(int capacity)

{

MyCollection stack = MakeStack(new Place());

for (int i = 1; i < capacity; i++)

{

MyCollection collection = new MyCollection(new Place());

collection.next = stack;

stack.pred = collection;

stack = collection;

}

count = capacity;

data = stack.pred.data;

pred = stack;

}

public MyCollection(MyCollection c)

{

data = c.data;

pred = c.pred;

next = c.next;

count = c.Count;

}

public MyCollection ShallowCopy()

{

return (MyCollection)MemberwiseClone();

}

public int IndexOf(Place place)

{

int index = 0;

MyCollection beg = this;

while (beg.pred != null) beg = beg.pred;

while(beg.next != null)

{

if (beg.data == place)

{

return index;

}

index++;

beg = beg.next;

}

if (beg.data == place)

{

return index;

}

return -1;

}

public object Clone()

{

return this.MemberwiseClone();

}

public void Clear()

{

data = null;

next = null;

pred = null;

count = 0;

}

public static MyCollection Add(MyCollection beg, Place place)

{

MyCollection collection = MakeStack(place);

collection.pred = beg;

beg.next = collection;

count++;

return beg;

}

public static MyCollection MakeList(int size)

{

int typeclass = random.Next(1, 4);

MyCollection beg = new MyCollection();

switch (typeclass)

{

case 1:

{

Place place = new Place();

place.RandomCreate();

Console.WriteLine("The element {0} is adding...", place.ToString());

beg = MakeStack(place);

}

break;

case 2:

{

Region region = new Region();

region.RandomCreate();

Console.WriteLine("The element {0} is adding...", region.ToString());

beg = MakeStack(region);

}

break;

case 3:

{

City city = new City();

city.RandomCreate();

Console.WriteLine("The element {0} is adding...", city.ToString());

beg = MakeStack(city);

}

break;

case 4:

{

Megapolis megapolis = new Megapolis();

megapolis.RandomCreate();

Console.WriteLine("The element {0} is adding...", megapolis.ToString());

beg = MakeStack(megapolis);

}

break;

}

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

{

typeclass = random.Next(1, 4);

MyCollection point = new MyCollection();

switch (typeclass)

{

case 1:

{

Place place = new Place();

place.RandomCreate();

Console.WriteLine("The element {0} is adding...", place.ToString());

point = MakeStack(place);

}

break;

case 2:

{

Region region = new Region();

region.RandomCreate();

Console.WriteLine("The element {0} is adding...", region.ToString());

point = MakeStack(region);

}

break;

case 3:

{

City city = new City();

city.RandomCreate();

Console.WriteLine("The element {0} is adding...", city.ToString());

point = MakeStack(city);

}

break;

case 4:

{

Megapolis megapolis = new Megapolis();

megapolis.RandomCreate();

Console.WriteLine("The element {0} is adding...", megapolis.ToString());

point = MakeStack(megapolis);

}

break;

}

point.next = beg;

beg.pred = point;

beg = point;

}

return beg;

}

public static MyCollection Remove(MyCollection beg)

{

MyCollection collection = beg;

while(collection.pred != null)

{

collection = collection.pred;

}

collection = collection.next;

collection.pred = null;

count--;

return collection;

}

public static MyCollection RemoveAt(MyCollection beg, int count)

{

MyCollection collection = beg;

while (collection.next != null) collection = collection.next;

for(int i = collection.Count; i > collection.Count - count; i--)

{

collection = collection.pred;

}

collection.next = null;

return collection;

}

public static MyCollection MakeStack(Place place)

{

MyCollection stack = new MyCollection(place);

return stack;

}

public static void ShowList(MyCollection beg)

{

// проверка наличия элементов в списке

if (beg == null)

{

Console.WriteLine("The List is empty");

return;

}

MyCollection p = beg;

// Идем к левому краю списка

while (p.pred != null)

{

p = p.pred;

}

while (p != null)

{

Console.Write(p.ToString());

p = p.next; //переход к следующему элементу

}

Console.WriteLine();

}

public IEnumerator GetEnumerator()

{

return GetEnumerator();

}

bool IEnumerator.MoveNext()

{

if (data != null && next != null)

{

position++;

return true;

}

else

{

return false;

}

}

void IEnumerator.Reset()

{

if (data != null)

{

position = 0;

}

else

{

position = -1;

}

}

}

**Пример демонстрационной программы:**

static void Main(string[] args)

{

Console.WriteLine("Часть 1");

Console.WriteLine("Работа с одноправленным списоком");

PointOne one = new PointOne();

one = PointOne.MakeList(10);

Console.WriteLine("Одноправленный список");

PointOne.ShowList(one);

Console.WriteLine("Добавление в конец");

one = PointOne.MakeListToEnd(7);

Console.WriteLine("Одноправленный список");

PointOne.ShowList(one);

Console.WriteLine();

Console.WriteLine("Работа с двунаправленным списоком");

PointTwo two = new PointTwo();

two = PointTwo.MakeList(10);

Console.WriteLine("Двунаправленный список");

PointTwo.ShowList(two);

Console.WriteLine("Удаление четных элементов");

two = PointTwo.RemoveEvenElems(two);

Console.WriteLine("Двунаправленный список");

PointTwo.ShowList(two);

Console.WriteLine();

Console.WriteLine("Работа с деревоом");

PointTree tree = new PointTree();

Console.WriteLine("Создание сбалансированного дерева");

tree = PointTree.IdealTree(10, tree);

Console.WriteLine("Дерево");

PointTree.ShowTree(tree, 200);

}

Ссылка на ГитХаб:

<https://github.com/Anni-HSE/Place-and-Collections>

1. за реализацию метода для удаления элемента из дерева ставятся дополнительные баллы [↑](#footnote-ref-1)