Текст программы в лабораторной работе №12

Класс Program

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using LibraryLaba10;

namespace laba12

{

class Program

{

static void Main(string[] args)

{

Point p = null;

BinaryTree bt = null;

BinaryTree root = null;

LPoint[] ht = null;

MyCollection<Challenge> collStack = null;

int choiceMainMenu;

do

{

Menu.MainMenu();

choiceMainMenu = LibraryLaba10.UserInterface.EnterInt(1, 5, "Выберите пункт меню: ");

switch (choiceMainMenu)

{

case 1:

{

#region Выполнение меню для двунаправленного списка

int choicePointMenu;

do

{

Menu.PointMenu();

choicePointMenu = LibraryLaba10.UserInterface.EnterInt(1, 5, "Выберите пункт меню: ");

switch (choicePointMenu)

{

case 1:

p = Point.MakeList(LibraryLaba10.UserInterface.EnterInt(1, 1000, "Введите кол-во элементов в списке: "));

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Двунаправленный список сформирован");

Console.ResetColor();

break;

case 2:

if (p != null)

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Получен двунаправленный список");

Console.ResetColor();

}

Point.PrintList(p);

break;

case 3: p = Point.DeletePoint(p); break;

case 4:

if (p != null)

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Очищение списка...");

p = null;

GC.Collect(); //принудительный запуск сборки мусора

Console.WriteLine("Список очищен");

Console.ResetColor();

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Двунаправленный список пуст");

Console.ResetColor();

}

break;

}

} while (choicePointMenu < 5);

break;

#endregion

}

case 2:

{

#region Выполнение меню для бинарного дерева

int choiceBinaryTreeMenu;

do

{

Menu.BinaryTreeMenu();

choiceBinaryTreeMenu = LibraryLaba10.UserInterface.EnterInt(1, 7, "Выберите пункт меню: ");

switch (choiceBinaryTreeMenu)

{

case 1:

Random rnd = new Random();

bt = BinaryTree.IdealTree(LibraryLaba10.UserInterface.EnterInt(1, 1000, "Введите кол-во элементов в дереве: "), bt);

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Идеально сбалансированное бинарное дерево сформировано");

Console.ResetColor();

break;

case 2:

if (bt != null)

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Получено дерево");

Console.ResetColor();

}

BinaryTree.PrintTree(bt, 1);

break;

case 3:

if (bt == null)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Дерево пустое");

Console.ResetColor();

}

else

{

Console.WriteLine($"Высота дерева = {bt.Run(bt)}");

}

break;

case 4:

if (bt == null)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Дерево пустое");

Console.ResetColor();

}

else

{

root = BinaryTree.First();

BinaryTree.Transform(root, bt);

}

break;

case 5:

if (root != null)

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Получено дерево");

Console.ResetColor();

}

BinaryTree.PrintTree(root, 1);

break;

case 6:

if (bt != null)

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Очищение дерева...");

bt = null;

GC.Collect(); //принудительный запуск сборки мусора

Console.WriteLine("Дерево очищено");

Console.ResetColor();

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Дерево пустое");

Console.ResetColor();

}

if (root != null)

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Очищение дерева поиска...");

root = null;

GC.Collect(); //принудительный запуск сборки мусора

Console.WriteLine("Дерево очищено");

Console.ResetColor();

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Дерево поиска пустое");

Console.ResetColor();

}

break;

}

} while (choiceBinaryTreeMenu < 7);

break;

#endregion

}

case 3:

{

#region Выполнение меню для хеш-таблицы

int choiceHTableMenu;

do

{

Menu.HashTableMenu();

choiceHTableMenu = LibraryLaba10.UserInterface.EnterInt(1, 7, "Выберите пункт меню: ");

switch (choiceHTableMenu)

{

case 1:

ht = HTable.MakeTable(LibraryLaba10.UserInterface.EnterInt(1, 1000, "Введите кол-во элементов в хеш-таблице: "));

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Хеш-таблица сформирована");

Console.ResetColor();

break;

case 2:

if (ht == null)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Таблица пуста");

Console.ResetColor();

}

else

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Получена хеш-таблица");

Console.ResetColor();

HTable.Print(ht);

}

break;

case 3:

if (ht == null)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Таблица пуста");

Console.ResetColor();

}

else

{

ht = HTable.FindPoint(0, ht);

}

break;

case 4:

if (ht == null)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Таблица пуста");

Console.ResetColor();

}

else

{

ht = HTable.FindPoint(1, ht);

}

break;

case 5:

ht = HTable.Append(ht);

break;

case 6:

if (ht != null)

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Очищение хеш-таблицы...");

ht = null;

GC.Collect(); //принудительный запуск сборки мусора

Console.WriteLine("Таблица очищена");

Console.ResetColor();

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Таблица пуста");

Console.ResetColor();

}

break;

}

} while (choiceHTableMenu < 7);

break;

#endregion

}

case 4:

{

#region Выполнение меню для коллекции

int choiceCollMenu;

do

{

Menu.CollMenu();

choiceCollMenu = LibraryLaba10.UserInterface.EnterInt(1, 10, "Выберите пункт меню: ");

switch (choiceCollMenu)

{

case 1:

collStack = new MyCollection<Challenge>(LibraryLaba10.UserInterface.EnterInt(1, 1000, "Введите кол-во элементов в коллекции: "));

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Коллекция сформирована");

Console.ResetColor();

break;

case 2:

if (collStack != null)

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Получена коллекция");

Console.ResetColor();

}

MyCollection<Challenge>.PrintList(collStack);

break;

case 3:

collStack = MyCollection<Challenge>.Add(collStack);

break;

case 4:

collStack = MyCollection<Challenge>.DeletePoint(collStack);

break;

case 5:

MyCollection<Challenge>.Search(collStack);

break;

case 6:

if (collStack == null)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Коллекция пуста");

Console.ResetColor();

}

else

{

Console.Clear();

//collStack = MyCollection<Challenge>.Add(collStack);

MyCollection<Challenge> collStackClone = MyCollection<Challenge>.Clone(collStack);

//collStack = MyCollection<Challenge>.DeletePoint(collStack);

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("КЛОН\n");

Console.ResetColor();

MyCollection<Challenge>.PrintList(collStackClone);

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("\nИЗНАЧАЛЬНАЯ КОЛЛЕКЦИЯ\n");

Console.ResetColor();

MyCollection<Challenge>.PrintList(collStack);

}

break;

case 7:

if (collStack == null)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Коллекция пуста");

Console.ResetColor();

}

else

{

Console.Clear();

//collStack = MyCollection<Challenge>.Add(collStack);

MyCollection<Challenge> collStackCopy = MyCollection<Challenge>.Copy(collStack);

//collStack = MyCollection<Challenge>.DeletePoint(collStack);

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("КОПИЯ\n");

Console.ResetColor();

MyCollection<Challenge>.PrintList(collStackCopy);

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("\nИЗНАЧАЛЬНАЯ КОЛЛЕКЦИЯ\n");

Console.ResetColor();

MyCollection<Challenge>.PrintList(collStack);

}

break;

case 8:

if (collStack == null)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Коллекция пуста");

Console.ResetColor();

}

else

{

foreach (MyCollection<Challenge> obj in collStack)

{

Console.WriteLine(obj.data.Print());

}

}

break;

case 9:

if (collStack == null)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Коллекция пуста");

Console.ResetColor();

}

else

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Удаление коллекции...");

collStack = null;

GC.Collect(); //принудительный запуск сборки мусора

Console.WriteLine("Коллекция удалена");

Console.ResetColor();

}

break;

}

} while (choiceCollMenu < 10);

break;

#endregion

}

}

} while (choiceMainMenu < 5);

}

}

}

Класс Point

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using LibraryLaba10;

namespace laba12

{

class Point

{

#region Определение двунаправленного списка

public Challenge data;

public Point next, pred;

public Point()

{

data = new Challenge();

next = null;

pred = null;

}

public Point(string \_name, int \_grade, int \_id)

{

data = new Challenge(\_name, \_grade, \_id);

next = null;

pred = null;

}

public override string ToString()

{

return data.Print();

}

#endregion

#region Формирование двунаправленного списка

static Point MakePoint(string \_name, int \_grade, int \_id)

{

Point p = new Point(\_name, \_grade, \_id);

return p;

}

public static Point MakeList(int size)

{

Random rnd = new Random();

string[] subject = { "Дискретная математика", "Программирование", "Анатомия головного мозга", "Альдегиды и кетоны", "Устойчивые конструкции в гражданском строительстве", "ВВП современных государств", "Паразитология", "Внешняя политика Александра II", "Русский язык", "Термодинамика", "Молекулы", "Балистика", "Биология", "Равноускоренное движение", "Закон сохранения импульса", "Закон сохранения энергии", "Законы Ньютона", "Силы в природе", "Электростатика", "Электродинамика", "История", "Переменный ток", "Индукция", "Молекулярно-кинетическая теория", "Комплексные числа", "История XX века", "Теория графов", "Исследование функции", "Диверсионный анализ", "История строительства в США", "История Западной Европы в конце XIX века", "Области пространства", "Лямблии", "Банки и их история", "Журналистика", "Решение задач по микроэкономике", "Логарифмы функций", "Функции нескольких переменных", "Синтаксис", "Пунктуация", "Научный метод познания", "География Пермского края", "История Великой Отечественной войны", "Распределительные системы", "Машина Тьюринга", "История современной медицины", "Квантовое бессмертие", "Теория Шредингера", "Экономика", "Культура Китая", "Математический анализ", "Линейная алгебра", "Основы программирования на Python", "История медиакультуры", "Отмена Крепостного права в Росии", "Отмена рабовладельчества в США", "Урочные лета", "Екатерина Великая", "Петр Великий", "История Первой мировой войны", "Детская психология", "Развитие плода во время беременности", "Эконочиеские показатели", "Социальная сфера общества", "Духовная сфера общества", "Экономическая сфера общества", "Политическая сфера общества", "Литература Золотого века", "Литература Серебряного века", "Произведения Л.Н. Толстого", "Басни Крылова", "Массовая культура", "Рюриковичи", "Романовы", "Реформы Столыпина", "Убийство царской семьи", "Единицы измерения", "Множества", "Десятичные дроби", "Задачи на движение", "Умножение и деление", "Смешанные числа", "Уравнения", "Параллелепипед", "Проценты", "Периметр и площадь", "Делимость чисел", "Геометрические фигуры", "Системы счисления", "Круги Эйлера-Венна", "Координаты на плоскости", "Диаграммы и графики", "Модуль числа", "Одночлены и многочлены", "Треугольники", "Формулы сокращенного умножения", "Движение по окружности", "Квадратные уравнения", "Биквадратные уравнения", "Области определения", "Подобные треугольники", "Теорема Виета", "Комбинаторика", "Центральные и вписанные углы", "Геометрическая прогрессия", "Арифметическая прогрессия", "Векторы", "Скалярное произведение векторов", "История СССР", "География" };

Point beg = MakePoint(subject[rnd.Next(0, 110)], rnd.Next(0, 11), rnd.Next(0, Int32.MaxValue));

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

{

Point p = MakePoint(subject[rnd.Next(0, 110)], rnd.Next(0, 11), rnd.Next(0, Int32.MaxValue));

p.next = beg;

beg.pred = p;

beg = p;

}

return beg;

}

#endregion

#region Методы класса

public static void PrintList(Point beg)

{

if (beg == null)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Двунаправленный список пуст");

Console.ResetColor();

}

else

{

Point p = beg;

while (p != null)

{

Console.WriteLine(p.data.Print());

Console.WriteLine();

p = p.next;

}

}

}

public static Point DeletePoint(Point beg)

{

if (beg == null)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Двунаправленный список пуст");

Console.ResetColor();

return beg;

}

else

{

bool isDel = false;

Console.ForegroundColor = ConsoleColor.Yellow;

Console.Write("Введите название предмета на удаление из списка: ");

Console.ResetColor();

string findSub = Console.ReadLine();

int findGrade = LibraryLaba10.UserInterface.EnterInt(0, 10, "Введите оценку предмета на удаление из списка: ");

Point p = beg;

if (beg.data.Name == findSub && beg.data.Grade == findGrade)

{

beg = beg.next;

p = beg;

isDel = true;

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Элемент удален");

Console.ResetColor();

}

while (p != null)

{

if (p.data.Name == findSub && p.data.Grade == findGrade)

{

p.pred.next = p.next;

isDel = true;

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Элемент удален");

Console.ResetColor();

}

p = p.next;

}

if (!isDel)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Данные элементы отсутствуют");

Console.ResetColor();

return beg;

}

return beg;

}

}

#endregion

}

}

Класс BinaryTree

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using LibraryLaba10;

namespace laba12

{

class BinaryTree

{

#region Определение дерева

public Challenge data;

public BinaryTree left, right; //адресное поле левого и правого поддерева

int countHeight = 0;

public BinaryTree()

{

data = new Challenge();

left = null;

right = null;

}

public BinaryTree(string \_name, int \_grade, int \_id)

{

data = new Challenge(\_name, \_grade, \_id);

left = null;

right = null;

}

public override string ToString()

{

return data.Print();

}

#endregion

#region Методы класса

public static void PrintTree(BinaryTree bt, int l)

{

if (bt == null)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Дерево пустое");

Console.ResetColor();

}

else

{

if (bt.left != null) PrintTree(bt.left, l + 3); //переход к левому поддереву

//формирование отступа

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

Console.WriteLine(bt.data.Print()); //печать узла

if (bt.right != null) PrintTree(bt.right, l + 3); //переход к правому поддереву

}

}

public int Run(BinaryTree bt)

{

if (bt != null)

{

this.countHeight++;

Run(bt.left);

}

return this.countHeight;

}

public static BinaryTree IdealTree(int size, BinaryTree bt)

{

Challenge[] mas = new Challenge[size];

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

{

mas[i] = new Challenge();

}

bool isCorrect = false;

int num = 0;

BinaryTree r;

Challenge c = new Challenge();

int nl, nr;

if (size == 0) { bt = null; return bt; }

nl = size / 2;

nr = size - nl - 1;

while (!isCorrect)

{

isCorrect = false;

c = (Challenge)c.Init();

for (int j = 0; j < size; j++)

{

if (mas[j].Name != c.Name && mas[j].Grade != c.Grade)

{

isCorrect = true;

break;

}

}

if (isCorrect)

{

mas[num] = c;

num++;

}

}

r = new BinaryTree(c.Name, c.Grade, c.ID);

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

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

return r;

}

public static BinaryTree First()

{

Random rnd = new Random();

BinaryTree p = new BinaryTree("root", 5, 1073741824);

return p;

}

public static BinaryTree Add(BinaryTree root, string \_name, int \_grade, int \_id)

{

BinaryTree p = root;

BinaryTree r = null;

bool ok = false;

while (p != null && !ok)

{

r = p;

if ((\_name == p.data.Name) && (\_grade == p.data.Grade) && (\_id == p.data.ID)) ok = true;

else

if (\_grade < p.data.Grade) p = p.left;

else p = p.right;

}

if (ok) return p;

BinaryTree NewPoint = new BinaryTree(\_name, \_grade, \_id);

if (\_grade < r.data.Grade) r.left = NewPoint;

else r.right = NewPoint;

return NewPoint;

}

public static void Transform(BinaryTree root, BinaryTree idtree)

{

if (idtree != null)

{

Add(root, idtree.data.Name, idtree.data.Grade, idtree.data.ID);

Transform(root, idtree.left);

Transform(root, idtree.right);

}

}

#endregion

}

}

Класс LPoint и HTable

using LibraryLaba10;

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace laba12

{

//элемент хеш-таблицы

class LPoint

{

public int key;

public Challenge value;

public LPoint next;

public LPoint(string \_name, int \_grade)

{

Random rnd = new Random();

value = new Challenge(\_name, \_grade, rnd.Next(0, Int32.MaxValue));

key = GetHashCode();

next = null;

}

public override string ToString()

{

return key + ":" + value.ToString();

}

public override int GetHashCode()

{

int code = 0;

foreach (char c in value.Name.ToString())

code += (int)c;

code += value.Grade;

return code;

}

}

//конец класса LPoint

//хеш-таблица

abstract class HTable

{

public static int Size { get; set; }

public static int Length { get; }

public int Count { get; set; }

public static LPoint[] table;

public static LPoint[] newTable;

//public int Size;

public HTable(int size)

{

Size = size;

table = new LPoint[Size];

}

public static bool IsFind(LPoint tableElem, LPoint[] tb)

{

int code = Math.Abs(tableElem.GetHashCode()) % tb.Length;

Console.WriteLine($"{tableElem.key} {Math.Abs(tableElem.GetHashCode())} {tableElem.value.Print()} {code}");

if (tb[code] != null && String.Compare(tableElem.ToString(), tb[code].ToString()) == 0) return true;

else

{

for (int i = code; i < tb.Length; i++)

{

if (tb[i] != null && String.Compare(tableElem.ToString(), tb[i].ToString()) == 0) return true;

}

}

return false;

}

public static int NullCount(LPoint[] tb)

{

int count = 0;

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

{

if (tb[i] == null) count++;

}

return count;

}

public static LPoint[] Resize(LPoint[] tb)

{

newTable = new LPoint[tb.Length \* 2];

ref LPoint[] tableRef = ref tb;

tableRef = new LPoint[tb.Length \* 2];

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

{

if (tb[i] != null)

{

Challenge elem = new Challenge(tb[i].value.Name, tb[i].value.Grade, tb[i].value.ID);

newTable = Add(elem, newTable);

}

}

Console.WriteLine(newTable);

tb = newTable;

Console.WriteLine("hhh");

Console.WriteLine(tb.Length);

return tb;

}

public static LPoint[] Add(Challenge elem, LPoint[] tb)

{

LPoint point = new LPoint(elem.Name, elem.Grade);

if (IsFind(point, tb)) return tb;

else

{

Console.WriteLine(tb.Length);

Console.WriteLine($"{(float)(tb.Length - NullCount(tb)) / tb.Length}");

int index = Math.Abs(point.GetHashCode()) % tb.Length;

if (tb[index] == null)

{

tb[index] = point;

return tb;

}

else

{

for (int i = index; i < tb.Length; i++)

{

if (tb[i] == null)

{

tb[i] = point;

return tb;

}

}

}

//if (!isAdd && table[table.Length - 1] != null)

//{

// Console.WriteLine("lkke");

// int begin = table.Length;

// table = Resize();

// for (int i = begin; i < table.Length \* 2; i++)

// {

// if (table[i] == null)

// {

// table[i] = point;

// isAdd = true;

// break;

// }

// }

//}

}

return tb;

//else

//{

// LPoint cur = table[index];

// while (!isAdd)

// {

// }

// //LPoint cur = table[index];

// //if (string.Compare(cur.ToString(), point.ToString()) == 0) return false;

// //while (cur.next != null)

// //{

// // if (string.Compare(cur.ToString(), point.ToString()) == 0) return false;

// //}

// //cur.next = point;

//}

//return true;

}

public static LPoint[] CreateElements()

{

Challenge c = (Challenge)new Challenge().Init();

table = Add(c, table);

//if (Math.Abs(((float)table.Length - NullCount(table) + 1) / table.Length) > 0.73)

//{

// table = Resize(table);

//}

return table;

}

public static LPoint[] MakeTable(int size)

{

Size = size;

table = new LPoint[Size];

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

{

table = CreateElements();

}

if (NullCount(table) != 0)

{

table = Resize(table);

for (int j = 0; j < NullCount(table); j++)

{

table = CreateElements();

}

}

Console.WriteLine(table.Length);

return table;

}

public static void Print(LPoint[] tb)

{

if (tb == null) Console.WriteLine("Таблица пустая!");

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

{

if (tb[i] == null) Console.WriteLine(i + " : ");

else

{

Console.Write(i + " : ");

LPoint p = tb[i];

while (p != null)

{

Console.Write(p.value.Print() + "\t");

p = p.next;

}

Console.WriteLine();

}

}

}

public static LPoint[] FindPoint(int mark, LPoint[] tb)

{

if (tb == null) Console.WriteLine("Таблица пустая!");

else

{

bool isFind = false;

Console.Write("Введите название испытания: ");

string \_name = Console.ReadLine();

int \_grade = LibraryLaba10.UserInterface.EnterInt(0, 10, "Введите оценку: ");

LPoint lp = new LPoint(\_name, \_grade);

int code = Math.Abs(lp.GetHashCode()) % tb.Length;

if (tb[code] != null && String.Compare(tb[code].value.Name.ToString(), \_name) == 0 && String.Compare(tb[code].value.Grade.ToString(), \_grade.ToString()) == 0)

{

Console.WriteLine(tb[code].value.Print());

isFind = true;

if (mark == 1)

{

tb[code] = null;

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Элемент удален");

Console.ResetColor();

if (NullCount(tb) == tb.Length) tb = null;

}

}

else

{

lp = tb[code];

for (int i = code; i < tb.Length; i++)

{

if (tb[code] != null && String.Compare(tb[i].value.Name.ToString(), \_name) == 0 && String.Compare(tb[i].value.Grade.ToString(), \_grade.ToString()) == 0)

{

Console.WriteLine(tb[i].value.Print());

isFind = true;

if (mark == 1)

{

table[code] = null;

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Элемент удален");

Console.ResetColor();

if (NullCount(tb) == tb.Length) tb = null;

}

break;

}

}

}

if (!isFind)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Элемент не найден");

Console.ResetColor();

}

}

return tb;

}

public static bool EmptyPosAfter(int code, LPoint[] tb)

{

for (int i = code; i < tb.Length; i++)

{

if (tb[i] == null) return true;

}

return false;

}

public static LPoint[] Append(LPoint[] tb)

{

if (tb == null)

{

tb = MakeTable(1);

return tb;

}

Challenge c = (Challenge)new Challenge().Init();

LPoint point = new LPoint(c.Name, c.Grade);

int code = point.GetHashCode();

if (NullCount(tb) == 0) tb = Resize(tb);

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

{

tb[i] = table[i];

}

tb = Add(c, tb);

table = new LPoint[tb.Length];

for (int j = 0; j < tb.Length; j++)

{

table[j] = tb[j];

}

return table;

}

} //конец класса HTable

}

Класс MyCollection

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using LibraryLaba10;

namespace laba12

{

class MyCollection<T> : IEnumerable

{

public Challenge data;

public MyCollection<T> next;

public int Count { get; set; }

public MyCollection()

{

data = null;

next = null;

Count = 0;

}

public MyCollection(int capacity)

{

MyCollection<T> tmp = MakeStack(capacity);

data = tmp.data;

next = tmp.next;

Count = capacity;

}

public MyCollection(string \_name, int \_grade, int \_id)

{

data = new Challenge(\_name, \_grade, \_id);

next = null;

}

public MyCollection(MyCollection<T> c)

{

data = c.data;

next = c.next;

Count = c.Count;

}

static MyCollection<T> MakePoint(string \_name, int \_grade, int \_id)

{

MyCollection<T> p = new MyCollection<T>(\_name, \_grade, \_id);

return p;

}

public static MyCollection<T> MakeStack(int size)

{

Random rnd = new Random();

string[] subject = { "Дискретная математика", "Программирование", "Анатомия головного мозга", "Альдегиды и кетоны", "Устойчивые конструкции в гражданском строительстве", "ВВП современных государств", "Паразитология", "Внешняя политика Александра II", "Русский язык", "Термодинамика", "Молекулы", "Балистика", "Биология", "Равноускоренное движение", "Закон сохранения импульса", "Закон сохранения энергии", "Законы Ньютона", "Силы в природе", "Электростатика", "Электродинамика", "История", "Переменный ток", "Индукция", "Молекулярно-кинетическая теория", "Комплексные числа", "История XX века", "Теория графов", "Исследование функции", "Диверсионный анализ", "История строительства в США", "История Западной Европы в конце XIX века", "Области пространства", "Лямблии", "Банки и их история", "Журналистика", "Решение задач по микроэкономике", "Логарифмы функций", "Функции нескольких переменных", "Синтаксис", "Пунктуация", "Научный метод познания", "География Пермского края", "История Великой Отечественной войны", "Распределительные системы", "Машина Тьюринга", "История современной медицины", "Квантовое бессмертие", "Теория Шредингера", "Экономика", "Культура Китая", "Математический анализ", "Линейная алгебра", "Основы программирования на Python", "История медиакультуры", "Отмена Крепостного права в Росии", "Отмена рабовладельчества в США", "Урочные лета", "Екатерина Великая", "Петр Великий", "История Первой мировой войны", "Детская психология", "Развитие плода во время беременности", "Эконочиеские показатели", "Социальная сфера общества", "Духовная сфера общества", "Экономическая сфера общества", "Политическая сфера общества", "Литература Золотого века", "Литература Серебряного века", "Произведения Л.Н. Толстого", "Басни Крылова", "Массовая культура", "Рюриковичи", "Романовы", "Реформы Столыпина", "Убийство царской семьи", "Единицы измерения", "Множества", "Десятичные дроби", "Задачи на движение", "Умножение и деление", "Смешанные числа", "Уравнения", "Параллелепипед", "Проценты", "Периметр и площадь", "Делимость чисел", "Геометрические фигуры", "Системы счисления", "Круги Эйлера-Венна", "Координаты на плоскости", "Диаграммы и графики", "Модуль числа", "Одночлены и многочлены", "Треугольники", "Формулы сокращенного умножения", "Движение по окружности", "Квадратные уравнения", "Биквадратные уравнения", "Области определения", "Подобные треугольники", "Теорема Виета", "Комбинаторика", "Центральные и вписанные углы", "Геометрическая прогрессия", "Арифметическая прогрессия", "Векторы", "Скалярное произведение векторов", "История СССР", "География" };

MyCollection<T> beg = MakePoint(subject[rnd.Next(0, 110)], rnd.Next(0, 11), rnd.Next(0, Int32.MaxValue));

MyCollection<T> r = beg;

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

{

MyCollection<T> p = MakePoint(subject[rnd.Next(0, 110)], rnd.Next(0, 11), rnd.Next(0, Int32.MaxValue));

r.next = p;

r = p;

}

return beg;

}

public static void PrintList(MyCollection<T> beg)

{

if (beg == null)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Коллекция пуста");

Console.ResetColor();

}

else

{

MyCollection<T> p = beg;

while (p != null)

{

Console.WriteLine(p.data.Print() + "\n");

p = p.next;

}

Console.WriteLine($"Размер коллекции: {beg.Count}");

}

}

public static MyCollection<T> Add(MyCollection<T> c)

{

Random rnd = new Random();

string[] subject = { "Дискретная математика", "Программирование", "Анатомия головного мозга", "Альдегиды и кетоны", "Устойчивые конструкции в гражданском строительстве", "ВВП современных государств", "Паразитология", "Внешняя политика Александра II", "Русский язык", "Термодинамика", "Молекулы", "Балистика", "Биология", "Равноускоренное движение", "Закон сохранения импульса", "Закон сохранения энергии", "Законы Ньютона", "Силы в природе", "Электростатика", "Электродинамика", "История", "Переменный ток", "Индукция", "Молекулярно-кинетическая теория", "Комплексные числа", "История XX века", "Теория графов", "Исследование функции", "Диверсионный анализ", "История строительства в США", "История Западной Европы в конце XIX века", "Области пространства", "Лямблии", "Банки и их история", "Журналистика", "Решение задач по микроэкономике", "Логарифмы функций", "Функции нескольких переменных", "Синтаксис", "Пунктуация", "Научный метод познания", "География Пермского края", "История Великой Отечественной войны", "Распределительные системы", "Машина Тьюринга", "История современной медицины", "Квантовое бессмертие", "Теория Шредингера", "Экономика", "Культура Китая", "Математический анализ", "Линейная алгебра", "Основы программирования на Python", "История медиакультуры", "Отмена Крепостного права в Росии", "Отмена рабовладельчества в США", "Урочные лета", "Екатерина Великая", "Петр Великий", "История Первой мировой войны", "Детская психология", "Развитие плода во время беременности", "Эконочиеские показатели", "Социальная сфера общества", "Духовная сфера общества", "Экономическая сфера общества", "Политическая сфера общества", "Литература Золотого века", "Литература Серебряного века", "Произведения Л.Н. Толстого", "Басни Крылова", "Массовая культура", "Рюриковичи", "Романовы", "Реформы Столыпина", "Убийство царской семьи", "Единицы измерения", "Множества", "Десятичные дроби", "Задачи на движение", "Умножение и деление", "Смешанные числа", "Уравнения", "Параллелепипед", "Проценты", "Периметр и площадь", "Делимость чисел", "Геометрические фигуры", "Системы счисления", "Круги Эйлера-Венна", "Координаты на плоскости", "Диаграммы и графики", "Модуль числа", "Одночлены и многочлены", "Треугольники", "Формулы сокращенного умножения", "Движение по окружности", "Квадратные уравнения", "Биквадратные уравнения", "Области определения", "Подобные треугольники", "Теорема Виета", "Комбинаторика", "Центральные и вписанные углы", "Геометрическая прогрессия", "Арифметическая прогрессия", "Векторы", "Скалярное произведение векторов", "История СССР", "География" };

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Элемент добавлен");

Console.ResetColor();

if (c == null)

{

c = new MyCollection<T>(subject[rnd.Next(0, 110)], rnd.Next(0, 11), rnd.Next(0, Int32.MaxValue));

c.Count = c.Count + 1;

return c;

}

else

{

MyCollection<T> tmp = new MyCollection<T>(subject[rnd.Next(0, 110)], rnd.Next(0, 11), rnd.Next(0, Int32.MaxValue));

tmp.next = c;

tmp.Count = c.Count + 1;

return tmp;

}

}

public static MyCollection<T> DeletePoint(MyCollection<T> beg)

{

if (beg == null)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Коллекция пуста");

Console.ResetColor();

}

else

{

if (beg.next == null)

{

beg.Count--;

return beg.next;

}

beg.next.Count = beg.Count - 1;

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Элемент удален");

Console.ResetColor();

return beg.next;

}

return beg;

}

public static void Search(MyCollection<T> beg)

{

MyCollection<T> p = beg;

bool isFind = false;

if (beg == null)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Коллекция пуста");

Console.ResetColor();

}

else

{

Console.Write("Введите название испытания: ");

string \_name = Console.ReadLine();

int \_grade = LibraryLaba10.UserInterface.EnterInt(0, 10, "Введите оценку за испытание: ");

while (p != null)

{

if (p.data.Name == \_name && p.data.Grade == \_grade)

{

isFind = true;

Console.WriteLine(p.data.Print());

}

p = p.next;

}

if (!isFind)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Элемент(ы) не найден(ы)");

Console.ResetColor();

}

}

}

public static MyCollection<T> Copy(MyCollection<T> tocopy)

{

MyCollection<T> copy = new MyCollection<T>(tocopy);

return copy;

}

public static MyCollection<T> Clone(MyCollection<T> toclone)

{

if (toclone == null)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("[Ошибка!] Коллекция пуста");

Console.ResetColor();

return null;

}

else

{

MyCollection<T> r = toclone;

while (r.next != null)

{

r = r.next;

}

MyCollection<T> clone = new MyCollection<T>(r.data.Name, r.data.Grade, r.data.ID);

clone.Count = 1;

r = toclone;

for (int i = 0; i < toclone.Count - 1; i++)

{

while (!(r.next.data.Grade == clone.data.Grade && r.next.data.Name == clone.data.Name))

{

r = r.next;

}

MyCollection<T> tmp = new MyCollection<T>(r.data.Name, r.data.Grade, r.data.ID);

tmp.next = clone;

clone = tmp;

r = toclone;

}

clone.Count = toclone.Count;

return clone;

}

}

public static void Clear(ref MyCollection<T> c)

{

c = null;

GC.Collect();

}

public IEnumerator GetEnumerator()

{

return new MyNumerator<T>(this);

}

}

}

Класс MyNumerator

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using LibraryLaba10;

namespace laba12

{

class MyNumerator<T> : IEnumerator

{

MyCollection<T> beg;

MyCollection<T> current;

public MyNumerator(MyCollection<T> collection)

{

beg = collection;

current = null;

}

object IEnumerator.Current

{

get

{

return current;

}

}

bool IEnumerator.MoveNext()

{

if (current == null)

{

current = beg;

}

else

{

current = current.next;

}

return current != null;

}

void IEnumerator.Reset()

{

current = this.beg;

}

}

}

Класс Menu

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace laba12

{

class Menu

{

#region Общее меню

static public void MainMenu()

{

Console.WriteLine("\n1. Двунаправленный список" +

"\n2. Бинарное дерево" +

"\n3. Хеш-таблица" +

"\n4. Обобщенная коллекция" +

"\n5. Завершение работы\n");

}

#endregion

#region Меню для двунаправленного списка

static public void PointMenu()

{

Console.WriteLine("\n1. Сформировать двунаправленный список" +

"\n2. Распечатать двунаправленный список" +

"\n3. Удалить элементы из двунаправленного списка" +

"\n4. Удалить двунаправленный список из памяти" +

"\n5. Назад\n");

}

#endregion

#region Меню для бинарного дерева

static public void BinaryTreeMenu()

{

Console.WriteLine("\n1. Сформировать идеально сбалансированное бинарное дерево" +

"\n2. Распечатать дерево" +

"\n3. Вывести высоту дерева" +

"\n4. Преобразовать идеально сбалансированное дерево в дерево поиска" +

"\n5. Распечатать дерево поиска" +

"\n6. Удалить дерево из памяти" +

"\n7. Назад\n");

}

#endregion

#region Меню для хеш-таблицы

static public void HashTableMenu()

{

Console.WriteLine("\n1. Сформировать хеш-таблицу" +

"\n2. Распечатать хеш-таблицу" +

"\n3. Поиск элемента в хеш-таблице" +

"\n4. Поиск и удаление элемента из хеш-таблицы" +

"\n5. Добавление элемента в хеш-таблицу" +

"\n6. Удаление хеш-таблицы из памяти" +

"\n7. Назад");

}

#endregion

#region Меню для обобщенной коллекции

static public void CollMenu()

{

Console.WriteLine("\n1. Сформировать коллекцию" +

"\n2. Распечатать коллекцию" +

"\n3. Добавление случайного объекта" +

"\n4. Удаление элемента" +

"\n5. Поиск объекта" +

"\n6. Клонирование коллекции" +

"\n7. Копирование коллекции" +

"\n8. Вывод всех объектов коллекции через foreach" +

"\n9. Удаление коллекции из памяти" +

"\n10. Назад\n");

}

#endregion

}

}

Unit-тесты

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using laba12new;

using LibraryLaba10;

namespace UnitTestLaba12

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestMethod1()

{

Point p = new Point("", 0, 0);

Assert.AreEqual(p.data, new Challenge("", 0, 0));

}

[TestMethod]

public void TestMethod2()

{

Point p = new Point();

Assert.AreEqual(p.data, new Challenge());

}

[TestMethod]

public void TestMethod3()

{

Point p = new Point("", 0, 0);

Assert.AreEqual(p.ToString(), new Challenge("", 0, 0).Print());

}

[TestMethod]

public void TestMethod4()

{

Point p = Point.MakeList(5);

Assert.AreEqual(p.ToString(), new Point(p.data.Name, p.data.Grade, p.data.ID).ToString());

}

[TestMethod]

public void TestMethod5()

{

Point p = Point.MakeList(5);

p = Point.DeletePoint(p, p.data.Name, p.data.Grade);

p = Point.DeletePoint(p, p.next.data.Name, p.next.data.Grade);

p = Point.DeletePoint(p, "", 0);

Assert.AreEqual(null, null) ;

}

[TestMethod]

public void TestMethod6()

{

Point p = null;

Point.PrintList(p);

p = Point.DeletePoint(p, "", 0);

Assert.AreEqual(p, null);

}

[TestMethod]

public void TestMethod7()

{

LPoint p = new LPoint("", 0);

Assert.AreEqual(p.value.Grade, 0);

}

[TestMethod]

public void TestMethod9()

{

LPoint p = new LPoint("", 0);

LPoint[] tb = HTable.MakeTable(10);

tb = HTable.FindPoint(0, tb, p.value.Name, p.value.Grade);

HTable.Print(tb);

Assert.AreEqual(tb, tb);

}

[TestMethod]

public void TestMethod10()

{

LPoint p = new LPoint("", 0);

LPoint[] tb = HTable.MakeTable(10);

tb = HTable.FindPoint(1, tb, p.value.Name, p.value.Grade);

Assert.AreEqual(HTable.FindPoint(1, tb, p.value.Name, p.value.Grade), tb);

}

[TestMethod]

public void TestMethod11()

{

LPoint[] tb = HTable.MakeTable(10);

tb = HTable.Append(tb);

Assert.AreEqual(tb, tb);

}

[TestMethod]

public void TestMethod12()

{

LPoint[] tb = null;

tb = HTable.Append(tb);

Assert.AreEqual(tb, tb);

}

[TestMethod]

public void TestMethod13()

{

BinaryTree bt = null;

BinaryTree.PrintTree(bt, 1);

bt = new BinaryTree();

bt = BinaryTree.IdealTree(10, bt);

BinaryTree.PrintTree(bt, 1);

Assert.AreEqual(bt.Run(bt), 4);

}

[TestMethod]

public void TestMethod14()

{

BinaryTree bt = null;

bt = BinaryTree.IdealTree(10, bt);

BinaryTree root = null;

root = BinaryTree.First();

BinaryTree.Transform(root, bt);

int h = root.Run(root);

Assert.AreEqual(h, h);

}

[TestMethod]

public void TestMethod15()

{

MyCollection<Challenge> collStack = null;

collStack = new MyCollection<Challenge>();

collStack = new MyCollection<Challenge>("", 0, 0);

collStack = new MyCollection<Challenge>(collStack);

collStack = new MyCollection<Challenge>(10);

MyCollection<Challenge> copy = MyCollection<Challenge>.Copy(collStack);

MyCollection<Challenge> clone = MyCollection<Challenge>.Clone(collStack);

MyCollection<Challenge>.Clear(ref collStack);

Assert.AreEqual(collStack, null);

}

[TestMethod]

public void TestMethod16()

{

MyCollection<Challenge> collStack = null;

MyCollection<Challenge>.PrintList(collStack);

collStack = MyCollection<Challenge>.DeletePoint(collStack);

collStack = MyCollection<Challenge>.Add(collStack);

MyCollection<Challenge> collOper = null;

collStack = new MyCollection<Challenge>(10);

collOper = collStack;

collStack = MyCollection<Challenge>.Add(collStack);

collStack = MyCollection<Challenge>.DeletePoint(collStack);

MyCollection<Challenge>.PrintList(collStack);

Assert.AreEqual(collStack, collOper);

}

[TestMethod]

public void TestMethod17()

{

MyCollection<Challenge> collStack = null;

MyCollection<Challenge>.Search(collStack, "", 0);

collStack = new MyCollection<Challenge>(10);

MyCollection<Challenge>.Search(collStack, "", 0);

MyCollection<Challenge>.Search(collStack, collStack.next.data.Name, collStack.next.data.Grade);

foreach (MyCollection<Challenge> obj in collStack)

{

Console.WriteLine(obj.data.Print());

}

Assert.AreEqual(collStack, collStack);

}

}

}