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

*Факультет экономики, менеджмента и бизнес-информатики*

Шульжик Кирилл Владимирович

**Отчёт по практической работе №12 (Вариант 16)**

студента образовательной программы бакалавриата «Программная инженерия» по направлению подготовки 09.03.04 Программная инженерия

Преподаватель

В.Д Марквирер

Оглавление

[1 Постановка задачи 4](#_Toc105678570)

[1.1 Задание 1 4](#_Toc105678571)

[1.2 Задание 2 4](#_Toc105678572)

[1.3 Задание 3 4](#_Toc105678573)

[1.4 Задание 4 5](#_Toc105678574)

[2 Диаграмма классов 6](#_Toc105678575)

[2.1 Диаграмма классов задания 1 6](#_Toc105678576)

[2.2 Диаграмма классов задания 2 7](#_Toc105678577)

[2.3 Диаграмма классов задания 3 7](#_Toc105678578)

[2.4 Диаграмма классов задания 4 8](#_Toc105678579)

[3 Код программы 9](#_Toc105678580)

[3.1 Код класса Program 9](#_Toc105678581)

[3.2 Код класса Point 9](#_Toc105678582)

[3.3 Код класса WdPoint 9](#_Toc105678583)

[3.4 Код класса Lpoint 9](#_Toc105678584)

[3.5 Код класса HTable 9](#_Toc105678585)

[3.6 Код класса Node<T> 9](#_Toc105678586)

[3.7 Код класса NodeStack<T> 9](#_Toc105678587)

[4 Код Unit-тестов 10](#_Toc105678588)

[5 Анализ покрытия кода тестами 11](#_Toc105678589)

[ПРИЛОЖЕНИЕ А *Код класса Program* 12](#_Toc105678590)

[ПРИЛОЖЕНИЕ Б *Код класса Point* 26](#_Toc105678591)

[ПРИЛОЖЕНИЕ В *Код класса WdPoint* 29](#_Toc105678592)

[ПРИЛОЖЕНИЕ Г *Код класса Lpoint* 33](#_Toc105678593)

[ПРИЛОЖЕНИЕ Д *Код класса HTable* 34](#_Toc105678594)

[ПРИЛОЖЕНИЕ Е *Код класса Node<T>* 37](#_Toc105678595)

[ПРИЛОЖЕНИЕ Ж *Код класса NodeStack<T>* 38](#_Toc105678596)

[ПРИЛОЖЕНИЕ И *Код Unit-тестов* 43](#_Toc105678597)

1 Постановка задачи

1.1 Задание 1

Задание 1 формулируется следующим образом:

1. Сформировать двунаправленный список, в информационное поле записать объекты из иерархии классов лабораторной работы №10.
2. Распечатать полученный список.
3. Выполнить обработку списка в соответствии с заданием (вариант 16 -удалить из списка первый элемент с заданным информационным полем).
4. Распечатать полученный список.
5. Удалить список из памяти.

1.2 Задание 2

Задание 2 формулируется следующим образом:

1. Сформировать идеально сбалансированное бинарное дерево, в информационное поле записать объекты из иерархии классов лабораторной работы №10.
2. Распечатать полученное дерево.
3. Выполнить обработку дерева в соответствии с заданием, вывести полученный результат (вариант 16 - найти минимальный элемент в дереве).
4. Преобразовать идеально сбалансированное дерево в дерево поиска.
5. Распечатать полученное дерево.
6. Удалить дерево из памяти.

1.3 Задание 3

Задание 3 формулируется следующим образом:

1. Создать хеш-таблицу и заполнить ее элементами.
2. Выполнить поиск элемента в хеш-таблице
3. Удалить найденный элемент из хеш-таблицы.
4. Выполнить поиск элемента в хеш-таблице
5. Показать, что будет при добавлении элемента в хеш-таблицу, если в таблице уже находится максимальное число элементов (для метода открытой адресации, для метода цепочек просто показать добавление в таблицу). Вариант 16 – метод цепочек, поиск и удаление поключу.

1.4 Задание 4

Реализовать обобщенную коллекцию, указанную в варианте (стек на базе двунаправленного списка). Для этого:

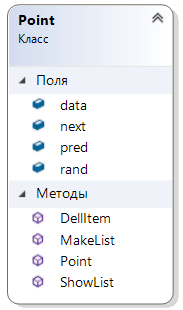
1. Реализовать конструкторы:
   1. public MyCollection() - предназначен для создания пустой коллекции.
   2. public MyCollection (int capacity) - создает пустую коллекцию с начальной емкостью, заданной параметром capacity.
   3. public MyCollection (MyCollection c) - служит для создания коллекции, которая инициализируется элементами и емкостью коллекции, заданной параметром с.
2. 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);
   * метод для поиска элемента по значению;
   * метод для клонирования коллекции;
   * метод для поверхностного копирования;
   * метод для удаления коллекции из памяти.
3. Реализовать интерфейсы IEnumerable и IEnumerator (если это необходимо).
4. 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.

2 Диаграмма классов

2.1 Диаграмма классов задания 1

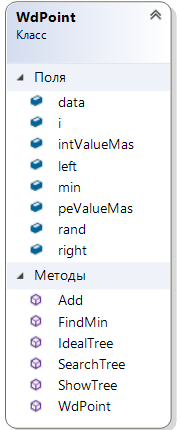
На рисунке 1 представлена диаграмма классов для задания 1



***Рисунок 1 – Диаграмма классов для задания 1***

2.2 Диаграмма классов задания 2

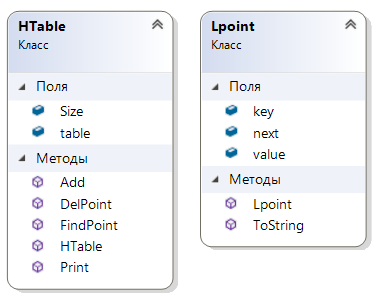
На рисунке 2 представлена диаграмма классов задания 2



***Рисунок 2 – Диаграмма классов для задания 2***

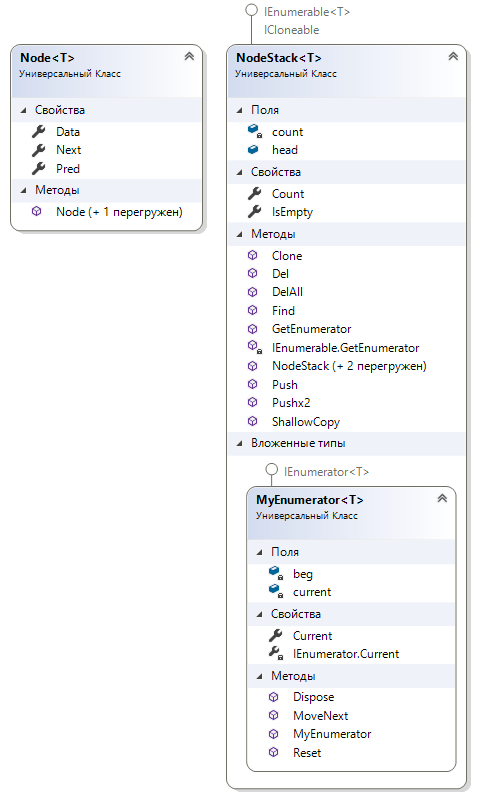
2.3 Диаграмма классов задания 3

На рисунке 3 представлена диаграмма классов для задания 3



***Рисунок 3 – Диаграмма классов для задания 3***

2.4 Диаграмма классов задания 4

На рисунке 4 представлена диаграмма классов для задания 4

***Рисунок 4 – Диаграмма классов для задания 4***

3 Код программы

3.1 Код класса Program

См. [ПРИЛОЖЕНИЕ А](#приложениеА).

3.2 Код класса Point

См. [ПРИЛОЖЕНИЕ Б](#приложениеБ).

3.3 Код класса WdPoint

См. [ПРИЛОЖЕНИЕ В](#приложениеВ).

3.4 Код класса Lpoint

См. [ПРИЛОЖЕНИЕ Г](#приложениеГ).

3.5 Код класса HTable

См. [ПРИЛОЖЕНИЕ Д](#приложениеД).

3.6 Код класса Node<T>

См. [ПРИЛОЖЕНИЕ Е](#приложениеЕ).

3.7 Код класса NodeStack<T>

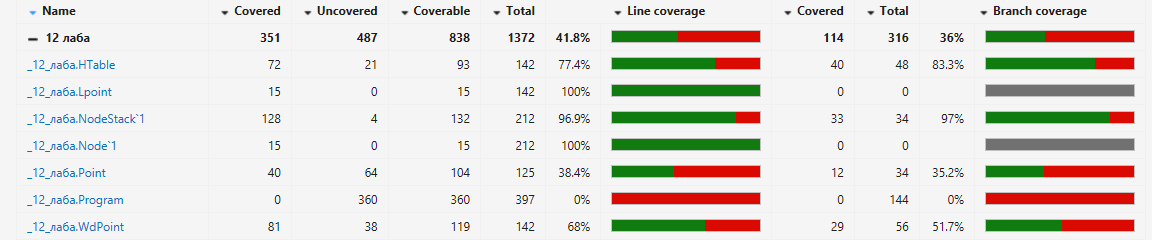
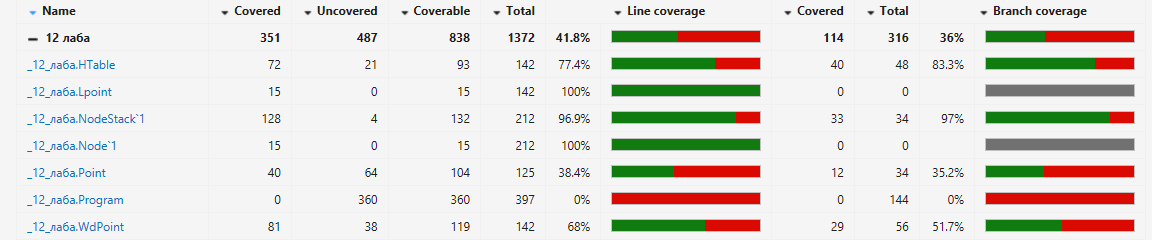
См. [ПРИЛОЖЕНИЕ Ж](#приложениеЖ).

4 Код Unit-тестов

См. [ПРИЛОЖЕНИЕ И](#приложениеИ).

**5 Анализ покрытия кода тестами**

Анализ покрытия кода тестами представлен на рисунке 5



***Рисунок 5 – Анализ покрытия кода тестами***

ПРИЛОЖЕНИЕ А

**Код класса Program**

using System;

using EditionLib;

using System.Collections.Generic;

namespace \_12\_лаба

{

public class Program

{

public static void CheckInt(out int x, string msg = "", int left = Int32.MinValue, int right = Int32.MaxValue) //Функция для получения целочисленного значения

{

x = 0;

if (right < left && right != Int32.MaxValue && left != Int32.MinValue)

Console.WriteLine("Неверно заданы границы диапазона");

else

{

if (msg != "")

Console.WriteLine(msg);

string bug = Console.ReadLine();

while ((!int.TryParse(bug, out x) || int.Parse(bug) < left || int.Parse(bug) > right))

{

Console.WriteLine();

if (left != Int32.MinValue && right != Int32.MaxValue && left == right)

{

Console.WriteLine($"Введите число {left}");

bug = Console.ReadLine();

}

else if (left != Int32.MinValue && right != Int32.MaxValue && right - left == 1)

{

Console.WriteLine($"Введите либо {left}, либо {right}");

bug = Console.ReadLine();

}

else if (left != Int32.MinValue && right != Int32.MaxValue)

{

Console.WriteLine($"Введите целое число, большее или равное {left} и меньшее или равное {right}");

bug = Console.ReadLine();

}

else if (left != Int32.MinValue)

{

Console.WriteLine($"Введите целое число, большее или равное {left}");

bug = Console.ReadLine();

}

else if (right != Int32.MaxValue)

{

Console.WriteLine($"Введите целое число, меньшее или равное {right}");

bug = Console.ReadLine();

}

else

{

Console.WriteLine($"Введите целое число");

bug = Console.ReadLine();

}

}

x = int.Parse(bug);

}

}

public static Print\_edition FindItem() //Нахождение введённого элемента в хэш-таблице

{

Console.WriteLine("\nВведите, какого типа вы хотите найти объект в хэш таблице");

Console.WriteLine(@"1. Book

2. Journal

3. Print edition

4. Textbook");

Print\_edition p1 = new();

string inputItem = Console.ReadLine();

while (inputItem != "1" && inputItem != "2" && inputItem != "3" &&

inputItem != "4")

{

Console.WriteLine("Введите существующий тип объекта");

inputItem = Console.ReadLine();

}

switch (inputItem)

{

case "1":

Console.WriteLine("\nВведите название книги");

string Name = Console.ReadLine();

CheckInt(out int Pages, "\nВведите кол-во страниц книги", 0);

Console.WriteLine("\nВведите жанр книги");

string Genre = Console.ReadLine();

Book bFind = new Book(Name, Pages, Genre);

return bFind;

case "2":

Console.WriteLine("\nВведите название журнала");

Name = Console.ReadLine();

CheckInt(out Pages, "\nВведите кол-во страниц журнала", 0);

Console.WriteLine("\nВведите период журнала");

string Period = Console.ReadLine();

Journal j = new Journal(Name, Pages, Period);

return j;

case "3":

Console.WriteLine("\nВведите название печатного издания");

Name = Console.ReadLine();

CheckInt(out Pages, "\nВведите кол-во страниц печатного издания");

Print\_edition p = new Print\_edition(Name, Pages);

return p;

case "4":

Console.WriteLine("\nВведите название учебника");

Name = Console.ReadLine();

CheckInt(out Pages, "\nВведите кол-во страниц учебника");

Console.WriteLine("\nВведите предметную область учебника");

string SubjectArea = Console.ReadLine();

Textbook t = new Textbook(Name, Pages, SubjectArea);

return t;

}

return p1;

}

static void Main(string[] args)

{

Random rand = new();

Console.WriteLine(@"Выберите, с какого задания следует начать

1. Двунаправленный список

2. Бинарное дерево

3. Хэш-таблица

4. Стэк на базе двунаправленного списка");

string answer = Console.ReadLine();

while (answer != "1" && answer != "2" && answer != "3" && answer != "4")

{

Console.WriteLine("Введите число от 1 до 4!\n");

answer = Console.ReadLine();

}

if (answer == "1")

{

Console.WriteLine();

//Двусвязный список

Point beg = Point.MakeList(10);

Point.ShowList(beg);

if (beg != null)

{

Console.Write("Введите название печати, которое хотите удалить из двусвязного списка: ");

string nameprint = Console.ReadLine();

Point.DellItem(nameprint, ref beg);

}

else

{

Console.WriteLine("Список пуст, удаление невозможно\n");

}

Point.ShowList(beg);

if (beg != null) //Проверка для удаления списка

{

beg = null;

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Список удалён из памяти\n");

Console.ResetColor();

}

Console.WriteLine("Нажмите любую клавишу, чтобы перейти ко 2 заданию");

Console.ReadKey();

}

if (answer == "1" || answer == "2")

{

Console.WriteLine();

//Создание идеально сбалансированного бинарного дерева

Console.WriteLine("\nИдеально сбалансированное дерево\n\n");

WdPoint p = new();

p = WdPoint.IdealTree(6, p);

if (p == null)

Console.WriteLine("Пустое дерево\n");

WdPoint.ShowTree(p);

//Вывод минимального элемента дерева

if (p == null)

Console.WriteLine("Минимального элемента нет, дерево пустое\n");

else

{

Console.WriteLine("Чтобы вывести минимальный элемент, нажмите любую клавишу\n");

Console.ReadKey();

Console.WriteLine("Минимальный элемент дерева = " + WdPoint.FindMin(p) + "\n"); //Вывод минимального элемента дерева

WdPoint.min = 2001;

}

//Создание дерева поиска по идеально сбалансированному

Console.WriteLine("Чтобы преобразовать идеально сбалансированное дерево в дерево поиска, нажмите любую клавишу\n");

Console.ReadKey();

if (p == null)

{

Console.WriteLine("Дерево пустое\n");

}

else

{

WdPoint STree = new();

Console.WriteLine("\nДерево поиска\n\n");

STree = WdPoint.SearchTree(STree, WdPoint.i);

WdPoint.i = 0;

WdPoint.ShowTree(STree);

}

//Удаление деревьев

if (p != null)

{

p = null;

WdPoint.peValueMas.Clear();

WdPoint.intValueMas.Clear();

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Дерево удалено из памяти\n");

Console.ResetColor();

}

Console.WriteLine("Чтобы перейти к 3 заданию, нажмите любую клавишу\n");

Console.ReadKey();

}

Print\_edition data = new();

if (answer == "3" || answer == "2" || answer == "1")

{

Console.WriteLine();

//Хэш - таблица

Console.WriteLine("Хэш-таблица:\n");

HTable Htable = new(10);

for (int i = 0; i < 10; i++)

{

int number = rand.Next(1, 5);

if (number == 1)

{

data = new Print\_edition();

data.Init();

}

if (number == 2)

{

data = new Book();

data.Init();

}

if (number == 3)

{

data = new Textbook();

data.Init();

}

if (number == 4)

{

data = new Journal();

data.Init();

}

Htable.Add(data);

}

Htable.Print();

Print\_edition p1 = new();

if (Htable.table != null)

{

p1 = FindItem();

if (Htable.FindPoint(p1) == true)

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("\nЭлемент найден\n");

Console.ResetColor();

if (Htable.DelPoint(p1))

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Элемент удалён\n");

Console.ResetColor();

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Удаление не произведено\n");

Console.ResetColor();

}

Console.WriteLine("Чтобы распечатать хэш-таблицу ещё раз, нажмите любую клавишу\n");

Console.ReadKey();

Htable.Print();

if (Htable.FindPoint(p1))

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Элемент найден\n");

Console.ResetColor();

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("\nЭлемент не найден\n");

Console.ResetColor();

}

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("\nЭлемент не найден\n");

if (Htable.DelPoint(p1))

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Элемент удалён\n");

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Удаление не произведено\n");

}

}

}

Console.ResetColor();

Console.WriteLine("Чтобы перейти к 4 заданию, нажмите любую клавишу\n");

Console.ReadKey();

}

if (answer == "1" || answer == "2" || answer == "3" || answer == "4")

{

//Стэк

Console.WriteLine("\nСтек:\n");

NodeStack<Print\_edition> stack = new();

for (int i = 0; i < 10; i++)

{

int number = rand.Next(1, 5);

if (number == 1)

{

data = new Print\_edition();

data.Init();

}

if (number == 2)

{

data = new Book();

data.Init();

}

if (number == 3)

{

data = new Textbook();

data.Init();

}

if (number == 4)

{

data = new Journal();

data.Init();

}

stack.Push(data);

}

foreach (Print\_edition x in stack)

Console.WriteLine(x);

Print\_edition t = FindItem();

if (stack.Find(t) == true)

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("\nЭлемент найден\n");

Console.ResetColor();

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("\nЭлемент не найден\n");

Console.ResetColor();

}

if (stack.head.Data != null && stack.head.Pred != null)

{

stack.Del();

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Удаление 2 верхних элементов стека произведено\n");

Console.ResetColor();

Console.WriteLine("Чтобы распечатать стек, нажмите любую клавишу\n");

Console.ReadKey();

foreach (Print\_edition x in stack)

Console.WriteLine(x);

}

else if (stack.head.Pred == null)

{

stack.Del();

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("Удаление верхушки стека произведено\n");

Console.ResetColor();

Console.WriteLine("Чтобы распечатать стек, нажмите любую клавишу\n");

Console.ReadKey();

foreach (Print\_edition x in stack)

Console.WriteLine(x);

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Удаление верхушки не произведено\n");

Console.ResetColor();

}

NodeStack<Print\_edition> ob = (NodeStack<Print\_edition>)stack.ShallowCopy();

NodeStack<Print\_edition> f = (NodeStack<Print\_edition>)stack.Clone();

stack.head.Data = new Print\_edition(4, "4");

Console.WriteLine($"\n\nИзменили верхушку стека на:\n{stack.head.Data.ToString()}\n");

Console.WriteLine("Чтобы посмотреть поверхностую копию стека, нажмите любую клавишу\n");

Console.ReadKey();

Console.WriteLine("\nПоверхностная копия стека:\n\n");

foreach (Print\_edition x in ob)

Console.WriteLine(x);

Console.WriteLine("\nЧтобы посмотреть клон стека, нажмите любую клавишу\n");

Console.ReadKey();

Console.WriteLine("\nКлон стека:\n\n");

bool isPrint = false;

foreach (Print\_edition x in f)

{

if (x != null)

{

Console.WriteLine(x);

isPrint = true;

}

}

if (isPrint == false)

Console.WriteLine("Пустой стек");

if (stack.head.Data != null)

{

stack.DelAll();

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("\nУдаление стэка из памяти произведено");

Console.ResetColor();

}

}

}

}

}

ПРИЛОЖЕНИЕ Б

**Код класса Point**

using System;

using EditionLib;

namespace \_12\_лаба

{

public class Point

{

public static Random rand = new();

public Print\_edition data;

public Point next, pred;

public Point()

{

int number = rand.Next(1, 5);

if (number == 1)

{

data = new Print\_edition();

data.Init();

}

if (number == 2)

{

data = new Book();

data.Init();

}

if (number == 3)

{

data = new Textbook();

data.Init();

}

if (number == 4)

{

data = new Journal();

data.Init();

}

next = null;

pred = null;

}

public static Point MakeList(int size)

{

if (size <= 0)

return null;

Point beg = new Point();

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

{

Point p = new Point();

p.next = beg;

beg.pred = p;

beg = p;

}

return beg;

}

public static void ShowList(Point beg)

{

if (beg == null)

{

Console.WriteLine("Пусто\n");

return;

}

Point p = beg;

while (p != null)

{

Console.WriteLine(p.data);

p = p.next;

}

}

public static void DellItem(string name, ref Point beg)

{

bool isDel = false;

Point p = beg;

if (p == null)

{

Console.WriteLine("Ошибка. Пустой список.\n");

return;

}

if (p.data.Name == name)

{

if (p.next == null && p.pred == null) //1 элемент в списке

{

beg = beg.next;

isDel = true;

}

else //Несколько элементов в списке

{

beg = beg.next;

beg.pred = null;

isDel = true;

}

}

else

{

while (p.next != null)

{

p = p.next;

if (p.data.Name == name)

{

if (p.next != null) //Не последний элемент в двусвязном списке

{

p.pred.next = p.next;

p.next.pred = p.pred;

p.data = null;

}

else

p.pred.next = null; //Последний элемент в двусвязном списке

isDel = true;

while (p.pred != null)

p = p.pred;

beg = p;

break;

}

}

}

if (isDel)

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("\nУдаление произведено\n");

Console.ForegroundColor = ConsoleColor.White;

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine($"\nНе удалось найти элемент с именем {name} в списке\n\n");

Console.ForegroundColor = ConsoleColor.White;

}

}

}

}

ПРИЛОЖЕНИЕ В

**Код класса WdPoint**

using System;

using EditionLib;

using System.Collections.Generic;

namespace \_12\_лаба

{

public class WdPoint

{

public static int i = 0;

public static Random rand = new();

public Print\_edition data;

public static int min = 2001;

public static List<int> intValueMas = new();

public static List<Print\_edition> peValueMas = new();

public WdPoint left, right;

public WdPoint()

{

int number = rand.Next(1, 5);

if (number == 1)

{

data = new Print\_edition();

data.Init();

}

if (number == 2)

{

data = new Book();

data.Init();

}

if (number == 3)

{

data = new Textbook();

data.Init();

}

if (number == 4)

{

data = new Journal();

data.Init();

}

left = null;

right = null;

}

public static WdPoint IdealTree(int size, WdPoint p)

{

if (size > 2000 || size <= 0)

{

if (size > 2000)

Console.WriteLine("Нельзя создать дерево более, чем c 2000 элементами\n"); //Каждый элемент дерева должен быть актуальным, у меня в таблице кол-во страниц формируется в пределах от 0 до 2000

else if (size < 0)

Console.WriteLine("Нельзя создать дерево с нулевым или отрицательным числом элементов\n");

return null;

}

WdPoint r = new WdPoint();

while (intValueMas.Contains(r.data.Pages\_num))

r = new WdPoint();

intValueMas.Add(r.data.Pages\_num);

peValueMas.Add(r.data);

int nl, nr;

nl = size / 2; nr = size - nl - 1;

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

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

return r;

}

public static void ShowTree(WdPoint p, int l = 3)

{

if (p != null)

{

ShowTree(p.left, l + 12);

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

Console.Write(" ");

Console.WriteLine("Название: " + p.data.Name);

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

Console.Write(" ");

Console.WriteLine("Кол-во страниц: " + p.data.Pages\_num);

if (p.data is Journal || p.data is Textbook || p.data is Book)

{

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

Console.Write(" ");

if (p.data is Journal j)

Console.WriteLine("Период: " + j.Period);

if (p.data is Textbook t)

Console.WriteLine("Предметная область: " + t.subject\_area);

if (p.data is Book b)

Console.WriteLine("Жанр: " + b.Genre);

Console.WriteLine();

}

else

Console.WriteLine();

ShowTree(p.right, l + 12);

}

}

public static int FindMin(WdPoint p)

{

if (p != null)

{

//Обход дерева слева направо

FindMin(p.left); //Переход к левой ветви дерева

if (p.data.Pages\_num < min)

min = p.data.Pages\_num;

FindMin(p.right); //Переход к правой ветви дерева

}

return min;

}

public static void Add(WdPoint root)

{

WdPoint p = root;

WdPoint r = null;

while (p != null)

{

r = p;

if (peValueMas[i].Pages\_num < p.data.Pages\_num)

p = p.left;

else

p = p.right;

}

WdPoint NewPoint = new WdPoint();

NewPoint.data = peValueMas[i];

if (NewPoint.data.Pages\_num < r.data.Pages\_num)

r.left = NewPoint;

else

r.right = NewPoint;

i += 1;

}

public static WdPoint SearchTree(WdPoint r, int size)

{

if (peValueMas.Count == 0)

{

Console.WriteLine("Невозможно преобразовать заданный объект в дерево поиска\n");

return null;

}

if (i == 0)

{

r.data = peValueMas[i];

i += 1;

}

if (i == intValueMas.Count)

return r;

Add(r);

SearchTree(r, i);

return r;

}

}

}

ПРИЛОЖЕНИЕ Г

**Код класса Lpoint**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using EditionLib;

namespace \_12\_лаба

{

public class Lpoint

{

public int key;

public Print\_edition value;

public Lpoint next;

public Lpoint(Print\_edition p)

{

value = p;

key = p.GetHashCode();

next = null;

}

public Lpoint()

{

value = null;

key = 0;

next = null;

}

public override string ToString()

{

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

}

}

}

ПРИЛОЖЕНИЕ Д

**Код класса HTable**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using EditionLib;

namespace \_12\_лаба

{

public class HTable

{

public Lpoint[] table;

public int Size;

public HTable(int size = 10)

{

if (size <= 0)

{

Size = 0;

table = null;

}

else

{

Size = size;

table = new Lpoint[Size];

}

}

public bool Add(Print\_edition p)

{

if (Size == 0)

return false;

if (p == null || p.Name == null) return false; //Нельзя добавить, если пустой элемент

Lpoint point = new Lpoint(p);

int index = Math.Abs(p.GetHashCode()) % Size;

if (table[index] == null) //Добавление, если место свободно

table[index] = point;

else //Иначе движение глубь

{

Lpoint cur = table[index];

if (string.Compare(cur.ToString(), point.ToString()) == 0)

return false;

while (cur.next != null)

{

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

cur = cur.next;

}

cur.next = point;

}

return true;

}

public void Print()

{

if (table == null)

{

Console.WriteLine("Таблица пустая!\n");

return;

}

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

{

if (table[i] == null)

Console.WriteLine(i + ":");

else

{

Console.WriteLine(i + ": ");

Lpoint p = table[i];

while (p != null)

{

Console.WriteLine(p.ToString());

p = p.next;

}

Console.WriteLine();

}

}

}

public bool FindPoint(Print\_edition p)

{

if (Size == 0)

return false;

Lpoint lp = new();

if (p != null)

{

lp = new Lpoint(p);

int code = Math.Abs(p.GetHashCode()) % Size;

if (table[code] != null && String.Compare(table[code].value.ToString(), p.ToString()) == 0)

return true; //Возврат true, если 1 элемент по индексу - p

lp = table[code];

while (lp != null) //Иначе движение вглубь

{

if (string.Compare(lp.value.ToString(), p.ToString()) == 0) return true;

lp = lp.next;

}

return false;

}

return false;

}

public bool DelPoint(Print\_edition p)

{

Lpoint lp = new Lpoint(p);

if (Size == 0)

return false;

int code = Math.Abs(p.GetHashCode()) % Size;

lp = table[code];

if (table[code] == null) return false;

if (String.Compare(table[code].value.ToString(), p.ToString()) == 0)

{

table[code] = table[code].next;

return true;

}

while (lp.next != null && (string.Compare(lp.next.value.ToString(), p.ToString()) != 0))

lp = lp.next;

if (lp.next != null)

{

lp.next = lp.next.next;

return true; ;

}

return false;

}

}

}

ПРИЛОЖЕНИЕ Е

**Код класса Node<T>**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using EditionLib;

namespace \_12\_лаба

{

public class Node<T>

{

public T Data { get; set; }

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

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

public Node(T data)

{

Data = data;

Next = null;

Pred = null;

}

public Node()

{

Data = default;

Next = null;

Pred = null;

}

}

}

ПРИЛОЖЕНИЕ Ж

**Код класса NodeStack<T>**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using EditionLib;

namespace \_12\_лаба

{

public class NodeStack<T>: IEnumerable<T>, ICloneable

{

public Node<T> head;

int count;

class MyEnumerator<T>: IEnumerator<T> //Реализация интерфейса IEnumarable

{

Node<T> beg;

Node<T> current;

public MyEnumerator(NodeStack<T> stack)

{

beg = stack.head;

current = stack.head;

}

object IEnumerator.Current

{

get { return current; }

}

public T Current

{

get { return current.Data; }

}

public bool MoveNext()

{

if (current == null || current.Data == null && current.Pred == null)

return false;

if (current.Pred == null && current.Next != null)

{

Reset();

return false;

}

else

{

if (current.Next == null)

{

current.Next = new Node<T>();

current.Next.Pred = current;

current = current.Next;

}

current = current.Pred;

return true;

}

}

public void Reset()

{

current = this.beg;

current.Next = null;

}

public void Dispose()

{

}

}

public bool IsEmpty

{

get { return count == 0; }

}

public int Count

{

get { return count; }

}

public NodeStack() //Создание пустого стека

{

head = new Node<T>();

}

public NodeStack(int capacity) //Создание пустого стека заданной ёмкости

{

head = new Node<T>();

for (int i = 0; i < capacity-1; i++)

{

Node<T> temp = new();

temp.Pred = head;

head.Next = temp;

head = temp;

}

}

public NodeStack(NodeStack<T> p) //Создание стека по имеющемуся стеку

{

head = new Node<T>();

List<T> tempMas = new();

foreach (var x in p)

tempMas.Add(x);

tempMas.Reverse(); //Поскольку стек перебирается задом наперёд (с последнего добавленного до первого добавленного), то лист нужно перевернуть

foreach (var x in tempMas)

{

Node<T> temp = new Node<T>(x);

temp.Pred = head;

head.Next = temp;

head = temp;

count += 1;

}

}

public void Push(T item)

{

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

if (head.Data == null)

{

head = new Node<T>(item);

count += 1;

return;

}

node.Pred = head;

head.Next = node;

head = node;

count++;

}

public void Pushx2(T item, T item1)

{

Push(item);

Push(item1);

}

public bool Del()

{

if (IsEmpty)

{

Console.WriteLine("Стэк пуст");

return false;

}

if (head.Pred == null) //Если стек состоит из 1 элемента

{

count -= 1;

head.Data = default;

}

else //Если стек состоит более, чем из 1 элемента

{

if (head.Pred.Pred == null) //Если стек состоит из 2 элементов

{

count -= 2;

head = head.Pred;

head.Next = null;

head.Data = default;

return true;

}

head = head.Pred.Pred; //Если стек состоит более, чем из 2 элементов

head.Next = null;

count -= 2;

}

return true;

}

public bool Find(T item)

{

if (head.Data == null)

return false;

Node<T> temp = head;

while (temp != null && String.Compare(temp.Data.ToString(), item.ToString()) != 0)

{

temp = temp.Pred;

}

if (temp != null) //Если проверка на эквивалентность показала, что элементы равны, то выход из цикла while (temp != null..) произойдёт раньше, чем temp станет null

return true;

return false; //Если temp стал null, то ни один из элементов стека не равен item

}

public object Clone()

{

return new NodeStack<T>(this);

}

//IEnumerator<T> IEnumerable<T>.GetEnumerator() //Реализация с помощью yield

//{

// Node<T> current = head;

// while (current != null)

// {

// yield return current.Data;

// current = current.Pred;

// }

//}

public object ShallowCopy()

{

return this.MemberwiseClone();

}

public void DelAll()

{

head.Pred = null;

head.Data = default;

count = 0;

}

public IEnumerator<T> GetEnumerator() //Реализация обобщенного энумератора

{

return new MyEnumerator<T>(this);

}

IEnumerator IEnumerable.GetEnumerator() //Реализация необобщенного энумератора, поскольку обобщённый произошёл от необобщённого

{

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

}

}

}

ПРИЛОЖЕНИЕ И

**Код Unit-тестов**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using EditionLib;

using \_12\_лаба;

using System;

namespace TestProject1

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void LinkedListEmptyCreateTest()

{

Point r = Point.MakeList(0);

Assert.AreEqual(r, null);

}

[TestMethod]

public void LinkedListNotEmptyCreateTest()

{

Point r = Point.MakeList(10);

Assert.AreNotEqual(r.next, null);

}

[TestMethod]

public void WoodCreateTest()

{

WdPoint r = new();

Assert.AreNotEqual(r, null);

}

[TestMethod]

public void WoodIdealTreeTest()

{

WdPoint z = new();

z = WdPoint.IdealTree(-4, z);

Assert.AreEqual(z, null);

z = WdPoint.IdealTree(43, z);

WdPoint.peValueMas.Clear();

WdPoint.intValueMas.Clear();

Assert.AreNotEqual(z, null);

}

[TestMethod]

public void IdealInSearchTest()

{

WdPoint p = new();

p = WdPoint.IdealTree(5, p);

p = WdPoint.SearchTree(p, WdPoint.i);

WdPoint.peValueMas.Clear();

WdPoint.intValueMas.Clear();

WdPoint.i = 0;

Assert.AreNotEqual(p, null);

}

[TestMethod]

public void NullIdealInSearchTest()

{

WdPoint r = new();

r = WdPoint.SearchTree(r, 534);

Assert.AreEqual(null, r);

}

[TestMethod]

public void HTableNegativeSizeCreateTest()

{

HTable p = new(-34);

Assert.AreEqual(p.Size, 0);

}

[TestMethod]

public void HTableNotNegativeSizeCreateTest()

{

HTable p = new(10);

Assert.AreNotEqual(p, null);

Assert.AreEqual(p.Size, 10);

}

[TestMethod]

public void NullHTableMethodTests()

{

HTable p = new(-4);

Assert.AreEqual(p.Add(new Print\_edition(4, "4")), false);

Assert.AreEqual(p.FindPoint(new Print\_edition(4, "4")), false);

Assert.AreEqual(p.DelPoint(new Print\_edition(4, "4")), false);

}

[TestMethod]

public void NotNullHTableMethodTests()

{

HTable p = new(10);

Assert.AreEqual(p.Add(new Print\_edition(4, "$")), true);

Assert.AreEqual(p.FindPoint(new Print\_edition(4, "4")), false);

Assert.AreEqual(p.DelPoint(new Print\_edition(4, "4")), false);

}

[TestMethod]

public void HTableNotFindTest()

{

Print\_edition p = new(4, "4");

HTable f = new(10);

Assert.AreEqual(f.FindPoint(p), false);

}

[TestMethod]

public void HTableFindTest()

{

Print\_edition p = new(4, "3");

Lpoint l = new Lpoint(p);

HTable f = new(1);

f.table[0] = l;

Assert.AreEqual(true, f.FindPoint(p));

}

[TestMethod]

public void HTableFindInTest()

{

Print\_edition p = new(4, "3");

Print\_edition p1 = new(4, "4");

Lpoint l = new Lpoint(p);

Lpoint l1 = new Lpoint(p1);

HTable f = new(1);

f.table[0] = l;

f.table[0].next = l1;

Assert.AreEqual(true, f.FindPoint(p1));

}

[TestMethod]

public void HTableFindNullTest()

{

Print\_edition p = new();

p = null;

HTable f = new();

Assert.AreEqual(false, f.FindPoint(p));

}

[TestMethod]

public void HTableNullAddTest()

{

Print\_edition p = new();

p = null;

HTable f = new();

Assert.AreEqual(false, f.Add(p));

}

[TestMethod]

public void HTableAddInTest()

{

Print\_edition p = new(4, "4");

Print\_edition p1 = new(3, "4");

Print\_edition p2 = new(4, "4");

Print\_edition p3 = new(3, "4");

Print\_edition p4 = new(33, "44");

Lpoint l = new(p);

HTable f = new(1);

f.table[0] = l;

Assert.AreEqual(true, f.Add(p1));

Assert.AreEqual(true, f.Add(p4));

Assert.AreEqual(false, f.Add(p2));

Assert.AreEqual(false, f.Add(p3));

Assert.AreEqual(false, f.Add(p4));

}

[TestMethod]

public void HTableDelInTest()

{

Print\_edition p = new();

Print\_edition p1 = new(4, "4");

Print\_edition p2 = new(3, "5");

HTable f = new(1);

Assert.AreEqual(false, f.DelPoint(p));

f.Add(p);

f.Add(p1);

f.Add(p2);

Assert.AreEqual(false, f.DelPoint(new Print\_edition(4, "34")));

Assert.AreEqual(true, f.DelPoint(p2));

Assert.AreEqual(true, f.DelPoint(p1));

Assert.AreEqual(true, f.DelPoint(p));

}

[TestMethod]

public void LpointToStringTest()

{

Lpoint p = new(new Print\_edition(4, "4"));

p.key = 1;

Assert.AreEqual(p.ToString(), "1:\nНазвание печатного издания: 4\nСтраниц: 4\n");

}

[TestMethod]

public void StackTestsWithEmptyStack()

{

NodeStack<Print\_edition> f = new();

Assert.AreEqual(f.head.Data, null);

Assert.AreEqual(f.Count, 0);

Assert.AreEqual(f.IsEmpty, true);

Assert.AreEqual(f.Del(), false);

Assert.AreEqual(f.Find(new Print\_edition(4, "4")), false);

}

[TestMethod]

public void NullNodeStackWithNotNullCapacityAbilityTest()

{

NodeStack<Print\_edition> f = new(10);

}

[TestMethod]

public void NewStackByAnother()

{

NodeStack<Print\_edition> f = new();

for (int i = 0; i < 5; i++)

{

Print\_edition p = new();

p.Init();

f.Push(p);

}

NodeStack<Print\_edition> g = new(f);

Assert.AreEqual(f.head.Data, g.head.Data);

f.head.Data = new Print\_edition(4, "3434");

Assert.AreNotEqual(f.head.Data, g.head.Data);

}

[TestMethod]

public void StackPushTest()

{

NodeStack<Print\_edition> f = new();

f.Push(new Print\_edition(4, "4"));

Assert.AreEqual(f.Count, 1);

f.Pushx2(new Print\_edition(4, "4"), new Print\_edition(4, "4"));

Assert.AreEqual(f.Count, 3);

}

[TestMethod]

public void StackDelTest()

{

NodeStack<Print\_edition> f = new();

f.Push(new Print\_edition(4, "4"));

Assert.AreEqual(true, f.Del());

Assert.AreEqual(0, f.Count);

f.Pushx2(new Print\_edition(4, "4"), new Print\_edition(3, "34"));

Assert.AreEqual(true, f.Del());

Assert.AreEqual(0, f.Count);

f.Pushx2(new Print\_edition(4, "4"), new Print\_edition(3, "34"));

f.Push(new Print\_edition(4, "4"));

Assert.AreEqual(true, f.Del());

Assert.AreEqual(1, f.Count);

}

[TestMethod]

public void StackFindTest()

{

NodeStack<Print\_edition> f = new();

f.Pushx2(new Print\_edition(4, "4"), new Print\_edition(3, "3"));

Assert.AreEqual(true, f.Find(new Print\_edition(4, "4")));

Assert.AreEqual(false, f.Find(new Print\_edition(3, "3434")));

}

[TestMethod]

public void ShallowCopyTest()

{

NodeStack<Print\_edition> f = new();

f.Pushx2(new Print\_edition(4, "4"), new Print\_edition(3, "3"));

NodeStack<Print\_edition> s = (NodeStack<Print\_edition>)f.ShallowCopy();

f.head.Data = new Print\_edition(4, "33434");

Assert.AreEqual(f.head.Data, s.head.Data);

}

[TestMethod]

public void CloneTest()

{

NodeStack<Print\_edition> f = new();

f.Pushx2(new Print\_edition(4, "4"), new Print\_edition(3, "3"));

NodeStack<Print\_edition> s = (NodeStack<Print\_edition>)f.Clone();

f.head.Data = new Print\_edition(4, "33434");

Assert.AreNotEqual(f.head.Data, s.head.Data);

}

[TestMethod]

public void ClearTest()

{

NodeStack<Print\_edition> f = new();

f.Pushx2(new Print\_edition(4, "4"), new Print\_edition(3, "3"));

f.DelAll();

Assert.AreEqual(null, f.head.Pred);

Assert.AreEqual(null, f.head.Data);

Assert.AreEqual(0, f.Count);

}

[TestMethod]

public void ForeachTest()

{

NodeStack<Print\_edition> f = new();

f.Pushx2(new Print\_edition(4, "34"), new Print\_edition(3, "343"));

int s = 0;

foreach (Print\_edition x in f)

s += 1;

Assert.AreEqual(2, s);

f.Del();

s = 0;

foreach (Print\_edition x in f)

s += 1;

Assert.AreEqual(0, s);

}

}

}