МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет   
имени академика С.П. Королева»

(Самарский университет)

Институт информатики, математики и электроники

Факультет информатики

Кафедра программных систем

**ОТЧЕТ ПО ПРАКТИКЕ**

Вид практики учебная

Тип практики Практика по получению первичных профессиональных умений и навыков,

в том числе первичных умений и навыков научно-исследовательской деятельности

Сроки прохождения практики: с 01.07.2019 г. по 18.07.2019 г.

(в соответствии с календарным учебным графиком)

по направлению подготовки 02.03.02

«Фундаментальная информатика и информационные технологии»

(уровень бакалавриата)

направленность (профиль) «Информационные технологии»

Студент группы № 6213-020302D \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Д. Павленко

Руководитель практики

от университета, доцент кафедры   
программных систем, к.т.н. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.А. Попова-Коварцева

Дата сдачи 18.07.2019 г.

Дата защиты 18.07.2019 г.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Самара 2019

**Содержание**

[Индивидуальное задание на практику 3](#_Toc13737806)

[Рабочий график (план) проведения выполнения индивидуального задания на практику 6](#_Toc13737807)

[Введение 7](#_Toc13737808)

[1 Работа с текстовыми файлами 8](#_Toc13737809)

[1.1 Текстовые файлы с числовой информацией 8](#_Toc13737810)

[1.2 Дополнительные задания на обработку текстовых файлов 9](#_Toc13737811)

[2 Работа с составными типами данных в процедурах и функциях 10](#_Toc13737812)

[2.1 Строки 10](#_Toc13737813)  
2.2 Дополнительные задания на обработку строк 10

[3 Работа с указателями и динамическими структурами данных 11](#_Toc13737814)

[3.1 Двусвязный список 11](#_Toc13737815)

[4 Работа с бинарными деревьями 12](#_Toc13737816)

[4.1 Формирование бинарного дерева 13](#_Toc13737817)

[4.2 Преобразование бинарного дерева 13](#_Toc13737818)

[Список использованных источников 16](#_Toc13737819)

[Приложение А Листинг программ из блока «Текстовые файлы» 17](#_Toc13737820)

[Приложение Б Листинг программ из блока «Функции с параметрами» 20](#_Toc13737821)

[Приложение В Листинг программ из блока «Динамические структуры данных» 22](#_Toc13737822)

[Приложение Г Листинг программ из блока «Бинарные деревья» 26](#_Toc13737823)

[ОТЗЫВ о прохождении практики 35](#_Toc13737824)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет   
имени академика С.П. Королева»

(Самарский университет)

Институт информатики, математики и электроники

Факультет информатики

Кафедра программных систем

Индивидуальное задание на практику

Студенту Павленко Вячеславу Дмитриевичу группы 6213-020302D

Тема индивидуального задания: Создание программ, позволяющих производить обработку текстовых файлов, использовать составные типы данных в процедурах и функциях, работать с динамическими структурами данных.

Направление на практику оформлено приказом по университету от 24.06.2019 г. № 68-ПР

в Самарский университет на кафедру программных систем ,

|  |  |  |
| --- | --- | --- |
| Планируемые результаты освоения образовательной программы (компетенции) | Планируемые результаты практики | Содержание задания |
| ПК-2 способностью понимать, совершенствовать и применять современный математический аппарат, фундаментальные концепции и системные методологии, международные и профессиональные стандарты в области информационных технологий | Знать: современный математический аппарат и профессиональные стандарты в области информационных технологий. Уметь: использовать современный математический аппарат и профессиональные стандарты в области информационных технологий. Владеть: навыками практического применения современного математического аппарата и профессиональных стандартов в области информационных технологий. | Изучить основные принципы и процедуры, применяемые при обработке текстовых файлов, использовании сложных типов данных в процедурах и функциях, написании рекурсивных алгоритмов, при работе с динамическими структурами данных. |
| ПК-3 способностью использовать современные инструментальные и вычислительные средства | Знать: современные инструментальные и вычислительные средства. Уметь: использовать современные инструментальные и вычислительные средства. Владеть: навыками использования современных инструментальных и вычислительных средств. | Используя язык программирования C#, составить алгоритмы решения индивидуальных заданий, касающихся обработки текстовых файлов, описания процедур и функций, составления рекурсивных алгоритмов и использования динамических структур |
| ПК-6 способностью эффективно применять базовые математические знания и информационные технологии при решении проектно-технических и прикладных задач, связанных с развитием и использованием информационных технологий | Знать:  базовые математические знания и информационные технологий при решении проектно-технических и прикладных задач, связанных с развитием и использованием информационных технологий. Уметь: использовать базовые математические знания и информационные технологии при решении проектно-технических и прикладных задач, связанных с развитием и использованием информационных технологий. Владеть: навыками применения базовых математических знаний и информационных технологий при решении проектно-технических и прикладных задач, связанных с развитием и использованием информационных технологий. | Изучить основные сведения о бинарных деревьях.  Изучить способы формирования деревьев с заданной структурой и преобразования деревьев. |
| ПК-7 способностью разрабатывать и реализовывать процессы жизненного цикла информационных систем, программного обеспечения, сервисов систем информационных технологий, а также методы и механизмы оценки и анализа функционирования средств и систем информационных технологий | Знать: методы разработки и реализации процессов жизненного цикла информационных систем и программного обеспечения Уметь: разрабатывать и реализовывать процессы жизненного цикла информационных систем и программного обеспечения Владеть: навыками оценки и анализа функционирования информационных систем и программного обеспечения | Осуществить математическую формализацию индивидуальных заданий, касающихся работы с бинарными деревьями. |
| ПК-8 способностью применять на практике международные и профессиональные стандарты информационных технологий, современные парадигмы и методологии, инструментальные и вычислительные средства | Знать: методы применения на практике международных и профессиональных стандартов информационных технологий, современных парадигм и методологий, инструментальных и вычислительных средств. Уметь: применять на практике международные и профессиональные стандарты информационных технологий, современные парадигмы и методологии, инструментальные и вычислительные средства. Владеть: навыками использования на практике международных и профессиональных стандартов информационных технологий, современных парадигм и методологий, инструментальных и вычислительных средств. | Используя язык C#, составить алгоритмы решения индивидуальных заданий, касающихся работы с бинарными деревьями.  Осуществить визуализацию обработки бинарных деревьев.  В соответствии со стандартами оформления учебных текстовых документов Самарского университета составить отчет по практике. |

Дата выдачи задания 01.07.2019 г.

Срок представления на кафедру отчета о практике 18.07.2019 г.

Руководитель практики от университета,

доцент кафедры программных   
систем, к.т.н. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.А. Попова-Коварцева

*(подпись)*

Задание принял к исполнению

студент группы № 6213-020302D \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. Д. Павленко

*(подпись)*

Рабочий график (план) проведения выполнения   
индивидуального задания на практику

|  |  |  |
| --- | --- | --- |
| Дата (период) | Содержание задания | Результаты практики |
| 01.07.2019 | Прохождение инструктажа по ознакомлению с требованиями охраны труда, техники безопасности, пожарной безопасности, а также правилами внутреннего распорядка. | Инструктаж пройден, с требованиями охраны труда, техники безопасности, пожарной безопасности и правилами внутреннего распорядка ознакомлен. |
| 02.07.2019 | Составление индивидуального задания и рабочего графика (плана) проведения практики. Утверждение задания руководителем практики. | Рабочий график (план) и индивидуальное задание составлены и утверждены руководителем практики. |
| 03.07.2019-14.07.2019 | Выполнение индивидуального задания: | Индивидуальные задания выполнены. |
| 03.07.2019-06.07.2019 | Выполнение заданий из первого блока «Текстовые файлы». | Все задания из блока «Текстовые файлы» выполнены. |
| 06.07.2019-09.07.2019 | Выполнение заданий из второго блока «Функции с параметрами». | Все задания из блока «Функции с параметрами» выполнены. |
| 10.07.2019-12.07.2019 | Выполнение заданий из третьего блока «Динамические структуры данных». | Все задания из блока «Динамические структуры данных» выполнены. |
| 12.07.2019-14.07.2019 | Выполнение заданий из четвертого блока «Бинарные деревья». | Все задания из блока «Бинарные деревья» выполнены. |
| 14.07.2018-16.07.2018 | Написание, оформление письменного отчета и его предъявление руководителю практики | Отчет написан и предъявлен руководителю практики. |
| 17.07.2018 | Получение отзыва от руководителя практики. | Отзыв от руководителя практики получен. |
| 17.07.2018 | Подготовка устного отчета о прохождении практики. | Устный отчет о прохождении практики подготовлен. |
| 18.07.2018 | Защита отчета. Получение зачета по практике. | Зачет по практике получен. |

Руководитель практики от университета,

доцент кафедры программных   
систем, к.т.н. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.А. Попова-Коварцева

*(подпись)*

Введение

Требуется выполнить четыре блока задач. В блоке «Текстовые файлы» необходимо изучить особенности работы с текстовыми записи: чтения файла, запись в файл, редактирование содержимого; и решить соответствующие задачи. Перед решением задач блока «Функции с параметрами» необходимо изучить особенности создания функций с параметрами в выбранном языке программирования. В блоке «Динамические структуры данных» необходимо изучить и реализовать структуру данных двусвязный список. Для выполнения задач блока «Бинарные деревья» необходимо изучить и реализовать структуру данных двоичное дерево.

1. Работа с текстовыми файлами

Условие «дан текстовый файл» означает, что в наборе исходных данных указано имя данного файла (текстовая строка). Все исходные файлы в заданиях считаются существующими.

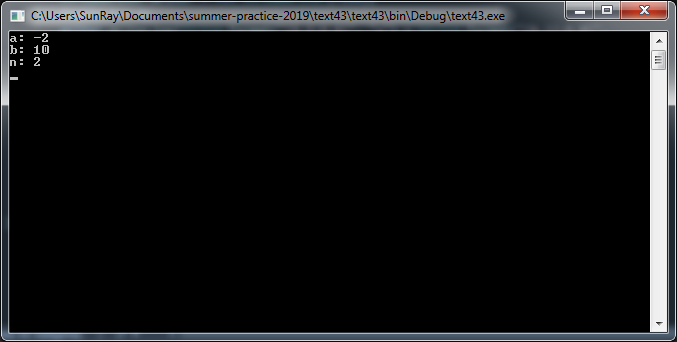
Если в задании требуется создать новый файл, то имя создаваемого файла также входит в набор используемых данных (и, как правило, является последним элементом этого набора).

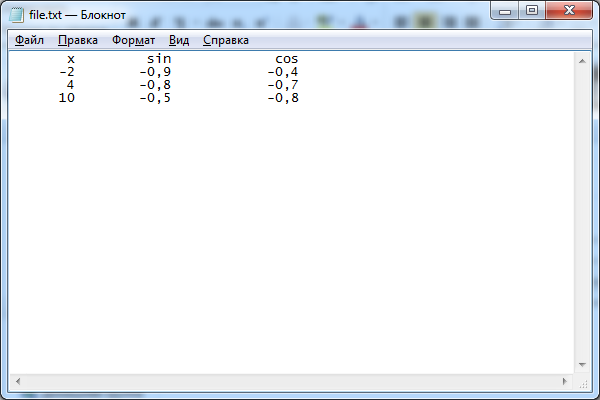
* 1. Текстовые файлы с числовой информацией

Даны вещественные чиста A, B и целое число N. Создать текстовый файл, содержащий таблицу значений функций sin(x) и cos(x) на промежутке [A, B] с шагом (B-A)/N. Таблица состоит из трех столбцов: с аргументами x (8 позиций, из них 4 под дробную часть) и со значениями sin(x) и cos(x). (по 12 позиций, из них 8 под дробную часть). Столбцы выравниваются по правому краю.

Для того чтобы выполнить данное задание, можем рассмотреть следующий алгоритм действий. Считываем входные данные и проверяем их на корректность. Создаем текстовый файл, в который форматированно выводим результаты функций sin(x) и cos(x).

Пример работы программы представлен на рисунках 1 и 2.

   
Рисунок 1 – Вводимые данные

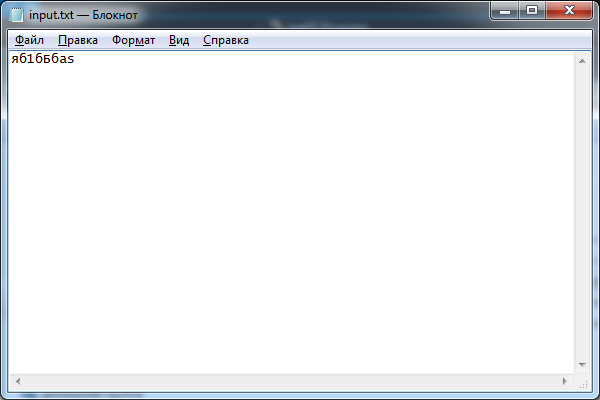
   
Рисунок 2 – Содержимое текстового файла после завершения работы программы

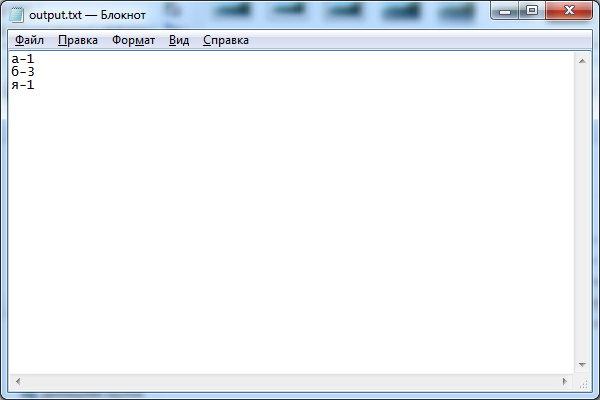
* 1. Дополнительные задания на обработку текстовых файлов

Дан текстовый файл. Посчитать количество появлений в нем каждой *строчной* (то есть маленькой) русской буквы и создать строковый файл, элементы которого имеют вид «<буква>-<число ее появлений>» (например, «а-25»). Буквы, отсутствующие в тексте, в файл не включать. Строки упорядочить по возрастанию кодов букв.

Для того чтобы выполнить данное задание, можем рассмотреть следующий алгоритм действий. Создаем структуру Pare, которая будет хранить букву и количество ее появлений. Затем считываем все символы из исходного файла input и обрабатываем их, добавляя каждую новую структуру Pare в список. В конце сортируем список по кодам букв и выводим в файл вывода output.

Содержимое исходного и конечного файлов представлены на рисунках 3 и 4.

   
Рисунок 3 – Содержимое исходного файла

   
Рисунок 4 – Содержимое конечного файла

1. Работа с составными типами данных в процедурах и функциях

В каждом задании требуется описать процедуру или функцию и затем использовать её для обработки исходных данных.

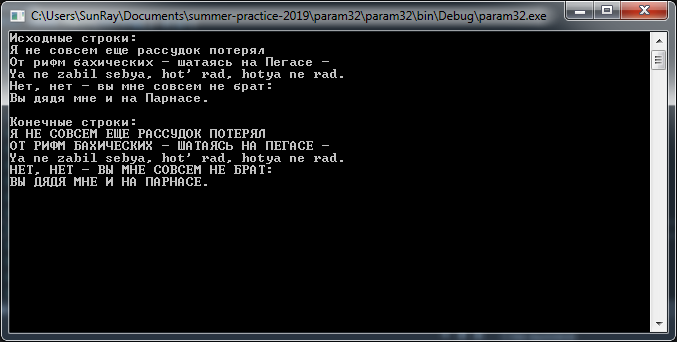
Все параметры любой функции считаются входными. Для процедур всегда указывается, какие параметры являются входными (или одновременно входными и выходными); если о виде параметра процедуры ничего не сказано, то он считается входным.

* 1. Строки

Описать процедуру UpCaseRus(*S*), преобразующую все строчные русские буквы строки S в прописные (остальные символы строки S не изменяются). Строка *S* является входным и выходным параметром. Используя процедуру UpCaseRus, преобразовать пять данных строк.

Для того чтобы выполнить данное задание, можем рассмотреть следующий алгоритм действий. Создадим пять строк и опишем процедуру UpCaseRus(*S*). Эта процедура возвращает преобразованную строку. Применим процедуру UpCaseRus ко всем строкам и выведем результат на консоль.

Результат работы программы представлен на рисунке 5.

   
Рисунок 5 – Пример работы программы

* 1. Дополнительные задания на обработку строк

Описать процедуру AddLineNumbers(*S*, *N*, *K*, *L*), добавляющую в начало каждой строки существующего текстового файла с именем *S* ее порядковый номер: первая строка получает номер *N*, вторая – *N*+1 и т.д. Номер отображается в *K* позициях, выравнивается по правому краю и отделяется от последующего текста *L* пробелами (*K*>0, *L*>0). Если строка файла является пустой, то она также нумеруется, но пробелы после номера не добавляются. Применить эту процедуру к данному файлу, используя указанные значения *N*, *K* и *L*.

Результат работы программы представлен на рисунках 6 и 7.

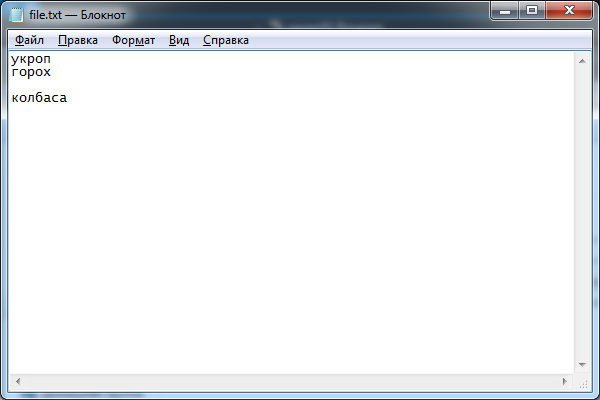


Рисунок 6 – Исходное содержимое файла

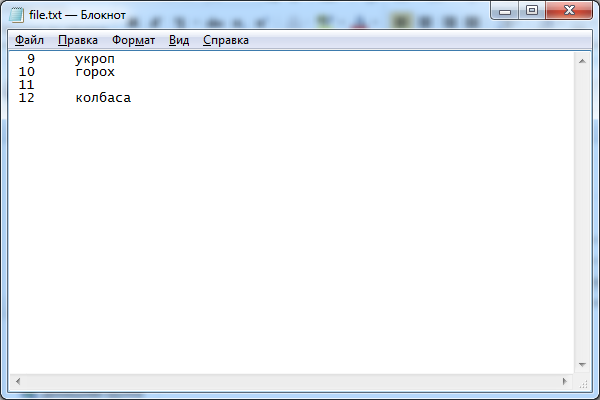


Рисунок 7 – Конечное содержимое файла

1. Работа с указателями и динамическими структурами данных

Все числа, упоминаемые в заданиях данной группы, являются целыми. Все указатели указывают на записи типа Node.

Если в задании говорится о номерах элементов списка, то предполагается, что элементы нумеруются от 1.

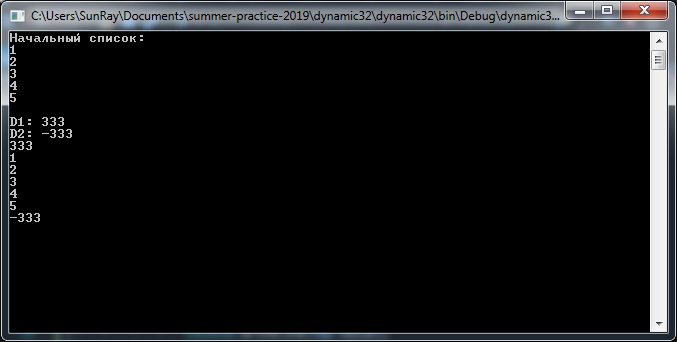
Так как переменные типа «указатель» предназначены для хранения адресов, в формулировках заданий слова «указатель» (на элемент данных) и «адрес» (элемента данных) используются как синонимы. Имя nil для нулевого указателя в формулировках заданий заимствовано из языка Pascal. В языке C# в качестве нулевого указателя используется null.

* 1. Двусвязный список

Даны числа *D1* и *D2* и указатель *P0* на один из элементов непустого двусвязного списка. Добавить в начало списка новый элемент со значением *D1*, а в конец – новый элемент со значением *D2*. Вывести адреса первого и последнего элементов полученного списка.

Для того чтобы выполнить данное задание, можем рассмотреть следующий алгоритм действий. Для реализации двусвязного списка используем класс LinkedList. В главной программе инициализируем список, состоящий из 5 элементов. Далее считаем входные значения *D1* и *D2* и добавим их в начало и конец списка соответственно с помощью циклов.

На рисунке 8 представлен результат работы программы.

   
Рисунок 8 – Пример работы программы

1. Работа с бинарными деревьями

Дерево – это иерархическая структура данных, состоящая из элементов (вершин, или узлов), которые связаны между собой отношениями типа «родительская вершина – дочерняя вершина». Определить дерево с вершинами типа Т можно следующим образов. Это либо пустое дерево, не содержащее ни одной вершины, либо некоторая вершина типа Т, соединяющая ветвями с конечным числом отдельных деревьев с вершинами типа Т (эти деревья называются поддеревьями).

Чаще всего дерево изображается в виде графа, вершинами которого являются вершины дерева, а ребрами – его ветви. Начальная вершина дерева, называемая корнем, изображается в виде верхней части графа, и считается, что она находится на нулевом уровне. Вершина Y, расположенная ниже вершины X и соединённая с ней ветвью, называется непосредственным потоком вершины X, или её дочерней вершиной (а вершина X, соответственно - непосредственным предком вершины Y, или её родительской вершиной). Если вершина X находится на уровне K, то считается, что все её непосредственные потомки находятся на уровне К+1. На графе, изображающем дерево, все вершины одного уровня располагаются на одной горизонтали.

Номер максимального уровня дерева называется глубиной (или высотой) дерева. Если вершина не имеет потомков, то она называется терминальной вершиной, или листом. Нетерминальная вершина называется внутренней. Число непосредственных потомков внутренней вершины дерева называется степенью этой вершины. Максимальная степень всех вершин дерева называется степенью дерева. Дерево называется упорядоченным, если все непосредственные потомки любой вершины упорядочены.

Особым видом деревьев являются бинарные деревья.

Бинарное дерево с вершинами типа Т можно определить следующим образом: это либо пустое дерево, не содержащее ни одной вершины, либо некоторая вершина типа Т, соединённая ветвями с двумя бинарными деревьями с вершинами типа Т (эти деревья называются левым и правым поддеревом).

Для выполнения заданий этого блока создадим класс вершины дерева Node, содержащий в себе значение вершины Value и ссылки на две дочерние вершины LeftChild и RightChild.

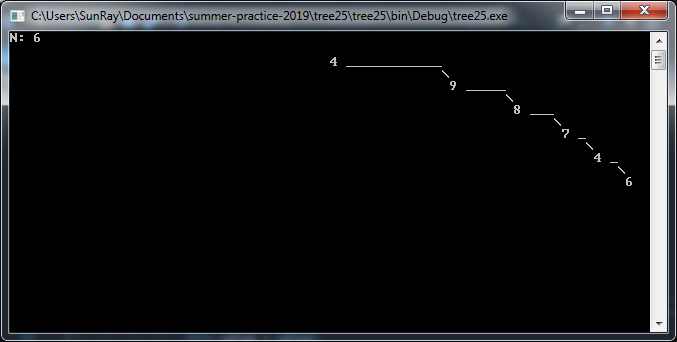
В главной программе проинициализируем дерево, создав нужное количество вершин и установив между ними связи.

* 1. Формирование бинарного дерева

Дано число N (> 0) и набор из N чисел. Создать дерево из N вершин, в котором каждая вершина (кроме корня) является правой дочерней вершиной. Каждой создаваемой вершине присваивать очередное значение из исходного набора. Вывести указатель на корень созданного дерева.

Для того чтобы выполнить данное задание, можем рассмотреть следующий алгоритм действий. Считаем входные данные в виде количества вершин N и создадим массив из N случайных чисел. Затем добавим корню правого ребенка N раз, каждый раз смещая указатель на уровень ниже. Для визуализации дерева воспользуемся процедурой treeOut().

Визуальное представление дерева представлено на рисунке 9.

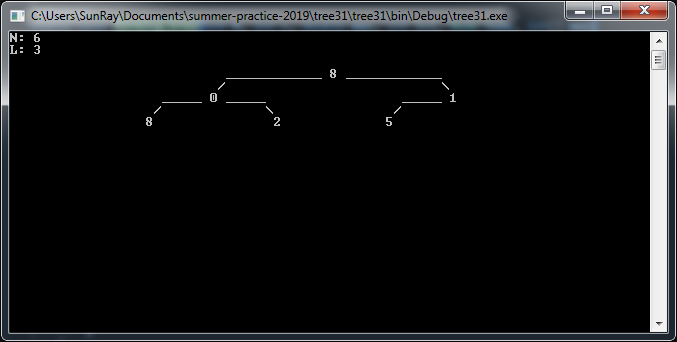
  
Рисунок 9 – Пример работы программы

* 1. Преобразование бинарного дерева

Даны положительные числа L, N (N > L) и набор из N чисел. Создать дерево глубины L, содержащее вершины со значениями из исходного набора. Вершины добавлять к дереву в префиксном порядке, используя алгоритм, который для каждой вершины уровня, не превышающего L, вначале создает саму вершину с очередным значением из исходного набора, затем ее левое поддерево соответствующей глубины, а затем ее правое поддерево. Если для заполнения дерева глубины L требуется менее N вершин, то оставшиеся числа из исходного набора не использовать. Вывести указатель на корень созданного дерева.

Для того чтобы выполнить данное задание, можем рассмотреть следующий алгоритм действий. Создадим рекурсивную процедуру forming\_tree(). Эта процедура проверяет наличие левого и правого ребенка у вершины и, при наличии оных, рекурсивно вызывает себя для левой и правой вершины соответственно.

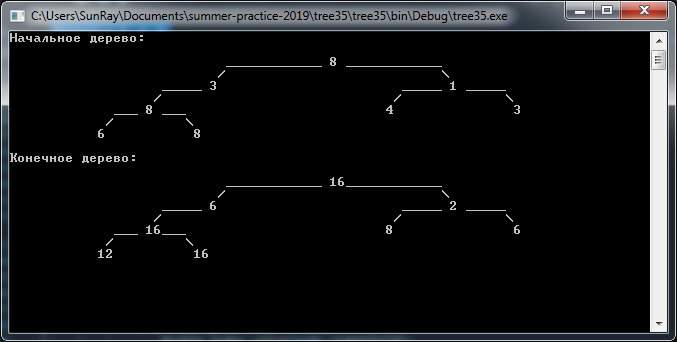
Визуальное представление дерева до и после преобразования представлено на рисунке 10.

  
Рисунок 10 – Пример работы программы

Дан указатель P1 на корень непустого дерева. Удвоить значение каждой вершины дерева.

Для того чтобы выполнить данное задание, можем рассмотреть следующий алгоритм действий. Создадим и заполним дерево случайными числами. Затем воспользуемся рекурсивной процедурой double\_nodes\_value(), которая удваивает значение вершины и рекурсивно запускает себя у левого и правого ребенка данной вершины.

Визуальное представление дерева до и после преобразования представлено на рисунке 11.

  
Рисунок 11 – Пример работы программы

Дан указатель P1 на корень непустого дерева. Для вершин дерева, имеющих две дочерние вершины, удалить одну из дочерних вершин: правую, если родительская вершина имеет четное значение, и левую в противном случае (вершины дерева перебирать в префиксном порядке, при удалении вершины удалять и всех ее потомков). Для удаленных вершин освобождать память, которую они занимали.

Для того чтобы выполнить данное задание, можем рассмотреть следующий алгоритм действий. Создаем дерево и наполняем его случайными значениями. Затем запускаем рекурсивную процедуру alg(*node*), принимающую на вход вершину *node*. Данная процедура определяет, является значение вершины *node* четным или нечетным и, в зависимости от ее четности, удаляет правого или левого ребенка. Затем процедура запускается с указанием оставшегося ребенка.

Визуальное представление дерева до и после преобразования представлено на рисунке 12.

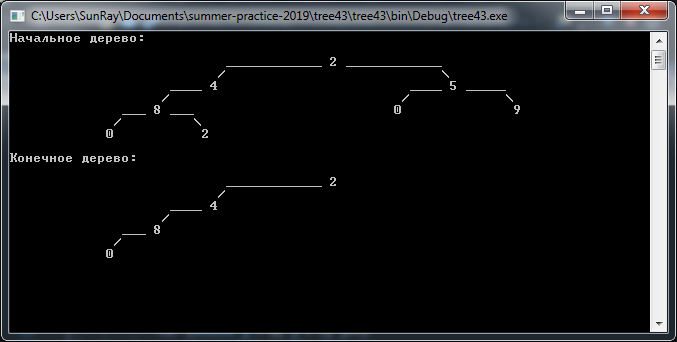


Рисунок 12 – Пример работы программы

Дан указатель P1 на корень непустого дерева. Ко всем вершинам дерева, которые являются листьями, добавить по две дочерние вершины-листа: левую со значением 10 и правую со значением 11.

Для того чтобы выполнить данное задание, можем рассмотреть следующий алгоритм действий. Создаем дерево и наполняем его случайными значениями. Затем запускаем рекурсивную процедуру add\_nodes\_to\_leaves(*node*), принимающую на вход вершину *node*. Эта процедура проверяет наличие детей у вершины node и, в случае отсутствия оных, добавляет ей левого и правого ребенка со значениями 10 и 11 соответственно. Если же у вершины node есть один или два ребенка, то процедура запускается для него (них).

Визуальное представление дерева до и после преобразования представлено на рисунке 13.

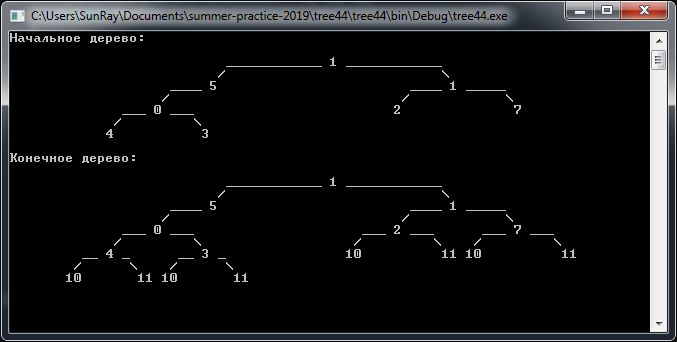


Рисунок 13 – Пример работы программы

Список использованных источников

1. Сайт о программировании METANIT.COM [Электронный ресурс]. URL: https://metanit.com/ (дата обращения: 04.07.2019).
2. Портал об алгоритмах и олимпиадном программировании [Электронный ресурс] URL: http://www.e-maxx-ru.1gb.ru (дата обращения: 07.07.2019).
3. Троелсен Э. С# и платформа .NET. Библиотека программиста. СПб.: Питер, 2007. 796 с.
4. СТО 02068410-004-2018 Общие требования к учебным текстовым документам. [Электронный ресурс]. URL: http://repo.ssau.ru/handle/Metodicheskie-izdaniya/Obshie-trebovaniya-k-uchebnym-tekstovym-dokumentam-Elektronnyi-resurs-STO-020684100042018-standart-organizacii-prinyat-9-okt-2007-g-s-izm-utverzhd-v-fevr-2018-g-72605?mode=simple (дата обращения: 14.07.2019).

Приложение А  
Листинг программ из блока «Текстовые файлы»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace text43

{

class Program

{

static void Main(string[] args)

{

bool error = false;

double a = 0;

double b = 0;

int n = 0;

try

{

Console.Write("a: ");

a = Convert.ToDouble(Console.ReadLine());

Console.Write("b: ");

b = Convert.ToDouble(Console.ReadLine());

Console.Write("n: ");

n = Convert.ToInt16(Console.ReadLine());

if (a >= b || n <= 0)

throw new System.FormatException();

}

catch (Exception e)

{

error = true;

Console.WriteLine(e.Message);

}

if (!error)

{

FileStream file = new FileStream("C:\\Users\\SunRay\\Documents\\summer-practice-2019\\text43\\file.txt", FileMode.Create);

StreamWriter writer = new StreamWriter(file);

double x = a;

writer.WriteLine(String.Format("{0,8:0.#}", "x") + String.Format("{0,12:0.#}",

"sin") + "\t" + String.Format("{0,12:0.#}", "cos"));

while (x <= b)

{

writer.WriteLine(String.Format("{0,8:0.#}", x) + String.Format("{0,12:0.#}",

Math.Sin(x)) + "\t" +String.Format("{0,12:0.#}", Math.Cos(x)));

x += (b-a)/n;

}

writer.Close();

file.Close();

}

Console.Read();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

//Посчитать в файле кол-во строчных русских букв и вывести их в другой файл

namespace text57

{

public class Pare

{

public char Letter { get; set; }

public int Amount { get; set; }

public Pare(char letter, int amount)

{

Letter = letter;

Amount = amount;

}

}

class Program

{

static void Main(string[] args)

{

FileStream file = new FileStream("C:\\Users\\SunRay\\Documents\\summer-practice-2019\\text57\\input.txt", FileMode.Open);

StreamReader reader = new StreamReader(file, Encoding.Default);

List<Pare> list = new List<Pare>();

char symbol = (char)reader.Read();

while (symbol != '\uffff')

{

if (symbol >= 'а' && symbol <= 'я')

{

Pare pare = list.Find(x => x.Letter == symbol);

if (pare == null)

{

Pare tmp = new Pare(symbol, 1);

list.Add(tmp);

}

else pare.Amount++;

}

symbol = (char)reader.Read();

}

reader.Close();

file.Close();

file = new FileStream("C:\\Users\\SunRay\\Documents\\summer-practice-2019\\text57\\output.txt", FileMode.Create);

StreamWriter writer = new StreamWriter(file);

var sortedList = list.OrderBy(pare=>pare.Letter);

foreach (Pare pare in sortedList)

{

writer.WriteLine(pare.Letter + "-" + pare.Amount);

}

writer.Close();

file.Close();

}

}

}

Приложение Б  
Листинг программ из блока «Функции с параметрами»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

//Преобразование прописных русских букв в заглавные

namespace param32

{

class Program

{

static void Main(string[] args)

{

string s1 = "Я не совсем еще рассудок потерял";

string s2 = "От рифм бахических — шатаясь на Пегасе —";

string s3 = "Ya ne zabil sebya, hot' rad, hotya ne rad.";

string s4 = "Нет, нет — вы мне совсем не брат:";

string s5 = "Вы дядя мне и на Парнасе.";

Console.WriteLine("Исходные строки:");

Console.WriteLine(s1);

Console.WriteLine(s2);

Console.WriteLine(s3);

Console.WriteLine(s4);

Console.WriteLine(s5);

Console.WriteLine("\nКонечные строки: ");

Console.WriteLine(UpCaseRus(s1));

Console.WriteLine(UpCaseRus(s2));

Console.WriteLine(UpCaseRus(s3));

Console.WriteLine(UpCaseRus(s4));

Console.WriteLine(UpCaseRus(s5));

Console.Read();

}

public static string UpCaseRus(string S)

{

int i = 0;

while (i < S.Length)

{

if (S[i] >= 'а' && S[i] <= 'я')

{

Char c = Char.ToUpper(S[i]);

S = S.Remove(i, 1).Insert(i, c.ToString());

}

i++;

}

return S;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace param51

{

class Program

{

static void Main(string[] args)

{

addLineNumbers("C:\\Users\\SunRay\\Documents\\summer-practice-2019\\param51\\file.txt", 9, 3, 5);

}

public static void addLineNumbers(string S, int N, int K, int L)

{

string format = "{0," + K + ":0}";

FileStream file1 = new FileStream(S, FileMode.Open);

string tmpFilePath = Path.GetTempFileName(); //создается файл и возвращается полный путь к нему

FileStream file2 = new FileStream(tmpFilePath, FileMode.Create);

StreamReader reader = new StreamReader(file1, Encoding.Default);

StreamWriter writer = new StreamWriter(file2, Encoding.Default);

string str = reader.ReadLine();

int i = N;

while (str != null)

{

writer.Write(format, i++);

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

writer.Write(" ");

writer.WriteLine(str);

str = reader.ReadLine();

}

writer.Close();

reader.Close();

file1.Close();

File.Delete(S);

File.Move(tmpFilePath, S);

}

}

}

Приложение В  
Листинг программ из блока «Динамические структуры данных»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

//Даны числа D1, D2 и указатель P0 на один из элементов двусвязного списка.

//Добавить D1 и D2 в начало и в конец списка соответственно. Вывести их адреса

namespace dynamic32

{

class Program

{

static void Main(string[] args)

{

//создание списка и его заполнение

LinkedList<int> list = new LinkedList<int>();

list.AddLast(1);

list.AddLast(2);

list.AddLast(3);

list.AddLast(4);

list.AddLast(5);

Console.WriteLine("Начальный список:");

LinkedListNode<int> tmp = list.First;

while (tmp.Next != null)

{

Console.WriteLine(tmp.Value);

tmp = tmp.Next;

}

Console.WriteLine(tmp.Value);

//ввод значений D1, D2, и указателя P0

Console.Write("\nD1: ");

int D1 = Convert.ToInt16(Console.ReadLine());

Console.Write("D2: ");

int D2 = Convert.ToInt16(Console.ReadLine());

LinkedListNode<int> P0 = list.First.Next.Next;

//добавление D1 в начало списка

while (P0.Previous != null)

P0 = P0.Previous;

list.AddBefore(P0, D1);

//C# не позволяет работать с адресами в связных списках, но если бы это было возможно,

//то строка ниже позволила бы нам получить адрес для D1

//Console.WriteLine("Адрес D1: " + &P0.Previous);

//добавление D2 в конец списка

while (P0.Next != null)

P0 = P0.Next;

list.AddAfter(P0, D2);

//вывод элементов списка

while (list.Count != 0)

{

Console.WriteLine(list.First.Value);

list.RemoveFirst();

}

Console.Read();

}

}

}

Приложение Г  
Листинг программ из блока «Бинарные деревья»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

//создание N-уровневого дерева только из правых детей

namespace tree25

{

class Program

{

static void Main(string[] args)

{

Console.Write("N: ");

try

{

int N = Convert.ToInt16(Console.ReadLine());

if (N > 0)

{

int[] mas = new int[N];

Random random = new Random();

//заполнение массива значений для дерева

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

mas[i] = random.Next(10);

//заполнение дерева значениями из массива

Node root = new Node(mas[0]);

Node tmp = root;

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

{

tmp.addRight(mas[i]);

tmp = tmp.rightChild;

}

//вывод дерева

Console.SetCursorPosition(40, 2);

root.treeOut(Console.WindowWidth / 2, 2, 0);

}

else Console.WriteLine("Некорректное значение N");

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

Console.Read();

}

class Node

{

public int value { get; set; }

public Node leftChild { get; set; }

public Node rightChild { get; set; }

public Node(int value)

{

this.value = value;

}

public void addLeft(int value)

{

this.leftChild = new Node(value);

}

public void addRight(int value)

{

this.rightChild = new Node(value);

}

public void treeOut(int offset, int level, int deep)

{

Console.SetCursorPosition(offset, level);

Console.Write(this.value);

if (leftChild != null)

{

int curPos = offset - 2;

int childPos = -1;

childPos = (int)(curPos - Console.WindowWidth / Math.Pow(2, deep + 2));

switch (deep)

{

case 0:

childPos += 8;

break;

case 1:

childPos += 6;

break;

case 2:

childPos += 2;

break;

case 3:

childPos += 1;

break;

}

if (childPos >= 0)

{

Console.SetCursorPosition(childPos, level + 1);

Console.Write("/");

while (curPos > childPos)

{

Console.SetCursorPosition(curPos, level);

Console.Write("\_");

curPos--;

}

this.leftChild.treeOut(childPos - 1, level + 2, deep + 1);

}

}

if (rightChild != null)

{

int curPos = offset + 2;

int childPos = -1;

childPos = (int)(curPos + Console.WindowWidth / Math.Pow(2, deep + 2));

switch (deep)

{

case 0:

childPos -= 8;

break;

case 1:

childPos -= 5;

break;

case 2:

childPos -= 2;

break;

case 3:

childPos -= 1;

break;

}

if (childPos >= 0)

{

Console.SetCursorPosition(childPos, level + 1);

Console.Write("\\");

while (curPos < childPos)

{

Console.SetCursorPosition(curPos, level);

Console.Write("\_");

curPos++;

}

this.rightChild.treeOut(childPos + 1, level + 2, deep + 1);

}

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace tree31

{

class Node

{

public int value { get; set; }

public Node leftChild { get; set; }

public Node rightChild { get; set; }

public Node(int value)

{

this.value = value;

}

public void addLeft(int value)

{

this.leftChild = new Node(value);

}

public void addRight(int value)

{

this.rightChild = new Node(value);

}

public void treeOut(int offset, int level, int deep) //offset - отступ слева, level - отступ сверху

{

Console.SetCursorPosition(offset, level);

Console.Write(this.value);

if (leftChild != null)

{

int curPos = offset - 2;

int childPos = -1;

childPos = (int)(curPos - Console.WindowWidth / Math.Pow(2, deep + 2));

switch (deep)

{

case 0:

childPos += 8;

break;

case 1:

childPos += 5;

break;

case 2:

childPos += 2;

break;

case 3:

childPos += 1;

break;

}

if (childPos >= 0)

{

Console.SetCursorPosition(childPos, level + 1);

Console.Write("/");

while (curPos > childPos)

{

Console.SetCursorPosition(curPos, level);

Console.Write("\_");

curPos--;

}

this.leftChild.treeOut(childPos - 1, level + 2, deep + 1);

}

}

if (rightChild != null)

{

int curPos = offset + 2;

int childPos = -1;

childPos = (int)(curPos + Console.WindowWidth / Math.Pow(2, deep + 2));

switch (deep)

{

case 0:

childPos -= 8;

break;

case 1:

childPos -= 5;

break;

case 2:

childPos -= 2;

break;

case 3:

childPos -= 1;

break;

}

if (childPos >= 0)

{

Console.SetCursorPosition(childPos, level + 1);

Console.Write("\\");

while (curPos < childPos)

{

Console.SetCursorPosition(curPos, level);

Console.Write("\_");

curPos++;

}

this.rightChild.treeOut(childPos + 1, level + 2, deep + 1);

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

//Формирование дерева из N вершин и L уровней в префиксном порядке

//Префиксный порядок - сначала действия происходят с вершиной, потом с ее левым и правым детьми

namespace tree31

{

class Program

{

public static int k = 1;

static void Main(string[] args)

{

Node root = null;

try

{

Console.Write("N: ");

int N = Convert.ToInt16(Console.ReadLine());

Console.Write("L: ");

int L = Convert.ToInt16(Console.ReadLine());

if (N <= 0 || L <= 0 || N <= L)

throw new Exception();

//заполнение массива значениями

Random random = new Random();

int[] mas = new int[N];

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

mas[j] = random.Next(10);

//максимальное кол-во вершин

double max = 0;

for (double j = 0; j < L; j++)

max += Math.Pow(2, j);

//формирование дерева

root = new Node(mas[0]);

forming\_tree(N, L, max, 1, root, mas);

}

catch (Exception)

{

}

root.treeOut(Console.WindowWidth / 2, 3, 0);

Console.Read();

}

static void forming\_tree(int N, int L, double max, int lvl, Node node, int[] mas)

{

if (lvl < L && k < N && k <= max)

{

if (node.leftChild == null)

{

lvl++;

node.leftChild = new Node(mas[k]);

k++;

forming\_tree(N, L, max, lvl, node.leftChild, mas);

lvl--;

}

if (node.rightChild == null)

{

lvl++;

node.rightChild = new Node(mas[k]);

k++;

forming\_tree(N, L, max, lvl, node.rightChild, mas);

lvl--;

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace tree35

{

class Node

{

public int value { get; set; }

public Node leftChild { get; set; }

public Node rightChild { get; set; }

public Node(int value)

{

this.value = value;

}

public void addLeft(int value)

{

this.leftChild = new Node(value);

}

public void addRight(int value)

{

this.rightChild = new Node(value);

}

public void treeOut(int offset, int level, int deep) //offset - отступ слева, level - отступ сверху

{

Console.SetCursorPosition(offset, level);

Console.Write(this.value);

if (leftChild != null)

{

int curPos = offset - 2;

int childPos = -1;

childPos = (int)(curPos - Console.WindowWidth / Math.Pow(2, deep + 2));

switch (deep)

{

case 0:

childPos += 8;

break;

case 1:

childPos += 5;

break;

case 2:

childPos += 2;

break;

case 3:

childPos += 1;

break;

}

if (childPos >= 0)

{

Console.SetCursorPosition(childPos, level + 1);

Console.Write("/");

while (curPos > childPos)

{

Console.SetCursorPosition(curPos, level);

Console.Write("\_");

curPos--;

}

this.leftChild.treeOut(childPos - 1, level + 2, deep + 1);

}

}

if (rightChild != null)

{

int curPos = offset + 2;

int childPos = -1;

childPos = (int)(curPos + Console.WindowWidth / Math.Pow(2, deep + 2));

switch (deep)

{

case 0:

childPos -= 8;

break;

case 1:

childPos -= 5;

break;

case 2:

childPos -= 2;

break;

case 3:

childPos -= 1;

break;

}

if (childPos >= 0)

{

Console.SetCursorPosition(childPos, level + 1);

Console.Write("\\");

while (curPos < childPos)

{

Console.SetCursorPosition(curPos, level);

Console.Write("\_");

curPos++;

}

this.rightChild.treeOut(childPos + 1, level + 2, deep + 1);

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

//Удвоение значений всех вершин

namespace tree35

{

class Program

{

static int k = 1;

static void Main(string[] args)

{

Console.WriteLine("Начальное дерево: ");

//заполнение массива значениями

Random random = new Random();

int[] mas = new int[8];

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

mas[i] = random.Next(10);

//максимальное кол-во вершин

double max = 0;

for (double j = 0; j < 3; j++)

max += Math.Pow(2, j);

//формирование дерева

Node root = new Node(mas[0]);

root.leftChild = new Node(mas[1]);

root.rightChild = new Node(mas[2]);

root.rightChild.rightChild = new Node(mas[3]);

root.rightChild.leftChild = new Node(mas[4]);

root.leftChild.leftChild = new Node(mas[5]);

root.leftChild.leftChild.leftChild = new Node(mas[6]);

root.leftChild.leftChild.rightChild = new Node(mas[7]);

root.treeOut(Console.WindowWidth / 2, 2, 0);

Console.SetCursorPosition(0, 10);

Console.WriteLine("Конечное дерево: ");

double\_nodes\_value(root);

root.treeOut(Console.WindowWidth / 2, 12, 0);

Console.Read();

}

static void double\_nodes\_value(Node node)

{

node.value = node.value \* 2;

if (node.leftChild != null)

double\_nodes\_value(node.leftChild);

if (node.rightChild != null)

double\_nodes\_value(node.rightChild);

}

static void formin\_tree(int nodes\_amount, int tree\_level, double max, int tmp\_lvl\_1, Node node, int[] value\_mas)

{

if (tmp\_lvl\_1 < tree\_level && k < nodes\_amount && k <= max)

{

if (node.leftChild == null)

{

tmp\_lvl\_1++;

node.leftChild = new Node(value\_mas[k]);

k++;

formin\_tree(nodes\_amount, tree\_level, max, tmp\_lvl\_1, node.leftChild, value\_mas);

tmp\_lvl\_1--;

}

if (node.rightChild == null)

{

tmp\_lvl\_1++;

node.rightChild = new Node(value\_mas[k]);

k++;

formin\_tree(nodes\_amount, tree\_level, max, tmp\_lvl\_1, node.rightChild, value\_mas);

tmp\_lvl\_1--;

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace tree43

{

class Node

{

public int value { get; set; }

public Node leftChild { get; set; }

public Node rightChild { get; set; }

public Node(int value)

{

this.value = value;

}

public void addLeft(int value)

{

this.leftChild = new Node(value);

}

public void addRight(int value)

{

this.rightChild = new Node(value);

}

public void treeOut(int offset, int level, int deep) //offset - отступ слева, level - отступ сверху

{

Console.SetCursorPosition(offset, level);

Console.Write(this.value);

if (leftChild != null)

{

int curPos = offset - 2;

int childPos = -1;

childPos = (int)(curPos - Console.WindowWidth / Math.Pow(2, deep + 2));

switch (deep)

{

case 0:

childPos += 8;

break;

case 1:

childPos += 6;

break;

case 2:

childPos += 2;

break;

case 3: childPos += 1;

break;

}

if (childPos >= 0)

{

Console.SetCursorPosition(childPos, level + 1);

Console.Write("/");

while (curPos > childPos)

{

Console.SetCursorPosition(curPos, level);

Console.Write("\_");

curPos--;

}

this.leftChild.treeOut(childPos - 1, level + 2, deep + 1);

}

}

if (rightChild != null)

{

int curPos = offset + 2;

int childPos = -1;

childPos = (int)(curPos + Console.WindowWidth / Math.Pow(2, deep + 2));

switch (deep)

{

case 0:

childPos -= 8;

break;

case 1:

childPos -= 5;

break;

case 2:

childPos -= 2;

break;

case 3:

childPos -= 1;

break;

}

if (childPos >= 0)

{

Console.SetCursorPosition(childPos, level + 1);

Console.Write("\\");

while (curPos < childPos)

{

Console.SetCursorPosition(curPos, level);

Console.Write("\_");

curPos++;

}

this.rightChild.treeOut(childPos + 1, level + 2, deep + 1);

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

//Удаление детей в префиксном порядке. Если вершина четная - удаляем правого, нечетная - левого

namespace tree43

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Начальное дерево: ");

//заполнение массива значениями

Random random = new Random();

int[] mas = new int[8];

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

mas[i] = random.Next(10);

//максимальное кол-во вершин

double max = 0;

for (double j = 0; j < 3; j++)

max += Math.Pow(2, j);

//формирование дерева

Node root = new Node(mas[0]);

root.leftChild = new Node(mas[1]);

root.rightChild = new Node(mas[2]);

root.rightChild.rightChild = new Node(mas[3]);

root.rightChild.leftChild = new Node(mas[4]);

root.leftChild.leftChild = new Node(mas[5]);

root.leftChild.leftChild.leftChild = new Node(mas[6]);

root.leftChild.leftChild.rightChild = new Node(mas[7]);

root.treeOut(Console.WindowWidth / 2, 2, 0);

Console.SetCursorPosition(0, 10);

Console.WriteLine("Конечное дерево: ");

alg(root);

root.treeOut(Console.WindowWidth / 2, 12, 0);

Console.Read();

}

static void alg(Node node)

{

if (node.leftChild != null && node.rightChild != null)

{

if (node.value % 2 == 1)

{

node.leftChild = null;

GC.Collect();

alg(node.rightChild);

}

else

{

node.rightChild = null;

GC.Collect();

alg(node.leftChild);

}

}

else

{

if (node.leftChild != null)

alg(node.leftChild);

if (node.rightChild != null)

alg(node.rightChild);

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace tree44

{

class Node

{

public int value { get; set; }

public Node leftChild { get; set; }

public Node rightChild { get; set; }

public Node(int value)

{

this.value = value;

}

public void addLeft(int value)

{

this.leftChild = new Node(value);

}

public void addRight(int value)

{

this.rightChild = new Node(value);

}

public void treeOut(int offset, int level, int deep) //offset - отступ слева, level - отступ сверху

{

Console.SetCursorPosition(offset, level);

Console.Write(this.value);

if (leftChild != null)

{

int curPos = offset - 2;

int childPos = -1;

childPos = (int)(curPos - Console.WindowWidth / Math.Pow(2, deep + 2));

switch (deep)

{

case 0:

childPos += 8;

break;

case 1:

childPos += 6;

break;

case 2:

childPos += 2;

break;

case 3: childPos += 1;

break;

}

if (childPos >= 0)

{

Console.SetCursorPosition(childPos, level + 1);

Console.Write("/");

while (curPos > childPos)

{

Console.SetCursorPosition(curPos, level);

Console.Write("\_");

curPos--;

}

this.leftChild.treeOut(childPos - 1, level + 2, deep + 1);

}

}

if (rightChild != null)

{

int curPos = offset + 2;

int childPos = -1;

childPos = (int)(curPos + Console.WindowWidth / Math.Pow(2, deep + 2));

switch (deep)

{

case 0:

childPos -= 8;

break;

case 1:

childPos -= 5;

break;

case 2:

childPos -= 2;

break;

case 3:

childPos -= 1;

break;

}

if (childPos >= 0)

{

Console.SetCursorPosition(childPos, level + 1);

Console.Write("\\");

while (curPos < childPos)

{

Console.SetCursorPosition(curPos, level);

Console.Write("\_");

curPos++;

}

this.rightChild.treeOut(childPos + 1, level + 2, deep + 1);

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

//Добавление ко всем листьям детей 10 и 11 слева и справа соответственно

namespace tree44

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Начальное дерево: ");

//заполнение массива значениями

Random random = new Random();

int[] mas = new int[8];

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

mas[i] = random.Next(10);

//максимальное кол-во вершин

double max = 0;

for (double j = 0; j < 3; j++)

max += Math.Pow(2, j);

//формирование дерева

Node root = new Node(mas[0]);

root.leftChild = new Node(mas[1]);

root.rightChild = new Node(mas[2]);

root.rightChild.rightChild = new Node(mas[3]);

root.rightChild.leftChild = new Node(mas[4]);

root.leftChild.leftChild = new Node(mas[5]);

root.leftChild.leftChild.leftChild = new Node(mas[6]);

root.leftChild.leftChild.rightChild = new Node(mas[7]);

root.treeOut(Console.WindowWidth / 2, 2, 0);

Console.SetCursorPosition(0, 10);

Console.WriteLine("Конечное дерево: ");

add\_nodes\_to\_leaves(root);

root.treeOut(Console.WindowWidth / 2, 12, 0);

Console.Read();

}

static void add\_nodes\_to\_leaves(Node node)

{

if (node.leftChild == null && node.rightChild == null)

{

node.leftChild = new Node(10);

node.rightChild = new Node(11);

}

else

{

if (node.leftChild != null)

add\_nodes\_to\_leaves(node.leftChild);

if (node.rightChild != null)

add\_nodes\_to\_leaves(node.rightChild);

}

}

}

}

ОТЗЫВ   
о прохождении практики

Вид практики учебная

Тип практики Практика по получению первичных профессиональных умений и навыков,

в том числе первичных умений и навыков научно-исследовательской деятельности

Сроки прохождения практики: с 01.07.2019 г. по 18.07.2019 г.

по направлению подготовки 02.03.02

«Фундаментальная информатика и информационные технологии»

(уровень бакалавриата)

направленность (профиль) «Информационные технологии»

студентом группы № 6213-020302D Павленко Вячеслава Дмитриевича

|  |  |  |
| --- | --- | --- |
| № п/п | Критерии оценки | Оценка (по 5-ти балльной шкале) |
|  | Общая систематичность и ответственность работы в ходе практики |  |
|  | Достижение планируемых результатов практики |  |
|  | Корректность в сборе, анализе и интерпретации представляемых данных |  |
|  | Степень личного участия и самостоятельности практиканта в представляемом отчете о практике |  |
|  | Качество оформления отчетной документации |  |
|  | ИТОГОВАЯ ОЦЕНКА |  |

Руководитель практики

от университета,

доцент кафедры программных   
систем, к.т.н. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.А. Попова-Коварцева