**Отчет по лабораторной работе № 23** по курсу алгоритмы и структуры данных

Студент группы М8О-101БВ-24 Волков Алексей Александрович, № по списку 6

Контакты e-mail

Работа выполнена: «22.04 » 2025 г.

Преподаватель: \_\_\_\_\_\_\_ каф. 806 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Входной контроль знаний с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отчет сдан « » \_\_\_\_\_\_\_\_\_202 \_\_ г., итоговая оценка \_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. **Тема:** Динамические структуры данных. Обработка деревьев.

1. **Цель работы:**

Ознакомиться с динамическими структурами данных, а именно с деревьями общего вида и упорядоченными бинарными деревьями на ЯП Си. Ознакомиться с реализацией базовых операция на соответствующими деревьями.

1. **Задание** (*вариант №*)**:**

Составить программу на языке Си для построения и обработка упорядоченного двоичного дерева, содержащего узлы типа double. Основные функции работы с деревьями реализовать в виде универсальных процедур или функций. После того, как дерево создано, его обработка должна производиться в режиме текстового меню со следующими действиями:

•добавление нового узла

•текстовая визуализация дерева

•удаление узла

•вычисление количества вершин двоичного дерева, имеющих ровно два поддерева.

1. **Оборудование** (лабораторное):

ЭВМ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ , процессор \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, имя узла сети \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ с ОП \_\_\_\_\_\_\_\_\_\_ Мб, НМД \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Мб. Терминал \_\_\_\_\_ адрес \_\_\_\_\_\_\_\_\_\_. Принтер \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Другие устройства \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Оборудование ПЭВМ студента, если использовалось:*

Процессор \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ с ОП \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ГБ, НМД \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ГБ. Монитор \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Другие устройства \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Программное обеспечение (лабораторное):**

Операционная система семейства \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, наименование \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ версия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

интерпретатор команд \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ версия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Система программирования \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ версия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Редактор текстов \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ версия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Утилиты операционной системы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Прикладные системы и программы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Местонахождение и имена файлов программ и данных \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Программное обеспечение ЭВМ студента, если использовалось:*

Операционная система семейства \_\_\_\_\_\_\_\_\_\_, наименование \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ версия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

интерпретатор команд \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ версия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Система программирования \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_версия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Редактор текстов \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ версия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Утилиты операционной системы, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Прикладные системы и программы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Местонахождение и имена файлов программ и данных на домашнем компьютере \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**6. Идея, метод, алгоритм** решения задачи(в формах:словесной,псевдокода,графической[блок-схема,диаграмма,рисунок,таблица] или формальные спецификации с пред- и постусловиями)

Реализуется обработка упорядоченного двоичного дерева, элементы которого имеют тип *double*. Для представления дерева используется динамическая структура – связанные узлы (*struct* *treeNode*), содержащие значение и указатели на левое и правое поддеревья.

**Основные функции:**

1. *add()* — добавление нового элемента в дерево с сохранением упорядоченности.
2. *removeNode()* — удаление узла по значению с корректной обработкой всех случаев (нет детей, один ребёнок, два ребёнка).
3. *printTree()* — текстовая визуализация структуры дерева с отступами, соответствующими глубине узла.
4. *countNodesWithTwoSubtrees()* — подсчёт числа узлов, у которых оба поддерева не пусты.
5. *contains()* — проверка наличия значения в дереве (для предотвращения дубликатов).
6. *destroy()* и *destroyRecursive()* — освобождение памяти одного узла или всего дерева соответственно.
7. *build()* — создание нового узла.
8. *createEmpty(),* *isEmpty()* — создание пустого дерева и проверка дерева на пустоту.

Для удобства управления программой реализовано текстовое меню (*textMenu(*)), предоставляющее пользователю интерфейс для работы с деревом.

**Метод**

Все основные операции реализованы как отдельные функции, принимающие указатель на корень дерева (*tree* *t*). Все изменения дерева возвращают новый указатель на корень, что предотвращает утечку памяти и обеспечивает корректную работу с динамически выделенными структурами.

1. Добавление *(add()) и удаление (removeNode())* реализованы рекурсивно, что упрощает обработку дерева и сокращает код.
2. Подсчёт узлов с двумя поддеревьями реализован также рекурсивно (*countNodesWithTwoSubtrees*()).
3. Визуализация структуры дерева (*printTree*()) осуществляется с помощью рекурсивного обхода, где глубина отражается количеством отступов перед значением.

**7. Сценарий выполнения работы** [план работы,первоначальный текст программы в черновике(можно на отдельном листе)итесты либо соображения по тестированию].

**1. Подготовка**

1. Определение структуры узла дерева struct *treeNode*, содержащей значение типа *double*, а также указатели на левое и правое поддерево.
2. Реализация функций для создания пустого дерева *createEmpty*, проверки дерева на пустоту *isEmpty*, создания нового узла *build*, получения значений и поддеревьев *getValue*, *getLeft*, *getRight*, а также освобождения памяти для одного узла *destroy* и для всего дерева *destroyRecursive*.
3. Написание вспомогательной функции для проверки наличия значения в дереве *contains*, предотвращающей дублирование элементов.

**2. Реализация**

1. Создание функции *добавления нового узла* — *add*. Функция рекурсивно ищет место для вставки и добавляет новый узел, если значение уникально.
2. Реализация функции *удаления узла* — *removeNode*. Функция корректно обрабатывает все возможные случаи: удаление листа, удаление узла с одним или двумя поддеревьями.
3. Оформление функции *текстовой визуализации дерева* — *printTree*. Функция выводит дерево в консоль с отступами, отражающими структуру.
4. Создание функции *подсчёта узлов с двумя поддеревьями* — *countNodesWithTwoSubtrees*. Функция рекурсивно обходится по дереву и считает нужные узлы.
5. Реализация основного пользовательского интерфейса — текстового меню *textMenu*, позволяющего работать с деревом интерактивно.

**3. Тестирование**

Тест 1: Пустое дерево

Описание: Программа запускается, дерево изначально пустое.

Ожидаемый результат: При попытке визуализации выводится "Дерево пустое", количество узлов с двумя поддеревьями — 0.

Тест 2: Последовательное добавление элементов

Действия: Вводятся значения `5`, `3`, `8`, `2`, `4`, `7`, `9`.

Структура дерева после добавления:

5

/ \

3 8

/ \ / \

2 4 7 9

Ожидаемый результат:

Визуализация показывает корректную структуру дерева.

Количество узлов с двумя поддеревьями *countNodesWithTwoSubtrees*: 3 (узлы 5, 3 и 8).

Тест 3: Удаление узла с одним поддеревом

Действия: Удаляется узел `3`.

Дерево после удаления 3:

5

/ \

4 8

/ / \

2 7 9

Ожидаемый результат:

Узел `3` удалён, его потомки корректно переподключены.

Структура дерева и количество узлов с двумя поддеревьями обновлены корректно.

Тест 4: Удаление листа

Действия: Удаляется листовой узел, например, `2`.

Дерево после удаления 2:

5

/ \

4 8

/ \

7 9

Ожидаемый результат: Дерево обновляется, удалён только нужный узел.

Тест 5: Попытка добавить уже существующий элемент

Действия: Попытка добавить значение `5` повторно.

Дерево после добавления уже имеющейся 5 (ничего не изменится):

5

/ \

4 8

/ \

7 9

Ожидаемый результат: Выводится сообщение об ошибке, структура дерева не изменяется.

Тест 6: Подсчёт количества узлов с двумя поддеревьями для различных конфигураций дерева

Действия: Последовательно добавляются и удаляются узлы, вызывается *countNodesWithTwoSubtrees* на каждом этапе.

Ожидаемый результат: Функция возвращает актуальное число узлов с двумя поддеревьями в дереве на каждом этапе.

**4. Отладка**

1. Проверка на корректное освобождение памяти при удалении дерева целиком *destroyRecursive*, а также при удалении отдельных узлов.
2. Отладка функций обработки деревьев, включая корректность обработки всех ветвлений при добавлении и удалении.
3. Проверка работы при граничных условиях (пустое дерево, дерево из одного узла, удаление всех узлов поочерёдно).

Пример работы программы (консольный сценарий):

Меню:

1. Добавить новый узел

2. Визуализировать дерево

3. Удалить узел

4. Подсчитать число узлов с двумя поддеревьями

5. Выйти

Введите ваш выбор: 1

Введите значение для добавления: 5

Узел добавлен.

*Пункты 1-7 отчета составляются строго до начала лабораторной работы.*

*Допущен к выполнению работы.*

**Подпись преподавателя**

**8. Распечатка протокола** (подклеить листинг окончательного варианта программы с тестовыми примерами,подписанныйпреподавателем)

**9. Дневник отладки** должен содержать дату и время сеансов отладки, и основные события(ошибки в сценарии и программе,нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

| № | Лаб. | Дата | Время | Событие | Действие по исправлению | Примечание |
| --- | --- | --- | --- | --- | --- | --- |
|  | или |  |  |  |  |  |
|  | дом. |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. **Замечания автора** по существу работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Выводы**

* В ходе выполнения лабораторной работы были изучены и реализованы основные операции для динамической структуры данных — упорядоченного двоичного дерева (бинарного дерева поиска) на языке программирования C.
* Разработаны функции для добавления, удаления, визуализации дерева, а также для вычисления количества узлов, имеющих ровно два поддерева. Особое внимание уделялось корректной обработке всех случаев при удалении узлов, включая ситуации с одним или двумя потомками.
* Структура бинарного дерева поиска позволяет поддерживать эффективный поиск, добавление и удаление элементов благодаря упорядоченности значений: в левом поддереве каждого узла располагаются элементы, меньшие значения данного узла, а в правом — большие.
* Текстовая визуализация дерева с отступами наглядно отражает его структуру, что облегчает отладку и анализ дерева.
* В ходе тестирования были проверены все основные сценарии работы: добавление новых элементов, удаление существующих (как листов, так и узлов с поддеревьями), а также предотвращение дублирования значений.
* Реализация рекурсивного подсчёта количества узлов с двумя поддеревьями позволила лучше понять структуру и логику обхода дерева.
* Практическая реализация двоичного дерева на языке C углубила понимание принципов работы с динамическими структурами данных, управления памятью, а также разработки модульных и универсальных функций для различных операций над деревьями.

Недочёты при выполнении задания могут быть устранены следующим образом: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_