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

Студент группы М8О-101БВ-24 Закревский Владислав Анатольевич, № по списку 10

Контакты e-mail

Работа выполнена: «16.05» 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:**Цель данной работы — закрепить и применить на практике знания по основам структур данных, в частности, построению и обработке деревьев с произвольным числом потомков (N-арные деревья), а также освоить навыки разработки интерактивных программ на языке Си, позволяющих пользователю динамически взаимодействовать с древовидной структурой: добавлять, удалять и искать элементы, визуализировать дерево, а также реализовать специальную функцию поиска нетерминальной вершины максимальной глубины. В рамках работы особое внимание уделяется не только разработке структур данных и алгоритмов работы с деревом, но и качеству пользовательского интерфейса, обработке ошибок и работе с памятью. Итогом работы является программа, наглядно демонстрирующая основные операции над деревьями и иллюстрирующая принцип их функционирования с помощью текстовой визуализации.
2. **Задание** (*вариант №***5**)**:**   
   Разработать программу на языке Си для построения и обработки упорядоченного бинарного дерева с данными типа float. Реализовать следующие операции:

* Добавление нового узла с учетом порядка бинарного дерева поиска.
* Текстовая визуализация дерева с отступами, пропорциональными глубине узлов.
* Удаление узла с сохранением целостности и порядка дерева.
* Определить значение нетерминальной вершины с максимальной глубиной (вариант 5).

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

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

В основе решения стоит реализация N-арного дерева посредством структуры с указателями на родителя, первого потомка и следующего брата, такая организация известна как "первый\_потомок — следующий\_брат" (first child – next sibling). Такой подход удобен, поскольку позволяет организовать список одноуровневых потомков через односвязный список, при этом навигация вверх по дереву также возможна (через указатель на родителя).

Структура узла дерева включает:

* значение (float), что обеспечивает универсальность;
* указатели на родителя, первого потомка и следующего брата.

Добавление узла производится так:

* Если дерево пусто, создаётся корневой элемент;
* Иначе — пользователь задаёт значение родительского узла, осуществляется поиск этого узла по значению. Если найден, создаётся новый узел и добавляется ко множеству потомков (либо как первый потомок, либо добавляется в конец списка братьев данного уровня).

Удаление узла организовано через функцию поиска узла по значению. Найденный узел отделяется от родителя и списка братьев, а затем вызывается рекурсивная очистка памяти для всего поддерева, если оно есть. Особое внимание уделяется корректности манипуляций с указателями, чтобы не нарушить структуру дерева и предотвратить утечки памяти.

Визуализация дерева реализована с помощью специальной функции, которая проходит дерево в ширину, используя рекурсию и массив булевых флагов для хранения информации о наличии потомков на предыдущих уровнях. Это позволяет корректно рисовать дерево псевдографическими символами ("│", "├──", "└──"), чтобы визуально отображать иерархию. Для повышения наглядности используются цветовые коды ANSI, выделяя корневой, нетерминальные и терминальные узлы разными цветами.

Поиск узла по значению реализуется обычным рекурсивным обходом дерева, начиная от корня.

Вычисление специальной функции (поиск глубоко расположенной нетерминальной вершины) организовано через рекурсивный обход всего дерева с передачей текущей глубины и структуры с результатом. При прохождении каждого узла, если у него есть потомки (то есть это нетерминальная вершина), выполняется проверка: если текущая глубина больше найденной максимальной, текущий узел сохраняется как лучший кандидат. При равенстве используется правило возврата самой первой найденной.

Пользовательское меню обеспечивает интуитивное управление и защищает от некорректного ввода, запрашивая подтверждения и обрабатывая возможные ошибки (например, поиск несуществующего родителя, неверный формат числа, попытка удаления из пустого дерева). Для повышения удобства реализована очистка экрана и запрос на нажатие клавиши Enter для продолжения между операциями.

Во всех операциях особое внимание уделено обработке ошибок, валидации входных данных, освобождению памяти и предотвращению утечек, что критически важно для программ на C. Исходный код написан так, чтобы быть максимально читаемым, прозрачным и легким для отладки и расширения.

Таким образом, вся система строится модульно: функции для создания, поиска, удаления, освобождения и визуализации узлов дерева могут быть легко переиспользованы или расширены. Алгоритмы максимально универсальны и не предполагают ограничений на количество потомков. Глубинный рекурсивный обход выбран из соображений читаемости кода и простоты реализации практически всех ключевых функций, кроме визуализации, где используется также некоторая логика "рисования" на лету.

Реализация полностью динамична: все изменения (добавление, удаление) происходят во время работы программы, а дерево хранится только в оперативной памяти. Таким образом, работа с деревом реализует все фундаментальные операции для таких структур, что и было требованием задания.

Методы тестирования также вкладываются в этот подход: корректность реализуется через ручные тесты — пошаговое создание дерева разной формы, повторные вызовы функции визуализации, удаление узлов (как листьев, так и веток), а также применение функции поиска глубокой нетерминальной вершины после различных модификаций структуры.

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

План работы:

* Анализ задачи и проектирование:

Определение структуры узла дерева (выбор оптимальной модели — первый потомок/следующий брат).

Построение основной логики обработки дерева: добавление, поиск, удаление, вывод.

* Реализация основных функций:

Написание функций создания (инициализации) узлов, их связывания, а также освобождения памяти.

Функция поиска по значению, необходимая для дальнейших операций.

* Организация пользовательского интерфейса:

Оформление главного меню;

Разработка логики обработки пользовательских команд и проверки на валидность.

* Реализация визуализации:

Проектирование алгоритма печати дерева с учетом вида иерархии;

Реализация цветового оформления для разных типов узлов.

* Имплементация функции поиска нетерминального узла максимальной глубины:

Продумывание алгоритма обхода, учета глубины, корректного возврата результата.

* Тестирование каждого этапа:

Проверка корректности операций добавления, отображения, удаления (в том числе листьев и внутреннего узла с потомками).

Проверка работы всех ветвей кода при ошибочном вводе.

Примерs сценария тестирования:

* Тест 1: Добавить корневой узел (например, 10). Визуализировать дерево. Ожидаемый результат — дерево из одного узла.
* Тест 2: Добавить к корню пару потомков: (20), (30). Визуализировать — увидеть три уровня: корень, два потомка.
* Тест 3: К одному из потомков добавить свои поддеревья: например, к 20 — 40, 50. К 30 — 70. После визуализации дерево должно быть уже четырёхуровневым.
* Тест 4: Добавить потомка к 50 — например, 90. Проверить результат.
* Тест 5: Попробовать удалить терминальный узел (например, 90). Проверить корректность удаления, а также отсутствие изменений в других узлах.
* Тест 6: Удалить нетерминальный узел (например, 20) — должно удалиться всё его поддерево (20, 40, 50, 90).
* Тест 7: Проверить то же самое после удаления корня — должно стереться всё дерево.
* Тест 8: Повторно создать древо с другой структурой, проверить функцию вычисления максимальной глубины нетерминального узла: ожидать корректный результат для различных конфигураций дерева, в том числе случая, когда дерево тривиально (все листья — результат = нет нетерминальных вершин).
* Тест 9: Проверить реакцию программы на некорректный ввод, в частности, попытку добавить узел к несуществующему родителю, попытку удалить несуществующий узел, попытку удалить из пустого дерева и так далее. Везде должны быть понятные сообщения об ошибке, программа не должна падать.
* Тест 10: Стабильность. После серии операций добавить-удалить, убедиться, что дерево корректно отображается и освобождается при завершении программы (можно проводить тесты утечек памяти с использованием инструментов типа valgrind в Unix-системах).

Замечания по тестированию:

Необходимо проверять не только "нормальный" сценарий, когда пользователь точно вводит ожидаемые значения, но и всевозможные крайние случаи и возможные ошибки со стороны пользователя (пустой ввод, неверный формат числа, удаление из пустого дерева и т.п.). Также важно при каждом добавлении/удалении дерева повторять визуализацию, чтобы видеть результат влияния операции на структуру дерева ― это частично заменяет unit-тесты. Кроме того, желательно тестировать ситуацию с глубоким деревом и большим количеством узлов для выявления возможных ошибок рекурсии и управления памятью.

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

*Допущен к выполнению работы.* **Подпись преподавателя**

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

1. **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. или дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  |  |  |  |  |  |  |

1. **Замечания автора** по существу работы

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

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

\_\_\_\_\_

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

1. **Выводы**

В результате работы была успешно реализована интерактивная программа с древовидной структурой на языке Си, поддерживающая полный спектр базовых операций с N-арным деревом: динамическое добавление, удаление узлов, визуализация структуры, а также вычисление нетривиальной функции поиска глубоко расположенного нетерминального узла. Программа продемонстрировала стабильную работу во множестве сценариев, в том числе при работе с пустым деревом и сложными иерархиями, при этом поддерживала интуитивно понятный пользовательский интерфейс и обработку всех типовых ошибок.

В процессе решения были закреплены ключевые понятия структур данных (деревья, рекурсия, работа с указателями и памятью, манипуляции со связанными структурами). Особое внимание уделялось безопасной работе с памятью — все выделенные ресурсы корректно освобождаются, что исключает появление утечек.

Программа получилась универсальной и легко расширяемой: при необходимости можно сохранить дерево в файл или дополнить набор операций. Особо стоит выделить удобство визуализации дерева, что значительно облегчает анализ структуры и поиск ошибок на всех этапах работы.

Работа способствует формированию прочных практических навыков разработки эффективных и надёжных алгоримов работы с древовидными структурами и их применению в более сложных прикладных задачах структурирования данных. Остаточное преимущество данной реализации — возможность легко интегрировать эти методы и принципы в более крупные или производственные проекты, требующие сложной обработки и хранения иерархических данных.

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

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

\_\_\_\_\_

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

\_\_\_\_\_

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

\_\_\_\_\_

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

\_\_\_\_\_

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

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