**Отчет по лабораторной работе № 23**

**по практикуму по циклу дисциплин "Информатика"**

Выполнил студент группы М8О-114БВ-24: Дробышев Егор Павлович, № по списку 9

Контакты e-mail: [tru.899@yandex.ru](mailto:tru.899@yandex.ru)

Работа выполнена: «10» марта 2025 г.

Преподаватель: доцент каф. 806 Никулин Сергей Петрович

Отчет сдан « » 2025г., итоговая оценка

Подпись преподавателя

1. **Тема:** «Динамические структуры данных. Обработка деревьев»
2. **Цель работы:** Составить программу на Си для построения и обработки дерева двоичного вида, содержащего узлы типа float.
3. **Задание**: Определить степень двоичного дерева (вариант – 27)

# Оборудование: оборудование ПЭВМ студента, если использовалось:

Процессор \_Intel Core i5\_ с ОП 8 Гб НМД 256 Гб. Монитор 1920x1080~60Hz. Другие устройства не использовались

# Программное обеспечение: программное обеспечение ЭВМ студента:

Операционная система семейства \_Linux\_, наименование \_Ubuntu\_ версия \_24.04

интерпретатор команд \_GNU bash\_ версия \_5.2.21(1)\_.

Редактор текстов emacs версия 29.3

Утилиты операционной системы: gcc, gdb

Местонахождение и имена файлов программ и данных на домашнем компьютере:

/home/tru\_\_

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

Программа выполняет 4 действия: добавление нового узла, визуализация дерева, удаление узла, вычисление степени дерева. Для этого необходимы следующие методы:

1. Создание узла (tree\_create\_el) – выделяет память, сохраняет data, обнуляет left и right.
2. Добавление корня (tree\_add\_root) – создаёт узел, если root == NULL.
3. Вставка узла (tree\_insert\_el) – рекурсивно находит место и вставляет узел.
4. Вывод (tree\_print) – рекурсивно обходит дерево, создает отступы.
5. Поиск минимума (find\_min) – идёт по левым узлам, пока не найдёт NULL.
6. Удаление узла (tree\_delete\_el) – если нет потомков, то просто удаляет; если один потомок, то заменяет узел потомком; если два потомка, то заменяет минимальным узлом из правого поддерева, затем удаляет его.
7. Определение степени (tree\_degree) – рекурсивно находит максимальное число детей (0, 1 или 2) среди всех узлов.
8. Поиск (tree\_find) – рекурсивно ищет значение, двигаясь влево или вправо.
9. **Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию].

#include <stdio.h>

#include <stdlib.h>

typedef struct Tree {

    float data;

    struct Tree\* left;

    struct Tree\* right;

} Tree;

Tree\* tree\_create\_el(float *value*) {

    Tree\* elem = (Tree\*)(malloc(sizeof(Tree)));

    elem->data = *value*;

    elem->left = NULL;

    elem->right = NULL;

    return elem;

}

Tree\* tree\_add\_root(Tree\* *root*, float *value*) {

    if (*root* == NULL) {

        return tree\_create\_el(*value*);

    }

    return *root*;

}

void tree\_insert\_el(Tree\* *root*, Tree\* *elem*) {

    if (*elem*->data < *root*->data) {

        if (*root*->left == NULL) {

*root*->left = *elem*;

        } else {

            tree\_insert\_el(*root*->left, *elem*);

        }

    } else {

        if (*root*->right == NULL) {

*root*->right = *elem*;

        } else {

        tree\_insert\_el(*root*->right, *elem*);

        }

    }

}

void tree\_print(Tree\* *root*, int *tabs*) {

    if (*root* == NULL) {

        return;

    }

    tree\_print(*root*->right, *tabs*+1);

    for (int i = 0; i < *tabs*; i++) {

        printf("\t");

    }

    printf("%.2f\n", *root*->data);

    tree\_print(*root*->left, *tabs*+1);

    }

Tree\* find\_min(Tree\* *root*) {

    while (*root*->left != NULL) {

*root* = *root*->left;

    }

    return *root*;

}

Tree\* tree\_delete\_el(Tree\* *root*, float *value*) {

    if (*root* == NULL) {

        return NULL;

    }

    if (*value* < *root*->data) {

*root*->left = tree\_delete\_el(*root*->left, *value*);

    }

    else if (*value* > *root*->data) {

*root*->right = tree\_delete\_el(*root*->right, *value*);

    }

    else {

        if (*root*->left == NULL) {

            Tree\* temp = *root*->right;

            free(*root*);

            return temp;

        }

        else if (*root*->right == NULL) {

            Tree\* temp = *root*->left;

            free(*root*);

            return temp;

        }

        else {

            Tree\* temp = find\_min(*root*->right);

*root*->data = temp->data;

*root*->right = tree\_delete\_el(*root*->right, temp->data);

        }

    }

    return *root*;

}

int tree\_degree(Tree\* *root*) {

    if (*root* == NULL) {

        return 0;

    }

    int degree = 0;

    if (*root*->left != NULL) {

        degree++;

    }

    if (*root*->right != NULL) {

        degree++;

    }

    int left\_degree = tree\_degree(*root*->left);

    int right\_degree = tree\_degree(*root*->right);

    int max\_subtree\_degree = 0;

    if (left\_degree > right\_degree) {

        max\_subtree\_degree = left\_degree;

    } else {

        max\_subtree\_degree = right\_degree;

    }

    if (max\_subtree\_degree > degree) {

        degree = max\_subtree\_degree;

        return degree;

    } else {

        return degree;

    }

}

int tree\_find(Tree\* *root*, float *value*) {

    if (*root* == NULL) {

        return 0;

    }

    if (*root*->data == *value*) {

        return 1;

    }

    if (*value* < *root*->data) {

        return tree\_find(*root*->left, *value*);

    } else {

        return tree\_find(*root*->right, *value*);

    }

}

void menu() {

    printf("Выберите действие\n");

    printf("1. Добавить новый узел\n");

    printf("2. Вывод дерева\n");

    printf("3. Удаление узла\n");

    printf("4. Определить степень двоичного дерева\n");

    printf("5. Выход\n");

    printf("\n");

}

int main() {

    Tree\* elem = NULL;

    float value;

    int mode = -1;

    while(mode) {

        menu();

        value = 0;

        scanf("%d", &mode);

        switch (mode) {

            case 1:

                printf("Введите значение: ");

                scanf("%f", &*value*);

                if (elem == NULL) {

                    elem = tree\_add\_root(elem, value);

                } else {

                    tree\_insert\_el(elem, tree\_create\_el(value));

                }

                break;

            case 2:

                printf("Дерево:\n");

                printf("\n");

                tree\_print(elem, 0);

                break;

            case 3:

                printf("Введите значение: ");

                scanf("%f", &*value*);

                if (!tree\_find(elem, value)) {

                    printf("Узел с таким значением не найден\n");

                } else {

                    elem = tree\_delete\_el(elem, value);

                    printf("Узел %.2f удалён\n", value);

                }

                break;

            case 4:

                printf("Степень дерева = %d\n", tree\_degree(elem));

                break;

            case 5:

                printf("Вы вышли из программы\n");

                mode = 0;

                break;

            default:

                printf("error\n");

                break;

        }

        printf("\n");

    }

}

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

PS C:\c proj> gcc derevo23.c -o derevo23

PS C:\c proj> ./derevo23

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

1

Введите значение: 4

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

1

Введите значение: 2.5

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

1

Введите значение: 5.8

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

1

Введите значение: 6.12

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

1

Введите значение: 3.14

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

2

Дерево:

6.12

5.80

4.00

3.14

2.50

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

1

Введите значение: 1

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

1

Введите значение: 7.77

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

2

Дерево:

7.77

6.12

5.80

4.00

3.14

2.50

1.00

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

4

Степень дерева = 2

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

3

Введите значение: 7.77

Узел 7.77 удалён

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

2

Дерево:

6.12

5.80

4.00

3.14

2.50

1.00

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

3

Введите значение: 2.5

Узел 2.50 удалён

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

2

Дерево:

6.12

5.80

4.00

3.14

1.00

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

3

Введите значение: 9

Узел с таким значением не найден

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

3

Введите значение: 3.14

Узел 3.14 удалён

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

3

Введите значение: 1

Узел 1.00 удалён

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

2

Дерево:

6.12

5.80

4.00

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

4

Степень дерева = 1

Выберите действие

1. Добавить новый узел

2. Вывод дерева

3. Удаление узла

4. Определить степень двоичного дерева

5. Выход

5

Вы вышли из программы

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 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

# Замечания автора по существу работы

1. **Выводы:**

В ходе выполнения лабораторной работы научился строить и обрабатывать двоичные деревья на Си. Недочёты при выполнении задания могут быть устранены следующим образом:

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