Отчет по лабораторной работе № 23 по курсу Практикум на ЭВМ

Студент группы М8О-104Б-22 Алхимова Дарья Игоревна, № по списку 02

Контакты www, e-mail, icq, skype fl81m@yandex.ru

Работа выполнена: « 1 » января 2022 г.

Преподаватель: асп. каф. 806 Потенко М.А.

Входной контроль знаний с оценкой _____

Отчет сдан « » _____ 202 __ г., итоговая оценка ___

Подпись преподавателя _____

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

2. **Цель работы:** Составить программу на языке Си для построения и обработки дерева общего вида или упорядоченного двоичного дерева, содержащего узлы типа float, int, char или enum.

3. **Задание** (*вариант № 7*): Проверить, находятся ли во всех листьях двоичного дерева элементы со значениями в заданном диапазоне.

4. **Оборудование** (*лабораторное*):
ЭВМ _____, процессор _____, имя узла сети _____с ОП _____Мб,
НМД _____Мб. Терминал _____адрес _____. Принтер _____
Другие устройства _____

Оборудование ПЭВМ студента, если использовалось:
Процессор _____Intel Core i5 _____с ОП _____8 _____Гб, НМД _____2097152 _____Мб. Монитор _____
Другие устройства _____

5. **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 __macOS __, наименование __macOS Catalina__версия __10.15.4__
интерпретатор команд _____cmd_____версия _____10.0.19044.2130_____
Система программирования _____версия _____
Редактор текстов _____nano _____версия _____2.0.6 _____
Утилиты операционной системы _____
Прикладные системы и программы _____
Местонахождение и имена файлов программ и данных на домашнем компьютере _____

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

Для реализаци дерева будем использовать файлы main.c (основной файл), tree.c (файл с функциями) и tree.h (заголовочный файл).
В файле tree.h определим структуру двоичного дерева, которая будет содержать значение узла и указатели на левое и правое поддерево.
В файле tree.c реализуем функции для создания, удаления и печати дерева, вставки, удаления и поиска узлов в дереве, а также функцию, которая будет проверять, находятся ли все элементы в листьях дерева в заданном диапазоне.
В файле main.c создадим меню, которое позволит пользователю выбрать действие (вставить, удалить узел, распечатать дерево, проверить, находятся ли все элементы в листьях в заданном диапазоне, завершить работу программы).
Также создадим файл Makefile, при помощи которого будем компилировать нашу программу.

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

Файл main.c:
В начале программы объявляются основные переменные и инициализируется корень дерева (root). Затем начинается бесконечный цикл while, в котором выводится меню с выбором действий: добавление элемента (вершины), удаление вершины, печать всего дерева, проверка, находятся ли во всех листьях элементы со значениями в заданном диапазоне и завершение работы программы.
В зависимости от выбранного пункта меню (choice), выполняется соответствующее действие (case).
- case 1: Добавление элемента. Запрашивается значение элемента и вызывается функция insert_node() для добавления нового элемента в дерево.
- case 2: Удаление элемента. Сначала проверяется, не пустое ли дерево. Если нет, то запрашивается значение элемента и вызывается функция remove_node() для удаления элемента из дерева.
- case 3: Печать дерева. Сначала проверяется, не пустое ли дерево. Если нет, то вызывается функция print_tree() для печати элементов дерева.
- case 4: Проверка, находятся ли во всех листьях элементы со значениями в заданном диапазоне. Сначала проверяется, не пустое ли дерево. Если нет, то запрашиваются границы диапазона и вызывается функция is_leaf_in_range().
- case 5: Завершение работы программы. Вызывается функция free_tree() для освобождения памяти, затем программа завершается.
- Если введено некорректное значение, то выводится сообщение об ошибке. - В конце функции main() еще раз вызывается функция free_tree() для освобождения памяти, затем функция завершается.
Файл tree.h:
В начале файла подключаются все необходимые библиотеки. После объявляется структура 'Tree', которая содержит значение узла (int value) и указатели на левое и правое поддерево (struct Tree* left, struct Tree* right). Затем объявляются функции для работы с деревом, которые будут реализованы в файле tree.c.

Файл tree.c:
- Tree* create_tree(int value) создает корень дерева с заданным значением и возвращает указатель на него.С помощью функции malloc выделяется память под новый узел дерева. У нового дерева нет дочерних узлов, поэтому указатели на левый и правый дочерние узлы устанавливаются NULL.
- Tree* insert_node(Tree* root, int value) добавляет новый узел в дерево. Если дерево пусто (корень равен NULL), она создает новое дерево. Если новое значение меньше значения корня, оно должно быть добавлено в левое поддерево. Если новое значение больше значения корня, оно должно быть добавлено в правое поддерево.
- Tree* remove_node(Tree* root, int key) удаляет узел из дерева. Если узел, который нужно удалить, имеет только одного потомка, то этот потомок заменяет удаляемый узел. Если у удаляемого узла есть оба потомка, то его заменяет узел с минимальным значением из правого поддерева.
- Tree* find_minimum(Tree* root) находит узел с минимальным значением в дереве. Функция перемещается влево по дереву до тех пор, пока не достигнет узла, у которого нет левого дочернего узла.
- int is_leaf_in_range(Tree* root, int min, int max) проверяет, есть ли в дереве лист в заданном диапазоне. Если значение листа находится в заданном диапазоне, функция возвращает 1 (истина), в противном случае - 0 (ложь). Функция рекурсивно проверяет все листья в дереве.
- void print_tree(Tree* root, int n) рекурсивно выводит значения узлов дерева, начиная с самого правого узла и идя до самого левого. Параметр n используется для определения количества табуляций перед выводом значения узла.
- void free_tree(Tree* root) освобождает память, выделенную под дерево, рекурсивно обходя все узлы дерева и освобождая память каждого узла.

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, подписанный преподавателем)

```c
#include <stdio.h>
#include <stdlib.h>
#include "tree.h"

int main() {
    Tree* root = NULL;
    int value;
    int min;
    int max;
    int choice = 0;

    while (1) {
        printf("1. Добавить вершину\n");
        printf("2. Удалить вершину\n");
        printf("3. Распечатать дерево\n");
        printf("4. Проверить, находятся ли во всех листьях элементы со значениями в заданном диапазоне\n");
        printf("5. Завершить работу программы\n");

        printf("Введите номер команды: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Введите значение элемента: ");
                scanf("%d", &value);
                root = insert_node(root, value);
                printf("Вершина добавлена\n");
                break;
            case 2:
                if (root == NULL) {
                    printf("Дерево итак пустое\n");
                    break;
                }
                printf("Введите значение элемента для удаления: ");
                scanf("%d", &value);
                root = remove_node(root, value);
                printf("Вершина удалена\n");
                break;
            case 3:
                if (root == NULL) {
                    printf("~Пустое дерево~\n");
                    break;
                }
                printf("\n\n");
                print_tree(root, 1);
                printf("\n\n");
                break;
            case 4:
                if (root == NULL) {
                    printf("Дерево пустое. Создайте дерево\n");
                    break;
                }
                printf("Введите границы диапазона: ");
                scanf("%d%d", &min, &max);
                if (is_leaf_in_range(root, min, max)) {
                    printf("Да.\n");
                } else {
                    printf("Нет.\n");
                }
                break;
            case 5:
                free_tree(root);
                printf("Работа программы завершена\n");
                return 0;
                break;
            default:
                printf("Ошибка: неправильный ввод. Попробуйте еще раз.\n");
                break;
        }
    }
    free_tree(root);

    return 0;
}
```

```c
#include "tree.h"

#include <stdio.h>

//Функция создания дерева
Tree* create_tree(int value) {
    Tree* new_Tree = (Tree*)malloc(sizeof(Tree));
    new_Tree->value = value;
    new_Tree->left = NULL;
    new_Tree->right = NULL;
    return new_Tree;
}

//Функция добавления новой вершины
Tree* insert_node(Tree* root, int value) {
    if(root == NULL)
        return create_tree(value);
        if(value < root->value)
            root->left = insert_node(root->left, value);
        else if(value > root->value)
            root->right = insert_node(root->right, value);
    return root;
}

//Функция поиска минимума
Tree* find_minimum(Tree* root) {
    while (root->left != NULL) {
        root = root->left;
    }
    return root;
}

//Функция удаления вершины
Tree* remove_node(Tree* root, int value) {
    if (root == NULL) {
        return root;
    }
    if (value < root->value) {
        root->left = remove_node(root->left, value);
    } else if (value > root->value) {
        root->right = remove_node(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;
        }
        Tree* temp = find_minimum(root->right);
        root->value = temp->value;
        root->right = remove_node(root->right, temp->value);
    }
    return root;
```

```c
//Функция вывода дерева
void print_tree(Tree* root, int n) {
    if (root == NULL) {
        return;
    }
    print_tree(root->right, n + 1);
    for (int i = 0; i < n; i++) printf("\t");
    printf("%d\n", root->value);
    print_tree(root->left, n + 1);
}

//Проверка, находится ли значение листа в диапазоне
int is_leaf_in_range(Tree* root, int min, int max)
{
    if (root == NULL)
        return 1;

    if (root->left == NULL && root->right == NULL)
        return (root->value >= min && root->value <= max);

    return is_leaf_in_range(root->left, min, max) && is_leaf_in_range(root->right, min, max);
}

//Функция удаления дерева
void free_tree(Tree* root) {
    if(root == NULL) return;
    free_tree(root->left);
    free_tree(root->right);
    free(root);
}
```

```c
#ifndef __TREE_H__
#define __TREE_H__
#include <stdlib.h>

typedef struct Tree {
    int value;
    struct Tree* left;
    struct Tree* right;
} Tree;

Tree* create_tree(int value);
Tree* insert_node(Tree* root, int value);
Tree* remove_node(Tree* root, int key);
Tree* find_minimum(Tree* root);
int is_leaf_in_range(Tree* root, int min, int max);
void print_tree(Tree* root, int n);
void free_tree(Tree* root);

#endif // __TREE_H__
```

```
CC = gcc
CFLAGS = -std=c99 -Wall -Werror
DEBUG = gdb
FILEOUT = tree.out
SOURCES = tree.c main.c

all:
	$(CC) $(CFLAGS) $(SOURCES)  -o $(FILEOUT)
clean:
	rm -f *.out
debug:
	$(CC) $(CFLAGS) -g $(SOURCES)
	sudo $(DEBUG) $(FILEOUT)
cmp:
	$(CC) $(CFLAGS) $(SOURCES) -o $(FILEOUT)
run:
	./$(FILEOUT)
```

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 ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

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

10. **Замечания автора** по существу работы: _____

_____

_____

11. **Выводы:** В ходе выполнения лабораторной работы я реализовала основные операции, связанные с обработкой деревьев: создание нового узла, вставка узла в дерево, удаление узла из дерева, а также вывод элементов дерева. В процессе работы над программой я также отработала навыки работы с динамическими структурами данных, научилась компилировать программы, состоящие из нескольких файлов, при помощи Makefile. Динамические структуры данных позволяют хранить и манипулировать данными в более гибкой и эффективной манере, чем это позволяют статические структуры данных. Они позволяют программистам эффективно управлять памятью и способствуют оптимальному использованию ресурсов, поэтому могут быть полезны при проектировании и реализации алгоритмов, которые требуют быстрого и эффективного доступа к данным, в области разработки программного обеспечения, а также в области искусственного интеллекта и машинного обучения. Деревья используются для представления иерархических структур, организации данных для быстрого поиска и сортировки, и во многих алгоритмах и структурах данных.

Недочёты при выполнении задания могут быть устранены следующим образом: _____

_____

Подпись студента _____