Отчет по лабораторной работе № 23 по курсу “Языки и методы программирования”

Студент группы М80-103Б-21 Зверева Елизавета Леонидовна, № по списку 11

e-mail: elizavetka.zvereva.2003@mail.ru , telegram: @banshee

Работа выполнена: «» сентября 2021г.

Преподаватель: каф. 806 Севастьянов Виктор Сергеевич

Отчет сдан « » _____20__ г., итоговая оценка _____

Подпись преподавателя _____

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

2. **Цель работы:** Научиться создавать и работать с деревом.

3. **Задание (вариант 11):** проверить монотонность убывания ширины дерева.

4. **Оборудование** (студента):

Процессор *AMD A9-9420 RADEON R5, 5 COMPUTE CORES 2C+3G 3.00 GHz* с ОП *8* Гб, НМД *512* Гб. Монитор *1920x1080*

5. **Программное обеспечение (**студента**):**

Операционная система семейства: *linux*, наименование: *ubuntu*, версия *20.04 focal*
интерпретатор команд: *bash* версия *5.0.17*
Редактор текстов *emacs* версия *3.24.14*

6.Идея, метод, алгоритм решения задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями)
Каждая задача (добавление нового узла, текстовая визуализация, удаление узла, вычисление количества листьев) реализуется с помощью отдельных функций, каждая из которых допускает рекурсию и работает со структурным типом, который мы создали в начале для представления дерева в памяти устройства с помощью языка Си.

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

1) чтобы реализовать создание дерева,напишем функцию,которая выделяет память
2)удаление всего дерева
3)удаление какого-то определенного узла
4)вывод деревьев
5)проверка на монотонность убывания

Для получения помощи в использовании программы напишите 'h'.

h

Команды 'insert' и 'ins', если дерево не создано -- создают дерево, если создано -- добавляют вершины в дерево.

Команды 'delete num' и 'del num' удаляют вершину и всех ее детей.

Команды 'print' и 'p' печатают вершины дерева.

Команды 'run' и 'r' проверяют монотонность убывания ширины уровня дерева.

Команды 'q' и 'exit' заканчивают работу программы.

Команды 'destroy' и 'des' удаляют все дерево.

8. Распечатка протокола
tree.h

```
#ifndef _TREE_H_
```

```
#define _THEE_H_

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef int TreeItem;
typedef struct _tree *Tree;

Tree tree_create(TreeItem value);
void tree_add_node(Tree tree, TreeItem parent, TreeItem value);
void tree_print(Tree tree);
void tree_print_node(Tree tree, int indent);
void tree_destroy(Tree tree);
void tree_del_node(Tree tree, TreeItem value);
Tree tree_find(Tree tree, TreeItem c);

int max_level(Tree tree, int deep);
void counting_nodes_on_the_lvls(Tree tree, int level, int *mat);
void check_monotonicity_of_decreaset(Tree tree);

#endif
```

tree.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#include "tree.h"

#define DEEP 0
#define max(x,y) ((x) > (y) ? (x) : (y))


struct _tree {
    TreeItem value;
    Tree child;
    Tree sibling;
};

Tree tree_create(TreeItem value)
{
    Tree tree = (Tree) malloc(sizeof(*tree));
    if(!tree) {
        fprintf(stderr, "%s\n", "Error: no memory");
        exit(1);
    }
    tree->value = value;
    tree->child = NULL;
    tree->sibling = NULL;

    return tree;
}

void tree_add_node(Tree tree, TreeItem parent, TreeItem value)
{
    Tree parent_node = tree_find(tree, parent);
```

```
    if(parent_node) {
        if(!parent_node->child) {
            parent_node->child = tree_create(value);
        } else {
            parent_node = parent_node->child;
            while(parent_node->sibling) {
                parent_node = parent_node->sibling;
            }
            parent_node->sibling = tree_create(value);
        }
    } else {
        printf("Указанная родительская вершина не найдена.\n");
    }
}

Tree tree_find(Tree tree, TreeItem c)
{
    if(!tree) {
        return NULL;
    }

    if(tree->value == c) {
        return tree;
    }

    Tree result = NULL;
    if(tree->child) {
        result = tree_find(tree->child, c);
        if(result) return result;
    }

    if(tree->sibling) {
        result = tree_find(tree->sibling, c);
        if(result) return result;
    }

}


void tree_print_node(Tree tree, int indent)
{
    for(int i = 0; i < indent; ++i) {
        printf("\t");
    }
    printf("%d\n", tree->value);
    if(tree->child) {
        tree_print_node(tree->child, indent + 1);
    }
    if(tree->sibling) {
        tree_print_node(tree->sibling, indent);
    }
}

void tree_print(Tree tree)
{
    tree_print_node(tree, 0);
}
```

```
void tree_destroy(Tree tree)
{

    if(tree->child) {
        tree_destroy(tree->child);
    }
    if(tree->sibling) {
        tree_destroy(tree->sibling);
    }
    free(tree);
    tree = NULL;
}


void tree_del_node(Tree tree, TreeItem value)
{
    if(tree->child) {
        if(tree->child->value == value) {
            Tree tmp = tree->child;
            tree->child = tree->child->sibling;
            if (tmp->child) {
                tree_destroy(tmp->child);
            }
            free(tmp);
            tmp = NULL;
            return;
        } else {
            tree_del_node(tree->child, value);
        }
    }


    if(tree->sibling) {
        if(tree->sibling->value == value) {
            Tree tmp = tree->sibling;
            tree->sibling = tree->sibling->sibling;
            if(tmp->child) {
                tree_destroy(tmp->child);
            }
            free(tmp);
            tmp = NULL;
            return;
        } else {
            tree_del_node(tree->sibling, value);
        }
    }
}

int max_level(Tree tree, int deep)
{
    if(!tree) return deep - 1;
    return max(max_level(tree->child, deep + 1), max_level(tree->sibling,
deep));
}

void couting_nodes_on_the_lvls(Tree tree, int level, int *mat)
```

```
{
    mat[level] += 1;
    if(tree->child) {
        couting_nodes_on_the_lvls(tree->child, level + 1, mat);
    }
    if(tree->sibling) {
        couting_nodes_on_the_lvls(tree->sibling, level, mat);
    }
}

void check_monotonicity_of_decreaset(Tree tree)
{
    bool more_one_lvl = false;
    bool decreasing = true;
    if(tree->child == NULL) {
        printf("Дерево содержит только корень, этого недостаточно для
определения монотонности.\n");
    } else {
        if(tree->child->child) {
            more_one_lvl = true;
        }
        for(Tree tmp = tree->child; tmp->sibling; tmp = tmp->sibling) {
            if(tmp->child != NULL) {
                more_one_lvl = true;
                break;
            }
        }
        if(more_one_lvl) {
            int deep = max_level(tree, DEEP);
            int level = 0;
            int mat[deep];
            for(int i = 0; i <= deep; ++i) {
                mat[i] = 0;
            }
            couting_nodes_on_the_lvls(tree, level, mat);
            for(int i = 1; i < deep; ++i) {
                if(mat[i] <= mat[i + 1]) {
                    printf("Дерево не убывает.\n");
                    decreasing = false;
                    break;
                }
            }
            if(decreasing) printf("Дерево убывает.\n");
        } else printf("Дерево содержит только 1 уровень, этого недостаточно для
определения монотонности\n");
    }
}

main.c
int main(void)
{
    char s[8];

    Tree tree = NULL;
    int root = 0, ver = 0, parent = 0;
```

```
    printf("\nДля получения помощи в использовании программы напишите 'h'.\n\
n");
    while (1) {
        scanf("%7s", s);
        if (!strcmp(s, "insert") || !strcmp(s, "ins")) {
            if(!tree) {
                printf("Введите значение корня дерева:\n");
                scanf("%d", &root);
                tree = tree_create(root);
            }
            while (scanf("%d%d", &parent, &ver)) {
                tree_add_node(tree, parent, ver);
            }
        } else if (!strcmp(s, "delete") || !strcmp(s, "del")) {
            if(!tree) printf("Дерева не существует.\n");
            else {
                scanf("%d", &ver);
                tree_del_node(tree, ver);
            }
        } else if (!strcmp(s, "exit") || !strcmp(s, "q")) {
            if (tree) tree_destroy(tree);
            break;
        } else if (!strcmp(s, "run") || !strcmp(s, "r")) {
            if(!tree) printf("Дерева не существует.\n");
            else {
                check_monotonicity_of_decrease(tree);
            }
        } else if (!strcmp(s, "print") || !strcmp(s, "p")) {
            if (!tree) printf("Дерева не существует.\n");
            else {
                printf("\n\n");
                tree_print(tree);
                printf("\n\n");
            }
        } else if (!strcmp(s, "destroy") || !strcmp(s, "des")) {
            if (!tree) printf("Дерева не существует.\n");
            else {
                tree_destroy(tree);
                tree = NULL;
            }
        } else if (!strcmp(s, "h")) {
            printf("\n\nКоманды 'insert' и 'ins', если дерево не создано --
создают дерево, если создано -- добавляют вершины в дерево.\n\n");
            printf("Команды 'delete num' и 'del num' удаляют вершину и всех ее
детей.\n\n");
            printf("Команды 'print' и 'p' печатают вершины дерева.\n\n");
            printf("Команды 'run' и 'r' проверяют монотонность убывания ширины
уровня дерева.\n\n");
            printf("Команды 'q' и 'exit' заканчивают работу программы.\n\n");
            printf("Команды 'destroy' и 'des' удаляют все дерево.\n\n");
        } else {
            printf("\n\nТакой команды не существует, воспользуйтесь командами
'h'. \n\n");
        }
    }
    return 0;
}
```

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. или дом.	Дата	Время	Событие	Действие по исправлению	Примечание
1	-			-	-	-

10. Замечания автора. Нет.

Выводы. Результатом выполнения работы стало глубокое изучение работы с памятью в Си, использование деревьев. Реализация оказалась не такой уж простой задачей.

Подпись студента _____