Отчет по лабораторной работе № 23 по курсу “Фундаментальная информатика”

Студент группы М80-101Б-21 Ершова Станислава Григорьевича, № по списку 8

Контакты e-mail, telegram: stas.ershov57@gmail.com , @stas_orel

Работа выполнена: «24» марта 2022г.

Преподаватель: каф. 806 Севастьянов Виктор Сергеевич

Отчет сдан «24» марта 2021 г., итоговая оценка _____

Подпись преподавателя _____

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

2. **Цель работы: Научиться работать с динамическими структурами данных и деревьями.**

3. **Задание** (*вариант №* **22): Определить число вершин дерева.**

4. **Оборудование** (студента):
Процессор *Intel Pentium N4200 1.1 ГГц* с ОП *8* Гб, SSD *128* Гб. Монитор *1920x1080*

5. **Программное обеспечение (**студента**):**
Операционная система семейства: *linux*, наименование: *ubuntu*, версия *20.04.3 LTS*
интерпретатор команд: *bash* версия *5.0.17*
Система программирования -- версия --**,** редактор текстов *nano* версия *4.8*
Утилиты операционной системы --
Прикладные системы и программы --
Местонахождение и имена файлов программ и данных на домашнем компьютере --

6. Идея, метод, алгоритм:

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

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

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <stdbool.h>


typedef struct _node {
    struct _node* parent;
    int value;
    int current_cnt;
    int available_cnt;
    struct _node** children;
} node;


int STANDART_ARRAY_SIZE = 5;
int vertex_cnt = 0;
node* genesis_node;
bool* borders;


int max(int a, int b) {
    if (a > b)
        return a;
    return b;
}


void create_node(node* u, int v) {
```

```
    node* nd = malloc(sizeof(node));
    nd->parent = u;
    nd->value = v;
    nd->current_cnt = 0;
    nd->available_cnt = STANDART_ARRAY_SIZE;
    nd->children = malloc(sizeof(node) * STANDART_ARRAY_SIZE);
    if(u->current_cnt == u->available_cnt - 1) {
        u->children = realloc(u->children, sizeof(node) * u->available_cnt * 2);
        u->available_cnt *= 2;
    }
    u->children[u->current_cnt] = nd;
    u->current_cnt++;
}


node* create_genesis_node(int v) {
    node* nd = malloc(sizeof(node));
    nd->parent = NULL;
    nd->value = v;
    nd->current_cnt = 0;
    nd->available_cnt = STANDART_ARRAY_SIZE;
    nd->children = malloc(sizeof(node) * STANDART_ARRAY_SIZE);
    return nd;
}


void delete_node(node* v) {
    for (int i = v->current_cnt - 1; i >= 0; --i) {
        delete_node(v->children[i]);
    }
    node* v_parent = v->parent;
    int del_index = 0;
    for (int i = 0; i < v_parent->current_cnt; ++i) {
        if (v_parent->children[i]->value == v->value) {
            del_index = i;
            vertex_cnt--;
            free(v);
            break;
        }
    }
    for (int i = del_index + 1; i < v_parent->current_cnt; ++i) {
        v_parent->children[i - 1] = v_parent->children[i];
    }
    v_parent->current_cnt--;
}


node* find_vertex(node* u, int vertex) {
    if (u->value == vertex)
        return u;
    node* vertex_address = NULL;
    for (int i = 0; i < u->current_cnt; ++i) {
        vertex_address = find_vertex(u->children[i], vertex);
        if (vertex_address != NULL && vertex_address->value == vertex)
            break;
    }
    return vertex_address;
}


int str_to_int(char* s) {
    int ans = 0;
    for (int i = 0; i < strlen(s); ++i) {
        if (s[i] >= '0' && s[i] <= '9') {
            ans = ans * 10 + (s[i] - '0');
        } else {
            return -2022;
        }
    }
    return ans;
}
```

```
void print_tree(node* u, int space_cnt) {
    printf("%d\n", u->value);

    if (sizeof(borders) / sizeof(bool) - 1 < space_cnt)
        borders = realloc(borders, sizeof(borders) * 2);
    borders[space_cnt] = true;

    for (int i = 0; i < u->current_cnt; ++i) {

        if (i == u->current_cnt - 1)
            borders[space_cnt] = false;

        for (int j = 0; j < space_cnt; ++j) {
            if (borders[j])
                printf("|");
            else
                printf(" ");
        }

        printf("|\n");
        for (int j = 0; j < space_cnt; ++j) {
            if (borders[j])
                printf("|");
            else
                printf(" ");
        }

        printf("+");
        printf("----");

        print_tree(u->children[i], space_cnt + 5);
    }

    borders[space_cnt] = false;
}


void print_info(bool first_print) {
    if (first_print) {
        printf("    Доступные операции:\n");
        printf("0 - Вывести справку\n");
        printf("1 - Добавить ребро в дерево\n");
        printf("2 - Удалить ребро из дерева\n");
        printf("3 - Вывести дерево\n");
        printf("4 - Вывести количество вершин в дереве\n");
    } else {
        printf("Сначала в отдельной строке вводится номер операции 0-4\n");
        printf("После чело для операции 1 - 2 вершины,\n");
        printf("          для операции 2 - 1 вершина,\n");
        printf("          для операций 3 и 4 дополнительные входные данный не требуются.\n");
    }
}


int main() {
    borders = malloc(sizeof(bool) * 30);
    bool is_genesis = true;
    char str[100];
    print_info(true);
    while (true) {
        printf("Операция: ");
        scanf("%s", str);
        int op = str_to_int(str);
        if (op < 0 || op > 4) {
            printf("Введенные данные некорректны\n");
            continue;
        }

        if(op == 0) {
            print_info(false);
        } else if (op == 1) {
            int u, v;
            char u1[100], v1[100];
```

```
            scanf("%s %s", u1, v1);
            u = str_to_int(u1);
            v = str_to_int(v1);
            if (u < 0 || v < 0) {
                printf("Введены некоректные данные, допустимы только целочесленные неотрицательные значения
вершин\n");
                continue;
            }
            if (is_genesis) {
                is_genesis = false;
                genesis_node = create_genesis_node(u);
                create_node(genesis_node, v);
                vertex_cnt += 2;
            } else {
                node* u_address = find_vertex(genesis_node, u);
                if (u_address == NULL) {
                    printf("Вершина %d отсутсвует в дереве\n", u);
                    continue;
                }
                node* v_address = find_vertex(genesis_node, v);
                if (v_address == NULL) {
                    create_node(u_address, v);
                    vertex_cnt++;
                } else {
                    printf("Такая вершина уже есть в дереве\n");
                }
            }
        } else if (op == 2) {
            char u_str[100];
            int u;
            scanf("%s", u_str);
            u = str_to_int(u_str);
            node* v = find_vertex(genesis_node, u);
            if (v == NULL)
                printf("Введены некорректные данные\n");
            else
                delete_node(v);
        } else if (op == 3) {
            print_tree(genesis_node, 0);
        } else if (op == 4) {
            printf("Вершин в дереве: %d\n", vertex_cnt);
        }
    }
}
```

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

10. **Замечания автора** по существу работы

11. **Выводы: Разобрался как работает динамическая память, надеюсь в будущем буду по минимуму с этим связываться, не понравилось. Научился делать крутой вывод дерева.**

Подпись студента

_____