**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](mailto: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.

1. **Цель работы:** Научиться создавать и работать с деревом.
2. **Задание (вариант 11):** проверить монотонность убывания ширины дерева.
3. **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*

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

1. **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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_