**Отчет по лабораторной работе № 23** по курсу по курсу “Практикум на ЭВМ”

Студент группы [**М8О-107Б-20**](https://vk.com/im?sel=c51) Чекменев Вячеслав Алексеевич, № по списку 27

Контакты e-mail: [chekmenev031@gmail.com](mailto:chekmenev031@gmail.com), telegram: @suraba03

Работа выполнена: «21» мая 2021 г.

Преподаватель: каф. 806 Найденов Иван Евгеньевич

Отчет сдан « » \_\_\_\_\_\_\_\_\_20 \_\_\_ г., итоговая оценка \_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

1. **Цель работы:** Составить программу на языке Си для построения и обработки дерева общего вида или упорядоченного двоичного дерева.
2. **Задание:** (вариант 15) выяснить, все ли листья находятся на одном уровне
3. **Оборудование** (студента):

Процессор *Intel Core i5-8265U* с ОП 7851 Мб, НМД 256 Гб. Монитор *1920x1080*

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

Операционная система семейства UNIX: linux, наименование: manjaro, версия: 20.1 Mikah

интерпретатор команд: bash, версия: 5.0.18.

текствый редактор: atom, версия: 5.2

Утилиты операционной системы --

Прикладные системы и программы --

Местонахождение и имена файлов программ и данных –

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

1) Написать структуру дерева, состоящую из целого числа(значения, хранимого в узле), указателей на правого и левого сына.

2) Напишем основные функции для обработки BST.

- Для создания дерева создадим его корень с некоторым значением целого типа, выделив память под структуру и заполнив поля указателей нулевыми значениями, а поле целого типа-некоторым значением.

- напишем функцию поиска мин значения в поддереве

- напишем функцию поиска узла по значению в нем

- Для добавления вершины в дерево по значению напишем функцию, которая по значению узла дерева возвращает указатель на эту вершину. Напишем функцию добавления вершины в дерево по правилам BST

- Для удаления вершины дерева рассмотрим три случая.

- Для графического представления будем выводить значения, хранимые в узлах слева направо.

3). Напишем функцию, проверяющую, все ли листья находятся на одном уровне, будем рекурсивно ходит в глубину, запоминая уровни листьев, на вовод будем подавать 1 или 0.

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

**тесты:**

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Описание тестируемого случая** |
| c 20 a 5 a 30 a 1 a 15 a 25 a 40 a 9 a 7 a 12 a 45 a 42 p l | а вот и дерево:  1  5  7  9  12  15  20  25  30  40  42  45  Не все листья на одном уровне | Проверка на разные удаления, разные добавления + вывод дерева и логическая функция |
| c 20 a 5 a 30 a 1 a 15 a 25 a 40 a 9 a 7 a 12 a 45 a 42 d 40 d 15 p l | а вот и дерево:  1  5  7  9  12  20  25  30  42  45  Не все листья на одном уровне | Другая проверка на логическую функцию |
| c 20 a 5 a 30 a 1 a 15 a 25 a 40 a 9 a 7 a 12 a 45 a 42 d 40 d 15 d 42 d 7 d 12 p l | а вот и дерево:  5  9  20  25  30  45  Все листья на одном уровне | Случай, когда все листья на одном уровне |

**8. Распечатка протокола** (подклеить листинг окончательного варианта программы с тестовыми примерами,подписанныйпреподавателем).

Tree.h --------------------------------------------------------------------------------------------------------------------------------------------------------

#ifndef \_TREE\_H\_

#define \_TREE\_H\_

#include <stdio.h>

#include <stdlib.h>

typedef struct tree {

int value;

struct tree \*right;

struct tree \*left;

} tree;

tree \*tree\_node\_create(int crt\_value);

tree \*search(tree \*root, int srch\_value);

void print\_tree(tree \*root, int level);

tree \*tree\_find\_minimum(tree \*root);

tree \*tree\_add\_node(tree \*root, int add\_value);

tree \*tree\_delete\_node(tree \*root, int del\_value);

void inorder(tree \*root);

int checkUtil(tree \*root, int level, int \*leafLevel);

int check(tree \*root);

void delete\_tree(tree \*curr);

#endif

Tree\_func.c --------------------------------------------------------------------------------------------------------------------------------------------------------

#include <stdio.h>

#include <stdlib.h>

#include "tree.h"

tree \*tree\_node\_create(int crt\_value)

{

tree \*result = (tree \*)malloc(sizeof(tree));

if (result != NULL) {

result->right = NULL;

result->left = NULL;

result->value = crt\_value;

return result;

} else {

printf("Error: not enough memory to create a node\n");

return NULL;

}

}

tree \*search(tree \*root, int srch\_value) // srch\_value is the searching value

{

if (root == NULL || root->value == srch\_value) { // if root->value == srch\_value then the element found or we reached the null node (reached the leaf's child)

return root;

}

else if (srch\_value > root->value) { // if srch\_value greater then root->value, so we will search the right subtree

return search(root->right, srch\_value);

}

else { // if srch\_value smaller then root->value, so we will search the left subtree

return search(root->left, srch\_value);

}

}

void print\_tree(tree \*root, int level)

{

if (root != NULL) {

print\_tree(root->left, level + 1);

for (int i = 0; i < level; i++) {

printf("\t");

}

printf("%d\n\n", root->value);

print\_tree(root->right, level + 1);

}

}

// tree\_find\_minimum function

// call the function, while the 1st condition is false (root (root->left from last call of func) == NULL).

// When 1st condition will become true,

// program will return current root value.

tree \*tree\_find\_minimum(tree \*root)

{

if (root == NULL) {

return NULL;

}

else if (root->left != NULL) {

return tree\_find\_minimum(root->left);

}

return root;

}

// tree\_add\_node function

// Assign function value to right (left) child's node, while the 1st condition is false.

// When 1st condition will become true,

// I'll create new node in place of the last tree\_add\_node function's output.

tree \*tree\_add\_node(tree \*root, int add\_value)

{

if (root == NULL) {

return tree\_node\_create(add\_value);

}

else if (add\_value > root->value) {

root->right = tree\_add\_node(root->right, add\_value);

}

else {

root->left = tree\_add\_node(root->left, add\_value);

}

return root;

}

tree \*tree\_delete\_node(tree \*root, int del\_value)

{

if (root == NULL) {

printf("Error: a node with value = %d does not exist in the tree\n", del\_value);

return NULL;

}

if (del\_value > root->value) {

root->right = tree\_delete\_node(root->right, del\_value);

}

else if (del\_value < root->value) {

root->left = tree\_delete\_node(root->left, del\_value);

}

else {

// no children

if (root->right == NULL && root->left == NULL) {

free(root);

return NULL;

}

// one child

else if (root->right == NULL || root->left == NULL) {

tree \*temp;

if (root->left == NULL) {

temp = root->right;

} else {

temp = root->left;

}

free(root);

return temp;

}

// two children

else {

tree \*temp = tree\_find\_minimum(root->right);

root->value = temp->value;

root->right = tree\_delete\_node(root->right, temp->value);

}

}

return root;

}

void inorder(tree \*root)

{

if (root != NULL) { // checking if the root is not null

inorder(root->left); // visiting left child

printf(" %d ", root->value); // printing data at root

inorder(root->right); // visiting right child

}

}

int checkUtil(tree \*root, int level, int \*leafLevel)

{

// Базовый вариант

if (root == NULL) {

return 1;

}

// Если встречается листовой узел

if (root->left == NULL && root->right == NULL)

{

// Когда листовой узел найден впервые

if (\*leafLevel == 0)

{

\*leafLevel = level; // Установить уровень первого найденного листа

return 1;

}

// Если это не первый листовой узел, сравниваем его уровень с

// уровень первого листа

return (level == \*leafLevel);

}

// Если этот узел не листовой, рекурсивно проверяем левое и правое поддеревья

return checkUtil(root->left, level + 1, leafLevel) && checkUtil(root->right, level + 1, leafLevel);

}

int check(tree \*root)

{

int level = 0, leafLevel = 0;

return checkUtil(root, level, &leafLevel);

}

void delete\_tree(tree \*curr)

{

if (curr) {

delete\_tree(curr->left);

delete\_tree(curr->right);

tree\_delete\_node(curr, curr->value);

}

}

client.c --------------------------------------------------------------------------------------------------------------------------------------------------------

#include <stdio.h>

#include "tree.h"

int main(int argc, char \*argv[])

{

printf("Напишите '?' для получения помощи в использовании программы:\n");

char c;

int value, node\_v;

int cnt\_root = 0;

tree \*root;

while ((c = getchar()) != EOF) {

if (c == '?') {

printf("Набор команд:\n");

printf("c - создать корень (введите значение корня).\n");

printf("a - добавить узел в дерево (введите значение нового узла).\n");

printf("d - удалить узел из дерева (введите значение удаляемого узла).\n");

printf("p - вывести дерево.\n");

printf("l - все ли листья на одном уровне?\n"); // лучше f

printf("b - удалить дерево.\n"); // лучше r

} else if (c == 'c' && cnt\_root == 0) {

cnt\_root++;

printf("введите значение корня: ");

scanf("%d", &value);

root = tree\_node\_create(value);

} else if (c == 'c' && cnt\_root != 0) {

printf("Корень уже существует!\n");

} else if (c == 'a' && cnt\_root != 0) {

printf("введите значение узла: ");

scanf("%d", &node\_v);

tree\_add\_node(root, node\_v);

} else if (c == 'a' && cnt\_root == 0) {

printf("Дерево не существует.\n");

} else if (c == 'd' && cnt\_root != 0) {

printf("введите значение узла: ");

scanf("%d", &node\_v);

root = tree\_delete\_node(root, node\_v);

} else if (c == 'd' && cnt\_root == 0) {

printf("Дерево не существует.\n");

} else if (c == 'p' && cnt\_root != 0) {

printf("а вот и дерево: \n");

print\_tree(root, 0);

} else if (c == 'p' && cnt\_root == 0) {

printf("Дерево не существует.\n");

} else if (c == 'l' && cnt\_root != 0) {

if (check(root)) {

printf("Все листья на одном уровне\n");

} else {

printf("Не все листья на одном уровне\n");

}

} else if (c == 'l' && cnt\_root == 0) {

printf("Дерево не существует.\n");

} else if (c == 'b' && cnt\_root != 0) {

delete\_tree(root);

cnt\_root--;

} else if (c == 'b' && cnt\_root == 0) {

printf("Дерево не существует.\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. |  |  |  |  |  |
| 23 | дома | 21.05 | 20 00 | закончил | никаких |  |

1. **Замечания автора** по существу работы
2. **Выводы**

В лабораторной работе я познакомился с BST. Работа была сложной при работе с динамической памятью, при построении дерева и реализации различных алгоритмов деревьев и при отладке программы, ведь с памятью нужно работать аккуратно. Деревья имеют огромное значения для хранения, представления различных данных + для поиска минимальльных, максимальных значений. Я уверен, что полученные знания мне пригодятся в будущем(например для будущих ЛР).

Подпись студента: