**Отчет по лабораторной работе № 23** по курсу

Практикум программирование

Студент группы: М8О - 110Б - 23, Гула Дмитрий Александрович, № по списку: 6,

Контакты: guladmitriy1331@gmail.com

Работа выполнена: ««4» мая 2024г.

Преподаватель: каф. 806 Почечура А. А.

Входной контроль знаний с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отчет сдан «4» мая 2024 г., итоговая оценка \_\_\_\_\_

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

1. **Тема:** Динамические структуры данных. Обработка деревьев 
2. **Цель работы: Составить и отладить программу на языке Си для построения и обработки дерева**
3. **Задание (**вариант 15 ): Проверить, находятся ли все листья двоичного дерева на одном уровне.

**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 i3-12500H \_ с ОП 8 Гб НМД 256 Гб. Монитор 1920x1080~144Hz Другие устройства



1. **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 \_Linux\_, наименование \_Ubuntu\_ версия 22.04.2 интерпретатор команд \_GNU bash\_ версия 5.1.16.

Система программирования С. Редактор текстов emacs версия 29.1

Утилиты операционной системы

Прикладные системы и программы Emacs

Местонахождение и имена файлов программ и данных на домашнем компьютере /home/

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

Программа состоит из трех подпрограмм:

1. Заголовочного файла functions\_and\_struct.h, в котором создаётся структурный тип tree заданного формата.
2. Программы treeFunctions.c, который расписывает все функции, которые мы используем в программе.
3. Программы main.c, которая выводит интерфейс и позволяет осуществлять с ним работу при помощи функций, заданных в предыдущем файле.
4. **Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию].

#ifndef C\_FUNCTIONS\_AND\_STRUCT\_H  
#define C\_FUNCTIONS\_AND\_STRUCT\_H  
  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <assert.h>  
#include <stdbool.h>  
  
typedef struct node node;  
  
struct node{  
 double val;  
 node\* left;  
 node\* right;  
};  
  
typedef node\* tree;  
  
  
tree create\_empty\_tree();  
void destroy(tree t);  
tree build\_tree(tree left, double val, tree right);  
bool tree\_is\_empty(tree t);  
double get\_val(tree t);  
tree get\_left(tree t);  
tree get\_right(tree t);  
tree insert(tree \*t, double val);  
void print\_tree(tree t, int h);  
int height(tree t);  
int level\_width(tree t, int level);  
bool is\_width\_monotonic(tree t);  
tree delete\_node(tree t, double val);  
int search\_val(tree t, double val);  
void print\_menu();  
  
#endif //C\_FUNCTIONS\_AND\_STRUCT\_H

#include "functions\_and\_struct.h"  
#include <stdio.h>  
#include <stdbool.h>  
  
tree create\_empty\_tree() {  
 return NULL;  
}  
  
void destroy(node \*root) {  
 if (root != NULL) {  
 destroy(root->left);  
 destroy(root->right);  
 free(root);  
 }  
}  
  
tree build\_tree(tree left, double val, tree right) {  
 tree tmp = (tree) malloc(sizeof(node));  
 tmp->left = left;  
 tmp->right = right;  
 tmp->val = val;  
 return tmp;  
}  
  
bool tree\_is\_empty(tree t) {  
 return t == NULL;  
}  
  
double get\_val(tree t) {  
 return t->val;  
}  
  
tree get\_left(tree t) {  
 return t->left;  
}  
  
tree get\_right(tree t) {  
 return t->right;  
}  
  
tree insert(tree \*t, double val) {  
 if (tree\_is\_empty(\*t)) {  
 \*t = build\_tree(NULL, val, NULL);  
 } else if (get\_val(\*t) > val) {  
 (\*t)->left = insert(&((\*t)->left), val);  
 } else if (get\_val(\*t) < val) {  
 (\*t)->right = insert(&((\*t)->right), val);  
 }  
 return \*t;  
}  
  
void print\_tree(tree t, int h) {  
 if (t != NULL) {  
 print\_tree(t->right, h + 1);  
 for (int i = 0; i < h; i++) {  
 printf("\t");  
 }  
 printf("%f\n", t->val);  
 print\_tree(t->left, h + 1);  
 }  
}  
  
int height(tree t) {  
 if (tree\_is\_empty(t)) {  
 return 0;  
 } else {  
 int left\_height = height(t->left);  
 int right\_height = height(t->right);  
  
 return (left\_height > right\_height) ? left\_height + 1 : right\_height + 1;  
 }  
}  
  
int level\_width(tree t, int level) {  
 if (tree\_is\_empty(t)) {  
 return 0;  
 }  
 if (level == 1) {  
 return 1;  
 } else if (level > 1) {  
 return level\_width(t->left, level - 1) + level\_width(t->right, level - 1);  
 }  
 return 0;  
}  
  
bool is\_width\_monotonic(tree t) {  
 int h = height(t);  
 for (int i = 2; i <= h; i++) {  
 if (level\_width(t, i) >= level\_width(t, i - 1)) {  
 return false;  
 }  
 }  
 return true;  
}  
  
tree find\_min(tree t) {  
 while (!tree\_is\_empty(get\_left(t))) {  
 t = get\_left(t);  
 }  
 return t;  
}  
  
//tree delete\_node(tree t, double val) {  
// if (tree\_is\_empty(t)) {  
// return t;  
// }  
//  
// if(val > t->val){  
// t->right = delete\_node(t->right, val);  
// }else if(val < t->val){  
// t->left = delete\_node(t->left, val);  
// }else{  
// if(t->left == 0 && t->right == 0){  
// free(t);  
// return NULL;  
// }else if(t->left == 0 || t->right == 0){  
// tree temp;  
// if(t->left == 0){  
// temp = t->right;  
// }else{  
// temp = t->left;  
// }  
// free(t);  
// return temp;  
// }else{  
// tree temp;  
// temp = find\_min(t->right);  
// t->val = temp->val;  
// t->right = delete\_node(t->right, temp->val);  
// }  
// }  
// return t;  
//}  
  
  
tree delete\_node(tree t, double val) {  
 if (tree\_is\_empty(t)) {  
 return t;  
 }  
 if (val < get\_val(t)) {  
 t->left = delete\_node(get\_left(t), val);  
 } else if (val > get\_val(t)) {  
 t->right = delete\_node(get\_right(t), val);  
 } else {  
 if (tree\_is\_empty(get\_left(t))) {  
 tree temp = get\_right(t);  
 free(t);  
 return temp;  
 } else if (tree\_is\_empty(get\_right(t))) {  
 tree temp = get\_left(t);  
 free(t);  
 return temp;  
 }  
 tree temp = find\_min(get\_right(t));  
 t->val = get\_val(temp);  
 t->right = delete\_node(get\_right(t), get\_val(temp));  
 }  
 return t;  
}  
  
int search\_val(tree t, double val) {  
 if (tree\_is\_empty(t)) {  
 return 0;  
 } else if (t->val == val) {  
 return 1;  
 } else if (t->val < val) {  
 if (t->right != 0) {  
 search\_val(t->right, val);  
 } else {  
 return 0;  
 }  
 } else if (t->val > val) {  
 if (t->left != 0) {  
 search\_val(t->left, val);  
 } else {  
 return 0;  
 }  
 }  
}  
  
void print\_menu() {  
 printf("\nВыберите действие из списка, написав его номер: \n");  
 printf("Введите 0, чтобы выйти.\n");  
 printf("Введите 1, чтобы ввести в дерево новый элемент.\n");  
 printf("Введите 2 чтобы распечатать дерево.\n");  
 printf("Введите 3, чтобы удалить из дерева элемент.\n");  
 printf("Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.\n");  
}

#include "functions\_and\_struct.h"  
  
int main() {  
 tree t = create\_empty\_tree();  
 print\_menu();  
 int menu\_number;  
 while (1) {  
 printf("Введите номер меню: ");  
 if (scanf("%d", &menu\_number) != 1) {  
 printf("Ошибка: введите число.\n");  
 while (getchar() != '\n');  
 continue;  
 }  
 if (menu\_number == 0) {  
 printf("Выход из программы...\n");  
 destroy(t);  
 exit(0);  
 }  
 switch (menu\_number) {  
 case 1: {  
 double number;  
 printf("Введите значение элемента: \n");  
 scanf("%lf", &number);  
 if (search\_val(t, number) == 1) {  
 printf("Такой элемент уже содержится в дереве, попробуйте другой. \n");  
 } else {  
 insert(&t, number);  
 printf("Значение элемента успешно добавлено в дерево. \n");  
 }  
 break;  
 }  
 case 2: {  
 if (!tree\_is\_empty(t)) {  
 printf("Полученное дерево: \n");  
 print\_tree(t, 0);  
 } else {  
 printf("Дерево пустое. \n");  
 }  
 break;  
 }  
 case 3: {  
 if (!tree\_is\_empty(t)) {  
 double number;  
 printf("Введите элемент, который хотите удалить:\n");  
 scanf("%lf", &number);  
 if (search\_val(t, number) == 1) {  
 t = delete\_node(t, number);  
 printf("Элемент %lf успешно удалён!\n", number);  
 } else {  
 printf("Такого элемента не содержится в дереве. \n");  
 }  
 } else {  
 printf("Я не могу удалить элемент из пустого дерева\n");  
 }  
 break;  
 }  
 case 4: {  
 if (!tree\_is\_empty(t)) {  
 if (is\_width\_monotonic(t)) {  
 printf("Ширина уровня бинарного дерева монотонно убывает.\n");  
 } else {  
 printf("Ширина уровня бинарного дерева не убывает монотонно.\n");  
 }  
 } else {  
 printf("Невозможно проверить пустое дерево\n");  
 }  
 break;  
 }  
 default:  
 printf("Ошибка: Неверный выбор меню. Попробуйте еще раз.\n");  
 }  
 print\_menu();  
 }  
 return 0;  
}

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

C:\Users\Дима\CLionProjects\C\LB23\cmake-build-debug\LB23.exe

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 1

Введите значение элемента:

100

Значение элемента успешно добавлено в дерево.

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 1

Введите значение элемента:

200

Значение элемента успешно добавлено в дерево.

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 1

Введите значение элемента:

50

Значение элемента успешно добавлено в дерево.

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 2

Полученное дерево:

200.000000

100.000000

50.000000

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 1

Введите значение элемента:

500

Значение элемента успешно добавлено в дерево.

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 1

Введите значение элемента:

300

Значение элемента успешно добавлено в дерево.

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 1

Введите значение элемента:

1000

Значение элемента успешно добавлено в дерево.

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 1

Введите значение элемента:

321

Значение элемента успешно добавлено в дерево.

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 2

Полученное дерево:

1000.000000

500.000000

321.000000

300.000000

200.000000

100.000000

50.000000

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 1

Введите значение элемента:

40

Значение элемента успешно добавлено в дерево.

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 1

Введите значение элемента:

60

Значение элемента успешно добавлено в дерево.

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 1

Введите значение элемента:

30

Значение элемента успешно добавлено в дерево.

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 2

Полученное дерево:

1000.000000

500.000000

321.000000

300.000000

200.000000

100.000000

60.000000

50.000000

40.000000

30.000000

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 3

Введите элемент, который хотите удалить:

50

Элемент 50.000000 успешно удалён!

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 2

Полученное дерево:

1000.000000

500.000000

321.000000

300.000000

200.000000

100.000000

60.000000

40.000000

30.000000

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 3

Введите элемент, который хотите удалить:

200

Элемент 200.000000 успешно удалён!

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 2

Полученное дерево:

1000.000000

500.000000

321.000000

300.000000

100.000000

60.000000

40.000000

30.000000

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 4

Ширина уровня бинарного дерева не убывает монотонно.

Выберите действие из списка, написав его номер:

Введите 0, чтобы выйти.

Введите 1, чтобы ввести в дерево новый элемент.

Введите 2 чтобы распечатать дерево.

Введите 3, чтобы удалить из дерева элемент.

Введите 4, чтобы проверить, что ширина уровня бинарного дерева монотонно убывает.

Введите номер меню: 0

Выход из программы...

Process finished with exit code 0

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



1. **Выводы**

В ходе выполнения данной работы я познакомился со структурой бинарного дерева и научился работать с ней: добавлять новые элементы, удалять и выводить.

Подпись студента