Отчёт по лабораторной работе № 23 по курсу 1

студента группы М80-108Б-19 Хренниковой Ангелины , № по списку 23

Адреса www, e-mail, jabber, skype: lina.khrennikova@mail.ru

Работа выполнена: "8" апреля 2020г.

Преподаватель: Поповкин А. В. каф.806

Входной контроль знаний с оценкой

Отчёт сдан "9" апреля 2020 г., итоговая оценка

Подпись преподавателя

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

2. **Цель работы**: Составить программу на языке Си для построения и обработки дерева общего вида или упорядоченного двоичного дерева. Основные функции работы с деревьями реализовать в виде универсальных процедур или функций. После того, как дерево создано, его обработка должна производиться в режиме текстового меню со следующими действиями: 1. Добавление нового узла(для двоичного дерева положение нового узла определяется в соответствии с требованием сохранения порядка); 2. Текстовая визуализация дерева; 3. Удаление узла(двоичное дерево перестраивается в соответствии с требованием сохранения целостности и порядка); 4. Вычисление значения некоторой функции от дерева.

3. **Задание** (*вариант №23*): Определить число вершин двоичного дерева(корень не является вершиной дерева)

4. **Оборудование** (*лабораторное*):
ЭВМ PC, процессор Intel® Core™ i7-3770 CPU @ 3.40GHz * 8, имя узла сети alise18 с ОП 15974,4 МБ, НМД 345,5 ГБ.
Терминал Gnome адрес 192.168.2.118/24 . Принтер
Другие устройства

Оборудование ПЭВМ студента, если использовалось:
Процессор Intel® Core™ i3-7020U CPU @ 2.30GHz * 4, ОП 8192 МБ, НМД 256 ГБ. Монитор LCD
Другие устройства

5. **Программное обеспечение** (*лабораторное*):
Операционная система семейства UNIX, наименование Ubuntu версия 18.04
Интерпретатор команд Bash версия 4.4.20(1)
Система программирования версия
Редактор текстов Nano версия 2.9.3
Утилиты операционной системы touch, make, cat, ls
Прикладные системы и программы
Местонахождения и имена файлов программ и данных home/stud/olen

Программное обеспечение ЭВМ студента, если использовалось:
Операционная система семейства UNIX, наименование Ubuntu версия 18.04
Интерпретатор команд Bash версия 4.4.19(1)
Система программирования версия
Редактор текстов Emacs версия 25.2.2

Утилиты операционной системы touch, cat, ls, make

Прикладные системы и программы

Местонахождения и имена файлов программ и данных home/lina_tucha/dir/lab23

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

Основные функции работы с деревьями реализованы в виде универсальных процедур или функций(добавление, удаление вершин, печать дерева и пр). Создано текстовое меню для работы с деревом, в одном из пунктов которого находится количество вершин дерева(считаются при добавлении/удалении вершины).

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

*Пункты 1-7 отчёта составляются **строго до** начала лабораторной работы.*

```
lina_tucha@LAPTOP-44CRFC1U:~/dir/lab23$ gcc lab23.c -o 1234
lina_tucha@LAPTOP-44CRFC1U:~/dir/lab23$ ./1234
Выберите действие:
1) Добавить вершину
2) Удалить вершину
3) Распечатать дерево
4) Вывести уровень с максимальным числом вершин
5) Вывести число вершин дерева
1
Введите вершину
5
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
1
Введите вершину
4
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
1
Введите вершину
6
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
1
Введите вершину
7
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
1
Введите вершину
23
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
1
Введите вершину
2
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
1
Введите вершину
33
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
1
Введите вершину
12
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
1
Введите вершину
4
Такая вершина уже есть
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
3
```

```
            33
          23
            12
        7
    6
5
    4
        2
```
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
2
Введите вершину
23
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
3
```
          33
            12
        7
    6
5
    4
        2
```
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
4
Уровень с максимальным числом вершин:
1
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
5
Число вершин дерева:
6
Выбирайте действие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
1
Введите вершину
101
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
1
Введите вершину
44
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
1
Введите вершину
15
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
1
Введите вершину
3
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
1
Введите вершину
1
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
2
Введите вершину
33
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
2
Введите вершину
5
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
3
```
          101
        44
            15
          12
    7
6
```

```
    4
        3
      2
        1
```
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
5
Число вершин дерева:
9
Выбирайте действие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
4
Уровень с максимальным числом вершин:
3
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
2
Введите вершину
6
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
2
Введите вершину
7
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
2
Введите вершину
4
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
3
```
        101
    44
        15
  12
        3
    2
        1
```
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
5
Число вершин дерева:
6
Выбирайте действие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
2
Введите вершину
44
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
2
Введите вершину
101
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
2
Введите вершину
15
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
2
Введите вершину
44
Такой вершины нет
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
1
Введите вершину
5
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
1
Введите вершину
6
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
1
Введите вершину
```
```

```
7
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
3
   12
                7
             6
          5
       3
    2
       1
Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
5
Число вершин дерева:
6
Выбирайте действие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result
```

Допущен к выполнению работы. Подпись преподавателя _____

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

```
lina_tucha@LAPTOP-44CRFC1U:~/dir/lab23$ cat lab23.c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <math.h>
#include <malloc.h>

typedef struct tree{
	int value;
	struct tree * left;
	struct tree * right;
}node;

void create(node **root, int value){
	node *tmp = malloc(sizeof(node));
	tmp -> value = value;
	tmp -> left = NULL;
	tmp -> right = NULL;
	*root = tmp;
}

void add(node **root, int value){
	node *root2 = *root;
	node *root3 = NULL;
	node *tmp = malloc(sizeof(node));
	tmp -> value = value;
	while (root2 != NULL){
	root3 = root2;
	if (value < root2 -> value)
	   root2 = root2 -> left;
	else
	   root2 = root2 -> right;
	}
	tmp -> left = NULL;
	tmp -> right = NULL;
	if (value < root3 -> value)
	      root3 -> left = tmp;
	else
	      root3 -> right = tmp;
}

int binary_check(node **root, int value){
	int boolean = 0;
	node *root2 = *root;
	while (1){
```

```
            if (value == root2 -> value){
                boolean = 1;
                break;
            }
            if (value < root2 -> value){
                if (root2 -> left == NULL)
                    break;
                root2 = root2 -> left;
            }
            if (value > root2 -> value){
                if (root2 -> right == NULL)
                    break;
                root2 = root2 -> right;
            }
        }
        return boolean;
}

void *delete(node **root, int value){
        node *l = *root;
        while (l -> value != value){
            if (value < l -> value)
                l = l -> left;
            else l = l -> right;
        }
        if (l -> left == NULL && l -> right == NULL){
            node *root2 = *root;
            node *root3 = *root;
            while (1){
            if (root2 -> left != NULL)
                if (root2 -> left -> value == value)
                    break;
            if (root2 -> right != NULL)
                if (root2 -> right -> value == value)
                    break;
            if (value < root2 -> value)
                root2 = root2 -> left;
            else root2 = root2 -> right;
            root3 = root2;
        }
            if (l == root3 -> right)
                root3 -> right = NULL;
            else root3 -> left = NULL;
            free(l);
        }
        if (l -> left == NULL && l -> right != NULL){
            node *root2 = *root;
            node *root3 = *root;
            while (1){
            if (root2 -> left != NULL)
                if (root2 -> left -> value == value)
                    break;
            if (root2 -> right != NULL)
                if (root2 -> right -> value == value)
                    break;
            if (value < root2 -> value)
                root2 = root2 -> left;
            else root2 = root2 -> right;
            root3 = root2;
        }
            if (l == root3 -> right) root3 -> right = l -> right;
            else root3 -> left = l -> right;
            free(l);
        }
        if (l -> left != NULL && l -> right == NULL){
```

```
        node *root2 = *root;
        node *root3 = *root;
        while (1){
        if (root2 -> left != NULL)
            if (root2 -> left -> value == value)
                break;
        if (root2 -> right != NULL)
            if (root2 -> right -> value == value)
                break;
        if (value < root2 -> value)
            root2 = root2 -> left;
        else root2 = root2 -> right;
        root3 = root2;
    }
        if (l == root3 -> right) root3 -> right = l -> left;
        else root3 -> left = l -> left;
        free(l);
    }
    if (l -> left != NULL && l -> right !=NULL){
        node *root2 = l;
        while (1){
            if (root2 -> right == NULL){
                root2 = root2 -> left;
                while (root2 -> right != NULL)
                    root2 = root2 -> right;
            }
            if (root2 -> right != NULL){
                root2 = root2 -> right;
                while (root2 -> left != NULL)
                    root2 = root2 -> left;
            }
            break;
        }
        node *root3 = *root;
        node *root4 = *root;
        while (1){
            if (root3 -> left != NULL)
                if (root3 -> left -> value == root2 -> value)
                    break;
            if (root3 -> right != NULL)
                if (root3 -> right -> value == root2 -> value)
                    break;
            if (value < root3 -> value)
                root3 = root3 -> left;
            else root3 = root3 -> right;
            root4 = root3;
        }
        l -> value = root2 -> value;
        if (root4 -> left -> value == root2 -> value){
            if (root4 -> left -> left != NULL)
                root4 -> left = root4 -> left -> right;
            else if (root4 -> left -> right != NULL)
                root4 -> left = root4 -> left -> right;
            else root4 -> left = NULL;
        }
        else {
            if (root4 -> left -> right != NULL)
                root4 -> left = root4 -> left -> right;
            else if (root4 -> right -> right != NULL)
                root4 -> right = root4 -> right -> right;
            else root4 -> right = NULL;
        }
        free(root2);
    }
}
```

```
void tree_print(node *root, int l){
	if (root == NULL)
		return;
	tree_print(root -> right, l+1);
	for (int i = 0; i < l; ++i)
		printf("    ");
	printf("%d\n",root -> value );
	tree_print(root -> left, l+1);
}

void get_solve(node *root, int l, int lvl[]){
	if (root == NULL)
		return;
	get_solve(root -> right, l + 1, lvl);
	++lvl[l];
	get_solve(root -> left, l + 1, lvl);
}

int main(){
	int hello=0;
	node *root;
	int size = 0, c, v, vertex_count = 0;
	printf("Выберайту действие\n");
	printf("1) Добавить вершину\n");
	printf("2) Удалить вершину\n");
	printf("3) Распечатать дерево\n");
	printf("4) Вывести уровень с максимальным числом вершин\n");
	printf("5) Вывести число вершин дерева\n");
	while (scanf("%d", &c) != EOF){
		if (c == 1){
			printf("Введите вершину\n");
			scanf("%d", &v);
			if (v > size)
				size = v;
			if (vertex_count == 0){
				create(&root, v);
				++vertex_count;
				printf("Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result\n");
			}
			else {
				if (binary_check(&root, v)){
					printf("Такая вершина уже есть\n");
					printf("Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result\n");
					continue;
				}
				add(&root, v);
				++vertex_count;
				hello++;
				printf("Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result\n");
			}
		}
		if (c == 2){
			printf("Введите вершину\n");
			scanf("%d", &v);
			if (vertex_count < 2){
				printf("Нельзя удалить дерево\n");
				printf("Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result\n");
				continue;
				free(root);
			}
			if (!binary_check(&root, v)){
				printf("Такой вершины нет\n");
				printf("Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result\n");
				continue;
```

```
            }else{
                --vertex_count;
                hello--;
                delete(&root, v);
                printf("Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result\n");
            }
        }
        if (c == 3){
            if (vertex_count == 0){
                printf("Дерева не существует\n");
                printf("Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result\n");
                continue;
            }
            tree_print(root, 1);
            printf("Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result\n");
        }
        if (c == 4){
            if (vertex_count == 0){
                printf("Дерева не существует\n");
                continue;
            }
            int *lvl = NULL;
            lvl = (int*) calloc (size + 1, sizeof(int));
            get_solve(root, 1, lvl);
            int answer = -1, maximum = -1;
            for (int i = 0; i < size; ++i){
                if (maximum < lvl[i]){
                    maximum = lvl[i];
                    answer = i;
                }
            }
            printf("Уровень с максимальным числом вершин:\n");
            printf("%d\n", answer - 1);
            printf("Выбирайте дейтсвие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result\n");
        }
        if (c == 5) {
            printf("Число вершин дерева:\n");
            printf("%i\n", hello);
            printf("Выбирайте действие: 1 - add, 2 - del, 3 - print, 4 - get lvl, 5 - result\n");
        }
    }
}
```

9. **Дневник отладки** должен содержать дату и время сеансов отладки, и основные ошибки (ошибки в сценарии и программе, не стандартные операции) 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. Замечание автора по существу работы _____

11. Выводы : Я составила программу на языке Си для построения и обработки дерева общего вида или упорядоченного двоичного дерева. Основные функции работы с деревьями реализовала в виде универсальных процедур или функций. После того, как дерево создано, его обработка производится в режиме текстового меню со следующими действиями: 1. Добавление нового узла(для двоичного дерева положение нового узла определяется в соответствии с требованием сохранения порядка); 2. Текстовая визуализация дерева; 3. Удаление узла(двоичное дерево перестраивается в соответствии с требованием сохранения целостности и порядка); 4. Вычисление значения некоторой функции от дерева

Недочеты, допущенные при выполнении задания, могут быть устранены следующим образом _____

Подпись студента Хренникова А. С.