**Отчет по лабораторной работе №23** по курсу практикум на ЭВМ

Студент группы М8О-101Б-20 Ядров Артем Леонидович, № по списку 28

Контакты www, e-mail, icq, skype temayadrow@gmail.com

Работа выполнена: « » \_\_\_\_\_\_\_\_\_\_\_\_202\_\_\_г.

Преподаватель: доцент каф. 806 Никулин Сергей Петрович

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

Отчет сдан « » \_\_\_\_\_\_\_\_\_202 \_\_ г., итоговая оценка \_\_\_\_\_

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

1. **Тема:** Динамические структуры данных. Обработка деревьев.
2. **Цель работы:** Научиться работать с динамическими структурами данных и обрабатывать деревья

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Задание** (*вариант №* *26* )**:**  Определить степень дерева
2. **Оборудование** (лабораторное):

ЭВМ Intel Pentium G2140, процессор 3.30 GHz , имя узла сети Cameron с ОП 8096 Мб, НМД 7906 Мб. Терминал ASUS адрес dev/pets/3 Принтер HP Laserjet 6P

Другие устройства \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Оборудование ПЭВМ студента, если использовалось:*

Процессор \_\_Intel core i5-7300HQ 2.50 GHz с ОП 8096 Мб, НМД 131072 Мб. Монитор ASUS

Другие устройства \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Операционная система семейства Unix , наименование Ubuntu версия 18.15.0

интерпретатор команд bash версия 4.4.20

Система программирования CLion версия 2020.3

Редактор текстов emacs версия 25.2.2

Утилиты операционной системы cat, gcc

Прикладные системы и программы

Местонахождение и имена файлов программ и данных stud/208104

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Программное обеспечение ЭВМ студента, если использовалось:*

Операционная система семейства Unix , наименование Fedora версия 33

интерпретатор команд bash версия 5.0.17

Система программирования CLion версия 2020.3

Редактор текстов emacs версия 25.2.2 Утилиты операционной системы cat, gcc

Прикладные системы и программы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Местонахождение и имена файлов программ и данных на домашнем компьютере home/Temi4

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Опишем следующие структуры:

* struct tnode {

float value;

struct tnode \*son;

struct tnode \*brother;

struct tnode \*parent;

};

Структура узла дерева. Хранит значение узла, указатель на старшего сына, указатель на следующего брата, указатель на родителя.

* typedef struct {

node \*root;

} Tree;

Структура самого дерева. Хранит указатель на корень.

С помощью этих структур реализуем функции для работы с деревом:

* node \*create\_node(float f, node \*par)

Функция создания узла дерева. Создает указатель на узел дерева, помещает в него значение и родителя, а затем возвращает указатель на созданный узел

* Tree \*create\_tree(float f)

Функция создания дерева. Создает указатель на дерево, затем создает корень дерева с помощью описанный выше функции. Возвращает указатель на созданное дерево.

* node \*search\_tree(node \*t, float f)

Функция поиска узла по значению. Работает путем поиска в глубину и возвращает первый найденный узел с значением f (т. е. самый «глубокий»).

* void add\_node\_in\_tree(Tree \*tree, float par\_f, float f)

Функция добавления узла дерева. Для создания узла ищется родитель узла с помощью функции search\_tree. Затем создается узел с помощью функции create\_node. Если у родителя нет сына, то создаваемый узел становится сыном родителя. В противном случае становится самым младшим братом старшего сына родителя.

* void delete\_node(Tree \*t, float f)

Функция удаления узла дерева. С помощью функции search\_tree находится удаляемый узел. Старшим сыном родителя выбранного узла становится следующий брат выбранного узла (если братьев нет, то, как и положено, указатель на сына становится NULL). Указатель на выбранный узел освобождается.

* void print\_tree(node \*t, int x)

Функция печати узла дерева. Печать осуществляется в порядке обхода КЛП. Братья находятся слева от родителя на одной вертикальной линии.

* int node\_degree(node \*t)

Функция возвращает степень узла дерева. Проходит старшего сына, а затем всех его братьев.

* int max(int a, int b)

Возвращает максимальное значение двух чисел.

* int task(node \*t, int mx)

Рекурсивная функция. Производит обход в глубину. Сначала обрабатывается сам узел (вычисляется степень узла), затем старший сын, а затем брат. Возвращает максимум из текущего максимума (mx), степени узла, степени старшего сына узла и степени брата узла.

В основной части программы будем использовать меню, в котором есть 6 опций:

1. Создание дерева (Create tree)

Запрашивает значение корня дерева, а затем создает дерево, вызывая функцию create\_tree

1. Добавление узла в дерево (Add node to tree)

Запрашивает значение добавляемого узла, а затем вызывает функцию add\_node

1. Удаление узла дерева (Delete node from tree)

Вызывает функцию delete\_node

1. Выполнение задания (вычисление степени дерева) (Task)

Вызывает функцию task от корня с максимальным значением 0, а затем выводит ответ.

1. Печать дерева (Print tree)

Вызывает функцию print\_tree

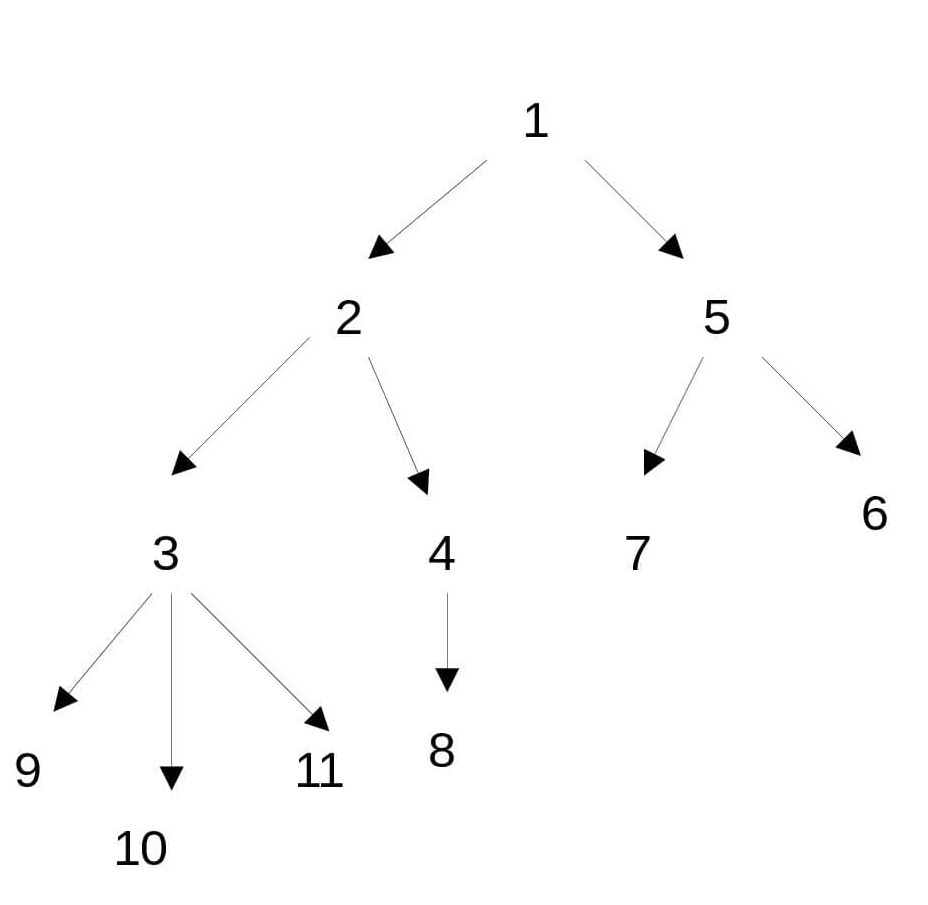
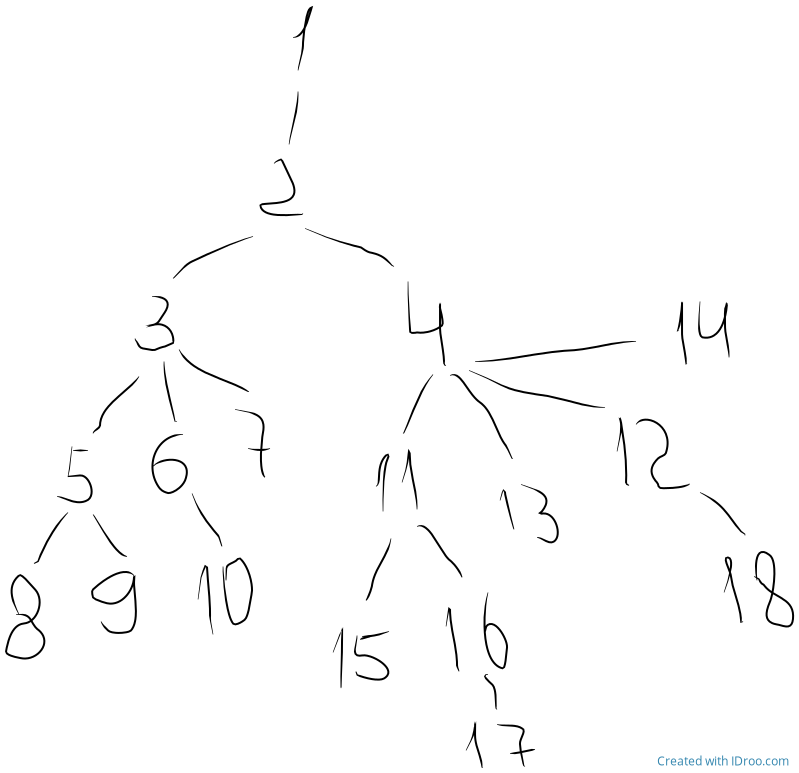
1. Выход (Exit)

Выходит из меню

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

Тесты:

* Дерево, состоящее из одного корня (1)
* Пустое дерево

****

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

*Допущен к выполнению работы.* **Подпись преподавателя****\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

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

**[Temi4@localhost 23]$ cat tree.c**

**#include <stdio.h>**

**#include <stdlib.h>**

**struct tnode {**

**float value;**

**struct tnode \*son;**

**struct tnode \*brother;**

**struct tnode \*parent;**

**};**

**typedef struct tnode node;**

**typedef struct {**

**node \*root;**

**} Tree;**

**struct Item {**

**node \*t;**

**struct Item \*next;**

**};**

**struct queue {**

**struct Item \*first;**

**struct Item \*last;**

**int size;**

**};**

**struct queue \*create\_queue() {**

**struct queue \*q = (struct queue \*) malloc(sizeof(struct queue));**

**q->first = (struct Item \*) malloc(sizeof(struct Item));**

**q->last = (struct Item \*) malloc(sizeof(struct Item));**

**q->size = 0;**

**return q;**

**}**

**int empty\_queue(struct queue \*q) {**

**return q->size == 0;**

**}**

**int Push(struct queue \*q, node \*t) {**

**if (!(q->last->next = (struct Item\*) malloc(sizeof (struct Item)))){**

**return 0;**

**}**

**q->last->t = t;**

**q->last = q->last->next;**

**q->size++;**

**return 1;**

**}**

**int Pop(struct queue \*q) {**

**if (q->first == q->last) {**

**return 0;**

**}**

**struct Item \*pi = q->first;**

**q->first = q->first->next;**

**q->size--;**

**free(pi);**

**return 1;**

**}**

**node \*Top(struct queue \*q) {**

**if (q->first != q->last) {**

**return q->first->t;**

**}**

**return NULL;**

**}**

**node \*create\_node(float f, node \*par) {**

**node \*t;**

**t = (node \*) malloc(sizeof(node));**

**t->value = f;**

**t->son = NULL;**

**t->brother = NULL;**

**t->parent = par;**

**return t;**

**}**

**Tree \*create\_tree(float f) {**

**Tree \*t;**

**t = (Tree \*) malloc(sizeof(Tree));**

**t->root = create\_node(f, NULL);**

**return t;**

**}**

**node \*search\_tree(node \*t, float f) {**

**if (t == NULL){**

**return t;**

**}**

**node \*tree = NULL;**

**if (t->value == f) {**

**return t;**

**}**

**if (t->son != NULL) {**

**tree = search\_tree(t->son, f);**

**}**

**if (tree == NULL) {**

**tree = search\_tree(t->brother, f);**

**}**

**return tree;**

**}**

**void add\_node\_in\_tree(Tree \*tree, float par\_f, float f) {**

**node \*t = tree->root;**

**t = search\_tree(t, par\_f);**

**if (t == NULL) {**

**printf("%-.2f not contains in tree\n", par\_f);**

**return;**

**}**

**if (t->son == NULL) {**

**t->son = create\_node(f, t);**

**} else {**

**t = t->son;**

**while (t->brother != NULL) {**

**t = t->brother;**

**}**

**t->brother = create\_node(f, t->parent);**

**}**

**}**

**void delete\_node(Tree \*tree, float f) {**

**node \*t = tree->root;**

**t = search\_tree(t, f);**

**if (t == NULL) {**

**printf("%-.2f not contains in tree\n", f);**

**return;**

**}**

**if (t->parent->son == t){**

**t->parent->son = t->brother;**

**}**

**else{**

**node \*tr = t->parent->son;**

**while (tr->brother != t){**

**tr = tr->brother;**

**}**

**tr->brother = t->brother;**

**}**

**free(t);**

**}**

**void print\_tree(node \*t, int x) {**

**if (t == NULL) {**

**return;**

**}**

**for (int i = 0; i < x; i++) {**

**printf("\t");**

**}**

**printf("%-.2f\n", t->value);**

**print\_tree(t->son, x + 1);**

**print\_tree(t->brother, x);**

**}**

**int task(Tree \*T) {**

**node \*t = T->root;**

**struct queue \*q = create\_queue();**

**Push(q, t);**

**int cur\_lvl = 0;**

**int prev\_lvl = 0;**

**node \*end\_cur\_level = T->root;**

**while (!empty\_queue(q)) {**

**t = Top(q);**

**cur\_lvl++;**

**Pop(q);**

**if (t == end\_cur\_level){**

**if (end\_cur\_level->son != NULL){**

**end\_cur\_level = end\_cur\_level->son;**

**while (end\_cur\_level->brother != NULL){**

**end\_cur\_level = end\_cur\_level->brother;**

**}**

**}**

**else {**

**node \*tree = end\_cur\_level->parent->son;**

**end\_cur\_level = NULL;**

**while (tree->brother != NULL) {**

**tree = tree->brother;**

**if (tree->son != NULL) {**

**end\_cur\_level = tree->son;**

**}**

**}**

**}**

**if (prev\_lvl < cur\_lvl){**

**prev\_lvl = cur\_lvl;**

**cur\_lvl = 0;**

**}**

**else{**

**return 0;**

**}**

**end\_cur\_level = NULL;**

**}**

**node \*tree = t->son;**

**if (tree == NULL) {**

**continue;**

**}**

**Push(q, tree);**

**while (tree->brother != NULL) {**

**Push(q, tree->brother);**

**tree = tree->brother;**

**}**

**}**

**return 1;**

**}**

**int main() {**

**Tree \*t = NULL;**

**int choose, g = 1;**

**while (g) {**

**printf("1. Create tree\t 2. Add node to tree\t 3. Delete node from tree\t 4. Task\t 5. Print tree\t 6. Exit \n");**

**scanf("%d", &choose);**

**switch (choose) {**

**case 1: {**

**printf("Write tree's root\n");**

**float f;**

**scanf("%f", &f);**

**t = create\_tree(f);**

**break;**

**}**

**case 2: {**

**printf("Write tree node value\n");**

**float f, par\_f;**

**scanf("%f", &f);**

**printf("Write parent value\n");**

**scanf("%f", &par\_f);**

**add\_node\_in\_tree(t, par\_f, f);**

**break;**

**}**

**case 3: {**

**printf("Write tree node value\n");**

**float f;**

**scanf("%f", &f);**

**delete\_node(t, f);**

**break;**

**}**

**case 4: {**

**if (task(t)) {**

**printf("The width of the tree level rises\n");**

**} else {**

**printf("The width of the tree level doesn't rise\n");**

**}**

**break;**

**}**

**case 5: {**

**print\_tree(t->root, 0);**

**break;**

**}**

**case 6: {**

**g = 0;**

**break;**

**}**

**default: {**

**printf("Wrong answer\n");**

**}**

**}**

**}**

**return 0;**

**}[Temi4@localhost 23]$ gcc tree.c**

**[Temi4@localhost laabs]$ ./a.out**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**1**

**Write tree's root**

**1**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**2**

**Write parent value**

**1**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**5**

**Write parent value**

**1**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**3**

**Write parent value**

**2**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**4**

**Write parent value**

**2**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**9**

**Write parent value**

**3**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**10**

**Write parent value**

**3**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**11**

**Write parent value**

**3**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**4**

**Write parent value**

**8**

**8.00 not contains in tree**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**8**

**Write parent value**

**4**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**7**

**Write parent value**

**5**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**6**

**Write parent value**

**5**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**5**

**1.00**

**2.00**

**3.00**

**9.00**

**10.00**

**11.00**

**4.00**

**8.00**

**5.00**

**7.00**

**6.00**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**4**

**tree grade is 3**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**3**

**Write tree node value**

**3**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**5**

**1.00**

**2.00**

**4.00**

**8.00**

**5.00**

**7.00**

**6.00**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**4**

**tree grade is 2**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**6**

**[Temi4@localhost laabs]$ ./a.out**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**1**

**Write tree's root**

**1**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**2**

**Write parent value**

**1**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**3**

**Write parent value**

**2**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**4**

**Write parent value**

**2**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**5**

**Write parent value**

**3**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**6**

**Write parent value**

**3**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**7**

**Write parent value**

**3**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**8**

**Write parent value**

**5**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**9**

**Write parent value**

**5**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**10**

**Write parent value**

**6**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**11**

**Write parent value**

**4**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**13**

**Write parent value**

**4**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**12**

**Write parent value**

**4**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**14**

**Write parent value**

**4**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**15**

**Write parent value**

**11**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**16**

**Write parent value**

**11**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**17**

**Write parent value**

**16**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**2**

**Write tree node value**

**18**

**Write parent value**

**12**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**5**

**1.00**

**2.00**

**3.00**

**5.00**

**8.00**

**9.00**

**6.00**

**10.00**

**7.00**

**4.00**

**11.00**

**15.00**

**16.00**

**17.00**

**13.00**

**12.00**

**18.00**

**14.00**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**4**

**tree grade is 4**

**1. Create tree 2. Add node to tree 3. Delete node from tree 4. Task 5. Print tree 6. Exit**

**6**

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. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  | или |  |  |  |  |  |
|  | дом. |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. **Замечания автора** по существу работы

1. **Выводы**

Я научился работать с динамическими структурами и обрабатывать деревья

Недочёты при выполнении задания могут быть устранены следующим образом: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_