**Отчет по лабораторной работе №** 23 по курсу 1

Студент группы М80-108Б-18, Коростелев Д.В. № по списку 12

E-mail [dmitry.k48@yandex.ru](mailto:dmitry.k48@yandex.ru)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Работа выполнена: « 23 » февраля 2019 г.

Преподаватель: Поповкин А.В. каф.806 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Отчет сдан « » \_\_\_\_\_\_\_\_\_ 201\_\_ 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 или упорядоченного двоичного дерева, содержащего узлы типа float. Основные функции работы с деревьями реализовать в виде универсальных процедур и функций.
3. **Задание (***вариант №* **):** Реализовать процедуру добавления нового узла, текстовой визуализации дерева, удаление узла, определения – является ли дерево – линейным списком вершин.
4. **Оборудование (лабораторное):**

ЭВМ Cameron, процессор Intel Core2 Duo CPU E8500 3.16ГГц, имя узла сити cameron с ОП 16029 Мб, НДМ \_\_\_\_\_\_\_\_\_ Гб. Терминал gnome, адрес 172.16.80.213. Принтер \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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

*Оборудование ПЭВМ студента, если использовалось:*

Процессор Intel Core i7-7700HQ 4x 2.808ГГц с ОП 3945 Мб, НДМ 48.9 Гб Монитор \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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

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

Операционная система семейства UNIX , наименование UBUNTU версия \_16.04\_\_\_\_\_\_\_\_\_\_\_\_, интерпретатор команд bash версия 4.3.48 , Система программирования \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ версия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Редактор текстов \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ версия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Утилиты операционной системы: cd, cat, nano, cc\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Прикладные системы и программы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Местонахождение и имена файлов программ и данных: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *Программное обеспечение ЭВМ студента, если использовалось:*

Операционная система семейства UNIX, наименование UBUNTU\_\_\_ версия \_\_\_\_\_\_18.04\_\_\_\_\_\_\_, интерпретатор команд \_\_\_bash\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ версия\_\_\_\_\_\_4.3.48\_\_\_\_\_\_\_

Система программирования \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ версия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Редактор текстов \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ версия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Утилиты операционной системы: cd, cat, nano, cc.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Прикладные системы и программы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Местонахождение и имена файлов программ и данных на домашнем компьютере: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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

Для решения данной лабораторной работы требуется реализовать ряд специальных функций по обработке деревьев такие как: функция создания дерева, добавления/удаления узла, текстовой визуализации, и функции, выполняющей определённую задачу, в соответствии с вариантом, также стоит написать функцию-меню, с помощью которой пользователь сможет сам выбирать какое действие нужно выполнить.

Функция-меню – реализуем через конструкцию switch-case, далее, определим функцию создания дерева. Эту функция будет просить ввести корень дерева, далее рекурсивно, в зависимости от того введено есть ли потомки у нынешней вершины, рекурсивно будем запрашивать ввод, остального дерева. В итоге функция создания дерева будет состоять из двух подфункций – функция ввода корня дерева, и рекурсивная функция ввода остал ьных ветвей.

Функция текстовой визуализации будет рекурсивно выводить ветви дерева с указанием номера вершины, а также каким потомком, левым или правым, является данная вершина своему родителю.

Функция добавления узла будет работать совместно с функцией визуализации. Программа сама выведет нынешнее дерево. Далее пользователь должен выбрать в какую ветку добавить новый узел, и в зависимости от того какой узел сейчас свободен вставить требуемое значение. Функция найдет нужную вершину и запишет в нее значение.

Функция удаления узла, схожа с функцией добавления узла. Пользователь вводит номер вершины, сама функция ищет нужную и вершину, а затем удаляет ее.

Функция линейности вершин, смотрит если у корня есть хотя бы один потомок, то список вершин дерева – линейный список вершин, иначе – нет.

1. **Сценарий выполнения работы** (план работы, первоначальный текст программы на черновике или отдельном листе, тесты либо соображения по тестированию)

|  |  |  |
| --- | --- | --- |
| **№** | **Функция** | **Описание** |
| 1. | void Menu() | Функция меню, не принимает никаких аргументов, выводит меню программы. |
| 2. | void ShowTree(struct tnode\* tree, int\* counter, int space, char side) | Рекурсивная функция показа дерева, принимает 4 аргумента – указатель на вершину, переменную-счетчик, с помощью которой будет отображаться номер вершины, аргумент глубины, с помощью которого, в зависимости от глубины вершины будет выводиться то или иное кол-во пробелов, и символьную переменную названия ветви. |
| 3. | struct tnode\* \_CreateMatrix(float x, struct tnode \*tree) | Подфункция главной функции создании дерева, принимает значение родителя текущей вершины (нужен формально, используется для создания интерфейса), указатель на вершину. |
| 4. | struct tnode\* CreateMatrix(struct tnode \*tree) | Функция создания дерева, принимает указатель на дерево, запускает рекурсивную функцию записи дерева |
| 5. | struct tnode\* \_CreateNode(struct tnode\* tree) | Функция, которая запрашивает ввод значения для записи в не инициализированную вершину |
| 6. | struct tnode\* \_AddNode(struct tnode\* tree, int \*counter) | Подфункция функции добавления узла, осуществляет поиск нужного узла, принимает указатель на вершину и номер вершины |
| 7. | void AddNode(struct tnode\* tree, int counter) | Функция добавления узла, в ней определяются все функция функциональные переменные, а потом запускает рекурсивную функцию \_AddNode, аргументы: указатель на вершину, номер вершины |
| 8. | void Function(struct tnode\* tree) | Функция выполняет задание 12-ого варианта, принимает указатель на вершину |
| 9. | struct tnode\* \_DeleteNode(struct tnode\* tree) | Функция, которая удаляет вершину и ее потомков |
| 10. | struct tnode\* \_fDeleteNode(struct tnode\* tree, int\* counter) | Функция осуществляет поиск нужной вершины по счетчику |
| 11. | void DeleteNode(struct tnode\* tree) | Функция удаления узла, запускает функции \_fDeleteNode и \_DeleteNode |

*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, подписанный преподавателем)

|  |
| --- |
| dmitry@dmitry-VirtualBox:~/lubs/23$ cat 12\_23.c  #include<stdio.h>  #include<stdlib.h>  #include<stdbool.h>  #include<locale.h>  #include<malloc.h>  //Описание узла дерева  struct tnode{  float Value;  struct tnode \*left;  struct tnode \*right;  };  //Функция текстовой визуализации дерева  void ShowTree(struct tnode \*tree, int \*counter,int space, char side){  if(tree)  {  int buf = \*counter;  printf("%d : ",buf+1);  for(int i = 0;i<space;i++)  {  printf(" ");  }  printf("%c:%.3f\n",side, tree->Value);  buf++;  \*counter = buf;  space++;  ShowTree(tree->left, counter, space,'L');  ShowTree(tree->right, counter, space,'R');  }  }  //Подфункция функции создания дерева  struct tnode\* \_CreateMatrix(float x, struct tnode \*tree){  float value;  if((scanf("%f",&value)) == 1)  {  tree = (struct tnode\*)malloc(sizeof(struct tnode));  tree->Value = value;  printf("Введите левого потомка %.3f: ", value);  tree->left = \_CreateMatrix(value, tree);  printf("Введите правого потомка %.3f: ", value);  tree->right = \_CreateMatrix(value, tree);  }  else{  tree = (struct tnode\*)malloc(sizeof(struct tnode));  tree = 0;  }  return (tree);  }  //Функция создания дерева  struct tnode\* CreateMatrix(struct tnode \*tree){  float value;  printf("Если у вершины нет потомка, введите : N\n");  printf("Введите первый узел дерева: ");  if((scanf("%f",&value)) == 1 )  {  tree->Value = value;  printf("Введите левого потомка %.3f: ", value);  tree->left = \_CreateMatrix(value, tree);  printf("Введите правого потомка %.3f: ", value);  tree->right = \_CreateMatrix(value, tree);  }  return (tree);  }  //Функция записис узла  struct tnode\* \_CreateNode(struct tnode\* tree){  float value;  printf("Введите новый узел: ");  scanf("%f", &value);  tree->Value = value;  tree->left = 0;  tree->right = 0;  return (tree);  }  //Подфункция функции добавления узла в дерево  struct tnode\* \_AddNode(struct tnode\* tree, int \*counter){  int buf = \*counter;  if(buf > 0){  buf--;  \*counter = buf;  if(tree->left){  tree->left = \_AddNode(tree->left, counter);  }  if(tree->right){  tree->right = \_AddNode(tree->right, counter);  }  }  else if(buf == 0)  {  if(tree->left == 0){  buf--;  \*counter = buf;  tree->left = (struct tnode\*)malloc(sizeof(struct tnode));  tree->left = \_CreateNode(tree->left);  }  else if(tree->right == 0){  buf--;  \*counter = buf;  tree->right = (struct tnode\*)malloc(sizeof(struct tnode));  tree->right = \_CreateNode(tree->right);  }  }  return (tree);  }  //Функция добавления узла в дерево  void AddNode(struct tnode\* tree, int counter){  printf("Введите номер узла к которому хотите добавить новый узел: ");  int\* n\_counter;  n\_counter = (int\*)malloc(sizeof(int));  \*n\_counter = counter;  int\* n\_node;  n\_node = (int\*)malloc(sizeof(int));  int input;  scanf("%d", &input);  \*n\_node = input-1;  tree = \_AddNode(tree,n\_node);  }  //Задание варианта : проверить, является ли дерево линейным списком врешин  void Function(struct tnode\* tree){  if(tree->left || tree->right){  printf("Данное дерево является линейным списком вершин\n");  }  else{  printf("Данное дерево не явлется линейным списком вершин\n");  }  }  //Функция очистики всего поддерева узла  struct tnode\* \_DeleteNode(struct tnode\* tree){  if(tree->left){  tree->left = \_DeleteNode(tree->left);  }  if(tree->right){  tree->right = \_DeleteNode(tree->right);  }  tree = 0;  free(tree);  return (tree);  }  //Функция поиска нужного узла  struct tnode\* \_fDeleteNode(struct tnode\* tree, int\* counter){  int buf = \*counter;  if(buf > 0){  buf--;  \*counter = buf;  if(tree->left){  tree->left = \_fDeleteNode(tree->left, counter);  }  if(tree->right){  tree->right = \_fDeleteNode(tree->right, counter);  }  }  else if(buf == 0)  {  buf--;  \*counter = buf;  tree = \_DeleteNode(tree);  }  return (tree);  }  //Функция удаления узла  void DeleteNode(struct tnode\* tree){  int \*n\_node;  n\_node = (int\*)malloc(sizeof(int));  int input;  printf("\nВведите номер узла, который хотите удалить: ");  scanf("%d",&input);  \*n\_node = input-1;  tree = \_fDeleteNode(tree, n\_node);  }  void Menu(){  printf("╔═══════════════════════════════════════════════════════════════════════╗\n");  printf("1 : Создание дерева ║\n");  printf("2 : Добавление новго узла. ║\n");  printf("3 : Текстовая визуализация дерева. ║\n");  printf("4 : Удаление узла. ║\n");  printf("5 : Проверить, является ли двоичное дерево линейным списком вершин. ║\n");  printf("6 : Закрыть программу. ║\n");  printf("╚═══════════════════════════════════════════════════════════════════════╝");  printf("\nВыберите действие: ");  }  int main(void){  setlocale(LC\_ALL, "rus");  int input;  int\* counter;  struct tnode\* tree;  tree = (struct tnode\*)malloc(sizeof(struct tnode));  while(1){  Menu();  scanf("%d", &input);  printf("\n");  switch (input){  case 1:  printf("\n");  tree = CreateMatrix(tree);  printf("\n\n");  break;  case 2:  printf("\n");  counter = (int\*)malloc(sizeof(int));  \*counter = 0;  ShowTree(tree, counter, 0,'O');  printf("\n");  int b\_counter = \*counter;  AddNode(tree, b\_counter);  printf("\n");  break;  case 3:  printf("\n");  counter = (int\*)malloc(sizeof(int));  \*counter = 0;  ShowTree(tree, counter, 0,'O');  printf("\n");  free(counter);  break;  case 4:  printf("\n");  counter = (int\*)malloc(sizeof(int));  \*counter = 0;  ShowTree(tree, counter, 0,'O');  DeleteNode(tree);  printf("\n");  break;  case 5:  printf("\n");  counter = (int\*)malloc(sizeof(int));  \*counter = 0;  ShowTree(tree, counter, 0,'O');  printf("\n");  Function(tree);  free(counter);  printf("\n");  break;  case 6:  printf("Закрытие программы...\n\n");  free(tree);  return 0;  default:  printf("\nКоманда не определена, повторите ввод.\n");  break;  }  }  return 0;  }  dmitry@dmitry-VirtualBox:~/lubs/23$ nano 12\_23.out  dmitry@dmitry-VirtualBox:~/lubs/23$ ./12\_3.out  bash: ./12\_3.out: No such file or directory  dmitry@dmitry-VirtualBox:~/lubs/23$ ./12\_23.out  ╔═══════════════════════════════════════════════════════════════════════╗  1 : Создание дерева ║  2 : Добавление новго узла. ║  3 : Текстовая визуализация дерева. ║  4 : Удаление узла. ║  5 : Проверить, является ли двоичное дерево линейным списком вершин. ║  6 : Закрыть программу. ║  ╚═══════════════════════════════════════════════════════════════════════╝  Выберите действие: 1  Если у вершины нет потомка, введите : N  Введите первый узел дерева: 12  Введите левого потомка 12.000: 23  Введите левого потомка 23.000: n  Введите правого потомка 23.000: n  Введите правого потомка 12.000: 34  Введите левого потомка 34.000: 56  Введите левого потомка 56.000: 43  Введите левого потомка 43.000: n  Введите правого потомка 43.000: n  Введите правого потомка 56.000: 21  Введите левого потомка 21.000: n  Введите правого потомка 21.000: n  Введите правого потомка 34.000: 32  Введите левого потомка 32.000: n  Введите правого потомка 32.000: n  ╔═══════════════════════════════════════════════════════════════════════╗  1 : Создание дерева ║  2 : Добавление новго узла. ║  3 : Текстовая визуализация дерева. ║  4 : Удаление узла. ║  5 : Проверить, является ли двоичное дерево линейным списком вершин. ║  6 : Закрыть программу. ║  ╚═══════════════════════════════════════════════════════════════════════╝  Выберите действие: 3  1 : O:12.000  2 : L:23.000  3 : R:34.000  4 : L:56.000  5 : L:43.000  6 : R:21.000  7 : R:32.000  ╔═══════════════════════════════════════════════════════════════════════╗  1 : Создание дерева ║  2 : Добавление новго узла. ║  3 : Текстовая визуализация дерева. ║  4 : Удаление узла. ║  5 : Проверить, является ли двоичное дерево линейным списком вершин. ║  6 : Закрыть программу. ║  ╚═══════════════════════════════════════════════════════════════════════╝  Выберите действие: 2  1 : O:12.000  2 : L:23.000  3 : R:34.000  4 : L:56.000  5 : L:43.000  6 : R:21.000  7 : R:32.000  Введите номер узла к которому хотите добавить новый узел: 7  Введите новый узел: 34.6  ╔═══════════════════════════════════════════════════════════════════════╗  1 : Создание дерева ║  2 : Добавление новго узла. ║  3 : Текстовая визуализация дерева. ║  4 : Удаление узла. ║  5 : Проверить, является ли двоичное дерево линейным списком вершин. ║  6 : Закрыть программу. ║  ╚═══════════════════════════════════════════════════════════════════════╝  Выберите действие:  3  1 : O:12.000  2 : L:23.000  3 : R:34.000  4 : L:56.000  5 : L:43.000  6 : R:21.000  7 : R:32.000  8 : L:34.600  ╔═══════════════════════════════════════════════════════════════════════╗  1 : Создание дерева ║  2 : Добавление новго узла. ║  3 : Текстовая визуализация дерева. ║  4 : Удаление узла. ║  5 : Проверить, является ли двоичное дерево линейным списком вершин. ║  6 : Закрыть программу. ║  ╚═══════════════════════════════════════════════════════════════════════╝  Выберите действие: 4  1 : O:12.000  2 : L:23.000  3 : R:34.000  4 : L:56.000  5 : L:43.000  6 : R:21.000  7 : R:32.000  8 : L:34.600  Введите номер узла, который хотите удалить: 4  ╔═══════════════════════════════════════════════════════════════════════╗  1 : Создание дерева ║  2 : Добавление новго узла. ║  3 : Текстовая визуализация дерева. ║  4 : Удаление узла. ║  5 : Проверить, является ли двоичное дерево линейным списком вершин. ║  6 : Закрыть программу. ║  ╚═══════════════════════════════════════════════════════════════════════╝  Выберите действие: 3  1 : O:12.000  2 : L:23.000  3 : R:34.000  4 : R:32.000  5 : L:34.600  ╔═══════════════════════════════════════════════════════════════════════╗  1 : Создание дерева ║  2 : Добавление новго узла. ║  3 : Текстовая визуализация дерева. ║  4 : Удаление узла. ║  5 : Проверить, является ли двоичное дерево линейным списком вершин. ║  6 : Закрыть программу. ║  ╚═══════════════════════════════════════════════════════════════════════╝  Выберите действие: 5  1 : O:12.000  2 : L:23.000  3 : R:34.000  4 : R:32.000  5 : L:34.600  Данное дерево является линейным списком вершин  ╔═══════════════════════════════════════════════════════════════════════╗  1 : Создание дерева ║  2 : Добавление новго узла. ║  3 : Текстовая визуализация дерева. ║  4 : Удаление узла. ║  5 : Проверить, является ли двоичное дерево линейным списком вершин. ║  6 : Закрыть программу. ║  ╚═══════════════════════════════════════════════════════════════════════╝  Выберите действие: 1  Если у вершины нет потомка, введите : N  Введите первый узел дерева: 12  Введите левого потомка 12.000: n  Введите правого потомка 12.000: n  ╔═══════════════════════════════════════════════════════════════════════╗  1 : Создание дерева ║  2 : Добавление новго узла. ║  3 : Текстовая визуализация дерева. ║  4 : Удаление узла. ║  5 : Проверить, является ли двоичное дерево линейным списком вершин. ║  6 : Закрыть программу. ║  ╚═══════════════════════════════════════════════════════════════════════╝  Выберите действие: 5  1 : O:12.000  Данное дерево не явлется линейным списком вершин  ╔═══════════════════════════════════════════════════════════════════════╗  1 : Создание дерева ║  2 : Добавление новго узла. ║  3 : Текстовая визуализация дерева. ║  4 : Удаление узла. ║  5 : Проверить, является ли двоичное дерево линейным списком вершин. ║  6 : Закрыть программу. ║  ╚═══════════════════════════════════════════════════════════════════════╝  Выберите действие: 6  Закрытие программы... |

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 | Дом. | 29.03.2019 | 12.50 | Неправильная запись дерева | Окаймил аргумент функции return в скобки |  |

1. **Замечания автора по существу работы:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. **Выводы:** благодаря данной лабораторной работе студент может научиться базовым навыкам обработки деревьев и взаимодействия с ними, а также выучить новую специализированную информацию, терминологию по данной теме.

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

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