**Отчет по лабораторной работе № 23** по курсу   
"Фундаментальная информатика"

Студент группы М8О-109Б-20 Чувилин Виталий Антонович, № по списку 19

Контакты e-mail: lemonvitaliy@mail.ru

Работа выполнена: « 25 » апреля\_\_2021 г.   
  
Преподаватель: каф. 806 Титов В.К.

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

Отчет сдан « » 2021 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   
  
**3.** **Задание** (*вариант № 34*)**:** Определить уровень дерева, на котором находится максимальное число вершин  
  
**4. Оборудование:***Оборудование ПЭВМ студента, если использовалось:*

Процессор AMD Ryzen 5 3500U с ОП 8 Гб, НМД 256 Гб.   
Монитор встроенный  
Другие устройства не использовались  
  
**5. Программное обеспечение:***Программное обеспечение ЭВМ студента, если использовалось:*

Операционная система семейства GNU\Linux, наименование Ubuntu 20.04 интерпретатор команд bash версия 5.0.17(1)-release  
Система программирования Не использовалась   
Редактор текстов LibreOffice

Утилиты

Местонахождение и имена файлов программ и данных   
  
**6. Идея, метод, алгоритм**

Для нахождения ответа будем использовать классический обход в глубину. Для начала посчитаем глубину дерева, путём обхода с соответствующим параметром. Далее зададим массив, в котором будет считать количество вершин на уровне. Начинаем обход из корня с параметром 1 и будем переходить во всех братьев с тем же параметром и во всех сыновей с параметром, увеличенным на единицу, параллельно подсчитывая количество вершин в массив заданный ранее. Теперь нам осталось пройтись по массиву, найти уровень с максимальным числом вершин и запомнить ответ.

Функция, решающая задачу, реализована отдельно и может быть найдена в меню программы под пунктом (7).

**7. Сценарий выполнения работы**

Код слишком большой и нечитаемый при прямом копировании в этот пункт, поэтому он будет подкреплен отдельным текстовым файлом и отображен в протоколе.

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

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

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

**user@magickbook14:~/study/labs/23 $ cat tree.cpp    
#include <stdio.h>   
#include <stdlib.h>   
#include <time.h>   
  
typedef int tdata;   
  
struct node;   
typedef node \* link;   
  
struct node{   
   tdata data;   
   link son;   
   link brother;   
};   
  
// Search node and prevnode by data   
void search\_node(tdata node, link \*pos, link \*prevpos, link cur, link prev, bool \*error){   
   if (cur->data == node) {   
       \*pos = cur;   
       \*prevpos = prev;   
       \*error = false;   
       return;   
   }   
   if (cur->son) search\_node(node, pos, prevpos, cur->son, cur, error);   
   if (cur->brother) search\_node(node, pos, prevpos, cur->brother, cur, error);   
}   
  
// Add son by father and son data   
void add\_son(tdata father, tdata son, link tree){   
  
   link cur, prev;   
   bool error = true;   
   search\_node(father, &cur, &prev, tree, tree, &error);   
  
   if (error){   
       printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");   
       printf("Wrong info\n");   
       return;   
   }   
  
   if (cur->data == father){   
      
       if (cur->son == NULL){   
           link new\_node = (link)malloc(sizeof(link));   
           new\_node->data = son;   
           new\_node->brother = NULL;   
           new\_node->son = NULL;   
           cur->son = new\_node;   
       }   
       else{   
           cur = cur->son;   
           while(cur->brother != NULL) cur = cur->brother;   
           link new\_node = (link)malloc(sizeof(link));   
           new\_node->data = son;   
           new\_node->brother = NULL;   
           new\_node->son = NULL;   
           cur->brother = new\_node;   
       }   
  
   }   
}   
  
// Removing process   
void removing(link cur){   
   if (cur->son){   
       removing(cur->son);   
   }   
   if (cur->brother){   
       removing(cur->brother);   
   }   
  
   cur->son = NULL;       
   cur->brother = NULL;   
   cur->data = 0;   
   free(cur);   
}   
  
// Remove subtree by data   
void remove\_node(tdata node, link tree){   
   link cur, prev;   
   bool error = true;   
   search\_node(node, &cur, &prev, tree, tree, &error);   
  
   if (error) {   
       printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");   
       printf("Wrong info\n");   
       return;   
   }   
  
   if (cur->son) {   
       removing(cur->son);   
       cur->son = NULL;   
   }   
  
   if (prev->brother && prev->brother->data == cur->data){   
       prev->brother = cur->brother;   
   }   
   else {   
       prev->son = cur->brother;   
   }   
  
   cur->son = NULL;   
   cur->brother = NULL;   
   cur->data = 0;   
   free(cur);   
}   
  
// Generate random tree witn n nodes   
void generate\_tree(link cur){   
   if (!cur->son && !cur->brother){   
       int flip = rand() % 2;   
  
       if (flip){   
           link new\_node = (link)malloc(sizeof(link));   
           new\_node->data = rand() % 20 + 1;   
           new\_node->son = NULL;   
           new\_node->brother = NULL;   
           cur->son = new\_node;   
       }   
       else{   
           link new\_node = (link)malloc(sizeof(link));   
           new\_node->data = rand() % 20 + 1;   
           new\_node->son = NULL;   
           new\_node->brother = NULL;   
           cur->brother = new\_node;   
       }   
       return;   
   }   
  
   int choose = rand() % 2;   
  
   if (choose){   
       if (cur->son) generate\_tree(cur->son);   
       else {   
           link new\_node = (link)malloc(sizeof(link));   
           new\_node->data = rand() % 20 + 1;   
           new\_node->son = NULL;   
           new\_node->brother = NULL;   
           cur->son = new\_node;   
       }   
   }    
   else{   
       if (cur->brother) generate\_tree(cur->brother);   
       else{   
           link new\_node = (link)malloc(sizeof(link));   
           new\_node->data = rand() % 20 + 1;   
           new\_node->son = NULL;   
           new\_node->brother = NULL;   
           cur->brother = new\_node;   
       }   
   }   
}   
  
// Set tree root   
void set\_root(link \*root){   
   printf("Input root value -> ");   
   int root\_value;   
   scanf("%d", &root\_value);   
  
   link new\_root;   
   new\_root = (link)malloc(sizeof(link));   
      
   new\_root->son = NULL;   
   new\_root->brother = NULL;   
   new\_root->data = root\_value;   
  
   \*root = new\_root;   
}   
  
// Set random tree root   
void set\_random\_root(link\* root) {   
  
   link new\_root;   
   new\_root = (link)malloc(sizeof(link));   
   new\_root->brother = NULL;   
   new\_root->data = rand() % 20 + 1;   
  
  
   link new\_node;   
  
   new\_node = (link)malloc(sizeof(link));   
   new\_node->son = NULL;   
   new\_node->brother = NULL;   
   new\_node->data = rand() % 20 + 1;   
  
   new\_root->son = new\_node;   
  
   \*root = new\_root;      
}   
  
// Remove whole tree   
void remove\_tree(link root, bool \*empty){   
   printf("You are going to remove whole tree. Are you sure? y/n\n");   
   char check;   
   scanf(" %c", &check);   
  
   if (check != 'y') {   
       printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");   
       printf("Canceled");   
       return;   
   }   
   \*empty = true;   
  
   if (root->son){   
       remove\_node(root->son->data, root);   
   }   
      
   root->data = 0;   
   root->son = NULL;   
   root->brother = NULL;   
   free(root);   
  
   printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");   
   printf("Tree was successfully removed\n");   
}   
  
// Tree visualisation   
void print\_tree(link cur, int n){   
   if (cur->son) print\_tree(cur->son, n + 1);   
   if (cur->brother) print\_tree(cur->brother, n);   
      
   for (int i = 0; i < n; i++) printf("   ");   
   printf("\\\_\_%d\n", cur->data);       
}   
  
// Find deep of tree   
void find\_deep(int \*deep, int n, link cur){   
   if (n > \*deep){   
       \*deep = n;    
   }   
  
   if (cur->son){   
       find\_deep(deep, n + 1, cur->son);   
   }   
   if (cur->brother){   
       find\_deep(deep, n, cur->brother);   
   }   
}   
  
// Function that calculating level with the most nodes   
void function\_34(int \*levels, int n, link cur){   
   levels[n]++;   
  
   if (cur->son){   
       function\_34(levels, n + 1, cur->son);   
   }   
   if (cur->brother){   
       function\_34(levels, n, cur->brother);   
   }   
}   
  
//------------------------------------------------------------   
int main(){   
      
   srand(time(NULL));   
  
   link tree;   
   bool empty = true, exit = false;   
   int key;   
  
   while(1){   
       printf("\n");   
       printf("\_\_\_\_\_\_\_\_\_\_Menu\_\_\_\_\_\_\_\_\_\_\n");   
       printf("1. Print tree\n");   
       printf("2. Generate random tree\n");   
       printf("3. Set root\n");   
       printf("4. Add son\n");   
       printf("5. Remove node\n");   
       printf("6. Remove whole tree\n");   
       printf("7. Find level with the most nodes\n");   
       printf("8. Exit\n");   
  
  
       printf("Input key -> ");   
       scanf("%d", &key);   
       printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");   
  
       switch (key){   
           // 1. Print tree   
           case 1:   
               if (!empty){   
                   print\_tree(tree, 0);   
               }   
               else{   
                   printf("Tree is empty\n");   
               }   
               break;   
           // 2. Generate random tree   
           case 2:   
               if (empty){   
                   printf("Input number of nodes -> ");   
                   int n;   
                   scanf("%d", &n);   
  
                   if (n < 2){   
                       printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");   
                       printf("Number must be greater then 1\n");   
                       break;   
                   }   
  
                   set\_random\_root(&tree);   
  
                   n -= 2;   
                   for (int i = 0; i < n; i++){   
                       generate\_tree(tree->son);   
                   }   
  
                   printf("Tree generated\n");   
  
                   empty = false;   
               }   
               else{   
                   printf("Tree already exists. Want to generated new? y/n\n");   
                   char check;   
                   scanf(" %c", &check);   
                      
                   if (check == 'y'){   
                       printf("Input number of nodes -> ");   
                       int n;   
                       scanf("%d", &n);   
  
                       if (n < 2) {   
                           printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");   
                           printf("Number must be greater then 1\n");   
                           break;   
                       }   
  
                       remove\_tree(tree, &empty);   
                       if (!empty) break;   
                       set\_random\_root(&tree);   
  
                       n -= 2;   
                       for (int i = 0; i < n; i++) {   
                           generate\_tree(tree->son);   
                       }   
  
                       empty = false;   
                       printf("Tree generated\n");   
                   }   
                   else{   
                       printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");   
                       printf("Canceled");   
                   }   
               }   
               break;   
           // 3. Set root   
           case 3:   
               if (empty){   
                   set\_root(&tree);   
  
                   empty = false;   
               }   
               else{   
                   printf("Root already exists\n");   
               }   
      
               break;   
           // 4. Add son   
           case 4:   
               if (!empty){   
                   printf("Input father <space> son -> ");   
                   int father, son;   
                   scanf("%d%d", &father, &son);   
                      
                   add\_son(father, son, tree);   
               }   
               else{   
                   printf("Tree is empty\n");   
               }   
  
               break;   
           // 5. Remove node   
           case 5:   
               if (!empty){   
                   printf("Input node -> ");   
                   int node;   
                   scanf("%d", &node);   
  
                   if (node == tree->data) {   
                       remove\_tree(tree, &empty);   
                   }   
                   else{   
                       remove\_node(node, tree);   
                   }   
               }   
               else{   
                   printf("Tree is empty\n");   
               }   
               break;   
           // 6. Remove whole tree   
           case 6:   
               if (!empty){   
                   remove\_tree(tree, &empty);   
               }   
               else{   
                   printf("Tree is empty\n");   
               }   
               break;   
           // 7. Find level with the most nodes   
           case 7:   
               if (!empty){   
                   int deep = 1;   
                   find\_deep(&deep, 1, tree);   
  
                   int levels[deep + 1];   
                   for (int i = 1; i < deep + 1; i++) {   
                       levels[i] = 0;   
                   }   
  
                   function\_34(levels, 1, tree);   
  
                   int mx = 0;   
                   int ans = 1;   
                   for (int i = 1; i < deep + 1; i++){   
                       if (levels[i] > mx){   
                           mx = levels[i];   
                           ans = i;   
                       }   
                   }   
  
                   printf("Level with the most nodes is: %d\n", ans);   
               }   
               else{   
                   printf("Tree is empty\n");   
               }   
               break;   
           // 8. Exit   
           case 8:   
               exit = true;   
               break;   
       }   
  
       if (exit) break;   
   }   
}user@magickbook14:~/study/labs/23 $ g++ tree.cpp && ./a.out   
  
\_\_\_\_\_\_\_\_\_\_Menu\_\_\_\_\_\_\_\_\_\_   
1. Print tree   
2. Generate random tree   
3. Set root   
4. Add son   
5. Remove node   
6. Remove whole tree   
7. Find level with the most nodes   
8. Exit   
Input key -> 1   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
Tree is empty   
  
\_\_\_\_\_\_\_\_\_\_Menu\_\_\_\_\_\_\_\_\_\_   
1. Print tree   
2. Generate random tree   
3. Set root   
4. Add son   
5. Remove node   
6. Remove whole tree   
7. Find level with the most nodes   
8. Exit   
Input key -> 2   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
Input number of nodes -> 12   
Tree generated   
  
\_\_\_\_\_\_\_\_\_\_Menu\_\_\_\_\_\_\_\_\_\_   
1. Print tree   
2. Generate random tree   
3. Set root   
4. Add son   
5. Remove node   
6. Remove whole tree   
7. Find level with the most nodes   
8. Exit   
Input key -> 1   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
        \\_\_12   
        \\_\_15   
     \\_\_11   
     \\_\_19   
        \\_\_15   
        \\_\_19   
     \\_\_1   
     \\_\_11   
  \\_\_5   
  \\_\_20   
  \\_\_9   
\\_\_5   
  
\_\_\_\_\_\_\_\_\_\_Menu\_\_\_\_\_\_\_\_\_\_   
1. Print tree   
2. Generate random tree   
3. Set root   
4. Add son   
5. Remove node   
6. Remove whole tree   
7. Find level with the most nodes   
8. Exit   
Input key -> 7   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
Level with the most nodes is: 3   
  
\_\_\_\_\_\_\_\_\_\_Menu\_\_\_\_\_\_\_\_\_\_   
1. Print tree   
2. Generate random tree   
3. Set root   
4. Add son   
5. Remove node   
6. Remove whole tree   
7. Find level with the most nodes   
8. Exit   
Input key -> 6   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
You are going to remove whole tree. Are you sure? y/n   
y   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
Tree was successfully removed   
  
\_\_\_\_\_\_\_\_\_\_Menu\_\_\_\_\_\_\_\_\_\_   
1. Print tree   
2. Generate random tree   
3. Set root   
4. Add son   
5. Remove node   
6. Remove whole tree   
7. Find level with the most nodes   
8. Exit   
Input key -> 2   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
Input number of nodes -> 10   
Tree generated   
  
\_\_\_\_\_\_\_\_\_\_Menu\_\_\_\_\_\_\_\_\_\_   
1. Print tree   
2. Generate random tree   
3. Set root   
4. Add son   
5. Remove node   
6. Remove whole tree   
7. Find level with the most nodes   
8. Exit   
Input key -> 1   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
        \\_\_10   
     \\_\_11   
     \\_\_6   
     \\_\_12   
     \\_\_12   
  \\_\_17   
  \\_\_14   
  \\_\_7   
  \\_\_7   
\\_\_4   
  
\_\_\_\_\_\_\_\_\_\_Menu\_\_\_\_\_\_\_\_\_\_   
1. Print tree   
2. Generate random tree   
3. Set root   
4. Add son   
5. Remove node   
6. Remove whole tree   
7. Find level with the most nodes   
8. Exit   
Input key -> 7   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
Level with the most nodes is: 2   
  
\_\_\_\_\_\_\_\_\_\_Menu\_\_\_\_\_\_\_\_\_\_   
1. Print tree   
2. Generate random tree   
3. Set root   
4. Add son   
5. Remove node   
6. Remove whole tree   
7. Find level with the most nodes   
8. Exit   
Input key -> 2   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
Tree already exists. Want to generated new? y/n   
3   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
Canceled   
\_\_\_\_\_\_\_\_\_\_Menu\_\_\_\_\_\_\_\_\_\_   
1. Print tree   
2. Generate random tree   
3. Set root   
4. Add son   
5. Remove node   
6. Remove whole tree   
7. Find level with the most nodes   
8. Exit   
Input key -> 2   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
Tree already exists. Want to generated new? y/n   
y   
Input number of nodes -> 3   
You are going to remove whole tree. Are you sure? y/n   
y   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
Tree was successfully removed   
Tree generated   
  
\_\_\_\_\_\_\_\_\_\_Menu\_\_\_\_\_\_\_\_\_\_   
1. Print tree   
2. Generate random tree   
3. Set root   
4. Add son   
5. Remove node   
6. Remove whole tree   
7. Find level with the most nodes   
8. Exit   
Input key -> 1   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
     \\_\_16   
  \\_\_15   
\\_\_1   
  
\_\_\_\_\_\_\_\_\_\_Menu\_\_\_\_\_\_\_\_\_\_   
1. Print tree   
2. Generate random tree   
3. Set root   
4. Add son   
5. Remove node   
6. Remove whole tree   
7. Find level with the most nodes   
8. Exit   
Input key -> 7   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
Level with the most nodes is: 1   
  
\_\_\_\_\_\_\_\_\_\_Menu\_\_\_\_\_\_\_\_\_\_   
1. Print tree   
2. Generate random tree   
3. Set root   
4. Add son   
5. Remove node   
6. Remove whole tree   
7. Find level with the most nodes   
8. Exit   
Input key -> 6   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
You are going to remove whole tree. Are you sure? y/n   
y   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
Tree was successfully removed   
  
\_\_\_\_\_\_\_\_\_\_Menu\_\_\_\_\_\_\_\_\_\_   
1. Print tree   
2. Generate random tree   
3. Set root   
4. Add son   
5. Remove node   
6. Remove whole tree   
7. Find level with the most nodes   
8. Exit   
Input key -> 1   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
Tree is empty   
  
\_\_\_\_\_\_\_\_\_\_Menu\_\_\_\_\_\_\_\_\_\_   
1. Print tree   
2. Generate random tree   
3. Set root   
4. Add son   
5. Remove node   
6. Remove whole tree   
7. Find level with the most nodes   
8. Exit   
Input key -> 8   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
user@magickbook14:~/study/labs/23 $**

**9. Дневник отладки.  
  
 Д**олжен 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 |
|  |  |  |  |  |  |  |

**10. Замечания автора** по существу работы: Код был написан с чистого листа лично мною. Я потратил на эту работу несколько дней и доволен результатом. Считаю, что получилось достойно. Вся программа реализована в виде множественных функций для удобства восприятия кода. Все функции тщательно тестировались и доводились до совершенства. Все случаи ввода пользователем учтены. Сбоев в работе программы замечено не было.

**11. Выводы**

Выполняю эту работу, я разобрался в том, что такое динамические структуры данных и как с ними работать. Параллельно с этим, я получил невероятный опыт в написании и отладке больших программ и написании понятного пользовательского интерфейса. Максимально доволен проделанной работой.

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

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