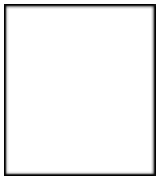

Отчёт по лабораторной работе № 23 по курсу 1

Практикум на ЭВМ

студента группы М80-104б-18 Сыроежкина Кирилла Геннадьевича, № по списку 18

Адреса www, e-mail, jabber, skype KrillsA@yandex.ru

Работа выполнена: "01" марта 2019г.

Преподаватель: Доцент каф.806 Никулин С.П.

Входной контроль знаний с оценкой

Отчёт сдан " " 2019 г., итоговая оценка

Подпись преподавателя

- **Тема**: Динамические структуры данных

- **Цель работы**: Составить программу на языке Си для построения и обработки двоичного дерева.
- **Задание** (17 вариант): проверить является ли дерево самоподнобным.
- **Оборудование** (*лабораторное*):
  ЭВМ 1 , процессор Intel Celeron i686 , имя узла сети client 1 с ОП 1000 МБ НМД 70 ГБ. Терминал lxterminal адрес: 192.168.2.37 . Принтер

Другие устройства

Оборудование ПЭВМ студента, если использовалось:
Процессор Intel core i7-7700 , ОП 16384 МБ, НМД 1024 ГБ. Монитор BENQ GW2470
Другие устройства

- **Программное обеспечение** (*лабораторное*):
  Операционная система семейства UNIX, наименование Ubuntu версия 16.04
  Интерпретатор команд bash версия
  Система программирования Си
  версия
  Редактор текстов emacs
  версия
  Утилиты операционной системы cmp ,comm, wc, dd, diff, grep, join, sort ,tail, tee, tr, uniq, od, sum

  Прикладные системы и программы gnuplot, bc

  Местонахождения и имена файлов программ и данных /std/188237

  Программное обеспечение ЭВМ студента, если использовалось:
  Операционная система семейства Windows , наименование Windows 10 версия 10.0.17763.316
  Интерпретатор команд cmd версия
  Система программирования Си версия
  Редактор текстов Sublime text 3 версия 3.1.1
  Утилиты операционной системы проводник
  Прикладные системы и программы Yandex Browser, notepad++
  Местонахождения и имена файлов программ и данных C:\Kirill
- **Идея, метод, алгоритм**

Прежде всего, объявляем структуру, которая будет описывать узел дерева (ключ, левое поддерево, правое поддерево).

Для программы понадобятся следующие функции:

1) **create_tree**(дерево, ключ для корня) - создание дерева (в ней мы выделим память для корня, присвоим ему значение и обнулим правое и левое поддерево)

2) **add_graph**(дерево, ключ узла) - добавление узла к дереву (выделяем память для узла и путем несложных сравнений (if/else) находим место для нового узла в дереве)

3)**tree_print**(дерево, вспомогательная переменная равная 0) - распечатка дерева(рекурсивная функция, которая печатает узлы в зависимости от их расположения в дереве(размер отступа зависит от глубины рекурсии))

4)**inorder_count**(дерево, переменная для записи результата) - симметричный обход дерева, который считает все правые и левые поддеревья у узлов.

5)**inorder_array**(дерево, массив для записи результата, вспомогательная переменная равная 0) - симметричный обход дерева, который проверяет наличие у всех узлов правых и левых поддеревьев(если есть то в массив записывается 1, если нет, то 0).

6)**delete**(дерево, ключ узла) - удаление узла(смотрит на наличие или отсутсвие одного или двух поддеревьев у узла и используя циклы и конструкции ветвления изменяет структуру дерева)

7)menu()- сама программа, оформленная как меню.

Меню реализуем с помощью switch/case.

Проверка на самоподобность

Используя функции **inorder_count** и **inorder_array** создаем массив, хранящий в себе информацию о наличии или отсутвии у всех узлов правых и левых поддеревьев. Проверяем этот массив на палиндром (если массив вида 101101, то дерево самоподобное).

- **Сценарий выполнения работы**

Создать структуру, которая описывает узел дерева.
Создать функции для работы с деревом (добавление узла, удаление узла, печать дерева).
Создать симметричные обходы дерева.
Создать меню.

Допущен к выполнению работы. Подпись преподавателя

- **Распечатка протокола**

```
kircatle@DESKTOP-70J5NO3:/mnt/c/Kirill/Dev/LabsMAI/SecondSem/lab23$
gcc lab23.c
kircatle@DESKTOP-70J5NO3:/mnt/c/Kirill/Dev/LabsMAI/SecondSem/lab23$ cat
lab23.c
#include <stdio.h>
#include <stdlib.h>
typedef struct tree
{
   int key;
   struct tree* left;
   struct tree* right;
}node;
void create_tree(node **tree, int key)
{
   node *tmp = malloc(sizeof(node));
   tmp -> key = key;
   tmp -> left = NULL;
   tmp -> right = NULL;
   *tree = tmp;
}
void add_graph(node **tree, int key)
{
   node *tree2 = *tree;
   node *tree3 = NULL;
   node *tmp = malloc(sizeof(node));
   while (tree2 != NULL)
   {
      tree3 = tree2;
      if (key < tree2 -> key)
         tree2 = tree2 -> left;
      else
         tree2 = tree2 -> right;
   }
   tmp -> key = key;
   tmp -> left = NULL;
   tmp -> right = NULL;
   if (key < tree3 -> key)
```

```
        tree3 -> left = tmp;
    else
        tree3 -> right = tmp;
}
void tree_print(node *tree, int gap)
{
    if (tree == NULL)
        return;
    tree_print(tree -> right, gap+1);
    for (int i = 0; i < gap; ++i)
        printf(" ");
    printf("%d\n",tree -> key );
    tree_print(tree -> left, gap+1);
}
void inorder_count(node* root, int* count)
{
if (root)
{
    inorder_count(root->left, &*count);
    if (root->left)
    {
        (*count)++;
    }
    else
    {
        (*count)++;

    }
    if (root->right)
    {
        (*count)++;

    }
    else
    {
        (*count)++;

    }
    inorder_count(root->right, &*count);
```

```
}
}
void inorder_array(node* root, int* array, int* i)
{
if (root)
{
   inorder_array(root->left, array, &*i);
   if (root->left)
   {
      array[*i]=1;
      (*i)++;
   }
   else
   {
      array[*i]=0;
      (*i)++;
   }
   if (root->right)
   {
      array[*i]=1;
      (*i)++;
   }
   else
   {
      array[*i]=0;
      (*i)++;
   }     inorder_array(root->right, array, &*i);
}
}
void *delete(node **root, int value)
{
   node *l = *root;
   while (l -> key != value)
   {

      if (value < l -> key)
         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 -> key == value)
            break;
      if (root2 -> right != NULL)
         if (root2 -> right -> key == value)
            break;
      if (value < root2 -> key)
         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 -> key == value)
            break;
      if (root2 -> right != NULL)
         if (root2 -> right -> key == value)
            break;
      if (value < root2 -> key)
         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 -> key == value)
                break;
        if (root2 -> right != NULL)
            if (root2 -> right -> key == value)
                break;
        if (value < root2 -> key)
            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 -> key == root2 -> key)
            break;
    if (root3 -> right != NULL)
        if (root3 -> right -> key == root2 -> key)
            break;
    if (value < root3 -> key)
        root3 = root3 -> left;
    else root3 = root3 -> right;
    root4 = root3;
}
l -> key = root2 -> key;
if (root4 -> left -> key == root2 -> key)
{
    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 print_menu()
{
    printf("\n1. Добавить узел.\n2. Напечатать дерево.\n3. Удалить узел.\n4.
Проверить на самоподобие.\nВыберите действие: ");
}
void menu()
{
    int i, count, gap=0, act, graph,c,c2 ;
    node* tree;
    int* array=NULL;
    print_menu();
    while(scanf("%d", &act)!=EOF)
    {
        switch(act)
        {
        case 1:
            printf("\n");
            scanf("%d", &graph);
            if (tree==NULL)
                create_tree(&tree, graph);
            else
                add_graph(&tree, graph);
            print_menu();
            break;
        case 2:
            printf("\n");
            tree_print(tree, gap);
            print_menu();
            break;
        case 3:
```

```
            printf("\n");
            scanf("%d", &graph);
            delete(&tree, graph);
            print_menu();
            break;
        case 4:
            count=0; c=0; c2=0, i=0;
            printf("\n");
            inorder_count(tree, &count);
            array=(int*)realloc(array ,count*sizeof(int));
            inorder_array(tree, array, &i);
            c=count-1; c2=count;
            if (!tree->left || !tree->right)
            {
                printf("Дерево не самоподобное\n");
                free(array);
                print_menu();
                break;
            }
            else
            {
                for(int l = 0; l < count; l++)
                {
                    if (array[c]-array[l]==0)
                        c2--;
                    c--;
                }
                if (c2==0)
                    printf("Дерево самоподобное\n");
                else
                    printf("Дерево не самоподобное\n");
                print_menu();
                free(array);
                printf("\n");
            }
            break;
        default:
            printf("\nПожалуйста, выберите один из представленных пунктов
меню!\n");
```

```
            break;
        }
    }
}
int main()
{
  menu();
}
```

<u>ТЕСТ</u>

```
kircatle@DESKTOP-70J5NO3:/mnt/c/Kirill/Dev/LabsMAI/SecondSem/lab23$
./a.out

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие: 1

5

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие: 1

4

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие: 1

2

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
```

4. Проверить на самоподобие.
Выберите действие: 1

3

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие: 2

5
 4
    3
   2

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие: 4

Дерево не самоподобное

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие: 1

8

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие: 1

10

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие: 1

9

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие: 2

```
    10
       9
  8
5
  4
       3
     2
```

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие: 4

Дерево самоподобное

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие:
3

10

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие: 2

```
   9
  8
5
  4
      3
    2
```

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие: 4

Дерево не самоподобное

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие:
3

5

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие: 2

```
  9
8
```

```
  4
    3
   2
```

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие: 3

4

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие: 3

2

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие: 2

```
  9
8
  3
```

1. Добавить узел.
2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие: 4

Дерево самоподобное

1. Добавить узел.

2. Напечатать дерево.
3. Удалить узел.
4. Проверить на самоподобие.
Выберите действие:

- **Дневник отладки**

№.	Лаб. или дом.	Дата	Время	Событие	Действие по исправлению	Примечание
1	дом	1.05.19	19:00	Массив, который должен содержать информацию об узлах, содержал странные значения	Забыл обнулять после каждой итерации значение переменной i	Gdb очень сильно помог

- Замечание автора по существу работы Теперь я знаком с бинарными деревьями
  Выводы Я создал программу на языке си по созданию и обработки двоичных деревьев

Подпись студента