МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Деревья

Студент гр. 9383 _____ Корсунов А.А.

Преподаватель _____ Попова Е.В.

Санкт-Петербург
2020

Цель работы.

Познакомиться и научиться работать с деревьями и написать программу на языке программирования C++.

Основные теоретические положения.

Дерево – конечное множество *T*, состоящее из одного или более узлов, таких, что

а) имеется один специально обозначенный узел, называемый *корнем* данного дерева;

б) остальные узлы (исключая корень) содержатся в $m \geq 0$ попарно не пересекающихся множествах *T*1, *T*2, ..., *Tm*, каждое из которых, в свою очередь, является деревом. Деревья *T*1, *T*2, ..., *Tm* называются *поддеревьями* данного дерева. Наиболее важным типом деревьев являются *бинарные деревья*. Удобно дать следующее формальное определение. *Бинарное дерево* − конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом.

Задание 10 (массив).

Рассматриваются бинарные деревья с элементами типа *Elem* (в качестве *Elem* использовать *char*). Заданы перечисления узлов некоторого дерева *b* в порядке ЛКП и ЛПК. Требуется:

- восстановить дерево *b* и вывести его изображение;
- перечислить узлы дерева *b* в порядке КЛП.

Ход работы.

После анализа поставленного задания была разработана программа с использованием ООП и рекурсии.

Класс, используемый в программе — Tree (файлы Tree.h и Tree.cpp):

Данный класс содержит следующие поля:

1) ptr_lkp[50] — массив для ЛКП обхода, хранящий элементы дерева;
2) ptr_lpk[50] — массив для ЛПК обхода, хранящий элементы дерева;
3) ptr_ready[50] — массив для КЛП обхода, хранящий элементы дерева;
4) ptr_ready_ind[50] – массив для хранения «уровней» для кажого элемента КЛП обхода согласно уровням в самом бинарном дереве;
5) ptr_len – поле, хранящая длину всех массивов (их для одинаковая);
6) place – вспомогательное поле, необходимое для заполнения КЛП массива в рекурсивном методе make tree;

Методы класса:

1) read_ptr(char ptr[50]) – в данном методе происходит посимвольный ввод для соответствующего массива (так же здесь определяется длина массивов);
2) zero_ptr() - заполнения массивов КЛП, чтобы в них не было мусора (заполнение происходит через цикл);
3) make_tree(char root, int degree, int start, int finish, int check) – данный метод является рекурсивной функцией для заполнения КЛП. Данная функция, принимая корень БД, рекурсивно обходит левую и правую части БД.
4) draw() - метод, который рисует дерево на основе КЛП массива;
5) print_klp() - метод, который выводит КЛП массив

Функция main() (файл main.cpp):

В данной функции происходит создание объекта класса Tree и поочередный вывод всех необходимых методов.

Пример работы программы.

№ п/п	Входные данные	Выходные данные	Комментарии
1	rdnbeakcl rndebklca	--a ----b ------d --------r --------n ------e ----c ------k ------l KLP:abdrneckl	
2	dbeack debkca	--a ----b ------d ------e ----c ------k KLP:abdeck	
3	dbeakcl debklca	Enter LKP: dbeakcl Enter LPK: debklca --a ----b ------d ------e ----c ------k ------l KLP:abdeckl	
4	cba cba	Enter LKP: cba Enter LPK: cba --a ----b ------c KLP:abc	
	ba ba	--a ----b KLP:ab	

Иллюстрация работы программы.

*IDE – Code::Blocks 20.03

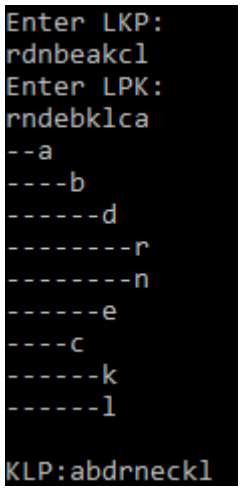

Рисунок 1 - Пример работы программы с входными данными №1

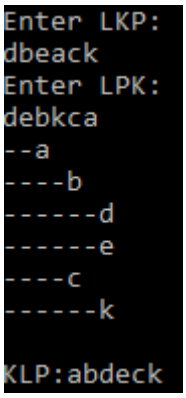

Рисунок 2 - Пример работы программы с входными данными №2

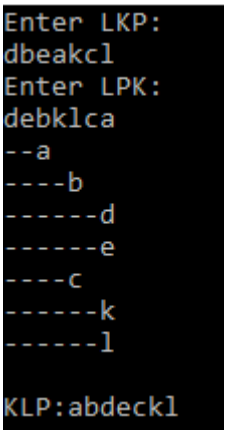

Рисунок 3 - Пример работы программы с входными данными №3

```
Enter LKP:
cba
Enter LPK:
cba
--a
----b
------c

KLP:abc
```

Рисунок 4 — Пример работы программы с входными данными №4

```
Enter LKP:
ba
Enter LPK:
ba
--a
----b

KLP:ab
```

Рисунок 5 — Пример работы программы с входными данными №5

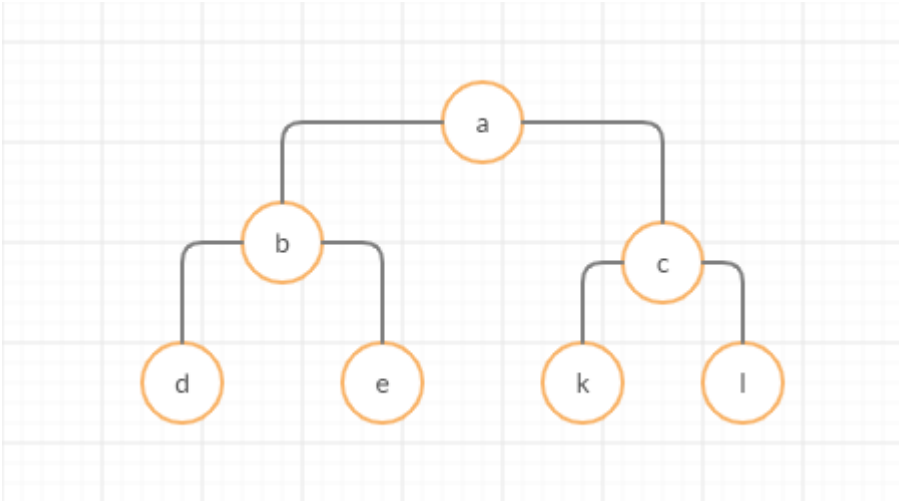

Рисунок 6 — графическое представление бинарного дерева, которое поступает на вход в третьем примере

Выводы.

Произошло ознакомление с деревьями и была написана программа на языке программирования C++, включающее реализацию БД.

ПРИЛОЖЕНИЕ А

Файл main.cpp

```
#include <iostream>
#include "Tree.h"

using namespace std;

int main()
{
   Tree obj;
   cout << "Enter LKP:\n";
   obj.read_ptr(obj.ptr_lkp); //ввод лкп обхода
   cout << "Enter LPK:\n";
   obj.read_ptr(obj.ptr_lpk); //ввод лпк обхода
    obj.zero_ptr(); //установка массивов для клп в нулевые значения, чтобы в них не было мусора
   obj.make_tree(obj.ptr_lpk[obj.ptr_len-1], 1, 0, obj.ptr_len-1, 0); //заполнение одного массива обходом клп, а второго "уровнем" соответственно бинарному дереву под каждое место первого массива
   obj.draw(); //рисовка дерева
   obj.print_klp(); //вывод клп
}
```

Файл Tree.h

```
#pragma once
```

```
class Tree
{
public:
   char ptr_lkp[50]; //массив для лкп
   char ptr_lpk[50]; //массив для лпк
   char ptr_ready[50]; //массив для клп
   int ptr_ready_ind[50]; //вспомогательный массив для клп
   int ptr_len = 0; //длина массивов
   int place = 0; //поле помощи заполнения массива

   void read_ptr(char ptr[50]);
   void draw();
   void zero_ptr();
   void make_tree(char root, int degree, int start, int finish, int check);
   void print_klp();
};
```

Файл Tree.cpp

```
#include "Tree.h"
#include <iostream>

using namespace std;

void Tree::read_ptr(char ptr[50]) // посимвольный ввод обходов в массивы
{
   char c;
   int i = 0;
   c = getchar();
```

```
    while(c != '\n')
    {
        ptr[i] = c;
        i++;
        c = getchar();
    }
    this->ptr_len = i;
    ptr[i] = '\0';
}

void Tree::zero_ptr() //заполнения массив, чтобы в них не было мусоров
{
    for (int i = 0; i < 50; i++)
    {
        this->ptr_ready[i] = ' ';
        this->ptr_ready_ind[i] = 0;
    }
    this->ptr_ready[49] = '\0';
}

void Tree::make_tree(char root, int degree, int start, int finish, int check) // соответсвенно root - текущий корень БД, текущий "уровень" члена БД, текущий старт, текущий финиш, check - переменная, которая отслеживает, когда происходит вызов метода для правой части массива
{
    ptr_ready[place] = root;
    ptr_ready_ind[place] = degree;
    place++;
    int index_root_lkp = -1; //переменная для отслеживания индекса в массиве лкп
    int index_root_lpk = -1; //переменная для отслеживания индекса в массиве лпк
```

```
   for (int i = start; i < finish+1; i++)
   {
      if (ptr_lkp[i] == root)
      {
         index_root_lkp = i;
      }
      if (ptr_lpk[i] == root)
      {
         index_root_lpk = i;
      }
   }
   if (start == finish)
   {
      return;
   }
   if (index_root_lkp-1 >= start && check == 0)
   {
        make_tree(ptr_lpk[index_root_lkp-1], degree+1, start, index_root_lkp-1, 0); //
для левой части, если до этого не было правой части
   }
   if (index_root_lkp-1 >= start && check == 1)
   {
        make_tree(ptr_lkp[index_root_lkp-1], degree+1, start, index_root_lkp-1, 0); //
для левой части, если до этого была правая части
   }
   if (index_root_lkp+1 <= finish)
   {
      check = 1;
           make_tree(ptr_lpk[index_root_lpk-1], degree+1, index_root_lkp+1, finish,
1); //для правой части
```

```
    }
}

void Tree::draw()
{
    for (int i = 0; i < ptr_len; i++)
    {
        for (int j = 0; j < ptr_ready_ind[i]; j++)
        {
            cout << "--";
        }
        cout << ptr_ready[i];
        cout << "\n";
    }
    cout << "\n";
}

void Tree::print_klp()
{
    cout << "KLP:";
    for (int i = 0; i < ptr_len; i++)
    {
        cout << ptr_ready[i];
    }
}
```