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

ОТЧЕТ

по лабораторной работе №4
по дисциплине «АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ»
Тема: Деревья

Студент гр. 8383 _____ Кормщикова А.О.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2019

Цель работы.

Изучить принцип работы таких структура данных, как дерево. Научиться реализовывать деревья на указателях. Научиться использовать для их обработки рекурсивные функции и решить с их помощью практическую задачу.

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, ..., T_m, каждое из которых, в свою очередь, является деревом. Деревья T_1, T_2, ..., T_m называются *поддеревьями* данного дерева.

При программировании и разработке вычислительных алгоритмов удобно использовать именно такое *рекурсивное* определение, поскольку рекурсивность является естественной характеристикой этой структуры данных.

Каждый узел дерева является корнем некоторого поддерева. В том случае, когда множество поддеревьев такого корня пусто, этот узел называется *концевым узлом*, или *листом*. *Уровень* узла определяется рекурсивно следующим образом: 1) корень имеет уровень 1; 2) другие узлы имеют уровень, на единицу больший их уровня в содержащем их поддереве этого корня. Используя для уровня узла a дерева T обозначение *уровень* (a, T), можно записать это определение в виде

$$\text{уровень}(a, T) = \begin{cases} 1, & \text{если } a - \text{ корень дерева } T \\ \text{уровень}(a, T_i) + 1, & \text{если } a - \text{ не корень дерева } T \end{cases}$$

где T_i – поддерево корня дерева T, такое, что $a \in T_i$.

Говорят, что каждый корень является *отцом* корней своих поддеревьев и что последние являются *сыновьями* своего отца и *братьями* между собой.

Говорят также, что узел *n* – *предок* узла *m* (а узел *m* – *потомок* узла *n*), если *n* – либо отец *m*, либо отец некоторого предка *m*.

Если в определении дерева существен порядок перечисления поддеревьев T_1, T_2, …, T_m, то дерево называют *упорядоченным* и говорят о «первом» (T_1), «втором» (T_2) и т. д. поддеревьях данного корня. Далее будем считать, что все рассматриваемые деревья являются упорядоченными, если явно не оговорено противное. Отметим также, что в терминологии теории графов определенное ранее упорядоченное дерево более полно называлось бы «конечным ориентированным (корневым) упорядоченным деревом».

Лес – это множество (обычно упорядоченное), состоящее из некоторого (быть может, равного нулю) числа непересекающихся деревьев. Используя понятие леса, пункт *б* в определении дерева можно было бы сформулировать так: *узлы дерева, за исключением корня, образуют лес*.

Рассмотрим функциональную спецификацию структуры данных дерева с узлами типа α: *Tree of* α = *Tree* (α). При этом лес деревьев *Forest* (α) определим как *L_list* (*Tree* (α)) через уже известную структуру линейного списка *L_list* с базовыми функциями *Cons*, *Head*, *Tail*, *Null* (см. 1.6). Базовые операции с деревом задаются набором функций:

1) *Root*: *Tree* → α;

2) *Listing*: *Tree* → *Forest*;

3) *ConsTree*: α ⊗ *Forest* → *Tree*

и аксиомами (∀*u:* α; ∀*f: Forest* (α); ∀*t*: *Tree* (α)):

A1) *Root* (*ConsTree* (*u*, *f*)) = *u*;

A2) *Listing* (*ConsTree* (*u*, *f*)) = *f*;

A3) *ConsTree* (*Root* (*t*), *Listing* (*t*)) = *t*.

Здесь функции *Root* и *Listing* – селекторы: *Root* выделяет корень дерева, а *Listing* выделяет лес поддеревьев корня данного дерева. Конструктор *ConsTree* порождает дерево из заданных узла и леса деревьев.

Рассмотрим функциональную спецификацию структуры данных бинарного дерева с узлами типа α: *BinaryTree* (α) ≡ *BT* (α). Здесь важно

различать ситуации обработки пустого и непустого бинарных деревьев, поскольку некоторые операции определяются только на непустых бинарных деревьях. Далее считаем, что значение типа *BT* есть либо Λ (пустое бинарное дерево), либо значение типа *NonNullBT*. Тогда базовые операции типа *BT* (α) задаются набором функций:

1) *Root*: *NonNullBT* → α;

2) *Left*: *NonNullBT* → *BT*;

3) *Right*: *NonNullBT* → *BT*;

4) *ConsBT*: α ⊗ *BT* ⊗ *BT* → *NonNullBT*;

5) *Null*: *BT* → *Boolean*;

6) Λ: → *BT*

и набором аксиом (∀*u*: α, ∀*b*: *NonNullBT* (α), ∀*b*1, *b*2: *BT* (α)):

A1) *Null* (Λ) = *true*;

A1') *Null* (*b*) = *false*;

A2) *Null* (*ConsBT* (*u*, *b*1, *b*2)) = *false*;

A3) *Root* (*ConsBT* (*u*, *b*1, *b*2)) = *u*;

A4) *Left* (*ConsBT* (*u*, *b*1, *b*2)) = *b*1;

A5) *Right* (*ConsBT* (*u*, *b*1, *b*2)) = *b*2;

A6) *ConsBT* (*Root* (*b*), *Left* (*b*), *Right* (*b*)) = *b*.

Здесь функции *Root*, *Left* и *Right* – селекторы: *Root* выделяет корень бинарного дерева, а *Left* и *Right* – его левое и правое поддеревья соответственно. Конструктор *ConsBT* порождает бинарное дерево из заданных узла и двух бинарных деревьев. Предикат *Null* – индикатор, различающий пустое и непустое бинарные деревья.

Бинарные деревья особенно полезны, в том числе потому, что существует естественное взаимно-однозначное соответствие между лесами и бинарными деревьями, и многие операции над лесом (деревом) могут быть реализованы как соответствующие операции над бинарным деревом, представляющим этот лес (дерево).

Задание

Вариант 7-д

7. (Обратная задача.) Для заданного бинарного дерева с произвольным типом элементов:

- получить лес, естественно представленный этим бинарным деревом;
- вывести изображение бинарного дерева и леса;
- перечислить элементы леса в горизонтальном порядке (в ширину).

Реализация

Для программы был реализован template класс дерева — BinaryTree.

```
template <class Elem>
class BinaryTree
```

Данных класс в качестве приватного поля содержит указатель на структуру:

```
struct binTree
    {
        binTree* left;
        binTree* right;
        Elem data;
    };
```

Для работы с деревом предусмотрены такие функции, как:

`Elem RootBT()` - Возвращает значения в корне дерева

`BinaryTree Left()` - Возвращает левую ветвь дерева

`BinaryTree Right()` - Возвращает правую ветвь дерева

`bool isNull()` - Проверяет существует ли дерево

`void destroy()` - Удаляет дерево

Для программы был реализован template класс дерева — FTree

```
struct treeNode    {
        vector<treeNode*> links;
        Elem data;
    };
```

Для работы с деревом предусмотрены такие функции, как:

`Elem RootBT()` - Возвращает значения в корне дерева

`bool isNull()` - Проверяет существует ли дерево

`void destroy()` - Удаляет дерево

`void addLink(FTree &tree)` - добавляет поддерево

`vector<FTree<Elem> > getSubtreeVec()` - возвращает вектор поддеревьев

В программе для выполнения задачи реализованы следующие функции:

`int menu(string& iFileName)` - Функция пользовательского интерфейса

`int _enterBT(BinaryTree<char> &b)` - Ввод

`void _outBT(BinaryTree<char> b)` - Вывод форматированного дерева строкой

`void displayBT(BinaryTree<char> b, int n)` -Вывод дарева в наглядном виде

`void displayFF(vector<FTree<char> > forest)` - Функция вывода леса

`void displayFT(FTree<char> b, int n=1)` - Функция вывода дерева

`void BFS(vector<FTree<char> > forest)` - Вывод элементов леса в горизонтальном порядке

`int toForest(BinaryTree<char> b, vector<FTree<char>> &forest)`- функция построение леса из бинарного дерева

`int toTreeNode(FTree<char> &root, BinaryTree<char> subtree)` - функция построения отдельного дерева из поддерева и корня.

В основной функции main последовательно происходит:

Вызов функций `menu(iFileName)`

Затем, если требуется, открытие файлов на вход;

Вызов функции `_enterBT` для считывания данных из потока ввода и формирование объекта класса `BinaryTree`.

Введенное дерево выводится в строковом стандартизированном виде и в наглядном перевернутом виде в поток вывода

Далее создается вектор леса. Вызывается функция для создания леса.

`int toForest(BinaryTree<char> b, vector<FTree<char>> &forest)`

Работа функции создания леса:

На вход функция получает бинарное дерево и адрес вектора леса. Если бинарное дерево пустое, функция заканчивает свою работу. Создается дерево из корня данного бинарного дерева. Оно кладется в вектор деревьев леса. Затем проверяется, есть ли у бинарного дерева правое поддерево. Данное поддерево, при его наличии, является корнем очередного дерева леса. Функция `toForest` вызывается рекурсивно, которой посылается тот же вектор и в качестве бинарного дерева правое поддерево. Затем проверяется, есть ли у бинарного дерева левое поддерево. Данное поддерево, при наличии, является поддеревом созданного дерева. Вызывается функция `toTreeNode`. в качестве корня подается созданное дерево, в качестве поддерева - левое поддерево бинарного дерева.

Работа функции создания дерева

`int toTreeNode(FTree<char> &root, BinaryTree<char> subtree)` - функция принимает на вход корень дерева и его бинарное дерево, корень которого будет поддеревом. Создается поддерево из корня бинарного дерева, добавляется в вектор поддеревьев корня. Затем проверяется наличие левого поддерева бинарного дерева. Если таковое существует - его корень является потомком поддерева. Функция вызывается рекурсивно, посылается поддерево в качестве корня, и левое поддерево. Затем проверяется наличие правого поддерева бинарного дерева. Если таковое существует - его корень является очередным поддеревом данного корня. Функция вызывается рекурсивно, посылается тот же самый корень и правое поддерево.

Продолжение работы main

Вызывается функция печати леса.

Затем вызывается функция вывода элементов леса в горизонтальном порядке. Работа этой функции основана на принципе очереди. В очередь поочередно записывается каждый корень дерева, затем последовательно убирается, при этом он сам печатается на экран, а его поддеревья записываются в конец очереди, итерационный процесс идет до того момента, пока очередь не опустеет.

Тесты.

Input	Output
(a(bc(dfr)) (tyu))	Input: (a (b (c / /) (d (f / /) (r / /))) (t (y / /) (u / /))) Flipped bin tree : a t u y b d r f c Flipped forest : u t y a r d f b c Forest elements in horizontal order: a t u b d r y c f
test.txt: (a (s f g) (d(e/f)l))	Enter the name of file /Users/pcho/c/tree/test.txt Input: (a (s (f / /) (g / /)) (d (e / (f / /)) (l / /))) Flipped bin tree : a d l e f s g f Flipped forest : l d f e a g s f Forest elements in horizontal order: a d l s g e f f

as((*3242))	Wrong input data!
(a//)	Input: (a / /) Flipped bin tree : a Flipped forest : a Forest elements in horizontal order: a
(a)	Input: (a / /) Flipped bin tree : a Flipped forest : a Forest elements in horizontal order: a
(a (j (g / (l)))(h(d/(r(ty(zi)))))	Input: (a (j (g / (l / /)) /) (h (d / (r (t (y / /) (z (i / /) /)) /)) (m / /))) Flipped bin tree : a h m d r t z i y j g l Flipped forest : m h r z i t y d a j l g Forest elements in horizontal order: a h m j d r g l t z y i

Выводы.

В результате работы была написана полностью рабочая программа решающая поставленную задачу при использовании изученных теоретических материалов. Программа было протестирована, результаты тестов удовлетворительны. Неверный ввод приводит к стабильному завершению программы

ПРИЛОЖЕНИЕ(ЛИСТИНГ ПРОГРАММЫ)

ДИНАМИЧЕСКАЯ РЕАЛИЗАЦИЯ БИНАРНОГО ДЕРЕВА

BTREE_.H

```
#ifndef Btree__h
#define Btree__h

#include <iostream>
#include <cstdlib>

using namespace std;

template <class Elem>////Template class
class BinaryTree
{
private:
    struct binTree ///Core structure
    {
        binTree* left;
        binTree* right;
        Elem data;
    };
    binTree* root;
    ///constructor
public:
    BinaryTree()///Constructor
    {
        root =NULL;
    }

    BinaryTree(binTree* _root)///Constructor from ref
    {
        root =_root;
    }

       BinaryTree(Elem _el,BinaryTree l,BinaryTree r)///Constructor from
values
    {
        root = new binTree;
        root->data=_el;
        root->left=l.root;
        root->right=r.root;///new binTree(_el,l.root,r.root);
    }
    BinaryTree(Elem _el){///Constructor from root only
        root = new binTree;
        root->data=_el;
        root->left=NULL;
        root->right=NULL;
    }
```

```
        ///prototypes
        bool isNull() const {
            return root == NULL;
        }///Empty tree
        Elem RootBT(){///Root value
            if(this->isNull()){ cerr << "Error: RootBT(null) \n"; exit(1); }
            else return root->data;
        }
        BinaryTree Left(){///Return left branch
            if(this->isNull()){ cerr << "Error: RootBT(null) \n"; exit(1); }
                 else return BinaryTree(root->left);///*(new BinaryTree(root-
>left));
            }
        BinaryTree Right(){///Return right branch
            if(this->isNull()){ cerr << "Error: RootBT(null) \n"; exit(1); }
                else return BinaryTree(root->right);///*(new BinaryTree(root-
>right));
        }

        void destroy(){///Destroy this tree
            if(!isNull()){
                Left().destroy();
                Right().destroy();
                delete root;
            }
        }
    };

    #endif /* Btree__h */H
```

ДИНАМИЧЕСКАЯ РЕАЛИЗАЦИЯ ДЕРЕВА

FTREE_.H

```
    #ifndef Ftree_h
    #define Ftree_h

    #include <iostream>
    #include <cstdlib>
    #include <vector>
    using namespace std;

    template <class Elem>////Template class
    class FTree{
    private:
        typedef struct treeNode ///Core structure
        {
            vector<treeNode*> links;//vector of subtrees
```

```
        Elem data;//root
    }treeNode;
    treeNode* root;
    ///constructor
public:
    FTree()///Constructor
    {
        root = NULL;
    }

    FTree(treeNode* _root)///Constructor from ref
    {
        root =_root;
    }

    FTree(Elem _el){///Constructor from root only
        root = new treeNode;

        root->data=_el;

    }

    ///prototypes
    bool isNull() const {
        return root == NULL;

    }///Empty tree
    Elem RootFT(){///Root value
        if(this->isNull()){
            cerr << "Error: RootBT(null) \n";
            exit(1);

        }
        else
            return root->data;
    }

    void destroy(){///Destroy this tree
        if(!isNull()){
            (root->links).clear();
            delete root;
        }
    }
    void addLink(FTree &tree){//add subtree
        (root->links).push_back(tree.root);
    }

    vector<FTree<Elem> > getSubtreeVec(){///getting subtree vector
        if(!isNull()){
            auto _links = this->root->links;
```

```
                vector<FTree<Elem> > vec;
                for(auto it = _links.begin();it!=_links.end();it++){
                    vec.push_back(FTree(*it));
                }
                return vec;
            }else{
                cerr << "Error: getSubtreeVec(null) \n";
                exit(1);
            }
        }
    };

    #endif /* Ftree_h */
```

ОСНОВНОЙ КОД

WORK_BT.CPP

```
    #include <iostream>
    #include <fstream>
    #include <fstream>
    #include <queue>
    #include <vector>
    #include <cstdlib>
    #include "Btree_.h"
    #include "Ftree.h"
    #define INS ( readFromFile?(*inFile):cin )

    using namespace std;


    istream *inFile = NULL; //for name input file
    bool readFromFile = false;

    void setIFile(istream* istr){
        inFile=istr;
    }


    BinaryTree<char> ConsBT(const char &x, BinaryTree<char> lst,
BinaryTree<char> rst) {
            return BinaryTree<char>(x, lst, rst);//*(new
BinaryTree<char>(x,lst,rst));
    }

    BinaryTree<char> ConsBT(const char &x) {
        return BinaryTree<char>(x);//*(new BinaryTree<char>(x));
    }
```

```
      int _enterBT(BinaryTree<char> &b);//Input

      void _outBT(BinaryTree<char> b);//Formatter str tree output

      void displayBT(BinaryTree<char> b, int n);//Fancy tree graph
      void displayFF(vector<FTree<char> > forest);
      void displayFT(FTree<char> b, int n=1);
      void BFS(vector<FTree<char> > forest);

      ///Utility stuff
      void printKLP(BinaryTree<char> b);//Print tree Rlr
      void printLKP(BinaryTree<char> b);//Print tree lRr
      void printLPK(BinaryTree<char> b);//Print tree lrR


      //------------------------
      int toTreeNode(FTree<char> &root, BinaryTree<char> subtree){//non-binary
tree creation function
          FTree<char> tree = FTree<char>(subtree.RootBT());//creating a subtree
           root.addLink(tree);                                  //adding it to the
list of subtrees of a given root
           if(!subtree.Left().isNull()){//the left subtree of the subtree is the
son of this subtree
              toTreeNode(tree, subtree.Left());//subrtee.Left - another subtree
          }
            if(!subtree.Right().isNull()){//the right subtree of the subtree is
the brother of this subtree
              toTreeNode(root, subtree.Right());
          }
          return 0;
      }
      //------------------------
      int toForest(BinaryTree<char> b, vector<FTree<char>> &forest){//forest
creation function
          if(b.isNull()){
              return 1;
          }
           FTree<char> tree = FTree<char>(b.RootBT());//creating a tree from a
sent root
          forest.push_back(tree);//adding tree to forest trees vector
           if(!b.Left().isNull()){//left subtree of a binary tree - the son of
tree
              toTreeNode((forest[forest.size()-1]), b.Left());
             // toTreeNode(tree, b.Left());//&
          }
           if(!b.Right().isNull()){//right binary tree subtree - another forest
tree
              toForest(b.Right(), forest);
          }
```

```
        return 0;
    }

    //----------------
    int menu(string& iFileName){//user interface
        int inputN;//for command
        while(true){
             cout<<"Conversion of a binary tree into a forest\n Select input.
\n 1 - input from the console \n 2 - input from a file"<<endl;
            cin>>inputN;
            cin.clear();
            cin.ignore(10000,'\n');
            if(cin.fail()||((inputN != 1)&&(inputN !=2))){
                cout<<"\nInvalid comand, try again.\n";
                continue;
            }
            if(inputN == 2){
                cout<<"Enter the name of file\n";
                cin>>iFileName;
                readFromFile = true;
                return 1;
                }
            if(inputN == 1){
                readFromFile=false;
                return 0;
            }
        }
    }

    int main() {
        //Variables
        string iFileName;//for file name
        BinaryTree<char> b;// for input binary tree

        ifstream iFile;

        if(menu(iFileName)){// call menu 1 - input from file
            iFile.open(iFileName);
            if (!iFile){
                cout<<"Cannot open file: "<< iFileName <<"\n";
                exit(1);
            }
            setIFile(&iFile);
        }
        else{//  0 - input from console
            cout<<"Enter tree sequense:\n";
        }
        //Main inpit-process-output-out cycle

        //Input
```

```
        if(_enterBT(b)){//call _enterBT. 1 - errors
            if(INS.eof()){
                cout<<"Input is over!\n";
                return 0;

            }
            if(INS.fail()){
                cout<<"Input stuck!\n";
                return 0;
            }
            cout<<("Wrong input data!\n");
            return 0;
        }
        cout<<"Input: ";
        _outBT(b);//Prints formatted string tree
        cout<<"\n";

        ///Representation
        cout<<"Flipped bin tree : \n";
        displayBT(b, 1);//displaying the tree in a visual form of the list

        ///Real processinn

        vector<FTree<char>> forest;
        toForest(b, forest);//creating a forest from binary tree
        cout<<"\n";
          displayFF(forest);//displaying the forest in a visual form of the
list
        BFS(forest);//output of forest elements in horizontal order
        b.destroy();
        return 0;
    }

    //--------------------------------------
    ///Read tree expression. Called from _enterBT
    int _readExpr(BinaryTree<char> &b) {
        // BinaryTree<char> b;
        char ch;
        char root;
        BinaryTree<char> l;
        BinaryTree<char> r;

        ///Extra spaces read
        INS >> ch;
        while (ch == ' ')
            INS >> ch;
        if (ch == ')') {
            b = BinaryTree<char>();
            return 0;
        }
```

```
///Read root

root = ch;

INS >> ch;

while (ch == ' ')
    INS >> ch;
if (ch == ')') {
    b = BinaryTree<char>(root);
    return 0;
}
///Read left
if (ch == '(') {
    if (_readExpr(l))
        return 1;
}
else if (ch == '/')
    l = BinaryTree<char>();
else
    l = ConsBT(ch);

INS >> ch;

while (ch == ' ')
    INS >> ch;
if (ch == ')') {
    b = BinaryTree<char>(root, l, BinaryTree<char>());
    return 0;
}
///Read right
if (ch == '(') {
    if (_readExpr(r))
        return 1;

}
else
    if (ch == '/')
        r = BinaryTree<char>();
else
    r = ConsBT(ch);
///Read to the end
INS >> ch;
while (ch == ' ')INS >> ch;
if (ch == ')') {
    b = BinaryTree<char>(root, l, r);
    return 0;
} else {
    ///WROOOOOOOONG
    return 1;
```

```
        }

    }

    ///User function for tree reading
    int _enterBT(BinaryTree<char> &b) {//Input

        char ch;
        INS >> ch;
        while (ch == ' ')
            INS >> ch;
        if (ch == '(') {
            return _readExpr(b);
        }
        else {
            //Everything is bad...
            b = BinaryTree<char>();
            return 1;
        }
    }
    //---------------------------------------

    void _outBT(BinaryTree<char> b) {///Prints formatted string tree
        if (!b.isNull()) {
             //if(b.Left().isNull() && b.Right().isNull()){cout<<b.RootBT()<<"
";return;}
            cout<<"( ";
            cout<<b.RootBT();
            cout<<" ";//b.RootBT()
            _outBT(b.Left());
            _outBT(b.Right());
            cout<<") ";
        }
        else
            cout<<"/ ";
    }


    //---------------------------------------
    void displayBT(BinaryTree<char> b, int n) {///Prints graphic bin tree
        if (!b.isNull()) {
            cout<< " ";
            cout<<b.RootBT();
            if (!b.Right().isNull()) {
                displayBT(b.Right(), n + 1);

            }
            else cout<<"\n";
            if (!b.Left().isNull()) {
                for (int i = 1; i <= n; i++)
```

```
                    cout<<"  ";
                displayBT(b.Left(), n + 1);
            }
        } ;
    }

    //--------------------------------------
    ///Main task stuff
    void BFS(vector<FTree<char> > forest){//output of forest elements in
horizontal order
        cout<<"Forest elements in horizontal order: \n";
        queue<FTree<char>> q;//create a queue
        int k = forest.size();
        for(int i = 0; i < k; i++){
            q.push(forest[i]);//push forest tree roots
        }
        while(!q.empty()){
            FTree<char> c = q.front();
            q.pop();                  //the root of the tree is pop at the top
of the queue
            cout<<c.RootFT()<<" ";  //prints this root
            vector<FTree<char>> subtree = c.getSubtreeVec();//get subtrees
            k = subtree.size();
            for(int i = 0; i<k; i++){
                q.push(subtree[i]);//pushing subtrees
            }
        }
        cout<<endl;
    }

    void displayFF(vector<FTree<char> > forest){
        cout<<"Flipped forest : \n";
        for(int i=forest.size()-1;i>=0;i--){//prints all trees
            displayFT(forest[i]);
        }
    }

    void displayFT(FTree<char> b, int n) {///Prints graphic tree
        if (!b.isNull()) {
            cout<< " ";
            cout<<b.RootFT();
            auto vec = b.getSubtreeVec();//getting all subtrees

            if(vec.size()!=0){
                auto rit=vec.rbegin();
                displayFT(*rit, n+1);
            }else{
                cout<<endl;
            }
            for(int j = vec.size()-2; j>=0 ; j--){
```

```
            for (int i = 1; i <= n; i++)
                cout<<"  ";//indent printing
            displayFT(vec[j], n+1);
        }

    }
}

//---------------------------------------




///Utility stuff

//---------------------------------------
void printKLP(BinaryTree<char> b) {//Print tree Rlr
    if (!b.isNull()) {
        cout<<b.RootBT();
        printKLP(b.Left());
        printKLP(b.Right());
    }
}

//---------------------------------------
void printLKP(BinaryTree<char> b) {//Print tree lRr
    if (!b.isNull()) {
        printLKP(b.Left());
        cout<<b.RootBT();
        printLKP(b.Right());
    }
}

//---------------------------------------
void printLPK(BinaryTree<char> b) {//Print tree lrR
    if (!b.isNull()) {
        printLPK(b.Left());
        printLPK(b.Right());
        cout<<b.RootBT();
    }
}
//---------------------------------------
```