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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных »

Тема: «Деревья»

Студент гр. 9384		Давыдов Д.С.
Преподаватель		Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Разобраться в представлении и реализации такой структуры, как бинарное дерево, способах её организации и обработки.

Задание.

Вариант 4.

Для заданного бинарного дерева b типа BT с произвольным типом элементов определить, есть ли в дереве b хотя бы два одинаковых элемента.

Анализ задачи.

В данной лабораторной работе дерево реализовано на основе списка.

Сначала считывается обычный иерархический список `Hlist list` (F), который представляет из себя лес, а потом с помощью функции `TreeFromForest` из его элементов создается бинарное дерево `BinaryTree b` (B).

Алгоритм по преобразованию можно представить следующим образом:

```
B (F) = if Null(F) then ^ (nullptr)
        else ConsBT(Root(Head(F)),B(Listing(Head(F))),B(Tail(F))).
```

В самой же программной функции он немного изменен.

Сначала проверяем указывает ли голова на атом, если да — то возвращаем конструктор бинарного дерева с корнем равным атому, левым поддеревом равным рекурсии функции от функции `isNextAtom` (принимает на вход хвост текущего элемента, если следующий элемент является атомом, то возвращает nullptr, то есть у текущего элемента нет сыновей, а если следующий элемент не атом, то получается что сын, возвращает хвост обратно) и правым поддеревом равным рекурсии самой функции от ближайшего нового дерева в лесу (которое находится с помощью функции `nextTreeMain`). Если голова не указывает на атом, но указывает на `nullptr` – вызываем ошибку и прерываем работу программы, тк нельзя создать дерево из пустого списка. Если же голова

не указывает на `nullptr`, то возвращаем конструктор с корнем равным атому головы элемента, на который указывает голова текущего элемента (указываем на сына), левым поддеревом равным рекурсии функции от хвоста элемента, на который указывает голова текущего элемента (указываем на возможного сына сына), правым поддеревом равным рекурсии функции от следующего сына текущего родителя (находится с помощью функции `nextTreeSub`).

Таким образом в конце выполнения функции у нас получится бинарное дерево.

Подсчет корней, скобочный вывод и запись всех корней в некоторый массив реализованы с помощью прямого обхода.

Выполнение работы.

Был разработан примитивный API, благодаря которому пользователь может выбрать, как ввести нелинейный список — из файла или из консоли.

Ошибки обрабатываются отдельной функцией error.

Функция, которая находит хотя бы два одинаковых элемента работает по следующему принципу: считается количество корней в бинарном дереве, создается массив этого размера из элементов этого дерева. В дереве в двух циклах сравниваются элементы. Как только находяться одинаковые элементы, их значение выводится на экран, цикл по перебору прерывается.

Функцией `displayBinaryTree` в терминале выводится бинарное дерево, где связь родитель→сыновья идет слева направо. Корректное отображение осуществляется параметром depth (увеличивающимся с каждым рекурсивным вызовом на +1), который отвечает за нужное количество табуляций.

Разработанный код смотреть в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 - результаты тестирования.

№ п/п	Входные данные	Выходные данные	Комментарии

1	//Введены из файла text1; (y(i) l j(o)(b)(e)(c(d(n))(e)(n)))	Binary tree in direct order: (y (i ^^)(l ^(j (o ^(b ^(e ^(c (d (n ^^)(e ^(n ^^)))^))))^)))) At least one identical element - e	Программа успешно нашла одинаковые элементы
2	//Введены из консоли)(a(b(c)))	Error: a character ")" was entered first into terminal	Программа выдала ошибку — список не может читаться с «)»
3	//Введены из консоли (n(m())(k))	Error: you can't create new tree from emtpy list	Программа выдала ошибку — нельзя создать дерево из пустого списка «()»
4	//Введено название несуществующего файла not_file.txt	Error: There is no such file	Программа выдала ошибку — не существует файла с данным названием
5	//Введены из консоли (a(b)h(i(o)))	Binary tree in direct order: (a (b ^^)(h (i (o ^^)^)^)) There are no identical elements	Программа не обнаружила одинаковых элементов в бинарном дереве.

Вывод.

В ходе выполнения лабораторной работы была создана рабочая программа с необходимым API интерфейсом, которая может считывать из файла или командой строки необходимые символы, создавая тем самым из них нелинейный список, затем составляя из него бинарное дерево и проверки на наличие в нем хотя бы два одинаковых корня. На практике были закреплены навыки по написанию и использованию рекурсивных функций, представлении и рекурсивной обработки иерархических списков и бинарных деревьев.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММ.

Название файла: main.cpp

```
#include <iostream>
#include "BinaryTree.h"
int index = 0;

int numberOfElements(BinaryTree<char> *root)
{
    int count = 1;
    if (root->leftSubTree != nullptr) {
        count += numberOfElements(root->leftSubTree);
    }
    if (root->rightSubTree != nullptr) {
        count += numberOfElements(root->rightSubTree);
    }
    return count;
}

void makeDirectOrderArray(BinaryTree<char> *tree,char *array){
    if(!tree->isNull(tree)){
        array[index] = tree->rootData;
        index++;
        makeDirectOrderArray(tree->leftSubTree,array);
        makeDirectOrderArray(tree->rightSubTree,array);
    }
}

void check(BinaryTree<char> *tree){
    if(!tree->isNull(tree)){
        cout<<tree->rootData;
        check(tree->leftSubTree);
        check(tree->rightSubTree);
```

```
    }
}

void atLeastTwoEqualElements(BinaryTree<char> *tree){
    int size = numberOfElements(tree);
    char *array = new char[size];
    makeDirectOrderArray(tree,array);

    int i, j;
    char duplicate = '\0';

    for(i = 0; i < size; i++) {
        for (j = i + 1; j < size; j++) {
            if (array[i] == array[j]) {
                duplicate = array[i];
                break;
            }
        }
        if(duplicate!='\0') break;
    }

    if(duplicate=='\0'){
        cout<<"There are no identical elements"<<endl;
    }

    else {
        cout<<"At least one identical element - "<<duplicate<<endl;
    }
}

void displayBinaryTree(BinaryTree<char> *tree, int depth = 0){
    if(!tree->isNull(tree)){
        displayBinaryTree(tree->rightSubTree,depth+1);
        for(unsigned int i = 0; i<depth; i++)
            std::cout << "\t";
        std::cout << tree->rootData << std::endl;
        displayBinaryTree(tree->leftSubTree, depth+1);
    }
}
```

```
int main(){

    BinaryTree<char> p;
    BinaryTree<char> *j;

     cout<<"Choose way to enter forest:\n1 - read from console\n2 - read
from file"<<endl;
    char c;
    scanf("%c",&c);
    getchar();

    switch(c){
        case '1':
             cout << "Enter forest like this (a(e) b c), where a, b and c
- main trees of forest"<<endl;
            p.readHList(p.list);
            break;
        case '2': p.readHListFile(p.list); break;
        default: cout<<"You haven't choose anything"<<endl; return 1;
    }

    j = p.TreeFromForest(p.list);

    cout<<"Binary tree in direct order: ";
    j->printBtDirectOrder(j);
    cout<<'\n';

                cout<<"Binary    tree    representation    in    console:\
n----------------------"<<endl;
    for(int i = 0; i< numberOfElements(j);i++){
        cout<<'\n';
    }
    displayBinaryTree(j);
    for(int i = 0; i< numberOfElements(j);i++){
        cout<<'\n';
    }
    cout<<"----------------------"<<endl;
```

```
    atLeastTwoEqualElements(j);
}
```

Название файла: BinaryTree.h

```
#ifndef TREETEST_BINARYTREE_H
#define TREETEST_BINARYTREE_H
#include <iostream>
#include <fstream>

using namespace std;
template <class T>
class BinaryTree {
public:
    T rootData;
    BinaryTree *leftSubTree;
    BinaryTree *rightSubTree;
    T character = '\0';

    struct S_expr;
    struct  two_ptr
    {
        S_expr *hd;
        S_expr *tl;
    } ;

    struct S_expr {
        bool tag;
        union
        {
            T atom;
            two_ptr pair;
        } node;
    };

    typedef S_expr *Hlist;

    Hlist list;

      BinaryTree(T  rootData  =  NULL,  BinaryTree  *leftSubTree  =  nullptr,
```

```
BinaryTree *rightSubTree = nullptr) {
        this->rootData = rootData;
        this->leftSubTree = leftSubTree;
        this->rightSubTree = rightSubTree;
    }

    //////////////methods binary tree//////////////

    bool isNull(BinaryTree *tree){
        return tree == nullptr;
    }

    Hlist nextTreeMain(Hlist tree){
        if(isNull(tree)) return nullptr;
        if(isAtom(head(tree))) return tree;
        else return nextTreeMain(tail(tree));
    }

    Hlist nextTreeSub(Hlist tree){
        if(isNull(tree)) return nullptr;
        if(isAtom(head(tree))) return nullptr;
        else return tree;
    }

    Hlist isNextAtom(Hlist tree){
        if (!isNull(tree)){
            if (isAtom(head(tree))){
                return nullptr;
            } else return tree;
        } else return nullptr;
    }

    BinaryTree<char>* TreeFromForest(Hlist forest){
        if(isNull(forest)) return nullptr;
        if(isAtom(head(forest))) {
            return new BinaryTree<char>(
                    getAtom(head(forest)),
                    TreeFromForest(isNextAtom(tail(forest))),
                    TreeFromForest(nextTreeMain(tail(forest))));
```

```
    } else {
        if(isNull(head(forest))) {
            error(10);
            exit(0);
        }
        else {
            return new BinaryTree<char>(
                    getAtom(head(head(forest))),
                    TreeFromForest(tail(head(forest))),
                    TreeFromForest(nextTreeSub(tail(forest))));
        }
    }
}

void printBtDirectOrder(BinaryTree *tree){
    if(isNull(tree)){
        cout<<'^';
    }
    else{
        cout << '(';
        cout<<tree->rootData << ' ';
        printBtDirectOrder(tree->leftSubTree);
        printBtDirectOrder(tree->rightSubTree);
        cout << ')';
    }
}

////////////methods hierarchical list///////////
Hlist head(const Hlist s) {
    if (s != nullptr)
        if (!isAtom(s)) return s->node.pair.hd;
        else {
            error(1);
            exit(0);
        }
    else {
        error(2);
        exit(0);
    }
```

```
}

bool isAtom(const Hlist s) {
    if (s == nullptr) return false;
    else return (s->tag);
}

bool isNull(const Hlist s) {
    return s == nullptr;
}

Hlist tail(const Hlist s) {
    if (s != nullptr)
        if (!isAtom(s)) return s->node.pair.tl;
        else {
            error(3);
            exit(0);
        }
    else {
        error(4);
        exit(0);
    }
}

Hlist cons(const Hlist h, const Hlist t) {
    Hlist p;
    if (isAtom(t)) {
        error(5);
        exit(0);
    } else {
            p = new S_expr;
            p->tag = false;
            p->node.pair.hd = h;
            p->node.pair.tl = t;
            return p;
        }
    }
```

```
Hlist makeAtom(const T x) {
    Hlist s;
    s = new S_expr;
    s->tag = true;
    s->node.atom = x;
    return s;
}

void destroy(Hlist s) {
    if (s != nullptr) {
        if (!isAtom(s)) {
            destroy(head(s));
            destroy(tail(s));
        }
        delete s;
    }
}

T getAtom(const Hlist s) {
    if (!isAtom(s)) {
        error(6);
        exit(0);
    } else return (s->node.atom);
}


void readHList (Hlist& y) {
    do character = getchar(); while (character == ' ');
    readS_expr(character,y);
}

void readS_expr(T prev, Hlist &y) {
    if (prev == ')') {
        error(7);
        exit(0);
    } else if (prev != '(') y = makeAtom(prev);
    else {readSeq(y);}
}
```

```
void readSeq(Hlist &y) {
    Hlist p1, p2;
    if (!(character = getchar())) {
        error(8);
        exit(0);
    } else {
        while (character == ' ') character = getchar();
        if (character == ')') y = nullptr;
        else {
            readS_expr(character, p1);
            readSeq(p2);
            y = cons(p1, p2);
        }
    }
}

void readHListFile(Hlist &y) {
    std::string filePathIn;
    std::cout << "Enter input file:\n";
    std::cin >> filePathIn;
    std::ifstream in;
    in.open(filePathIn);
    if (!in.is_open()) {
        error(9);
        exit(0);
    }
    T x;
    do in >> x; while (x == ' ');
    readS_exprFile(x, y, in);
}

void readS_exprFile(T prev, Hlist &y, ifstream &infile) {
    if (prev == ')') {
        error(7);
        exit(0);
    } else if (prev != '(') y = makeAtom(prev);
    else readSeqFile(y, infile);
}
```

```
void readSeqFile(Hlist &y, ifstream &infile) {
    T x;
    Hlist p1, p2;

    if (!(infile >> x)) {
        error(8);
        exit(0);
    } else {
        while (x == ' ') infile >> x;
        if (x == ')') y = nullptr;
        else {
            readS_exprFile(x, p1, infile);
            readSeqFile(p2, infile);
            y = cons(p1, p2);
        }
    }
}

void printHList(const Hlist x) {
    if (isNull(x)) cout << " ()";
    else if (isAtom(x)) cout << ' ' << x->node.atom;
    else {
        cout << " (";
        printSeq(x);
        cout << " )";
    }
}

void printSeq(const Hlist x) {
    if (!isNull(x)) {
        printHList(head(x));
        printSeq(tail(x));
    }
}


////////////// errors ////////////

void error(unsigned int number){
```

```
        switch(number){
            case 1: cout << "Error: Head(atom) \n"<< endl; break;
            case 2: cout << "Error: Head(nil) \n"<< endl; break;
            case 3: cout << "Error: Tail(atom) \n"<< endl; break;
            case 4: cout << "Error: Tail(nil) \n"<< endl; break;
            case 5: cout << "Error: Cons(*, atom)\n"<< endl; break;
            case 6: cout << "Error: getAtom(s) for !isAtom(s) \n"<< endl;
break;
             case 7: cout << "Error: a character \")\" was entered first
into terminal" << endl; break;
              case 8: cout << "Error: a character \"(\" was entered and
nothing after it"<<endl; break;
               case 9: cout << "Error: There is no such file" << endl;
break;
            case 10: cout << "Error: you can't create new tree from emtpy
list"<<endl;break;
            default: exit(0);
        }
    }
};
#endif
```