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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарные деревья и лес

Студентка гр. 7383 _____ Маркова А.В.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

Содержание

Цель работы

Цель работы: изучить структуру бинарного дерева и леса. Научиться выполнять операции с бинарными деревьями и лесом.

Формулировка задачи:

Вариант 6-в:

Для заданного бинарного дерева:

a) получить лес, естественно представленные этим бинарным деревом;
b) вывести изображение бинарного дерева и леса;
c) перечислить элементы леса в горизонтальном порядке (в ширину).

Реализация задачи

```
struct bt_element{
        Type value;
        int left;
        int right;
};
```

`value` – значение хранящееся в листе бинарного дерева;

`left` – индекс левого сына в массиве;

`right` – индекс правого сына в массиве.

```
struct binTree{
        int size;
        bt_element * storage;
        int depth;
};
```

`size` – размер массива для хранения бинарного дерева;

`storage` – указатель на массив элементов бинарного дерева;

`depth` – глубина бинарного дерева.

```
void binTree::print_tree(int index,int depth)
```

index – текущая позиция в бинарном дереве.

depth – глубина дерева, включающая текущий лист и сыновей.

Метод печатает представление бинарного дерева.

```
struct forest_element{
          Type value;
          int * sons;
};
```

value – значение хранящееся в листе леса;

sons – указатель на массив индексов сыновей элемента.

```
struct Forest{
          int size;
          int max_sons;
          forest_element * storage;
          int depth;
};
```

size – размер массива для хранения леса;

max_sons – максимальное число сыновей для конкретного леса;

storage – указатель на массив элемнтов леса;

depth – глубина леса.

```
void Forest::print_forest(int index,int depth)
```

index – текущая позиция в лесе.

depth – глубина леса, включающая текущий лист и сыновей.

Метод печатает представление леса.

```
void print_horizontal(std::string * out,int index, int level)
```

out – указатель на массив строк, в которых содержатся перечисление элементов каждого уровня;

index – текущая позиция в лесе.

`level` – текущий уровень.

```
int fill_from_str(const binTree<char> * bt,const std::string str_bt, int * position, int root);
```

`bt` – указатель на бинарное дерево;

`str_bt` – строка, в которой записано скобочное представление бинарного дерева;

`position` – текущая позиция в строке;

`root` – отец, к которому присоединяем сыновей (если они есть).

```
void convert_bt_to_forest(const binTree<char> * bt, Forest<char> * bt_forest,int father,int num_son,int * posf,int pos);
```

`bt` – указатель на бинарное дерево;

`bt_forest` – указатель на лес;

`father` – отец текущего элемента;

`num_son` – количество братьев элемента и сыновей отца;

`posf` – текущая позиция в лесе;

`pos` – текущая позиция в бинарном дереве.

Программа запрашивает как пользователь хочет ввести денные из файла или из терминала, воспользовавшись оператором `switch`, затем начинается запись элементов бинарного дерева в соответствующую структуру. В ходе заполнения проверятся все ли скобки закрыты, высчитывается максимальная глубина вложенности. Также не допускается введение двух символов подряд.

Происходит вывод графического представления бинарного дерева, затем конвертация его в лес с дальнейшим выводом на экран.

Начинается обход леса по горизонтали с последующим выводом получившейся последовательности.

Программа реализована для всех типов входных данных, используются шаблоны класса.

Тестирование

Программа была собрана в компиляторе g++ в OS Linux Ubuntu 16.04 при помощи g++. В других системах тестирование не проводилось. Результаты тестирования приведены в приложении А.

Выводы

В ходе лабораторной работы были изучены принципы работы с бинарным деревом и лесом, его заполнение, печать элементов, конвертация одного в другое.

ПРИЛОЖЕНИЕ А

ТЕСТОВЫЕ СЛУЧАИ

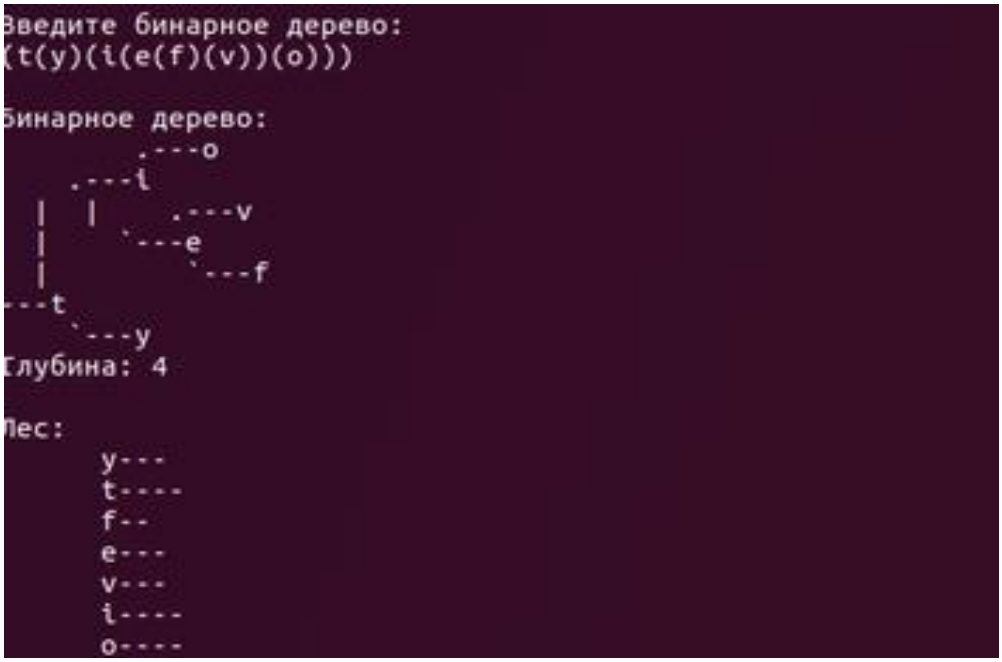

Рисунок 1 – Тестовый случай 1

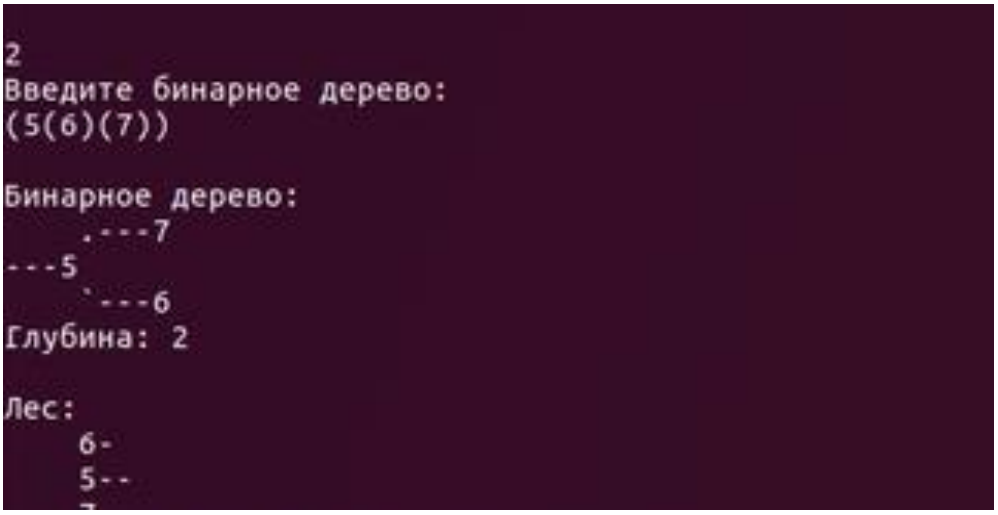

Рисунок 2 – Тестовый случай 2

Рисунок 3 – Тестовый случай 3

ПРИЛОЖЕНИЕ Б

КОД ПРОГРАММЫ

```
#include "fun.h"
#define Type int
#define elMax 50

int main() {
    int check = 1, run = 1;

    std::ifstream file;                         //определяем файловый
поток
    char name[20];                              //название, используемого
файла
    std::string str;                            //строка, вводимая
пользователем
    int i = -1;
    std::cout << "\033[34m\tЗдравствуйте!Выберите, что вы
хотите:\033[0m\n 1) Нажмите 1, чтобы считать с файла.\n 2) Нажмите 2,
чтобы считать с консоли.\n 3) Нажмите 3, чтобы выйти из программы.\n"
<< std::endl;
    while(run) {
        if(!check) {
           binTree<Type> user_bt(elMax);
           char c;
           for(c = getchar(); c == ' ' && c != '\n'; c = getchar());
           while(c != EOF && c != '\n'){
                     if(c != '('){
                           std::cout << "Incorrect input" << std::endl;
                           exit(1);
                     }
                     user_bt.fill_from_input(i);
                     for(c = getchar(); c == ' ' && c != '\n'; c =
getchar());
                }
            std::cout << "\n"<< "Бинарное дерево:" << std::endl;
            user_bt.printTree(0, nullptr, true);
            int bt_depth = user_bt.depth(0);
            std::cout << "Глубина: " <<  bt_depth << std::endl;
            Forest<Type> user_forest(elMax+1,bt_depth,bt_depth);
            int posf=1;
            convert_bt_to_forest(&user_bt,&user_forest,0,0,&posf,0);
            std::cout << "\n"<< "Лес:" << std::endl;
            user_forest.print_forest(0,bt_depth);
         std::string * levels= new std::string[bt_depth];
```

```
            user_forest.print_horizontal(levels,0,0);
            std::cout << "Выберите следующую команду:" << std::endl;

        }
      std::cin >> check;
        std::cin.ignore();
        switch(check) {
        case(1):
            std::cout << "Введите имя нужного файла:" << std::endl;
            std::cin.getline(name,20);
            file.open(name);
            if (!file.is_open())
                std::cout << "\n\033[31mВходной файл не может быть
открыт!\033[0m\nВыберите следующую команду:" << std::endl;
            else {
                std::getline(file,str);
                file.close();
                check=0;
                continue;
            }
            break;
        case(2):
            std::cout << "Введите бинарное дерево:" << std::endl;
            check = 0;
            continue;
        case(3):
            std::cout<<"\033[34mEnd!"<<std::endl;
            run = 0;
            break;
        default:
            std::cout<<"\033[31mIncorrect pick!\033[0m"<<std::endl;
            run = 0;
            break;
        }
    }
}

#include <iostream>
#include <fstream>
#include <string>
#include <cmath>

#define V_MAX 50
```

```
struct Trunk {
    Trunk *prev;
    std::string str;
    Trunk(Trunk *prev, std::string str) {
        this->prev = prev;
        this->str = str;
    }
};


template <typename Type>
class btElem{
public:
    void read_elem();
public:
    Type value; //корень, используем любой тип
    int left;   //индекс левого сына
    int right;  //индекс правого сына
};

template <class Type>                //шаблон класса
struct binTree {                     //бинарное дерево
    binTree(int n = 0): size(n)     //количество элементов в
массиве(дереве)
    {
        storage = new btElem<Type>[size];     //создание массива
        for (int i = 0; i < size; i++) {        //инициализация полей
струкуры
            storage[i].left = -1;  //значение правого сына(индекс в
массиве)
            storage[i].right = -1; //значение левого сына(индекс в
массиве)
        }
    }
    ~binTree() //деструктор
    {
        delete [] storage;              //очистка массива(освобождение
памяти, выделенной командой new[])
    }
    int fill_from_input(int&);
    void printTree(int index,Trunk *prev, bool isLeft);
    int depth(int root);
    int size;              //размер массива структур
    btElem<Type> * storage;  //объявление массива структур
```

```
};

template <class Type> //шаблон класса
struct Forest {
    Forest(int n = 0, int num_sons = 3, int fdepth = 0): size(n),
max_sons(num_sons + 1), depth(fdepth) //инициализация объекта класса,
по умолчанию будут 0... или пользовательские значения
    {
        storage = new bt_element[size];               //создание
массива сруктур
        for (int i=0; i < size; i++) {                  //заполнение
значений
            storage[i].value=0;                       //инициаизация
корня
            storage[i].sons = new int[max_sons];      //массив
указателей(индексов) на сыновей
            for (int j=0; j < max_sons; j++)
//инициализация сыновей
                storage[i].sons[j]=-1;                //нет сына
        }
    }
    ~Forest() //деструктор
    {
        for (int i=0; i<size; i++)
            delete [] storage[i].sons;         //очистка памяти,
выделенной под сыновей
        delete [] storage;                     //очистка структур леса
    }
    void print_forest(int i,int depth) {
        int j=0; //функция печати леса
        while (storage[i].sons[j]>-1) {
//пока есть сыновья
            print_forest(storage[i].sons[j],depth-1);      //выводим
сына
            j++;
//перемещаемся по массиву сыновей
        }
        std::string tabs(depth+1,'-');         //с увеличением глубины
увеличиваем "-"                               //кол-во сыновей(кол-во
напечатанных)
        if (storage[i].value) {                //если корень
            std::cout.width(this->depth+3);
//задаем ширину поля
```

```
            std::cout << storage[i].value << tabs << std::endl;
//выводим значение элемента дерева
        }

    }

    void print_horizontal(std::string * out, int i, int level) {
//функция обхода леса в ширину, указатель на массив строк(каждый
уровень отдельно)
        int j = 0;
//кол-во сыновей
        while (storage[i].sons[j] > -1) {
//пока есть элементы леса
            out[level]+=storage[storage[i].sons[j]].value;
//выводим значения
            j++;
        }
        j = 0;
        while (storage[i].sons[j] > -1) {
//пока есть деревья в лесу
            Forest::print_horizontal(out, storage[i].sons[j],
level+1);      //выводим
            j++;
        }

    }
    struct bt_element {    //структура элемента массива
        Type value;        //значение корня
        int * sons;        //указатель на массив сыновей
    };
    int size;              //размер массива указателей
    int max_sons;          //максимальное число сыновей(деревьев леса)
    bt_element * storage;  //массив структур
    int depth;             //глубина
};

template <typename T>
void btElem<T>::read_elem(){
    char c;
    for(c = getchar(); c == ' ' && c != '\n' && c != EOF; c =
getchar());
    if(c == '\n' || c == EOF){
        std::cout << "Incorrect input!" << std::endl;
        exit(1);
```

```
    }
    else
        ungetc(c, stdin);
    std::cin >> value;
    if(std::cin.fail()){
        std::cout << "Input error!" << std::endl;
        exit(1);
    }
    return;
}

template <typename T>
int binTree<T>::fill_from_input(int& root) { //заполнение дерева,
исходя их входных данных, указатель на объект класса, входная строка,
указатель на элемент строки, индекс отца
    int curr_pos = 0;
//текущая позиция внутри массива структур
    int left_son = 2*root + 1;
//в какой индекс записываем сыновей(левый)
    int right_son = 2*root + 2;
//правый
    if (root > -1) {
//указатель на отца
        if (storage[root].left == -1)
//сначала записываем левого сына
            curr_pos = left_son;
//потом правого
        else
            curr_pos = right_son;

    }
    char c;

    if (curr_pos >= size) return 0;
//проверка на переполнение
    while(1) {
        for(c = getchar(); c == ' ' && c != '\n' && c != EOF; c =
getchar());
        if (c == ')') {
            if (storage[curr_pos].left == 0)                    //нет
сыновей
                storage[curr_pos].left = -1;
            if (storage[curr_pos].right == 0)
                storage[curr_pos].right = -1;
```

```
            break;
        }
        if (c == '#') {                             //пустой левый сын
по умолчанию
            storage[curr_pos].left = -1;
            continue;
        }
        if (c == '(') {
            fill_from_input(curr_pos);    //новая глубина
            continue;
        }
        ungetc(c, stdin);
        storage[curr_pos].read_elem();
        if (root == -1) continue;                       //если нет
отца, то не заполняем
        if (curr_pos == left_son)                       //если
текущий элемент является левым сыном текущего отца (root)
            storage[root].left = left_son;
        if (curr_pos == right_son)
            storage[root].right = right_son;
//пропуск разделительных знаков
    }
    return 0;
}

void showTrunks(Trunk *p) {
    if (p == nullptr)
        return;
    showTrunks(p->prev);
    std::cout << p->str;
}

template <class Type>
void binTree<Type>::printTree(int index, Trunk *prev, bool isLeft){
    if (index == -1)
        return;

    std::string prev_str = "    ";
    Trunk *trunk = new Trunk(prev, prev_str);
    printTree(storage[index].right ,trunk, true);

    if (!prev)
        trunk->str = "---";
    else if (isLeft){
```

```
        trunk->str = ".---";
        prev_str = "  |";
    }
    else{
        trunk->str = "`---";
        prev->str = prev_str;
    }

    showTrunks(trunk);
    std::cout << storage[index].value << std::endl;
    if (prev)
        prev->str = prev_str;

    trunk->str = "  |";
    printTree(storage[index].left, trunk, false);
}

int max(int a, int b){
    if (a > b)
        return a;
    else
        return b;
}

template <class Type>
int binTree<Type>::depth(int root){
    if(root == -1)
        return 0;
    return max(depth(storage[root].right), depth(storage[root].left))
+ 1;
}

template <class Type>
void convert_bt_to_forest(const binTree<Type> * bt, Forest<Type> *
bt_forest,int father,int num_son,int * posf,int pos) { //функция
конвертирования дерева в лес, num_son - кол-во сыновой, указатель на
позицию в лесе/в бинарном дереве
    bt_forest->storage[*posf].value=bt->storage[pos].value;
//заполняем значение элемента леса корнем дерева(по индексу pos)
    bt_forest->storage[father].sons[num_son]=*posf;
//записываем индекс сына для отца
    if (bt->storage[pos].left>-1) {
//сначала идем по всем левым сыновьям вглубь
```

```
        (*posf)++;
//
        convert_bt_to_forest(bt,bt_forest,(*posf)-1,0,posf,bt-
>storage[pos].left);      //индекс сыновей = 0, тк они всегда первые
идут при обходе, ...left - переход к левому сыну
    }
    if (bt->storage[pos].right>-1) {
//проверка на правого сына, в обратном ходе рекурсии
        (*posf)++;
        convert_bt_to_forest(bt,bt_forest,father,num_son+1,posf,bt-
>storage[pos].right);//father у братьев один, num_son двигаемся по
сыновьям
    }
    return;
}
```