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. 9381 _____ Игнашов В.М.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Познакомиться с работой с деревьями, обработкой их узлов, обходу деревьев и анализа.

Задание.

6в. Задано бинарное дерево b типа BT с произвольным типом элементов. Используя очередь, напечатать все элементы дерева b по уровням: сначала - из корня дерева, затем (слева направо) - из узлов, сыновних по отношению к корню, затем (также слева направо) - из узлов, сыновних по отношению к этим узлам, и т. д.

Выполнение работы.

Для реализации программы был создан класс узла бинарного дерева. В классе присутствуют следующие поля:

- int elementsUnder – переменная, в которой содержится количество элементов в данном поддереве
- int depth – глубина относительно главного дерева
- string inputLine – строка, содержащая информацию о самом элементе и левом/правым наследником
- string c – элемент
- string leftStr – строка, содержащая информацию о левом элементе
- string rightStr – строка, содержащая информацию о правом элементе
- int left – индекс левого наследника
- int right – индекс правого наследника

А также, методы:

- int getElementsUnder() – метод, без аргументов, возвращающая количество элементов в данном поддереве
- string getC() – метод, без аргументов, возвращающая сам элемент
- int getLeft() – метод, без аргументов, возвращающий индекс левого наследника

- int getRight() – метод, без аргументов, возвращающий индекс правого наследника
- int divide() – метод, без аргументов, разделяющий строку inputLine на c, leftStr, rightStr, возвращает 0 при ошибке, 1 если ошибок не было
- int createTree() – метод, без аргументов, создает узел, возвращает 0 при ошибке, 1 если ошибок не было
- void printTree() – метод, без аргументов, выводящий дерево
- Конструктор BinPart(string, int) – принимает строку, описывающую конкретный узел, число, представляющее собой глубину и присваивает эти значения конкретному узлу.

Также, созданы глобальные переменные – массив ссылок (BinPart** vec) на элементы дерева(узлы) и переменная int last, в которой хранится индекс пустого места в массиве.

void main():

Главная функция, предлагает пользователю ввести способ ввода данных, считывает строку и выполняет основную задачу программы – Создает из заданной строки дерево, выделив память под главный элемент списка, с помощью метода createTree()(в случае, если ввод неверный – выводит ошибку и выходит из программы). Выводим дерево с помощью метода printTree(), создаем очередь элементов дерева. Первый элемент – главный элемент. Вызываем функцию bfs() – обход дерева по ширине, создавая очередь из элементов дерева по уровням. Выводим очередь, очищаем дерево.

void bfs(BinPart**, int, int):

Функция обхода дерева по ширине, принимает три аргумента – сама очередь, сейчас исследуемый объект и место, в которое добавляем новые. Если настоящий исследуемый элемент == месту в которое добавляем – выходим, тк очередь заполнена. Если индекс левого элемента не -1 – добавляем его в очередь, аналогично с правым. Запускаем заново функцию. В конечном итоге имеем в первом аргументе полную очередь.

Конструктор BinPart(string, int):

Принимает строку, описывающую конкретный узел, число, представляющее собой глубину и присваивает эти значения конкретному узлу.

Метод int divide():

Метод разделения строки на сам элемент, левую и правую часть. Сразу проверяем, если элементы закончились в конкретной ветке. Далее проверяем простейшие ошибки ввода. Отделяем строку от скобок, идем до первой скобки или ‘+’ (отсутствия элемента). Это строка левого элемента. Все, что до – сам элемент Аналогично ищем строку правого элемента. При отсутствии правого элемента заменяем строку на +, дабы упростить работу с алгоритмами.

Метод int createTree():

Метод создания самого узла, проверяем если ввод неверный, одновременно разделяем на левую, правую часть и сам элемент методом divide(). Если левая строка не ‘+’, значит в ней свои элементы-наследники, вызываем для левого элемента снова этот метод. Аналогично для правой части. При ошибке в создании наследников – выходим с ошибкой.

Метод void printTree():

Метод выводит картинку дерева – отступ в глубину элемента, сам элемент, пока не доходим до конца – повторяем.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	(a(b(d(z)(h))(e))(c(f(i)(j))(g(k(u)(l))(m(t)(p)))))	Creating a unit: left-|(b(d(z)(h))(e))| right-|(c(f(i)(j))(g(k(u)(l))(m(t)(p))))| __Creating b unit: left-|(d(z)(h))| right-|(e)| ____Creating d unit: left-|(z)| right-|(h)| ______Creating z unit: left-|+| right-|+| ______Creating h unit: left-|+| right-|+| ____Creating e unit: left-|+| right-|+| __Creating c unit: left-|(f(i)(j))| right-|(g(k(u)

		(l))(m(t)(p)))| ____Creating f unit: left-|(i)| right-|(j)| _____Creating i unit: left-|+| right-|+| _____Creating j unit: left-|+| right-|+| ____Creating g unit: left-|(k(u)(l))| right-|(m(t)(p))| _____Creating k unit: left-|(u)| right-|(l)| _____Creating u unit: left-|+| right-|+| _____Creating l unit: left-|+| right-|+| _____Creating m unit: left-|(t)| right-|(p)| _____Creating t unit: left-|+| right-|+| _____Creating p unit: left-|+| right-|+| a |--b | |--d | | |--z | | |--h | |--e |--c | |--f | | |--i | | |--j | |--g | | |--k | | | |--u | | | |--l | | |--m | | | |--t | | | |--p For |a| adding |b| |c| to queue For |b| adding |d| |e| to queue For |c| adding |f| |g| to queue For |d| adding |z| |h| to queue

		For |e| adding nothing to queue For |f| adding |i| |j| to queue For |g| adding |k| |m| to queue For |z| adding nothing to queue For |h| adding nothing to queue For |i| adding nothing to queue For |j| adding nothing to queue For |k| adding |u| |l| to queue For |m| adding |t| |p| to queue For |u| adding nothing to queue For |l| adding nothing to queue For |t| adding nothing to queue For |p| adding nothing to queue BFS: abcdefgzhijkmultp
2.	(a(b(d+(h))(e))(c(f(i)(j))(g+(k(l)))))	Creating a unit: left-|(b(d+(h))(e))| right-|(c(f(i)(j))(g+(k(l))))| __Creating b unit: left-|(d+(h))| right-|(e)| ____Creating d unit: left-|+| right-|(h)| _____Creating h unit: left-|+| right-|+| ____Creating e unit: left-|+| right-|+| __Creating c unit: left-|(f(i)(j))| right-|(g+(k(l)))| ____Creating f unit: left-|(i)| right-|(j)| _____Creating i unit: left-|+| right-|+| _____Creating j unit: left-|+| right-|+| ____Creating g unit: left-|+| right-|(k(l))| _____Creating k unit: left-|(l)| right-|+| _____Creating l unit: left-|+| right-|+| a |--b | |--d | | |--h | |--e

		|--c | |--f | | |--i | | |--j | |--g | | |--k | | | |--l For |a| adding |b| |c| to queue For |b| adding |d| |e| to queue For |c| adding |f| |g| to queue For |d| adding |h| to queue For |e| adding nothing to queue For |f| adding |i| |j| to queue For |g| adding |k| to queue For |h| adding nothing to queue For |i| adding nothing to queue For |j| adding nothing to queue For |k| adding |l| to queue For |l| adding nothing to queue BFS: abcdefghijkl
3.	((())))	Error!
4.	(a(b)(c))	Creating a unit: left-|(b)| right-|(c)| __Creating b unit: left-|+| right-|+| __Creating c unit: left-|+| right-|+| a |--b |--c For |a| adding |b| |c| to queue For |b| adding nothing to queue

		For |c| adding nothing to queue BFS: abc
5.		Error!

Выводы.

Были изучены методы работы с деревьями, методы их анализа, обработки, построения.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <string>
#include <fstream>

//(a(b(d(z)(h))(e))(c(f(i)(j))(g(k(u)(l))(m(t)(p)))))
//(a(b(d+(h))(e))(c(f(i)(j))(g+(k(l)))))

using namespace std;

class BinPart{
private:
    int elementsUnder=1;//Количество элементов в дереве
    int depth=0;
    string inputLine;//Строка узла/листа
    string c;//Содержание
    string leftStr;//Левая строка
    string rightStr;//Правая строка
    int left;//Ссылка на левую часть
    int right;//Ссылка на правую часть
public:
    int getElementsUnder();
    string getC();
    int getLeft();
    int getRight();
    int divide();
    int createTree();
    void printTree();
    BinPart(string inputLine, int depth);
};

BinPart** vec=new BinPart*[1024];
int last=0;

BinPart::BinPart(string inputLine, int depth) {//Конструктор
    this->inputLine=inputLine;
    this->depth=depth;
}

int BinPart::divide(){//Функция разделения строки узла/листа на поля
    //Чек простейших ошибок ввода
    if(inputLine.compare("+")==0){
        return 1;
    }
    if(!(inputLine.length()>2&&inputLine[0]=='('&&inputLine[inputLine.length()-
1]==')'))
        return 0;
    //Отделение строки от скобочек
    inputLine=inputLine.substr(1,inputLine.length()-2);

    if(inputLine[0]=='('||inputLine[0]==')'||inputLine[0]=='+')
        return 0;

    int i=0;
    //Встречаем левую часть
    while(inputLine[i]!='+'&&inputLine[i]!='(') {
        i++;
        if (i == inputLine.length()) {
            c = inputLine;
            leftStr = "+";
```

```
            rightStr = "+";
            return 1;
        }
    }
    //Содержание самого узла
    c=inputLine.substr(0,i);
    //Если левая часть - отсутствует
    if(inputLine[i]=='+'){
        leftStr="+";
        i++;
        rightStr=inputLine.substr(i,inputLine.length()-i);
    }else{//Иначе ищем ее определение
        int k=0;
        while(inputLine[i-1]!=')'||k!=0){
            if(inputLine[i]=='(')
                k++;
            if(inputLine[i]==')')
                k--;
            i++;
            if(i==inputLine.length()+1){
                return 0;
            }
        }
        leftStr=inputLine.substr(c.length(),i-c.length());
        rightStr=inputLine.substr(i,inputLine.length()-i);
        //Правая часть не записана => отсутствует
        if(rightStr.compare("")==0){
            rightStr="+";
        }
    }

    return 1;
}

int BinPart::createTree(){//Функция создания узла/листа
    //Чек на неверный ввод
    if(divide()==0){
        return 0;
    }

    for(int i=0;i<depth;i++){
        cout << "__";
    }

    cout << "Creating " << c <<" unit: left-|" << leftStr << "| right-|" <<
rightStr << "|" << endl;
    //Рекурсия для левой части
    if(leftStr!="+") {
        left=last;
        last++;
        vec[left] = new BinPart(leftStr, depth+1);
        if (vec[left]->createTree() == 0) {
            return 0;
        }else{
            elementsUnder+=vec[left]->getElementsUnder();
        }
    }else{
        left = -1;
    }
    //Рекурсия для правой части
    if(rightStr!="+") {
        right=last;
        last++;
        vec[right] = new BinPart(rightStr, depth+1);
        if(vec[right]->createTree()==0){
            return 0;
```

```
        }else{
            elementsUnder+=vec[right]->getElementsUnder();
        }
    }else{
        right = -1;
    }

    return 1;
}

void BinPart::printTree() {//Функция печати дерева

    for (int i = 0; i < depth - 1; i++) {
        cout << "|   ";
    }
    if (depth != 0)
        cout << "|--";
    cout << c << endl;
    if (left != -1)
        vec[left]->printTree();
    if (right != -1)
        vec[right]->printTree();
}

void bfs(BinPart** queue, int now, int lastBfs){//Функция обхода дерева по
ширине
    //Если последний элемент в очереди  и есть исследуемый - значит, других нет
    if(now==lastBfs)
        return;
    //Добавление в очередь левого
    cout << "For |" << queue[now]->getC() << "| adding ";
    if(queue[now]->getLeft() == -1 && queue[now]->getRight() == -1 )
        cout << "nothing ";
    if(queue[now]->getLeft() != -1) {
        queue[lastBfs] = vec[queue[now]->getLeft()];
        cout << "|"<< queue[lastBfs]->getC() << "| ";
        lastBfs++;
    }
    //Добавление в очередь правого
    if(queue[now]->getRight() != -1) {
        queue[lastBfs] = vec[queue[now]->getRight()];
        cout << "|"<< queue[lastBfs]->getC() << "| ";
        lastBfs++;
    }
    cout << "to queue" << endl;
    //Сдвиг по очереди
    bfs(queue,now+1,lastBfs);
}

int main(){
    //Ввод входной строки
    int choose=0;
    cout << "Choose input:\n0.CIN\n1.FILE\n";
    cin >> choose;//Выбираем между вводом с консоли и из файла
    string inputLine;
    if(choose==0){
        cin.ignore(1);
        getline(cin, inputLine);
    }else{
        ifstream ff;
        ff.open("Lab3Input.txt");
        if(ff.is_open())
            getline(ff, inputLine);
        else
            return 0;
        cout << "In file: " << inputLine << endl << endl;
```

```
    }

    BinPart *tree = new BinPart(inputLine, 0);
    //Запуск алгоритма
    if(tree->createTree()==0){
        cout << "Error!";
        system("pause");
        return 0;
    };
    //Вывод дерева
    cout << endl;
    tree->printTree();
    cout << endl;
    //Создание очереди
    BinPart** queue= new BinPart*[tree->getElementsUnder()];
    queue[0]=tree;
    bfs(queue,0,1);
    //Вывод очереди
    cout << endl << "BFS: ";
    for(int i=0;i<tree->getElementsUnder();i++){
        cout << queue[i]->getC();
    }
    cout << endl;
    //Освобождение памяти
    for(int i=0;i<last;i++)
        delete vec[i];
    delete[] vec;
    delete[] queue;

    system("pause");

    return 0;
}
//Геттеры
int BinPart::getElementsUnder(){
    return elementsUnder;
}

string BinPart::getC(){
    return c;
}

int BinPart::getLeft(){
    return left;
}

int BinPart::getRight(){
    return right;
}
```