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

ОТЧЁТ

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

Студент гр. 9303 _____ Эйсвальд М.И.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

Цель работы.

Изучить бинарное дерево как структуру данных, научиться реализовывать бинарное дерево самостоятельно, приобрести навыки работы с бинарными деревьями.

Задание.

Вариант **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.

Основные теоретические положения.

Бинарное дерево — конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся *бинарных деревьев*, называемых правым поддеревом и левым поддеревом.

Поскольку определение данной структуры данных рекурсивно, как правило, работа с бинарным деревом производится с помощью рекурсивных процедур и функций.

Скобочная запись бинарного дерева определяется так:

< БД > ::= < пусто > | < непустое БД >,

< пусто > ::= Λ,

< непустое БД > ::= (< корень > < БД > < БД >).

Выполнение работы.

Реализация бинарного дерева на базе вектора в данной работе выглядит следующим образом: узлы бинарного дерева, имеющие поле `data` для хранения данных и указатели на правое и левое поддеревья, записываются в массив в порядке их создания; при этом у отцов вновь созданных узлов появляются указатели на добавленные в массив узлы. С точки зрения клиентского кода такое бинарное дерево выглядит точно так же, как и реализованное в динамической памяти: имея указатель на узел, можно получить доступ к его сыновьям, но нельзя получить доступ к его «соседям» по массиву, так

как поля класса дерева, относящиеся к его внутренней реализации, сделаны приватными.

Считывать дерево было решено в упрощённой скобочной записи, но без внешних окаймляющих скобок. Для обозначения пустого поддерева (это необходимо, если у корня есть правый сын, но нет левого) используется пробел. Для чтения дерева был реализован метод `read()` класса бинарного дерева. Метод считывает очередной элемент дерева в массив, после чего, если у элемента есть сыновья (на что указывает открывающая скобка), считывает сыновей и записывает в нужные поля исходного элемента их адреса. Процедура чтения рекурсивна: если у элемента есть сыновья, чтение элемента включает чтение его сыновей.

Очередь для печати элементов дерева по уровням также была реализована самостоятельно. Помимо стандартных функций, класс очереди также содержит метод `pushTree()` для упорядоченного добавления в очередь элементов разных уровней. Сначала в очередь добавляется корень дерева, потом в цикле из очереди извлекаются все элементы, а взамен добавляются их сыновья, пока на каком-то этапе очередь не окажется пустой — это значит, что у узлов предыдущего уровня уже нет сыновей.

Функция печати очереди также является методом класса очереди, а не класса дерева. Она печатает элементы сформированной очереди по уровням, оформляя информацию в удобочитаемой форме.

Полный код программы см. в приложении А.

Тестирование.

Скриншоты с результатами тестирования программы приведены в приложении Б.

Вывод.

В ходе выполнения лабораторной работы была изучена очередная рекурсивно определённая структура данных — бинарное дерево. Был реализован класс бинарного дерева, использующий вектор для хранения узлов дерева. Результатом работы стала программа, строящая бинарное дерево по его скобочному представлению и выводящая его элементы по уровням.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: binarytree.h

```
#ifndef BINARYTREE_H
#define BINARYTREE_H

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

#define SIZE 100

using namespace std;

namespace BT{

//Node---------------------------------

template<typename T>
struct Node{
public:
Node<T>* left;
Node<T>* right;
Node();
void print(ofstream* outfile);
void read(ifstream* infile);
T getData();
private:
T data;
};

template<typename T>
Node<T>::Node() : left(nullptr), right(nullptr) {}

template<typename T>
void Node<T>::print(ofstream* outfile){
std::cout << ”[” << this->data << ”]”;
(*outfile) << ”[” << this->data << ”]”;
}

template<typename T>
T Node<T>::getData(){
return this->data;
}

template<typename T>
void Node<T>::read(ifstream* infile){
T tmp;
(infile->is_open() ? *infile : std::cin) >> tmp;
cout << tmp;//
this->data = tmp;
}

//pair----------------------

template<typename T>
class Pair{
```

```
public:
Node<T>* ptr;
int level;
Pair(Node<T>* ptr, int level) : ptr(ptr) , level(level) {}
Pair() {}
};

//customQueue--------------------

template<typename T>
struct CustomQueue{
public:
CustomQueue(ofstream* pointer);
void push(Pair<T>);
Pair<T> pop();
bool empty();
bool full();
void clear();
void pushTree(Node<T>*, int);
void print();
int size();
private:
Pair<T> array[SIZE];
int start; //первый незаполненный
int end; //последний заполненный
int length;
ofstream* outfile;
};

template<typename T>
CustomQueue<T>::CustomQueue(ofstream* pointer) :
 start(0), end(0),
 length(0), outfile(pointer) {}

template<typename T>
void CustomQueue<T>::clear(){
start = 0;
end = 0;
length = 0;
}

template<typename T>
bool CustomQueue<T>::full(){
return (length == SIZE);
}

template<typename T>
bool CustomQueue<T>::empty(){
return (!length);
}

template<typename T>
int CustomQueue<T>::size(){
return length;
}

template<typename T>
void CustomQueue<T>::push(Pair<T> element){
if(this->full()){
std::cout << ”Attempt to push to the full queue rejected. Exiting...\n”;
```

```
exit(1);
}
array[start] = element;
start = (start + 1) % SIZE;
++length;
}

template<typename T>
Pair<T> CustomQueue<T>::pop(){
if(this->empty()){
std::cout << ”Tried to pop from the empty queue. Exiting...\n”;
exit(1);
}
--length;
int tmp = end;
end = (end + 1) % SIZE;
return array[tmp];
}

template<typename T>
void CustomQueue<T>::pushTree(Node<T>* root, int level){
CustomQueue<T>* tmp = new CustomQueue(nullptr);

int counter = 0;
int length;
Pair<T> pair;
pair.ptr = root;
pair.level = counter;
tmp->push(pair);
while(!tmp->empty()){
++counter;
length = tmp->size();
for(int i = 0; i < length; ++i){
pair = tmp->pop();
this->push(pair);
if(pair.ptr->left) tmp->push(Pair<T>(pair.ptr->left , counter));
if(pair.ptr->right) tmp->push(Pair<T>(pair.ptr->right , counter));
}
}
delete tmp;
}

template<typename T>
void CustomQueue<T>::print(){
if(this->empty()) return;
Pair<T> elem;
int width = 4*ceil((this->length)/2) - 1 + 9;
Pair<T> prev = this->pop();
std::cout << ”Level ” << prev.level << ”: ”;
(*outfile) << ”Level ” << prev.level << ”: ”;
prev.ptr->print(outfile);
while(!this->empty()){
elem = this->pop();
if(elem.level > prev.level){
std::cout << ”\n”;
(*outfile) << ’\n’;
for(int i = 0; i < width; ++i) {std::cout << ”-”; (*outfile) << ”-”;}
std::cout << ’\n’;
(*outfile) << ’\n’;
std::cout << ”Level ” << elem.level << ”: ”;
```

```
(*outfile) << ”Level ” << elem.level << ”: ”;
}
elem.ptr->print(outfile);
prev = elem;
}
}

//tree-----------------------------

template<typename T>
struct BinTree{
public:
BinTree(ofstream* ptr);
void read(ifstream* infile);
void reset();
Node<T>* root();
private:
Node<T> array[SIZE];
int top;
ofstream* outfile;
};

template<typename T>
BinTree<T>::BinTree(ofstream* ptr): top(0), outfile(ptr) {}

template<typename T>
void BinTree<T>::reset(){
top = 0;
for(int i = 0; i < SIZE; ++i) array[i] = Node<T>();
}

template<typename T>
Node<T>* BinTree<T>::root(){
return array;
}

template<typename T>
void BinTree<T>::read(ifstream* infile){
char chr;
array[top].read(infile);
(*outfile) << array[top].getData();
int tmp = top;
(infile->is_open() ? *infile : std::cin) >> chr;
if( (infile->is_open() && infile->eof()) || (!(infile->is_open())) && cin.eof()){
cout << ”\b”;
exit(0);
}
switch(chr){
case ’(’:
cout << ’(’; (*outfile) << ’(’;
(infile->is_open() ? *infile : std::cin) >> chr;
switch(chr){
case ’ ’:
cout << ’ ’; (*outfile) << ’ ’;
break;
case ’)’:
std::cout << chr << ” <- ERROR\n”;
(*outfile) << chr << ” <- ERROR” << endl;
exit(1);
default:
```

```
(infile->is_open() ? *infile : std::cin).unget();
++top;
array[tmp].left = array + top;
this->read(infile);
}
(infile->is_open() ? *infile : std::cin) >> chr;
switch(chr){
case '(':
std::cout << chr << " <- ERROR\n";
(*outfile) << chr << " <- ERROR" << endl;
exit(1);
case ')':
std::cout << ')'; (*outfile) << ')';
break;
case ' ':
std::cout << chr << " <- ERROR\n";
(*outfile) << chr << " <- ERROR" << endl;
                        exit(1);
default:
(infile->is_open() ? *infile : std::cin).unget();
++top;
array[tmp].right = array+top;
read(infile);
//считать оставшуюся скобку?
(infile->is_open() ? *infile : std::cin) >> chr;
if(chr != ')'){
std::cout << chr;
std::cout << " <- ERROR\n";
(*outfile) << chr << " <- ERROR" << endl;
exit(1);
}
cout << ')'; (*outfile) << ")";
}
break;
case'\n':
(infile->is_open() ? *infile : std::cin).unget();
//infile->is_open() ? std::cin : std::cin.unget();
return;
default:
(infile->is_open() ? *infile : std::cin).unget();
}
}

}

#endif
```

Название файла: main.cpp

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

#include "binarytree.h"

#define OUTFILE "output.txt"

int main(){

std::string str;
```

```
std::cout << ”Specify input file name: ”;
getline(std::cin, str);
std::ifstream infile(str.c_str());
std::ofstream outfile(OUTFILE);
if(!outfile.is_open()){
std::cout << ”Failed to create output file. Exiting...\n”;
return 0;
}

BT::BinTree<char> tree = BT::BinTree<char>(&outfile);

if(!infile.is_open()){
std::cout << ”Failed to open file. Expecting input from console...\n”;
std::cout << ”Enter EOF (i.e. Ctrl+D) to finish.\n”;
std::cin >> noskipws;
}
else{
infile >> noskipws;
}

BT::CustomQueue<char> queue(&outfile);

do{
tree.read(&infile);
getline( (infile.is_open() ? infile : std::cin) , str);
std::cout << ”\n”;
std::cout << ”Finished reading...\n”;
outfile << ”\nFinished reading...” << std::endl;

queue.pushTree(tree.root(),0);

std::cout << ”\nTree levels:\n”;
outfile << ”\nTree levels:\n”;
queue.print();
std::cout << ”\n”;
outfile << std::endl;

tree.reset();
queue.clear();
}while(1);
return 0;
}
```

ПРИЛОЖЕНИЕ Б
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ

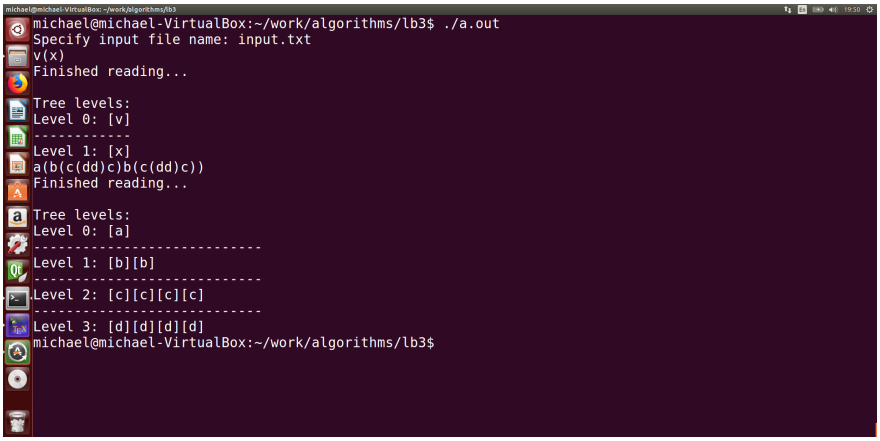

Рисунок 1 – Обработка данных из файла

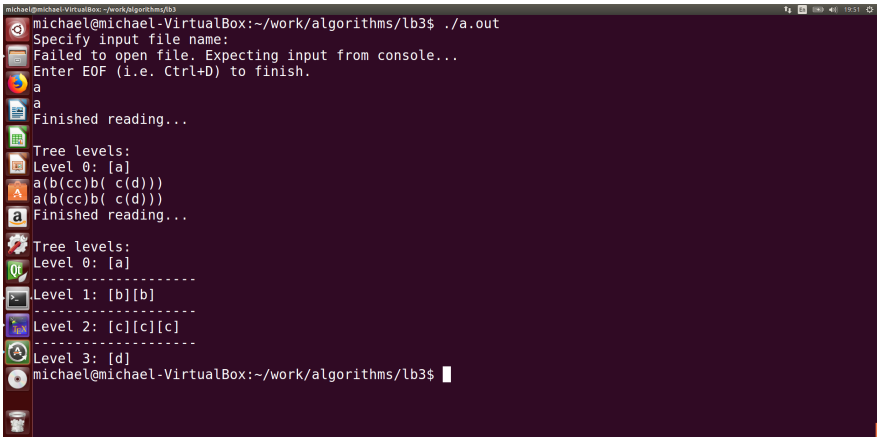

Рисунок 2 – Обработка данных с консоли

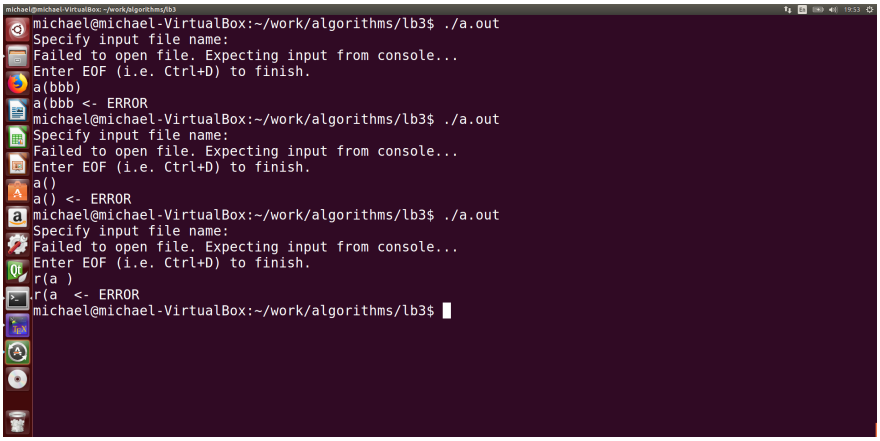

Рисунок 3 – Обработка ошибок