**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 |  | Федоров И.А |
| Преподаватель |  | Фирсов М.А |

Санкт-Петербург

2019

**Цель работы.**

Ознакомиться с абстрактными нелинейными структурами данных, такими как деревья и способами работы с ними. Изучить методы обработки деревьев, а также способы обработки бинарных деревьев.

**Основные теоретические положения.**

*Дерево* – конечное множество *Т*, 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* ≥ 0 попарно не пересекающихся множествах *Т*1, *Т*2, ..., *Тm*, каждое из которых, в свою очередь, является деревом. Деревья *Т*1, *Т*2, ..., *Тm*называются *поддеревьями* данного дерева.

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

Наиболее важным типом деревьев являются *бинарные деревья*. Удобно дать следующее формальное определение. *Бинарное дерево* − конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом. Формально бинарное дерево имеет такое определение:

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

< пусто > ::= Λ,

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

Здесь пустое дерево имеет специальное обозначение − Λ.

Например, бинарное дерево имеет полное скобочное представление вида:

(*a* (*b* (*d* Λ (*h* Λ Λ)) (*e* Λ Λ)) (*c* (*f* (*i* Λ Λ) (*j* Λ Λ)) (*g* Λ (*k* (*l* Λ Λ) Λ)))).

**Задание.**

Вариант 5-д (динамически): Задано бинарное дерево *b* типа *BT* с произвольным типом элементов. Используя очередь и операции над ней, 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.

Тип содержимого узлов деревьев должен быть шаблонным (template в c++). При этом в качестве параметра шаблона достаточно использовать лишь один тип (например, char).

**Реализация**

Для представления дерева в памяти в данной работе используется следующий способ – разветвляющийся список. Узлы дерева - объекты, связи между которыми осуществляются через указатели. Графический вид приведен на рис.1.

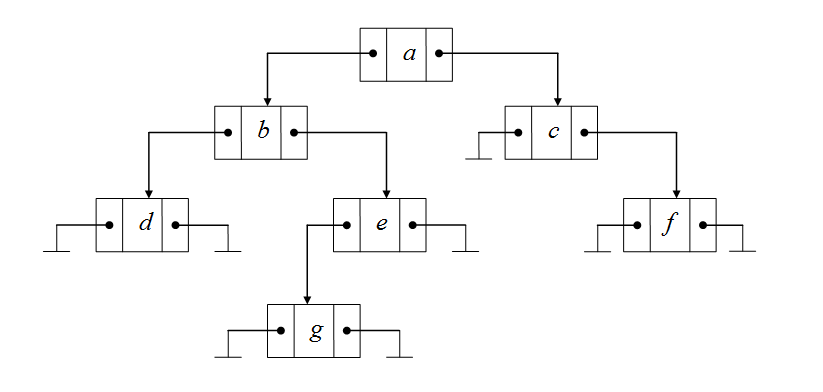


Рисунок 1 - Ссылочное представление дерева

Пустому дереву в данном случае соответствует указатель на NULL.

**template** <**typename** base> **//указываем как шаблонный**

**class** node {

public: **//структура узла**

base data;

node<base> \*left; **//Указатели на левого и**

node<base> \*right; **//правого сыновей**

node ();

**friend** **class** Tree;

};

**typedef** node<**char**> \*binTree;

**class** tree{ **// вспомогательный класс "дерево"**

**public**:

binTree root;

**int** width,depth,offset; **//глубина, смещение корня**

**char**\*\* screen;

**void** outTree(); **//вспомогательный функции для**

**void** clrscr(); **//отображения дерева в консоли**

**void** outNodes(binTree,**int**,**int**); };

**template** <**typename** Qtype> **class** node1 { // структура элемента очереди

**public**:

Qtype data;

node1<Qtype>\* next;

node1<Qtype>\* prev;

node1() { next = prev = NULL;}

};

//typedef node\* pNode;

**template** <**typename** Qtype> **class** Queue{

node1<Qtype> \*head;

node1<Qtype> \*tail;

**public**:

Queue() { head = tail = NULL;}

**void** pushTail(Qtype x);

Qtype popHead();

**void** printQueue();

**bool** empty();

};

1) Структура ***node*** является структурой узла бинарного дерева и содержит следующие элементы:

• ***data***- значение узла бинарного дерева.

• ***left*** - указатель на объект структуры node, являющейся левым "сыном" данного дерева.

• ***right***- указатель на правого "сына" данного бинарного дерева.

В качестве "дружественного" класса объявлен вспомогательный класс *tree.* Он необходим для отображения в наглядном виде заданного бинарного дерева в консоли. Класс *tree* хранит глубину заданного дерева *depth*, а также матрицу *screen*, в которой будет хранится расположение узлом дерева для вывода.

2) binTree retLeft(binTree obj);

binTree retRight(binTree obj);

Функции *retLeft* и *retRight* возвращают указатель на левого/правого сына данного узла.

3) bool isNull ( binTree b );

Функция *isNull* проверяет, является ли переданный ей узел пустым (указателем на NULL), и возвращает *true* или *false* в зависимости от этого.

4) binTree сonsBT(base x, binTree &lst, binTree &rst);

Функция *consBt* – функция-конструктор, которая создает бинарное дерево из заданного узла и двух бинарных деревьев, которые станут правым и левыми сыновьям созданного дерева.

5) void Tree::outTree();

void Tree::outNodes;

Функции *outTree и outNodes* – функции-члены класса *tree*, которые выводят заданное бинарное дерево в наглядном виде в консоль.

6) void passWidth (binTree b);

Функция***passWidth***– основная функция, выводящая элементы дерева по уровням: сначала из корня, затем слева направо из узлов, сыновних корню, затем также из узлов, сыновних этим узлам и т.д. Для этого функция выполняет обход в ширину заданного бинарного дерева, используя при этом очередь. Алгоритм заключается в следующем: функция заносит корень дерева в очередь ***queue***, затем функция просматривает левого и правого сыновей (если они есть) и так же кладет их в очередь, при этом удаляя первый ее элемент (в данном случае корень). При этом удаленный элемент выводится в консоль (тем самым в конце работы будет выведен список элементов дерева в порядке обхода в ширину). Дальше аналогичные действия выполняются для левого (***left***) и правого (***right***) сыновей - их сыновья также помещаются в очередь, а сами элементы удаляются из нее. Условием конца работы алгоритма является пустая очередь (значит что все элементы просмотрены). Графический вид работы приведена на рис. 2.

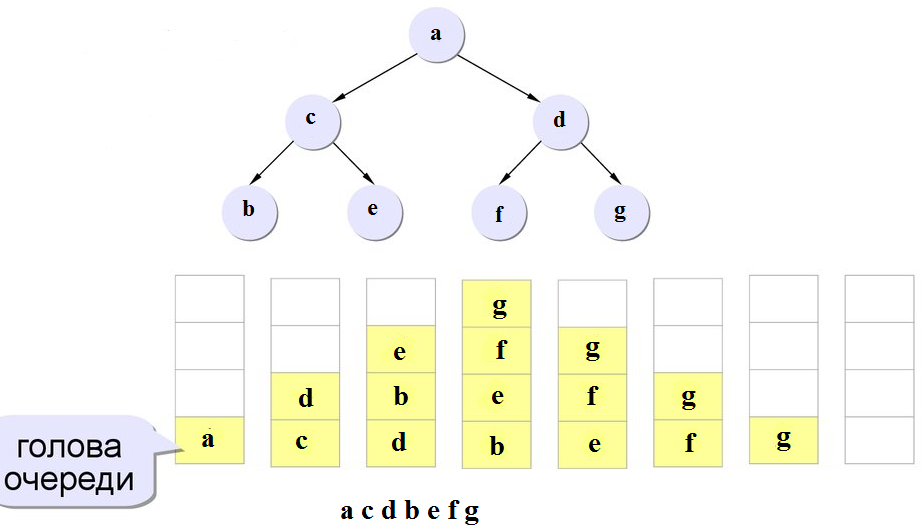


Рисунок 2 - Графический вид работы алгоритма

Кратко алгоритм можно описать так:

1. Положить элемент в очередь.
2. Просмотреть сыновей и занести их в очередь.
3. Удалить элемент из очереди (вывести его).
4. Если очередь пуста, конец работы алгоритма.

7) binTree enterBT (istream& stream);

Функция *enterBT*- рекурсивная функция, отвечающая за ввод бинарного дерева из файла или из консоли (в зависимости от параметра ***stream***), заданного в виде КЛП, где символ "**/**" будет означать отсутствие поддерева. Функция создает бинарное дерево и затем рекурсивно вызывает себя для левого и правого сыновей.

8) int hBT ( binTree b);

Функция *hBT* вычисляет максимальную высоту переданного бинарного дерева и возвращает его значение.

9)void readRoot ( istream &stream, binTree& obj);

void readLeft (istream &stream, binTree& obj);

void readRight( istream &stream, binTree& obj);

Функции *readRoot, readLeft и readRight* – рекурсивные функции, которые считывают и создают дерево, заданное в полной скобочной записи. Функции рекурсивно вызывают друг друга и создают узлы дерева путема вызова фунции-конструктура *consBT*, который описывался ранее. В качестве параметров функции передается поток, из которого будет считано дерево, и указатель на узел дерева (в начале передается корень).

10) Основная функция программы (main) создает указатель на бинарное дерево ***bintree*** и осуществляет ввод данных из консоли, сопровождая его соответствующим диалогом. Пользователь выбирает один из способов ввода дерева (из файла или из консоли) и способ задания дерева (в форме КЛП или в форме полной скобочной записи, пустой узел обозначается как / или **#**). После обработки пользователь может продолжить работу с программой или завершить её.

**Выводы.**

Ознакомился с абстрактными нелинейными структурами данных, такими как деревья, а в частности, с бинарными деревьями. Ознакомился с основами обработки бинарного дерева и способами его обхода. Написал программу, выводящую узлы дерева в порядке обхода дерева в ширину с использованием очереди.

Протокол

Таблица 1 - тестирование

|  |  |  |
| --- | --- | --- |
| № теста | Входные данные | Выходные данные |
| 1. | (a # #) | a |
| 2. | ( a (b #(o##)) (c(e##) #) ) | a b c o e |
| 3. | abcdefg**//// ////** | a b c d e f g |
| 4. | ( ) | Ошибка! Некорректный ввод. |
| 5. | abd**/**g**///**ce**//**fi**//**jk**///** | a b c d e f g i j k |
| 6. | (# # #) | Ошибка! Нельзя вводить пустое дерево. |
| 7. | (a (b (d **/** (h **/ /**)) (e **/ /**)) (c (f (i **/ /**) (j **/ /**)) (g **/** (k (l / /) **/**)))) | a b c d e f g h i j k l |
| 8. | (a # (c )) | Ошибка! Неправильный ввод. |
| 9. | ( a (b ##) (c (e (f##) (g##) ) (h (m ##) (l##)))) | a b c e h f g m l |
| 10. | "пустой файл" | Ошибка! Не найден конец дерева. |

Таблица 2 - из файла.

|  |  |  |  |
| --- | --- | --- | --- |
| № теста | Содержимое файла | Выходные данные | Комментарий |
| 1. | "пустой файл" | Ошибка! Не найден конец дерева. | Так как файл пуст, то выводится сообщение об ошибке. |
| 2. | abd**/**g**///**ce**//**fi**//**jk**///** | a b c d e f g i j k | - |
| 3. | (a (b (d **/** (h **/ /**)) (e **/ /**)) (c (f (i **/ /**) (j **/ /**)) (g **/** (k (l / /) **/**)))) | a b c d e f g h i j k l | - |

Действия программа при некорректном вводе (см. рис 3):

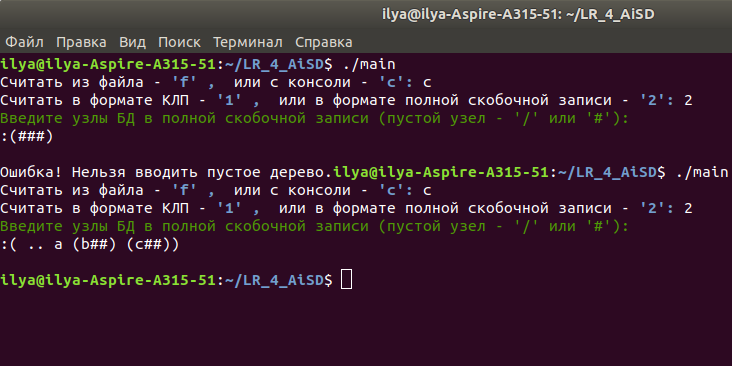


Рисунок3 - (неверный ввод)

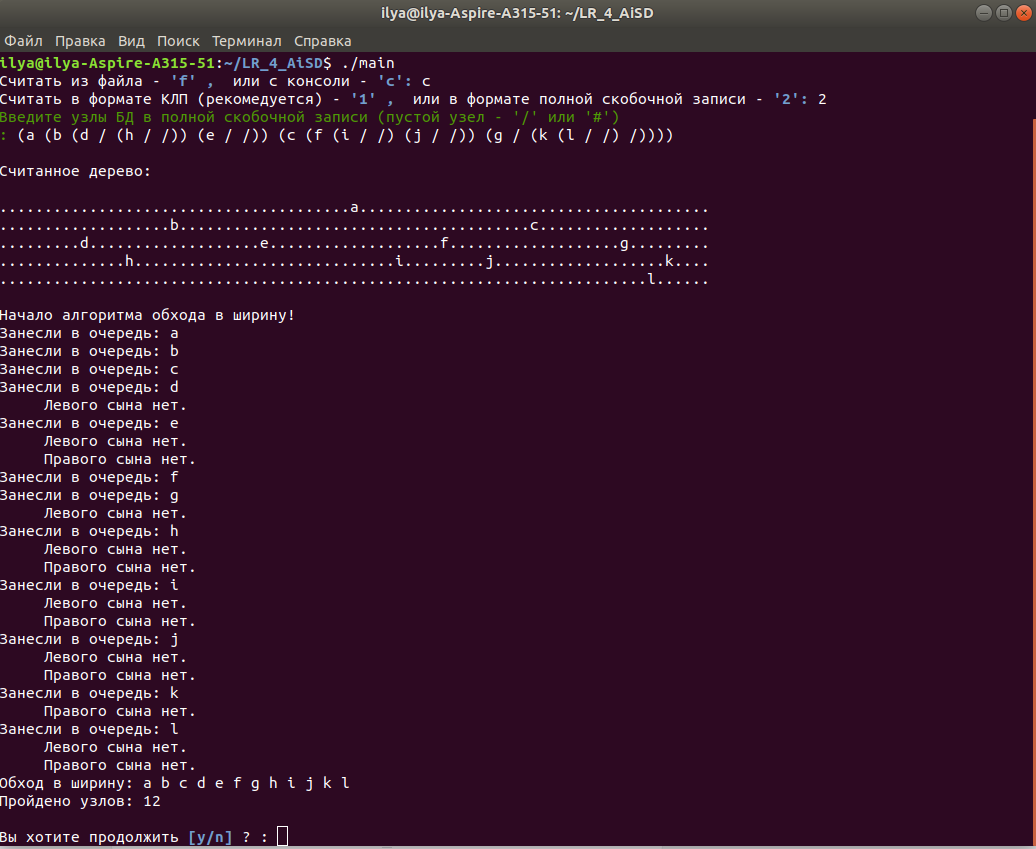
Пример полной работы (см. рис 4)

Рисунок 4 - Полная работа программы

**Приложение (код программы).**

#include <iostream>

#include <fstream>

#include <fstream>

#include <cstring>

#include <cstdlib>

#include <queue>

#include "Queue.h"

#define Len 80

**using** **namespace** std;

string file = "f";

string con = "c";

string stop = "n";

string again = "y";

**template** <**typename** base>

**class** node {

**public**:

base data;

node<base> \*left;

node<base> \*right;

node () { left = NULL; right = NULL;}

**friend** **class** Tree;

};

**typedef** node<**char**> \*binTree; **// "представитель" бинарного дерева**

**class** Tree{

**public**:

binTree root;

**int** width;

**int** maxrow, offset; **// максимальная глубина, смещение корня**

**char**\*\* Screen;

**void** OutTree();

**void** clrscr();

**void** OutNodes(binTree v, **int** r, **int** c); **// выдача поддерева**

Tree(**int** maxrow, binTree rot);

~Tree();

};

binTree Left(binTree obj){ **// для непустого бин.дерева**

**if** (obj == NULL) {

cerr << "Error: Left(null) \n"; exit(1); **//переданный узел нулевой**

}

**else**

**return** obj->left;

}

binTree Right(binTree obj){ **// для непустого бин.дерева**

**if** (obj == NULL) {

cerr << "Error: Right(null) \n"; **//переданный узел нулевой**

exit(1);

}

**else**

**return** obj->right;

}

**template** <**typename** base>

binTree ConsBT(base x, binTree &lst, binTree &rst){

binTree p;

p = **new** node<base>;

**if** (p != NULL) {

p->data = x;

p->left = lst;

p->right = rst;

**return** p;

}

**else** {

cerr << "Memory not enough\n"; exit(1);

}

}

**void** destroy (binTree &tmp){

**if** (tmp != NULL) {

destroy (tmp->left);

destroy (tmp->right);

**delete** tmp;

tmp = NULL;

}

}

**bool** isNull(binTree b){

**return** (b == NULL);

}

**void** outputTree(binTree Ptr, **int** totalSpace)

{

**while**(Ptr != NULL)

{

outputTree(Ptr -> right, totalSpace + 5);

**for**(**int** i = 1; i <= totalSpace; i++)

cout << " ";

cout << Ptr -> data << "\n";

Ptr = Ptr -> left;

totalSpace += 5;

}

}

**///////////////////////////////**

**//Функции класса**

Tree::Tree(**int** depth, binTree rot){

maxrow = depth;

root = rot; width = 80;

**if** (maxrow > 5)

width = 120;

offset = width/2;

Screen = **new** **char**\* [maxrow];

**for**(**int** i = 0; i < maxrow; i++)

Screen[i] = **new** **char**[width];

}

Tree::~Tree(){

**for**(**int** i = 0; i < maxrow; i++)

**delete** []Screen[i];

**delete** []Screen;

//delete root;

}

**void** Tree::OutTree(){

clrscr();

OutNodes(root, 1, offset);

**if** (root == NULL){

cout << " ерево пусто ";

**return**;

}

**for**(**int** i = 0; i < maxrow; i++){

Screen[i][width-1] = 0;

cout << '\n' << Screen[i];

}

cout << '\n';

}

**void** Tree::clrscr(){

**for**(**int** i = 0; i < maxrow; i++)

memset(Screen[i], '.', width);

}

**void** Tree::OutNodes(binTree v, **int** r, **int** c){

**if** (r && c && (c < width))

Screen[r-1][c-1] = v->data; // вывод метки

**if** (r < maxrow){

**if** (v->left) OutNodes(v->left, r+1, c - (offset >> r));

**if** (v->right) OutNodes(v->right, r+1, c + (offset >> r));

}

}

**void** passWidth(binTree b){

**int** n = 0; **int** i = -1;

**char** res[80];

Queue<binTree> q;

binTree p;

q.pushTail( b);

cout << "\nНачало алгоритма обхода в ширину!";

**while**(!q.empty()){

p = q.front();

cout << "\nВнесли в очередь: " << p->data;

q.pop(); res[++i] = p->data;

n++;

**if** (!isNull(Left(p)))

q.push(Left(p));

**else**

cout << "\n Левого сына нет.";

**if** (!isNull(Right(p)))

q.push(Right(p));

**else**

cout << "\n Правого сына нет.";

}

cout << "\nОбход в ширину:";

**for** (**int** j = 0; j<= i; j++)

cout << " " << res[j];

cout << "\nПройдено узлов: "<< n << "\n";

}

binTree enterBT (istream& stream){

**char** ch;

binTree p, q;

stream >> ch;

**if** (ch=='/')

**return** NULL;

**else** {

p = enterBT(stream);

q = enterBT(stream);

**return** ConsBT(ch, p, q);

}

}

**int** hBT(binTree b)

{

**if** (isNull(b)) **return** 0;

**else** **return** max (hBT (Left(b)), hBT(Right(b))) + 1;

}

**void** readElem(istream& stream,**char** prev, binTree&);

**void** readRight(istream &stream, binTree& obj, **char** prev);

**void** readLeft(istream &stream, binTree& obj, **char** prev);

**void** readRoot(istream &stream, binTree& obj, **char** prev);

**void** readTree(istream &stream, binTree& obj){

**char** x;

**do**

stream >> x;

**while** (x == ' ');

readElem(stream , x, obj);

}

**void** readElem(istream &stream,**char** prev, binTree& obj){ //prev - ранее прочитанный символ

**if** (prev == ')'){

cerr << "\033[1;31mОшибка!\033[0m Символ \033[1;34m\')\'\033[0m встречен раньше \033[1;34m\'(\'\033[0m " << endl; exit(1); }

**else** **if** (isalpha(prev) || (prev <= '9' && prev>= '0') || prev == '#' || prev == '/'){

cout << "\nНеверный ввод!";

exit(1);

}

**else** **if** (prev == '(')

readRoot(stream, obj,prev);

}

**void** readRoot(istream &stream, binTree& obj, **char** prev){

**char** x;

binTree p1,p2;

**if** (!(stream >> x)){

cerr << "\033[1;31mОшибка!\033[0m Не найден конец списка." << endl; exit(1);

}

**while**(x == ' ')

stream >> x;

**if** (x == ')'){

cerr << "\033[1;31mОшибка!\033[0m Символ \033[1;34m\')\'\033[0m встречен раньше \033[1;34m\'(\'\033[0m " << endl; exit(1);

}

**else** **if** (isalpha(x)) {

readLeft(stream,p1,prev);

readRight(stream,p2,prev);

obj = ConsBT(x, p1 , p2);

stream >> x;

**if** (x != ')')

//ungetc(x, stream);

stream.unget();

}

}

**void** readLeft(istream &stream, binTree& obj, **char** prev){

**char** x;

**if** (!(stream >> x)){

cerr << "\033[1;31mОшибка!\033[0m Не найден конец списка." << endl; exit(1);

}

**while**(x == ' ')

stream >> x;

**if** (x == '('){

readRoot(stream, obj, prev);

}

**else** **if** (x == '/' || x == '#')

obj = NULL;

**else** {

cout << "\nНеправильный ввод";

exit(1);

}

}

**void** readRight(istream &stream, binTree& obj, **char** prev){

**char** x,y;

binTree p1,p2;

**if** (!(stream >> x)){

cerr << "\033[1;31mОшибка!\033[0m Не найден конец списка." << endl; exit(1);

}

**while**(x == ' ')

stream >> x;

**if** (x == '('){

readRoot(stream, obj, prev);

}

**else** **if** (x == '/' || x == '#'){

stream >> y;

**while** (y == ' ')

stream >> y;

**if** (y == ')')

obj = NULL;

**else**{

cout << "\nОшибка ввода";

exit(1);

}

}

}

**void** fixed(){

cin.clear();

**while** (cin.get() != '\n');

}

**void** toLower(string &str){

**int** i = 0;

**while** (str[i]){

str[i] = tolower(str[i]);

i++;

}

}

**int** main(){

string str;

**do**{

binTree res;

cout << "Считать из файла - \033[1;34m\'f\' , \033[0m или с консоли - \033[1;34m\'c\': \033[0m";

cin >> str; toLower(str);

**while** (str!=file && str!=con)

{

cout << "\033[1;31mОшибка ввода, попробуйте снова: \033[0m" ;

cin >> str;

}

**if** (str == file)

{

string fileName;

cout << "\033[0;32m ведите имя файла:\033[0m ";

cin >> fileName;

ifstream in(fileName.c\_str(),ios::in | ios::binary);

**while**(!in)

{

cout <<"\033[1;31mНе удается открыть файл. Введите корректное имя: \033[0m";

cin >> fileName;

in.open(fileName.c\_str(),ios::in | ios::binary);

}

//res = enterBT(in);

readTree(in, res);

in.close();

}

**else** {

cout << "\033[0;32m введите узлы БД в формате КЛП (пустой узел - \'/\') :\033[0m ";

//res = enterBT(cin);

readTree(cin, res);

}

Tree tree(hBT(res),res);

cout << "\nСчитанное дерево:\n";

tree.OutTree();

passWidth(res);

cout << "\n" << " вы хотите продолжить \033[1;34m[y/n]\033[0m ? : ";

cin >> str;

toLower(str);

**while** (str!=stop && str!=again){

cout << "\n\033[1;31mОшибка, введите [y/n] :\033[0m";

cin >> str; toLower(str);

}

destroy(res);

}**while** (str != "n");

**return** 0;

}

**Приложение (файл Queue.h)**

**using** **namespace** std;

**template** <**typename** Qtype> **class** node1 { // структура элемента очереди

**public**:

Qtype data;

node1<Qtype>\* next;

node1<Qtype>\* prev;

node1() { next = prev = NULL;}

};

//typedef node\* pNode;

**template** <**typename** Qtype> **class** Queue{

node1<Qtype> \*head;

node1<Qtype> \*tail;

**public**:

Queue() { head = tail = NULL;}

**void** pushTail(Qtype x);

Qtype popHead();

**void** printQueue();

**bool** empty();

};

**template** <**typename** Qtype>

**void** Queue<Qtype>::pushTail(Qtype x){

node1<Qtype>\* newNode;

newNode = **new** node1<Qtype>; // создать новый узел

newNode->data = x; //добавить в новый узел элемент

newNode->prev = tail; //теперь в хвосте новый элемент

newNode->next = NULL;

**if** (tail != NULL) // добавить узел в конец очереди

tail->next = newNode;

tail = newNode;

**if** (!head)

head = tail; //если это первый элемент

}

**template** <**typename** Qtype>

Qtype Queue<Qtype>::popHead(){

node1<Qtype>\* top = head;

Qtype x;

**if** (top == NULL){ //очерель пуста

cout << "\nОчередь пуста!!!";

exit(1);

}

x = top->data; //запомнить первый элемент

//obj.head = top->next;

// head->next->prev = NULL;

head = head->next;

**if** (head == NULL) // если в очереди ничего нет

tail = NULL;

**delete** top;

**return** x;

}

**template** <**typename** Qtype>

**bool** Queue<Qtype>::empty(){

**return** head == NULL;

}

**template** <**typename** Qtype>

**void** Queue<Qtype>::printQueue(){

node1<Qtype>\* tmp = **this**->tail;

cout << "\n ";

**if** (tmp == NULL){

cout << "PESTO"; **return**;

}

**while** (tmp != NULL){

cout << tmp->data << " ";

tmp = tmp->prev;

}

}