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. 6383 |  | Спас А.А. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург 2017

## Задание

**Вариант 4:**

Задано бинарное дерево *b* типа *ВТ* 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.

## 2.Пояснение задания

Входные данные: бинарное дерево *b* типа *ВТ* с произвольным типом элементов, бинарное дерево задается из файла или с клавиатуры.

Выходные данные: 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 выводятся в файл, на консоль выводятся пояснения к работе алгоритма.

**3. Основные теоретические сведения**

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

Примером может служить структура на рисунке 1.

*a*

*c*

*b*

*f*

*d*

*h*

*e*

*j*

*g*

*i*

*k*

*l*

Рис. 1 Пример бинарного дерева

Скобочное представление бинарного дерева (БД):

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

< пусто > ::= Λ,

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

Скобочное представление дерева на рисунке 1 выглядит следующим образом:

(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))))

Применив правила упрощения скобочной записи:

1.(<корень> <непустое БД> Λ) ≡ (<корень> <непустое БД>),

2. (< корень > Λ Λ) ≡ (< корень >),

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

## 4.Описание алгоритма

В функцию goriz передаем индекс дерева. В очередь передаем этот индекс, пока очередь не пуста будем циклически выполнять следующий алгоритм:

1.Берем из очереди индекс узла дерева,

1. С помощью этого индекса выводим на экран или в файл корень,
2. Если левое поддерево существует, то заносим его индекс в очередь,
3. Если правое поддерево существует, то заносим его индекс в очередь.

## 5.Сигнатура

bool isNull(binTree, int); //проверка на нулевой узел base RootBT (int, binTree); //взятие корня поддерева int Left (int, binTree); //взятие индекса правого узла int Right (int, binTree); //взятие индекса левого узла

int ConsBT(base x, int lst, int rst, binTree, int); //добавление узла в список на базе вектора void destroy (int, binTree); // удаление дерева

int enterBT (int, binTree, ifstream &); // считывание данных из файла int readBT(int, binTree); // считывание данных с клавиатуры

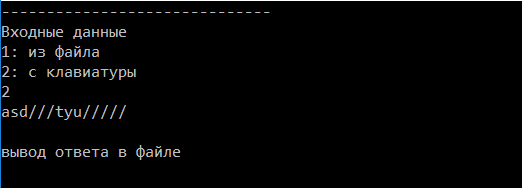
void goriz(int index,binTree b, ofstream &); // основная функция, которая выполняет алгоритм обхода в ширину

int lastIndex; //последний индекс массива

int lt; //индекс левого узла int rt; //индекс правого узла

## 5.Тестирование

* 1. **Ввод с консоли**

****

Выходные данные в файле “output.txt”



## Ввод из файла

Входные данные в файле “input.txt”



Выходные данные в файле “output.txt”



# Тестирование программы

(задан набор вещественнызх элементов и значения a и b )

|  |  |
| --- | --- |
| Исходные данные | Результат |
| fghj///k/// | fghkj |
| abgm///sd//r//hf//k// | abhgsfkmdr |
| fg/// | fg |
| rty//i//// | rtyi |
| oiuy//i///k//// | oikuyi |
| f// | f |

**Вывод:** Разработана программа , которая выполняет обход дерева в ширину. В результате выполнения работы были получены знания о структуре программ языка С++ , классах, очереди, изучены синтаксис и типы данных.

## Приложение А. Код программы

Файл Queue.h

#include <iostream>

#include <cstdlib>

using namespace std;

template <typename T>

struct Nod //Структура для очереди

{

T znach;

Nod \*Next;

};

template <typename T>

class Queue //Очередь

{

Nod<T> \*First, \*Last;

int kol;

public:

Queue() :First(NULL), Last(NULL) { kol = 0; }; //конструкор

~Queue(); //деструктор

void Put(T x);//добавление нового элемента в конец очереди

T Get();//возвращение значения первого элемента и его удаление

int Kol();//возвращает количество элементов в очереди

bool Empty();//возвращает True если очередь пуста

};

template <typename T>

void Queue<T>::Put(T x) //добавление нового элемента в конец очереди

{

Nod<T> \*temp = new Nod<T>;//создание указателя на структуру Nod и выделение под него памяти

temp->znach = x;

temp->Next = NULL;

if (First == NULL)

{

First = temp;

Last = temp;

kol++;

}

else

{

Last->Next = temp;

Last = temp;

kol++;

}

};

template <typename T>

Queue<T>::~Queue() //удаление всех эелементов

{

Nod<T> \*temp;

kol = 0;

while (First != NULL)

{

Nod<T> \*temp;

temp = First;

First = First->Next; //переходим на следующий элемент

delete temp;

}

};

template <typename T>

T Queue<T>::Get() //возвращение значения первого элемента и его удаление

{

if (First == NULL) { printf("Очередь пуста \n"); exit(1); }

Nod<T> \*temp;

T x;

x = First->znach;

temp = First;

First = First->Next;

delete temp;

kol--;

return x;

};

template <typename T>

int Queue<T>::Kol() //возвращает количество элементов в очереди

{

return kol;

};

template <typename T>

bool Queue<T>::Empty() //возвращает True если очередь пуста

{

return First == NULL;

};

Файл Btree.h

#include <iostream>

#include <fstream>

using namespace std;

template <typename T>

struct Node //Структура дерева

{

T info = NULL;

int lt; //индекс левого узла

int rt; //индекс правого узла

};

typedef Node<char> binTree[100];

bool isNull(binTree, int); //проверка на нулевой узел

char RootBT(int, binTree); //взятие корня поддерева

int Left(int, binTree); //взятие индекса левого узла

int Right(int, binTree); //взятие индекса правого узла

int ConsBT(char x, int lst, int rst, binTree, int); //добавление узла в список на базе вектора

void destroy(int, binTree); // удаление дерева

int enterBT(int, binTree, istream &input); // считывание данных из файла

int lastIndex = 0; //последний индекс массива

bool isNull(binTree b, int index) //проверка на нулевой узел

{

return b[index].info == NULL;

}

char RootBT(int index, binTree b) //взятие корня поддерева

{

if (b == NULL)

{

cerr << "Error: RootBT(null) \n";

exit(1);

}

else

return b[index].info;

}

int Left(int index, binTree b) //взятие индекса левого узла

{

if (b == NULL)

{

cerr << "Error: Left(null) \n";

exit(1);

}

else

return b[index].lt;

}

int Right(int index, binTree b) //взятие индекса правого узла

{

if (b == NULL)

{

cerr << "Error: Right(null) \n";

exit(1);

}

else

return b[index].rt;

}

int ConsBT(char x, int lst, int rst, binTree b, int index) //добавление узла в список на базе вектора

{

if (b != NULL)

{

b[index].info = x;

b[index].lt = lst;

b[index].rt = rst;

if (lastIndex < rst)

lastIndex = rst;

return index;

}

else

{

cerr << "Memory not enough\n";

exit(1);

}

}

void destroy(int index, binTree b) // удаление дерева

{

if (b != NULL)

{

destroy(b[index].lt, b);

destroy(b[index].rt, b);

delete b;

b = NULL;

}

}

int enterBT(int index, binTree b, istream &input) // считывание данных из файла

{

char ch;

int p, q;

input >> ch;

lastIndex = index;

if (ch == '/')

{

b[index].info = NULL;

return index;

}

else

{

p = enterBT(++index, b, input);

index = lastIndex;

q = enterBT(++index, b, input);

index = q;

return ConsBT(ch, p, q, b, index - (q - p + 1));

}

}

Файл main.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdlib.h>

#include <fstream>

#include <cstdlib>

#include <Btree.h>

#include <Queue.h>

using namespace std;

void goriz(int index, binTree b, ofstream &);// основная функция

int main() {

setlocale(LC\_ALL, "russian");

binTree b;

int input;

ifstream fin("KLP.txt");

ofstream fout("output.txt");

cout << "------------------------------\n";

cout << "Входные данные \n";

cout << "1: из файла \n2: с клавиатуры \n";

cin >> input;

switch (input)

{

case 1:

enterBT(0, b, fin);

goriz(0, b, fout);

cout << endl << "вывод ответа в файле";

break;

case 2:

enterBT(0, b, cin);

goriz(0, b, fout);

cout << endl << "вывод ответа в файле";

break;

default:

cout << "Incorrect command ";

break;

}

fin.close(); fout.close();

cout << endl;

system("pause > nul");

return 0;

}

void goriz(int index, binTree b, ofstream &fout)

{

Queue<int> q; // Создаем очередь с элементами типа int

q.Put(index);//заношу в очередь индекс корневого элемента дерева

while (!q.Empty())// пока очередь не пуста

{

index = q.Get();//Выдераем из очереди индекс элемента дерева

fout << RootBT(index, b) << " ";

if (!isNull(b, Left(index, b))) //если слева есть элемент то заносим его индекс в очередь

{

q.Put(Left(index, b));

}

if (!isNull(b, Right(index, b)))//если справа есть элемент то заносим его индекс в очередь

{

q.Put(Right(index, b));

}

}

}