**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: Бинарные деревья

|  |  |  |
| --- | --- | --- |
| Студент гр. 6383 |  | Любчук Д.В. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

**Цель работы.**

2 а) определить максимальную глубину дерева b, т. е. число ветвей в самом длинном из путей от корня дерева до листьев;

**Содержательная постановка задачи.**

На входе программа получает последовательность символов из файла “input.txt”. Для решения поставленных задач используются рекурсивные процедуры.

Бинарное дерево - конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом. Так определенное бинарное дерево не является частным случаем дерева.

Определим скобочное представление бинарного дерева (БД):

* < БД > ::= < пусто > | < непустое БД >,
* < пусто > ::= /,
* < непустое БД > ::= ( < корень > < БД > < БД > ).
* Здесь пустое дерево имеет специальное обозначение - /.

Исходные данные представляют собой строку, содержащую элементы бинарного дерева. Результат работы программы представляет собой числа – значения требуемых выражений.

**Спецификация программы**

**Место и форма представления входных данных**: файл

**Входные данные** : строка

**Выходные данные:** На экран выводится информация о заданном бинарном дереве

**Описание функций**

|  |  |
| --- | --- |
| Имя функции | Назначение |
| binTree enterBT(); | Считывает бинарное дерево из файла . |
| int getMaxDepth(binTree q, int depth) | Считает глубину дерева .Выводит текущую глубину на экран.  На вход подается узел бинарного дерева |
| void displayBT(binTree b, int n) | Функция вывода бинарного дерева в консоль. |

Тестирование

|  |  |  |
| --- | --- | --- |
| № опыта | Входные данные | Исходные данные |
| 1 | (a//) | Высота БД 1 |
| 2 | (a(b//)(c//)) | Высота БД 2 |
| 3 | (a(b(c//)/)(d/(e//))) | Высота БД 3 |
| 4 | (a(b(c(d//)(e//))/)(f(/(g//))) | Высота БД 4 |
| 5 | (/) | **Пустое БД** |

Код программы

**Исходный код.cpp**

#include <iostream>

#include <fstream>

#include <fstream>

#include <cstdlib>

#include "Btree.h"

#include <windows.h>

#include <conio.h>

#include <algorithm>

using namespace std;

using namespace binTree\_modul;

typedef unsigned int unInt;

binTree enterBT();

void displayBT(binTree b, int n);

int getMaxDepth(binTree b, int depth = 1);

ifstream infile("input.txt");

int main()

{

binTree b;

SetConsoleCP(1251); // для вывода кирилицы

SetConsoleOutputCP(1251); // для вывода кирилицы

b = enterBT();

int depth = 1;

if (isNull(b)) cout << "Пустое БД" << endl;

else

{

cout << "Бинарное дерево (повернутое): " << endl;

displayBT(b, 1);

cout << "\nВычисление высоты дерева: " << endl;

int a = getMaxDepth(b) - 1;

cout << "Высота дерева = " << a << "\n\n";

destroy(b);

}

\_getch();

}

//---------------------------------------

binTree enterBT()

{

char ch;

binTree p, q;

infile >> ch;//считывание из файла

while (ch == '(' || ch == ')') infile >> ch;//проверка на скобки

if (ch == '/') { return NULL; }//проверка на пустой элемент

else { //иначе рекурсивная запись элементов

p = enterBT();

q = enterBT();

return ConsBT(ch, p, q);

}

}

int getMaxDepth(binTree q, int depth)//находит глубину дерева

{

if (q != NULL)//если узел не пуст

cout << "Узел " << q->info << " Текущая глубина=" << depth << endl; //Печать узла и текущей глубины

if (q == NULL) { return depth; }//Если элемент пуст, возвращается значение глубины

else return max(getMaxDepth(q->rt, depth + 1), getMaxDepth(q->lt, depth + 1));//В противном случае, рекурсивно продолжается процесс

}

//---------------------------------------

void displayBT(binTree b, int n) //Для печати повернутого дерева

{ // n - уровень узла

if (b != NULL)

{

cout << ' ' << RootBT(b); //Проверка элемента

if (!isNull(Right(b)))

displayBT(Right(b), n + 1); //Если правая ветка не пустая, то печатаем элементы

else cout << endl; // спускаемся вниз

if (!isNull(Left(b))) //Если левая ветка не пустая, то печатаем элементы

{

for (int i = 1; i <= n; i++)

cout << " "; // вправо

displayBT(Left(b), n + 1);

}

}

}

**Btree.h**

namespace binTree\_modul

{

typedef char base;

struct node {

base info;

node \*lt;

node \*rt;

// constructor

node() { lt = NULL; rt = NULL; }

};

typedef node \*binTree; // "представитель" бинарного дерева

binTree Create(void);

bool isNull(binTree);

bool RooBT(binTree);

base RootBT(binTree); // для непустого бин.дерева

binTree Left(binTree);// для непустого бин.дерева

binTree Right(binTree);// для непустого бин.дерева

binTree ConsBT(const base &x, binTree &lst, binTree &rst);

void destroy(binTree&);

} // end of namespace binTree\_modul

/\*

1) RootBT: NonNullBT -> ?;

2) Left: NonNullBT -> BT;

3) Right: NonNullBT -> BT;

4) ConsBT: ? \* BT \* BT -> NonNullBT;

5) Null: BT -> Boolean;

6) Create: -> BT

\*/

**bt\_func.cpp**

// Implementation - Реализация АТД "Бинарное дерево" (в процедурно-модульной парадигме)

#include <iostream>

#include <cstdlib>

#include "Btree.h"

using namespace std;

namespace binTree\_modul

{

binTree Create() { return NULL; }

bool isNull(binTree b) { return (b == NULL); } //Проверяет не лустое ли дерево

base RootBT(binTree b) // для непустого бин.дерева

{

if (b == NULL) { cerr << "Error: RootBT(null) \n"; exit(1); } //Если пусто

else return b->info; //Если нет, возврат данных элемента

}

bool RooBT(binTree b)//Проверка элемента

{

if (b == NULL) return false; //если пустой

else return true; //Если не пуст

}

binTree Left(binTree b) // для непустого бин.дерева

{

if (b == NULL) { cerr << "Error: Left(null) \n"; exit(1); }

else return b->lt;

}

binTree Right(binTree b) // для непустого бин.дерева

{

if (b == NULL) { cerr << "Error: Right(null) \n"; exit(1); }

else return b->rt;

}

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

{

binTree p;

p = new node;

if (p != NULL)

{

p->info = x;

p->lt = lst;

p->rt = rst;

return p;

}

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

}

void destroy(binTree &b)

{

if (b != NULL)

{

destroy(b->lt);

destroy(b->rt);

delete b;

b = NULL;

}

}

}