**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по Лабораторной работе №4  
По курсу алгоритмы и структуры данных**

**Тема: «Бинарные деревья»**

|  |  |  |  |
| --- | --- | --- | --- |
| Студентка гр. 6383 |  | Гомонова А.А. |  |
| Преподаватель |  | Шолохова О. М. |  |

Санкт-Петербург

2017г.

**Цель работы**

Используя бинарное дерево, написать программу, реализующую поставленную задачу.

**Задание**

Для заданного бинарного дерева *b* типа *BT* с произвольным типом элементов:

напечатать элементы из всех листьев дерева *b*;

подсчитать число узлов на заданном уровне *n* дерева *b* (корень считать узлом 1-го уровня);

**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.

**Алгоритм работы программы**

1. На вход в программу поступает бинарное дерево, происходит проверка на пустоту;
2. Определение и вывод листьев данного дерева;
3. Ввод уровня, на котором ищем количество узлов.

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

**Тестирование**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исходная последовательность | Найденные листья | Уровень | Количество узлов на данном уровне | Сообщение об ошибке |
| (a(b(c//)(d//))(c/(o/(e(w//)(r//)))) | c d w r | 3 | 3 | - |
| (b//) | b | 3 | - | Такого уровня не существует |
| (b(a(с//)(d//))(q(w//)(r(j//)(p//)))) | c d w j p | 2 | 2 | - |
| (a(o//)(i//)) | o i | 1 | 1 | - |
| (a(o(y//)(t//))(i(f//)(g//)) | y t f g | 3 | 4 | - |
| (//) | Пустое БД | | | |

**Вывод**

В ходе выполнения лабораторной работы по программированию алгоритмов с бинарными деревьями мы познакомились с такой нелинейной структурой данных, как бинарное дерево. Так же мы освоили обработку дерева: вывод листьев и узлов.

**ПРИЛОЖЕНИЕ**

**Текст программы**

**main.cpp**

#include <iostream>

#include <fstream>

#include <fstream>

#include <cstdlib>

#include <windows.h>

#include <conio.h>

#include <algorithm>

#include <locale.h>

using namespace std;

typedef unsigned int unInt;

typedef char base;

struct node {

base info;

node \*lt;

node \*rt;

// constructor

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

};

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

binTree enterBT();

void displayBT(binTree b, int n);

unInt NumOfLvl(binTree b, int a, int c = 1);

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

void levelBT(binTree b);

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

binTree Create(void);

bool isNull(binTree), RooBT(binTree);

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

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

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

void destroy(binTree&);

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);

int a = getMaxDepth(b) - 1; // высота дерева

cout << "\nПоиск листьев дерева:" << endl; levelBT(b);

cout << "\nНахождение количества узлов на N-ом уровне. Введите N:" << endl;

int num;

cin >> num;

while ((num < 1) || (num > a))

{

cout << "Такого уровня не существует\n";

cout << "Введите N: \n";

cin >> num;

}

cout << "Количество узлов на заданном уровне = " << NumOfLvl(b, num);

destroy(b);

}

\_getch();

}

binTree enterBT() //функция ввода бинарного дерева

{

char ch;

binTree p, q;

infile >> ch;

while (ch == '(' || ch == ')') infile >> ch;

if (ch == '/') { return NULL; }

else {

p = enterBT(); q = enterBT();

return ConsBT(ch, p, q);

}

}

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

{

if (q == NULL) { return depth; }

else return max(getMaxDepth(q->rt, depth + 1), getMaxDepth(q->lt, depth + 1));

}

void displayBT(binTree b, int n) // функция для вывода бд на консоль

{

if (b != NULL) {

displayBT(Right(b), n + 1);//Прохождение по правым

for (int i = n + 1; i > 0; i--)

cout << " ";

cout << b->info << endl;//Вывод текущего

displayBT(Left(b), n + 1);//Прохождение по левым

}

}

void levelBT(binTree b) // определение является ли листом

{

if (!isNull(b))

{

if (!(isNull(b->lt) && isNull(b->rt)))

cout << "Узел:" << b->info << endl;

levelBT(b->lt);

if (isNull(b->lt) && isNull(b->rt))

cout << "Лист:" << b->info << endl;

levelBT(b->rt);

}

}

unInt NumOfLvl(binTree b, int lvl, int lvl\_now) //функция, определяющая наличие узлов на заданном уровне

{

if (b != NULL&&lvl >= lvl\_now)cout << lvl << " vs " << lvl\_now << " Узел - " << b->info << "\n";

{

if (lvl == lvl\_now)

{

cout << " Узел на заданном уровне существует!\n";

}

else cout << " Это узел не того уровня\n";

}

if (lvl == lvl\_now) return 1;

else if (lvl >= lvl\_now)return(((b->lt) ? NumOfLvl(b->lt, lvl, lvl\_now + 1) : 0) + ((b->rt) ? NumOfLvl(b->rt, lvl, lvl\_now + 1) : 0));

}

binTree Create()

{

return NULL;

}

bool isNull(binTree b)

{

return (b == NULL);

}

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

{

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

else return b->info;

}

bool RooBT(binTree b)

{

if (b == NULL) return false;

else return true;

}

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;

}

}