**Отчет по лабораторной работе № 23** по курсу “ Практикум на ЭВМ ”

Студент группы М8О-102Б-21, Яценко Александр Владимирович, № по списку 20

Контакты: e-mail - alexander.iatsenko@gmail.com

Работа выполнена: « » \_\_\_\_\_\_\_\_\_\_\_\_201\_\_\_г.

Преподаватель: Никулин Сергей Петрович Каф.806\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Входной контроль знаний с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отчет сдан « » \_\_\_\_\_\_\_\_\_201 \_\_ г., итоговая оценка \_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**1. Тема:** динамические структуры данных, обработка деревьев

**2. Цель работы:** составить программу на языке си для построения и обработки двоичного дерева, содержащего узлы типа char

**3. Задание (**задача 25**):** определить глубину двоичного дерева

**4. Оборудование:**

*Оборудование ПЭВМ студента:*

Процессор: AMD Ryzen 5 4600H, с ОП 6 Мб (виртуальная машина), НМД 25600 Мб. Монитор: Huawei 16,1 IPS 1920×1080, 137 PP.

**5. Программное обеспечение ЭВМ студента, если использовалось:**

Операционная система семейства Linux, наименование Fedora, версия 5.13.12-200. Fc24.x86\_64 GNU/Linux

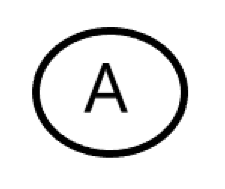
Прикладные системы и программы: CLion

**6. Идея, метод, алгоритм** решения задачи(в формах:словесной,псевдокода,графической[блок-схема,диаграмма,рисунок,таблица] или формальные спецификации с пред- и постусловиями)

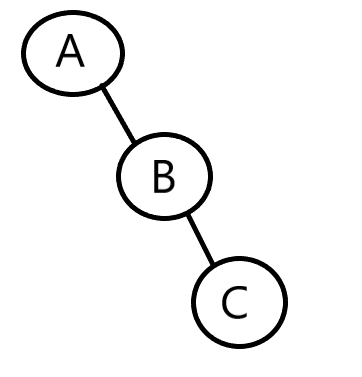
* Реализовать структуру для двоичного дерева
* Реализовать различные функции для работы с деревом:  
  добавление вершины, удаление, печать дерева
* Реализовать функцию поиска глубины дерева

Тесты:

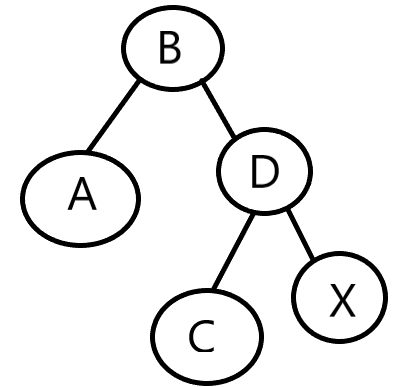
1. Глубина – 0



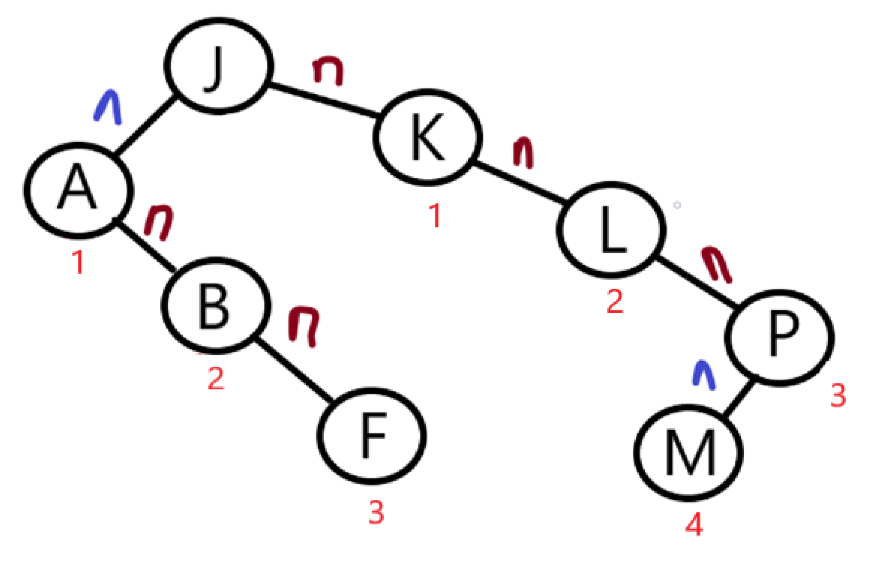
1. Глубина – 2



1. Глубина – 2



1. Глубина - 4



**7. Сценарий выполнения работы** [план работы,первоначальный текст программы в черновике(можно на отдельном листе)итесты либо соображения по тестированию]

Код программы:

#include <stdio.h>

#include <malloc.h>

typedef struct Node Node;

struct Node {

char value;

Node\* leftChild;

Node\* rightChild;

};

Node\* makeNode(double val) {

Node\* node = (Node\*)malloc(sizeof(Node));

node->value = val;

node->leftChild = node->rightChild = NULL;

return node;

}

void insert(Node\* current, Node\* val) {

if (val->value < current->value) {

if (current->leftChild == NULL) {

current->leftChild = val;

return;

}

insert(current->leftChild, val);

} else {

if (current->rightChild == NULL) {

current->rightChild = val;

return;

}

insert(current->rightChild, val);

}

}

void print(Node\* current, int d) {

if (current->rightChild != NULL) {

print(current->rightChild, d + 1);

}

printf("%\*s%c\n", 4 \* d, " ", current->value);

if (current->leftChild != NULL) {

print(current->leftChild, d + 1);

}

}

void destroy(Node\* leaf) {

if (leaf != NULL) {

destroy(leaf->leftChild);

destroy(leaf->rightChild);

free(leaf);

}

}

Node\* deleteNode(Node\* root, double val) {

if (root == NULL)

return root;

if (root->value > val) {

root->leftChild = deleteNode(root->leftChild, val);

return root;

}

if (root->value < val) {

root->rightChild = deleteNode(root->rightChild, val);

return root;

}

if (root->leftChild == NULL) {

Node\* temp = root->rightChild;

free(root);

return temp;

}

if (root->rightChild == NULL) {

Node\* temp = root->leftChild;

free(root);

return temp;

}

Node\* tmpParent = root;

Node\* tmp = root->rightChild;

while (tmp->leftChild != NULL) {

tmpParent = tmp;

tmp = tmp->leftChild;

}

if (tmpParent != root)

tmpParent->leftChild = tmp->rightChild;

else

tmpParent->rightChild = tmp->rightChild;

root->value = tmp->value;

free(tmp);

return root;

}

int depth(Node\* tree) {

int val = 0;

if (tree != NULL) {

int lDepth = depth(tree->leftChild);

int rDepth = depth(tree->rightChild);

val = lDepth + 1 > rDepth + 1 ? lDepth + 1 : rDepth + 1;

}

return val - 1;

}

int main() {

Node\* root = NULL;

int comm;

printf("Commands:\n\t1. Insert\n\t2. Delete\n\t3. Print\n\t4. Depth\n\t5. Break\n");

do {

printf("Enter command: ");

if (scanf("%i", &comm) != 1)

getchar();

char val;

switch (comm) {

case 1:

printf("Enter a char: ");

scanf(" %c", &val);

if (root == NULL) {

root = makeNode(val);

} else {

insert(root, makeNode(val));

}

break;

case 2:

printf("Enter a char`s value: ");

scanf(" %c", &val);

root = deleteNode(root, val);

break;

case 3:

if (root != NULL) {

printf("\nTree:\n");

print(root, 0);

printf("\n");

} else {

printf("Empty tree!\n");

}

break;

case 4:

printf("Depth: %d\n", depth(root));

break;

default:

break;

}

} while (comm != 5);

destroy(root);

return 0;

}

*Пункты 1-7 отчета составляются строго до начала лабораторной работы.*

*Допущен к выполнению работы.* **Подпись преподавателя****\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **Распечатка протокола** (подклеить листинг окончательного варианта программы с тестовыми примерами,подписанныйпреподавателем)

Commands:

1. Insert

2. Delete

3. Print

4. Depth

5. Break

Enter command: 1

Enter a char: a

Enter command: 3

Tree:

a

Enter command: 4

Depth: 0

Enter command: 2

Enter a char`s value: a

Enter command: 3

Empty tree!

Enter command: 1

Enter a char: a

Enter command: 1

Enter a char: b

Enter command: 1

Enter a char: c

Enter command: 3

Tree:

c

b

a

Enter command: 4

Depth: 2

Enter command: 2

Enter a char`s value: b

Enter command: 3

Tree:

c

a

Enter command: 4

Depth: 1

Enter command: 2

Enter a char`s value: a

Enter command: 2

Enter a char`s value: c

Enter command: 4

Depth: 0

Enter command: 1

Enter a char: b

Enter command: 1

Enter a char: a

Enter command: 1

Enter a char: d

Enter command: 1

Enter a char: c

Enter command: 1

Enter a char: x

Enter command: 3

Tree:

x

d

c

b

a

Enter command: 4

Depth: 2

Enter command: 2

Enter a char`s value: b

Enter command: 3

Tree:

x

d

c

a

Enter command: 2

Enter a char`s value: a

Enter command: 2

Enter a char`s value: d

Enter command: 2

Enter a char`s value: c

Enter command: 2

Enter a char`s value: x

Enter command: 3

Empty tree!

Enter command: 1

Enter a char: j

Enter command: 1

Enter a char: k

Enter command: 1

Enter a char: l

Enter command: 1

Enter a char: p

Enter command: 1

Enter a char: a

Enter command: 1

Enter a char: b

Enter command: 1

Enter a char: m

Enter command: 1

Enter a char: f

Enter command: 3

Tree:

p

m

l

k

j

f

b

a

Enter command: 4

Depth: 4

Enter command: 5

Process finished with exit code 0

1. **Дневник отладки** должен содержать дату и время сеансов отладки, и основные события(ошибки в сценарии и программе,нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  | или |  |  |  |  |  |
|  | дом. |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. **Замечания автора** по существу работы: -
2. **Выводы:** работа выполнена

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_