**Отчёт по лабораторной работе №23**

по курсу «Языки и методы программирования».

Выполнил студент группы **М8О-114БВ-24**: **Дылдин Сергей Владиславович** № по списку **10**.

Контакты: [**dsergey010206@gmail.com**](mailto:dsergey010206@gmail.com)

Работа выполнена: «9» мая 2025 г.

Преподаватель: **каф. 806 Никулин Сергей Петрович**

Входной контроль знаний с оценкой: **\_\_\_\_\_**

Отчет сдан «10» мая 2025 г.

Итоговая оценка: \_\_\_\_\_

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

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

2. **Цель работы:** составить программу на языке Си для построения и обработки дерева общего вида, содрежащего узлы типа int.

3. **Задание:** 26— определить степень дерева.

4. **Оборудование**: *Оборудование ПЭВМ студента, если использовалось:*

Процессор **Intel core i5 12400H**, ОП **16** ГБ, SSD **1**TB.

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

Операционная система семейства **Ubuntu**, наименование версия **VirtualBox Ubuntu 20.04.3 LTS**,

интерпретатор команд **bash** версия **5.0.17**. Система программирования **C**.Редактор текстов **VI** версия **8.1**

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

*Составление программы на Си, выполняющей следующие задачи:*

*(Под деревом подразумевается - корневой узел)  
 1. Добавление нового узла: программе передается значения узла - отца и значение нового узла, программа находит родительский узел и записывает новый узел в структуру, делаю его самым младшим листом. Задействованные функции: createnode, addchild, search.*

*2. Текстовая визуализация дерева: значение каждого узла выводится в строку(повернутый формат вывода на 90 градусов влево), функция printtree получает на вход корневой узел и с помощью рекурсий начинает вывод с самых младших узлов дерева.*

*3. Удаление узла: в дереве общего вида удаляется вся ветка, следующая за переданным узлом. Задейстованные функции: deletetree(дерево, значение вершины)->search(дерево, значение узла)->findparent(дерево, переданный узел) -> freesubtree(узел для удаления) : такая последовательность находит всех потомков переданного узла и поочередно их удаляет, освобождая память*

*4. Нахождение степени дерева - максимальная степень вершин - максимальное кол-во потомков узла: degree(дерево) рекурсивно проходится по всем потомкам, начиная с потомков корня, сравнивая кол-во их потомков с максимальным.*

*5. выход из программы, при этом освобождается память дерева.*

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

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct treenode {

int value;

int childcount;

struct treenode\* children[10];

} treenode;

treenode \*createnode(int value) {

treenode\* result = malloc(sizeof(treenode));

if (result == NULL) {

return NULL;

}

result->value = value;

result->childcount = 0;

return result;

}

void addchild(treenode \*parent, treenode \*child) {

if (parent->childcount < 10) {

parent->children[parent->childcount++] = child;

}

}

treenode \*search(treenode \*root, int value) {

if (root == NULL) return NULL;

if (root->value == value) return root;

for (int i = 0; i < root->childcount; i++) {

treenode\* found = search(root->children[i], value);

if (found != NULL) return found;

}

return NULL;

}

treenode\* findParent(treenode \*root, treenode \*target) {

if (root == NULL) return NULL;

for (int i = 0; i < root->childcount; i++) {

if (root->children[i] == target) {

return root;

}

treenode \*parent = findParent(root->children[i], target);

if (parent != NULL) {

return parent;

}

}

return NULL;

}

void freeSubtree(treenode \*node) {

if (node == NULL) return;

for (int i = 0; i < node->childcount; i++) {

freeSubtree(node->children[i]);

}

free(node);

}

void deletetree(treenode \*root, int value) {

treenode \*found = search(root, value);

if (found == NULL) return;

treenode \*parent = findParent(root, found);

if (parent != NULL) {

int index = -1;

for (int i = 0; i < parent->childcount; i++) {

if (parent->children[i] == found) {

index = i;

break;

}

}

if (index != -1) {

for (int j = index; j < parent->childcount - 1; j++) {

parent->children[j] = parent->children[j + 1];

}

parent->childcount--;

}

}

freeSubtree(found);

}

void printtree(treenode\* root, int level) {

if (root == NULL) return;

int middle = root->childcount / 2;

for (int i = root->childcount - 1; i >= middle; i--) {

if (i == root->childcount - 1) {

printf("\n");

}

printtree(root->children[i], level + 1);

}

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

for (int i = middle - 1; i >= 0; i--) {

printtree(root->children[i], level + 1);

if (i == 0) {

printf("\n");

}

}

}

int degree(treenode \*root) {

int max\_degree = root->childcount;

if (root == NULL) return 0;

for (int i = 0; i < root->childcount; i++){

int child\_degree = degree(root->children[i]);

if (child\_degree > max\_degree) max\_degree = child\_degree;

}

return max\_degree;

}

int main() {

treenode \*root = createnode(1);

treenode \*child1 = createnode(2);

treenode \*child2 = createnode(3);

treenode \*child3 = createnode(4);

treenode \*child4 = createnode(5);

treenode \*child5 = createnode(6);

treenode \*child6 = createnode(7);

treenode \*child7 = createnode(8);

treenode \*child8 = createnode(9);

treenode \*child9 = createnode(10);

treenode \*child10 = createnode(11);

treenode \*child11 = createnode(12);

treenode \*child12 = createnode(13);

treenode \*child13 = createnode(14);

treenode \*child14 = createnode(15);

addchild(root, child1);

addchild(root, child2);

addchild(root, child3);

addchild(child1, child4);

addchild(child1, child5);

addchild(child1, child6);

addchild(child2, child7);

addchild(child2, child8);

addchild(child2, child9);

addchild(child3, child10);

addchild(child3, child11);

addchild(child4, child12);

addchild(child4, child13);

addchild(child13, child14);

int choice, parent\_val, child\_val, del\_val;

while(1) {

printf("\n--- Tree Operations Menu ---\n");

printf("1. Add child to parent node\n");

printf("2. Print tree\n");

printf("3. Delete branch\n");

printf("4. tree degree\n");

printf("5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch(choice) {

case 1:

printf("Enter parent value: ");

scanf("%d", &parent\_val);

printf("Enter child value: ");

scanf("%d", &child\_val);

treenode\* parent = search(root, parent\_val);

if(parent) {

if(parent->childcount < 10) {

treenode\* new\_child = createnode(child\_val);

addchild(parent, new\_child);

printf("Child %d added to parent %d\n", child\_val, parent\_val);

} else {

printf("Parent already has maximum children!\n");

}

} else {

printf("Parent not found!\n");

}

break;

case 2:

printf("\nCurrent Tree Structure:");

printtree(root, 0);

break;

case 3:

printf("Enter value to delete: ");

scanf("%d", &del\_val);

if(del\_val == root->value) {

printf("Cannot delete root node!\n");

break;

}

if(search(root, del\_val)) {

deletetree(root, del\_val);

printf("Branch %d deleted successfully\n", del\_val);

} else {

printf("Node not found!\n");

}

break;

case 4:

printf("Maximum degree of the tree: %d\n", degree(root));

break;

case 5:

freeSubtree(root);

printf("Tree deleted. Exiting...\n");

exit(0);

default:

printf("Invalid choice! Try again.\n");

}

}

return 0;

}

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

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

8. **Распечатка протокола** *(подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем).*

PS D:\apps\labi\2sem\laba2> gcc ./treees.c -o tree

PS D:\apps\labi\2sem\laba2> ./tree

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 2

Current Tree Structure:

9

3

8

1

7

6

2

5

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 4

Maximum degree of the tree: 3

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 1

Enter parent value: 2

Enter child value: 10

Child 10 added to parent 2

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 4

Maximum degree of the tree: 4

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 3

Enter value to delete: 2

Branch 2 deleted successfully

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 2

Current Tree Structure:

9

3

8

1

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 3

Enter value to delete: 3

Branch 3 deleted successfully

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 2

Current Tree Structure:1

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 1

Enter parent value: 1

Enter child value: 2

Child 2 added to parent 1

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 2

Current Tree Structure:

2

1

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 1

Enter parent value: 1

Enter child value: 3

Child 3 added to parent 1

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 2

Current Tree Structure:

3

1

2

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 1

Enter parent value: 2

Enter child value: 4

Child 4 added to parent 2

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 2

Current Tree Structure:

3

1

4

2

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 1

Enter parent value: 2

Enter child value: 5

Child 5 added to parent 2

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 2

Current Tree Structure:

3

1

5

2

4

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 4

Maximum degree of the tree: 2

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 3

Enter value to delete: 2

Branch 2 deleted successfully

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 2

Current Tree Structure:

3

1

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 4

Maximum degree of the tree: 1

--- Tree Operations Menu ---

1. Add child to parent node

2. Print tree

3. Delete branch

4. tree degree

5. Exit

Enter your choice: 5

Tree deleted. Exiting...

PS D:\apps\labi\2sem\laba2>

9. **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. или дом.** | **Дата** | **Время** | **Событие** | **Действие по исправлению** | **Примечание** |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |

10. **Замечания автора** по существу работы: замечания отсутствуют.

11. **Выводы:** научился работать с деревьями общего вида в СП Си.

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