**ЛАБОРАТОРНАЯ РАБОТА №4**

**«ПРОГРАММИРОВАНИЕ НЕЛИНЕЙНЫХ**

**СТРУКТУР ДАННЫХ НА ЯЗЫКЕ С/С++»**

**Вариант 9**

**4.1 Цель работы**

Изучение нелинейных структур данных и приобретение навыков разработки и отладки программ, использующих древовидные структуры данных. Исследование особенностей работы с поисковыми бинарными деревьями на языке С/С++.

**4.2 Вариант задания**

Структуру данных из предыдущей лабораторной работы использовать в качестве информационного поля бинарного дерева. Написать функции:

- организации дерева,

- просмотра (обхода) дерева,

- отображения структуры дерева;

- добавления узла в дерево,

- исключения узла из дерева,

- сохранения в файл,

- загрузки данных из файла и создания по этим данным нового дерева,

- освобождения динамической памяти, занимаемой деревом (обязательно вызывать при выходе из программы),

- функцию, которая заменяет в дереве все элементы меньшие, чем некоторое положительное число А, на это число (по полю «номер поезда»).

- функцию печати информации о поездах, отправляющихся из пункта, название которого введено с клавиатуры, а если таких поездов нет, то выдать на дисплей соответствующее сообщение.

Значения и количество записей в таблице пользователь выбирает самостоятельно (т.е. количество строк таблицы не задается, память под узлы дерева выделяется динамически). Работу программы необходимо оформить в виде меню.

**4.3 Ход работы**

Были разработаны структурные схемы для функций программы на языке C (рисунки Ж.1 – Ж.17 приложение Ж).

Была разработана программа на языке программирования С, которая решала поставленную задачу (листинг 4.1).

Листинг 4.1 – Код программы на языке С

#include <stdio.h>

#include <stdlib.h>

struct time {

int hours;

int minutes;

};

struct TRAIN {

char from[20];

char to[20];

int number;

struct time time;

};

struct Node {

struct TRAIN train;

struct Node\* left;

struct Node\* right;

};

struct TRAIN setTrain();

struct time setTime();

struct Node\* addTrain(struct Node\* Node);

struct Node\* addNode(struct Node\* Node, struct Node\* newNode);

void printTrain (struct TRAIN train);

void printAllTrains(struct Node\* train);

void printTree(struct Node\* node, int otstup);

void saveTrain(struct Node\* train, FILE\* file);

void saveTree(struct Node\* train);

void freeTree(struct Node\* train);

struct Node\* makeTree(struct Node\* Node);

struct Node\* deleteNode(struct Node\* node, int number);

struct Node\* loadData(struct Node\* node);

struct Node\* loadTree(struct Node\* Node, FILE\* file);

struct Node\* changeNumbers (struct Node\* node, int number);

int printTrainsByLetter(struct Node\* node, char letter);

int main() {

struct Node\* Node = NULL;

int isActive = 1;

while (isActive) {

int choice;

printf("%s\n", "1 - Add train");

printf("%s\n", "2 - Print struct tree");

printf("%s\n", "3 - Print all trains");

printf("%s\n", "4 - Make tree");

printf("%s\n", "5 - Delete train");

printf("%s\n", "6 - Change numbers under");

printf("%s\n", "7 - Print trains with letter");

printf("%s\n", "8 - Load data");

printf("%s\n", "9 - Save data");

printf("%s\n", "10 - Free memory and exit");

scanf("%d", &choice);

switch (choice) {

case 1:

Node = addTrain(Node);

break;

case 2:

printTree(Node, 0);

break;

case 3:

printAllTrains(Node);

break;

case 4:

Node = makeTree(Node);

break;

case 5:

int number = 0;

printf("Train number: "); scanf("%d", &number);

Node = deleteNode(Node, number);

break;

case 6:

int number1 = 0;

printf("Train number: "); scanf("%d", &number1);

Node = changeNumbers(Node, number1);

break;

case 7:

printf("Letter: ");

char letter;

scanf("%s", &letter);

if (!printTrainsByLetter(Node, letter))

printf("No trains with first letter: %c\n", letter);

break;

case 8:

Node = loadData(Node);

break;

case 9:

saveTree(Node);

break;

case 10:

freeTree(Node);

exit(0);

break;

}

}

return 0;

}

struct TRAIN setTrain() {

struct TRAIN newTrain;

printf("From: "); scanf("%s", newTrain.from);

printf("To: "); scanf("%s", newTrain.to);

printf("Number: "); scanf("%d", &(newTrain.number));

newTrain.time = setTime();

return newTrain;

}

struct time setTime() {

struct time time;

printf("Hours: "); scanf("%d", &time.hours);

printf("Minutes: "); scanf("%d", &time.minutes);

return time;

}

struct Node\* addTrain(struct Node\* node) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->train = setTrain();

newNode->left=NULL;

newNode->right=NULL;

node = addNode(node, newNode);

return node;

}

struct Node\* addNode(struct Node\* node, struct Node\* newNode) {

if (!node)

node = newNode;

else if (node->train.number > newNode->train.number)

node->left = addNode(node->left, newNode);

else

node->right = addNode(node->right, newNode);

return node;

}

struct Node\* deleteNode(struct Node\* node, int number) {

if (!node)

return NULL;

if (node->train.number == number) {

struct Node\* left = node->left;

left = addNode(left, node->right);

free(node);

return left;

}

else if (node->train.number < number)

node->right = deleteNode(node->right, number);

else

node->left = deleteNode(node->left, number);

return node;

}

void printTrain (struct TRAIN train) {

printf("From: %s\n", train.from);

printf("To: %s\n", train.to);

printf("Number: %d\n", train.number);

printf("Time: %02d:%02d\n", train.time.hours, train.time.minutes);

printf("\n");

return;

}

void printAllTrains(struct Node\* Node) {

if (Node == NULL)

return;

printAllTrains(Node->left);

printTrain(Node->train);

printAllTrains(Node->right);

return;

}

void printTree(struct Node \*node, int otstup) {

if (node) {

otstup+=3;

printTree(node->right, otstup);

for(int i=0; i < otstup; i++) printf(" ");

printf("%d\n",node->train.number);

printTree(node->left,otstup);

}

}

void saveTrain(struct Node\* Node, FILE\* file) {

if (!Node)

return;

fwrite(Node, sizeof(struct Node), 1, file);

saveTrain(Node->left, file);

saveTrain(Node->right, file);

return;

}

void saveTree(struct Node\* Node) {

FILE\* file = fopen("trainData.dat", "wb");

if (!file)

printf("Open file error");

saveTrain(Node, file);

printf("%s\n", "Data saved");

fclose(file);

}

struct Node\* loadData(struct Node\* node) {

freeTree(node);

printf("\tOld tree deleted\n");

FILE\* file = fopen("trainData.dat", "rb");

if (!file)

printf("Open file error");

node = loadTree(node, file);

printf("Data loaded\n");

fclose(file);

return node;

}

struct Node\* loadTree(struct Node\* Node, FILE\* file) {

if (!feof(file)) {

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

fread(Node, sizeof(struct Node), 1, file);

if (Node->left)

Node->left = loadTree(Node->left, file);

if (Node->right)

Node->right = loadTree(Node->right, file);

}

else {

return NULL;

}

return Node;

}

void freeTree(struct Node\* Node) {

if (!Node)

return;

freeTree(Node->left);

freeTree(Node->right);

free(Node);

return;

}

struct Node\* makeTree(struct Node\* Node) {

freeTree(Node);

printf("\tOld tree deleted\n");

Node = NULL;

int choice = 0;

do

{

Node = addTrain(Node);

printf("1 - next, 0 - stop\n");

scanf("%d", &choice);

} while (choice);

return Node;

}

struct Node\* changeNumbers (struct Node\* node, int number) {

if (!node)

return NULL;

if (node->train.number < number) {

node->train.number = number;

}

node->left = changeNumbers(node->left, number);

node->right = changeNumbers(node->right, number);

return node;

}

int printTrainsByLetter(struct Node\* node, char letter) {

if (!node)

return 0;

int exist = printTrainsByLetter(node->right, letter) + printTrainsByLetter(node->left, letter);

if (node->train.from[0] == letter) {

printTrain(node->train);

exist += 1;

}

return exist;

}

Были проведены тесты программы (рисунки З.1 – З.2 приложение З).

**4.4 Вывод**

В ходе лабораторной работы были изучены нелинейные структуры данных и приобретены навыки разработки и отладки программ, использующих древовидные структуры данных. Были исследованы особенности работы с поисковыми бинарными деревьями на языке С/С++.

**Приложение Ж**

**Структурные схемы**

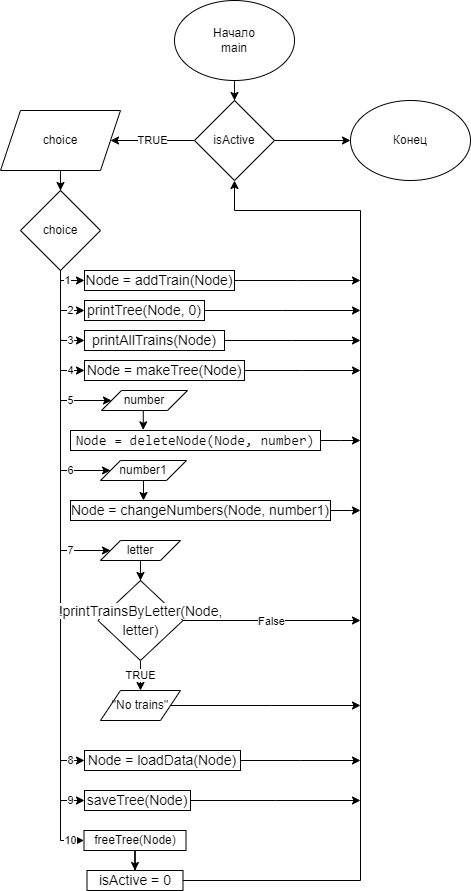


Рисунок Ж.1 – Структурная схема main

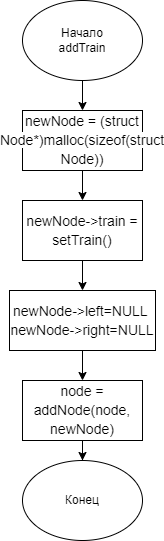


Рисунок Ж.2 – Структурная схема addTrain

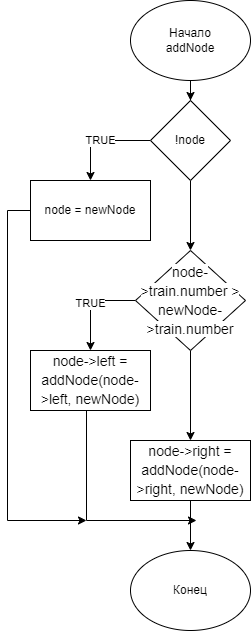


Рисунок Ж.3 – Структурная схема addNode

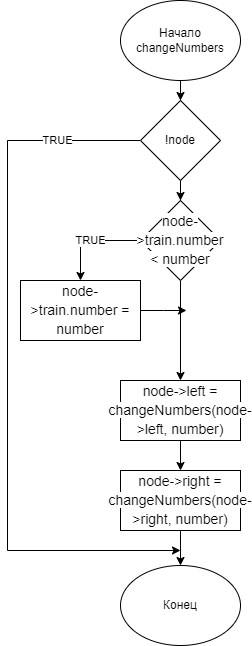


Рисунок Ж.4 – Структурная схема changeNumbers

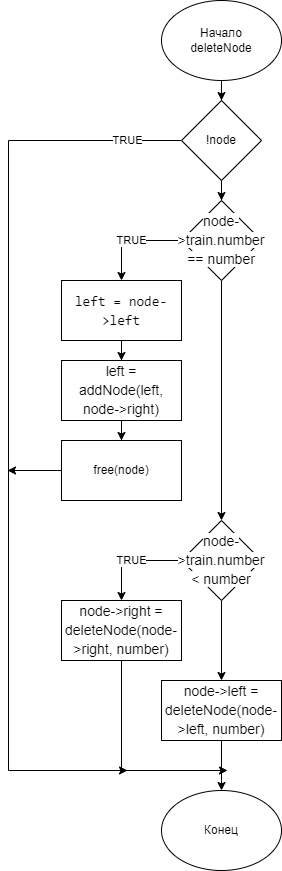


Рисунок Ж.5 – Структурная схема deleteNode

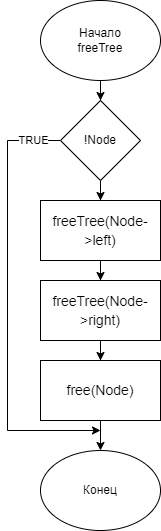


Рисунок Ж.6 – Структурная схема freeTree



Рисунок Ж.7 – Структурная схема setTime

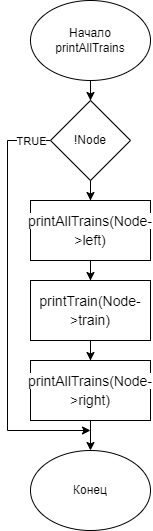


Рисунок Ж.8 – Структурная схема printAllTrains

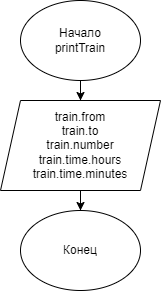


Рисунок Ж.9 – Структурная схема printTrain

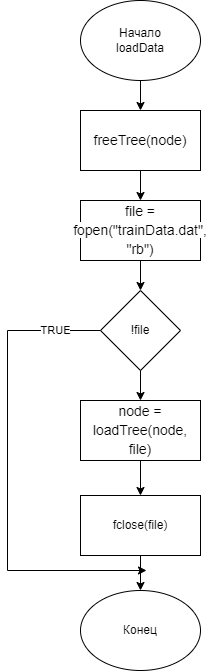


Рисунок Ж.10 – Структурная схема loadData

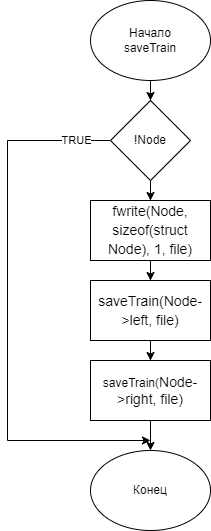


Рисунок Ж.11 – Структурная схема saveTrain

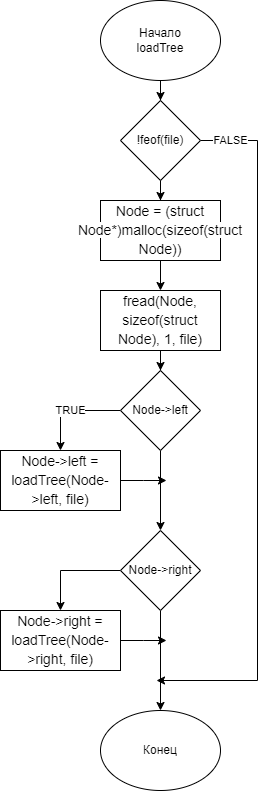


Рисунок Ж.12 – Структурная схема loadTree

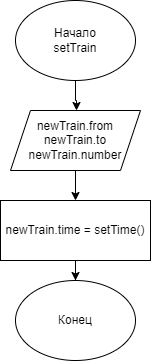


Рисунок Ж.13 – Структурная схема setTrain

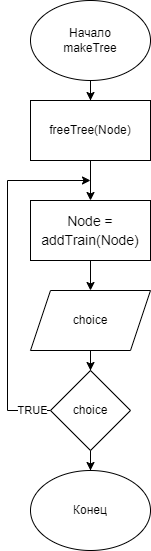


Рисунок Ж.14 – Структурная схема makeTree

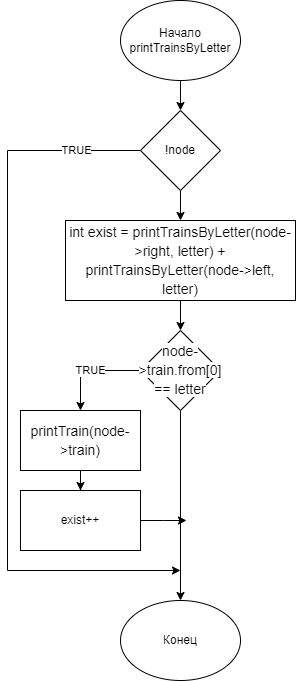


Рисунок Ж.15 – Структурная схема printTrainsByLetter

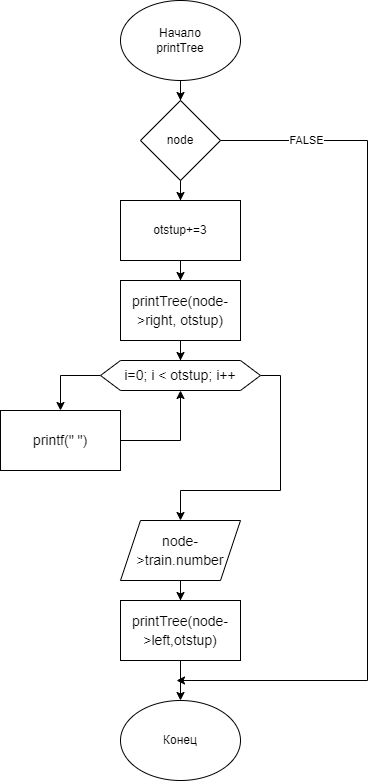


Рисунок Ж.16 – Структурная схема printTree

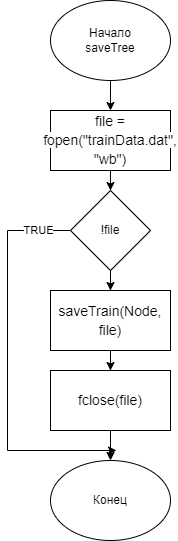


Рисунок Ж.17 – Структурная схема saveTree

**Приложение З**

**Тесты**

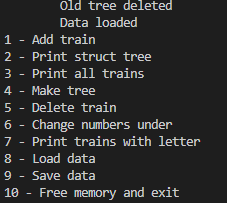
****

Рисунок З.1 – Тест 1

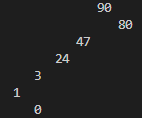


Рисунок З.2 – Структура дерева