1. Лабораторная работа № 9

«ПРОГРАММИРОВАНИЕ НЕЛИНЕЙНЫХ СТРУКТУР ДАННЫХ»

* 1. Цель работы

Исследование нелинейных структур данных и приобретение навыков разработки и отладки программ, использующих древовидные структуры. Исследование особенностей работы с поисковыми бинарными деревьями.

## Постановка задачи

Научиться работать с древовидными структурами данных, осуществлять поиск в бинарных деревьях.

* 1. Ход выполнения работы

Вариант выполнения задания – 18.

* + 1. Описание алгоритма решения задачи
       1. Определение входных и выходных данных

В меню на вход поступает число от 1 до 7, любое другое число будет проигнорировано. В соответствие с выбранным действием запускается выполнение определённой процедуры.

При создании дерева или добавлении элемента в дерево требуется ввести ФИО, Год рождения, Пол, Семейное положение, Количество детей и Оклад.

На каждое действие пользователя программа реагирует сообщением.

* + - 1. Структурная схема алгоритма

На Рисунках 9.1 – 9.4 представлены структурные схемы процедур, функции и основной программы.

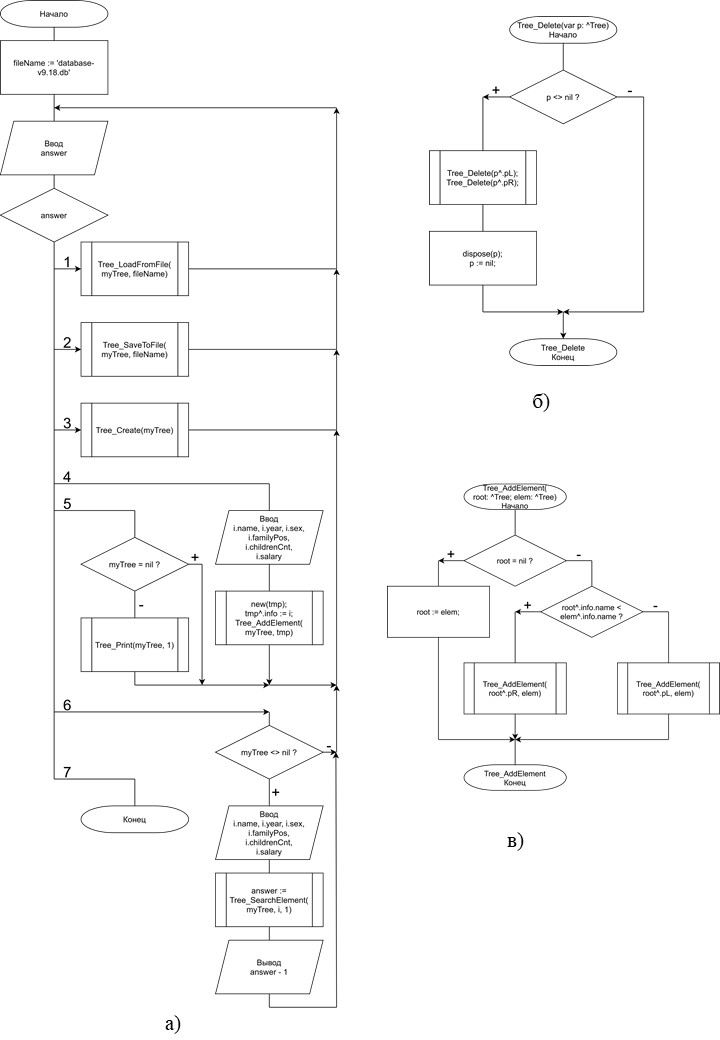


Рисунок 9.1 – Структурные схемы: а) основной программы;

б) процедуры Tree\_Delete; в) процедуры Tree\_AddElement

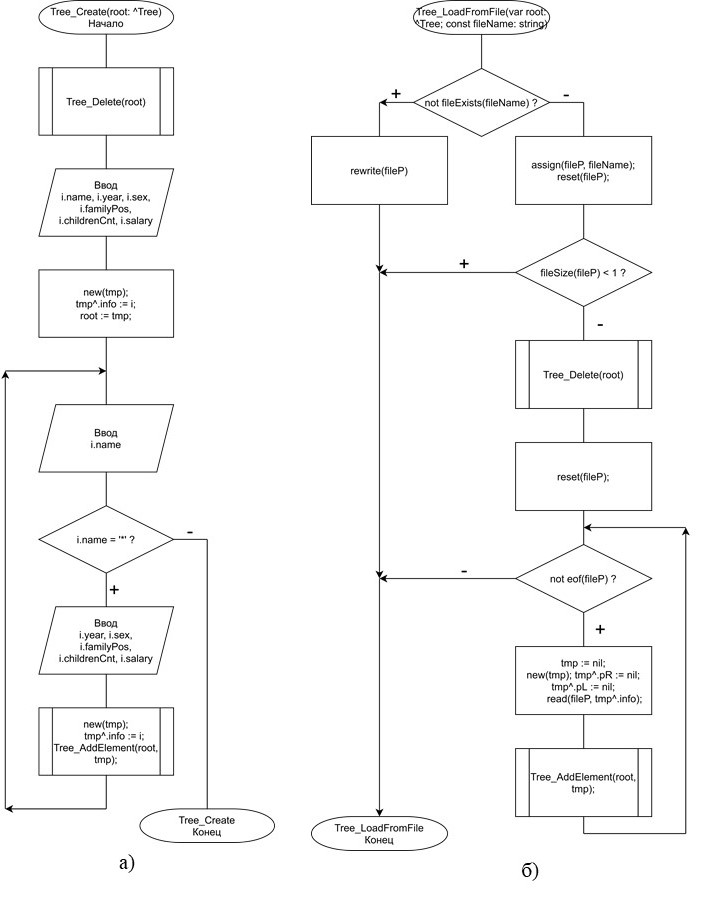


Рисунок 9.2 – Структурные схемы: а) процедуры Tree\_Create;

б) процедуры Tree\_LoadFromFile

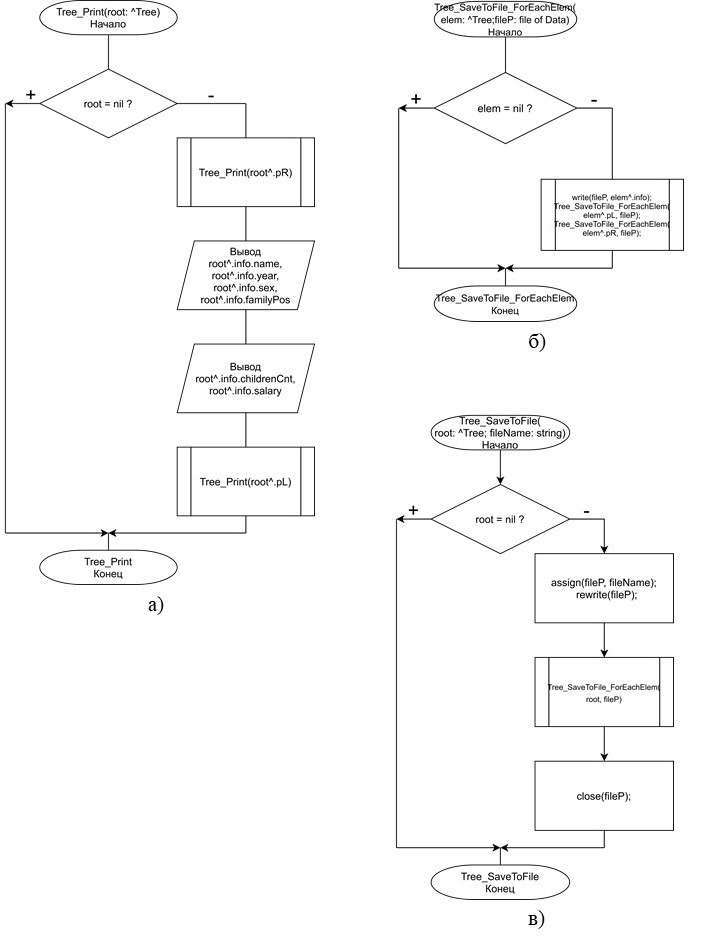


Рисунок 9.3 – Структурные схемы: а) процедуры Tree\_Print;

б) процедуры Tree\_SaveToFile\_ForEachElem;

в) процедуры Tree\_SaveToFile

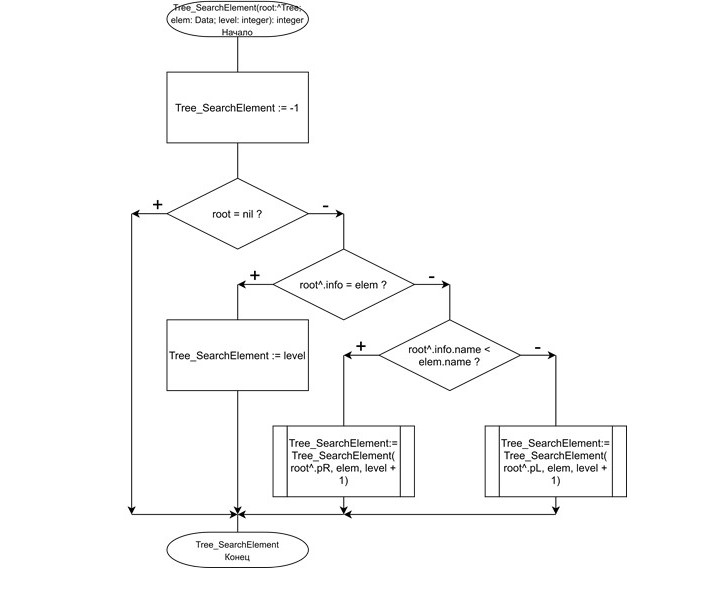


Рисунок 9.4 – Структурная схема процедуры Tree\_SearchElement

* + 1. Текст программы

Ниже представлен реализация программы на языке программирования Pascal.

Uses Crt;

Type

Data = record

name: string[32];

year: integer;

sex: string[10];

familyPos: string[16];

childrenCnt: word;

salary: integer;

end;

Tree = record

info: Data;

pL, pR: ^Tree;

end;

// Процедура интерактивного вывода сообщений

procedure alert(const messageType: word; const message: string);

const maxWidth = 68;

begin

case (messageType div 100) of

1: writeln('<===================>', ' ':(maxWidth-40), '<===================>');

end;

case ((messageType mod 100) div 10) of

1: begin writeln('| ', message, ' |':(maxWidth-length(message))); end;

2: begin writeln('| Введите ', message); write ('| пользователь$: ') end;

3: begin writeln('| ', ' ( ! ) Внимание ( ! )', ' |':(maxWidth-22));

writeln('| ', message, ' |':(maxWidth-length(message))) end;

4: begin writeln('| ', ' ( x ) Ошибка ( x )', ' |':(maxWidth-20));

writeln('| ', message, ' |':(maxWidth-length(message))) end;

5: begin writeln('| ', message) end;

end;

case (messageType mod 10) of

1: writeln('<===================>', ' ':(maxWidth-40), '<===================>');

end;

end;

// Процедура удаления дерева и его поддеревьев

procedure Tree\_Delete(var p: ^Tree);

begin

if p <> nil then

begin

// Очистка левой и правой ветви

Tree\_Delete(p^.pL); Tree\_Delete(p^.pR);

// Удаление корня

dispose(p); p := nil

end

end;

// Процедура рекурсивная для добавления элементов дерева

procedure Tree\_AddElement(var root: ^Tree; const elem: ^Tree);

begin

if root = nil then

begin root := elem; exit end;

if root^.info.name < elem^.info.name then Tree\_AddElement(root^.pR, elem)

else Tree\_AddElement(root^.pL, elem);

end;

// Процедура создания нового дерева или нового поддерева

procedure Tree\_Create (var root: ^Tree);

var tmp: ^Tree; i: Data;

begin

Tree\_Delete(root);

alert(020, 'ФИО товарища (выйти нельзя)'); readln(i.name);

alert(020, 'год рождения'); readln(i.year);

alert(020, 'пол товарища'); readln(i.sex);

alert(020, 'семейное положение'); readln(i.familyPos);

alert(020, 'количество детей'); readln(i.childrenCnt);

alert(020, 'оклад'); readln(i.salary);

new(tmp); tmp^.info := i; root := tmp;

alert(110, 'Подготовка к вводу второго товарища (ввод можно будет отменить).');

alert(120, 'любую кнопку для ввода товарища...'); readkey;

while true do

begin

clrscr;

alert(110, 'Переход к вводу следующего товарища...');

alert(020, 'ФИО (\* для выхода)'); readln(i.name);

if i.name = '\*' then break;

alert(020, 'год рождения'); readln(i.year);

alert(020, 'пол товарища'); readln(i.sex);

alert(020, 'семейное положение'); readln(i.familyPos);

alert(020, 'количество детей'); readln(i.childrenCnt);

alert(020, 'оклад'); readln(i.salary);

new(tmp); tmp^.info := i; Tree\_AddElement(root, tmp);

alert(110, 'Подготовка к вводу следующего товарища.');

alert(120, 'любую кнопку для ввода следующего товарища...');

readkey;

end;

end;

// Процедура отображения дерева или его подчасти

procedure Tree\_Print(const root: ^Tree; const level: integer);

var m: integer;

begin

if root = nil then exit;

m := round( ( log(level) + 0.2 ) \* 8 );

writeln(); Tree\_Print(root^.pR, level + 1);

write('':(m\*(level-1))); alert(050, concat(' Уровень: ', intToStr(level)));

write('':(m\*(level-1))); alert(050, concat(' ФИО: ', root^.info.name));

write('':(m\*(level-1))); alert(050, concat('Год рождения: ', intToStr(root^.info.year)));

write('':(m\*(level-1))); alert(050, concat('Пол товарища: ', root^.info.sex));

write('':(m\*(level-1))); alert(050, concat('Семейное положение: ', root^.info.familyPos));

write('':(m\*(level-1))); alert(050, concat('Количество детей: ', intToStr(root^.info.childrenCnt)));

write('':(m\*(level-1))); alert(050, concat(' Оклад: ', intToStr(root^.info.salary)));

writeln(); Tree\_Print(root^.pL, level + 1);

end;

// Процедура сохранение одного элемента

procedure Tree\_SaveToFile\_ForEachElem(const elem: ^Tree; var fileP: file of Data);

begin

if elem = nil then exit;

write(fileP, elem^.info);

Tree\_SaveToFile\_ForEachElem(elem^.pL, fileP);

Tree\_SaveToFile\_ForEachElem(elem^.pR, fileP);

end;

// Процедура сохранения дерева в файл

procedure Tree\_SaveToFile(const root: ^Tree; const fileName: string);

var fileP: file of Data;

begin

if root = nil then

begin

alert(040, 'Нечего сохранять в файл, а значит данные в файле сохранены.'); exit

end;

assign(fileP, fileName); rewrite(fileP); Tree\_SaveToFile\_ForEachElem(root, fileP);

close(fileP); alert(010, 'Данные успешно загружены в файл.');

end;

// Процедура загрузки дерева из файла

procedure Tree\_LoadFromFile(var root: ^Tree; const fileName: string);

var fileP: file of Data; tmp: ^Tree; counter: integer;

begin

assign(fileP, fileName);

if not fileExists(fileName) then

begin

alert(040, 'Файл сохранений не доступен для чтения. Будет создан новый файл.'); rewrite(fileP);

end

else

begin

reset(fileP);

if fileSize(fileP) < 1 then

alert(040, 'Данные в файле отсутствуют. Ничего не загруженено, всё сохранено.')

else

begin

Tree\_Delete(root); counter := 0; reset(fileP);

while not eof(fileP) do

begin

new(tmp); tmp^.pR := nil; tmp^.pL := nil; read(fileP, tmp^.info);

Tree\_AddElement(root, tmp); tmp := nil; counter := counter + 1;

end;

alert(010, concat('Успешно импортировано записей: ', intToStr(counter), '.'));

end

end;

close(fileP)

end;

// \*my\* Процедура поиска элемента в дереве

function Tree\_SearchElement(const root:^Tree; const elem: Data; const level: integer): integer;

begin

Tree\_SearchElement := -1;

if root = nil then exit;

if root^.info = elem then Tree\_SearchElement := level

else

if root^.info.name < elem.name then

Tree\_SearchElement := Tree\_SearchElement(root^.pR, elem, level + 1)

else

Tree\_SearchElement := Tree\_SearchElement(root^.pL, elem, level + 1);

end;

// Основная программа

var myTree, tmp: ^Tree; answer: integer; answerChar: char; fileName: string[128];

i: Data; isDataChanged: boolean;

BEGIN

isDataChanged := false; fileName := 'database-v9.18.db';

repeat

clrscr;

alert(110, 'Интерактивное меню программы');

alert(010, concat('1 -> Загрузить дерево из файла (по умолчанию "', fileName, '").'));

alert(010, concat('2 -> Сохранить дерево в файл (по умолчанию "', fileName, '").'));

alert(010, '3 -> Удалить старое и создать новое дерево.');

alert(010, '4 -> Добавить элемент в дерево.');

alert(010, '5 -> Печать дерева на экран (слева направо).');

alert(010, '6 -> \* my \* Найти элемент и вывести длину пути.');

alert(010, '7 -> Выход из программы.');

alert(120, 'номер действия'); readln(answer); clrscr;

case answer of

1: begin

alert(130, 'Дерево в памяти будет удалено.');

alert(011, 'Запущен процесс загрузки нового дерева...');

Tree\_LoadFromFile(myTree, fileName);

alert(020, 'какой-то символ для продолжения...');

isDataChanged := false;

end;

2: begin

alert(130, 'Сохранённое в файле дерево будет удалено.');

alert(011, 'Запущен процесс сохранения дерева...');

Tree\_SaveToFile(myTree, fileName);

alert(020, 'какой-то символ для продолжения...'); isDataChanged := false;

end;

3: begin

alert(130, 'Старое дерево удалено.');

alert(011, 'Запущен процесс создания нового дерева...');

Tree\_Create(myTree);

alert(110, 'Новые элементы были успешно добавлены.');

alert(020, 'какой-то символ для продолжения...');

isDataChanged := true;

end;

4: begin

alert(111, 'Запуск процесса добавления элемента в дерево...');

alert(020, 'ФИО (выйти нельзя)'); readln(i.name);

alert(020, 'год рождения'); readln(i.year);

alert(020, 'пол товарища'); readln(i.sex);

alert(020, 'семейное положение'); readln(i.familyPos);

alert(020, 'количество детей'); readln(i.childrenCnt);

alert(020, 'оклад'); readln(i.salary);

new(tmp); tmp^.info := i;

Tree\_AddElement(myTree, tmp);

alert(110, 'Новый элемент был успешно добавлен.');

alert(020, 'какой-то символ для продолжения...');

isDataChanged := true;

end;

5: begin

alert(111, 'Запуск процесса печати дерева на экран...');

if myTree = nil then alert(040, 'Дерево в оперативной памяти не существует.')

else

begin

Tree\_Print(myTree, 1); alert(110, 'Всё дерево расположилось на ЖК экране.');

end;

alert(020, 'какой-то символ для продолжения...');

end;

6:

begin

alert(111, 'Запуск процесса поиска элемента в дереве...');

if myTree <> nil then

begin

alert(020, 'ФИО (выйти нельзя)'); readln(i.name);

alert(020, 'год рождения'); readln(i.year);

alert(020, 'пол товарища'); readln(i.sex);

alert(020, 'семейное положение'); readln(i.familyPos);

alert(020, 'количество детей'); readln(i.childrenCnt);

alert(020, 'оклад'); readln(i.salary);

answer := Tree\_SearchElement(myTree, i, 1);

if answer = -1 then alert(141, 'Увы, но введённый элемент не найден...')

else

alert(111, concat('Найдено. Длина пути: ', intToStr(answer - 1), ' у.е.'));

end

else alert(041, 'Для начала заполните дерево.');

alert(020, 'какой-то символ для продолжения...')

end;

7: begin

alert(110, 'Поступило предложение выйти из программы.');

if isDataChanged then

begin

alert(130, 'Есть несохранённые данные.');

alert(020, 'символ для выбора действия (Y,Д/n,н)'); readln(answerChar);

if not ( (lowerCase(answerChar) = 'n') or (lowerCase(answerChar) = 'н') ) then

Tree\_SaveToFile(myTree, fileName);

end;

alert(020, 'какой-то символ для выхода...');

end;

end;

readkey;

until answer = 7;

END.

* + 1. Описание тестовых примеров

На Рисунках 9.5 – 9.14 продемонстрированы тестовые примеры, показывающие работоспособность программы.

На Рисунке 9.5 показано меню программы.

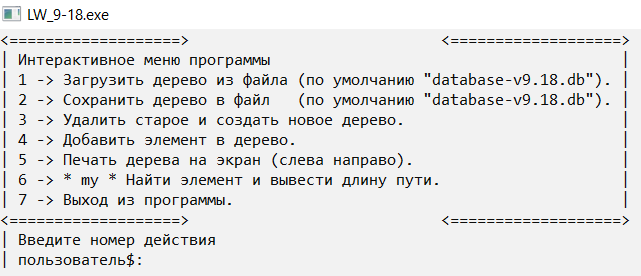


Рисунок 9.5 – Меню программы

На Рисунке 9.6 показана работоспособность процедуры, импортирующей данные из файла. Файл уже хранит какие-то данные.

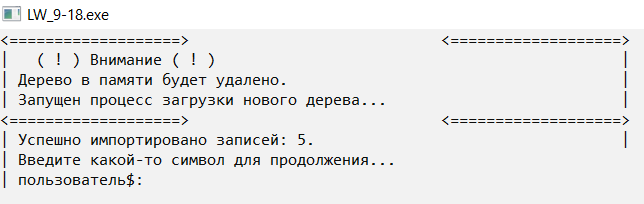


Рисунок 9.6 – Импорт данных в программу (данные в файле присутствуют)

На Рисунке 9.7 показана работоспособность процедуры, импортирующей данные из файла. Файл уже хранит какие-то данные.

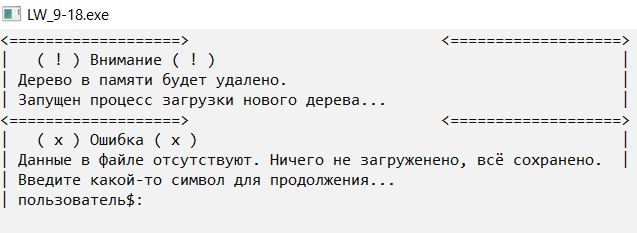


Рисунок 9.7 – Импорт данных в программу (данные в файле отсутствуют)

На Рисунке 9.8 показан процесс организации дерева на примере ввода одного элемента. Ввод будет осуществляться до тех пор, пока пользователь вместо ФИО не введёт звёздочку.

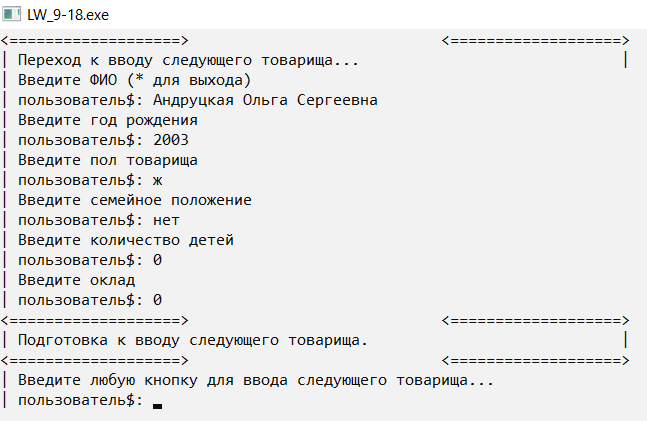


Рисунок 9.8 – Процесс организации дерева

На Рисунке 9.9 показан процесс добавления одного элемента в дерево.

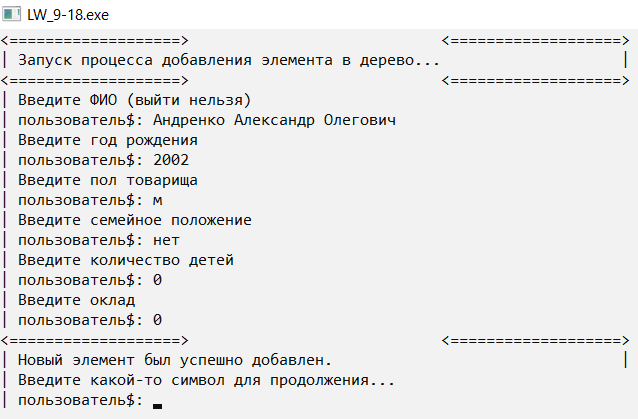


Рисунок 9.9 – Добавление элемента в дерево

На Рисунке 9.10 показано, как программа отображает дерево на экране.

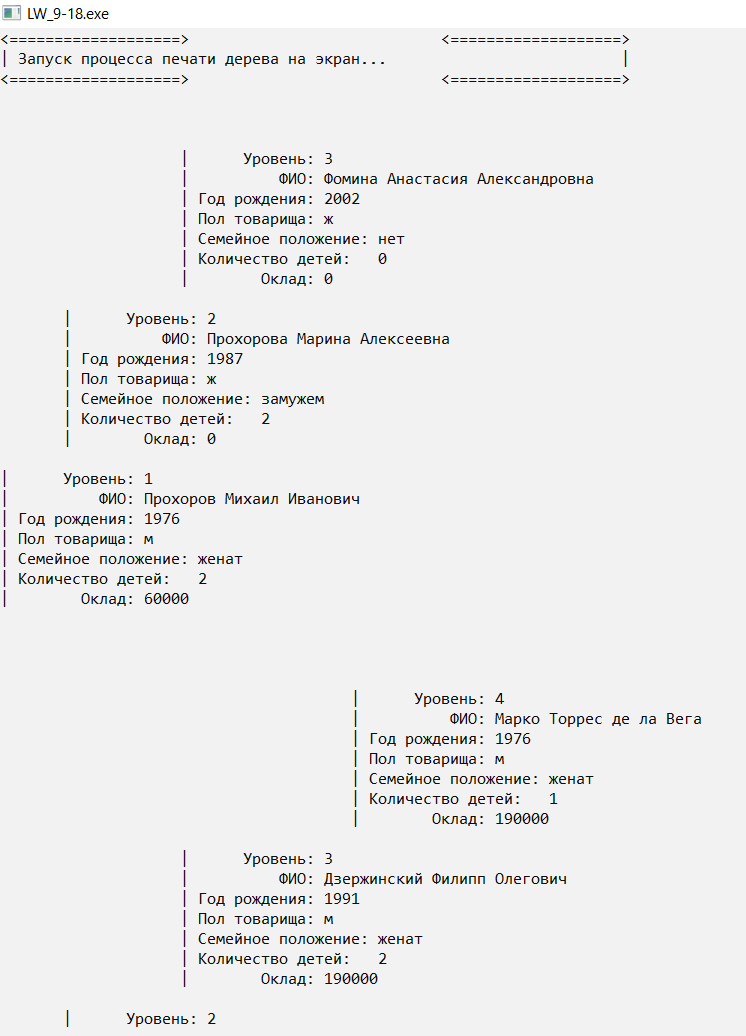


Рисунок 9.10 – Отображение дерева на экране

На Рисунках 9.11 и 9.12 продемонстрирована работоспособность процедуры поиска элемента в дереве и, если этот элемент существует, программа также выводит длину пути. В первом случае элемент был найден, и поэтому программа вывела длину пути. Во втором случае найти введённый элемент не удалось, поэтому программа вывела соответствующую ошибку.

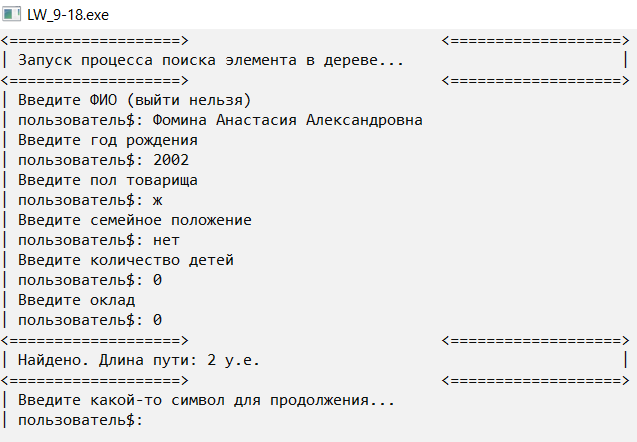


Рисунок 9.11 – Поиск элемента в дереве (элемент существует)

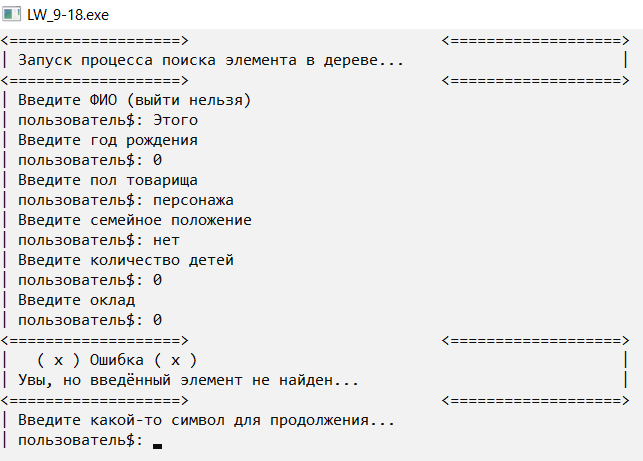


Рисунок 9.12 – Поиск элемента в дереве (элемент не существует)

На Рисунке 9.13 показан процесс сохранения данных в файл. Данные, содержащиеся файле, автоматически удаляются.

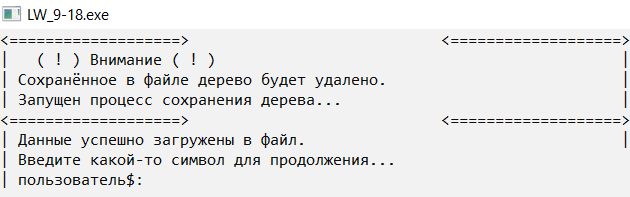


Рисунок 9.13 – Сохранение данных в файл

Дополнительная функция программы продемонстрирована на Рисунке 9.14. Она заключается в следующем: если после совершения каких-либо действий над деревом пользователь забыл сохранить данные, программа предупредит об этом и предложит сохраниться.

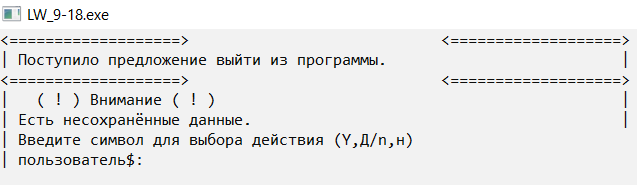


Рисунок 9.14 – Дополнительная функция

## Вывод

При выполнении данной лабораторной работы были получены навыки разработки приложений, умеющих работать с бинарными деревьями; написаны процедуры и функции, осуществляющие главные операции над деревьями, а именно: Организация дерева, Добавление элементов, Форматированная печать дерева и Полное удаление дерева. Также были повторно закреплены навыки отладки программы, работы с файлами, а именно чтение данных из бинарного типизированного файла и запись в него, работы с ссылками. Полученные навыки в будущем помогут создавать более сложные структуры деревьев, использовать деревья для упорядоченного хранения данных и эффективного поиска в них.