# ЛАБОРАТОРНАЯ РАБОТА №4

# «ПРОГРАММИРОВАНИЕ НЕЛИНЕЙНЫХ СТРУКТУР ДАННЫХ»

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++.

Задания

1. Структуру данных из предыдущей лабораторной работы использовать в качестве информационного поля бинарного дерева. Написать функции организации и просмотра дерева, отображения структуры дерева, добавления узла в дерево, исключения узла из дерева, сохранения дерева в файл, загрузки данных из файла и создания по этим данным нового дерева, освобождения динамической памяти (обязательно вызывается при выходе из программы). Работу программы оформить в виде меню;
2. Написать функцию, печатающую элементы всех листьев дерева, и функцию, печатающую информацию о человеке, чей номер телефона был введен с клавиатуры (если такого нет – функция должна выдать на дисплей соответствующее сообщение);
3. Разработать структурные схемы алгоритма для ряда функций решения задачи (main, просмотр списка, удаление узла, поиск и вывод требуемого узла);
4. Разработать тестовые примеры и выполнить отладку программы;

Текст программы

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

*//---------структуры и константы-------------*

*//структура для даты*

**struct** date {

**short** day; *//число*

**short** month; *//месяц*

**short** year; *//год*

};

*//структура для данных о человеке*

**struct** NOTE {

**int** n; *//номер*

**char** name[50]; *//имя*

**char** num[15]; *//номер телефона*

**date** birth; *//дата рождения*

};

*//структура для дерева*

**struct** tree {

**struct** **NOTE** inf; *//инф. поле*

**struct** **tree** \*right; *//правое ПД*

**struct** **tree** \*left; *//левое ПД*

};

*//------------прототипы функций-------------*

**tree**\* create\_tree(**tree** \*start); *//организация дерева*

**void** bypass\_tree(**tree** \*node); *//обход дерева*

**void** print\_tree(**tree** \*node, **int** l); *//просмотр структуры дерева*

**tree**\* del\_node(**tree** \*node, **int** x); *//удаление узла*

**tree**\* add\_node(**tree** \*start); *//добавление узла*

**void** load\_tree(**tree** \*t, **FILE** \*f); *//выгрузка в файл*

**tree**\* dwnld\_tree(**tree** \*node); *//загрузка из файла*

**void** print\_leafs(**tree** \*node); *//просмотр листьев*

**int** info(**tree** \*t, **char** \*s, **int** x); *//просмотр искомого узла*

**void** clean\_tree(**tree** \*node); *//очищение памяти*

*//-----главная функция - меню программы-----*

**int** main() {

**tree** \*start=NULL; *//указатель на начало списка*

**while** (1) {

*//текстовые указания к меню*

puts("----Menu----");

puts("1 - Organize tree");

puts("2 - Bypass tree");

puts("3 - Add node");

puts("4 - Delete node");

puts("5 - Print tree's structure");

puts("6 - Load tree in file");

puts("7 - Downoad tree from file");

puts("8 - Print all leafs");

puts("9 - Search by phone number");

puts("0 - End");

**int** c;

printf("\nEnter command - ");

scanf("%d",&c);

*//вызов функций меню*

**switch** (c) {

**case** 1: { *//организация дерева*

printf("\n----Tree organiser----\n");

start=create\_tree(start); **break**;

}

**case** 2: { *//обход дерева*

printf("\n----Bypass tree----\n");

**if** (!start) {

printf("No tree found\n\n");

**break**; }

bypass\_tree(start); **break**;

}

**case** 3: { *//добавление узлов*

printf("\n----Add nodes----\n");

**if** (!start) {

printf("No tree found\n\n");

**break**; }

start=add\_node(start); **break**;

}

**case** 4: { *//удаление узла*

**if** (!start) {

printf("No tree found\n\n");

**break**; }

printf("Enter key number - ");

**int** x; scanf("%d",&x); *//номер искомого узла*

start=del\_node(start, x);

puts("Node deleted\n"); **break**;

}

**case** 5: { *//вывод структуры дерева*

printf("\n----Tree's structure----\n");

**if** (!start) {

printf("No tree found\n\n");

**break**; }

print\_tree(start, 3);

puts(""); **break**;

}

**case** 6: { *//выгрузка в файл*

**if** (!start) {

printf("No tree found\n\n");

**break**; }

**FILE** \*f;

**if** ((f = fopen("data.txt","wt"))==NULL) {

fprintf(stderr,"Can't open output file \n");

**break**; }

load\_tree(start, f);

printf("Tree loaded\n\n");

fclose(f); **break**;

}

**case** 7: { *//загрузка из файла*

start=dwnld\_tree(start);

printf("Tree downloaded\n\n");

**break**;

}

**case** 8: { *//вывод листьев дерева*

printf("\n----Tree's leafs----\n");

**if** (!start) {

printf("No tree found\n\n");

**break**; }

print\_leafs(start); **break**;

}

**case** 9: { *//поиск и вывод узла*

**if** (!start) {

printf("No tree found\n\n");

**break**; }

**char** srch[15]; *//искомый номер телефона*

printf("Enter key phone number - ");

scanf("%s",&srch);

**if** (!info(start, srch, 0))

printf("No such node\n\n");

**break**;

}

**case** 0: { *//конец программы*

clean\_tree(start);

printf("\nThe end.");

**return** 1;

}

*//команды не существует*

**default**: puts("No such command\n");

}

}

}

*//------заполнение информационного поля-----------*

**NOTE** read\_note() {

**NOTE** name;

printf("\n Number (0-exit) - "); *//номер*

scanf("%d",&name.n);

**if** (!name.n) { *//конец записи*

printf("\n");

**return** name; }

printf(" Name - "); *//имя*

scanf("%s",&name.name);

printf(" Phone number - "); *//телефон*

scanf("%s",&name.num);

printf(" Day - "); *//день*

scanf("%d",&name.birth.day);

printf(" Month - "); *//месяц*

scanf("%d",&name.birth.month);

printf(" Year - "); *//год*

scanf("%d",&name.birth.year);

**return** name;

}

*//------вывод информационного поля-----------*

**void** print\_note(**NOTE** name) {

printf(" Number - %d\n",name.n);

printf(" Name - %s\n",name.name);

printf(" Phone number - %s\n",name.num);

printf(" Birthday - ");

**if** (name.birth.day>9)

printf("%d.",name.birth.day);

**else** printf("0%d.",name.birth.day);

**if** (name.birth.month>9)

printf("%d.",name.birth.month);

**else** printf("0%d.",name.birth.month);

printf("%d\n\n",name.birth.year);

}

*//---------организация дерева--------------*

**tree**\* create\_tree(**tree** \*start) {

*//очистка памяти*

clean\_tree(start);

start=NULL;

*//создание корня списка*

start=(**tree**\*)malloc(**sizeof**(**tree**));

start->inf = read\_note();

start->left = NULL;

start->right = NULL;

**if** (start->inf.n==0)

**return**(NULL);

*//заполнение элементов*

start=add\_node(start);

**return** start;

}

*//---------добавление узлов--------------*

**tree**\* add\_node(**tree** \*start) {

**while** (1) {

*//создание узла*

**tree** \*t;

t=(**tree**\*)malloc(**sizeof**(**tree**));

t->inf = read\_note();

t->left = NULL;

t->right = NULL;

*//проверка на конец ввода (n=0)*

**if** (t->inf.n==0) {

free(t);

**return**(start); }

*//добавление узла в дерево*

**tree** \*end=start; *//последний лист*

**tree** \*pnlt; *//предпоследний лист*

**while** (1) {

pnlt=end;

**if** (t->inf.n < end->inf.n) {

end=end->left;

**if** (end==NULL) {

pnlt->left=t;

**break**; }

}

**else** {

end=end->right;

**if** (end==NULL) {

pnlt->right=t;

**break**; } } } }

}

*//------------обход дерева-----------------*

**void** bypass\_tree(**tree** \*t) {

**if** (t) {

bypass\_tree(t->left);

print\_note(t->inf);

bypass\_tree(t->right);

**return**; }

}

*//---------удаление узла--------------*

**tree**\* del\_node(**tree** \*node, **int** x) {

**if** (!node) **return** node; *//проверка наличия узла*

**if** (x == node->inf.n) { *//найден искомый узел*

**tree** \*t;

**if** (!node->right) t=node->left; *//нет ПД справа*

**else** {

**tree** \*p=node->right; *//вспом. указатель (правое ПД)*

**if** (!p->left) { *//у правого ПД нет ПД слева*

p->left=node->left;

t=p;

}

**else** {

**tree** \*pmin = p->left; *//самый левый узел в правом ПД*

**while** (pmin->left) { *//поиск pmin*

p=pmin;

pmin=p->left;

}

p->left = pmin->right;

pmin->left = node->left;

pmin->right = node->right;

t = pmin;

}

}

free(node);

**return** t; }

**else** *//поиск в левом ПД*

**if** (x<node->inf.n)

node->left=del\_node(node->left, x);

**else** *//поиск в правом ПД*

node->right=del\_node(node->right, x);

**return** node;

}

*//-----------вывод структуры дерева-------------*

**void** print\_tree(**tree** \*t, **int** l) {

**if** (t) {

print\_tree(t->right, l+3); *//левое ПД*

**for** (**int** i=0; i<l; i++) *//вывод отступа*

printf(" ");

printf("%s\n",t->inf.name);

print\_tree(t->left, l+3); *//правое ПД*

**return**; }

}

*//------------листья дерева-----------------*

**void** print\_leafs(**tree** \*t) {

**if** (!t) **return**; *//проверка на NULL*

print\_leafs(t->left); *//поиск в левом ПД*

*//проверка на наличие ПД и вывод листа*

**if** ((!t->left) && (!t->right))

print\_note(t->inf);

print\_leafs(t->right); *//поиск в правом ПД*

}

*//---------поиск по номеру телефона--------------*

**int** info(**tree** \*t, **char** \*s, **int** x) {

**if** (!t) **return** 0; *//проверка на NULL*

**if** (strcmp(t->inf.num,s)==0) {

*//вывод элемента*

printf("\n----Searched node----\n");

print\_note(t->inf);

**return** 1;

}

**else** {

x=x+info(t->left, s, x); *//поиск в левом ПД*

x=x+info(t->right, s, x); *//поиск в правом ПД*

}

**return** x;

}

*//------------запись в файл-----------------*

**void** load\_tree(**tree** \*t, **FILE** \*f) {

**if** (!t) **return**;

load\_tree(t->left, f);

fprintf(f,"%d\n",t->inf.n);

fprintf(f,"%s\n",t->inf.name);

fprintf(f,"%s\n",t->inf.num);

fprintf(f,"%d\n",t->inf.birth.day);

fprintf(f,"%d\n",t->inf.birth.month);

fprintf(f,"%d\n",t->inf.birth.year);

load\_tree(t->right, f);

**return**;

}

*//------чтение элемента списка из файла-----------*

**tree**\* read\_node(**FILE** \*f, **tree** \*t) {

**if** (fscanf(f,"%d",&t->inf.n)==EOF) **return** NULL;

**if** (fscanf(f,"%s",&t->inf.name)==EOF) **return** NULL;

**if** (fscanf(f,"%s",&t->inf.num)==EOF) **return** NULL;

**if** (fscanf(f,"%d",&t->inf.birth.day)==EOF) **return** NULL;

**if** (fscanf(f,"%d",&t->inf.birth.month)==EOF) **return** NULL;

**if** (fscanf(f,"%d",&t->inf.birth.year)==EOF) **return** NULL;

t->left = NULL;

t->right = NULL;

**return** t;

}

*//------загрузка списка из файла----------*

**tree**\* dwnld\_tree(**tree** \*start) {

*//указатель на файл и его проверка*

**FILE** \*f;

**if** ((f = fopen("data.txt","rt"))==NULL) {

printf("Can't open output file \n");

**return** start; }

*//очищение дерева*

clean\_tree(start);

start=(**tree**\*)malloc(**sizeof**(**tree**));

*//чтение коря дерева*

start=read\_node(f, start);

**if** (!start) {

fclose(f);

**return** NULL; }

*//чтение остальных узлов*

**tree** \*t;

**while** (1) {

t=(**tree**\*)malloc(**sizeof**(**tree**));

t = read\_node(f, t);

*//конец файла*

**if** (!t) {

fclose(f);

**return** start; }

*//добавление узла в дерево*

**tree** \*end=start; *//последний лист*

**tree** \*pnlt; *//предпоследний лист*

**while** (1) {

pnlt=end;

**if** (t->inf.n < end->inf.n) {

end=end->left;

**if** (end==NULL) {

pnlt->left=t;

**break**;

}

}

**else** {

end=end->right;

**if** (end==NULL) {

pnlt->right=t;

**break**; } } } }

}

*//---------очищение памяти--------------*

**void** clean\_tree(**tree** \*node) {

**if** (!node) **return**; *//проверка на NULL*

clean\_tree(node->left); *//очистка левого ПД*

clean\_tree(node->right); *//очистка правого ПД*

free(node);

**return**;

}

Структурная схема программы

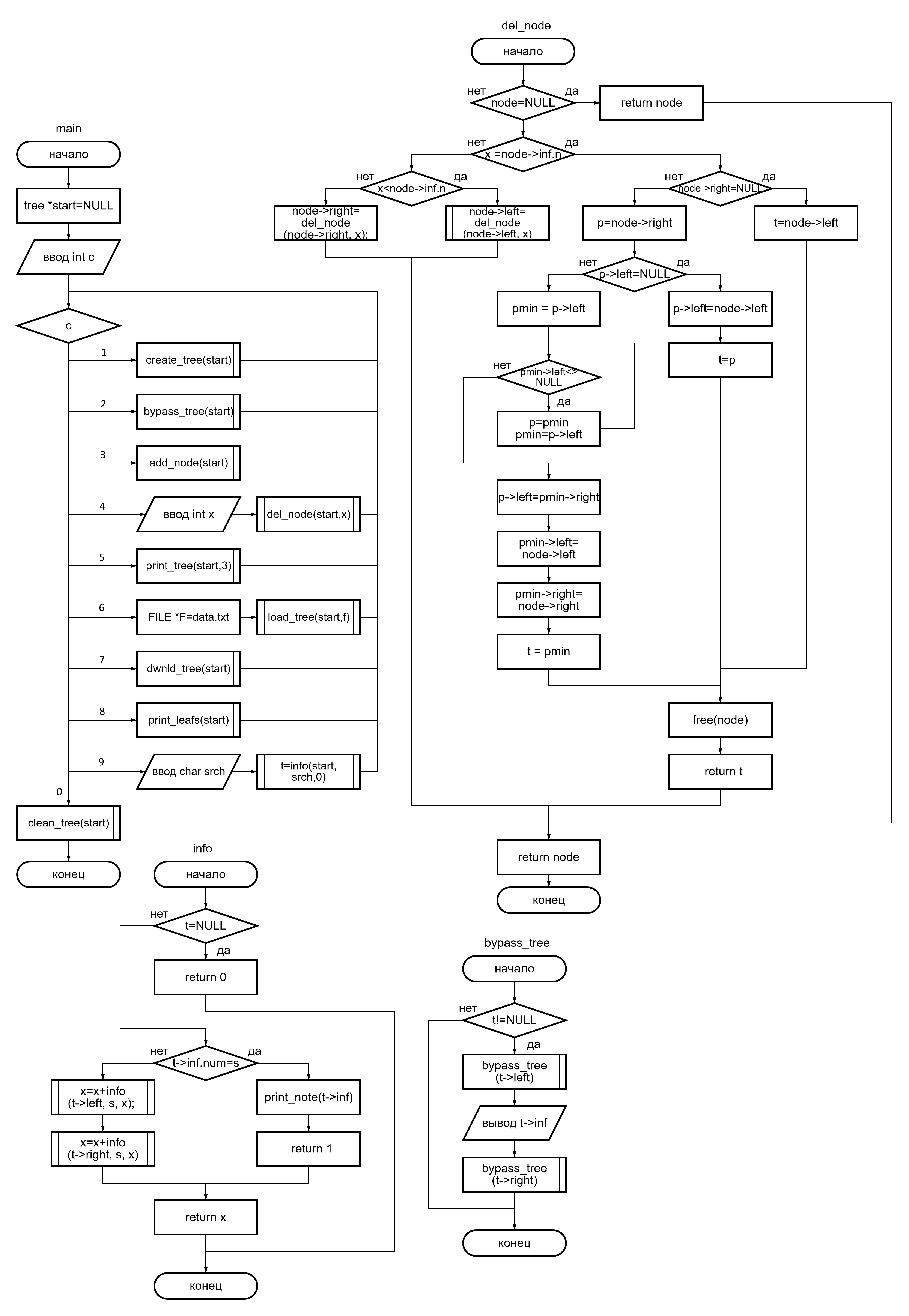


Рисунок 1 – Структурная схема функций программы

Тестовые примеры

С помощью функции организации дерева в программу были введены данные нового бинарного дерева.

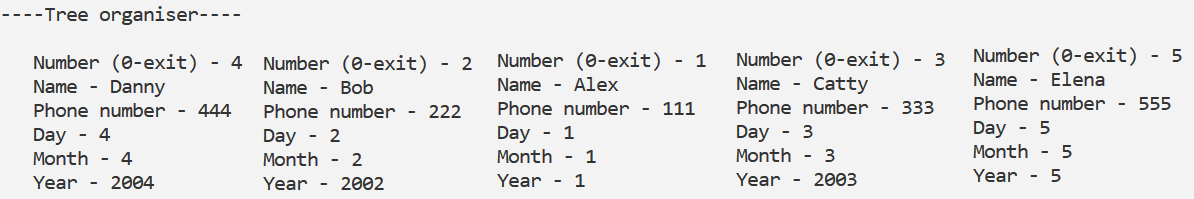


Рисунок 2 – Ввод бинарного дерева

С помощью функции обхода дерева введённое бинарное дерево выведено в консоль. Обход дерева произведён слева направо. В результате значения дерева были выведены в верном отсортированном порядке.

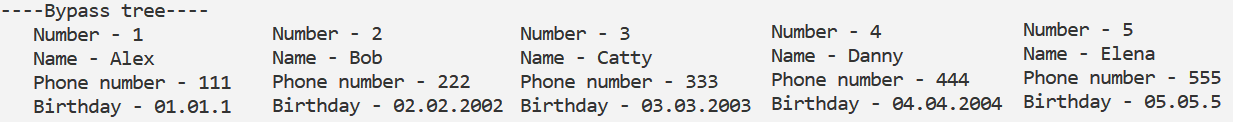


Рисунок 3 – Обход бинарного дерева

С помощью соответствующей функции на экран выведена структура введённого ранее бинарного дерева (структура расположена в горизонтальном виде слева направо и представлена в виде имён соответствующих узлов).

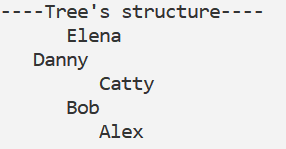


Рисунок 4 – Структура бинарного дерева

Из бинарного дерева удалён узел с номером 2. Далее на экран выведена структура обновлённого дерева.

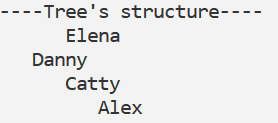


Рисунок 5 – Структура обновлённого дерева

С помощью соответствующей функции на экран выведены листья бинарного дерева.

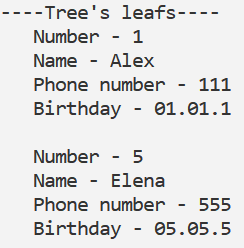


Рисунок 6 – Вывод листьев дерева

Произведена попытка поиска в дереве узла с номером телефона 7. В результате на экран выведено сообщение об ошибке.

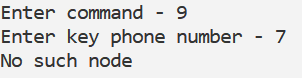


Рисунок 7 – Сообщение об ошибке

При поиске в дереве узла с номером телефона 111 на экран выведены данные соответствующего узла.

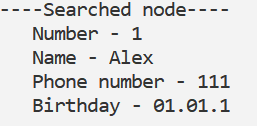


Рисунок 8 – Вывод искомого узла

Данные бинарного дерева выведены в текстовый файл data.txt.

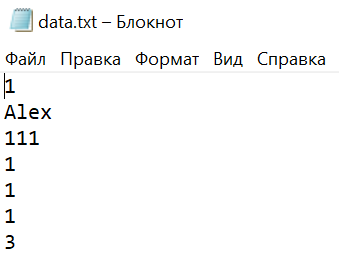


Рисунок 9 – Данные дерева в текстовом файле

Файл data.txt переписан новыми значениями узлов дерева. Вызвана функция чтения бинарного дерева из файла. Структура нового бинарного дерева выведена на экран.

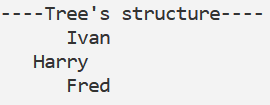


Рисунок 10 – Структура прочитанного дерева

Вызвана функция конца программы. Память под дерево была очищена, программа завершена.

Вывод

В ходе работы были изучены принципы работы с древовидными структурами данных на языках C/C++. Освоены алгоритмы обработки бинарных деревьев. Результатом работы стала программа, выполняющая структурирование данных в виде бинарного дерева и различные манипуляции с ними (добавление и удаление элементов, выгрузка и загрузка из файла, различные способы вывода данных, поиск в дереве и др.).