# Лабораторная работа № 8

«Программирование нелинейных структур данных на языке с/с++»

# Цель работы

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

# Ход выполнения работы

# Задание

Представить приведенную в предыдущей работе таблицу в виде бинарного дерева. Написать функции создания и обхода дерева, а также одну из функций, приведенных ниже. Значения полей и количество записей в таблице студент выбирает самостоятельно. Программа должна сохранять дерево в файле и создавать его заново при её повторном запуске. Таблица 7.3Функцию, которая находит в непустом дереве Т длину (число ветвей) пути от корня до вершины с элементом Е.

# Описание алгоритма решения задания

# Определение входных и выходных данных

В главном меню программы от пользователя требуется ввести число от 0 до 6. При вводе любого другого числа программа выводит ошибку.

При создании элемента дерева от пользователя требуется ввести ФИО (строка), Год рождения (число), Пол (символ), Семейное положение (строка), Количество детей (число), Оклад (число).

В меню поиска элемента в дереве от пользователя требуется ввести ФИО (строка). Если элемент не найден, программа выводит ошибку. Если элемент найден, то программа выводит Уникальный ID элемента и Длину пути.

# Структурная схема алгоритма

Структурная схемы алгоритма представлена в Приложении Б на Рисунках Б.4 – Б.7.

# Исходный код программы (ЯП С, ОС GNU/Linux Ubuntu, компилятор gcc)

* + - 1. Файл main.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "./tree.h"

#include "./files.h"

static unsigned long int LAST\_ID = 0;

void menu\_create\_tree (t\_leaf \*\*root);

int menu\_create\_from\_file (char \*fname, t\_leaf \*\*root);

int menu\_save\_to\_file (char \*fname, t\_leaf \*\*root);

void menu\_search\_elem (t\_leaf \*root);

void menu\_add\_one (t\_leaf \*\*root);

int main ()

{

t\_leaf \*root = NULL;

int answer = -1;

do

{

printf("\n\n\n");

printf("Действия: \n");

printf(" (0) Выйти из программы.\n");

printf(" (1) Загрузить из файла \".database.8\".\n");

printf(" (2) Сохранить в файл \".database.8\".\n");

printf(" (3) Создать новое дерево.\n");

printf(" (4) Добавить один элемент.\n");

printf(" (5) Печать дерева.\n");

printf(" (6) \*my\* Найти элемент в дереве.\n\n");

printf("$: "); scanf("%d", &answer);

printf("\n\n\n");

switch (answer)

{

case 0:

goto A;

break;

case 1:

if (!menu\_create\_from\_file(".database.8", &root))

printf("Может быть загружено!\n");

else

printf("Не загружено!\n");

break;

case 2:

if (!menu\_save\_to\_file(".database.8", &root))

printf("Может быть сохранено!\n");

else

printf("Не сохранено!\n");

break;

case 3:

menu\_create\_tree(&root);

break;

case 4:

menu\_add\_one(&root);

break;

case 5:

if (root)

print\_branch(root, 1);

else

printf("Дерево не существует!\n");

break;

case 6:

menu\_search\_elem(root);

break;

default:

printf("Неверное действие.\n");

break;

}

} while (1);

A:

return 0;

}

void menu\_create\_tree (t\_leaf \*\*root)

{

del\_branch(root); LAST\_ID = 0;

t\_leaf \*new\_leaf = NULL;

char answer[MAX\_CHAR\_SIZE\_ + 1];

do

{

new\_leaf = (t\_leaf \*) malloc( sizeof(t\_leaf) );

printf("Введите ФИО: "); scanf("%s", new\_leaf->data.FIO); fseek(stdin, 0, SEEK\_END);

printf("Введите Год рождения: "); scanf("%ld", &(new\_leaf->data.year)); fseek(stdin, 0, SEEK\_END);

printf("Введите Пол: "); scanf("%s", new\_leaf->data.gender); fseek(stdin, 0, SEEK\_END);

printf("Введите Семейное положение: "); scanf("%s", new\_leaf->data.status); fseek(stdin, 0, SEEK\_END);

printf("Введите Количество детей: "); scanf("%lu", &(new\_leaf->data.chcnt)); fseek(stdin, 0, SEEK\_END);

printf("Введите Оклад: "); scanf("%lu", &(new\_leaf->data.salary)); fseek(stdin, 0, SEEK\_END);

new\_leaf->id = LAST\_ID++;

add\_leaf(root, new\_leaf);

printf("Новый элемент добавлен. Его U\_ID: %lu. Продолжить ввод [Y/n] ?: ", new\_leaf->id); scanf("%s", answer);

} while (\*answer != 'n' && \*answer != 'N' && \*answer != 'н');

}

int menu\_create\_from\_file (char \*fname, t\_leaf \*\*root)

{

FILE \*f;

if (!(f = fopen(fname, "r"))) return 1;

del\_branch(root); LAST\_ID = 0;

t\_data data;

while (file\_getData(f, &data))

{

t\_leaf \*new\_leaf = create\_leaf(&data);

new\_leaf->id = LAST\_ID++;

add\_leaf(root, new\_leaf);

}

fclose(f);

return 0;

}

int menu\_save\_to\_file (char \*fname, t\_leaf \*\*root)

{

FILE \*f;

if (!(f = fopen(fname, "w"))) return 1;

save\_branch(\*root, f);

fclose(f);

return 0;

}

void menu\_search\_elem (t\_leaf \*root)

{

if (!root) {printf("Дерево пустое!\n"); return; }

char FIO[FIO\_LENGHT\_ \* MAX\_CHAR\_SIZE\_ + 1];

printf("Введите ФИО: "); scanf("%s", FIO); fseek(stdin, 0, SEEK\_END);

t\_leaf \*tmp = root; int result = 0, level = 0;

while (tmp)

{

level++;

result = strcmp(tmp->data.FIO, FIO);

if (!result)

{

printf("Элемент найден. Его u\_id: %lu. Длина: %d.\n", tmp->id, level - 1);

return;

}

else

if (result >= 0)

tmp = tmp->right;

else

tmp = tmp->left;

}

printf("Элемент не найден!\n");

}

void menu\_add\_one (t\_leaf \*\*root)

{

t\_leaf \*new\_leaf = (t\_leaf \*) malloc( sizeof(t\_leaf) );

printf("Введите ФИО: "); scanf("%s", new\_leaf->data.FIO); fseek(stdin, 0, SEEK\_END);

printf("Введите Год рождения: "); scanf("%ld", &(new\_leaf->data.year)); fseek(stdin, 0, SEEK\_END);

printf("Введите Пол: "); scanf("%s", new\_leaf->data.gender); fseek(stdin, 0, SEEK\_END);

printf("Введите Семейное положение: "); scanf("%s", new\_leaf->data.status); fseek(stdin, 0, SEEK\_END);

printf("Введите Количество детей: "); scanf("%lu", &(new\_leaf->data.chcnt)); fseek(stdin, 0, SEEK\_END);

printf("Введите Оклад: "); scanf("%lu", &(new\_leaf->data.salary)); fseek(stdin, 0, SEEK\_END);

new\_leaf->id = LAST\_ID++;

add\_leaf(root, new\_leaf);

printf("Новый элемент добавлен. Его U\_ID: %lu.\n", LAST\_ID);

}

* + - 1. Файл tree.h

#ifndef TREE\_H

#define TREE\_H 1

#ifndef MAX\_CHAR\_SIZE\_

#define MAX\_CHAR\_SIZE\_ 6

#endif

#define FIO\_LENGHT\_ 1024

#define STATUS\_LENGHT\_ 512

typedef struct T\_DATA {

char FIO[FIO\_LENGHT\_ \* MAX\_CHAR\_SIZE\_ + 1]; // ФИО

long int year; // Год рождения

char gender[MAX\_CHAR\_SIZE\_ + 1]; // Пол

char status[STATUS\_LENGHT\_ \* MAX\_CHAR\_SIZE\_ + 1]; // Семейное положение

unsigned long int chcnt; // Количество детей

unsigned long int salary; // Оклад

} t\_data;

typedef struct T\_LEAF {

unsigned long int id;

t\_data data;

struct T\_LEAF \*left;

struct T\_LEAF \*right;

} t\_leaf;

size\_t file\_saveData(FILE \*f, t\_data \*d);

int file\_getData (FILE \*f, t\_data \*d);

/\*\*

\* @brief Создаёт лист дерева.

\*

\* @param d

\* @return t\_leaf\*

\*/

t\_leaf \*create\_leaf (t\_data \*d)

{

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

t->right = t->left = NULL;

t->data = \*d;

return t;

}

/\*\*

\* @brief Удаляет ветку.

\*

\* @param p

\*/

void del\_branch (t\_leaf \*\*p)

{

if (!(\*p)) return;

del\_branch(&((\*p)->left));

del\_branch(&((\*p)->right));

free(\*p); \*p = NULL;

}

/\*\*

\* @brief Печать ветки на экран.

\*

\* @param p

\* @param margin\_left

\*/

void print\_branch (t\_leaf \*p, unsigned long int level)

{

if (!p) return;

int i;

print\_branch(p->left, level + 1);

for (i = 1; i++ < level; printf(" ")); printf("УРОВЕНЬ %lu <u\_id %lu>\n", level, p->id);

for (i = 1; i++ < level; printf(" ")); printf("ФИО: %-16s\n", p->data.FIO);

for (i = 1; i++ < level; printf(" ")); printf("Год рождения: %lu\n" , p->data.year);

for (i = 1; i++ < level; printf(" ")); printf("Пол: %s\n", p->data.gender);

for (i = 1; i++ < level; printf(" ")); printf("Семейное положение: %-16s\n", p->data.status);

for (i = 1; i++ < level; printf(" ")); printf("Количество детей: %lu\n", p->data.chcnt);

for (i = 1; i++ < level; printf(" ")); printf("Оклад: %lu\n", p->data.salary);

print\_branch(p->right, level + 1);

}

void add\_leaf (t\_leaf \*\*branch, t\_leaf \*leaf)

{

if (!(\*branch)) { \*branch = leaf; return; }

if ( strcmp( (\*branch)->data.FIO, leaf->data.FIO ) < 0 )

add\_leaf ( &((\*branch)->left), leaf );

else

add\_leaf ( &((\*branch)->right), leaf );

}

void save\_branch (t\_leaf \*elem, FILE \*f)

{

if (!elem) return;

file\_saveData(f, &(elem->data));

save\_branch(elem->left, f);

save\_branch(elem->right, f);

}

#endif

* + - 1. Файл files.h

#ifndef FILES\_H\_

#define FILES\_H\_ 1

size\_t file\_saveData(FILE \*f, t\_data \*d);

int file\_getData (FILE \*f, t\_data \*d);

size\_t file\_saveData (FILE \*f, t\_data \*d)

{

return fwrite(d, sizeof(t\_data), 1, f);

}

int file\_getData (FILE \*f, t\_data \*d)

{

return fread(d, sizeof(t\_data), 1, f);

}

#endif

# Описание тестовых примеров

Тестовые примеры представлены в Приложении В на Рисунках В.7 – В.11.

На Рисунке В.7 представлено главное меню программы.

На Рисунке В.8 продемонстрирован процесс создания элемента дерева. Пользователь вводит “Alex\_Fomin”, “1996”, “m”, “женат”, “1”, “76000”.

На Рисунке В.9 показан результат функции Вывода дерева на экран. Дерево состоит из двух элементов. Соблюдена иерархия.

На Рисунках В.10 и В.11 показан результат поиска в ранее созданном дереве. На первом Рисунке показан вывод программы, если элемент существует. На втором – если элемент не существует.

# Вывод

При выполнении данной лабораторной работы были получены навыки разработки приложений, умеющих работать с бинарными деревьями; написаны 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.