Санкт-Петербургский государственный политехнический университет

Кафедра “Прикладная математика”

**Отчет по лабораторной работе 2**

**“Алгоритмы и структуры данных”**

Студент группы № 5030102/20001

ФИО: Тишковец Сергей Евгеньевич

Выполнил (дата) 27.10.2023

Оглавление

[**Постановка задачи** 3](#_Toc149322136)

[**Описание алгоритма** 3](#_Toc149322137)

[**Текст программы** 5](#_Toc149322138)

[**Описание тестирования** 10](#_Toc149322139)

# **Постановка задачи**

***Вариант 1: Записная книжка***

**Написать программу, осуществляющую работу с базой данных «Записная книжка». Элемент данных - фамилия и телефон. Ключ для поиска – фамилия. Базу данных хранить в памяти в виде массива самоорганизующихся списков проиндексированного буквами алфавита. Добавление выполнять в начало соответствующего списка. Написать процедуры поиска, удаления и сортировки заданного списка. Базу данных зачитывать и сохранять в файл.**

# **Описание алгоритма**

Организация базы данных:

База данных представляет собой массив списков, который проиндексирован буквами английского алфавита. Списки реализованы c помощью структуры. Структура представляет собой узел из фамилии, номера телефона и указателя на следующий узел.

Функция для добавления людей в базу данных:

1. По коду первой буквы фамилии определяется список, в который будет добавлен человек.
2. Создается новый узел для хранения данных о человеке.
3. Новый узел добавляется в соответствующий список на первую позицию.

Функция для поиска человека в базе данных по фамилии:

1. По коду первой буквы фамилии определяется список, в котором будет производиться поиск.
2. Создается узел для хранения данных о человеке.
3. Далее посимвольно сравниваются элементы строки, переданной пользователем в функцию, и элементы соответствующего списка.
4. Возвращается либо информация о человеке, либо NULL.

Функция для удаления человека из базы данных:

1. По коду первой буквы фамилии определяется список, в котором будет производиться поиск.
2. Создается узел для хранения данных о человеке.
3. Далее посимвольно сравниваются элементы строки, переданной пользователем в функцию, и элементы соответствующего списка.
4. В случае совпадения информация о человеке удаляется из базы данных.

Функция для сортировки элементов списка данных по списку:

1. Просматриваются все списки базы данных.
2. Создаются узлы для хранения данных о человеке.
3. Каждый список сортируется по алфавиту с помощью посимвольного сравнения фамилий людей в соответствующем списке.

Функция для сохранения базы данных в файл:

1. Открывается файл для записи данных.
2. Создается узел для хранения данных о человеке.
3. Начиная с первого, данные каждого списка записываются в файл. Каждый следующий человек записывается с новой строки.
4. Закрывается файл.

Функция для загрузки базы данных из файла:

1. Открывается файл для считывания данных.
2. Создается узел для хранения данных о человеке.
3. По коду первой буквы фамилии определяется список, в который будет добавлен человек.
4. Считывается информация о человеке в соответствующий список.
5. Закрывается файл.

# **Текст программы**

#pragma warning(disable: 4996)

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

#define MAX\_SURNAME\_LENGTH 50 //длина фамилии

#define MAX\_PHONE\_LENGTH 11 //длина номера телефона

#define ENGLISH\_ALPHABET 26 //количество списков в бд

//структура, представляющая собой узел

struct People {

char surname[MAX\_SURNAME\_LENGTH];

char phone[MAX\_PHONE\_LENGTH];

struct People\* next;

};

struct PeopleList {

struct People\* head;

};

//массив для хранения базы данных

struct PhoneBook {

struct PeopleList alphabet[ENGLISH\_ALPHABET];

};

//инициализирует базу данных

void initPhoneBook(struct PhoneBook\* phonebook) {

for (int i = 0; i < ENGLISH\_ALPHABET; i++) {

phonebook->alphabet[i].head = NULL;

}

}

//добавляет людей в базу данных

void addPeople(struct PhoneBook\* phonebook, const char\* surname, const char\* phone) {

struct PeopleList\* list;

/\* определяет индекс массива, куда будет записан человек

65 - это код символа 'A' \*/

int index = surname[0] - 65;

list = &phonebook->alphabet[index];

struct People\* newPeople = (struct People\*)malloc(sizeof(struct People));

if (newPeople == NULL){ //проверка успешного выделения памяти

printf("Ошибка выделения памяти");

return;

}

//копирует данные из строки surname и phone в узел

strcpy(newPeople->surname, surname);

strcpy(newPeople->phone, phone);

newPeople->next = list->head;

list->head = newPeople;

}

//ищет в базе данных человека по фамилии

struct People\* findPeople(struct PhoneBook\* phonebook, const char\* surname) {

struct PeopleList\* list;

/\* определяет индекс массива, куда будет записан человек

65 - это код символа 'A' \*/

int index = surname[0] - 65;

list = &phonebook->alphabet[index];

struct People\* current = list->head;

/\* побайтно сравнивает коды символов двух строк и возвращает :

1. 0 – если сравниваемое строки идентичны

2. Положительное число – если строки отличаются и код первого

отличающегося символа в строке current больше кода символа на той же позиции в строке surname

3. Отрицательное число - наоборот (т.е. меньше) \*/

while (current != NULL) {

if (strcmp(current->surname, surname) == 0) {

return current;

}

current = current->next;

}

return NULL;

}

//удаляет из базы данных человека по фамилии

void deletePeople(struct PhoneBook\* phonebook, const char\* surname) {

struct PeopleList\* list;

/\* определяет индекс массива, куда будет записан человек

65 - это код символа 'A' \*/

int index = surname[0] - 65;

list = &phonebook->alphabet[index];

struct People\* current = list->head; //текущий

struct People\* prev = NULL; //предыдущий

while (current != NULL) {

/\* побайтно сравнивает коды символов двух строк и возвращает:

1. 0 – если сравниваемое строки идентичны

2. Положительное число – если строки отличаются и код первого

отличающегося символа в строке current больше кода символа на той же позиции в строке surname

3. Отрицательное число - наоборот (т.е. меньше) \*/

if (strcmp(current->surname, surname) == 0) {

if (prev != NULL) {

prev->next = current->next;

}

else {

list->head = current->next;

}

free(current); //удаляет информацию о человеке

return;

}

prev = current;

current = current->next;

}

}

//сортирует список по алфавиту

void insertPeopleSorted(struct People\*\* sortedList, struct People\* newPeople) {

struct People\* current;

/\* побайтно сравнивает коды символов двух строк и возвращает:

1. 0 – если сравниваемое строки идентичны

2. Положительное число – если строки отличаются и код первого

отличающегося символа в строке current больше кода символа на той же позиции в строке surname

3. Отрицательное число - наоборот (т.е. меньше) \*/

if (\*sortedList == NULL || strcmp(newPeople->surname, (\*sortedList)->surname) <= 0) {

newPeople->next = \*sortedList;

\*sortedList = newPeople;

}

else {

current = \*sortedList;

while (current->next != NULL && strcmp(newPeople->surname, current->next->surname) > 0) {

current = current->next;

}

newPeople->next = current->next;

current->next = newPeople;

}

}

//сортирует каждый список базы данных по алфавиту

void sortPhoneBook(struct PhoneBook\* phonebook) {

struct PeopleList\* list;

for (int i = 0; i < ENGLISH\_ALPHABET; i++) {

list = &phonebook->alphabet[i];

struct People\* current = list->head;

struct People\* sortedList = NULL;

struct People\* next;

while (current != NULL) {

next = current->next;

insertPeopleSorted(&sortedList, current);

current = next;

}

list->head = sortedList;

}

}

//сохраняет базу данных в файл

void savePhoneBook(struct PhoneBook\* phonebook, const char\* filename) {

FILE\* file = fopen(filename, "wb"); //открывает файл для записи данных

if (!file) { //проверка успешного открытия файла

printf("Ошибка при открытии файла");

return;

}

struct PeopleList\* list;

struct People\* current;

for (int i = 0; i < ENGLISH\_ALPHABET; i++) {

list = &phonebook->alphabet[i];

current = list->head;

while (current != NULL) { //записывает в файл данные

fwrite(current, sizeof(struct People), 1, file);

fwrite("\n", sizeof(char), 1, file);

current = current->next;

}

}

fclose(file);

}

//загружает базу данных из файла

void loadPhoneBook(struct PhoneBook\* phonebook, const char\* filename) {

FILE\* file = fopen(filename, "rb"); //открывает файл для чтения

if (!file) { //проверка успешного открытия файла

printf("Ошибка при открытии файла");

exit(1);

}

initPhoneBook(phonebook);

struct People people;

/\* определяет, в какой список (элемент массива) добавить человека

и потом записывает информацию в базу данных \*/

while (fread(&people, sizeof(struct People), 1, file) == 1) {

char first = people.surname[0];

int index = first - 65;

addPeople(&phonebook->alphabet[index], people.surname, people.phone);

}

fclose(file);

}

int main() {

setlocale(LC\_CTYPE, "Russian");

struct PhoneBook phonebook;

initPhoneBook(&phonebook);

//проверка успешной загрузки базы данных из файла

loadPhoneBook(&phonebook, "PhoneBook.txt");

//проверка успешного добавления 5 человек в базу данных

addPeople(&phonebook, "Tishkovets", "89111234567");

addPeople(&phonebook, "Andreev", "89819876543");

addPeople(&phonebook, "Sergeev", "89956752134");

addPeople(&phonebook, "Borisenko", "89218983434");

addPeople(&phonebook, "Drekalov", "89054217889");

//проверка успешного поиска человека по фамилии

struct People\* foundPeople = findPeople(&phonebook, "Sergeev");

if (foundPeople != NULL) {

printf("Найдено: %s %s\n", foundPeople->surname, foundPeople->phone);

}

else printf("Нет совпадений\n");

//проверка успешного удаления человека из базы данных

deletePeople(&phonebook, "Tishkovets");

//проверка успешной сортировки

sortPhoneBook(&phonebook);

//проверка успешного сохранения базы данных в файл

savePhoneBook(&phonebook, "PhoneBook.txt");

return 0;

}

# **Описание тестирования**

Для тестирования данной программы производился ее неоднократный запуск с вводом конкретных данных, при котором проверялось:

1. Стабильность работы программы при одинаковых входных данных;
2. Корректное выполнение всех заявленных процедур;
3. Корректное завершение программы;

Приведем пример тестирования на скриншотах, представленных ниже:

1. Изображение выглядит как текст, снимок экрана, Шрифт

   Автоматически созданное описаниеfindPeople(&phonebook, "Sergeev");
2. Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

   Автоматически созданное описаниеfindPeople(&phonebook, "Stepanov");

При проведении тестирования такого рода никаких проблем обнаружено не было, что позволяет судить о корректности работы программы в целом.