МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ФГБОУ ВО «ВГТУ»)

Факультет информационных технологий и компьютерной безопасности

Кафедра Систем управления и информационных технологий в строительстве

КУРСОВОЙ ПРОЕКТ

по дисциплине Основы программирования и алгоритмизации

Тема: Разработка программы для работы с файловой базой данных «Ноутбуки»

**Расчетно-пояснительная записка**

Разработал студент Е.А. Ледовской

Подпись, дата Инициалы, фамилия

Руководитель Н.В.Акамсина

Подпись, дата Инициалы, фамилия

Нормоконтролер

Подпись, дата Инициалы, фамилия

Защищена Оценка

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ФГБОУ ВО «ВГТУ»)

Кафедра Систем управления и информационных технологий в строительстве

ЗАДАНИЕ

на курсовой проект

по дисциплине: «Основы программирования и алгоритмизации»

Тема: «Разработка программы для работы с файловой базой данных «Ноутбуки»»

Студент бТИИ-241 Ледовской Егор Андреевич

Группа, фамилия, имя, отчество

База данных «Ноутбуки», Признак поиска: процессор, объем оперативной памяти, Вариант сортировки: производитель, операционная система.

Технические условия Windows 10, Microsoft VisualStudio 2022, язык про граммирования C

Содержание и объем проекта (графические работы, расчеты и прочее):

стр, рисунков, таб, приложение

Сроки выполнения этапов анализ и постановка задачи ;

Разработка пошаговой детализации программы ;

Реализация про граммы ;

Тестирование программы ;

Оформление пояснительной записки .

Срок защиты курсового проекта:

Руководитель Н.В.Акамсина

Подпись, дата Инициалы, фамилия

Задание принял студент Е.А.Ледовской

Подпись, дата Инициалы, фамилия

Замечания руководителя

**Содержание**

[Введение 5](#_Toc185964028)

[Постановка задачи 6](#_Toc185964029)

[Реализация программы 8](#_Toc185964030)

[Тестирование 10](#_Toc185964031)

[Заключение 19](#_Toc185964032)

[Список используемых источников 20](#_Toc185964033)

[Приложение 21](#_Toc185964034)

**Введение**

Файловая база – это файл, в котором хранятся упорядоченные записи данных, описывающих заданную предметную область. Для работы с ними необходимо реализовать программу, позволяющую создавать новые записи, изменят существующие, выполнять поиск по заданным критериям.

Целью курсового проекта является разработка программы для реализации файловой базы данных с заданным набором функций.

Для достижения поставленной цели, необходимо решить следующие задачи:

1. Обосновать выбор структуры данных для хранения отдельных записей в файле, а также формат файла.

2. Реализовать простой и понятный интерфейс для взаимодействия пользователя с программой, который будет работать, пока пользователь не захочет выйти из программы.

3. Обеспечить выполнение программой функции создания записей.

**Постановка задачи**

Программа предназначена для работы с записями данных различного типа заданной предметной области «Ноутбуки».

Каждая запись должна состоять из не менее пяти полей различных типов данных (но не менее 3).

Название полей по индивидуальному варианту:

* Производитель;
* Операционная система;
* Диагональ экрана;
* Процессор;
* Объем оперативной памяти.

Программа должна предусматривать выполнение следующих функций:

1. создание новой записи;
2. поиск записи по заданным пользователем значениям полей операционной системы и объема оперативной системы;
3. запись и чтение всех данных из файла;
4. печать всех записей, упорядоченных по производителю или процессору на экран;
5. изменение отдельных записей из ранее сохраненных в файле;
6. добавление произвольного числа новых записей в файл базы данных.

Интерфейс программы должен обеспечивать следующие возможности:

1. выбор одной из функций программы;
2. ввод значений полей для новой или редактируемой записи;
3. вывод результатов выполнения функции программы;
4. вывод диагностических сообщений в ходе проверки корректности ввода данных;
5. пользовательский выбор полей записи для упорядочивания массива структур;
6. задание пользователем значения одного из двух поисковых полей или обоих сразу.

**Реализация программы**

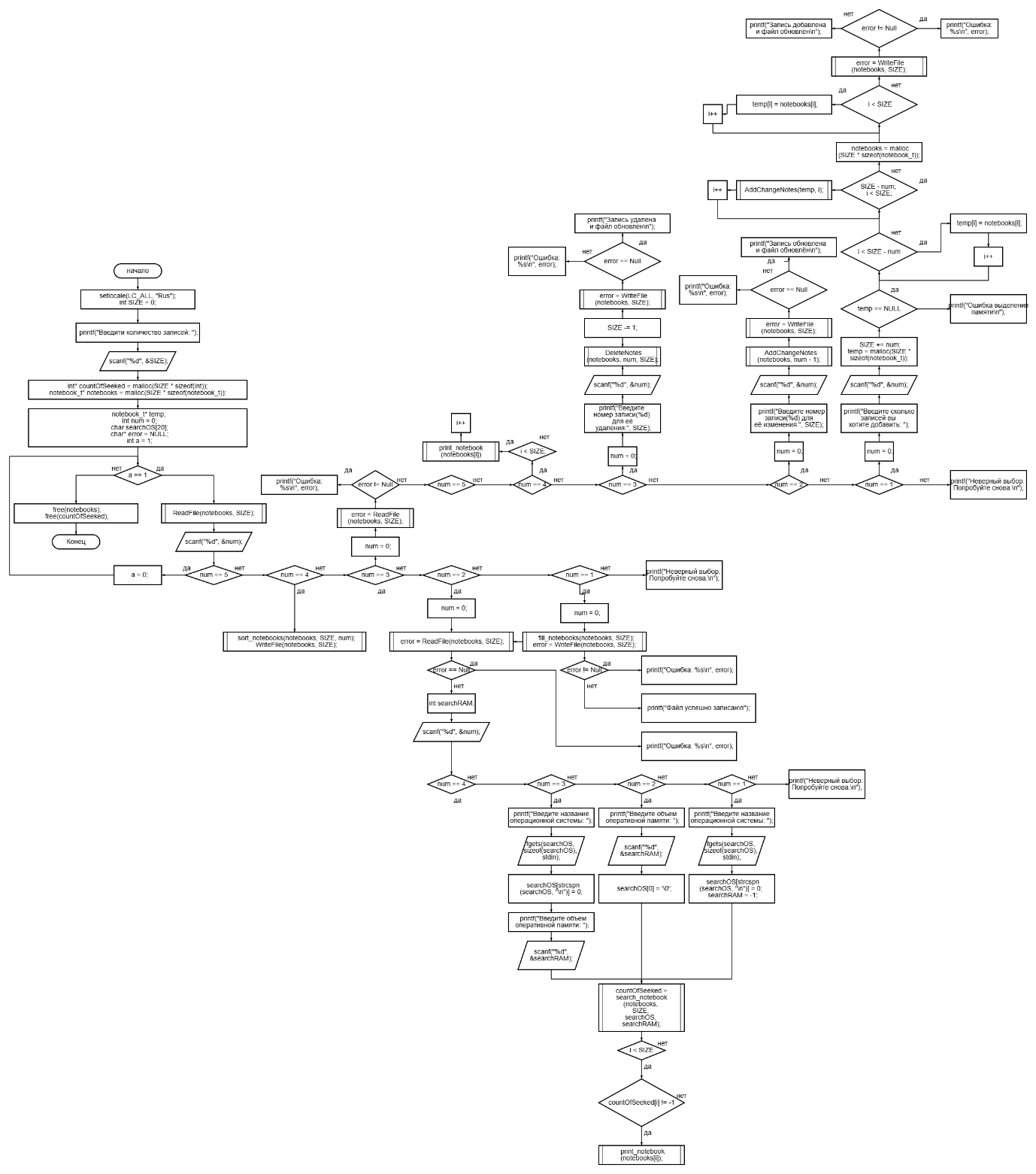
Функция main() – управление всеми действиями и она же является меню общения с пользователем. В ней происходит объявление нужных переменных, диалог с пользователем и вызов других объявленных функций. Блок-схема функции main() представлена на рисунке 1. 

Рисунок 1 - Блок-схема функции main

Функция notebook\_t\* fill\_notebooks(notebook\_t\* notebooks, int size) – заполняет структуру, на которую указывает параметр notebooks данным. Параметр size указывает на количество заполняемых элементов структуры. Блок-схема функции представлена на рисунке 2.

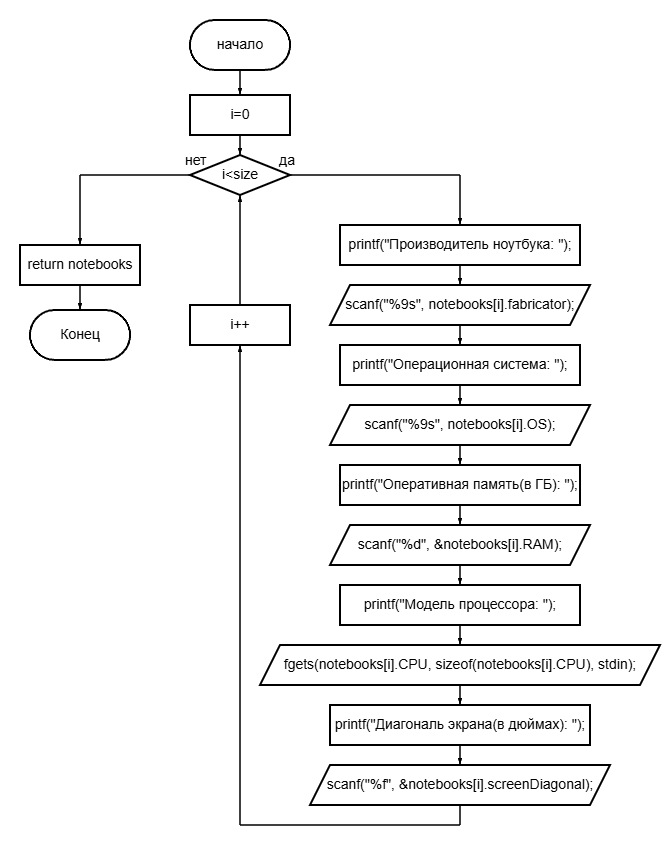


Рисунок 2 - Блок-схема функции fill\_notebooks

Функция void print\_notebook(notebook\_t notebooks) – выводит структуру, на которую указывает параметр notebooks в консоль. Блок-схема функции представлена на рисунке 3.

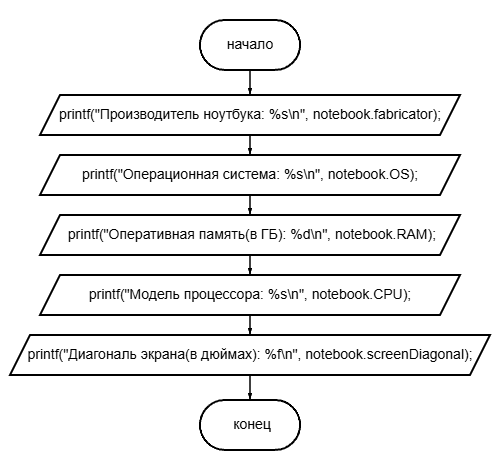


Рисунок 3 – Блок-схема функции print\_notebook

Функция int\* search\_notebook(notebook\_t\* notebooks, int SIZE, char\* CPU, int RAM) – Выполняет поиск по структуре, на которую указывает параметр notebooks. Параметр SIZE указывает на количество элементов этой структуры. Параметры CPU и RAM указывают на значения процессора и объема оперативной памяти, по которым совершается поиск в структуре. Функция возвращает массив чисел, в котором нужные записи структуры помечены значением 0, остальные равны -1. Блок-схема функции представлена на рисунке 4.

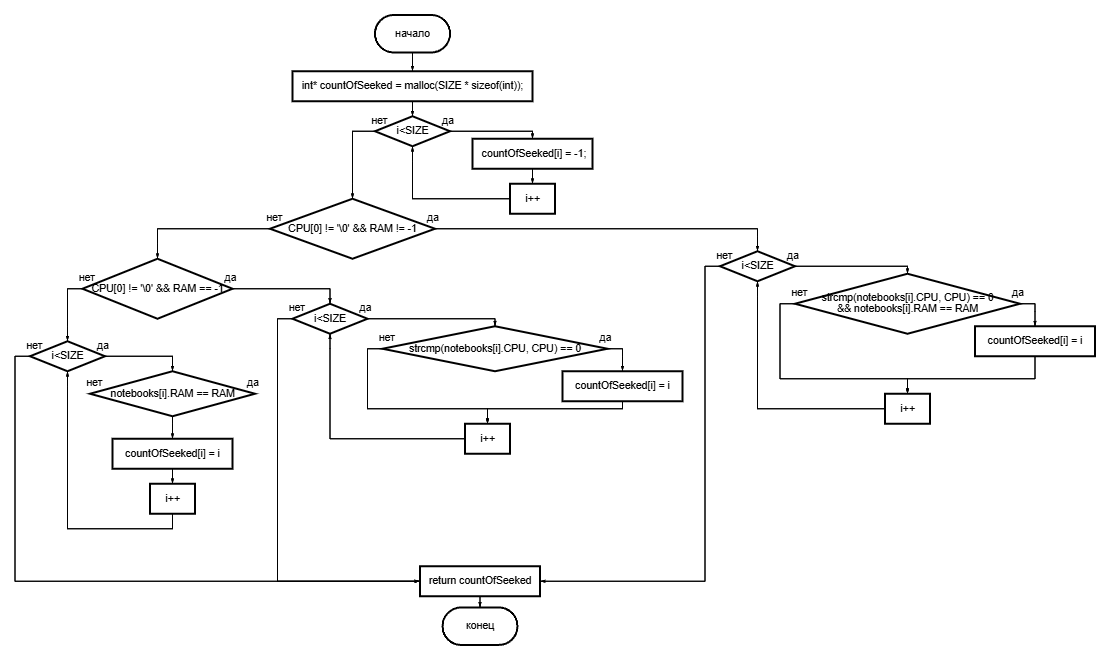


Рисунок 4 – Блок-схема функции search\_notebook

Функция int compare\_fabricator(const void\* a, const void\* b) – сравнивает элементы массива с ключом по производителю и возвращает число в зависимости от того, какой из производителей в алфавитном порядке стоит первый. Блок-схема функции представлена на рисунке 5.

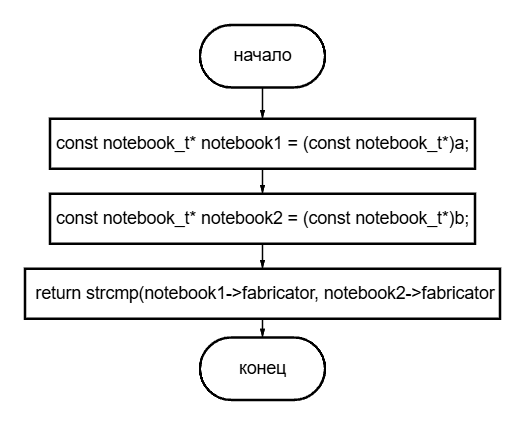


Рисунок 5 – Блок-схема функции compare\_fabricator

Функция int compare\_OS(const void\* a, const void\* b) – сравнивает элементы массива с ключом по операционной системе и возвращает число в зависимости от того, какая из операционных систем в алфавитном порядке стоит первой. Блок-схема функции представлена на рисунке 6.

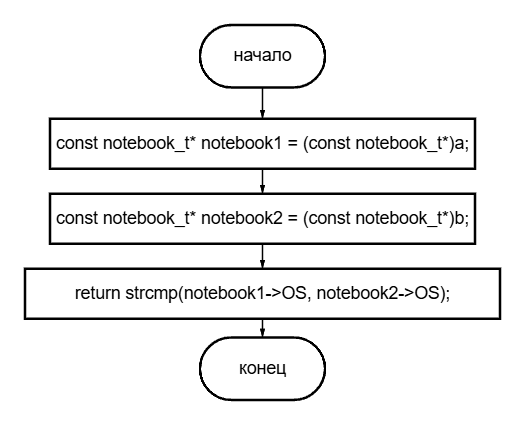


Рисунок 6 – Блок-схема функции compare\_OS

Функция int compare\_fabricator\_and\_OS (const void\* a, const void\* b) – сравнивает элементы массива с ключом по операционной системе и возвращает число в зависимости от того, какая из операционных систем в алфавитном порядке стоит первой. Блок-схема функции представлена на рисунке 7.

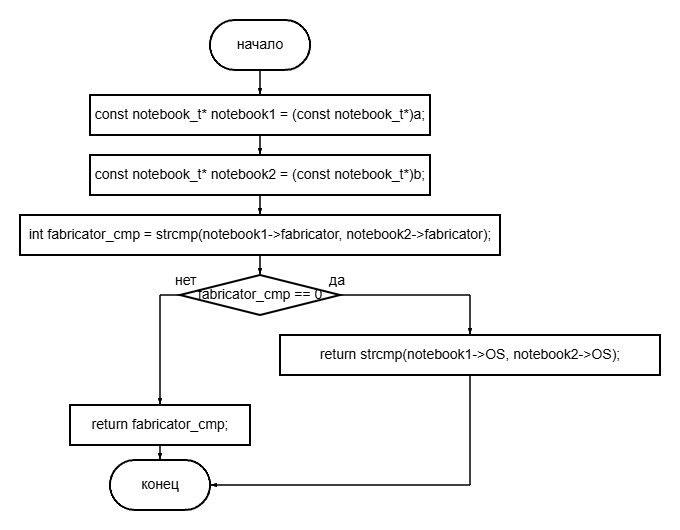


Рисунок 7 - Блок-схема функции compare\_fabricator\_and\_OS

Функция notebook\_t\* sort\_notebooks(notebook \_t\* notebooks, int size, int criteria)– сортирует элементы структуры , на который указывает параметр notebooks , Параметр size указывает на количество элементов структуры, параметр criteria указывает на выбранный пользователем вид сортировки: сортировка по производителю, сортировка по операционной системе. Блок-схема функции представлена на рисунке 8.

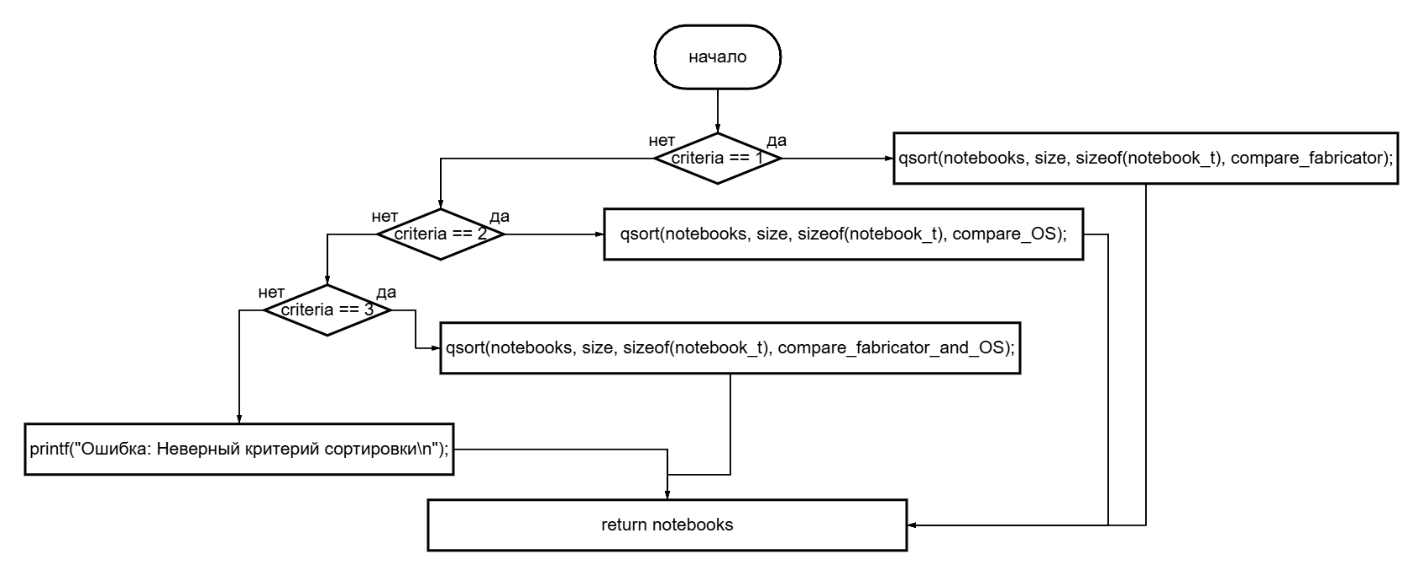
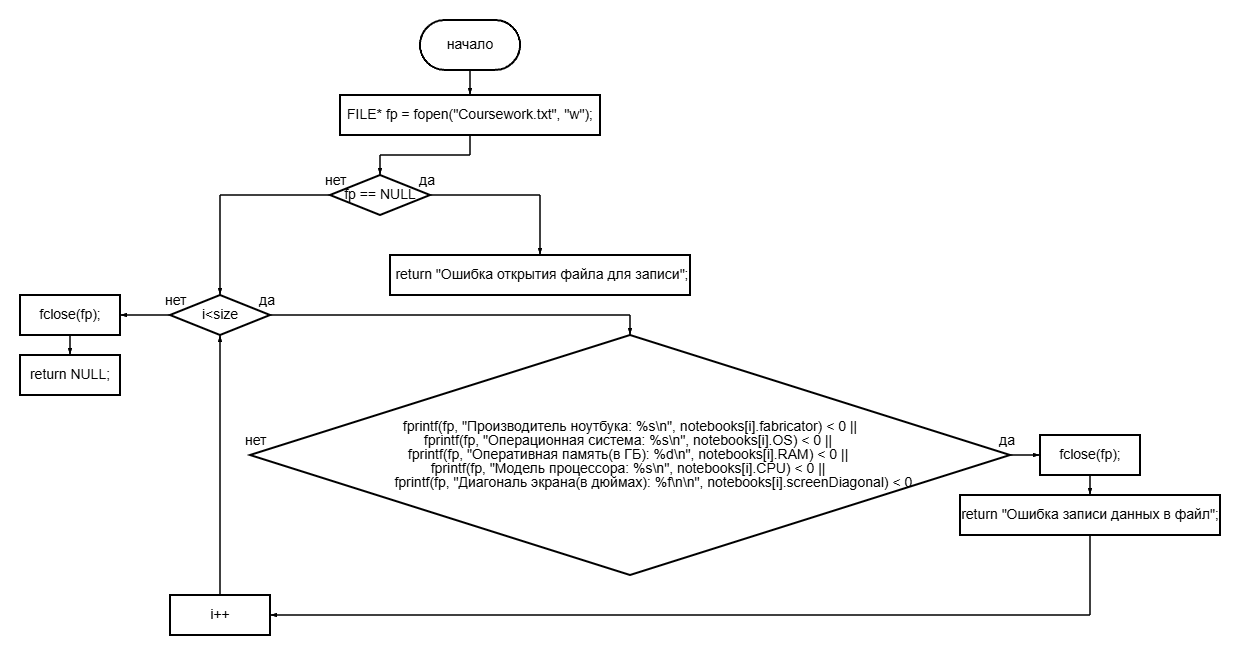


Рисунок 8 – Блок-схема функции sort\_notebooks

Функция char\* WriteFile(notebook \_t\* notebooks, int size)– выполняет запись массива структур из параметра notebooks в файл, параметр size указывает на количество записываемых в файл элементов. Возвращаемое значение задается массивом символов и при возникновении ошибки возвращает ее, иначе Null. Блок-схема функции представлена на рисунке 9 Рисунок 9 – Блок-схема функции WriteFile

Функция char\* ReadFile(notebook\_t\* notebooks, int size)– выполняет чтение из файла текста в массив структур параметра notebooks, параметр size указывает на количество записываемых из файла элементов. Возвращаемое значение задается массивом символов и при возникновении ошибки возвращает ее, иначе Null. Блок-схема функции представлена на рисунке 10.

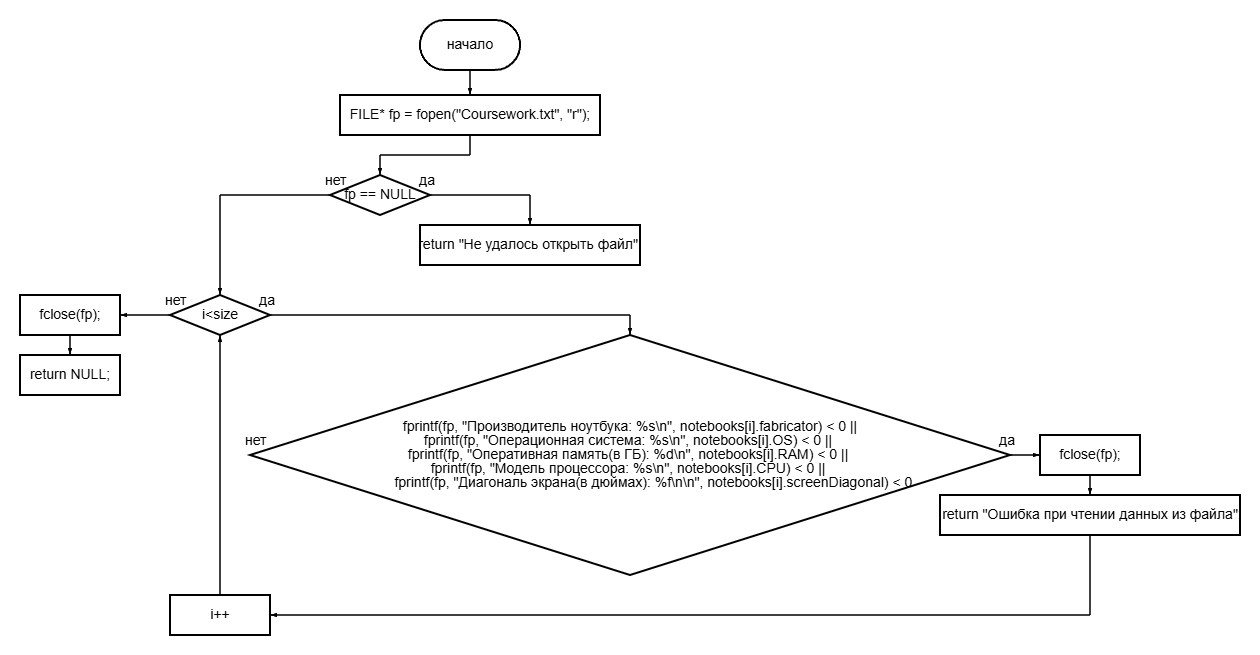


Рисунок 10 – Блок-схема функции ReadFile

Функция notebook\_t\* AddChangeNotes(notebook\_t\* notebooks, int num) – обновляет массив структур, на который указывает параметр notebooks , Параметр num указывает на индекс записи которой хотим обновить. Блок-схема функции представлена на рисунке 11.

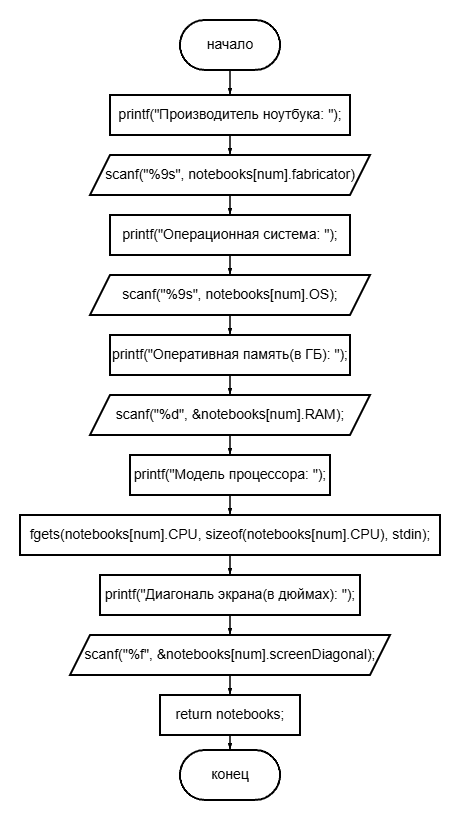


Рисунок 11 – Блок-схема функции AddChangeNotes

Функция notebook\_t\* DeleteNotes(notebook\_t\* notebooks, int num, int SIZE) – удаляет структуру из массива, на который указывает параметр notebooks , Параметр num указывает на индекс записи которую хотим удалить, параметр SIZE указывает на размерность массива структур, которая изменится при удалении. Блок-схема функции представлена на рисунке 12.

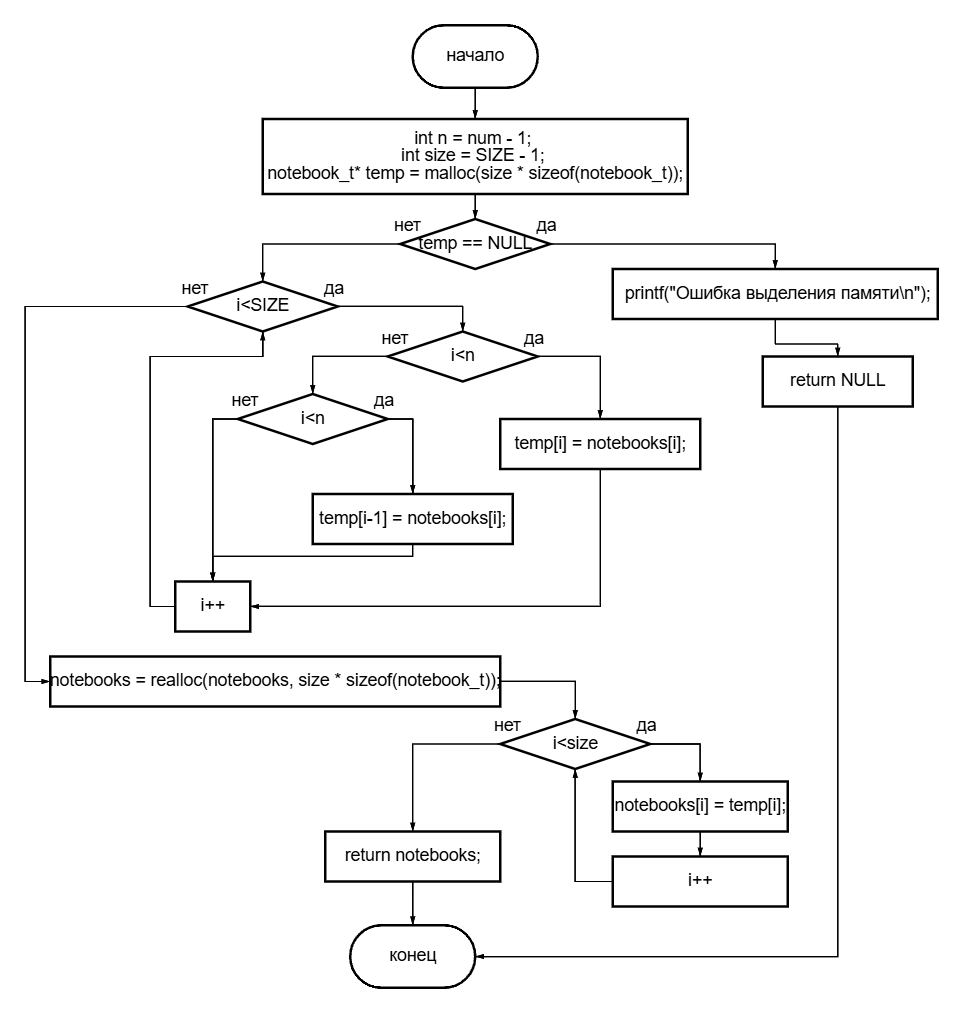


Рисунок 12 – Блок-схема функции DeleteNotes

**Тестирование**

При первом запуске приложения выберем количество записей, которые необходимо создать (рисунок 13).

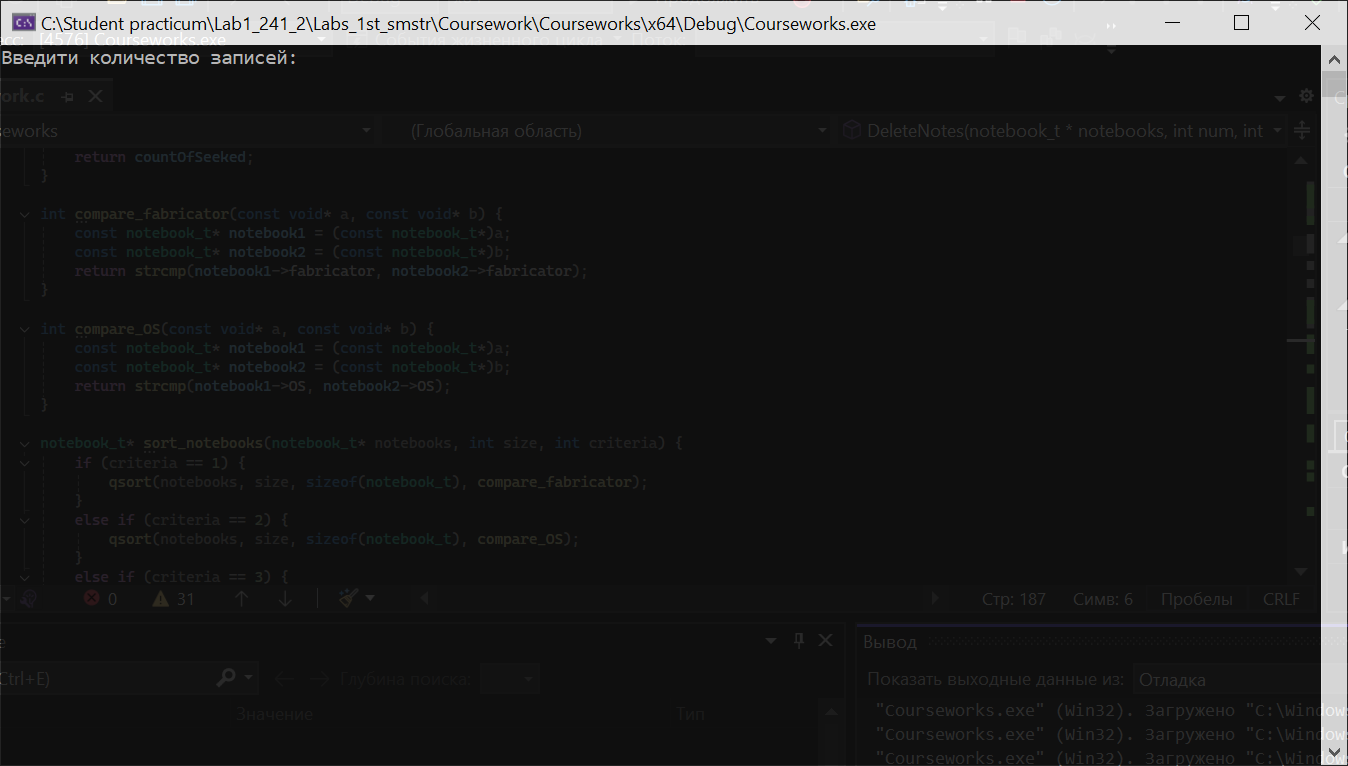


Рисунок 13 – Количество создания записей

Выберем пункт 1 «Создать запись файла» и заполнили файл новыми записями.

Результаты представлены на рисунке 14-15.

Изображение выглядит как текст, снимок экрана, меню, Шрифт

Автоматически созданное описание

Рисунок 14 – Процесс заполнение записей через клавиатуру

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 15 – Результат заполнения Coursework.txt

Выполним добавление новой записи, для этого зайдем в пункт номер 3 «Запись и чтение файла» и выберем пункт 1 «Добавление записи». Выберем какое количество записей хотим добавить. Заполняем с клавиатуры данные о новых записях.

Результаты добавления новых записей представлен на рисунке 16-17.

Изображение выглядит как текст, снимок экрана, меню

Автоматически созданное описание

Рисунок 16 – Процесс добавления новых записей

Изображение выглядит как текст, снимок экрана, Шрифт, меню

Автоматически созданное описание

Рисунок 17 – Результат добавления новых записей в файл coursework.txt.

Выполним поиск по названию операционной системы. Выбираем пункт 2 «Выполнить поиск по файлу». Выбираем пункт 1 «Процессору». Вводим название процессора и получаем результат.

Процесс выполнения поиска по процессору представлен на рисунке 18.

Изображение выглядит как текст, снимок экрана, меню, Шрифт

Автоматически созданное описание

Рисунок 18 – Процесс поиска по процессору

Выполним аналогичным способом поиск по другим критериям. Результаты представлены на рисунках 19–20.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 19 – Процесс поиска по оперативной памяти

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 20 – Процесс комбинированного поиска

Для обновления записи номер 3, представленной на рисунке 17, выбираем пункт 3 «Запись и чтение файла», затем пункт 2 «Изменить запись», далее номер записи для изменения и изменяем необходимую запись. Результаты представлены на рисунках 21-22.

Изображение выглядит как текст, снимок экрана, черно-белый, монохромный

Автоматически созданное описание

Рисунок 21 – Процесс обновления записи

Изображение выглядит как текст, снимок экрана, Шрифт, меню

Автоматически созданное описание

Рисунок 22 – Результат обновления записи файла Coursework.txt

Выполним удаление записи номер 1, представленной на рисунке 22. Выбираем пункт 3 «Запись и чтение файла», далее выбираем пункт 3 «Удалить запись», затем номер записи для удаления. Результаты представлены на рисунках 23-24.

**Изображение выглядит как текст, снимок экрана, меню, Шрифт

Автоматически созданное описание**

Рисунок 23 – Процесс удаления записи

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 25 – Результат удаления записи в файле Coursework.txt

Выполним чтение записей. Выбираем пункт 3 «Запись и чтение файла», далее выбираем пункт 4 «Чтение файла». Результаты представлены на рисунке 26.

Изображение выглядит как текст, снимок экрана, меню, черно-белый

Автоматически созданное описание

Рисунок 26 – Результат чтения данных

Выполним сортировку записей. Перед этим изменим записи в файле, чтобы было видно результат сортировки (Рис.27). Выбираем пункт 4 «Сортировать данные файла», далее выбираем какой из способов сортировки применить: по производителю или операционной системе или по обеим критериям сразу(приоритетно по производителю). Результаты представлены на рисунках 28-30.

Изображение выглядит как текст, снимок экрана, меню, Шрифт

Автоматически созданное описание

Рисунок 27 – Не отсортированные данные в файле Coursework.txt

Изображение выглядит как текст, снимок экрана, меню, Шрифт

Автоматически созданное описание

Рисунок 28 – Результат сортировки данных по производителю в файле Coursework.txt

Изображение выглядит как текст, снимок экрана, меню, Шрифт

Автоматически созданное описание

Рисунок 29 – Результат сортировки данных по операционной системе в файлe Coursework.txt

Изображение выглядит как текст, снимок экрана, меню, Шрифт

Автоматически созданное описание

Рисунок 30 – Результат сортировки данных по обеим критериям сразу в файле Coursework.txt

При неправильном указании количества записей может происходить ошибка при удалении записей, связанной с невозможностью выделения памяти (Рисунок 31).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 31 – Ошибка выделения памяти

При неудачном чтении из файла или открытии файла программа выводит ошибку на экран представленной на рисунках 32-33.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 32 – Ошибка чтения из файла

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 33 – Ошибка открытия файла

При неправильном выборе в меню и подменю на экран выводится ошибка о неправильном выборе действия (рисунок 34).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 34 – Неправильно выбранное действие

**Заключение**

В ходе выполнения курсовой работы была разработана программа, предназначенная для работы с записями данных предметной области «Ноутбуки». Программа успешно реализует функции создания, хранения, редактирования и замены записей, что позволяет эффективно управлять файловой базой данных.

Основной функционал программы включает в себя:

Создание новых записей с заполнением всех обязательных полей.

Поиск записей по значениям полей излучателей и объема оперативной памяти, что позволяет быстро находить необходимую информацию.

Запись данных в файл и их последующее чтение обеспечивают долговременное хранение информации.

Вывод всех записей на экран приводит к удостоверению производителю или процессору для удобного анализа данных.

Редактирование существующих записей и добавление новых записей в базы данных.

Интерфейс программы разработан с учетом удобства использования. Он предоставляет возможность выбора функций, ввода данных для новых и редактируемых записей, просмотра результатов работы программы, а также информирует пользователя о возможных ошибках ввода данных. Реализована гибкость в работе с записями: пользователь может выбирать критерии сортировки и выполнять поиск по одному или обоим полям одновременно.

В результате выполнения работы создано простое и функциональное приложение, способное эффективно решать поставленные задачи. Программа соответствует заявленным требованиям и может быть использована для работы с данными в заданной предметной области.

Ссылка: https://github.com/FalsFord

**Список используемых источников**

1. Брайан Керниган, Денис Ричи Язык программирования С. - Вильямс, 2007.

2. Стивен Прата С. Программирование для профессионалов. - Питер, 2015.

3. Майк Микальчик Как стать профессиональным программистом на C . - БХВ-Петербург, 2016.

4. Кеннет Н. Кинг, Современный язык программирования C. - Вильямс, 2007.

5. П. Дж. Плажер С. Библиотека Стандартных Функций. - Питер, 1992.

6. Скотт Чакон Git. Управление версиями. - Вильямс, 2014.

7. Скотт Чакон, Бен Страуб Pro Git. - Вильямс, 2015, перевод Галина Панкратова.

8. Андрей Малашенко Гит для профессионалов. - БХВ-Петербург, 2017.

9. Аарон Мартин Git. Интуитивное руководство. - Манн, Иванов и Фербер, 2015

10. Джон Лоэлигер Версионный контроль с Git. - Питер, 2018

**Приложение**

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

typedef struct Notebook {

char fabricator[10];

char OS[10];

int RAM;

char CPU[20];

float screenDiagonal;

} notebook\_t;

notebook\_t\* fill\_notebooks(notebook\_t\* notebooks, int size) {

for (int i = 0; i < size; i++) {

printf("Производитель ноутбука: ");

scanf("%9s", notebooks[i].fabricator);

getchar();

printf("Операционная система: ");

scanf("%9s", notebooks[i].OS);

getchar();

printf("Оперативная память(в ГБ): ");

scanf("%d", &notebooks[i].RAM);

getchar();

printf("Модель процессора: ");

fgets(notebooks[i].CPU, sizeof(notebooks[i].CPU), stdin);

notebooks[i].CPU[strcspn(notebooks[i].CPU, "\n")] = 0; // Удаляем символ новой строки

printf("Диагональ экрана(в дюймах): ");

scanf("%f", &notebooks[i].screenDiagonal);

getchar();

printf("\n");

}

return notebooks;

}

void print\_notebook(notebook\_t notebook) {

printf("Производитель ноутбука: %s\n", notebook.fabricator);

printf("Операционная система: %s\n", notebook.OS);

printf("Оперативная память(в ГБ): %d\n", notebook.RAM);

printf("Модель процессора: %s\n", notebook.CPU);

printf("Диагональ экрана(в дюймах): %f\n", notebook.screenDiagonal);

printf("\n");

}

int\* search\_notebook(notebook\_t\* notebooks, int SIZE, char\* CPU, int RAM) {

int\* countOfSeeked = malloc(SIZE \* sizeof(int));

for (int i = 0; i < SIZE; i++) {

countOfSeeked[i] = -1;

}

if (CPU[0] != '\0' && RAM != -1) {

for (int i = 0; i < SIZE; i++) {

if (strcmp(notebooks[i].CPU, CPU) == 0 && notebooks[i].RAM == RAM) countOfSeeked[i] = i;

}

}

else if (CPU[0] != '\0' && RAM == -1) {

for (int i = 0; i < SIZE; i++) {

if (strcmp(notebooks[i].CPU, CPU) == 0) countOfSeeked[i] = i;

}

}

else {

for (int i = 0; i < SIZE; i++) {

if (notebooks[i].RAM == RAM) countOfSeeked[i] = i;

}

}

return countOfSeeked;

}

int compare\_fabricator(const void\* a, const void\* b) {

const notebook\_t\* notebook1 = (const notebook\_t\*)a;

const notebook\_t\* notebook2 = (const notebook\_t\*)b;

return strcmp(notebook1->fabricator, notebook2->fabricator);

}

int compare\_OS(const void\* a, const void\* b) {

const notebook\_t\* notebook1 = (const notebook\_t\*)a;

const notebook\_t\* notebook2 = (const notebook\_t\*)b;

return strcmp(notebook1->OS, notebook2->OS);

}

int compare\_fabricator\_and\_OS(const void\* a, const void\* b) {

const notebook\_t\* notebook1 = (const notebook\_t\*)a;

const notebook\_t\* notebook2 = (const notebook\_t\*)b;

int fabricator\_cmp = strcmp(notebook1->fabricator, notebook2->fabricator);

if (fabricator\_cmp == 0) {

return strcmp(notebook1->OS, notebook2->OS);

}

return fabricator\_cmp;

}

notebook\_t\* sort\_notebooks(notebook\_t\* notebooks, int size, int criteria) {

if (criteria == 1) {

qsort(notebooks, size, sizeof(notebook\_t), compare\_fabricator);

}

else if (criteria == 2) {

qsort(notebooks, size, sizeof(notebook\_t), compare\_OS);

}

else if (criteria == 3) {

qsort(notebooks, size, sizeof(notebook\_t), compare\_fabricator\_and\_OS);

}

else {

printf("Ошибка: Неверный критерий сортировки\n");

}

return notebooks;

}

char\* WriteFile(notebook\_t\* notebooks, int size) {

FILE\* fp = fopen("Coursework.txt", "w");

if (fp == NULL) {

return "Ошибка открытия файла для записи";

}

for (int i = 0; i < size; i++) {

if (fprintf(fp, "Производитель ноутбука: %s\n", notebooks[i].fabricator) < 0 ||

fprintf(fp, "Операционная система: %s\n", notebooks[i].OS) < 0 ||

fprintf(fp, "Оперативная память(в ГБ): %d\n", notebooks[i].RAM) < 0 ||

fprintf(fp, "Модель процессора: %s\n", notebooks[i].CPU) < 0 ||

fprintf(fp, "Диагональ экрана(в дюймах): %f\n\n", notebooks[i].screenDiagonal) < 0) {

fclose(fp);

return "Ошибка записи данных в файл";

}

}

fclose(fp);

return NULL;

}

char\* ReadFile(notebook\_t\* notebooks, int size) {

FILE\* fp = fopen("Coursework.txt", "r");

if (fp == NULL) {

return "Не удалось открыть файл";

}

for (int i = 0; i < size; i++) {

if (fscanf(fp, "Производитель ноутбука: %9s\n", notebooks[i].fabricator) != 1 ||

fscanf(fp, "Операционная система: %9s\n", notebooks[i].OS) != 1 ||

fscanf(fp, "Оперативная память(в ГБ): %d\n", &notebooks[i].RAM) != 1 ||

fscanf(fp, "Модель процессора: %19[^\n]\n", notebooks[i].CPU) != 1 ||

fscanf(fp, "Диагональ экрана(в дюймах): %f\n\n", &notebooks[i].screenDiagonal) != 1) {

fclose(fp);

return "Ошибка при чтении данных из файла";

}

}

fclose(fp);

return NULL;

}

notebook\_t\* AddChangeNotes(notebook\_t\* notebooks, int num) {

printf("Производитель ноутбука: ");

scanf("%9s", notebooks[num].fabricator);

getchar();

printf("Операционная система: ");

scanf("%9s", notebooks[num].OS);

getchar();

printf("Оперативная память(в ГБ): ");

scanf("%d", &notebooks[num].RAM);

getchar();

printf("Модель процессора: ");

fgets(notebooks[num].CPU, sizeof(notebooks[num].CPU), stdin);

notebooks[num].CPU[strcspn(notebooks[num].CPU, "\n")] = 0;

printf("Диагональ экрана(в дюймах): ");

scanf("%f", &notebooks[num].screenDiagonal);

getchar();

printf("\n");

return notebooks;

}

notebook\_t\* DeleteNotes(notebook\_t\* notebooks, int num, int SIZE)

{

int n = num - 1;

int size = SIZE - 1;

notebook\_t\* temp = malloc(size \* sizeof(notebook\_t));

if (temp == NULL) {

printf("Ошибка выделения памяти\n");

return;

}

for (int i = 0; i < SIZE; i++) {

if (i < n) {

temp[i] = notebooks[i];

}

else if (i > n) {

temp[i - 1] = notebooks[i];

}

}

notebooks = realloc(notebooks, size \* sizeof(notebook\_t));

for (int i = 0; i < size; i++) {

notebooks[i] = temp[i];

}

return notebooks;

}

void main() {

setlocale(LC\_ALL, "Rus");

int SIZE = 0;

printf("Введити количество записей: ");

scanf("%d", &SIZE);

int\* countOfSeeked = malloc(SIZE \* sizeof(int));

notebook\_t\* notebooks = malloc(SIZE \* sizeof(notebook\_t));

notebook\_t\* temp;

int num = 0;

char searchProc[20];

char\* error = NULL;

int a = 1;

while (a) {

ReadFile(notebooks, SIZE);

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

printf("\* Menu \*\n");

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

printf("\* 1. Создать запись файла. \*\n");

printf("\* 2. Выполнить поиск по файлу. \*\n");

printf("\* 3. Запись и чтение файла. \*\n");

printf("\* 4. Сортировать данные файла. \*\n");

printf("\* 5. Завершить работу \*\n");

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

scanf("%d", &num);

switch (num) {

case 1:

num = 0;

fill\_notebooks(notebooks, SIZE);

error = WriteFile(notebooks, SIZE);

if (error) {

printf("Ошибка: %s\n", error);

}

else {

printf("Файл успешно записан\n");

}

break;

case 2:

num = 0;

int searchRAM;

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

printf("\* Поиск по \*\n");

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

printf("\* 1. Процессору. \*\n");

printf("\* 2. Оперативной памяти. \*\n");

printf("\* 3. Процессору и оперативной памяти. \*\n");

printf("\* 4. Вернуться в меню. \*\n");

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

scanf("%d", &num);

switch (num) {

case 1:

printf("Введите название процессора: ");

getchar();

fgets(searchProc, sizeof(searchProc), stdin);

searchProc[strcspn(searchProc, "\n")] = 0; // Удаляем символ новой строки

searchRAM = -1;

break;

case 2:

printf("Введите объем оперативной памяти: ");

scanf("%d", &searchRAM);

searchProc[0] = '\0';

break;

case 3:

getchar();

printf("Введите название процессора: ");

fgets(searchProc, sizeof(searchProc), stdin);

searchProc[strcspn(searchProc, "\n")] = 0; // Удаляем символ новой строки

printf("Введите объем оперативной памяти: ");

scanf("%d", &searchRAM);

break;

case 4:

break;

}

countOfSeeked = search\_notebook(notebooks, SIZE, searchProc, searchRAM);

for (int i = 0; i < SIZE; i++) {

if (countOfSeeked[i] != -1) print\_notebook(notebooks[i]);

}

break;

case 3:

num = 0;

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

printf("\* Запись и чтение файла \*\n");

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

printf("\* 1. Добавить запись. \*\n");

printf("\* 2. Изменить записи. \*\n");

printf("\* 3. Удалить записи. \*\n");

printf("\* 4. Чтение файла. \*\n");

printf("\* 5. Вернуться в меню. \*\n");

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

scanf("%d", &num);

switch (num) {

case 1:

error = ReadFile(notebooks, SIZE);

if (error) {

printf("Ошибка: %s\n", error);

break;

}

printf("Введите сколько записей вы хотите добавить: ");

num = 0;

scanf("%d", &num);

SIZE += num;

temp = realloc(notebooks, SIZE \* sizeof(notebook\_t));

if (temp == NULL)

{

printf("Ошибка выделения памяти\n");

continue;

}

for (int i = 0; i < SIZE - num; i++) {

temp[i] = notebooks[i];

}

for (int i = SIZE - num; i < SIZE; i++)

{

AddChangeNotes(temp, i);

}

notebooks = malloc(SIZE \* sizeof(notebook\_t));

for (int i = 0; i < SIZE; i++) {

notebooks[i] = temp[i];

}

error = WriteFile(notebooks, SIZE);

if (error) {

printf("Ошибка: %s\n", error);

}

else {

printf("Запись добавлена и файл обновлён\n");

}

break;

case 2:

num = 0;

printf("Введите номер записи(%d) для её изменения:", SIZE);

scanf("%d", &num);

AddChangeNotes(notebooks, num - 1);

error = WriteFile(notebooks, SIZE);

if (error) {

printf("Ошибка: %s\n", error);

}

else {

printf("Запись обновлена и файл обновлён\n");

}

break;

case 3:

num = 0;

printf("Введите номер записи(%d) для её удаления:", SIZE);

scanf("%d", &num);

DeleteNotes(notebooks, num, SIZE);

SIZE -= 1;

error = WriteFile(notebooks, SIZE);

if (error) {

printf("Ошибка: %s\n", error);

}

else {

printf("Запись удалена и файл обновлён\n");

}

break;

case 4:

error = ReadFile(notebooks, SIZE);

if (error) {

printf("Ошибка: %s\n", error);

}

else {

printf("Файл успешно прочитан\n");

printf("\n");

}

for (int i = 0; i < SIZE; i++) print\_notebook(notebooks[i]);

break;

case 5:

break;

}

break;

case 4:

num = 0;

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

printf("\* Сортировать по \*\n");

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

printf("\* 1. Производителю. \*\n");

printf("\* 2. Операционной системе. \*\n");

printf("\* 3. Производителю и операционной системе. \*\n");

printf("\* 4. Вернуться в меню. \*\n");

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

scanf("%d", &num);

ReadFile(notebooks, SIZE);

switch (num) {

case 1:

sort\_notebooks(notebooks, SIZE, num);

WriteFile(notebooks, SIZE);

break;

case 2:

sort\_notebooks(notebooks, SIZE, num);

WriteFile(notebooks, SIZE);

break;

case 3:

sort\_notebooks(notebooks, SIZE, num);

WriteFile(notebooks, SIZE);

break;

case 4:

break;

}

break;

case 5:

a = 0;

break;

default:

printf("Неверный выбор. Попробуйте снова.\n");

}

}

free(notebooks);

free(countOfSeeked);

}