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 «Жесткий диск».

**Расчетно-пояснительная записка**

Разработал студент А.В. Гречишникова

Подпись, дата Инициалы, фамилия

Руководитель Н.В. Акамсина

Подпись, дата Инициалы, фамилия

Нормоконтролер Н.В. Акамсина

Подпись, дата Инициалы, фамилия

Защищена Оценка

дата

2024

Министерство науки и высшего образования

Российской федерации

Федеральное государственное бюджетное образова-тельное учреждение высшего образования «воронеж-

ский 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 «Жесткий диск».

Студентка группы бТИИ-241 Гречишникова Алёна Васильевна

Фамилия, имя, отчество

База данных «Жесткий диск», Признак: производитель, объем, скорость вращения, интерфейс, объем буфера (кеша), скорость обмена данных Вариант сортировки: производитель и объем

Технические условия Windows 10, MicrosoftVisualStudio2022, язык программирования С

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

Сроки выполнения этапов анализ и постановка задачи (10.9-5.10.24); разработка пошаговой детализации программы (6.10 -11.11.24); реализация программы (11.11-5.12.24); тестирование программы (6.12-11.12.24); оформление пояснительной записки (11.12-14.12.24).

Срок защиты курсового проекта

Руководитель Н. В. Акамсина

Подпись, дата Инициалы, фамилия

Задание принял студент А.В. Гречишникова

Подпись, дата Инициалы, фамилия

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc187860305)

[1 ПОСТАНОВКА ЗАДАЧИ 5](#_Toc187860306)

[2 РЕАЛИЗАЦИЯ ПРОГРАММЫ 8](#_Toc187860307)

[3 ТЕСТИРОВАНИЕ ПРОГРАММЫ 15](#_Toc187860308)

[ЗАКЛЮЧЕНИЕ 20](#_Toc187860309)

[СПИСОК ЛИТЕРАТУРЫ 21](#_Toc187860310)

[ПРИЛОЖЕНИЕ А 22](#_Toc187860311)

[ПРИЛОЖЕНИЕ В 23](#_Toc187860312)

# 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, изменять существующие, выполнять поиск по заданным критериям.

Целью данной работы является разработка программы для работы с файловой базой данных «Жесткий диск».

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. Привести блок-схемы используемых алгоритмов для понимания работы данной программы.
4. Обеспечить выполнение программой функции создания записи.
5. Обеспечить выполнение программой функции добавления произвольного количества записей в текущую базу данных.
6. Обеспечить возможность выполнения сортировки текущей базы данных по различным полям в порядке возрастания и убывания.
7. Обезопасить выполнение программы от ошибок при вводе данных пользователем.

# 1 ПОСТАНОВКА ЗАДАЧИ

Задача курсового проекта – написание программы для работы с файловой базой данных «Жесткий диск», содержащей записи из полей различного типа данных. Данная программа должна предусматривать выполнение хранения, поиска, добавления и сортировки данных.

Запись в базе данных представляют собой структуру «Жесткий диск», содержащую данные о производителе, объеме, скорости вращения, интерфейсе, объеме буфера (кеша), скорости обмена данных:

1. Производитель – данные этого поля будут храниться в массиве przv[10] типа char. Массив может содержать строку размером до 9 символов включительно, что является достаточным для многих наименований.
2. Объем – данные этого поля будут храниться в переменной space типа float, что позволит отображать в записи более точный объем.
3. Скорость вращения – переменная sv типа int; данный тип наиболее предпочтителен, поскольку скорость часто представляется целым значением оборотов в минуту.
4. Интерфейс – для хранения данных используется массив intrfs[10] типа char. Поскольку большинство названий для интерфейса являются аббревиатурами (USB, ATA, SATA, SCSI), то 9 символов будет достаточно.
5. Объем буфера – данные этого поля хранятся в переменной sp\_b типа float. Данный тип позволит отображать объем более точно.
6. Скорость обмена данных – для хранения будет использоваться переменная s\_data типа int (количество оборотов в минуту).

Для хранения и использования вышеперечисленных полей, имеющих различные типы, удобно объединить их в структуру данных. В файле massiv1.txt будет располагаться исходная база данных, в которой каждая строка представляет собой совокупность перечисленных полей для каждого жесткого диска. Пример записи в базе данных: «Производитель: Seagate Объем: 10 Скорость вращения: 7200 Интерфейс: USB Объем буфера: 8 Скорость обмена данных: 700».

Текстовый файл massiv1.txt содержит 10 исходных записей, что является достаточным для выполнения сортировки и поиска, а также проверки корректности работы созданной программы:

Производитель: Seagate Объем: 10,000000 Скорость вращения: 7200 Интерфейс: USB Объем буфера: 8,000000 Скорость обмена данных: 700

Производитель: Toshiba Объем: 7,000000 Скорость вращения: 4200 Интерфейс: USB Объем буфера: 16,000000 Скорость обмена данных: 300

Производитель: Sony Объем: 3,000000 Скорость вращения: 5400 Интерфейс: ATA Объем буфера: 2,000000 Скорость обмена данных: 200

Производитель: Samsung Объем: 2,000000 Скорость вращения: 4200 Интерфейс: SAS Объем буфера: 32,000000 Скорость обмена данных: 300

Производитель: Adata Объем: 5,000000 Скорость вращения: 4200 Интерфейс: IDE Объем буфера: 2,000000 Скорость обмена данных: 600

Производитель: WD Объем: 4,000000 Скорость вращения: 5400 Интерфейс: USB Объем буфера: 8,0 Скорость обмена данных: 200

Производитель: Hitachi Объем: 1,000000 Скорость вращения: 7200 Интерфейс: eSATA Объем буфера: 16,000000 Скорость обмена данных: 600

Производитель: Quantum Объем: 1,000000 Скорость вращения: 5400 Интерфейс: PATA Объем буфера: 8,000000 Скорость обмена данных: 300

Производитель: Intel Объем: 3,000000 Скорость вращения: 4200 Интерфейс: SCSI Объем буфера: 16,000000 Скорость обмена данных: 500

Производитель: Lenovo Объем: 4,000000 Скорость вращения: 4200 Интерфейс: USB Объем буфера: 2,000000 Скорость обмена данных: 300

Разработанная программа должна выполнять следующие операции с базой данных:

1. Создание исходной базы данных (ввод данных пользователем или использование готового массива);
2. Запись базы данных в текстовый файл
3. Открытие файла, содержащего готовую базу данных;
4. Редактирование записи;
5. Добавление записей в готовую базу;
6. Поиск записи пользователем (должен осуществляться по полям «Объем» и/или «Интерфейс»);
7. Сортировка базы данных (выполняется в любом порядке по одному или нескольким критериям – производитель и/или объем;
8. Печать базы данных.

Для выполнения вышеописанных операций с базой данных необходимо реализовать взаимодействие с пользователем; его алгоритм описан блок-схемой (рисунок 1).

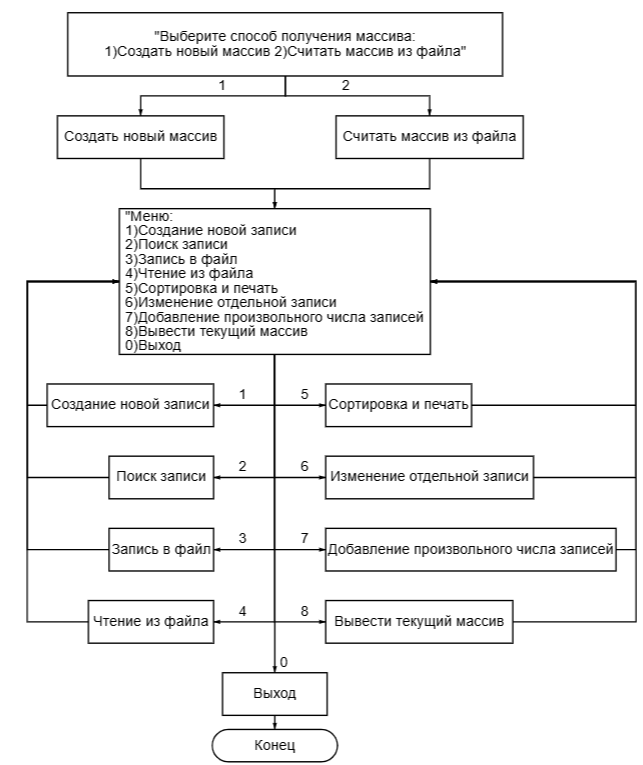


Рисунок 1 – Блок-схема взаимодействия с пользователем

# 2 РЕАЛИЗАЦИЯ ПРОГРАММЫ

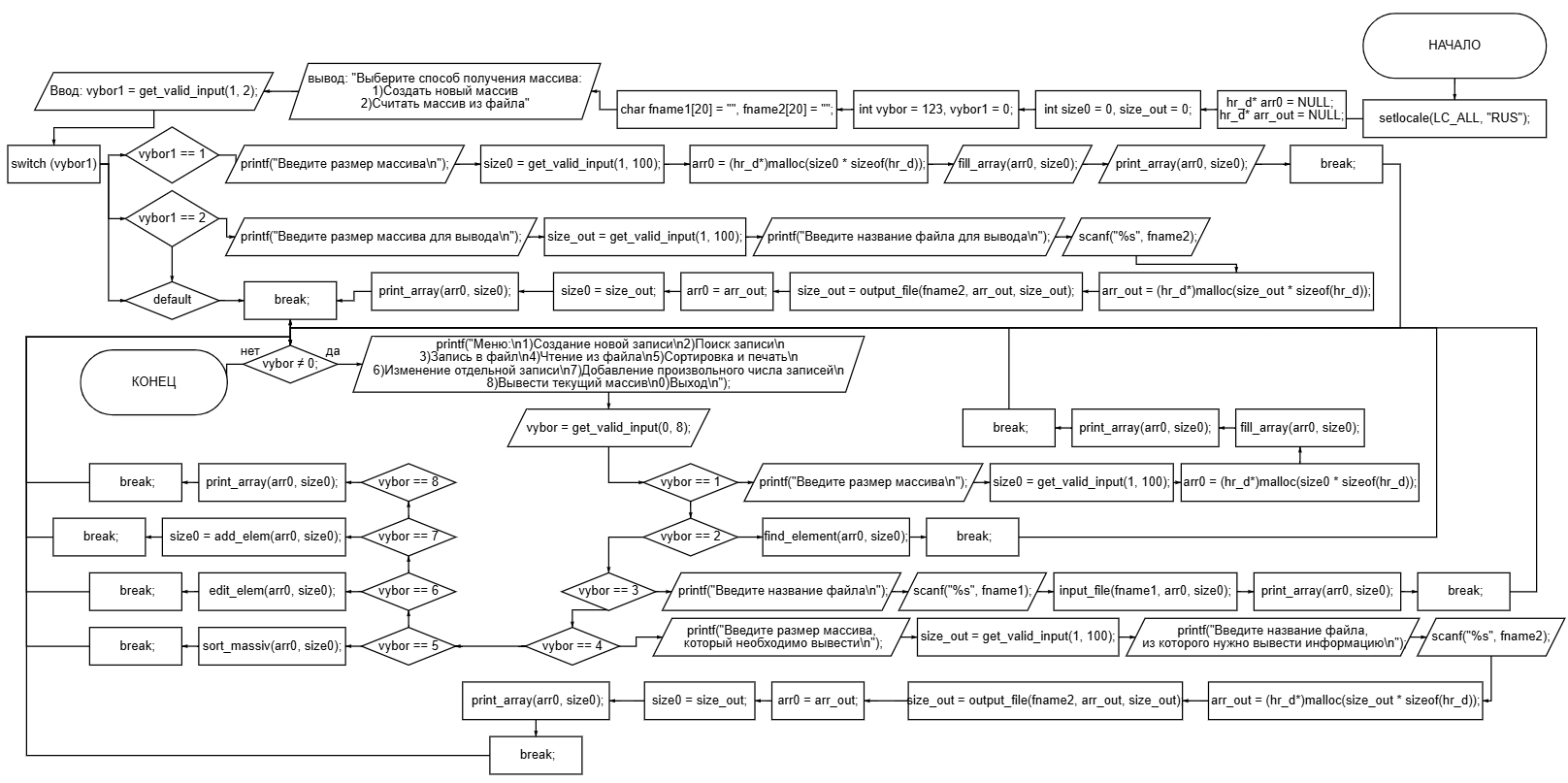
****

Рисунок 2 – Блок-схема функции main

На рисунке 2 приведена блок-схема функции main, которая включает общую информацию о программе и вызов функций.

Функция int fill\_array(hr\_d\* arr, int size) предназначена для заполнения массива значениями, которые вводит пользователь с клавиатуры. Функция принимает указатель на данный массив структурного типа и целочисленный размер массива, возвращает размер массива. Блок-схема функции fill\_array приведена на рисунке 3.

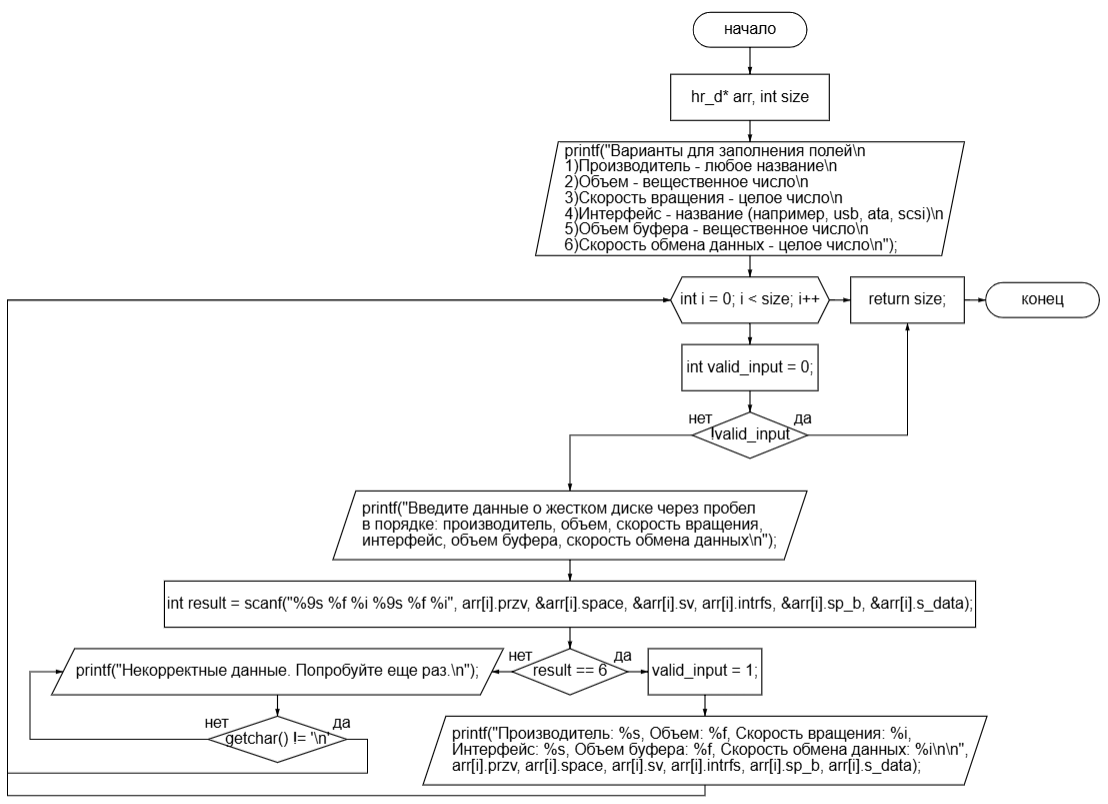


Рисунок 3 – Блок-схема функции int fill\_array(hr\_d\* arr, int size)

Функция int print\_array(hr\_d\* arr, int size) предназначена для вывода базы данных на экран. Функция принимает указатель структурного типа на массив и размер массива (целое число). Возвращает размер массива. Функция приведена на рисунке 4.

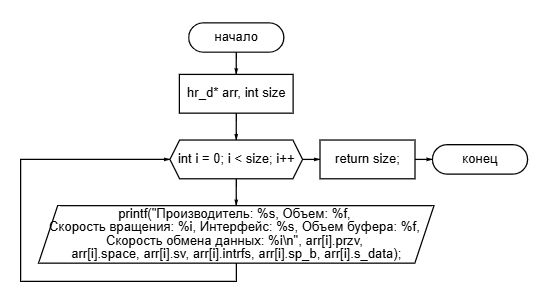


Рисунок 4 – Блок-схема функции int print\_array(hr\_d\* arr, int size)

Функция int input\_file(char\* fname, hr\_d\* arr, int size) предназначена для записи в файл созданной базы данных. Функция принимает указатель на название файла для записи, указатель структурного типа на массив и размер массива, возвращает размер. Блок-схема функции приведена на рисунке 5.

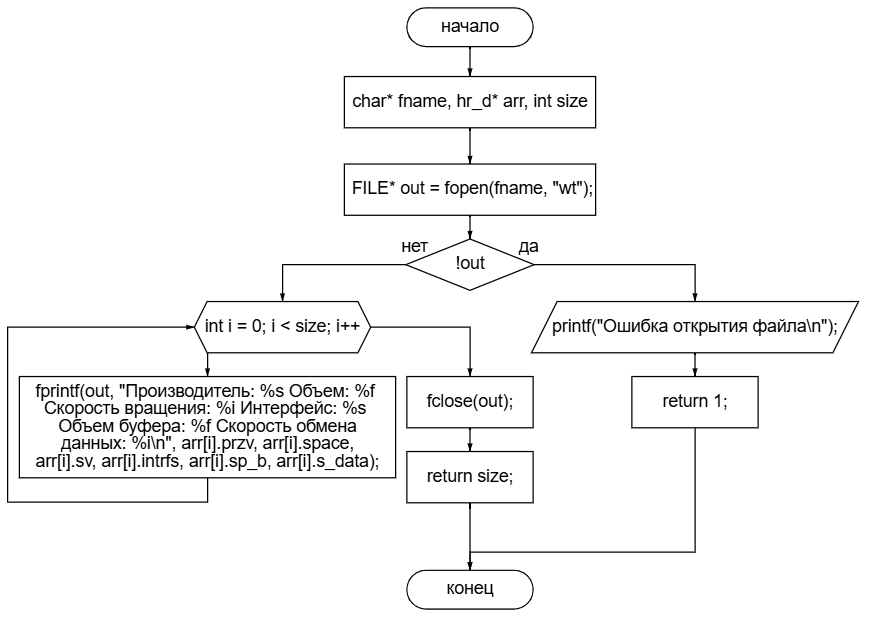


Рисунок 5 – Блок-схема функции int input\_file(char\* fname, hr\_d\* arr, int size)

Функция int output\_file(char\* fname, hr\_d\* arr, int size) предназначена для чтения базы данных из файла. Принимает параметры: указатель на название файла, указатель структурного типа на массив, в который записываются данные, целочисленный размер данного массива. Возвращает количество считанных структур. Блок-схема функции представлена на рисунке 6.

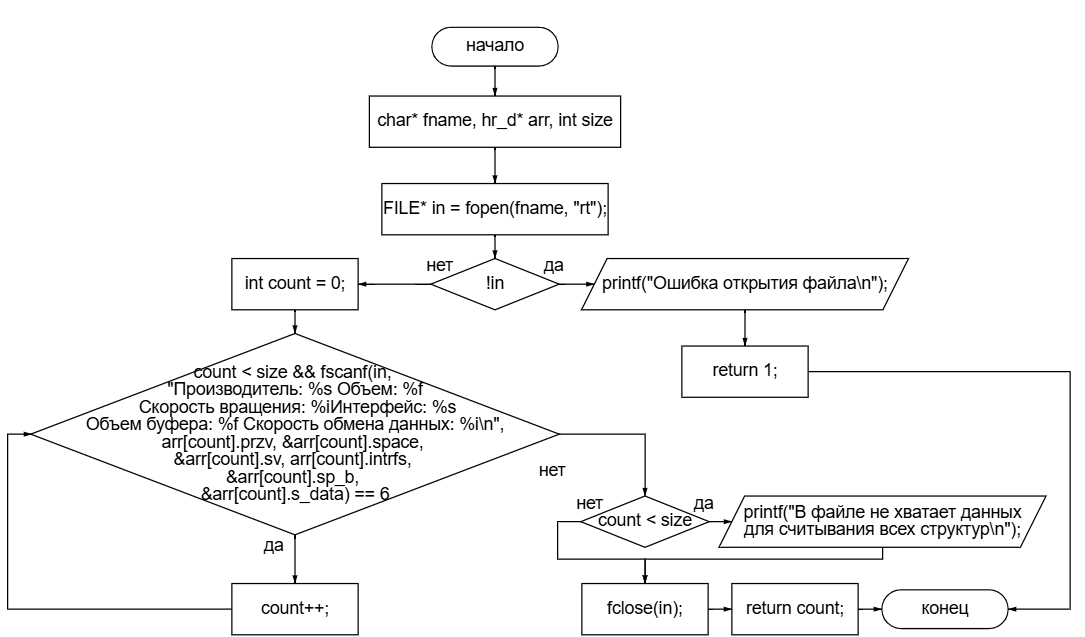


Рисунок 6 – Блок-схема функции int output\_file(char\* fname, hr\_d\* arr, int size)

Функция int find\_element(hr\_d\* arr, int size) предназначена для поиска записи в базе данных по одному или двум полям. Функция принимает указатель структурного типа на исходный массив и его размер (целое число). Возвращает размер. Блок-схема функции представлена на рисунке 7.

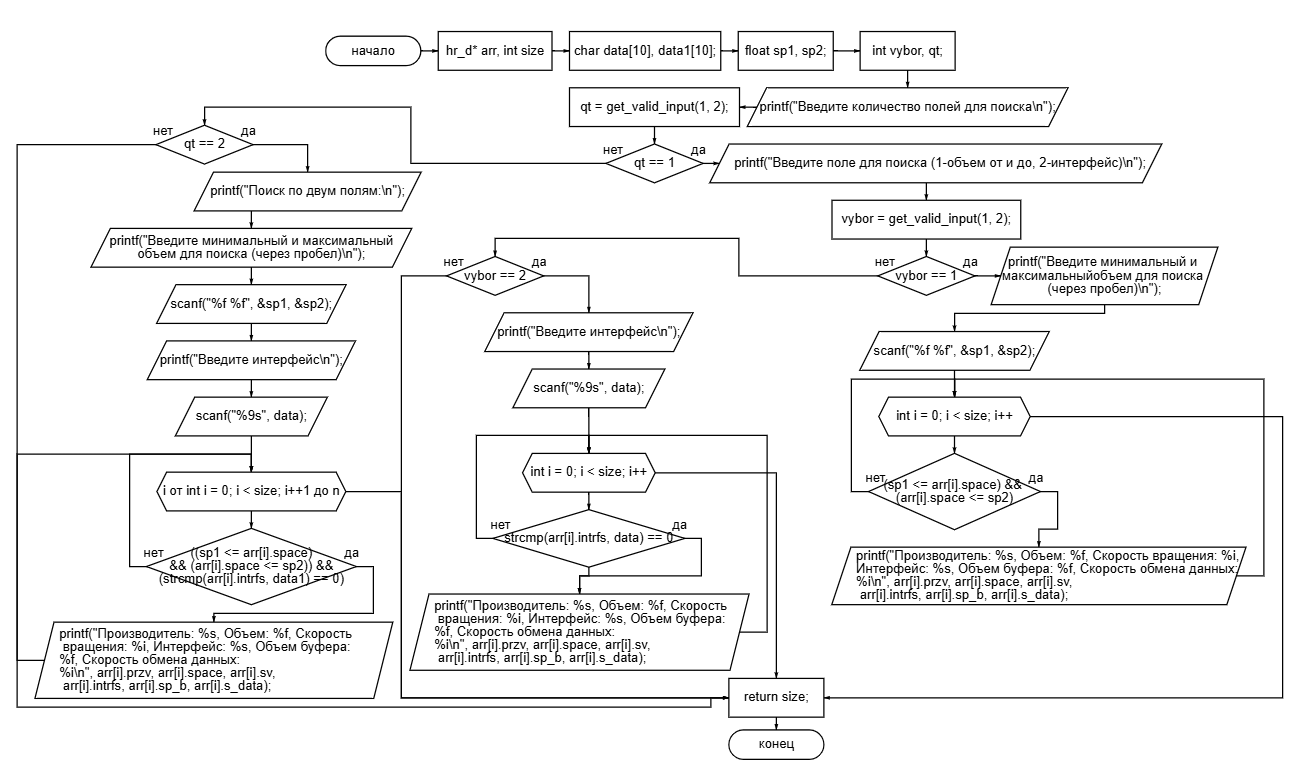


Рисунок 7 – Блок-схема функции int find\_element(hr\_d\* arr, int size)

Функция int sort\_massiv(hr\_d\* arr, int size) предназначена для сортировки базы данных по одному из полей. Функция принимает указатель структурного типа на исходный массив и размер массива, возвращает размер массива. Блок-схема функции представлена на рисунке 8.

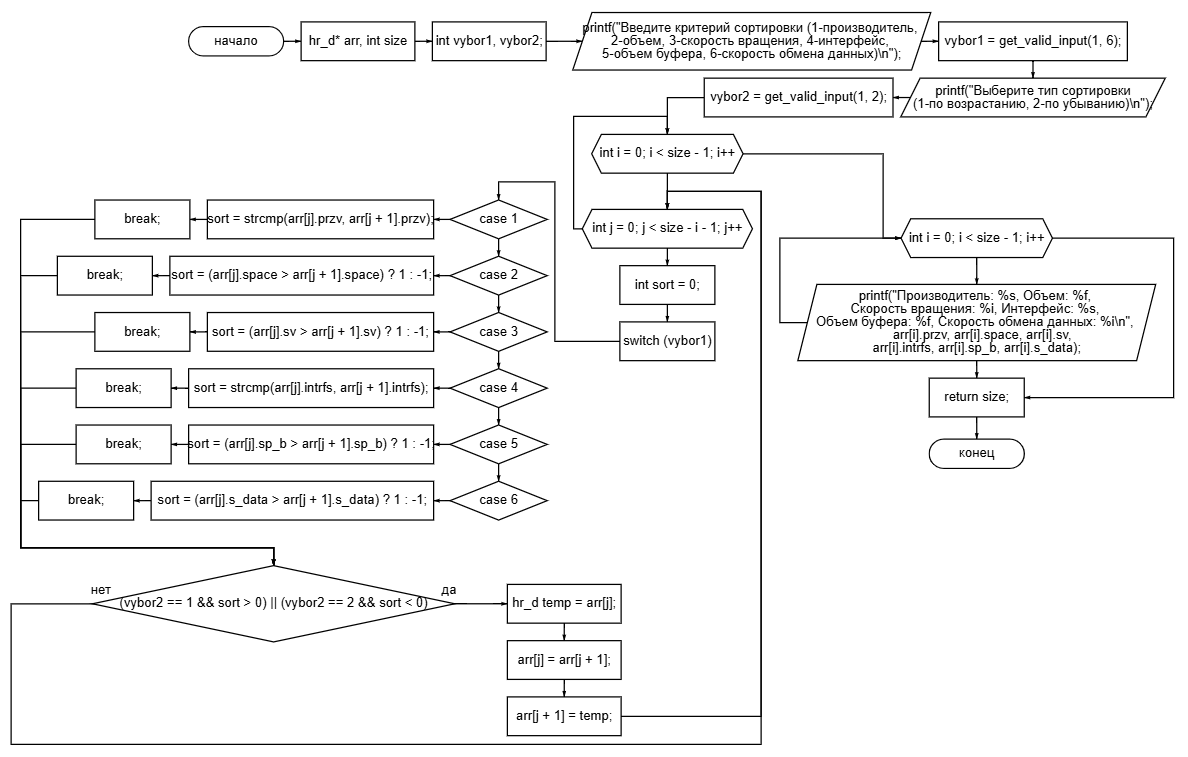


Рисунок 8 – Блок-схема функции int sort\_massiv(hr\_d\* arr, int size)

Функция int add\_elem(hr\_d\* arr, int size) предназначена для добавления в базу данных произвольного количества элементов. Принимает указатель структурного типа на исходный массив и целочисленный размер массива; возвращает новый размер массива. Блок-схема функции представлена на рисунке 9.

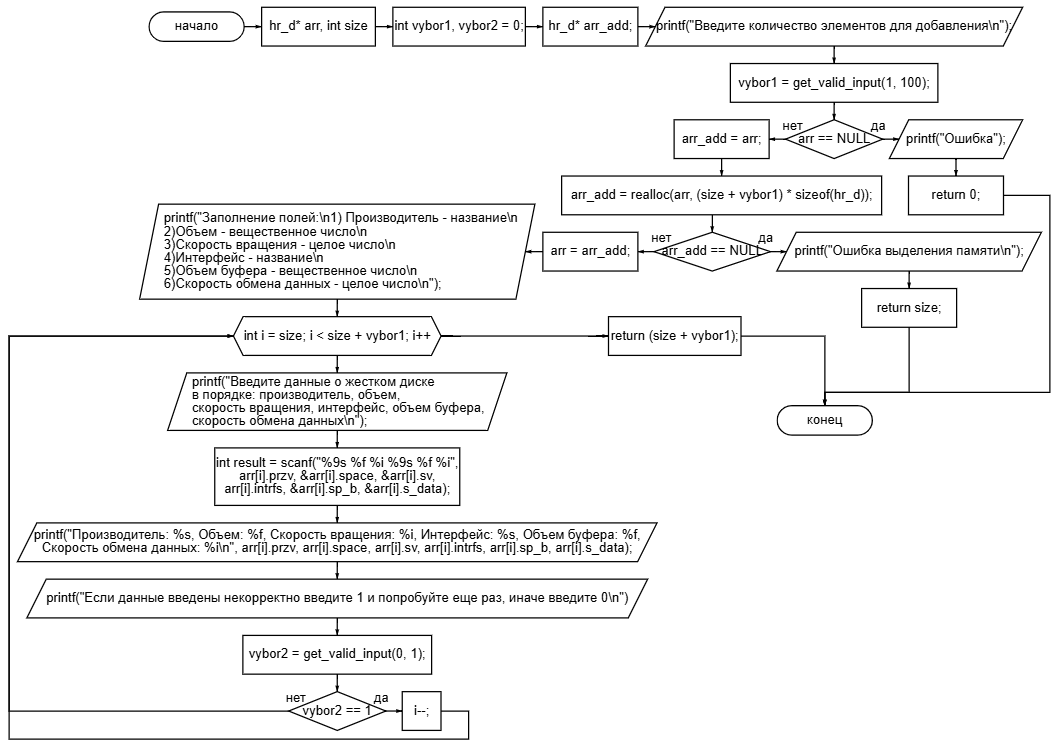


Рисунок 9 – Блок-схема функции int add\_elem(hr\_d\* arr, int size)

Функция int edit\_elem(hr\_d\* arr, int size) предназначена для изменения записей в базе данных. Принимает указатель структурного типа на массив и размер массива, возвращает размер. Блок-схема функции представлена на рисунке 10.

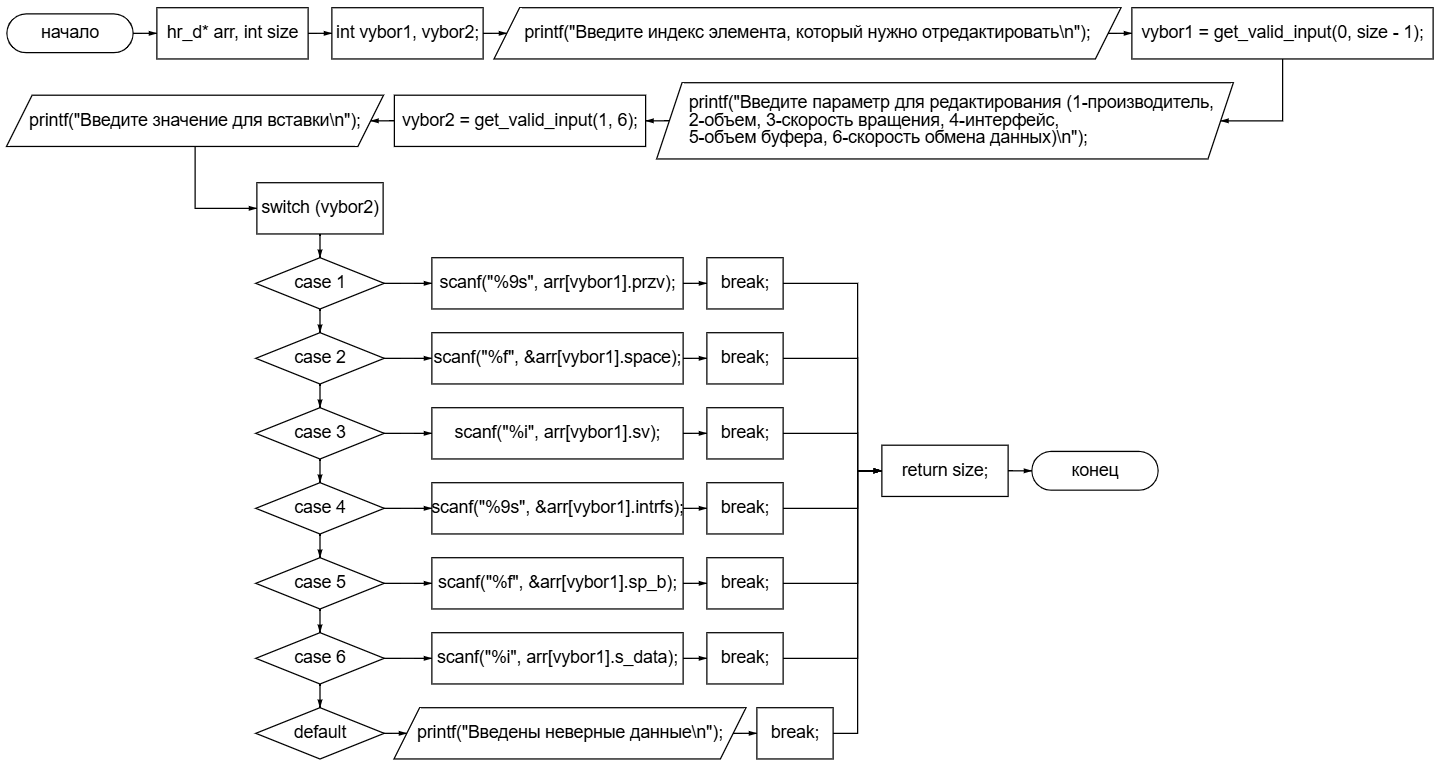


Рисунок 10 – Блок-схема функции int edit\_elem(hr\_d\* arr, int size)

Все собственные функции, используемые для работы с базой данных, приведены в таблице 1 с описанием:

|  |  |
| --- | --- |
| Функции | Назначение |
| int fill\_array(hr\_d\* arr, int size) | Заполнение массива данными, введенными пользователем |
| int print\_array(hr\_d\* arr, int size) | Вывод на экран всех записей |
| int input\_file(char\* fname, hr\_d\* arr, int size) | Запись текущего массива в файл с именем, введенным пользователем |
| int output\_file(char\* fname, hr\_d\* arr, int size) | Считывание данных из файла, указанного пользователем |
| int find\_element(hr\_d\* arr, int size) | Поиск записи в массиве по одному или двум критериям |
| int sort\_massiv(hr\_d\* arr, int size) | Сортировка текущего массива по указанному критерию |
| int add\_elem(hr\_d\* arr, int size) | Добавление в текущий массив произвольного числа записей |
| int edit\_elem(hr\_d\* arr, int size) | Изменение записи на введенное пользователем значение |

# 3 ТЕСТИРОВАНИЕ ПРОГРАММЫ

После запуска на экран будет выведено меню с выбором источника изначальной базы данных (рисунок 11).



Рисунок 11 – Получение массива

При выборе создания нового массива программа запросит размер массива, после чего выведет памятку о заполнении полей в базе данных (рисунок 12). При вводе некорректных данных программа выведет ошибку (рисунок 13).

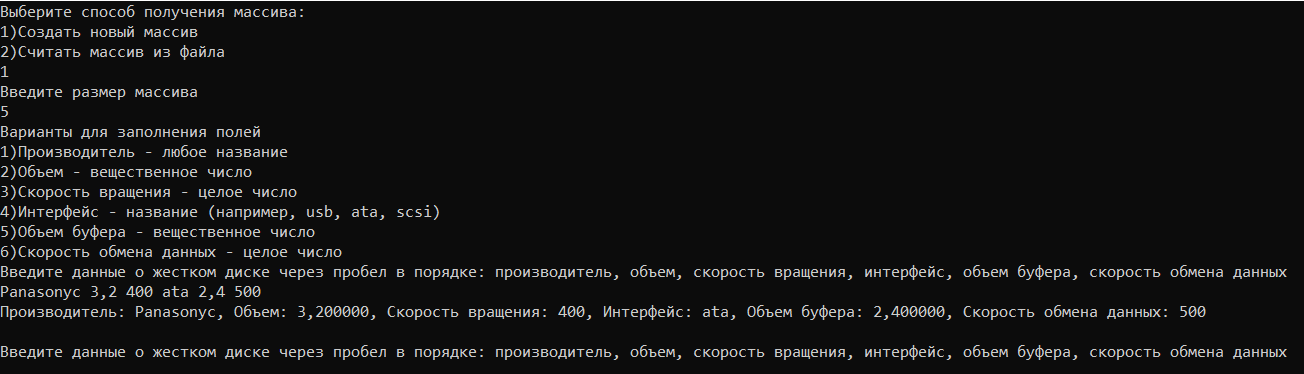


Рисунок 12 – Заполнение нового массива данными

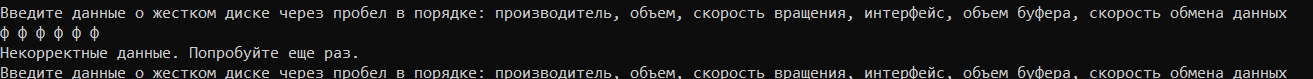


Рисунок 13 – Предупреждение о неправильном вводе данных

При выборе считывания массива из файла потребуется ввести количество записей и название файла (рисунок 14). Если требуемое количество записей меньше, чем количество записей в файле, будет считано нужное количество. Если требуемое количество превышает имеющееся, будет выведено предупреждение (рисунок 15).

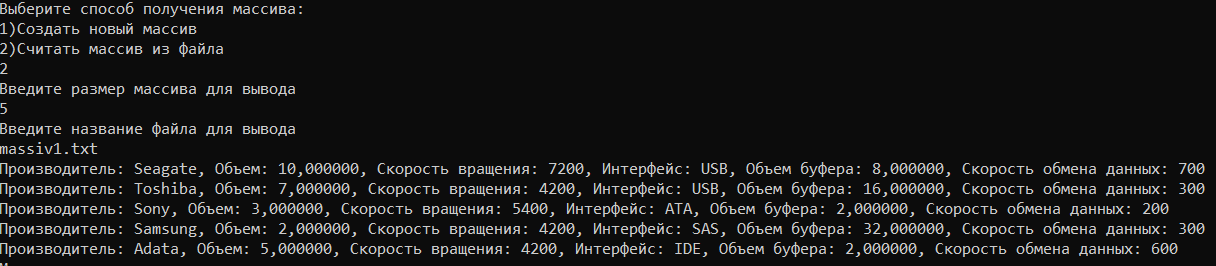


Рисунок 14 – Считывание данных из файла

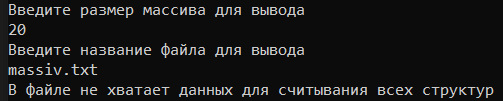


Рисунок 15 – Предупреждение о нехватке данных в файле

Содержание файла приведем на рисунке 16.

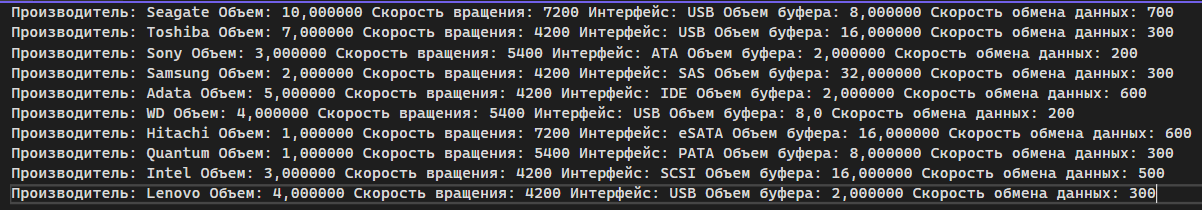


Рисунок 16 – Содержание файла massiv1.txt

После получения массива будет выведено меню действий с базой данных (рисунок 17).

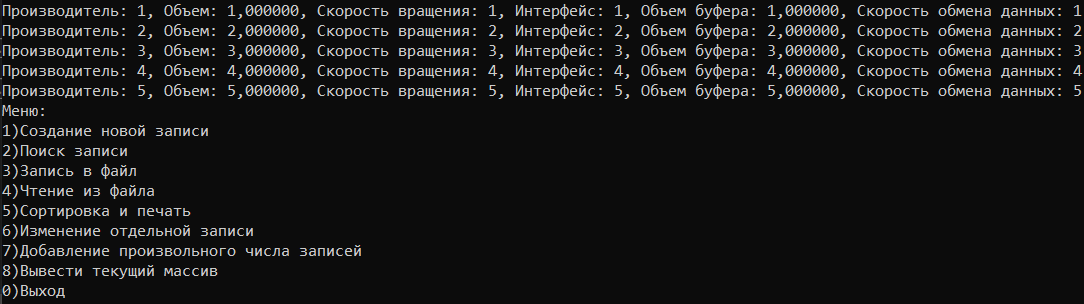


Рисунок 17 – Меню действий с массивом

При выборе поиска записи будет предложено ввести количество критериев поиска и информацию для поиска (рисунок 18, 19).

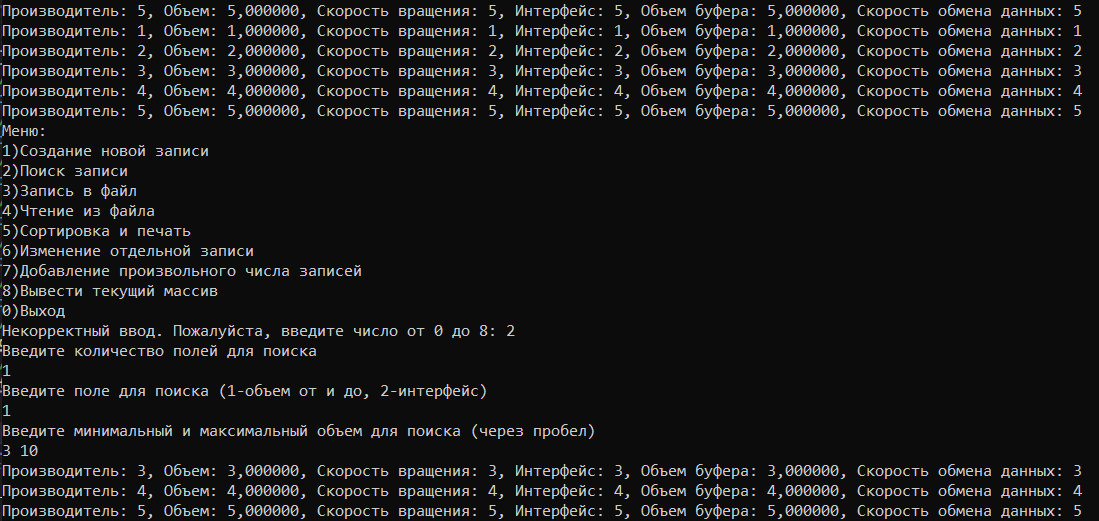


Рисунок 18 – Поиск по одному полю

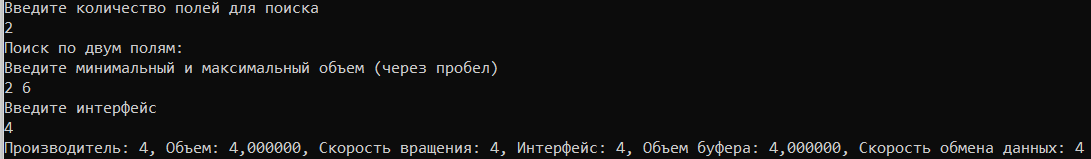


Рисунок 19 – Поиск по двум полям

При выборе записи в файл (действие 3) будет предложено ввести название файла. При успешной записи в файл будет выведена информация из него (рисунок 20). При выборе чтения из файла (действие 4) созданный массив будет заменен считанным из файла (рисунок 14).

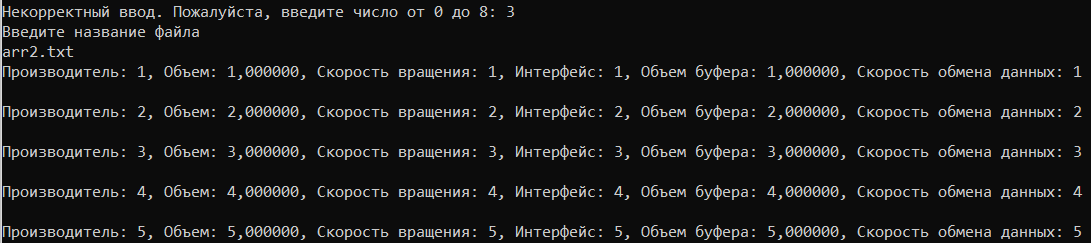


Рисунок 20 – Запись массива в файл

При выборе сортировки и печати (действие 5) программа отсортирует текущий массив по введенному пользователем критерию в введенном порядке и выведет его на экран (рисунок 21).

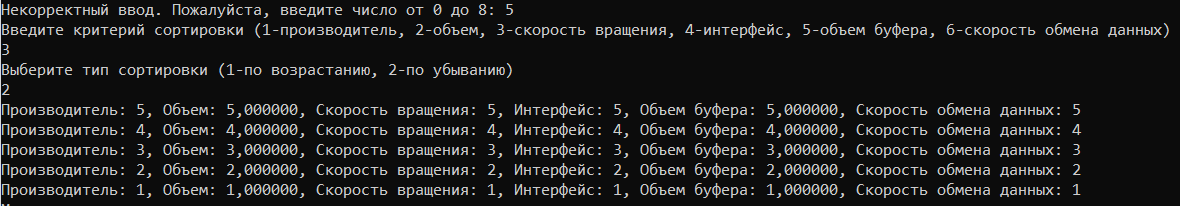


Рисунок 21 – Сортировка массива по убыванию

При выборе изменения отдельной записи (действие 6) будет предложено ввести индекс редактируемой записи и номер изменяемого поля (рисунок 22).

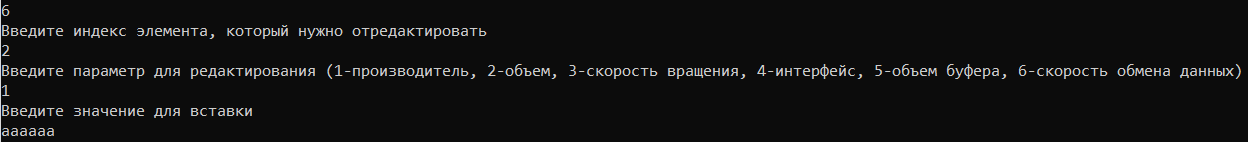


Рисунок 22 – Изменение отдельной записи

Чтобы увидеть изменения, можно выбрать печать текущего массива (действие 8). Приведем результат на рисунке 23.

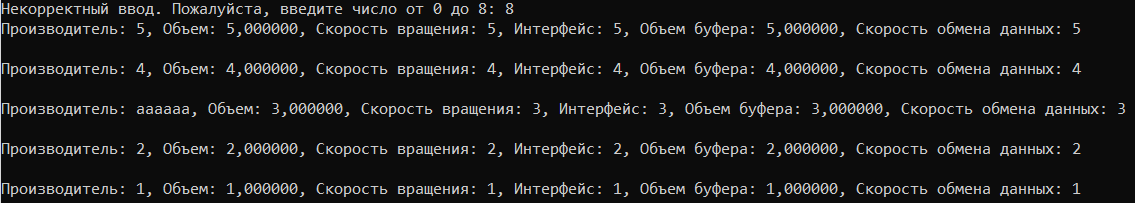


Рисунок 23 – Печать текущего массива

При выборе добавления произвольного количества записей (действие 7) будет предложено ввести количество добавляемых элементов. Дальше пользователь должен вводить новые записи по одной, после каждой записи будет предложено отредактировать введенную информацию в случае неверно-

го ввода (рисунки 24, 25) Чтобы увидеть новый массив, необходимо выбрать действие 8 (рисунок 26).



Рисунок 24 – Ввод количества новых записей

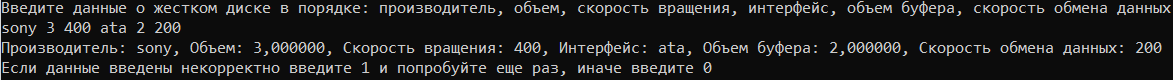


Рисунок 25 – Проверка корректности введения записи

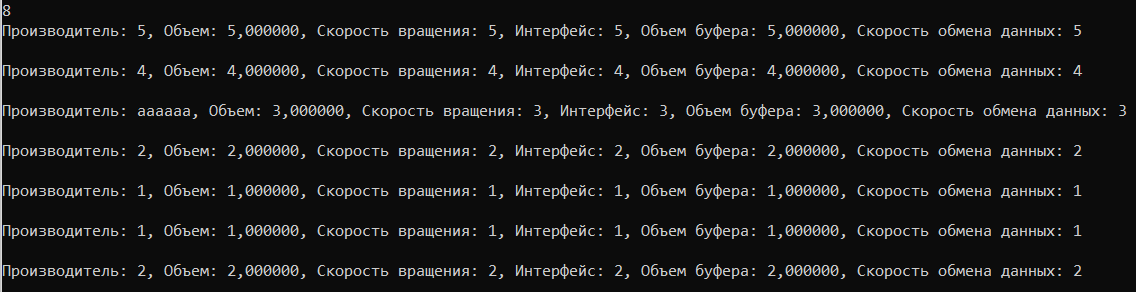


Рисунок 26 – Добавление записей в массив

При выборе выхода (действие 0) программа завершается (рисунок 27).

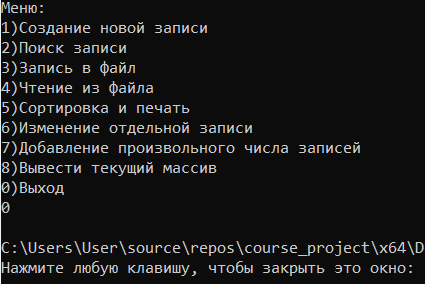


Рисунок 27 – Выход из программы

# ЗАКЛЮЧЕНИЕ

При выполнении курсового проекта была разработана программа для работы с файловыми базами данных. Были выполнены все поставленные задачи. Была создана структура для записей, реализован интерфейс для взаимодействия с пользователем. В программе предусмотрены: поиск записи по одному или двум критериям; сортировка массива по любому полю в порядке убывания или возрастания; добавление произвольного количества новых записей; редактирование произвольного поля произвольной записи; запись базы данных в файл и чтение из файла. Данная программа также включает проверки на корректность введения данных пользователем и на существование создаваемых файлов и массивов, что позволяет обезопасить ввод и хранение данных. Таким образом, в ходе выполнения данной курсовой работы была создана программа для хранения данных о жестких дисках, которая может быть использована для различных задач.

# СПИСОК ЛИТЕРАТУРЫ

3. Кнут, Д. Искусство программирования. В 3 т. — М.: Вильямс, 2012.

7. Павловская, Т. А. С/С++. Программирование на языке высокого уровня. — СПб., 2003. — 461 с.

9. Подбельский, В. В., Фомин, С. С. Программирование на языке Си: Учеб. пособие. 2-е доп. изд. — М.: Финансы и статистика, 2004. — 600 с.

10. Гукин, Д. Программирование на C для чайников. — М.: Диалектика, 2019. — 384 с.

12. Гергель, В. П. Современные языки и технологии параллельного программирования: Учебник. — М.: МГУ, 2016. — 408 с.

# ПРИЛОЖЕНИЕ А

ОПИСАНИЕ СТРУКТУРНОГО ТИПА ДАННЫХ

struct hard\_drive {

char przv[10]; // производитель

float space; // объем

int sv; // скорость вращения

char intrfs[10]; // интерфейс

float sp\_b; // объем буфера

int s\_data; //скорость обмена данных

};

typedef struct hard\_drive hr\_d;

# ПРИЛОЖЕНИЕ В

ЛИСТИНГ ПРОГРАММЫ

#include <locale.h>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include <ctype.h>

#include <time.h>

//-------функции-------

// заполнение массива

int fill\_array(hr\_d\* arr, int size) {

printf("Варианты для заполнения полей\n1)Производитель - любое название\n2)Объем - вещественное число\n3)Скорость вращения - целое число\n4)Интерфейс - название (например, usb, ata, scsi)\n5)Объем буфера - вещественное число\n6)Скорость обмена данных - целое число\n");

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

int valid\_input = 0; // корректность ввода

while (!valid\_input) {

printf("Введите данные о жестком диске через пробел в порядке: производитель, объем, скорость вращения, интерфейс, объем буфера, скорость обмена данных\n");

int result = scanf("%9s %f %i %9s %f %i", arr[i].przv, &arr[i].space, &arr[i].sv, arr[i].intrfs, &arr[i].sp\_b, &arr[i].s\_data);

if (result == 6) {

valid\_input = 1;

printf("Производитель: %s, Объем: %f, Скорость вращения: %i, Интерфейс: %s, Объем буфера: %f, Скорость обмена данных: %i\n\n", arr[i].przv, arr[i].space, arr[i].sv, arr[i].intrfs, arr[i].sp\_b, arr[i].s\_data);

}

else {

printf("Некорректные данные. Попробуйте еще раз.\n");

while (getchar() != '\n'); // очистка буфера ввода

}

}

}

return size;

}

// печать массива

int print\_array(hr\_d\* arr, int size) {

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

printf("Производитель: %s, Объем: %f, Скорость вращения: %i, Интерфейс: %s, Объем буфера: %f, Скорость обмена данных: %i\n\n", arr[i].przv, arr[i].space, arr[i].sv, arr[i].intrfs, arr[i].sp\_b, arr[i].s\_data);

}

return size;

}

// запись в файл

int input\_file(char\* fname, hr\_d\* arr, int size) {

FILE\* out = fopen(fname, "wt"); // файл для записи массива

if (!out) {

printf("Ошибка открытия файла\n");

return 1;

}

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

fprintf(out, "Производитель: %s Объем: %f Скорость вращения: %i Интерфейс: %s Объем буфера: %f Скорость обмена данных: %i\n",

arr[i].przv, arr[i].space, arr[i].sv, arr[i].intrfs, arr[i].sp\_b, arr[i].s\_data);

}

fclose(out);

return size;

}

// чтение из файла

int output\_file(char\* fname, hr\_d\* arr, int size) {

FILE\* in = fopen(fname, "rt"); // файл, откуда считывается массив

if (!in) {

printf("Ошибка открытия файла\n");

return 1;

}

int count = 0; // количество считанных структур

while (count < size && fscanf(in, "Производитель: %s Объем: %f Скорость вращения: %i Интерфейс: %s Объем буфера: %f Скорость обмена данных: %i\n", arr[count].przv, &arr[count].space, &arr[count].sv, arr[count].intrfs, &arr[count].sp\_b, &arr[count].s\_data) == 6) {

count++;

}

if (count < size) {

printf("В файле не хватает данных для считывания всех структур\n");

}

fclose(in);

return count;

}

// поиск элемента

int find\_element(hr\_d\* arr, int size) {

char data[10], data1[10];// данные об интерфейсе

float sp1, sp2; // данные об объеме: sp1 - минимальный объем; sp2 - максимальный

int vybor, qt; // vybor - выбор поля, qt - количество искомых полей

printf("Введите количество полей для поиска\n");

qt = get\_valid\_input(1, 2);

if (qt == 1) {

printf("Введите поле для поиска (1-объем от и до, 2-интерфейс)\n");

vybor = get\_valid\_input(1, 2);

if (vybor == 1) {

printf("Введите минимальный и максимальный объем для поиска (через пробел)\n");

scanf("%f %f", &sp1, &sp2);

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

if ((sp1 <= arr[i].space) && (arr[i].space <= sp2)) printf("Производитель: %s, Объем: %f, Скорость вращения: %i, Интерфейс: %s, Объем буфера: %f, Скорость обмена данных: %i\n", arr[i].przv, arr[i].space, arr[i].sv, arr[i].intrfs, arr[i].sp\_b, arr[i].s\_data);

}

}

else if (vybor == 2) {

printf("Введите интерфейс\n");

scanf("%9s", data);

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

if (strcmp(arr[i].intrfs, data) == 0)

printf("Производитель: %s, Объем: %f, Скорость вращения: %i, Интерфейс: %s, Объем буфера: %f, Скорость обмена данных: %i\n",

arr[i].przv, arr[i].space, arr[i].sv, arr[i].intrfs, arr[i].sp\_b, arr[i].s\_data);

}

}

}

else if (qt == 2) {

printf("Поиск по двум полям:\n");

printf("Введите минимальный и максимальный объем (через пробел)\n");

scanf("%f %f", &sp1, &sp2);

printf("Введите интерфейс\n");

scanf("%9s", data1);

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

if (((sp1 <= arr[i].space) && (arr[i].space <= sp2)) && (strcmp(arr[i].intrfs, data1) == 0)) printf("Производитель: %s, Объем: %f, Скорость вращения: %i,

Интерфейс: %s, Объем буфера: %f, Скорость обмена данных: %i\n", arr[i].przv, arr[i].space, arr[i].sv, arr[i].intrfs, arr[i].sp\_b, arr[i].s\_data);

}

}

return size;

}

// сортировка массива

int sort\_massiv(hr\_d\* arr, int size) {

int vybor1, vybor2; // выбор пользователя

printf("Введите критерий сортировки (1-производитель, 2-объем, 3-скорость вращения, 4-интерфейс, 5-объем буфера, 6-скорость обмена данных)\n");

vybor1 = get\_valid\_input(1, 6);

printf("Выберите тип сортировки (1-по возрастанию, 2-по убыванию)\n");

vybor2 = get\_valid\_input(1, 2);

for (int i = 0; i < size - 1; i++) {

for (int j = 0; j < size - i - 1; j++) {

int sort = 0;

switch (vybor1) {

case 1: sort = strcmp(arr[j].przv, arr[j + 1].przv); break;

case 2: sort = (arr[j].space > arr[j + 1].space) ? 1 : -1; break;

case 3: sort = (arr[j].sv > arr[j + 1].sv) ? 1 : -1; break;

case 4: sort = strcmp(arr[j].intrfs, arr[j + 1].intrfs); break;

case 5: sort = (arr[j].sp\_b > arr[j + 1].sp\_b) ? 1 : -1; break;

case 6: sort = (arr[j].s\_data > arr[j + 1].s\_data) ? 1 : -1; break;

}

if ((vybor2 == 1 && sort > 0) || (vybor2 == 2 && sort < 0)) {

hr\_d temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

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

printf("Производитель: %s, Объем: %f, Скорость вращения: %i, Интерфейс: %s, Объем буфера: %f, Скорость обмена данных: %i\n",

arr[i].przv, arr[i].space, arr[i].sv, arr[i].intrfs, arr[i].sp\_b, arr[i].s\_data);

}

return size;

}

// добавление произвольного количества элементов

int add\_elem(hr\_d\* arr, int size) {

int vybor1, vybor2 = 0; // выбор пользователя

hr\_d\* arr\_add; // измененный массив

printf("Введите количество элементов для добавления\n");

vybor1 = get\_valid\_input(1, 100); // ограничение количества добавляемых элементов

if (arr == NULL) {

printf("Ошибка");

return 0;

}

arr\_add = arr;

arr\_add = realloc(arr, (size + vybor1) \* sizeof(hr\_d));

if (arr\_add == NULL) {

printf("Ошибка выделения памяти\n");

return size;

}

arr = arr\_add;

printf("Заполнение полей:\n1) Производитель - название\n2)Объем - вещественное число\n3)Скорость вращения - целое число\n4)Интерфейс - название\n5)Объем буфера - вещественное число\n6)Скорость обмена данных - целое число\n");

for (int i = size; i < size + vybor1; i++) {

printf("Введите данные о жестком диске в порядке: производитель, объем, скорость вращения, интерфейс, объем буфера, скорость обмена данных\n");

int result = scanf("%9s %f %i %9s %f %i", arr[i].przv, &arr[i].space, &arr[i].sv, arr[i].intrfs, &arr[i].sp\_b, &arr[i].s\_data);

if (result != 6) {

printf("Некорректные данные. Попробуйте еще раз.\n");

while (getchar() != '\n'); // очистка буфера ввода

i--;

continue;

}

printf("Производитель: %s, Объем: %f, Скорость вращения: %i, Интерфейс: %s, Объем буфера: %f, Скорость обмена данных: %i\n", arr[i].przv, arr[i].space, arr[i].sv, arr[i].intrfs, arr[i].sp\_b, arr[i].s\_data);

printf("Если данные введены некорректно введите 1 и попробуйте еще раз, иначе введите 0\n");

vybor2 = get\_valid\_input(0, 1);

if (vybor2 == 1) i--;

}

return (size + vybor1);

}

// изменение элемента

int edit\_elem(hr\_d\* arr, int size) {

int vybor1, vybor2; // выбор пользователя

printf("Введите индекс элемента, который нужно отредактировать\n");

vybor1 = get\_valid\_input(0, size - 1);

printf("Введите параметр для редактирования (1-производитель, 2-объем, 3-скорость вращения, 4-интерфейс, 5-объем буфера, 6-скорость обмена данных)\n");

vybor2 = get\_valid\_input(1, 6);

printf("Введите значение для вставки\n");

switch (vybor2) {

case 1: scanf("%9s", arr[vybor1].przv); break;

case 2: scanf("%f", &arr[vybor1].space); break;

case 3: scanf("%i", arr[vybor1].sv); break;

case 4: scanf("%9s", &arr[vybor1].intrfs); break;

case 5: scanf("%f", &arr[vybor1].sp\_b); break;

case 6: scanf("%i", arr[vybor1].s\_data); break;

default:

printf("Введены неверные данные\n");

break;

}

return size;

}

// функция проверки корректности ввода числа

get\_valid\_input(int min, int max) {

int input;

char buffer[256];

while (1) {

if (fgets(buffer, sizeof(buffer), stdin) != NULL) {

if (sscanf(buffer, "%d", &input) == 1 && input >= min && input <= max) {

return input;

}

else {

printf("Некорректный ввод. Пожалуйста, введите число от %d до %d: ", min, max);

}

}

}

}

// функция для проверки существования файла

file\_have(const char\* filename) {

FILE\* file = fopen(filename, "r");

if (file) {

fclose(file);

return 1;

}

return 0;

}

//-----основная программа с интерфейсом-----

int main() {

setlocale(LC\_ALL, "RUS");

hr\_d\* arr0 = NULL; // исходный массив

hr\_d\* arr\_out = NULL; // массив для вывода из файла

int size0 = 0, size\_out = 0; // размер массивов

int vybor = 123, vybor1 = 0; // выбор вариантов

char fname1[20] = "", fname2[20] = ""; // названия файлов

// получение массива для работы

printf("Выберите способ получения массива:\n1)Создать новый массив\n2)Считать массив из файла\n");

vybor1 = get\_valid\_input(1, 2);

switch (vybor1) {

case 1:

printf("Введите размер массива\n");

size0 = get\_valid\_input(1, 100); // огранение размера массива

arr0 = (hr\_d\*)malloc(size0 \* sizeof(hr\_d));

fill\_array(arr0, size0);

print\_array(arr0, size0);

break;

case 2:

printf("Введите размер массива для вывода\n");

size\_out = get\_valid\_input(1, 100); // ограничим размер массива до 100

printf("Введите название файла для вывода\n");

scanf("%s", fname2);

if (!file\_have(fname2)) {

printf("Файл с таким именем не существует. Введите корректное имя файла.\n");

break;

}

arr\_out = (hr\_d\*)malloc(size\_out \* sizeof(hr\_d));

size\_out = output\_file(fname2, arr\_out, size\_out);

arr0 = arr\_out;

size0 = size\_out;

print\_array(arr0, size0);

break;

default:

break;

}

//-------основной цикл-------

while (vybor != 0) {

printf("Меню:\n1)Создание новой записи\n2)Поиск записи\n3)Запись в файл\n4)Чтение из файла\n5)Сортировка и печать\n6)Изменение отдельной

записи\n7)Добавление произвольного числа записей\n8)Вывести текущий массив\n0)Выход\n");

vybor = get\_valid\_input(0, 8);

switch (vybor) {

case 1:// создание новой записи

printf("Введите размер массива\n");

size0 = get\_valid\_input(1, 100); // ограничение размера массива

arr0 = (hr\_d\*)malloc(size0 \* sizeof(hr\_d));

fill\_array(arr0, size0);

print\_array(arr0, size0);

break;

case 2:// поиск элемента

find\_element(arr0, size0);

break;

case 3:// запись в файл

printf("Введите название файла\n");

scanf("%s", fname1);

input\_file(fname1, arr0, size0);

print\_array(arr0, size0);

break;

case 4:// чтение из файла

printf("Введите размер массива, который необходимо вывести\n");

size\_out = get\_valid\_input(1, 100); // ограничение размера массива

printf("Введите название файла, из которого нужно вывести информацию\n");

scanf("%s", fname2);

if (!file\_have(fname2)) {

printf("Файл с таким именем не существует. Пожалуйста, введите корректное имя файла.\n");

break;

}

arr\_out = (hr\_d\*)malloc(size\_out \* sizeof(hr\_d));

size\_out = output\_file(fname2, arr\_out, size\_out);

arr0 = arr\_out;

size0 = size\_out;

print\_array(arr0, size0);

break;

case 5:// сортировка массива

sort\_massiv(arr0, size0);

break;

case 6:// изменение записи

edit\_elem(arr0, size0);

break;

case 7:// добавление произвольного количества элементов

size0 = add\_elem(arr0, size0);

break;

case 8:// вывод массива

print\_array(arr0, size0);

break;

default:

break;

}

}

free(arr0);// освобождение памяти

free(arr\_out);// освобождение памяти

}