МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ФГБОУ ВО «ВГТУ»)

Факультет информационных технологий и компьютерной безопасности

(Факультет)

Кафедра Систем управления и информационных технологий в строительстве

КУРСОВОЙ ПРОЕКТ

по дисциплине: Основы программирования и алгоритмизации

Тема: «Разработка файловой базы данных «Процессоры»

**Расчетно-пояснительная записка**

Разработал студент В.С.Крикуненко

Подпись, дата Инициалы, фамилия

Руководитель О. В. Курипта

Подпись, дата Инициалы, фамилия

Нормоконтролер О.В. Курипта

Подпись, дата Инициалы, фамилия

Защищена \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ФГБОУ ВО «ВГТУ»)

Кафедра Систем управления и информационных технологий в строительстве

ЗАДАНИЕ  
на курсовой̆ проект

по дисциплине: Основы программирования и алгоритмизации

Тема: «Разработка файловой базы данных «Процессоры»

Студент группы бИД-242 Крикуненко Валерий Сергеевич

Фамилия, имя, отчество

Технические условия Windows 11, MicrosoftVisualStudio2022, язык программирования C

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

32 страниц, 17 рисунков, 3 таблиц, 1 приложение

Сроки выполнения этапов анализ и постановка задачи (10.09.24 – 12.10.24); разработка пошаговой детализации программы (13.10.24. – 11.11.24); реализация программы (12.11.24.-10.12.24); тестирование программы (11.12.24 – 12.12.24); оформление пояснительной записки (13.12.24 – 16.12.24).

Срок защиты курсового проекта:

Руководитель О. В. Курипта

Подпись, дата Инициалы, фамилия

Задание принял студент В. С. Крикуненко

Подпись, дата Инициалы, фамилия

Замечания руководителя

СОДЕРЖАНИЕ

[Введение 5](#_Toc187154307)

[1 Постановка задачи 6](#_Toc187154308)

[2 Реализация программы 8](#_Toc187154309)

[3 Тестирование программы 17](#_Toc187154310)

[Заключение 21](#_Toc187154311)

[Список использованных источников 22](#_Toc187154312)

[Приложение 23](#_Toc187154313)

# 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 данной работы 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.

## 1 Постановка задачи

Для начала стоит определится с полями, входящими в структуру. Структура должна включать в себя следующие поля:

* Имя (типа «char»);
* Сокет (типа «char»);
* Цена (типа «int»);
* Количество ядер (типа «int»);
* Год выпуска (типа «\*int»);

Готовая программа будет содержать динамический массив элементов структуры, пример подобного массива, представленного в виде таблицы, можно увидеть в таблице 1.1.

Таблица 1.1 – Пример массива процессоров

| Название | Год | Цена | Сокет | Кол-во ядер |
| --- | --- | --- | --- | --- |
| Intel\_i5\_12900kf | 2017 | 12999 | LGA150 | 12 |
| Intel\_i7\_11900kf | 2015 | 11990 | LGA160 | 12 |
| Intel\_i7\_14900kf | 2021 | 23999 | LGA170 | 24 |
| Intel\_i9\_12700kf | 2020 | 19399 | LGA160 | 16 |
| Pentium\_g4400\_oem | 2010 | 5800 | LGA140 | 2 |
| Ryzen\_5\_5600x | 2017 | 14999 | AMD4 | 16 |

Программа будет взаимодействовать с пользователем по средствам внутреннего меню, которое будет содержать следующие пункты:

* Добавить процессор в базу данных;
* Удалить процессор из базы данных;
* Вывести все процессоры;
* Найти процессор по названию;
* Отсортировать;
* Изменить значение полей;
* Закончить сеанс;

Пользователь будет взаимодействовать с меню, вводя в консоль число, соответствующее пункту в меню.

Сохранения массива в файл будет производиться автоматически при каждом изменении массива пользователем, для минимизации лишних действий и возможной потери данных.

При взаимодействии с меню, пользователю будет предоставлен ввод данных для каждого действия. Требуемые данные для каждого действия показаны в таблице 1.2.

Таблица 1.2 – Действия и данные, требуемые для их выполнения

| Действие | Данные |
| --- | --- |
| Добавление процессора в базу данных | Значения полей нового планшета |
| Удаление процессора из базы данных | Название процессора |
| Вывод базы данных | Не требует дополнительного ввода данных |
| Сортировка | Номер действия для сортировки |
| Изменение значений полей | Название процессора |
| Поиск процессора | Название процессора или сокета |

## 2 Реализация программы

В данной программе использованы пять стандартных библиотек:

* <stdio.h> – для работы с вводом и выводом;
* <stdlib.h> – для выделения памяти под динамические массивы;
* <string.h> – для работы со сроками;
* <locale.h> – для русской локализации.
* <ctype.h> - функции для работы с символами;

Все программы в языке С должны содержать функцию «main» - точку входа в программу, т.е. она запускается при запуске приложения.

В программе будут созданы собственные функции, описанные в таблице 2.1.

Таблица 2.1 – Описание собственных функций

| Название функции | Назначение | Примечание |
| --- | --- | --- |
| int main() | Главная функция | Выводит информацию о программе и вызывает другие функции. |
| CPU\* readfile(const char\* fname, CPU\*\* inf, int\* count) | Инициализация массива из файла | Инициализирует массив читая данные из файла |
| int writefile(const char\* fname, CPU\* inf, int count) | Добавление нового процессора в массив и сохранение его в файл | Сохраняет данные из массива в файл |
| CPU\* deleteCPU(CPU\* inf, const char\* name, int\* count) | Удаление процессора из массива | Удаляет процессор с указанным названием |
| int strcasecmp\_custom(const char\* s1, const char\* s2) | Для чтения названий из файла и правильного сравнения с вводимым словом с консоли | Cравнивает две строки s1 и s2 без учета регистра. |
| void searchCPU(const char\* fname, const char\* query) | Нахождения массива элементов | Находит элемент с заданным названием и формирует из них новый массив и выводит его |
| CPU\* bubble\_sort\_by\_name(CPU\* cpus, int count) | Сортировка массива | Сортирует массив по названиям процессоров |
| CPU\* update(CPU\* inf, const char\* name, int count) | Задавание новых значений строкам массива | Задаёт значение, полученное с помощью ввода с консоли, строкам |
| CPU\* bubble\_sort\_by\_price(CPU\* cpus, int count) | Сортировка массива | Сортирует массив по цене процессоров |

int main().

Главная функция, в ней запускается бесконечный цикл, в котором даётся выбор действий с помощью меню, реализованного с помощью оператора switch case. Блок-схема представлена на рисунках 2.1, 2.2.

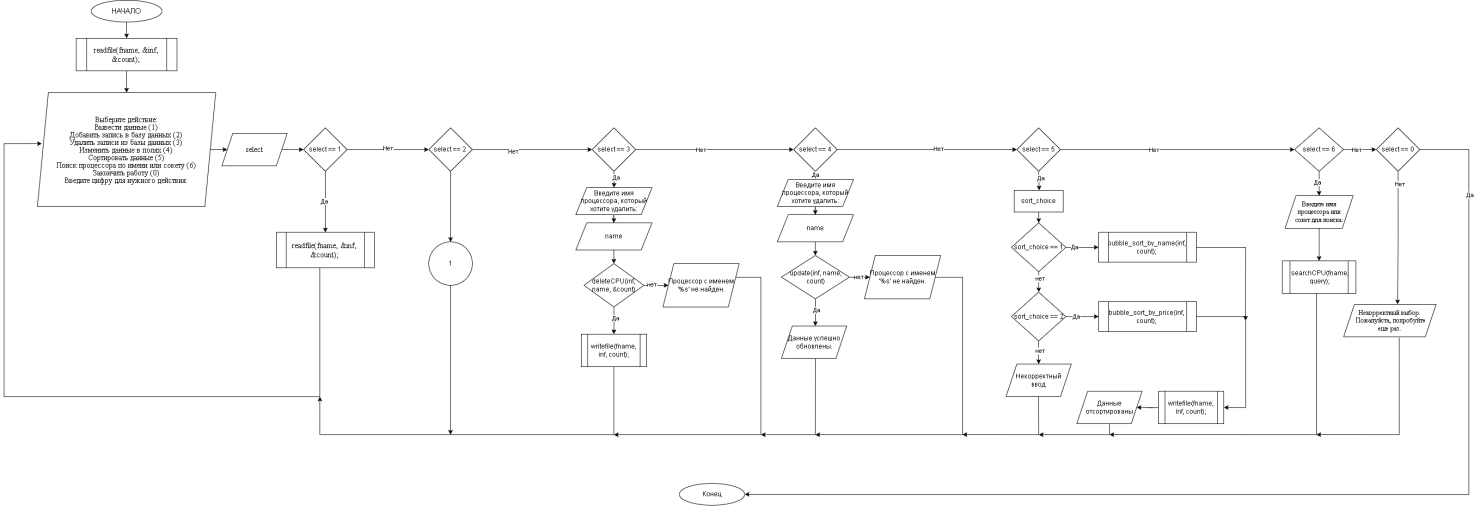


Рисунок 2.1 – Блок-схема функции main (1 часть)

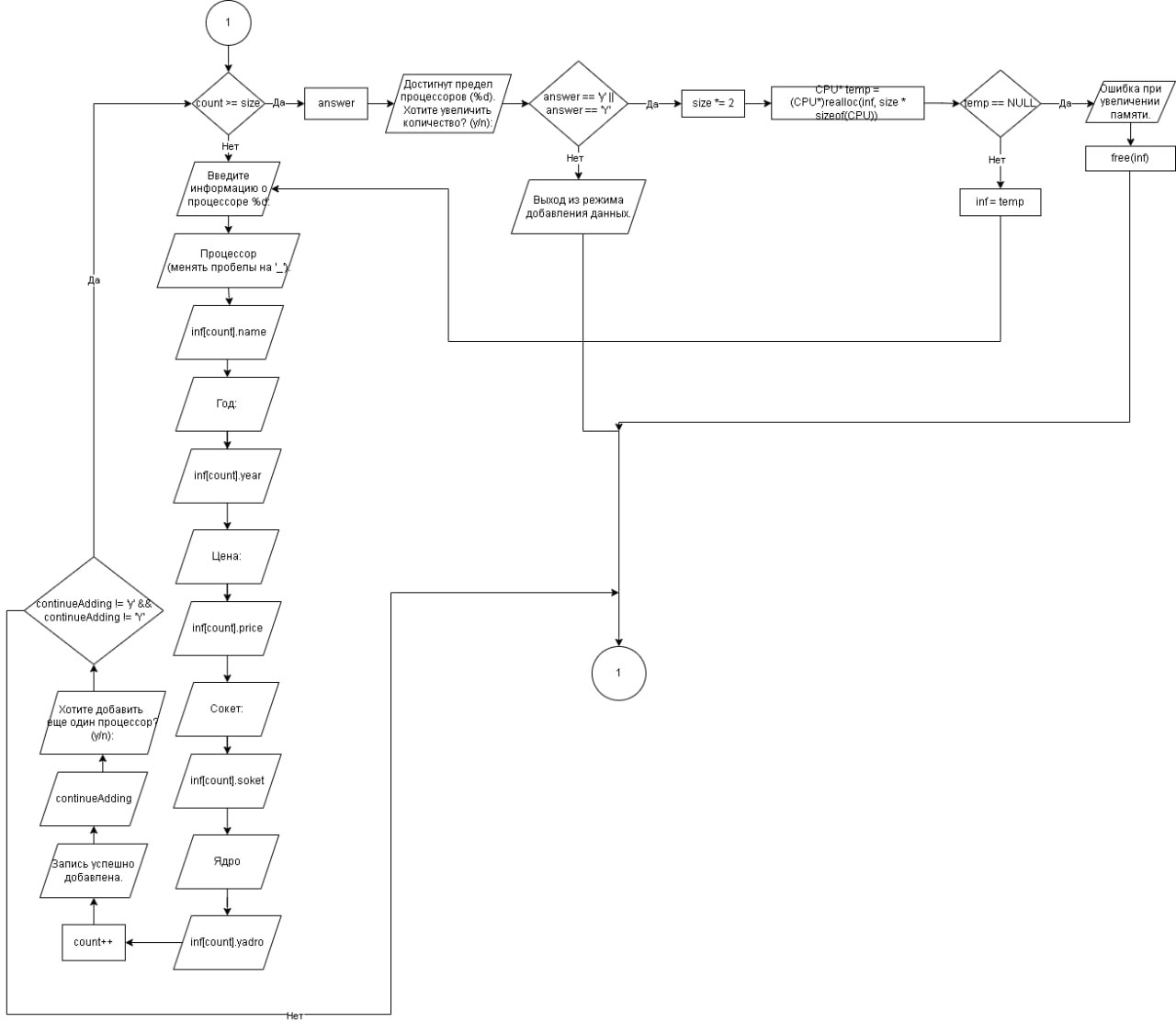


Рисунок 2.2 – Блок-схема функции main (2 часть)

int writefile(const char\* fname, CPU\* inf, int count)

Функция, предназначенная для инициализации массива процессоров. Она принимает массив процессоров, адрес размера массива и количество процессоров в массиве. Создаёт файл или добавляет новые данные по ходу использования и сохраняет данные в текстовый файл “CPU.txt”. Блок схема на рисунке 2.3.

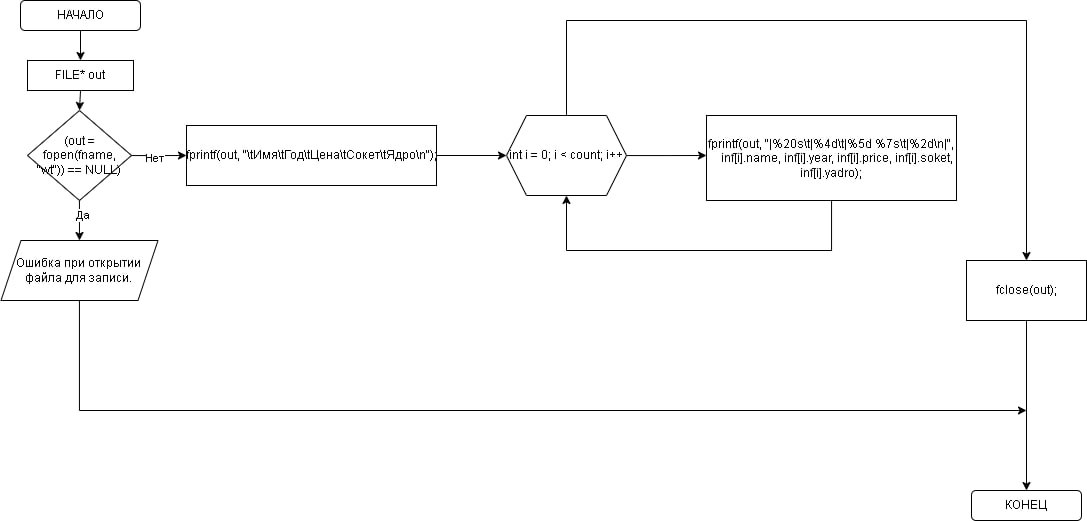


Рисунок 2.3 – Блок-схема функции writefile

CPU\* readfile(const char\* fname, CPU\* inf, int count)

Данная функция считывает массив структур с файла “CPU.txt” и выводит его на экран.

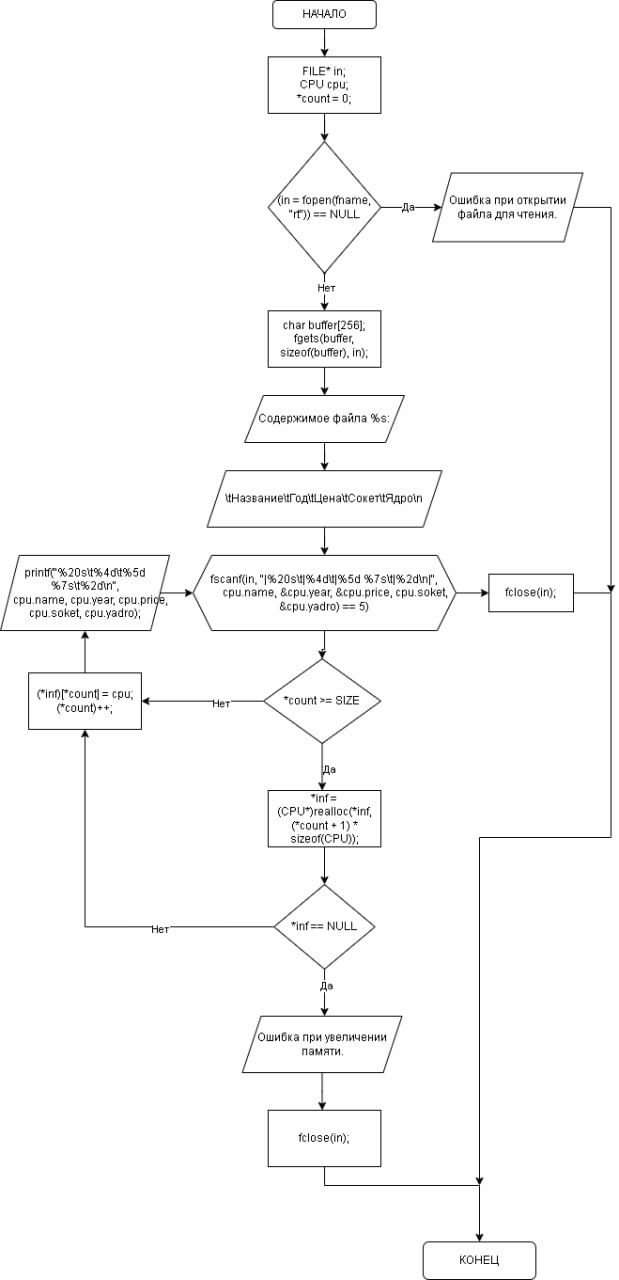


Рисунок 2.4 – Блок-схема функции readfile

CPU\* update(CPU\* inf, const char\* name, int count);

Функция изменяет процессор по названию, переданному ей. Она присваивает полям старого процессора новое значения полей процессора, при этом если какое-либо поле не будет иметь значения, то поле старого процессора не изменится. Возвращает изменённый процессор. Блок-схема функции на рисунке 2.5.

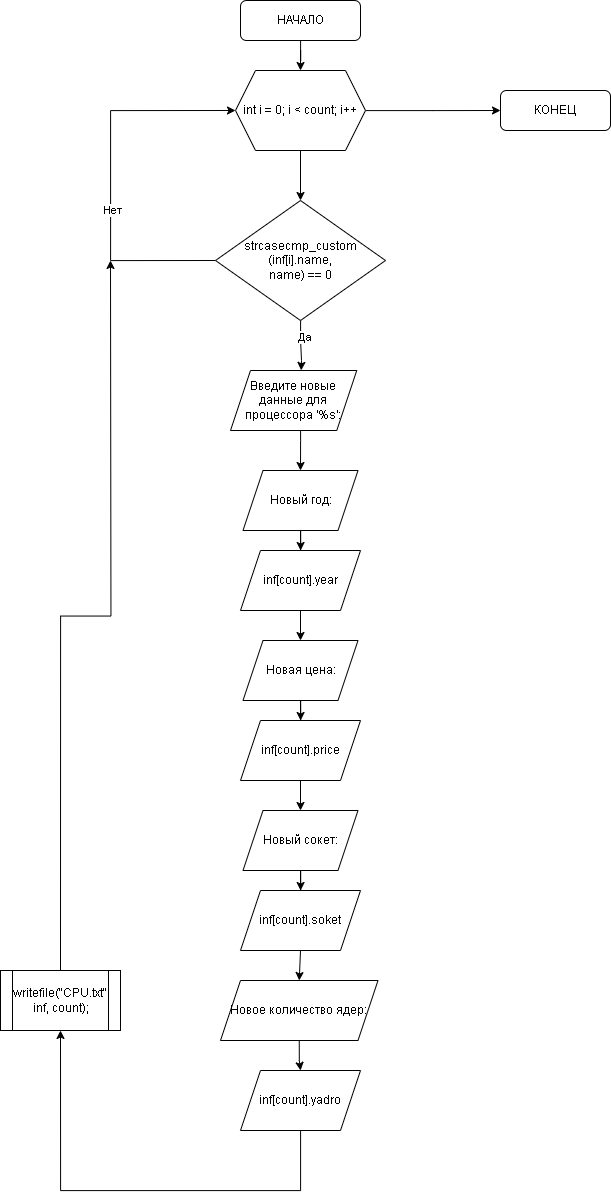


Рисунок 2.5 – Блок-схема функции update

CPU\* deleteCPU(Tab\* ptr\_tabs, int\* ptr\_size, int index\_del)

Функция, удаляющая процессор из списка, название которого в неё передаётся. Возвращает список без удалённого элемента. Блок-схема на рисунке 2.6.

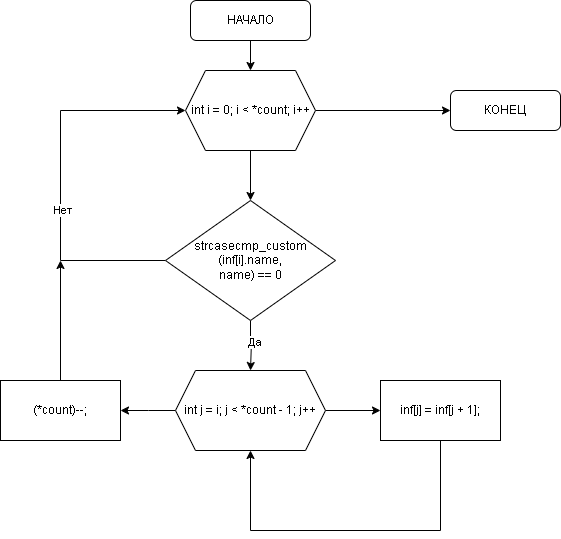


Рисунок 2.6 – Блок-схема функции deleteCPU

void print(Tab\* ptr\_tabs, int size)

Функция пробегается по каждой букве строки и превращает в строчный символ, чтобы название процессора, введённое в консоли считывалось и сравнивалось с названием в файле. Блок-схема изображена на рисунке 2.7.

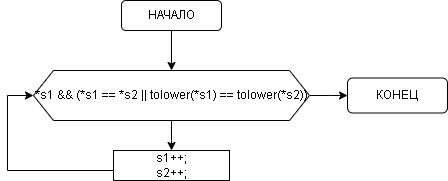


Рисунок 2.7 – Блок-схема функции strcasecmp\_custom

void searchCPU(const char\* fname, const char\* query)

Функция находит все процессоры с названием, введённым в консоль или сокетом, который она получает в аргументе «query» спрашиваемый в main. После выводит все данные о найденом/ых процессоре/ах Блок-схема на рисунке 2.8.

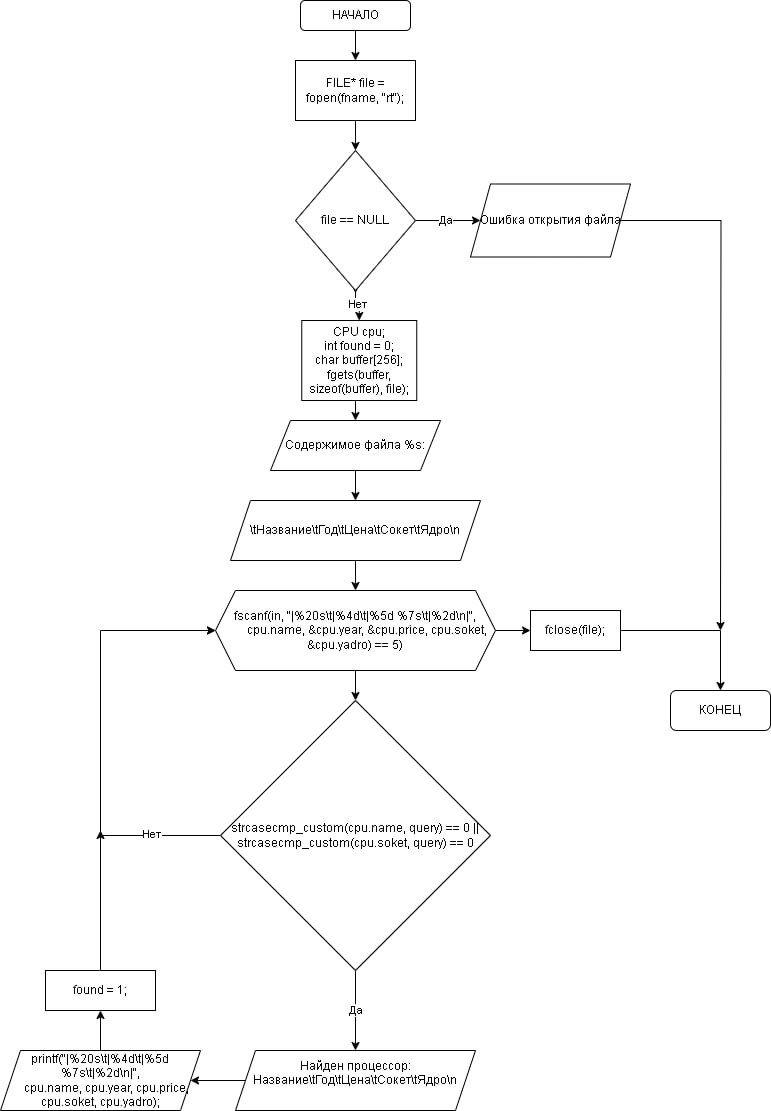


Рисунок 2.8 – Блок-схема функции searchCPU

CPU\* bubble\_sort\_by\_name(CPU\* cpus, int count)

CPU\* bubble\_sort\_by\_price(CPU\* cpus, int count)

Функции сортируют массив процессоров по названию от а до я и по цене в порядке возрастания, зависит от выбора в main(1 - по названию, 2 - по цене) которое определяется значением переменной «sort\_choice». За основу взята сортировка пузырьком. Возвращает новый отсортированный массив. Блок-схема изображена на рисунке 2.9, 2.10.

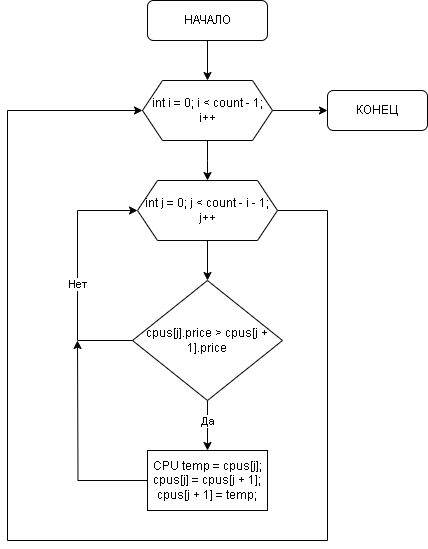
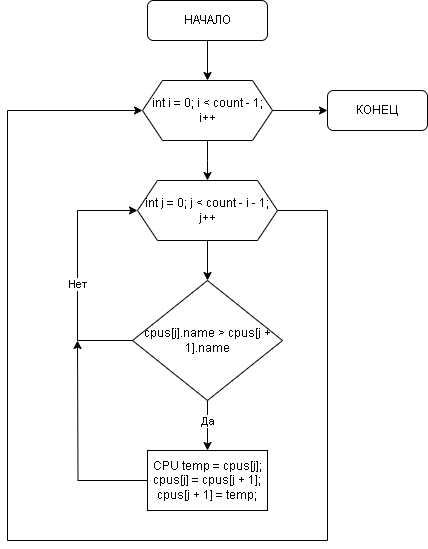


Рисунок 2.9, 2.10 – Блок-схемы функции bubble\_sort\_by\_price, bubble\_sort\_by\_name

## 3 Тестирование программы

Для начала протестируем вывод и добавление процессоров. Для этого введём новые данные процессоров в таблицу (рис. 3.1).

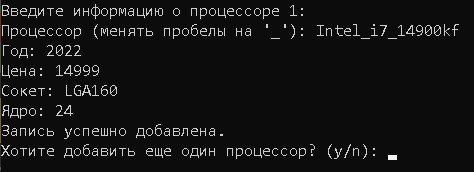


Рисунок 3.1 – Ввод данных в таблицу

Затем воспользовавшись функцией добавления заполним эту таблицу значениями из таблицы 1. Потом выведем массив снова и посмотрим результат (рис. 3.2).

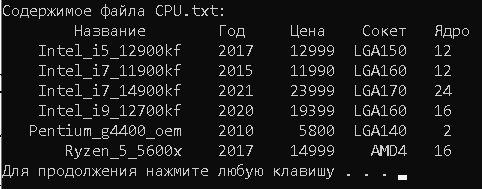


Рисунок 3.2 – Результат выполнения функции writefile

Затем проверим работу функции нахождения процессора по названию (searchCPU). Для этого в главном меню выберем шестой пункт, затем перед нами покажут все доступные процессоры и введём название процессора нужного нам. Результат на рисунке 3.3.

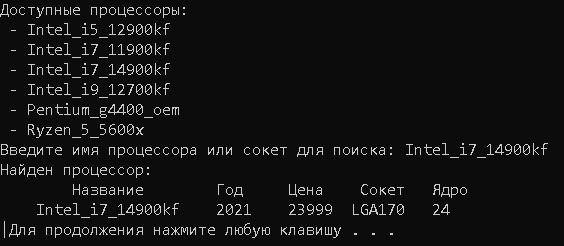


Рисунок 3.3 - Результат поиска по названию процессора

Потом для полного тестирования попробуем найти по названию сокета процессора. Результат на рисунке 3.4.

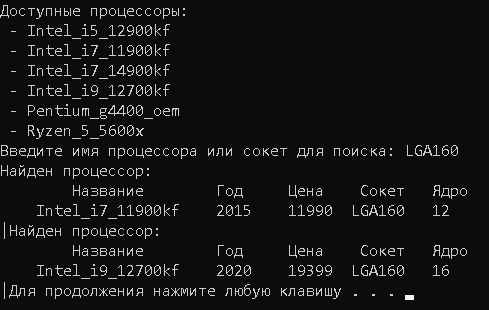


Рисунок 3.4 – Результат поиска по сокету

Далее будем тестировать функцию сортировки, для этого выберем пятый пункт и отсортируем функцию по цене. Результат на рисунке 3.5.

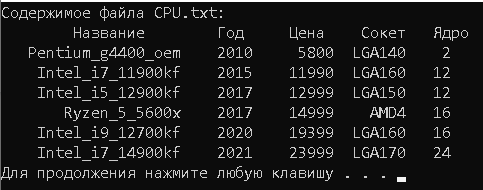


Рисунок 3.5 – Результат сортировки

Следом проверим редактирование полей процессоров. Чтобы это сделать нужно выбрать четвёртый пункт меню затем ввести название процессора. Появляется изменение всех полей старого процессора(Год, цена, Сокет, Ядра) Процесс на рисунке 3.6.

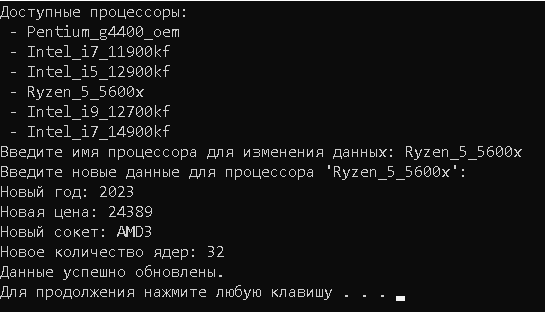


Рисунок 3.6 – Изменение массива процессора

Затем протестируем удаление планшета из массива. Выберем третий пункт меню и удалим “Pentium\_g4400\_oem”. Потом выведем массив на экран. Результат изображён на рисунке 3.7.



Рисунок 3.7 – Массив после удаления процессора

Листинг программы представлен в приложении 1.

# Заключение

В ходе данной курсовой работы был исследован процесс создания баз данных и интерфейса для работы с ними при помощи языка программирования C. Были осуществлены различные методы работы с массивом структур, такие как поиск добавление и удаление планшета из базы данных, вывод всех планшетов на экран, поиск планшета по заданному полю, сортировка базы данных по заданному полю и изменение планшета находящегося в базе данных.

Эта работы продемонстрировала, что базы данных – это полезный инструмент, позволяющий создавать виртуальную таблицу элементов и удобно с ней взаимодействовать.

# Список использованных источников

1. Керниган, Б.У. Язык программирования С / Б.У. Керниган, Д.М. Ритчи; Пер. с англ. В.Л. Бродовой. — М.: Вильямс, 2016. — 304 c.
2. Керниган, Б. Язык программирования C. 2-е изд. / Б. Керниган, Д.М. Ритчи. — М.: Вильямс, 2016. — 288 c.
3. Наумов, В.П., Перевозчиков, П.В., "Основы программирования на языке C и С++", издательство "Лань", 2019 г.
4. Дейтел, П. "Как программировать на C/C++", издательство "Питер", 2018 г.
5. Давыдов, В.Г. Д 13 «Программирование и основы алгоритмизации: Учеб. Пособие»/В.Г. Давыдов. — М.: Высш. шк., 2003. — 447 е.
6. Курипта О.В. Основы программирования и алгоритмизации: практикум / О.В.Курипта, О.В. Минакова, Д.К. Проскурин; Воронежский ГАСУ. – Воронеж, 2015. – 132 с.
7. Кауфман, В.Ш. Языки программирования. Концепции и принципы / В.Ш. Кауфман. — М.: ДМК, 2017. — 464 c.
8. Дорогов, В.Г. Основы программирования на языке С: Учебное пособие / В.Г. Дорогов, Е.Г. Дорогова; Под общ. ред. проф. Л.Г. Гагарина. — М.: ИД ФОРУМ, НИЦ ИНФРА-М, 2017. — 224 c.

# Приложение

# ПРИЛОЖЕНИЕ А – ЛИСТИНГ ПРОГРАММЫ

#define \_CRT\_SECURE\_NO\_WARNINGS // Отключение предупреждений

#define SIZE 2 // начальный размер массива

// Подключение стандартных библиотек

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

#include <ctype.h>

// Определение констант

// Описание полей структуры

typedef struct cpu {

char\* name[20]; // наименование

char\* soket[20]; // сокет

int price; // цена

int year; // год выпуска

int cores; // ядра

}; CPU

// Прототипы функций

// Функции работающие с файлом

int writefile(const char\* fname, CPU\* inf, int count); // сохранения текущего массива процессора в файл

CPU\* readfile(const char\* fname, CPU\*\* inf, int\* count); // чтение массива структур из файла и вывод в консоль

CPU\* update(CPU\* inf, const char\* name, int count); // редактирование процессора находящегося в массиве

CPU\* deleteCPU(CPU\* inf, const char\* name, int\* count); // удаление процессора из массива

int strcasecmp\_custom(const char\* s1, const char\* s2); // удаление планшета из массива

void searchCPU(const char\* fname, const char\* query); // Сравнение названий процессоров без учёта регистра

CPU\* bubble\_sort\_by\_name(CPU\* cpus, int count); // Сортировка по названию

CPU\* bubble\_sort\_by\_price(CPU\* cpus, int count); // Сортировка по цене

// Главная функция

int main() {

// Объявление переменных

int select; // Номер действия

int size = SIZE; // Объявление размера

int count = 0; // Счётчик количества процессоров

char fname[20] = "CPU.txt"; // Объявление названия файла

// Инициализация массива структур

CPU\* inf = (CPU\*)malloc(size \* sizeof(CPU));

// Локализация

setlocale(LC\_ALL, "RUS");

// Проверка выделения памяти

if (inf == NULL) {

printf("Ошибка выделения памяти.\n");

return 1;

}

// Чтение файла

readfile(fname, &inf, &count);

while (1) {

system("cls");

printf("\nВыберите действие:\n");

printf("Вывести данные (1)\n");

printf("Добавить запись в базу данных (2)\n");

printf("Удалить записи из базы данных (3)\n");

printf("Изменить данные в полях (4)\n");

printf("Сортировать данные (5)\n");

printf("Поиск процессора по имени или сокету (6)\n");

printf("Закончить работу (0)\n");

printf("Введите цифру для нужного действия: ");

scanf("%d", &select);

switch (select) {

case 1: { // вывод массива структур

system("cls");

readfile(fname, &inf, &count);

} break;

case 2: { // добавление

while (1) {

system("cls");

if (count >= size) {// проверка на переполнение

char answer;

printf("Достигнут предел процессоров (%d). Хотите увеличить количество? (y/n): ", size);

scanf(" %c", &answer);

if (answer == 'y' || answer == 'Y') {

size \*= 2; // увеличение размера в 2 раза

CPU\* temp = (CPU\*)realloc(inf, size \* sizeof(CPU));

if (temp == NULL) {

printf("Ошибка при увеличении памяти.\n");

free(inf);

return 1;

}

inf = temp;

}

else {

printf("Выход из режима добавления данных.\n");

break;

}

}

printf("Введите информацию о процессоре %d:\n", count + 1);

printf("Процессор (менять пробелы на '\_'): ");

scanf("%19s", inf[count].name);

printf("Год: ");

scanf("%d", &inf[count].year);

printf("Цена: ");

scanf("%d", &inf[count].price);

printf("Сокет: ");

scanf("%19s", inf[count].soket);

printf("Ядро: ");

scanf("%d", &inf[count].yadro);

count++;

if (!writefile(fname, inf, count)) {

printf("Ошибка при открытии файла.\n");

}

else {

printf("Запись успешно добавлена.\n");

}

char continueAdding;

printf("Хотите добавить еще один процессор? (y/n): ");

scanf(" %c", &continueAdding);

if (continueAdding != 'y' && continueAdding != 'Y') {

break;

}

}

} break;

case 3: {// удаление

system("cls");

printf("Доступные процессоры:\n");

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

printf(" - %s\n", inf[i].name);

}

char name[20];

printf("Введите имя процессора, который хотите удалить: ");

scanf("%19s", name);

if (deleteCPU(inf, name, &count)) {

printf("Процессор '%s' успешно удален.\n", name);

writefile(fname, inf, count);

}

else {

printf("Процессор с именем '%s' не найден.\n", name);

}

} break;

case 4: {// обновление данных

system("cls");

printf("Доступные процессоры:\n");

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

printf(" - %s\n", inf[i].name);

}

char name[20];

printf("Введите имя процессора для изменения данных: ");

scanf("%19s", name);

if (update(inf, name, count)) {

printf("Данные успешно обновлены.\n");

}

else {

printf("Процессор с именем '%s' не найден.\n", name);

}

} break;

case 5: {// сортировка

system("cls");

int sort\_choice; // переменная выбора сортировки

printf("Выберите способ сортировки:\n");

printf("1. По имени процессора\n");

printf("2. По цене\n");

printf("Введите цифру для выбора: ");

scanf("%d", &sort\_choice);

if (sort\_choice == 1) {

bubble\_sort\_by\_name(inf, count);

}

else if (sort\_choice == 2) {

bubble\_sort\_by\_price(inf, count);

}

else {

printf("Некорректный ввод.\n");

break;

}

writefile(fname, inf, count);

printf("Данные отсортированы.\n");

} break;

case 6: {// поиск процессора

system("cls");

printf("Доступные процессоры:\n");

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

printf(" - %s\n", inf[i].name);

}

char query[20];

printf("Введите имя процессора или сокет для поиска: ");

scanf("%19s", query);

searchCPU(fname, query);

} break;

case 0: {// завершение программы

free(inf);

printf("Выход из программы.\n");

return 0;

} break;

default: // Ошибка выбора

printf("Некорректный выбор. Пожалуйста, попробуйте еще раз.\n");

break;

}

system("pause");

}

free(inf); // освобождение памяти

return 0;

}

/\*

\* Функция записи процессора в файл.

\* Принимает массив планшетов, указатель на его размер и новый планшет.

\* Добавляет новый процессор в конец массива.

\* Возвращает 1 при успешном действие, 0 при ошибке.

\*/

int writefile(const char\* fname, CPU\* inf, int count) {

FILE\* out;

if ((out = fopen(fname, "wt")) == NULL) {

printf("Ошибка при открытии файла для записи.\n");

return 0;

}

fprintf(out, "\tИмя\tГод\tЦена\tСокет\tЯдро\n");

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

fprintf(out, "|%20s\t|%4d\t|%5d %7s\t|%2d\n|",

inf[i].name, inf[i].year, inf[i].price, inf[i].soket, inf[i].yadro);

}

fclose(out);

return 1;

}

/\*

\* Функция инициализации массива.

\* Принимает массив и указатель на его размер.

\* Читает данные из файла и записывает их в массив.

\* Возвращает массив процессоров.

\*/

CPU\* readfile(const char\* fname, CPU\*\* inf, int\* count) {

FILE\* in;

CPU cpu;

\*count = 0;

if ((in = fopen(fname, "rt")) == NULL) {

printf("Ошибка при открытии файла для чтения.\n");

return;

}

char buffer[256];

fgets(buffer, sizeof(buffer), in);

printf("Содержимое файла %s:\n", fname);

printf("\tНазвание\tГод\tЦена\tСокет\tЯдро\n");

while (fscanf(in, "|%20s\t|%4d\t|%5d %7s\t|%2d\n|",

cpu.name, &cpu.year, &cpu.price, cpu.soket, &cpu.yadro) == 5) {

if (\*count >= SIZE) {

\*inf = (CPU\*)realloc(\*inf, (\*count + 1) \* sizeof(CPU));

if (\*inf == NULL) {

printf("Ошибка при увеличении памяти.\n");

fclose(in);

}

}

(\*inf)[\*count] = cpu;

(\*count)++;

printf("%20s\t%4d\t%5d\t%7s\t%2d\n",

cpu.name, cpu.year, cpu.price, cpu.soket, cpu.yadro);

}

fclose(in);

return inf;

}

int strcasecmp\_custom(const char\* s1, const char\* s2) {

while (\*s1 && (\*s1 == \*s2 || tolower(\*s1) == tolower(\*s2))) {

s1++;

s2++;

}

return tolower(\*(unsigned char\*)s1) - tolower(\*(unsigned char\*)s2);

}

CPU\* update(CPU\* inf, const char\* name, int count) {

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

if (strcasecmp\_custom(inf[i].name, name) == 0) {

printf("Введите новые данные для процессора '%s':\n", inf[i].name);

printf("Новый год: ");

scanf("%d", &inf[i].year);

printf("Новая цена: ");

scanf("%d", &inf[i].price);

printf("Новый сокет: ");

scanf("%19s", inf[i].soket);

printf("Новое количество ядер: ");

scanf("%d", &inf[i].yadro);

writefile("CPU.txt", inf, count);

return 1;

}

}

return inf;

}

/\*

\* Функция удаления процессора.

\* Принимает массив процессоров, указатель на его размер и название удаляемого процессора.

\* Создаёт новый массив без удаляемого процессора.

\* Возвращает новый массив.

\*/

CPU\* deleteCPU(CPU\* inf, const char\* name, int\* count) {

for (int i = 0; i < \*count; i++) {

if (strcasecmp\_custom(inf[i].name, name) == 0) {

for (int j = i; j < \*count - 1; j++) {

inf[j] = inf[j + 1];

}

(\*count)--;

}

}

return inf;

}

void searchCPU(const char\* fname, const char\* query) {

FILE\* file = fopen(fname, "rt");

if (file == NULL) {

perror("Ошибка открытия файла");

return;

}

CPU cpu;

int found = 0;

char buffer[256];

fgets(buffer, sizeof(buffer), file);

while (fscanf(file, "|%20s\t|%4d\t|%5d %7s\t|%2d\n|",

cpu.name, &cpu.year, &cpu.price, cpu.soket, &cpu.yadro) == 5) {

if (strcasecmp\_custom(cpu.name, query) == 0 || strcasecmp\_custom(cpu.soket, query) == 0) {

printf("Найден процессор:\n");

printf("\tНазвание\tГод\tЦена\tСокет\tЯдро\n");

printf("|%20s\t|%4d\t|%5d %7s\t|%2d\n|",

cpu.name, cpu.year, cpu.price, cpu.soket, cpu.yadro);

found = 1;

}

}

fclose(file);

}

/\*

\* Функция сортировки.

\* Принимает массив процессоров, его размер и количество. Сортировка проводится по имени в порядке возрастания

\* За основу взята сортировка пузырьком.

\* Возвращает отсортированный массив.

\*/

CPU\* bubble\_sort\_by\_name(CPU\* cpus, int count) {

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

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

if (strcmp(cpus[j].name, cpus[j + 1].name) > 0) {

CPU temp = cpus[j];

cpus[j] = cpus[j + 1];

cpus[j + 1] = temp;

}

}

}

return cpus;

}

/\*

\* Функция сортировки.

\* Принимает массив процессоров, его размер. Сортировка проводится по цене в порядке возрастания

\* За основу взята сортировка пузырьком.

\* Возвращает отсортированный массив.

\*/

CPU\* bubble\_sort\_by\_price(CPU\* cpus, int count) {

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

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

if (cpus[j].price > cpus[j + 1].price) {

CPU temp = cpus[j];

cpus[j] = cpus[j + 1];

cpus[j + 1] = temp;

}

}

}

return cpus;

}