Министерство образования и науки Российской Федерации

Федеральное государственное образовательное учреждение высшего профессионального образования

«Московский авиационный институт»

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

2 семестр

Курсовой проект

По курсу «Алгоритмы и структуры данных»

Задание VI

Работ сдал: студент группы М8О-204Б-22

Филиппов Фёдор Иванович

Работу принял: преподаватель информатики

Потенко Максим Алексеевич

Москва, 2023

Оглавление

Цель работы

Разработать последовательную структуру данных для представления простейшей базы данных на файлах в СП Си в соответствии с заданным вариантом. Составить программу генерации внешнего нетекстового файла заданной структуры, содержащего представительный набор записей (15-20).

Действие по выборке данных из файла оформить в виде отдельной программы с параметрами запроса получаемыми из командной строки UNIX.

Задание

Содержимое и структура файла: Сведения о составе комплектующих личных ПЭВМ в студенческой группе: фамилия владельца, число и тип процессоров, объем памяти, тип видеоконтроллера и объем видеопамяти, число и ёмкость винчестеров, количество интегрированных контроллеров и внешних устройств, операционная система.

Действия: Для всех студентов, имеющих более одного компьютера, распечатать сведения о самом мощном из них.

Теоретическая справка

В данной курсовой работе мне следует написать несколько программ на языке Си, которые будут направлены на прямую работу с файлами. Главной задачей данной работы является создание простого варианта базы данных, которую можно будет дополнять, анализировать и т.д. Однако перед описанием программ по её обработке следует разобраться в том, каким образом какой-либо процесс операционной системы (в том числе наша программа) получает доступ к файлам и оперирует ими.

Дескриптор файла

Дескриптор файла (от лат. descriptor «описывающий») — это неотрицательное целое число, закреплённое за определённым потоком ввода-вывода. Когда создается новый поток ввода-вывода, ядро возвращает

процессу, создавшему поток ввода-вывода, его файловый дескриптор. Файл также является потоком записи и/или чтения информации.

В целом дескриптор файла можно представлять как некий идентификатор файла, данный некоторому процессу, который закрепляется за файлом и позволяет оперировать им даже если тот в последствие будет изменен, переименован или даже удалён. Таким образом, это гарантирует сохранность целостности данных.

Для хранения индексов всех открытых файлов в системе существует специальная таблица, ячейки которой обычно заполняются последовательно. Когда вы создаете новый файл или открываете существующий, ему присваивается номер. Следующий файл получает следующий номер по очередности — например, 50, 51, 52 и так далее. Однако в такой таблице всегда есть три неизменно зафиксированных дескриптора – это индексы 0, 1 и 2. Индекс 0 отвечает за стандартный поток ввода данных в программу (stdin), 1 – за стандартный поток вывода данных (stdout), а 2 – за стандартный поток сообщений об ошибках (stderr). Когда работа с файлом завершается, присвоенный ему дескриптор также освобождается и возвращается в область свободных номеров.

Мы разобрались, что дескрипторы файлов выполняют роль индексов таблицы дескрипторов, которая создается ядром для каждого процесса. Чаще всего процесс получает дескрипторы с помощью операций **open** и **creat**, а также путем наследования от родительского процесса.

Важным стоит отметить то, что в структуре системной таблицы открытых файлов также хранится смещение указателя в файле. Это исходит из того, что при выполнении операции чтения-записи система выполняет неявный сдвиг указателя внутри потока.

Например, при чтении или записи *x* байт указатель также будет перемещен на *x* байт. Однако сама программа, которая выполняется в пользовательском

пространстве, не работает с файлами непосредственно. Вместо этого, программа взаимодействует с ядром

операционной системы, чтобы выполнить операции с файлами.
Процесс работы с системным файлом через ядро операционной системы обычно включает следующие шаги:

1. Открытие файла: Процесс должен открыть файл, чтобы получить доступ к его
   содержимому и метаданным. Для этого процесс вызывает соответствующую
   системную функцию, предоставляемую операционной системой.

2. Авторизация и проверка прав доступа: При открытии файла операционная система проверяет права доступа процесса к файлу. Это включает проверку разрешений на чтение, запись и выполнение, а также проверку других
   ограничений доступа, установленных на файле и в операционной системе.

3. Чтение и запись данных: После успешного открытия файла процесс получает соответствующий дескриптор файла и может выполнять операции чтения и записи данных в файл. Он может использовать системные вызовы, чтобы передать данные ядру операционной системы, которое в свою очередь будет
   выполнять операции ввода-вывода с файлом.

4. Управление позицией в файле: Процесс может контролировать текущую позицию
   чтения или записи в файле с помощью операций перемещения указателя файла. Например, он может переместить указатель в начало файла, в конец файла или к определенной позиции.

5. Закрытие файла: По окончании работы с файлом процесс должен закрыть его, чтобы освободить ресурсы и уведомить операционную систему о завершении использования файла. Это позволяет другим процессам получить доступ к файлу и гарантирует целостность данных.

Во время выполнения этих операций ядро операционной системы обрабатывает системные вызовы и взаимодействует с файловой системой, чтобы выполнить требуемые операции с файлом. Ядро осуществляет контроль доступа, управляет позицией в файле, выполняет операции чтения и записи, а также обрабатывает другие операции, связанные с файлами.

Это значит, что процессы работают в своих собственных виртуальных адресных пространствах, и ядро операционной системы обеспечивает изоляцию и защиту данных между процессами, чтобы предотвратить несанкционированный доступ к файлам и обеспечить безопасность системы.

Файловый поток

Файловый ввод/вывод в языке программирования C позволяет работать с файлами, выполнять чтение данных из файлов (ввод) и запись данных в файлы (вывод). Для работы с файлами в C используются стандартные библиотечные функции, определенные в заголовочном файле <stdio.h>.

Вот основные функции, которые используются для файлового ввода/вывода в C:

1. fopen: Функция fopen используется для открытия файла. Она принимает два параметра: имя файла и режим открытия. Режимы открытия могут быть следующие: "r" (чтение), "w" (запись), "a" (добавление), "rb" (бинарное чтение), "wb" (бинарная запись) и другие. Функция возвращает указатель на файловую структуру типа FILE, которая используется для последующих операций

с файлом.

2. fprintf и fscanf: Функция fprintf форматирует и записывает данные в файл. Она принимает указатель на FILE и форматированную строку с аргументами. Функция fscanf считывает данные из файла в соответствии с указанным
форматом.

3. fseek: Функция fseek в языке C используется для установки позиции в файле. Она
позволяет перемещаться по файлу и устанавливать указатель на определенную позицию, чтобы выполнить операции чтения или записи. Прототип функции fseek выглядит следующим образом: int fseek(FILE *stream, long offset, int origin). Origin – это определение базовой позиции, относительно которой будет выполняться перемещение. Допустимые значения: SEEK_SET (0) - начало файла, SEEK_CUR (1) - текущая позиция в файле, SEEK_END (2) - конец файла.

4. fclose: Функция fclose закрывает файл, ранее открытый с помощью fopen. Она принимает указатель на файловую структуру FILE в качестве параметра. После закрытия файла все буферизованные данные записываются на диск, и связанные ресурсы освобождаются.

В языке C файловый дескриптор не доступен программисту в явном виде, а скрыт внутри файловой структуры FILE. Вместо этого, программист работает с файловыми указателями и вызывает соответствующие функции для выполнения тех или иных операций чтения/записи.

Аргументы командной строки

Аргумент командной строки — это информация, которая вводится в командной строке операционной системы вслед за именем программы. Чтобы принять аргументы командной строки, используются два специальных встроенных аргумента: argc и argv .

Параметр argc содержит количество аргументов в командной строке и является целым числом, причем он всегда не меньше 1, потому что первым аргументом считается имя программы. А параметр argv является указателем на массив указателей на строки. В этом массиве каждый элемент указывает на какой-либо аргумент командной строки. Все аргументы командной строки являются строковыми, поэтому преобразование каких бы то ни было чисел в нужный двоичный формат должно быть предусмотрено в программе при ее разработке.

В языке C оба этих параметра определены в функции main.

Используемое оборудование

ЭВМ iMac 2012 Late 2012 21.5’

Процессор Intel Core i5 4 ядра

ОП 16324 Мб

НМД 524288 Мб

Реализация программы

Структура файла с данными

Перед обсуждением структуры программы по работе с простейшей базой данной, следует рассмотреть формат хранения информации в файлах.

Для этого достаточно будет привести пример содержимого одного из таких файлов. По задумке, данные будут храниться в формате CSV (comma separated values) строка за строкой. Это обеспечит более простое чтение текстовых файлов программой:

Fedor,4,i9+i9+i5+i7,1024,1060,6144,2,2+2,3,Ubuntu

Masha,3,i3+i3+i7,2048,1050,2048,3,8+16+8,4,Windows8

Petya,1,i5,4096,1050,2048,2,16+2,3,Windows7

Fedor,3,i9+i7+i5,6144,1060,4096,3,16+1+32,4,Windows10

Vitaly,4,i7+i9+i3+i9,5120,1060,2048,2,16+16,3,Windows7

Ibragim,2,i7+i7,6144,1050,6144,2,16+16,5,Windows7

Vitaly,1,i3,6144,1080,2048,2,32+3,3,Windows8

То есть каждый отдельный компьютер хранится в отдельной строке. Поэтому, если у студента имеется несколько компьютеров, они всё равно будут записаны в разных строках.

Заметим также, что если компьютер студента включает в себя несколько однотипных элементов, они должны быть разделены символом «+».

Таким образом, чтение файла в дальнейшем будет достигаться посредством токенизации в процессе лексического анализа строки относительно разделителей – запятых и символов «+».

Программа и структуры

Основной частью всей работы является главная программа – main. Именно в ней происходит анализ содержимого файлов и сравнение компьютеров студентов. Однако следует отметить, что она базируется на структуре student и массиве student* s_list[MAX_STUDENTS]. Без них было бы трудно себе представить работу с базой данных.

Data

Как можно понять, данные о компьютере студента должны храниться в специальной структуре с предопределенными полями. Такой структурой служит Data, которая содержит в себе информацию о каждой спецификации отдельного ПК, а также хранит в себе хэш-таблицу с указателями на данные свойства. Описание структуры и её методов расположено в заголовочном файле data.h.

Рассмотрим же сами методы, относящиеся сюда: ими являются конструктор структуры, геттеры некоторых её свойств, чтение целой структуры из текста, а также из бинарного файла (об этом позже), соответственно запись структуры, пользовательский вывод, и наконец – деструктор.

```
typedef struct student {
    char* name;
    int cpu_num;
    char* cpu_type;
    int mem;
    char* gpu;
    int gpu_mem;
    int hdd_num;
    char* hdds;
    int devices;
    char* os;
} student;
```

Данные методы в полной мере позволяют оперировать классом. В таблице ниже представлено более полное описание этих функций:

- student* create_student(): Создает и возвращает указатель на структуру student. Выделяет память под поля структуры и инициализирует их значениями по умолчанию.

- void free_student(student* s): Освобождает память, выделенную под структуру student и ее поля.

- int student_read_txt(student* s, FILE* fd): Считывает данные студента из текстового файла, представленного указателем fd, и сохраняет их в

структуре student. Возвращает 0 при успешном считывании и -1 при ошибке.

- int student_write_txt(student* s, FILE* fd): Записывает данные студента в текстовый файл, представленный указателем fd. Возвращает 0 при успешной записи и -1 при ошибке.

- int student_read_bin(student* s, FILE* fd): Считывает данные студента из бинарного файла, представленного указателем fd, и сохраняет их в структуре student. Возвращает 0 при успешном считывании и -1 при ошибке.

- int student_write_bin(student* s, FILE* fd): Записывает данные студента в бинарный файл, представленный указателем fd. Возвращает 0 при успешной записи и -1 при ошибке.

- void print_student(student* s): Выводит информацию о студенте на стандартный вывод.

- void print_studlist(student** sl, int len): Выводит список студентов на стандартный вывод. sl - массив указателей на структуры student, len - длина массива.

- student** find_student(char* name, student** list, int list_len): Ищет студентов с заданным именем name в списке list длиной list_len. Возвращает массив указателей на студентов с совпадающими именами.

- int cmp_cpu(char* a, char* b): Сравнивает два типа процессоров a и b и возвращает 0, если первый тип лучше, и 1 в противном случае.

- int cmp_gpu(char* a, char* b): Сравнивает два типа графических процессоров a и b и возвращает 0, если первый тип лучше, и 1 в противном случае.

- int cmp_hdd(char* a, char* b): Сравнивает два типа жестких дисков a и b и возвращает 0, если первый тип лучше, и 1 в противном случае.

- int cmp_os(char* a, char* b): Сравнивает две операционные системы a и b и возвращает 0, если первая лучше, и 1 в противном случае.

- student* cmp_computers(student* list[]): Сравнивает компьютеры, представленные в массиве указателей list, и возвращает указатель на студента с наилучшей конфигурацией компьютера.

Идея сравнения компьютеров

Функция cmp_computers принимает массив указателей на структуры student в качестве входного параметра. Этот массив представляет собой список студентов, и каждый студент имеет характеристики своего компьютера, такие как тип процессора (cpu_type), тип видеокарты (gpu), тип жестких дисков (hdds) и операционную систему (os).

Цель этой функции - найти лучший компьютер среди студентов в списке. "Лучший" компьютер определяется как компьютер с лучшими характеристиками в каждой из следующих категорий: процессор, видеокарта, жесткие диски и операционная система.

Работа функции:

Если входной массив list пуст или первый элемент равен NULL, функция возвращает NULL, так как нет студентов для сравнения.
Инициализируется указатель best как первый студент в списке.

Затем функция начинает перебирать студентов в массиве, начиная со второго элемента (индекс 1).

Она сравнивает каждого студента с текущим "лучшим" студентом (best) в каждой из четырех категорий: процессор, видеокарта, жесткие диски и операционная система.

Если текущий студент в какой-либо из категорий не хуже лучшего студента, то цикл переходит к следующей итерации. Если текущий студент лучше во всех категориях, он становится новым "лучшим" студентом.

В конце цикла, best будет указывать на студента с лучшим компьютером среди всех студентов в списке.

Функция возвращает указатель на лучшего студента.

Подпрограммы

Функция tobin выполняет конвертацию данных из текстового файла (.txt) в бинарный файл (.bin). В этой функции:

Открывается текстовый файл для чтения и бинарный файл для записи.

1. Создается структура student, и считываются данные из текстового файла с помощью функции student_read_txt.

2. Считанные данные записываются в бинарный файл с использованием функции student_write_bin.

3. Процесс считывания и записи данных повторяется до тех пор, пока не будет достигнут конец текстового файла.

4. После завершения преобразования файлы закрываются, и данные успешно сконвертированы в бинарный формат.

Функция rand генерирует случайные данные для структуры student и записывает их в указанный текстовый файл. В этой функции:

1. Генерируются случайные значения для характеристик студента, такие как имя, количество процессоров, тип процессоров, объем памяти и т. д.

2. Сгенерированные данные форматируются в строку в формате CSV (значения, разделенные запятыми).

3. Строка данных записывается в указанный текстовый файл.
Шаги 1-3 повторяются заданное количество раз (количество студентов, указанное пользователем). В конце работы программы текстовый файл содержит данные о случайно сгенерированных студентах.

Распечатка программы

DATA.C

```
#include "data.h"


student* create_student() {
    student* s = (student*)malloc(sizeof(student));
    if (s == NULL) {
        perror("Ошибка создания студента");
        exit(1);
    }
    s->name = calloc(STAT_LEN, sizeof(char));
    s->cpu_type = calloc(STAT_LEN, sizeof(char));
    s->gpu = calloc(STAT_LEN, sizeof(char));
    s->hdds = calloc(STAT_LEN, sizeof(char));
    s->os = calloc(STAT_LEN, sizeof(char));
    // Инициализировать поля в NULL значения
    s->cpu_num = 0;
    s->mem = 0;
    s->gpu_mem = 0;
    s->hdd_num = 0;
    s->devices = 0;


    return s;
}


void free_student(student* s) {
    if (s == NULL) {
        return; // Нечего удалять, если студента не существует
```

```
    }

    // Освободить динамически распределяемые поля
    free(s->name);
    free(s->cpu_type);
    free(s->gpu);
    free(s->hdds);
    free(s->os);
    // Освобождаем саму структуру студента
    free(s);
}

// Функция для чтения записи о студенте из текстового файла
int student_read_txt(student* s, FILE* fd) {
    char buffer[MAX_LEN]; // Буфер для чтения линий из файла
    // Прочитать данные из CSV файла
    if (fgets(buffer, MAX_LEN, fd) == NULL) {
        return 0; // Конец файла или ошибка
    }
    // Спарсить в CSV линию в структуру студентов участников
    char* token = strtok(buffer, ",");
    s->name = strndup(token, STAT_LEN);
    token = strtok(NULL, ",");
    s->cpu_num = atoi(token);
    token = strtok(NULL, ",");
    s->cpu_type = strndup(token, STAT_LEN);
    token = strtok(NULL, ",");
    s->mem = atoi(token);
    token = strtok(NULL, ",");
```

```
    s->gpu = strndup(token, STAT_LEN);
    token = strtok(NULL, ",");
    s->gpu_mem = atoi(token);
    token = strtok(NULL, ",");
    s->hdd_num = atoi(token);
    token = strtok(NULL, ",");
    s->hdds = strndup(token, STAT_LEN);
    token = strtok(NULL, ",");
    s->devices = atoi(token);
    token = strtok(NULL, ",");
    s->os = strndup(token, STAT_LEN);
    return 1; // Успешно прочитать запись о студенте
}

// Функция для записи информации о студенте в текстовый файл
int student_write_txt(student* s, FILE* fd) {
    char* template = "%s,%d,%s,%d,%s,%d,%d,%s,%d,%s";
    if (s->os[strlen(s->os) - 1] != '\n') {
        template = "%s,%d,%s,%d,%s,%d,%d,%s,%d,%s\n";
    }
    if (fprintf(fd, template,
            s->name, s->cpu_num, s->cpu_type, s->mem, s->gpu,
            s->gpu_mem, s->hdd_num, s->hdds, s->devices, s->os) < 0) {
        return 0; // Ошибка при записи в файл
    }
    return 1; // Успешно записаны сведения о студенте
}

int student_read_bin(student *s, FILE* fd)
```

```
{
    if (fread(s->name, sizeof(char), STAT_LEN, fd) == 0) {
        return 0;
    };
    fread(&(s->cpu_num), sizeof(int), 1, fd);
    fread(s->cpu_type, sizeof(char), STAT_LEN, fd);
    fread(&(s->mem), sizeof(int), 1, fd);
    fread(s->gpu, sizeof(char), STAT_LEN, fd);
    fread(&(s->gpu_mem), sizeof(int), 1, fd);
    fread(&(s->hdd_num), sizeof(int), 1, fd);
    fread(s->hdds, sizeof(char), STAT_LEN, fd);
    fread(&(s->devices), sizeof(int), 1, fd);
    fread(s->os, sizeof(char), STAT_LEN, fd)  ;

    return 1;
}

int student_write_bin(student *s, FILE* fd)
{
    if (fwrite(s->name, sizeof(char), STAT_LEN, fd) == 0) {
        return 0;
    };
    fwrite(&(s->cpu_num), sizeof(int), 1, fd);
    fwrite(s->cpu_type, sizeof(char), STAT_LEN, fd);
    fwrite(&(s->mem), sizeof(int), 1, fd);
    fwrite(s->gpu, sizeof(char), STAT_LEN, fd);
    fwrite(&(s->gpu_mem), sizeof(int), 1, fd);
    fwrite(&(s->hdd_num), sizeof(int), 1, fd);
    fwrite(s->hdds, sizeof(char), STAT_LEN, fd);
```

```
    fwrite(&(s->devices), sizeof(int), 1, fd);
    fwrite(s->os, sizeof(char), STAT_LEN, fd);

    return 1;
}

// Функция для вывода записи о студенте
void print_student(student* s) {
    printf("    Name: %s\n", s->name);
    printf("    CPU Number: %d\n", s->cpu_num);
    printf("    CPU Type: %s\n", s->cpu_type);
    printf("    Memory [MB]: %d MB\n", s->mem);
    printf("    GPU: %s\n", s->gpu);
    printf("    GPU Memory [MB]: %d MB\n", s->gpu_mem);
    printf("    Number of HDDs: %d\n", s->hdd_num);
    printf("    HDD sizes [TB]: %s\n", s->hdds);
    printf("    Number of Devices: %d\n", s->devices);
    printf("    Operating System: %s\n", s->os);
}

void print_studlist(student** sl, int len) {
    for (int i = 0; i < len; i++) {
        printf("===\n");
        print_student(sl[i]);
    }
}

student** find_student(char* name, student** list, int list_len) {
    student** result = (student**)malloc(MAX_STUDENTS * sizeof(student*));
```

```
    int top = 0;

    for (int i = 0; i < list_len; i++) {
        if (list[i] == NULL) {
            break;
        }
        if (strncmp(list[i]->name, name, MAX_LEN) == 0) {
            result[top] = list[i];
            top++;
        }
    }

    result[top] = NULL;
    return top == 0 ? NULL : result;
}

int cmp_cpu(char* a, char* b) {
    int res1, res2;
    char* c = a;
    while (c < a + strlen(a)) {
        if (!isdigit(*c)) {
            c++;
            continue;
        }
        res1 += *c - '0';
        c++;
    }
    c = b;
    while (c < b + strlen(b)) {
```

```
        if (!isdigit(*c)) {
            c++;
            continue;
        }
        res2 += *c - '0';
        c++;
    }
    return res1 > res2 ? 0 : 1;
}

int cmp_gpu(char* a, char* b) {
    int res1, res2;
    char* c = a;
    while (c < a + strlen(a)) {
        if (!isdigit(*c)) {
            c++;
            continue;
        }
        res1 += *c - '0';
        c++;
    }
    c = b;
    while (c < b + strlen(b)) {
        if (!isdigit(*c)) {
            c++;
            continue;
        }
        res2 += *c - '0';
        c++;
```

```
    }
    return res1 > res2 ? 0 : 1;
}

int cmp_hdd(char* a, char* b) {
    int res1, res2;
    char* c = a;
    while (c < a + strlen(a)) {
        if (!isdigit(*c)) {
            c++;
            continue;
        }
        res1 += *c - '0';
        c++;
    }
    c = b;
    while (c < b + strlen(b)) {
        if (!isdigit(*c)) {
            c++;
            continue;
        }
        res2 += *c - '0';
        c++;
    }
    return res1 > res2 ? 0 : 1;
}

int cmp_os(char* a, char* b) {
    int res1, res2;
```

```
if (strcmp(a, "Windows 7") == 0) {
   res1 = 1;
}
else if (strcmp(a, "Windows 8") == 0) {
   res1 = 2;
}
else if (strcmp(a, "Windows 10") == 0) {
   res1 = 3;
}
else if (strcmp(a, "Ubuntu") == 0) {
   res1 = 4;
}
else {
   return -1;
}

if (strcmp(a, "Windows 7") == 0) {
   res2 = 1;
}
else if (strcmp(a, "Windows 8") == 0) {
   res2 = 2;
}
else if (strcmp(a, "Windows 10") == 0) {
   res2 = 3;
}
else if (strcmp(a, "Ubuntu") == 0) {
   res2 = 4;
}
else {
```

```
        return -1;
    }

    return res1 > res2 ? 0 : 1;
}

student* cmp_computers(student* list[]) {
    if (list[0] == NULL) {
        return NULL;
    }
    student* best = list[0];
    for (int i = 1; list[i] != NULL; i++) {
        student* pc1 = best;
        student* pc2 = list[i];
        if (pc2 == NULL) {
            break;
        }
        int c1 = 0, c2 = 0;

        cmp_cpu(pc1->cpu_type, pc2->cpu_type) == 0 ? c1++ : c2++;
        pc1->mem > pc2->mem ? c1++ : c2++;
        cmp_gpu(pc1->cpu_type, pc2->cpu_type) == 0 ? c1++ : c2++;
        pc1->gpu_mem > pc2->gpu_mem ? c1++ : c2++;
        cmp_hdd(pc1->cpu_type, pc2->cpu_type) == 0 ? c1++ : c2++;
        pc1->devices > pc2->devices ? c1++ : c2++;
        cmp_os(pc1->cpu_type, pc2->cpu_type) == 0 ? c1++ : c2++;

        if (c1 < c2) {
            best = pc2;
```

```
        }
    }

    return best;
}
```

DATA.H

```
#ifndef DATA
#define DATA

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include <ctype.h>

#define MAX_LEN 256
#define STAT_LEN 64

#define MAX_STUDENTS 50


typedef struct student {
    char* name;
    int cpu_num;
    char* cpu_type;
    int mem;
    char* gpu;
    int gpu_mem;
```

```
    int hdd_num;
    char* hdds;
    int devices;
    char* os;
} student;

student* create_student();

void free_student(student* s);

int student_read_txt(student* s, FILE* fd);

int student_write_txt(student* s, FILE* fd);

int student_read_bin(student* s, FILE* fd);

int student_write_bin(student* s, FILE* fd);

void print_student(student* s);

void print_studlist(student** sl, int len);

// Функция для нахождения студентов в заданном списке
student** find_student(char* name, student** list, int list_len);

// Функция для сравнения двух процессоров и return 0, если первый лучше, в ином случае -1
int cmp_cpu(char* a, char* b);
```

```
// Функция для сравнения двух видеокарт и return 0, если первый лучше, в ином случае - 1
int cmp_gpu(char* a, char* b);

// Функция для сравнения двух твердотельных накопителей и return 0, если первый лучше, в ином случае - 1
int cmp_hdd(char* a, char* b);

// Функция для сравнения двух ОС и return 0, если первый лучше, в ином случае - 1
int cmp_os(char* a, char* b);

// Функция для сравнения двух компьютеров по их комплектующим и вернуть лучший из данных
student* cmp_computers(student* list[]);



#endif
```

MAIN.C

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>

#include "data.h"

// Функция для удаления студента из массива
bool delete_student(char* name, student* s_list[], int* list_size) {
    int i, j;
    for (i = 0; i < *list_size; i++) {
```

```
        if (s_list[i] != NULL && strcmp(s_list[i]->name, name) == 0) {

            free_student(s_list[i]);

            for (j = i; j < *list_size - 1; j++) {
                s_list[j] = s_list[j + 1];
            }
            // Уменьшение размеров списка
            (*list_size)--;
            // Изменить значение последнего элемента на NULL
            s_list[*list_size] = NULL;
            return true; // Студент найден и удален, выходим из функции
        }
    }
    return false;
}

int main(int argc, char **argv) {
    if (argc != 2 || argv[1] == NULL) {
        printf("Использование: lab <файл>\n");
        exit(1);
    }

    bool bin = false;

    char* ext = strrchr(argv[1], '.');
    if (ext == NULL || strcmp(ext, ".txt") == 0) {
        printf("Работа с .txt файлом.\n");
    }
```

```
else if (ext != NULL && strcmp(ext, ".bin") == 0) {
    bin = true;
    printf("Работа с .bin файлом.\n");
}
else {
    perror("Неверное расширение файла\n");
    exit(1);
}

FILE* fd;

student* s_list[MAX_STUDENTS];
int list_size;

if (bin) {
    fd = fopen(argv[1], "rb");
    if (fd == NULL) {
        perror("Ошибка открытия файла\n");
        exit(1);
    }
    for (list_size = 0; true; list_size++) {
        student* s = create_student();
        if (student_read_bin(s, fd) == 0) {
            break;
        }
        s_list[list_size] = s;
    }
} else {
    fd = fopen(argv[1], "r");
```

```
    if (fd == NULL) {
        perror("Ошибка открытия файла\n");
        exit(1);
    }
    for (list_size = 0; true; list_size++) {
        student* s = create_student();
        if (student_read_txt(s, fd) == 0) {
            break;
        }
        s_list[list_size] = s;
    }
}

int value;
int choice = 0;

while (choice != 5) {

    printf("1. Добавить студента\n");
    printf("2. Удалить студента\n");
    printf("3. Показать список\n");
    printf("4. Найти лучший компьютер студента\n");
    printf("5. Выйти\n");

    scanf("%d", &choice);

    switch (choice) {
        case 1:
        {
```

```
student* s = create_student();
printf("Введите имя: ");
scanf("%s", s->name);
printf("Введите кол-во процессоров: ");
scanf("%d", &(s->cpu_num));
printf("Введите процессоры (пр: 'i5+i7'): ");
scanf("%s", s->cpu_type);
printf("Введите кол-во памяти (Мб): ");
scanf("%d", &(s->mem));
printf("Введите видеокарту (пр: 1080): ");
scanf("%s", s->gpu);
printf("Введите кол-во видеопамяти: ");
scanf("%d", &(s->gpu_mem));
printf("Введите кол-во HDD: ");
scanf("%d", &(s->hdd_num));
printf("Введите размеры HDD (пр: 2+4): ");
scanf("%s", s->hdds);
printf("Введите кол-во периф. устройств: ");
scanf("%d", &(s->devices));
printf("Введите операционную систему (без пробела): ");
scanf("%s", s->os);
printf("\n");
s_list[list_size++] = s;
fclose(fd);
if (bin) {
   fd = fopen(argv[1], "ab");
   student_write_bin(s, fd);
} else {
   fd = fopen(argv[1], "a");
```

```
            student_write_txt(s, fd);
        }
        fclose(fd);
        if (bin) {
            fd = fopen(argv[1], "rb");
            student_write_bin(s, fd);
        } else {
            fd = fopen(argv[1], "r");
            student_write_txt(s, fd);
        }
        break;
    }
    case 2:
    {
        char name[STAT_LEN];
        printf("Введите имя: ");
        scanf("%s", name);
        while (delete_student(name, s_list, &list_size)) {}

        FILE* tmp = fopen("temp", "w");
        for (int i = 0; i < list_size; i++) {
            if (bin) {
                student_write_bin(s_list[i], tmp);
            } else {
                student_write_txt(s_list[i], tmp);
            }
        }
        fclose(tmp);
        fclose(fd);
```

```
    remove(argv[1]);
    rename("temp", argv[1]);
    if (bin) {
        fd = fopen(argv[1], "rb");
    } else {
        fd = fopen(argv[1], "r");
    }
    printf("Студент удален из списка\n");
    break;
}
case 3:
    print_studlist(s_list, list_size);
    break;
case 4:
{
    char name[STAT_LEN];
    printf("Введите имя: ");
    scanf("%s", name);
    student** found = find_student(name, s_list, list_size);
    if (found == NULL) {
        printf("Студент не найден\n");
        break;
    }
    student* best = cmp_computers(found);
    printf("Лучший компьютер %s:\n", name);
    print_student(best);
    free(found);
    break;
}
```

```
            case 5:
                // Просто выходим
                break;

        }
    }

    // Удаляем студентов из памяти
    for (int i = 0; i < list_size; i++) {
        if (s_list[i] == NULL) {
            continue;
        }
        free_student(s_list[i]);
    }

    return 0;
}
```

RAND.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "data.h"

#define BUF_SIZE 64
#define MAX_STUD 30

char* random_student() {
```

```
    char* cpus[] = { "i3", "i5", "i7", "i9" };
    char* oss[] = { "Windows7", "Windows8", "Windows10", "Ubuntu" };
    char* gpus[] = { "1080", "1070", "1060", "1050" };
    char* names[] = { "Masha", "Petya", "Sergei", "Vitaly", "Fedor", "Ibragim", "Nikita",
"Maksim", "Diana" };
    int hdds[] = { 1, 2, 3, 4, 8, 16, 32 };

    // Случайным образом выбираются значения
    char* name = names[rand() % 8];
    int cpu_num = 1 + rand() % 4; // Случайное число процессоров (от 1 до 4)
    char cpu_types[MAX_LEN] = "";
    for (int i = 0; i < cpu_num; i++) {
        strcat(cpu_types, cpus[rand() % 4]);
        if (i < cpu_num - 1) {
            strcat(cpu_types, "+");
        }
    }
    int mem = (1 + rand() % 8) * 1024; // Случайный размер оперативной памяти кусками по
1024
    char* gpu = gpus[rand() % 4];
    int gpu_mem = (1 + rand() % 6) * 1024; // Случайный размер видео памяти кусками по
1024
    int hdd_num = 1 + rand() % 3;
    char hdd_sizes[MAX_LEN] = "";
    for (int i = 0; i < hdd_num; i++) {
        char hdd_size[BUF_SIZE];
        sprintf(hdd_size, "%d", hdds[rand() % 7]);
        strcat(hdd_sizes, hdd_size);
        if (i < hdd_num - 1) {
            strcat(hdd_sizes, "+");
```

```
        }
    }
    int devices = 2 + rand() % 5;
    char* os = oss[rand() % 4];

    // Создание строки в CSV формате
    char* str = (char*)malloc(MAX_LEN);
    snprintf(str, MAX_LEN, "%s,%d,%s,%d,%s,%d,%d,%s,%d,%s\n", name, cpu_num,
cpu_types, mem, gpu, gpu_mem, hdd_num, hdd_sizes, devices, os);

    return str;
}

int main(int argc, char **argv) {
    if (argc != 2) {
        perror("Использование: rand.out <имя_файла>\n");
        exit(1);
    }
    FILE* fp;
    fp = fopen(argv[1], "w");
    if (fp == NULL) {
        perror("Файл не найден\n");
        exit(1);
    }

    // Создание сида для генерации случайных чисел
    srand(time(NULL));

    int n = 1 + rand() % MAX_STUD;
```

```
    for (int i = 0; i < n; i++) {
        char* student_data = random_student();
        fprintf(fp, "%s", student_data);
        free(student_data);
    }

    fclose(fp);
    return 0;
}
```

TOBIN.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "data.h"

int main(int argc, char **argv) {
    if (argc != 3) {
        perror("Использование: tobin.out <*.txt> <*.bin>\n");
        exit(1);
    }

    FILE* txt = fopen(argv[1], "r");
    if (txt == NULL) {
        perror("Файл не найден");
        return 1;
    }
```

```
    FILE* bin = fopen(argv[2], "wb");
    if (bin == NULL) {
        perror("Ошибка открытия файла");
        fclose(txt);
        return 1;
    }

    student* s = create_student();

    while (student_read_txt(s, txt)) {
        printf("|");
        if (student_write_bin(s, bin) == 0) {
            perror("Ошибка записи");
            break;
        }
    }

    free_student(s);

    fclose(txt);
    fclose(bin);

    printf("\nФайл конвертирован.\n");
    return 0;
}
```

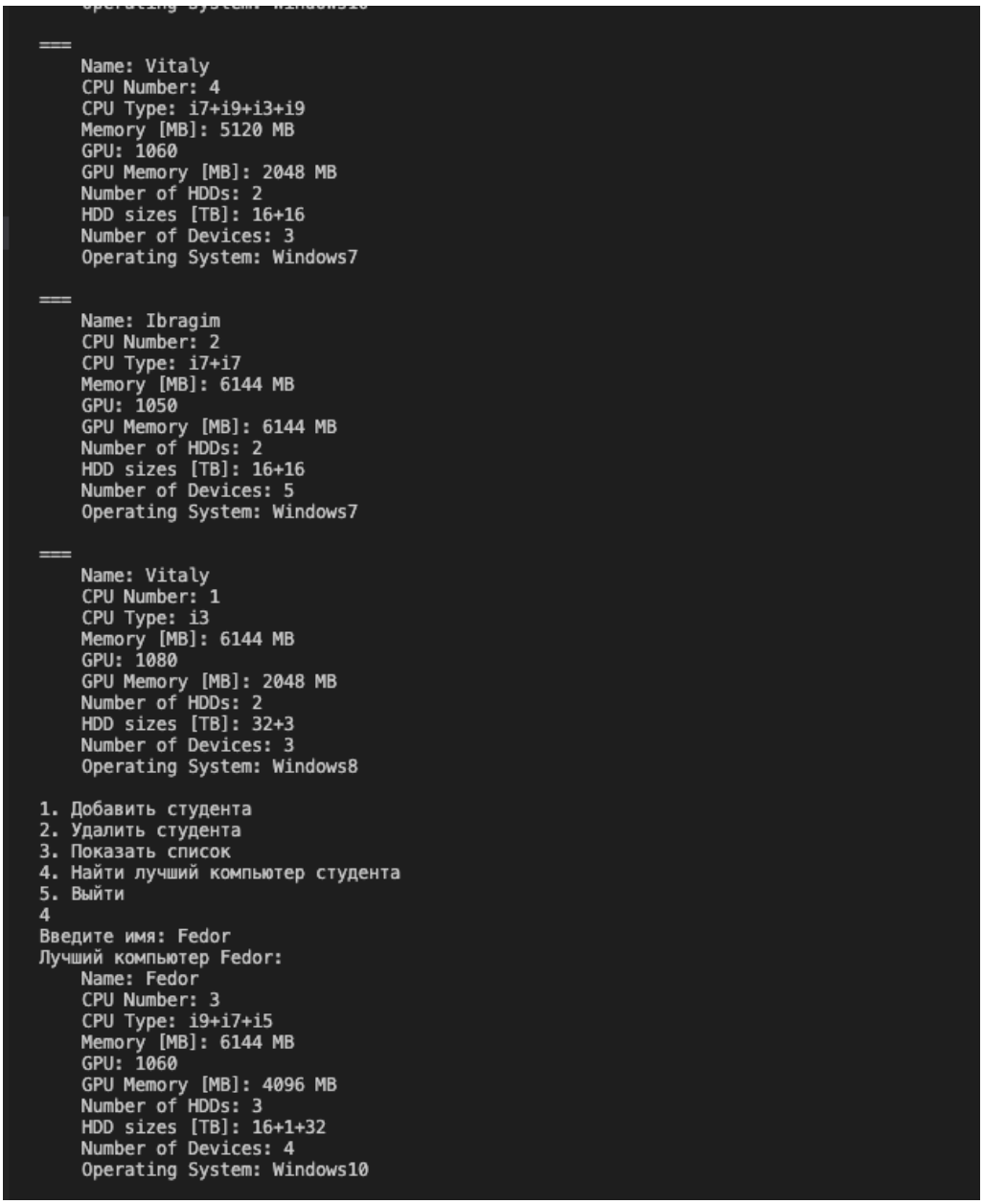
Результат работы программы

```
fedorkolya@iMac-fedor kp6 % ./lab haha.txt
Работа с .txt файлом.
1. Добавить студента
2. Удалить студента
3. Показать список
4. Найти лучший компьютер студента
5. Выйти
3
===
    Name: Fedor
    CPU Number: 4
    CPU Type: i9+i9+i5+i7
    Memory [MB]: 1024 MB
    GPU: 1060
    GPU Memory [MB]: 6144 MB
    Number of HDDs: 2
    HDD sizes [TB]: 2+2
    Number of Devices: 3
    Operating System: Ubuntu

===
    Name: Masha
    CPU Number: 3
    CPU Type: i3+i3+i7
    Memory [MB]: 2048 MB
    GPU: 1050
    GPU Memory [MB]: 2048 MB
    Number of HDDs: 3
    HDD sizes [TB]: 8+16+8
    Number of Devices: 4
    Operating System: Windows8

===
    Name: Petya
    CPU Number: 1
    CPU Type: i5
    Memory [MB]: 4096 MB
    GPU: 1050
    GPU Memory [MB]: 2048 MB
    Number of HDDs: 2
    HDD sizes [TB]: 16+2
    Number of Devices: 3
    Operating System: Windows7

===
    Name: Fedor
    CPU Number: 3
    CPU Type: i9+i7+i5
    Memory [MB]: 6144 MB
    GPU: 1060
    GPU Memory [MB]: 4096 MB
    Number of HDDs: 3
    HDD sizes [TB]: 16+1+32
    Number of Devices: 4
    Operating System: Windows10
```



Вывод

Благодаря данной курсовой работе я более ясно разобрался в том, как работать с системными файлами посредством программ, в частности, написанных на языке программирования Си.

В этой работе мне пришлось познакомиться с фундаментальными понятиями о файловом дескрипторе, потоках ввода\вывода. Теперь я имею представление о том, как система предоставляет процессам файлы для последующей работы с ними.

Благодаря этому я разработал собственную простейшую базу данных, основанную на текстовых и бинарных файлах, которая обрабатывается несколькими программами, написанными на Си. Таким образом, применив полученные знания на практике, я еще лучше осознал как процессы выполняют операции чтения, записи и обработки данных.

Полученные знания и навыки безусловно будут полезны мне в будущем. Например, они могут пригодиться при разработке приложений, работающих с файловой системой, а также при работе с другими системными ресурсами. Понимание основ работы с файлами позволит мне более эффективно управлять данными, обеспечивать их безопасность и оптимальную организацию.

Использованные источники

1. https://ru.wikipedia.org/wiki/Файловый_дескриптор
2. https://timeweb.com/ru/community/articles/chto-takoe-faylovyy-deskriptor-prostymi-slovami
3. https://ravesli.com/urok-110-argumenty-komandnoj-stroki/#:~:text=Аргументы%20командной%20строки%20—%20это,предоставлять%20входные%20данные%20для%20программы
4. https://metanit.com/cpp/tutorial/8.2.php
5. https://learn.microsoft.com/ru-ru/windows/win32/fileio/file-streams