Задача:

Разработать последовательную структуру данных для представления простейшей базы данных (здесь и далее – БД) на файлах СП Си. Составить программу генерации внешнего нетекстового файла заданной структуры, содержащего минимальный представительный набор данных (не менее 10 записей). Распечатать содержимое файла в виде таблицы. Выполнить над файлом заданное действие и распечатать результат. Действие по выборке данных из файла оформить в виде отдельной программы.

База данных содержит сведения о составе комплектующих личных ПЭВМ в студенческой группе: фамилия владельца, число и тип процессоров, объём памяти, тип видеоконтроллера и объём видеопамяти; тип, число и ёмкость винчестеров, операционная система.

Вариант 4:

Отпечатать список студентов, компьютеры которых нуждаются в апгрейде (более p устройств).

Использование программы:

Программа собирается командой «*make*» в корне проекта.

Исполняемый файл	Назначение
generate	Наполнение БД ранее заготовленными образцами
print	Вывод содержимого БД
execute	Вывод содержимого БД, удовлетворяющего условию варианта

Типичный сценарий использования программы – выполнить парсинг Яндекс.Маркета через get_reference, выполнить *generate* для создания БД, после чего изучить её содержимое с помощью *print* и *execute*.

Структура программы:

Файлы	Значение
structure.h	Реализация структуры БД
io.h io.c	Организация ввода, вывода и печати файлов
owner.h owner.c	Реализация списка владельцев и использования этого списка
get_reference.py	Парсинг Яндекс.Маркета для получения эталонных значений для

	выполнения задания варианта
generate.c	Создание бинарного файла БД из текстового
print.c	Печать данных из бинарного файла
execute.c	Выполнение задания варианта

structure.h:

typedef struct CPU;
typedef struct GPU;
typedef struct HDD;
typedef struct Computer;
typedef struct Reference;

В структурах CPU, GPU, HDD хранятся характеристики процессора, видеокарты и жесткого диска соответственно. В структуре Computer хранится информация, относящаяся к владельцу, и характеристики компьютера. В структуре Reference хранятся эталонные значения характеристик компьютера, полученные парсингом 200 самых популярных моделей компьютеров из Яндекс.Маркета.

io.h, io.c:

uint32_t computer_read_txt(Computer *computer, FILE *in);

Чтение данных из текстового файла.

uint32_t computer_read_bin(Computer *computer, FILE *in);

Чтение данных из двоичного файла.

void computer_write_bin(Computer *computer, FILE *out);

Запись данных в двоичный файл.

void computer_print(Computer *computer);

Печать данных.

uint32_t reference_read_txt(Reference *reference, FILE *in);

Чтение данных, полученных в результате парсинга.

owner.h, owner.c:

typedef struct NodeOwner;
typedef struct ListOwner;

Структура списка владельцев представляет собой простой линейный односвязный список.

ListOwner *list_create();

Создание списка владельцев.

void list_add(ListOwner *next_owner, Computer *computer);

Добавление владельца в список. Запись добавляется в начало списка.

void list_destroy(ListOwner *list);

Удаление всего списка.

void list_print(ListOwner *list);

Печать списка. Стоит заметить, что список владельцев печатается в обратном порядке в сравнении с тем, как владельцы расположены в БД.

generate.c:

В этом модуле генерируется БД. На вход подается 2 файла: первый – текстовый файл, из которого зачитываются данные, второй – двоичный файл, в который записываются данные.

print.c:

В этом модуле печатается БД. На вход подается 1 файл – двоичный файл, в котором хранится БД.

execute.c:

В этом модуле выполняется задание варианта. На вход подается двоичный файл с БД и параметр, по которому производится проверка соответствия. Далее каждая запись из БД сравнивается с эталонным значением, и если компьютер нуждается в апгрейде, запись из БД копируется в список владельцев. После прохода по всей БД, количество записей в списке владельцев сравнивается с параметром. Если параметр больше, то слишком мало компьютеров нуждается в апгрейде и печать списка не производиться, программа завершается. Если параметр меньше, то печатается список владельцев.

Вывод:

Несмотря на пригодность, у БД есть существенные недостатки, в основном обусловленные смещением баланса удобства в сторону разработчика БД, а не её пользователя.

Среди минусов так же стоит отметить отсутствие индексации, которая позволила бы осуществлять более быстрый поиск по ключам. Сейчас, для того, чтобы выполнить задание варианта, необходимо каждый раз проходить по всей базе, а с индексами такой проход осуществлялся бы однократно при генерации БД. Для ее реализации необходимо нормализовать базу, добавить таблицы связи (если отношение владелец-компьютер не 1 к 1, причем non-nullable) и в простейшем случае искать по Primary Key, который представлял бы id каждого компьютера.

Из плюсов можно отметить динамический подсчет эталонных значений характеристик компьютера. Если через какое-то время мы захотим проверить работу программы для более новых характеристик, в этом не будет проблемы. Одним из возможных вариантов замены этого модуля могло бы быть создание алгоритма расчета скользящего среднего на основе данных в БД для данной задачи, но моя квалификация не позволяет реализовать данный метод.

Листинг программного кода:

structure.h

```
#ifndef _STRUCTURE_H_
#define _STRUCTURE_H_
#include <inttypes.h>
#define STR_SIZE 32
typedef struct {
    char name[STR_SIZE];
    uint32_t freq;
} CPU;
typedef struct {
    char name[STR_SIZE];
    uint32_t memory;
} GPU;
typedef struct {
    char type[STR_SIZE];
    uint32_t size;
} HDD;
typedef struct {
    char owner[STR_SIZE];
    char name_OS[STR_SIZE];
    uint32_t num_cpu;
    CPU cpu;
    GPU gpu;
    uint32_t num_hdd;
    HDD *hdds;
    uint32_t ram;
} Computer;
typedef struct {
    double num_cpu;
    double cpu_freq;
    double gpu_memory;
    double hdd_size;
    double ram;
} Reference;
#endif
```

io.h

```
#ifndef _IO_H_
#define _IO_H_
#include <stdio.h>
#include <inttypes.h>
#include "structure.h"
uint32_t computer_read_txt(Computer *computer, FILE *in);
uint32_t computer_read_bin(Computer *computer, FILE *in);
void computer_write_bin(Computer *computer, FILE *out);
void computer_print(Computer *computer);
uint32_t reference_read_txt(Reference *reference, FILE *in);
#endif
```

io.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <inttypes.h>
#include "structure.h"
#include "io.h"
uint32_t computer_read_txt(Computer *computer, FILE *in)
{
    fscanf(in, "%s", computer->owner);
    fscanf(in, "%s", computer->name_OS);
    fscanf(in, "%u", &computer->num_cpu);
    fscanf(in, "%s", computer->cpu.name);
    fscanf(in, "%u", &computer->cpu.freq);
    fscanf(in, "%s", computer->gpu.name);
    fscanf(in, "%u", &computer->gpu.memory);
    fscanf(in, "%u", &computer->num_hdd);
    computer->hdds = (HDD *) malloc(computer->num_hdd * sizeof(HDD));
    for (int i = 0; i < computer->num_hdd; ++i) {
        fscanf(in, "%s", computer->hdds[i].type);
        fscanf(in, "%u", &computer->hdds[i].size);
```

```
    }
    //fscanf(in, "%s", computer->hdd.type);
    //fscanf(in, "%u", &computer->hdd.size);
    fscanf(in, "%u", &computer->ram);
    return !feof(in);
}
uint32_t computer_read_bin(Computer *computer, FILE *in)
{
    fread(computer->owner, sizeof(char), STR_SIZE, in);
    fread(computer->name_OS, sizeof(char), STR_SIZE, in);
    fread(&computer->num_cpu, sizeof(uint32_t), 1, in);
    fread(computer->cpu.name, sizeof(char), STR_SIZE, in);
    fread(&computer->cpu.freq, sizeof(uint32_t), 1, in);
    fread(computer->gpu.name, sizeof(char), STR_SIZE, in);
    fread(&computer->gpu.memory, sizeof(uint32_t), 1, in);
    fread(&computer->num_hdd, sizeof(uint32_t), 1, in);
    computer->hdds = (HDD *) malloc(computer->num_hdd * sizeof(HDD));
    for (uint32_t i = 0; i < computer->num_hdd; ++i) {
        fread(computer->hdds[i].type, sizeof(char), STR_SIZE, in);
        fread(&computer->hdds[i].size, sizeof(uint32_t), 1, in);
    }
    //fread(computer->hdd.type, sizeof(char), STR_SIZE, in);
    //fread(&computer->hdd.size, sizeof(uint32_t), 1, in);
    fread(&computer->ram, sizeof(uint32_t), 1, in);
    return !feof(in);
}
void computer_write_bin(Computer *computer, FILE *out)
{
    fwrite(computer->owner, sizeof(char), STR_SIZE, out);
    fwrite(computer->name_OS,  sizeof(char), STR_SIZE, out);
    fwrite(&computer->num_cpu, sizeof(uint32_t), 1, out);
    fwrite(computer->cpu.name, sizeof(char), STR_SIZE, out);
    fwrite(&computer->cpu.freq, sizeof(uint32_t), 1, out);
    fwrite(computer->gpu.name, sizeof(char), STR_SIZE, out);
    fwrite(&computer->gpu.memory, sizeof(uint32_t), 1, out);
    fwrite(&computer->num_hdd, sizeof(uint32_t), 1, out);
    for (int i = 0; i < computer->num_hdd; ++i) {
        fwrite(computer->hdds[i].type, sizeof(char), STR_SIZE, out);
        fwrite(&computer->hdds[i].size, sizeof(uint32_t), 1, out);
    }
    //fwrite(computer->hdd.type, sizeof(char), STR_SIZE, out);
    //fwrite(&computer->hdd.size, sizeof(uint32_t), 1, out);
    fwrite(&computer->ram, sizeof(uint32_t), 1, out);
}
void computer_print(Computer *computer)
{
    printf("%s\n", computer->owner);
    printf("%s\n", computer->name_OS);
    printf("%u\n", computer->num_cpu);
    printf("%s\n", computer->cpu.name);
    printf("%u\n", computer->cpu.freq);
    printf("%s\n", computer->gpu.name);
    printf("%u\n", computer->gpu.memory);
    printf("%u\n", computer->num_hdd);
    for (int i = 0; i < computer->num_hdd; ++i) {
        printf("%s\n", computer->hdds[i].type);
        printf("%u\n", computer->hdds[i].size);
    }
    //printf("%s\n", computer->hdd.type);
    //printf("%u\n", computer->hdd.size);
    printf("%u\n", computer->ram);
    printf("\n");
}
uint32_t reference_read_txt(Reference *reference, FILE *in)
{
    char spec[STR_SIZE];
    for(int i = 0; i < 5; ++i) {
        fscanf(in, "%s", spec);
        if(!strcmp(spec, "num_cpu:")) {
            fscanf(in, "%lf", &reference->num_cpu);
        } else if(!strcmp(spec, "cpu_freq:")) {
            fscanf(in, "%lf", &reference->cpu_freq);
```

```
        } else if(!strcmp(spec, "gpu_memory:")) {
            fscanf(in, "%lf", &reference->gpu_memory);
        } else if(!strcmp(spec, "hdd_size:")) {
            fscanf(in, "%lf", &reference->hdd_size);
        } else if(!strcmp(spec, "ram:")) {
            fscanf(in, "%lf", &reference->ram);
        }
    }
    return !feof(in);
}
```

owner.h

```
#ifndef _OWNER_H_
#define _OWNER_H_
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "structure.h"
typedef struct _nodeowner{
    char name[STR_SIZE];
    struct _nodeowner *next;
} NodeOwner;
typedef struct {
    NodeOwner* head;
    //NodeOwner* last;
} ListOwner;
ListOwner *list_create();
void list_add(ListOwner *next_owner, Computer *computer);
//void list_delete(NodeOwner *after_node);
//bool list_is_empty(ListOwner *list);
void list_destroy(ListOwner *list);
void list_print(ListOwner *list);
#endif
```

owner.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "owner.h"
ListOwner *list_create()
{
    ListOwner *list = (ListOwner*) malloc(sizeof(ListOwner));
    list->head = (NodeOwner*) malloc(sizeof(NodeOwner));
    list->head->next = NULL;
    //list->last->next = NULL;
    return list;
}
void list_add(ListOwner *next_owner, Computer *computer)
{
        NodeOwner *tmp = (NodeOwner *) malloc(sizeof(NodeOwner));
        strcpy(tmp->name, computer->owner);
        tmp->next = next_owner->head;
        next_owner->head = tmp;
}
void list_destroy(ListOwner *list)
{
    NodeOwner *n = list->head->next;
    while (list->head != NULL)
    {
        n = list->head->next;
        free(list->head);
        list->head = n;
    }
    list->head = NULL;
}
void list_print(ListOwner *list)
{
    for(NodeOwner* tmp = list->head; tmp->next; tmp = tmp->next)
        printf("%s\n", tmp->name);
}
```

get_reference.py

```
import requests as req
import re
import webbrowser as webb
import time
import random
from bs4 import BeautifulSoup
from user_agent import generate_user_agent
def get_proxies():
    request = req.get('https://free-proxy-list.net/')
    soup = BeautifulSoup(request.text, 'lxml')
    table = soup.find('table', {'class': 'fpltable'})
    tbody = table.find('tbody')
    rows = tbody.find_all('tr')
    proxies = []
    for row in rows:
        tds = row.find_all('td')
        if tds[4].text.lower().strip() == 'elite proxy':
            proxies.append('%s:%s'%(tds[0].text, tds[1].text))
    return proxies
def get_links(url, headers, proxy):
    '''Возвращет словарь ссылок на товар с одной страницы'''
    url_head = 'https://market.yandex.ru'
    links = {}
    first_page = req.get(url, headers=headers, proxies=proxy)
    soup = BeautifulSoup(first_page.text, 'lxml')
    # Если капча
    if soup.find_all('input', 'form__key'):
        return 'Error'
    for line_block in soup.find_all('div', 'b-serp-item'):
        link = line_block.find('a', {'class' : 'b-link'}, href=True)['href']
        title = line_block.find('div', {'class' : 'b-serp-description__title'}).get_text()
        links.update({title : 'https://market.yandex.ru{}/spec'.format(re.findall('\/product\/[\d+]', link)[0])})
    return links
def get_spec(url, headers, proxy):
    '''Возвращает словарь с нужными характеристиками'''
    page = req.get(url, headers=headers, proxies=proxy)
    soup = BeautifulSoup(page.text, 'lxml')
    # Если капча
    if soup.find_all('img', 'form__captcha'):
        return 'Error'
    needed_spec = ['количество ядер процессора', 'частота процессора', 'объем видеопамяти',
                   'объем жесткого диска', 'объем оперативной памяти']
    items_dict = {}
    specs = soup.findAll("dl", {"class": "n-product-spec"})
    for spec in specs:
        spec_title = spec.find('span', 'n-product-spec__name-inner').get_text().lower().strip()
        if spec_title in needed_spec:
            items_dict.update({spec_title :\
                               int(re.findall('\d+', spec.find("span", "n-product-spec__value-inner").get_text())[0])})
    return items_dict
def to_file(specs_dict):
    for key in specs_dict.keys():
        if key == 'количество ядер процессора':
            specs_dict['cpu_num'] = specs_dict.pop(key)
        elif key == 'частота процессора':
            specs_dict['cpu_freq'] = specs_dict.pop(key)
        elif key == 'объем видеопамяти':
            specs_dict['gpu_memory'] = specs_dict.pop(key)
        elif key == 'объем жесткого диска':
            specs_dict['hdd_size'] = specs_dict.pop(key)
        elif key == 'объем оперативной памяти':
            specs_dict['ram'] = specs_dict.pop(key)
    with open('output.txt', 'w') as f:
        for key in specs_dict.keys():
            f.write(key+': '+str(specs_dict[key])+'\n')
    f.close()
if __name__ == '__main__':
    specs_dict = {'количество ядер процессора' : [], 'частота процессора' : [], 'объем видеопамяти' : [],
                  'объем жесткого диска' : [], 'объем оперативной памяти' : []}
    proxies = []
```

```
    print('Getting proxies')
    for proxy in get_proxies():
        proxies.append({'http' : proxy})
    print('Done')
    headers = {'accept-encoding': 'gzip, deflate', 'user-agent': generate_user_agent(), 'connection': 'keep-alive', 'accept': '*/*'}
    n_proxy = 0
    # Перебираем 20 страниц
    for i in range(1,22):
        url = 'https://m.market.yandex.ru/catalog/54544/list?page={}'.format(i)
        links = get_links(url, headers, proxies[n_proxy])
        print(proxies[n_proxy]) #CHECK
        # Если капча
        while links == 'Error':
            time.sleep(5)
            n_proxy = (n_proxy+1)%len(proxies)
            headers = {'accept-encoding': 'gzip, deflate', 'user-agent': generate_user_agent(), 'connection': 'keep-alive', 'accept':
'*/*'}
            links = get_links(url, headers, proxies[n_proxy])
            print('Changing proxy', end = '')
            print(proxies[n_proxy])
        for idx, key in enumerate(links.keys()):
            print('Собираем информацию о модели: {}'.format(key), '({}/200)'.format(i*10+idx-9))
            item_dict = get_spec(links[key], headers, proxies[n_proxy])
            # Если капча
            while item_dict == 'Error':
                time.sleep(5)
                n_proxy = (n_proxy+1)%len(proxies)
                headers = {'accept-encoding': 'gzip, deflate', 'user-agent': generate_user_agent(), 'connection': 'keep-alive',
'accept': '*/*'}
                item_dict = get_spec(url, headers, proxies[n_proxy])
                print('Changing proxy')
                print(proxies[n_proxy])
            for key in item_dict.keys():
                specs_dict[key].append(int(item_dict[key]))
            time.sleep(random.randint(5,11))
            print('Done')
    # Посчитаем средние значения
    for key in specs_dict.keys():
        specs_dict[key] = sum(specs_dict[key])/len(specs_dict[key])
    # Сохраняем в файл
    to_file(specs_dict)
```

reference.txt

```
num_cpu: 2.562
cpu_freq: 2065.625
gpu_memory: 2355.200
hdd_size: 467.750
ram: 6.062
```

generate.c

```
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "structure.h"
#include "io.h"
void main(uint32_t argc, char *argv[])
{
    FILE *in = fopen(argv[1], "r");
    FILE *out = fopen(argv[2], "w");
    if (argc != 3) {
        fprintf(stdin, "Usage:\n\t./generate FILE_FROM FILE_TO\n");
        exit(1);
    }
    if (!in || !out) {
        fprintf(stderr, "I/O Error: can't open file.\n");
        exit(2);
    }
    Computer computer;
    while (computer_read_txt(&computer, in)) {
        computer_write_bin(&computer, out);
```

```
    }
    fclose(in);
    fclose(out);
}
```

print.c

```
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include "structure.h"
#include "io.h"
void main(uint32_t argc, char *argv[])
{
    FILE *in = fopen(argv[1], "r");
    if (argc != 2) {
        fprintf(stdin, "Usage:\n\t./print DB_FILE\n");
        exit(1);
    }
    if (!in) {
        fprintf(stderr, "I/O Error: can't open file.\n");
        exit(2);
    }
    Computer computer;
    while (computer_read_bin(&computer, in)) {
        computer_print(&computer);
    }
    fclose(in);
}
```

execute.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <inttypes.h>
#include "structure.h"
#include "io.h"
#include "owner.h"
#define REFERENCE_FILE "reference.txt"
uint32_t compare(Computer *computer, Reference *reference)
{
    bool hdds_upgrade = false;
    for (uint32_t i = 0; i < computer->num_hdd; ++i) {
        if (computer->hdds[i].size < reference->hdd_size) {
            hdds_upgrade = true;
        }
    }
    return (computer->num_cpu < reference->num_cpu) ||
           (computer->cpu.freq < reference->cpu_freq) ||
           (computer->gpu.memory < reference->gpu_memory) ||
           hdds_upgrade ||
           (computer->ram < reference->ram);
}
uint32_t main(uint32_t argc, char *argv[])
{
    FILE *in = fopen(argv[1], "r");
    FILE *ref_txt = fopen(REFERENCE_FILE, "r");
    if (argc != 3) {
        fprintf(stderr, "Usage:\n\t./execute DB_FILE MINIMUM_COMPUTERS_NEED_UPGRADE\n");
        exit(1);
    }
    if (!in) {
        fprintf(stderr, "I/O Error: can't open file\n");
        exit(2);
    }
    if (!ref_txt) {
        fprintf(stderr, "I/O Error: can't open file\n");
        exit(2);
    }
    uint32_t upgrade = 0, need_upgrade = 0;
```

```
    Computer computer;
    Reference reference;
    reference_read_txt(&reference, ref_txt);
    ListOwner *owner_print = list_create();
    while (computer_read_bin(&computer, in)) {
        upgrade = compare(&computer, &reference);
        if (upgrade) {
            ++need_upgrade;
            list_add(&owner_print, &computer);
        }
    }
    if (need_upgrade < atoi(argv[2])) {
        fprintf(stderr, "Error: too few computers need upgrade\n");
        exit(4);
    } else {
        list_print(&owner_print);
    }
    list_destroy(&owner_print);
    fclose(in);
    fclose(ref_txt);
    return 0;
}
```