**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

Курсовая работа

**по дисциплине «Программирование»**

Тема: «**Разработка электронной картотеки»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1308 |  | Лепов А. В. |
| Преподаватель |  | Морозов С. М. |

Санкт-Петербург

2022

**Цель работы.**

Целью данной работы является составление электронной картотеки для предметной области «процессоры».

**Задание.**

**Постановка задачи и описание решения.**

По техническому заданию курсовой работы необходимо реализовать электронную картотеку, исходные данные которой хранятся на диске, и программу, обеспечивающую взаимодействие с ней.

*Реализуем программный код по следующему алгоритму.*

Ввод исходных значений пользователем.

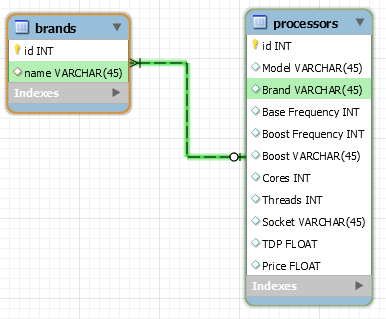
Сперва пользователю необходимо выбрать из предложенных программой действий из меню:

* вывести на экран список структур;
* создать новый элемент;
* обновить элемент списка;
* удалить элемент из списка;
* отсортировать список по параметрам;
* выполнить выборку элементов по параметру;
* провести операции CRUD над дополнительным списком производителей.
* выйти из программы.

После ввода активизируется работа программы:

* если пользователь выбрал вывод на экран, то программа считывает данные с файла и выводит список структур на экран пользователя.
* если пользователь выбрал создание нового элемента, то программа вызывает форму, после заполнения которой эти данные дописываются в файл.
* если пользователь выбрал удаление обновление из списка процессоров, то при вводе выборе определенного элемента, программа последовательно требует на ввод новые данные.
* если пользователь выбрал удаление элемента из списка процессоров, то при вводе какого-либо значения, программа удаляет элемент, удовлетворяющий условию отбора.
* если пользователь выбрал сортировку, то пользователю необходимо ввести параметры отбора, а программа по соответствующим параметрам производит сортировку элементов из списка и выводит результат на экран.
* при опции выборки, пользователю необходимо ввести параметры отбора, а программа по соответствующим параметрам производит отбор элементов из списка и выводит результат на экран.
* при выборе предпоследней опции, пользователю открывается подменю для редактирования дополнительной таблицы.
* если пользователь выбрал последнюю опцию из меню, то программа очищает динамическую память от двусвязного списка структур и завершает работу программы.

*ER-диаграмма табличных данных.*



1. *ER-диаграмма*

*Подключаемые библиотеки, используемые операторы и функции.*

Для выполнения лабораторной работы потребуется подключение стандартной библиотеки «stdio.h», «stdlib.h», «string.h», чтобы у пользователя была возможность вводить значения с клавиатуры и просматривать результаты с экрана, и чтобы программа могла корректно использовать функции для использования динамической памяти и функции для работы со строками.

Необходимо использование одномерных массивов типа «char», «int», структур «struct DLNode», «struct DLHead», «struct LNode», «struct LHead», операторов строгого и нестрогого сравнения, условий «if-else», циклов, а также функций для работы с динамической памятью и строками.

*Алгоритм программы будет состоять из следующих функций:*

* int defOS(); /\* Define operating system \*/
* int separatorsCounting(char C[]); /\* Count separators in line of file \*/
* int fileLinesCounting(FILE\* f, char FN[]); /\* Count lines if file \*/
* void menuOpen(Head\* lst\_struct\_head, B\_Head\* lst\_struct\_brand\_head); /\* Open menu \*/
* void printStructure(Head\* p0); /\* Print list of structures in screen \*/
* void addFirst(Head\* p0, Node\* p); /\* Add first structure to the head \*/
* void separatorCoordinates(char\* P, int sep[]); /\* Remember the position of separators \*/
* void connectLast(Head\* head, Node\* lst\_cur, Node\* lst\_new); /\* Add next structures to the list \*/
* void deleteSelected(Head\* my\_head, Node\* current\_node); /\* Delete selected structure from list \*/
* void addStructureRecord(Head\* lst\_struct\_head, B\_Head\* lst\_struct\_brand\_head); /\* add new structure element to array \*/
* void addStructureRecordBrand(B\_Head\* lst\_struct\_brand\_head); /\* add new structure element to array \*/
* void writeIntoFile(Head\* lst\_struct\_head); /\* write structure list into data file \*/
* void writeIntoFileBrand(B\_Head\* lst\_struct\_brand\_head); /\* write structure list into brand data file \*/
* void printSelected(Head\* head, int option, char\* word); /\* print selected structures \*/
* void sortStructures(Head\* head, char option\_sort); /\* sort stuctures \*/
* void printBrandStructure(B\_Head\* p0); /\* Print list of structures in screen \*/
* void addFirstBrand(B\_Head\* p0, B\_Node\* p); /\* Add first structure to the head \*/
* void deleteSelectedBrand(B\_Head\* my\_head, B\_Node\* current\_node, Head\* head); /\* Delete selected structure from list \*/
* void connectLastPrev(B\_Head\* head, B\_Node\* lst\_cur, B\_Node\* lst\_new); /\* Add next and previous structures to the list \*/
* char\* wordSepAndStr(int x1, int x2, char C[]); /\* Parse file line to character array \*/
* proc\* readFromFileArray(proc \*arr); /\* read from file into structure array \*/
* Head\* createHead(); /\* Create empty head \*/
* Head\* readFromFile(Head\* lst\_struct\_head, B\_Head\* lst\_struct\_brand\_head); /\* Read the initial data from CSV file \*/
* Node\* createNode(char str[], int sep[], int nos, B\_Head\* lst\_struct\_brand\_head); /\* Create structure without next and previous fields \*/
* Node\* selectListElement(Head\* my\_head, int option, char\* word); /\* Select element from list \*/
* B\_Head\* createBrandHead(); /\* Create empty head \*/
* B\_Head\* readBrandFromFile(B\_Head\* lst\_struct\_brand\_head); /\* Read the initial data from CSV file \*/
* B\_Node\* selectBrandListElement(B\_Head\* my\_head, int id); /\* Select element from list \*/
* B\_Node\* createBrandNode(char str[], int sep[], int nos); /\* Create structure without next and previous fields \*/

**Описание переменных.**

Таблица 1. Описание переменных.

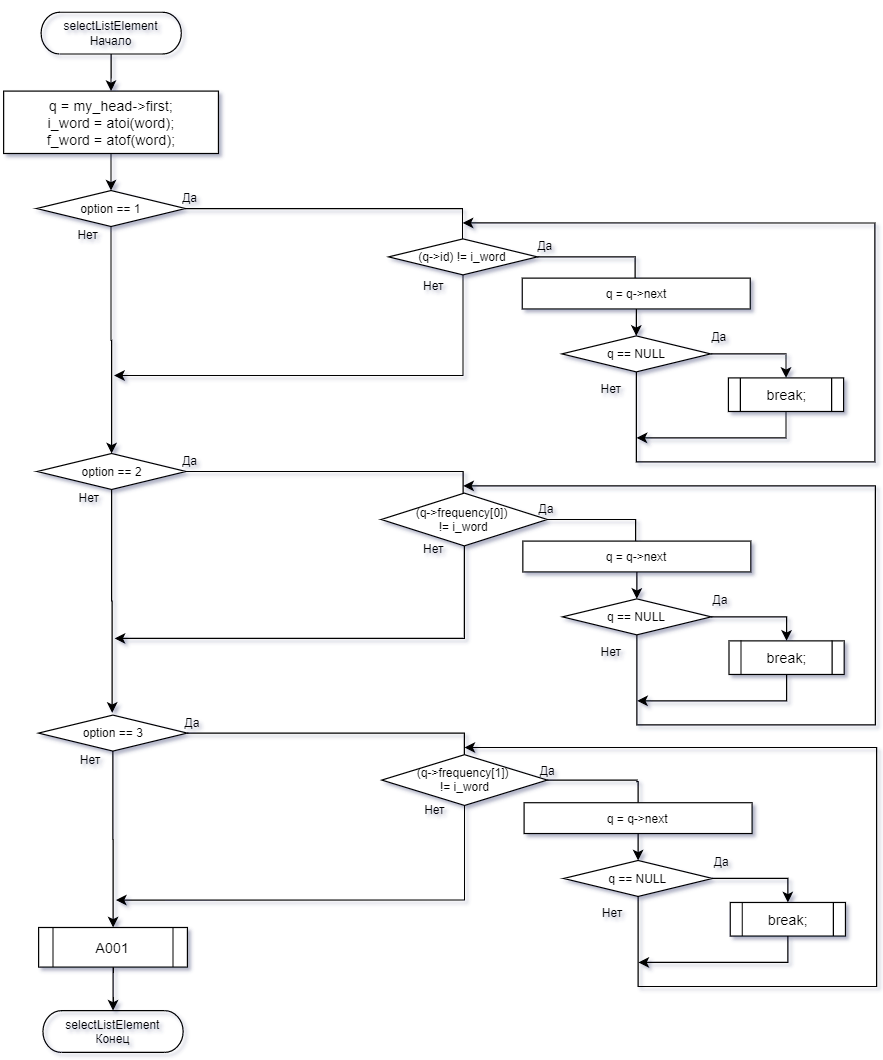
|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| Глобальные переменные и структуры | | | |
| 1 | - | struct LNode (Node) | Структурный тип данных, имеющий поля:   * int id; * char\* model; * struct DLNode \*brand; * int frequency[2]; * char\* boost; * int cores; * int threads; * char\* socket; * float tdp; * float price; * struct LNode \*next; |
| 2 | - | struct LHead  (Head) | Структурный тип данных, имеющий поля:   * int cnt; * struct LNode \*first; * struct LNode \*last; |
| 3 | - | struct DLNode (B\_Node) | Структурный тип данных, имеющий поля:   * int id; * char\* name; * struct DLNode \*next; * struct DLNode \*prev; |
| 4 | - | struct DLHead (B\_Head) | Структурный тип данных, имеющий поля:   * int cnt; * struct DLNode \*first; * struct DLNode \*last; |
| 5 | - | struct processors  (proc) | Структурный тип данных, имеющий поля:   * char model[MAX\_LEN]; * char brand[MAX\_LEN]; * int frequency[2]; * char boost[MAX\_LEN]; * int cores; * int threads; * char socket[MAX\_LEN]; * float tdp; * float price; |
| 6 | MAX\_LEN | - | Максимальное количество символов в строке. |

|  |  |  |  |
| --- | --- | --- | --- |
| int main() | | | |
| 1 | lst\_struct\_brand | B\_Node\* | Переменная-указатель на структуру, в которую считывается информация с файла расширением «csv» при запуске программы. |
| 2 | lst\_struct\_brand\_head | B\_Head\* | Указатель головы на структуру, в которую считывается информация с файла расширением «csv» при запуске программы. |
| 3 | lst\_struct | Node\* | Переменная-указатель на структуру, в которую считывается информация с файла расширением «csv» при запуске программы. |
| 4 | lst\_struct\_head | Head\* | Указатель головы на структуру, в которую считывается информация с файла расширением «csv» при запуске программы. |
| 5 | i | int | Счетчик для очистки списка структур. |
| void menuOpen(Head\* lst\_struct\_head, B\_Head\* lst\_struct\_brand\_head) | | | |
| 1 | option | Int | Переменная для выбора функций из меню. |
| 2 | option\_delete | Int | Переменная для выбора функций из меню. |
| 3 | option\_delete\_confirm | Int | Переменная для выбора функций из меню. |
| 4 | option\_update | Int | Переменная для выбора функций из меню. |
| 5 | option\_update\_confirm | Int | Переменная для выбора функций из меню. |
| 6 | brand\_option | Int | Переменная для выбора функций из меню. |
| 7 | option\_sort | Int | Переменная для выбора функций из меню. |
| 8 | option\_select | Int | Переменная для выбора функций из меню. |
| 9 | res | Int | Переменная для проверки вводимого пользователем значения. |
| 10 | slen | Int | Длина строки, вводимой пользователем для удаления элемента структуры. |
| 11 | l | Int | Переменная для проверки количества удаленных строк. |
| 12 | i | Int | Счетчик функции для использования циклов. |
| 13 | id | Int | Вводимая пользователем переменная. |
| 14 | word | Char\* | Вводимая пользователем строка. |
| 15 | lst\_struct | Node\* | Структура данных. |
| 16 | lst\_struct\_brand | B\_Node\* | Структура данных производителя. |

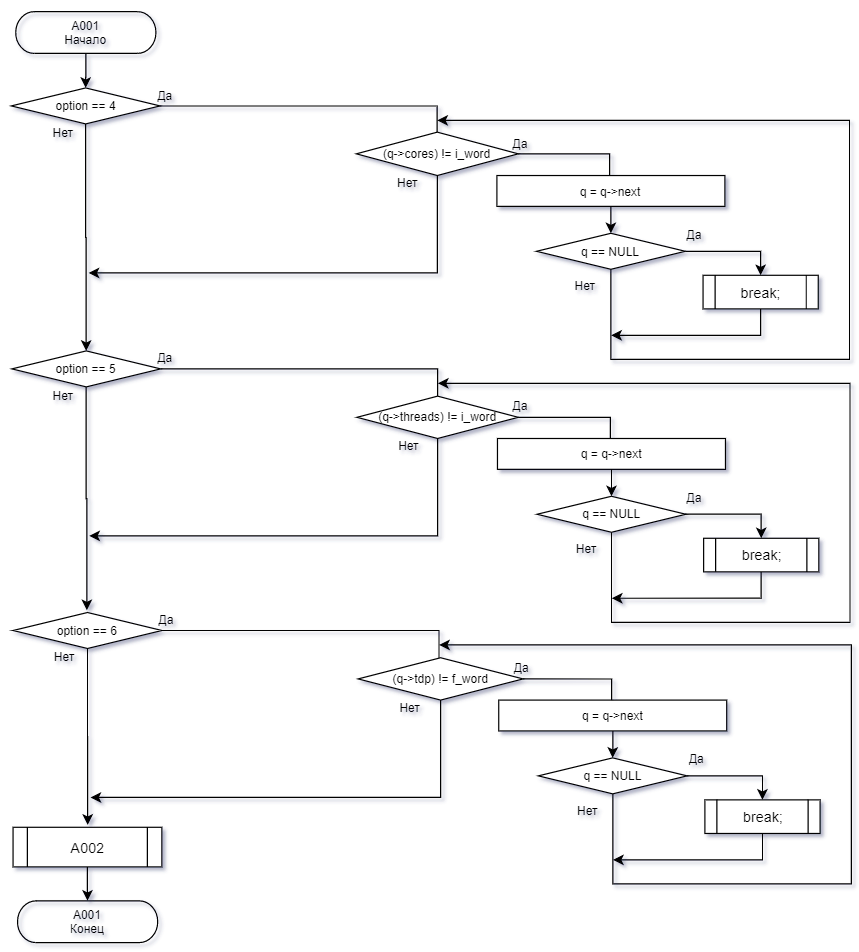
|  |  |  |  |
| --- | --- | --- | --- |
| Head\* readFromFile(Head\* lst\_struct\_head, B\_Head\* lst\_struct\_brand\_head) | | | |
| 1 | file\_data | FILE \* | Указатель на файл с расширением «csv» - таблица исходных данных. |
| 2 | lst\_struct | Node\* | Структура данных. |
| 3 | lst\_struct1 | Node\* | Структура данных. |
| 4 | str | char \* | Переменная, необходимая для чтения файла с исходными данными. |
| 5 | s | int | Переменная, в которую записывается разделитель «;». |
| 6 | nstring | int | Счетчик столбцов в таблице – элементы структуры. |
| 7 | nos | int | Счетчик строк в таблице – элементы массива структур. |
| 8 | sep | Int\* | Массив позиций разделителей. |
| B\_Head\* readBrandFromFile(B\_Head\* lst\_struct\_brand\_head) | | | |
| 1 | file\_data | FILE \* | Указатель на файл с расширением «csv» - таблица исходных данных. |
| 2 | lst\_struct | Node\* | Структура данных. |
| 3 | lst\_struct1 | Node\* | Структура данных. |
| 4 | str | char \* | Переменная, необходимая для чтения файла с исходными данными. |
| 5 | s | int | Переменная, в которую записывается разделитель «;». |
| 6 | nstring | int | Счетчик столбцов в таблице – элементы структуры. |
| 7 | nos | int | Счетчик строк в таблице – элементы массива структур. |
| 8 | sep | Int\* | Массив позиций разделителей. |
| void printStructure(Head\* p0) | | | |
| 1 | p | Node\* | Переменная структурного типа данных для вывода списка структур. |
| void printBrandStructure(B\_Head\* p0) | | | |
| 1 | p | B\_Node\* | Переменная структурного типа данных для вывода списка структур. |
| Node\* selectListElement(Head\* my\_head, int option, char\* word) | | | |
| 1 | q | Node\* | Указатель на структуру. |
| 2 | i\_word | Int | Переменная, вводимая пользователем для нахождения совпадений в числовых полях. |
| 3 | f\_word | Float | Переменная, вводимая пользователем для нахождения совпадений в полях с числами с плавающей точкой. |
| B\_Node\* selectBrandListElement(B\_Head\* my\_head, int id) | | | |
| 1 | q | Node\* | Указатель на структуру. |
| 2 | i | Int | Счетчик функции для использования циклов. |
| void deleteSelected(Head\* my\_head, Node\* current\_node) | | | |
| 1 | q | Node\* | Указатель на структуру, которому присваивается значение первого элемента списка. |
| 2 | q1 | Node\* | Указатель на структуру, которому присваивается значение последнего элемента списка. |
| void deleteSelectedBrand(B\_Head\* my\_head, B\_Node\* current\_node, Head\* lst\_struct\_head) | | | |
| 1 | q | B\_Node\* | Указатель на структуру, которому присваивается значение первого элемента списка. |
| 2 | q1 | B\_Node\* | Указатель на структуру, которому присваивается значение последнего элемента списка. |
| 3 | q2 | B\_Node\* | Указатель на структуру, которому присваивается значение первого элемента списка. |
| 4 | i | Int | Счетчик функции для использования циклов. |
| Node\* createNode(char str[], int sep[], int nos, B\_Head\* lst\_struct\_brand\_head) | | | |
| 1 | p | Node\* | Указатель на структуру. |
| 2 | p\_brand | B\_Node\* | Указатель на структуру производителей. |
| 3 | word | Char\* | Считанная с файла строка. |
| B\_Node\* createBrandNode(char str[], int sep[], int nos) | | | |
| 1 | p | B\_Node | Указатель на структуру. |

|  |  |  |  |
| --- | --- | --- | --- |
| Head\* createHead() | | | |
| 1 | ph | Head\* | Указатель на голову структуры. |
| B\_Head\* createBrandHead() | | | |
| 1 | ph | B\_Head\* | Указатель на голову структуры. |
| char\* wordSepAndStr(int x1, int x2, char C[]) | | | |
| 1 | word | char\* | Указатель на слово. |
| 2 | m | int | Указатель на последний символ в выбранном слове. |
| 3 | i | int | Счетчик функции для использования циклов. |
| int fileLinesCounting(FILE\* f, char FN[]) | | | |
| 1 | n | int | Счетчик количества символов в строке файла. |
| void seraratorCoordinates(char\* P, int sep[]) | | | |
| 1 | m | int | Указатель на последний символ в выбранном слове. |
| 2 | i | int | Счетчик функции для использования циклов. |
| int separatorsCounting(char C[]) | | | |
| 1 | m | int | Указатель на последний символ в выбранном слове. |
| 2 | i | int | Счетчик функции для использования циклов. |
| int defOS() | | | |
| 1 | os | int | Переменная, необходимая для определения типа операционной системы. |
| void addStructureRecordBrand(B\_Head\* lst\_struct\_brand\_head) | | | |
| 1 | fp1 | FILE\* | Указатель на файл. |
| 2 | str | char \*\* | Массив строк для записи данных. |
| 3 | i | Int | Счетчик функции для использования циклов. |
| 4 | len | Int | Длина строки. |
| void addStructureRecord(Head\* lst\_struct\_head, B\_Head\* lst\_struct\_brand\_head) | | | |
| 1 | fp1 | FILE\* | Указатель на файл. |
| 2 | str | char \*\* | Массив строк для записи данных. |
| 3 | i | Int | Счетчик функции для использования циклов. |
| 4 | len | Int | Длина строки. |
| 5 | lst\_struct\_brand | B\_Node \* | Указатель на бренд. |
| void writeIntoFile(Head\* lst\_struct\_head) | | | |
| 1 | fp1 | FILE\* | Указатель на файл. |
| 2 | lst\_struct | Node\* | Указатель на структуру. |
| void writeIntoFileBrand(B\_Head\* lst\_struct\_brand\_head)ё | | | |
| 1 | fp1 | FILE\* | Указатель на файл. |
| 2 | lst\_struct\_brand | B\_Node\* | Указатель на структуру. |
| void printSelected(Head\* head, int option, char\* word) | | | |
| 1 | l | Int | Переменная для проверки количества строк. |
| 2 | i | Int | Счетчик функции для использования циклов. |
| 3 | p | Node\* | Указатель на структуру. |
| 4 | i\_word | Int | Преобразование строки в число. |
| 5 | f\_word | float | Преобразование строки в число. |
| void sortStructures(Head\* head, char option\_sort) | | | |
| 1 | i | int | Счетчик функции для использования циклов. |
| 2 | j | int | Счетчик функции для использования циклов. |
| 3 | arr | Proc \* | Указатель на массив структур. |
| 4 | arr\_temp | Proc | Временная структура, предназначенная для обмена данных двух структур. |
| 5 | fp1 | FILE\* | Указатель на файл. |
| proc\* readFromFileArray(proc \*arr) | | | |
| 1 | fp | FILE\* | Указатель на файл. |
| 2 | Buff | Char[] | Буфер, в который записывается весь файл. |
| 3 | field | Char[] | Содержимое одной ячейки данных. |
| 4 | field\_count | int | Счетчик функции для использования циклов. |
| 5 | row\_count | int | Счетчик функции для использования циклов. |
| 6 | i | int | Счетчик функции для использования циклов. |

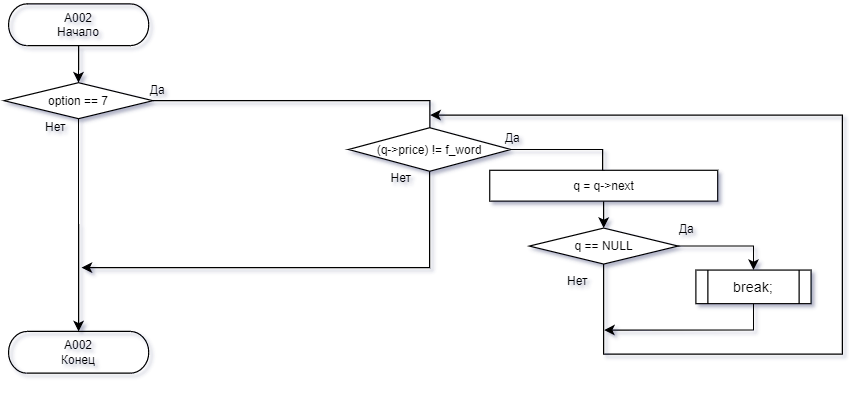
**Схемы алгоритма.**



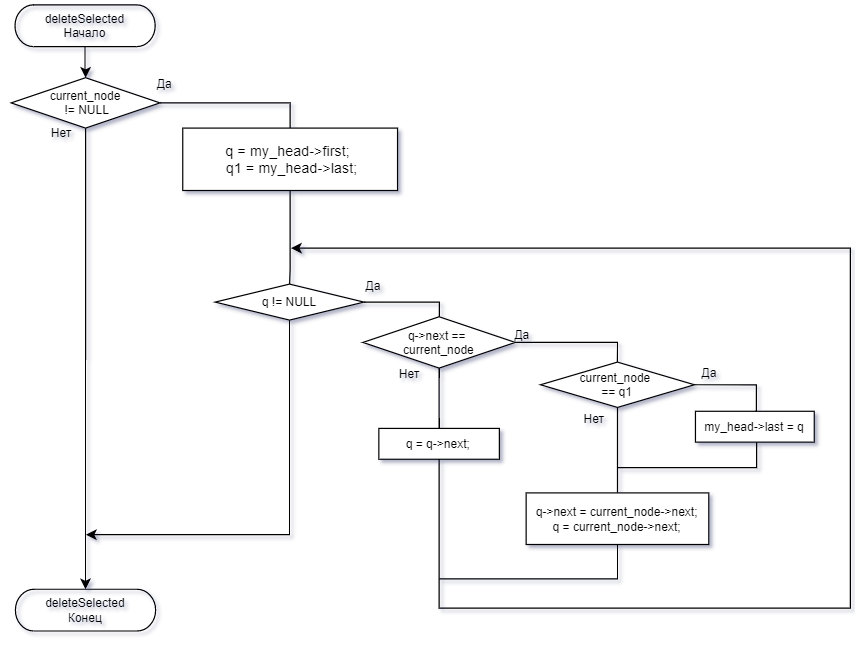
1. *Блок-схема функции SelectListElement*



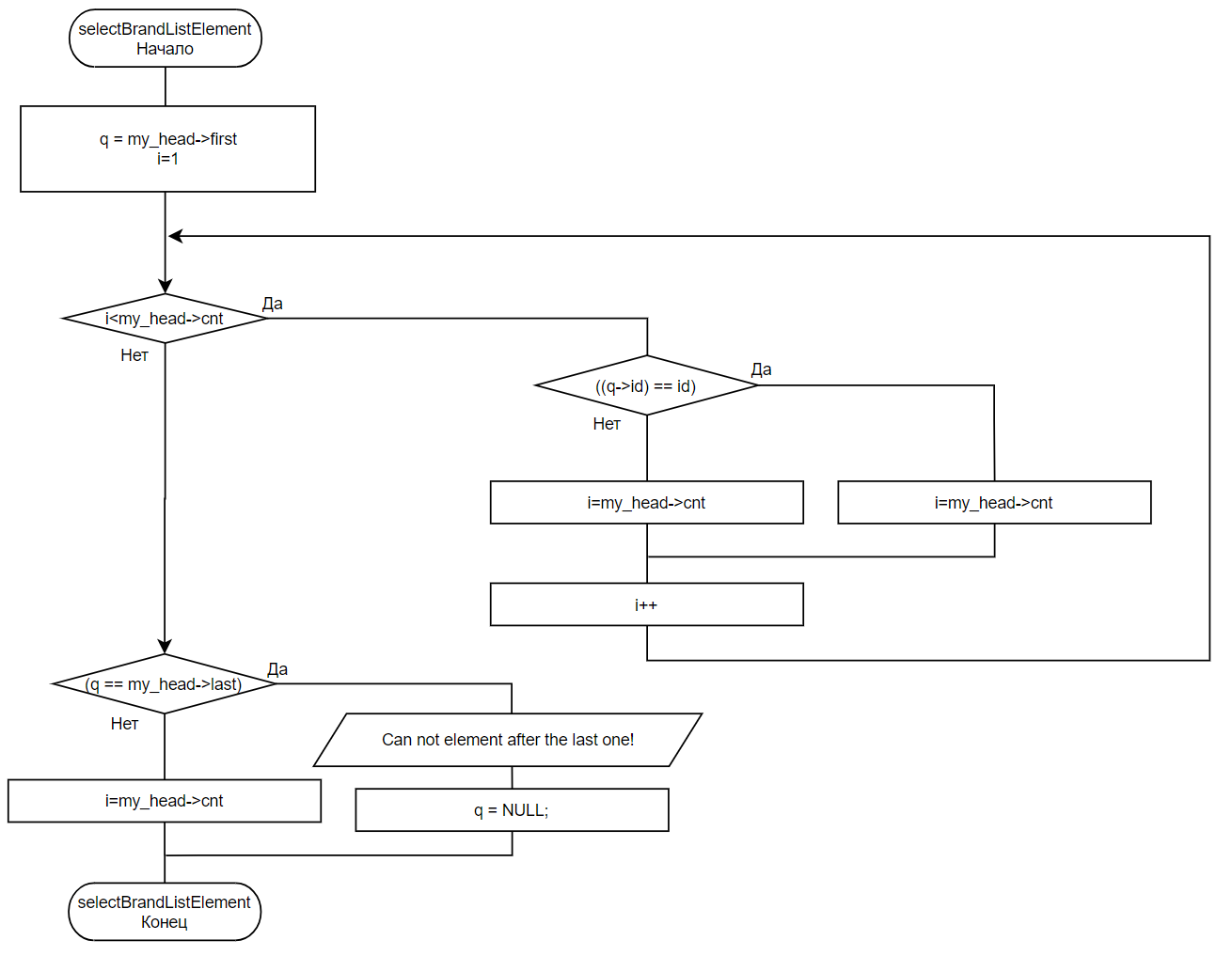
1. *Блок-схема функции A001*



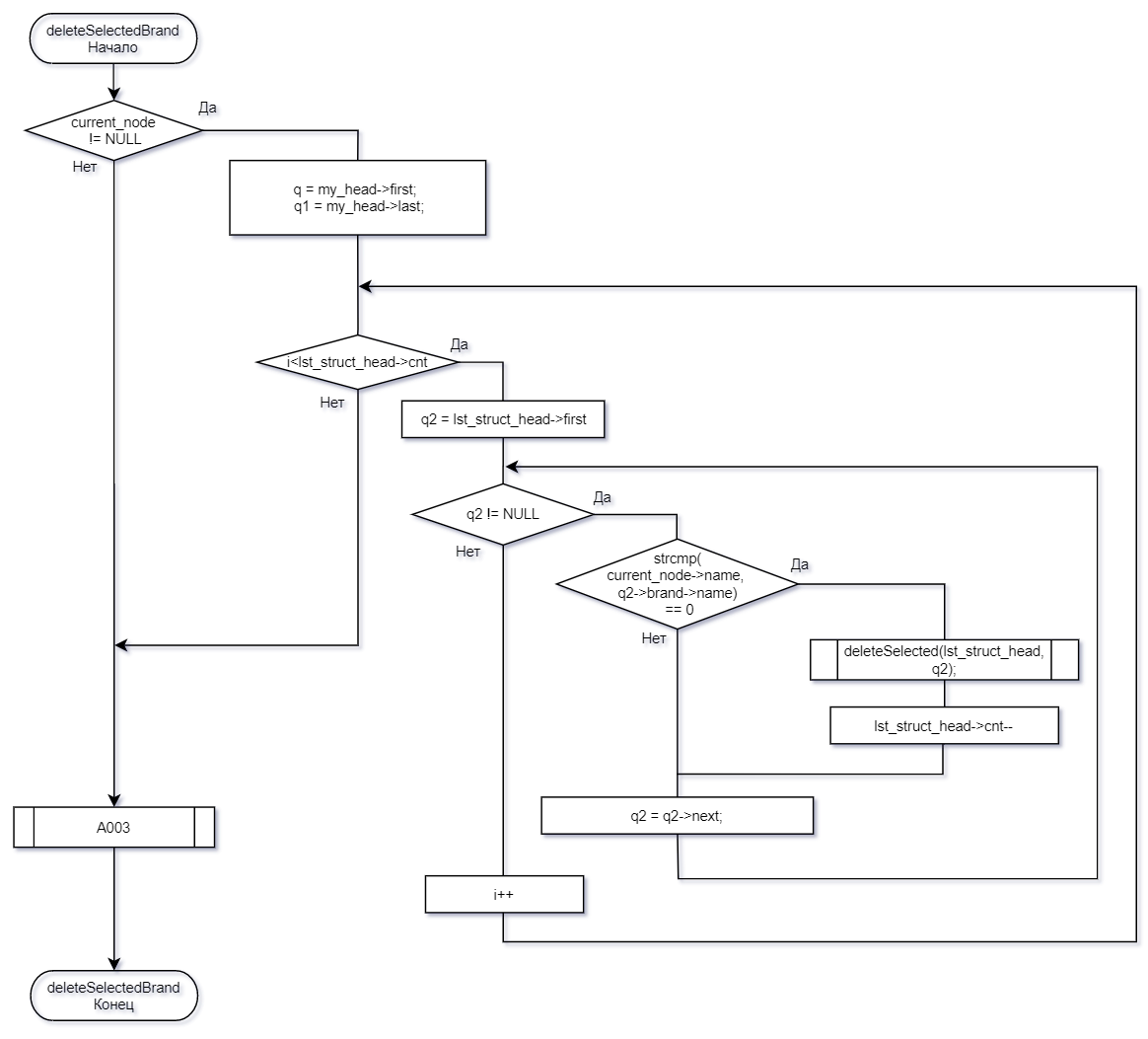
1. *Блок-схема функции A002*



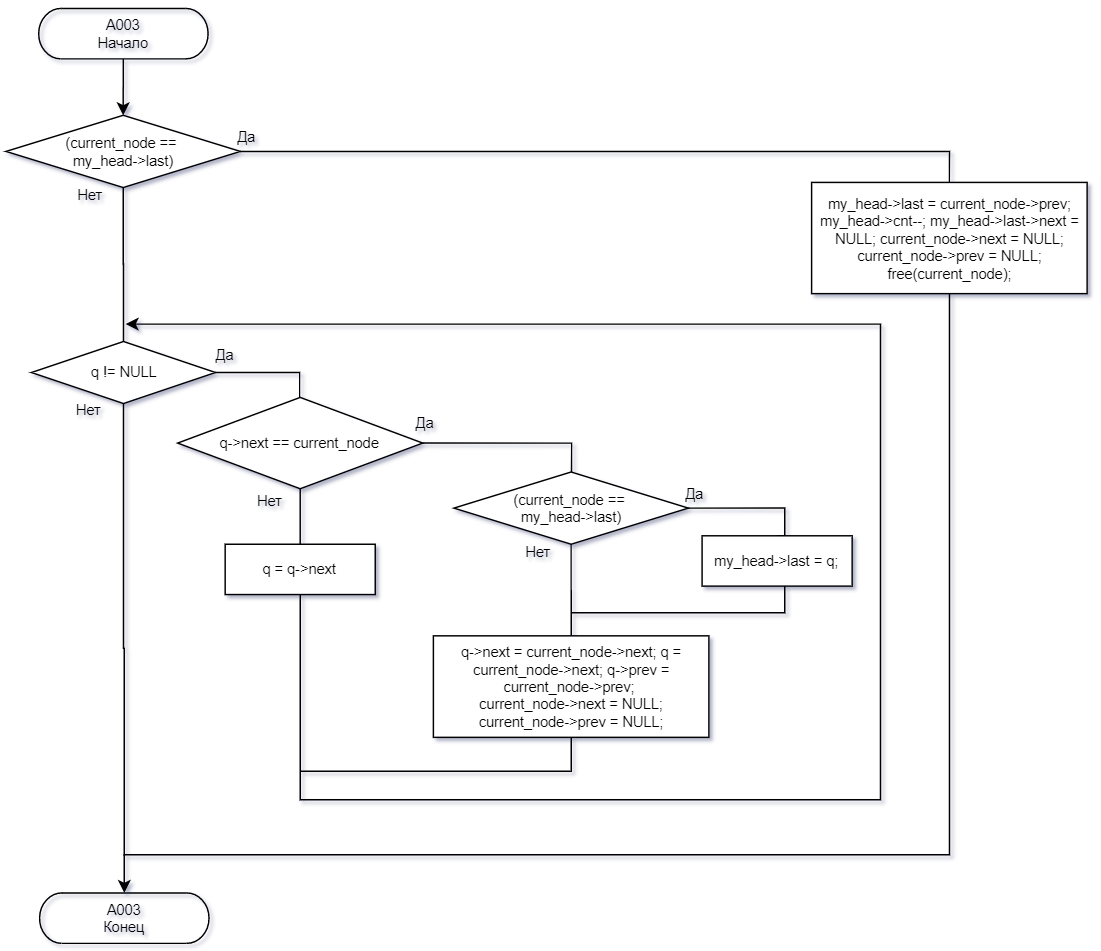
1. *Блок-схема функции deleteSelected*



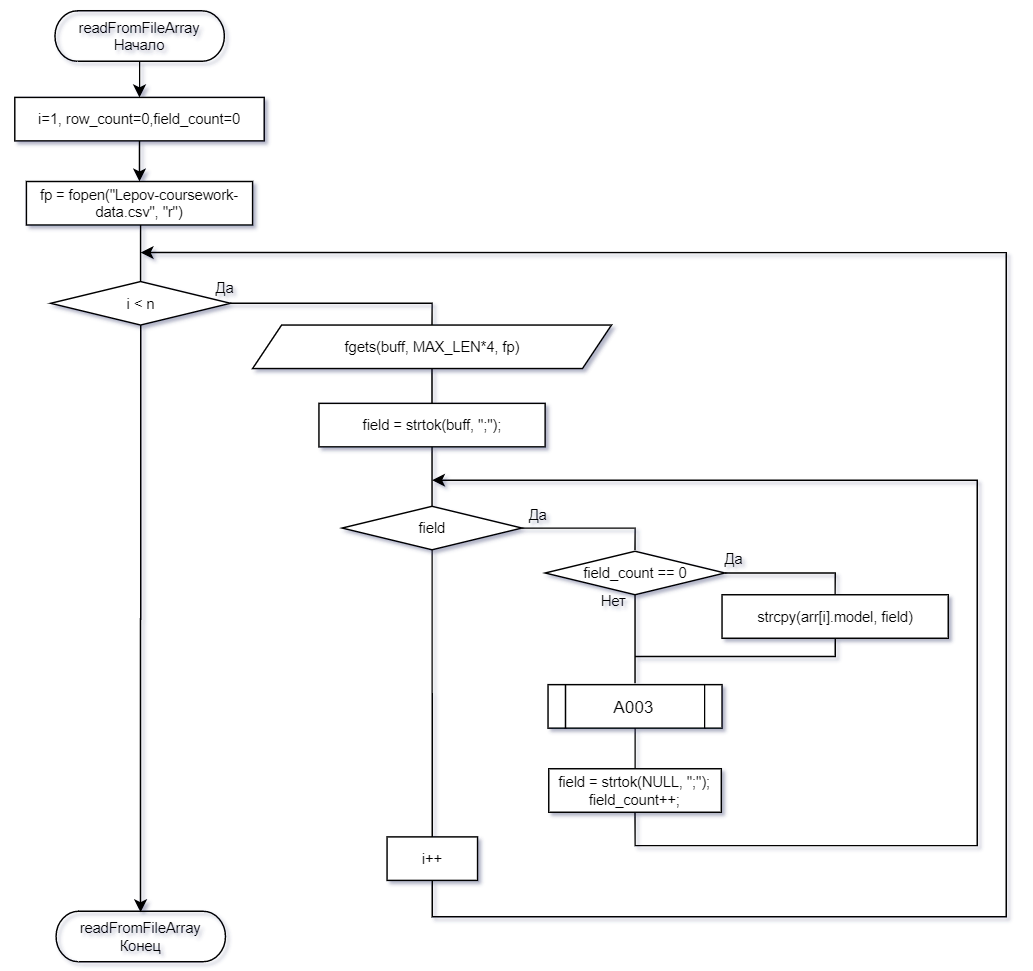
1. *Блок-схема функции* *selectBrandListElement*



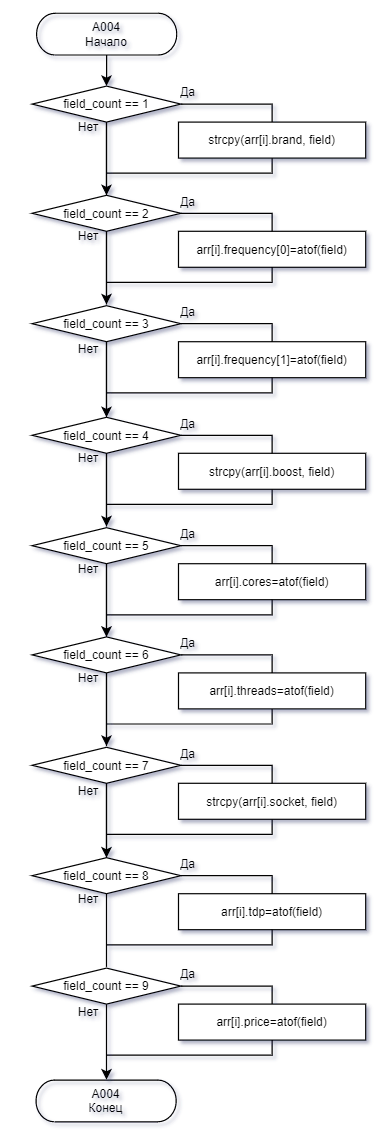
1. *Блок-схема функции* *deleteSelectedBrand*



1. *Блок-схема функции* *A003*



1. *Блок-схема функции readFromFileArray*



1. *Блок-схема функции* *A004*

**Контрольные примеры.**

Таблица 2. Контрольные примеры.

|  |  |  |
| --- | --- | --- |
| № | Входные данные | Результаты работы |
| 1 | option = 1 | Программа, выполняет вывод списка структур на экран. |
| 2 | option = 2  1 - Model: New  2 - Choose brand ID: 2  3 - Base frequency: 1000  4 - Boost frequency: 1200  5 - Boost (Yes/No): Yes  6 - Cores: 2  7 - Threads: 4  8 - Socket: IMB2.0  9 - TDP: 45  10 - Price: 10000 | Осуществление добавления новой записи в таблицу. |
| 3 | option = 3  word[0]=17  1 - Model: Updated proc  2 - Choose brand ID: 4 | Вывод сообщения об ошибки. Отсутствие в дополнительной таблице такого элемента. |
| 4 | option = 3  option\_delete = 17 | Программа выполняет удаление элемента под номером 17. |

**Текст программы.**

/\* ===================================================================================

 \* CRUD console application

 \* ===================================================================================

 \*  @version   1.0

 \*  @author    Lepov Alexey <Alexeylepov@gmail.com>

 \* =================================================================================== \*/

/\* ===================================================================================

 \* Includes

 \* ===================================================================================

 \* This block of code contains:

 \* - included libraries

 \* - defined constants

 \* =================================================================================== \*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_LEN 256

/\* ===================================================================================

 \* Defines

 \* ===================================================================================

 \* This block of code contains:

 \* - global structures

 \* - variable types

 \* - used functions

 \* =================================================================================== \*/

typedef struct DLNode { /\* Two-linked structure list \*/

    int id;

    char\* name;

    struct DLNode \*next;

    struct DLNode \*prev;

} B\_Node;

typedef struct DLHead { /\* Two-linked structure list head \*/

    int cnt;

    struct DLNode \*first;

    struct DLNode \*last;

} B\_Head;

typedef struct LNode { /\* Structure list \*/

    int id;

    char\* model;

    struct DLNode \*brand;

    int frequency[2];

    char\* boost;

    int cores;

    int threads;

    char\* socket;

    float tdp;

    float price;

    struct LNode \*next;

} Node;

typedef struct LHead { /\* Structure list head \*/

    int cnt;

    struct LNode \*first;

    struct LNode \*last;

} Head;

typedef struct processors { /\* Structure array (for sorting) \*/

    char model[MAX\_LEN];

    char brand[MAX\_LEN];

    int frequency[2];

    char boost[MAX\_LEN];

    int cores;

    int threads;

    char socket[MAX\_LEN];

    float tdp;

    float price;

} proc;

int defOS(); /\* Define operating system \*/

int separatorsCounting(char C[]); /\* Count separators in line of file \*/

int fileLinesCounting(FILE\* f, char FN[]); /\* Count lines if file \*/

void menuOpen(Head\* lst\_struct\_head, B\_Head\* lst\_struct\_brand\_head); /\* Open menu \*/

void printStructure(Head\* p0); /\* Print list of structures in screen \*/

void addFirst(Head\* p0, Node\* p); /\* Add first structure to the head \*/

void separatorCoordinates(char\* P, int sep[]); /\* Remember the position of separators \*/

void connectLast(Head\* head, Node\* lst\_cur, Node\* lst\_new); /\* Add next structures  to the list \*/

void deleteSelected(Head\* my\_head, Node\* current\_node); /\* Delete selected structure from list \*/

void addStructureRecord(Head\* lst\_struct\_head, B\_Head\* lst\_struct\_brand\_head); /\* add new structure element to array \*/

void addStructureRecordBrand(B\_Head\* lst\_struct\_brand\_head); /\* add new structure element to array \*/

void writeIntoFile(Head\* lst\_struct\_head); /\* write structure list into data file \*/

void writeIntoFileBrand(B\_Head\* lst\_struct\_brand\_head); /\* write structure list into brand data file \*/

void printSelected(Head\* head, int option, char\* word); /\* print selected structures \*/

void sortStructures(Head\* head, char option\_sort); /\* sort stuctures \*/

void printBrandStructure(B\_Head\* p0); /\* Print list of structures in screen \*/

void addFirstBrand(B\_Head\* p0, B\_Node\* p); /\* Add first structure to the head \*/

void deleteSelectedBrand(B\_Head\* my\_head, B\_Node\* current\_node, Head\* head); /\* Delete selected structure from list \*/

void connectLastPrev(B\_Head\* head, B\_Node\* lst\_cur, B\_Node\* lst\_new); /\* Add next and previous structures  to the list \*/

char\* wordSepAndStr(int x1, int x2, char C[]); /\* Parse file line to character array \*/

proc\* readFromFileArray(proc \*arr); /\* read from file into structure array \*/

Head\* createHead(); /\* Create empty head \*/

Head\* readFromFile(Head\* lst\_struct\_head, B\_Head\* lst\_struct\_brand\_head); /\* Read the initial data from CSV file \*/

Node\* createNode(char str[], int sep[], int nos, B\_Head\* lst\_struct\_brand\_head); /\* Create structure without next and previous fields \*/

Node\* selectListElement(Head\* my\_head, int word); /\* Select element from list \*/

B\_Head\* createBrandHead(); /\* Create empty head \*/

B\_Head\* readBrandFromFile(B\_Head\* lst\_struct\_brand\_head); /\* Read the initial data from CSV file \*/

B\_Node\* selectBrandListElement(B\_Head\* my\_head, int id); /\* Select element from list \*/

B\_Node\* createBrandNode(char str[], int sep[], int nos); /\* Create structure without next and previous fields \*/

/\* ===================================================================================

 \* Main function

 \* ===================================================================================

 \* NAME: main()

 \* TYPE: int

 \*

 \* Returns int value 0 if completed successfully

 \* =================================================================================== \*/

int main()

{

    Node\* lst\_struct = NULL;

    Head\* lst\_struct\_head = NULL;

    int i;

    B\_Node\* lst\_struct\_brand = NULL;

    B\_Head\* lst\_struct\_brand\_head = NULL;

    /\* Creating a heads \*/

    lst\_struct\_brand\_head = readBrandFromFile(lst\_struct\_brand\_head);

    lst\_struct\_head = readFromFile(lst\_struct\_head, lst\_struct\_brand\_head);

    /\* Opening menu \*/

    menuOpen(lst\_struct\_head, lst\_struct\_brand\_head);

    /\* Clearing memory \*/

    lst\_struct = lst\_struct\_head->first;

    for (i=0; i<lst\_struct\_head->cnt; i++)

    {

        if (lst\_struct->model != NULL) { free(lst\_struct->model); lst\_struct->model = NULL; }

        if (lst\_struct->boost != NULL) { free(lst\_struct->boost); lst\_struct->boost = NULL; }

        if (lst\_struct->socket != NULL) { free(lst\_struct->socket); lst\_struct->socket = NULL; }

        lst\_struct = lst\_struct->next;

    }

    if (lst\_struct != NULL) { free(lst\_struct); lst\_struct = NULL; }

    if (lst\_struct\_head != NULL) { free(lst\_struct\_head); lst\_struct\_head = NULL; }

    /\* Clearing memory \*/

    lst\_struct\_brand = lst\_struct\_brand\_head->first;

    for (i = 0; i < lst\_struct\_brand\_head->cnt; i++)

    {

        if (lst\_struct\_brand->name != NULL) { free(lst\_struct\_brand->name); lst\_struct\_brand->name = NULL; }

        lst\_struct\_brand = lst\_struct\_brand->next;

    }

    return 0;

}

/\* ===================================================================================

 \* Open menu

 \* ===================================================================================

 \* NAME: menuOpen(Head\* lst\_struct\_head, B\_Head\* lst\_struct\_brand\_head)

 \* TYPE: void

 \*

 \* Prints on screen information about the program and gets information from user

 \* =================================================================================== \*/

void menuOpen(Head\* lst\_struct\_head, B\_Head\* lst\_struct\_brand\_head)

{

    Node\* lst\_struct;

    B\_Node\* lst\_struct\_brand;

    int option, res, slen, i, id, l=0;

    int option\_delete, option\_delete\_confirm;

    int option\_update\_confirm, brand\_option;

    int option\_sort, option\_select;

    char word[11][MAX\_LEN];

    /\* Filling structure lists \*/

    lst\_struct\_brand\_head = readBrandFromFile(lst\_struct\_brand\_head);

    lst\_struct\_head = readFromFile(lst\_struct\_head, lst\_struct\_brand\_head);

    do {/\* console clearing \*/

        if (defOS()==0) system("cls");

        else system("clear");

        /\* about program \*/

        printf("\033[1;33;40m");

        puts("====================================================");

        puts("             CRUD console application               ");

        puts("====================================================");

        printf("\033[0m");

        puts("\033[1;34;40mEnter the number of the action you would like to do: \033[0m");

        puts("\033[1;32;40m 1 \033[0m- to PRINT list on screen");

        puts("\033[1;32;40m 2 \033[0m- to CREATE new structure");

        puts("\033[1;32;40m 3 \033[0m- to UPDATE single structure");

        puts("\033[1;32;40m 4 \033[0m- to DELETE single structure");

        puts("\033[1;32;40m 5 \033[0m- to SORT structures by field");

        puts("\033[1;32;40m 6 \033[0m- to SELECT structures from list");

        puts("\033[1;32;40m 7 \033[0m- to EDIT additional BRAND list");

        puts("----------------------------------------------------");

        puts("\033[1;32;40m 8 \033[0m- to EXIT the program");

        puts("----------------------------------------------------");

        /\* entering number of action \*/

        printf("\033[1;34;40mType the number of action: \033[0m");

        res = scanf("%d", &option);

        while (getchar() != '\n');

        if (res != 1) option = 0;

    } while(((option<1)||(option>8))||(res!=1));

    /\* =================================================================================== \*/

    /\* PRINT \*/

    /\* =================================================================================== \*/

    if (option==1)

    {   /\* console clearing \*/

        if (defOS()==0) system("cls");

        else system("clear");

        printStructure(lst\_struct\_head);

        puts("\nPress ENTER when ready");

        getchar();

        menuOpen(lst\_struct\_head, lst\_struct\_brand\_head);

    }

    /\* =================================================================================== \*/

    /\* CREATE \*/

    /\* =================================================================================== \*/

    else if (option==2)

    {

        addStructureRecord(lst\_struct\_head, lst\_struct\_brand\_head);

        puts("\nPress ENTER when ready");

        getchar();

        menuOpen(lst\_struct\_head, lst\_struct\_brand\_head);

    }

    /\* =================================================================================== \*/

    /\* UPDATE \*/

    /\* =================================================================================== \*/

    else if (option==3)

    {   /\* console clearing \*/

        if (defOS()==0) system("cls");

        else system("clear");

        /\* print structure list on screen \*/

        printStructure(lst\_struct\_head);

        puts("\033[1;34;40mUpdating element by ID: \033[0m");

        fgets(word[0], MAX\_LEN, stdin);

        slen = strlen(word[0]); word[0][slen - 1] = '\0';

        /\* selecting fields \*/

        lst\_struct = lst\_struct\_head->first;

        lst\_struct = selectListElement(lst\_struct\_head, atoi(word[0]));

        /\* console clearing \*/

        if (defOS()==0) system("cls");

        else system("clear");

        /\* print structure on screen \*/

        printf("\nSelected record:\n");

        printf("==================================================================================================================\n");

        printf("|%2d|%27s|%5s|%9d|%15d|%5s|%5d|%7d|%12s|%6.2f|%9.2f|\n",

        lst\_struct->id,lst\_struct->model,lst\_struct->brand->name,lst\_struct->frequency[0],

        lst\_struct->frequency[1], lst\_struct->boost,lst\_struct->cores, lst\_struct->threads,

        lst\_struct->socket, lst\_struct->tdp, lst\_struct->price);

        printf("==================================================================================================================\n");

        /\* update \*/

        if (lst\_struct != NULL)

        {

            printf("\033[1;32;40m 1 \033[0m- Model:           ");

            fgets(word[2],MAX\_LEN,stdin);

            slen = strlen(word[2]); word[2][slen-1] = '\0';

            lst\_struct->model = word[2];

            printBrandStructure(lst\_struct\_brand\_head);

            printf("\033[1;32;40m 2 \033[0m- Choose brand ID: ");

            fgets(word[1],MAX\_LEN,stdin);

            slen = strlen(word[1]); word[1][slen-1] = '\0';

            brand\_option = atoi(word[1]);

            lst\_struct\_brand = lst\_struct\_brand\_head->first;

            if (brand\_option>0 && brand\_option<=lst\_struct\_brand\_head->cnt)

            {

                for(i=0; i<lst\_struct\_brand\_head->cnt; i++)

                {

                    if (brand\_option==lst\_struct\_brand->id)

                    {

                        lst\_struct->brand = lst\_struct\_brand;

                    }

                    lst\_struct\_brand = lst\_struct\_brand->next;

                }

                printf("\033[1;32;40m 3 \033[0m- Base frequency:  ");

                fgets(word[3],MAX\_LEN,stdin);

                slen = strlen(word[3]); word[3][slen-1] = '\0';

                lst\_struct->frequency[0] = atoi(word[3]);

                printf("\033[1;32;40m 4 \033[0m- Boost frequency: ");

                fgets(word[4],MAX\_LEN,stdin);

                slen = strlen(word[4]); word[4][slen-1] = '\0';

                lst\_struct->frequency[1] = atoi(word[4]);

                printf("\033[1;32;40m 5 \033[0m- Boost (Yes/No):  ");

                fgets(word[5],MAX\_LEN,stdin);

                slen = strlen(word[5]); word[5][slen-1] = '\0';

                lst\_struct->boost = word[5];

                printf("\033[1;32;40m 6 \033[0m- Cores:           ");

                fgets(word[6],MAX\_LEN,stdin);

                slen = strlen(word[6]); word[6][slen-1] = '\0';

                lst\_struct->cores = atoi(word[6]);

                printf("\033[1;32;40m 7 \033[0m- Threads:         ");

                fgets(word[7],MAX\_LEN,stdin);

                slen = strlen(word[7]); word[7][slen-1] = '\0';

                lst\_struct->threads = atoi(word[7]);

                printf("\033[1;32;40m 8 \033[0m- Socket:          ");

                fgets(word[8],MAX\_LEN,stdin);

                slen = strlen(word[8]); word[8][slen-1] = '\0';

                lst\_struct->socket = word[8];

                printf("\033[1;32;40m 9 \033[0m- TDP:             ");

                fgets(word[9],MAX\_LEN,stdin);

                slen = strlen(word[9]); word[9][slen-1] = '\0';

                lst\_struct->tdp = atof(word[9]);

                printf("\033[1;32;40m10 \033[0m- Price:           ");

                fgets(word[10],MAX\_LEN,stdin);

                slen = strlen(word[10]); word[10][slen-1] = '\0';

                lst\_struct->price = atof(word[10]);

                l++;

            }

            else

            {

                printf("\033[1;31;40m\nValue error occurred! \nNeeds to be in range of %d to %d \n\033[0m",1,lst\_struct\_brand\_head->cnt);

            }

        }

        if (l != 0)

        {

            printf("\033[1;32;40m\nFound item was successfully updated!\033[0m\n");

            /\* print structure list on screen \*/

            printStructure(lst\_struct\_head);

            /\* confirming changes \*/

            printf("\033[1;31;40m \nWrite changes to a file? \033[0m \n");

            printf("Type \033[1;32;40m1\033[0m to confirm changes: \033[0m");

            scanf("%d", &option\_update\_confirm);

            if (option\_update\_confirm == 1)

            {

                writeIntoFile(lst\_struct\_head);

                printf("\033[1;32;40mChanges successfully written to a file.\n \033[0m");

            }

            else printf("\033[1;31;40mChanges not saved.\n \033[0m");

        }

        else printf("\033[1;31;40m\nNo matching result found!\033[0m\n");

        puts("\nPress ENTER when ready");

        getchar();

        menuOpen(lst\_struct\_head, lst\_struct\_brand\_head);

    }

    /\* =================================================================================== \*/

    /\* DELETE \*/

    /\* =================================================================================== \*/

    else if (option==4)

    {   /\* console clearing \*/

        if (defOS()==0) system("cls");

        else system("clear");

        /\* print structure list on screen \*/

        printStructure(lst\_struct\_head);

        puts("\033[1;34;40mDeleting element by ID: \033[0m");

        scanf("%d", &option\_delete);

        /\* selecting fields \*/

        lst\_struct = lst\_struct\_head->first;

        lst\_struct = selectListElement(lst\_struct\_head, option\_delete);

        /\* console clearing \*/

        if (defOS()==0) system("cls");

        else system("clear");

        /\* print structure on screen \*/

        printf("\nSelected record:\n");

        printf("==================================================================================================================\n");

        printf("|%2d|%27s|%5s|%9d|%15d|%5s|%5d|%7d|%12s|%6.2f|%9.2f|\n",

        lst\_struct->id, lst\_struct->model, lst\_struct->brand->name,lst\_struct->frequency[0],

        lst\_struct->frequency[1], lst\_struct->boost,lst\_struct->cores, lst\_struct->threads,

        lst\_struct->socket, lst\_struct->tdp, lst\_struct->price);

            printf("==================================================================================================================\n");

        /\* delete \*/

        if (lst\_struct != NULL)

        {

            deleteSelected(lst\_struct\_head, lst\_struct);

            lst\_struct\_head->cnt--;

            l++;

        }

        if (l != 0)

        {

            printf("\033[1;32;40m\nFound item was successfully deleted!\033[0m\n");

            /\* print structure list on screen \*/

            printStructure(lst\_struct\_head);

            /\* confirming changes \*/

            printf("\033[1;31;40m \nWrite changes to a file? \033[0m \n");

            printf("Type \033[1;32;40m1\033[0m to confirm changes: \033[0m");

            scanf("%d", &option\_delete\_confirm);

            if (option\_delete\_confirm == 1)

            {

                writeIntoFile(lst\_struct\_head);

                printf("\033[1;32;40mChanges successfully written to a file.\n \033[0m");

            }

            else printf("\033[1;31;40mChanges not saved.\n \033[0m");

        }

        else printf("\033[1;31;40m\nNo matching result found!\033[0m\n");

        puts("\nPress ENTER when ready");

        getchar(); getchar();

        menuOpen(lst\_struct\_head, lst\_struct\_brand\_head);

    }

    /\* =================================================================================== \*/

    /\* SORT \*/

    /\* =================================================================================== \*/

    else if (option==5)

    {   /\* console clearing \*/

        if (defOS()==0) system("cls");

        else system("clear");

        puts("\033[1;34;40mSorting options: \033[0m");

        puts("\033[1;32;40m 1 \033[0m- Model");

        puts("\033[1;32;40m 2 \033[0m- Brand");

        puts("\033[1;32;40m 3 \033[0m- Base Frequency");

        puts("\033[1;32;40m 4 \033[0m- Boost Frequency");

        puts("\033[1;32;40m 5 \033[0m- Boost (Yes/No)");

        puts("\033[1;32;40m 6 \033[0m- Cores");

        puts("\033[1;32;40m 7 \033[0m- Threads");

        puts("\033[1;32;40m 8 \033[0m- Socket");

        puts("\033[1;32;40m 9 \033[0m- TDP");

        puts("\033[1;32;40m10 \033[0m- Price");

        /\* entering number of action \*/

        printf("\033[1;34;40mType the number of field: \033[0m");

        scanf("%d", &option\_sort);

        if (option\_sort>=1 && option\_sort<=10)

        {

            sortStructures(lst\_struct\_head, option\_sort);

            /\* print structure list on screen \*/

            lst\_struct\_head = readFromFile(lst\_struct\_head, lst\_struct\_brand\_head);

            printStructure(lst\_struct\_head);

            printf("\033[1;32;40m\nList was successfully sorted!\033[0m\n");

        }

        else

        {

            printf("\033[1;31;40m\nError occurred! Your number should be in range of 1 to 11\033[0m\n");

        }

        puts("\nPress ENTER when ready");

        getchar(); getchar();

        menuOpen(lst\_struct\_head, lst\_struct\_brand\_head);

    }

    /\* =================================================================================== \*/

    /\* SELECT \*/

    /\* =================================================================================== \*/

    else if (option==6)

    {   /\* console clearing \*/

        if (defOS()==0) system("cls");

        else system("clear");

        /\* print structure list on screen \*/

        printStructure(lst\_struct\_head);

        puts("\033[1;34;Sorting options: \033[0m");

        puts("\033[1;32;40m 1 \033[0m- ID              - Int");

        puts("\033[1;32;40m 2 \033[0m- Model           - Char[]");

        puts("\033[1;32;40m 3 \033[0m- Brand           - Char[]");

        puts("\033[1;32;40m 4 \033[0m- Base Frequency  - Int");

        puts("\033[1;32;40m 5 \033[0m- Boost Frequency - Int");

        puts("\033[1;32;40m 6 \033[0m- Boost (Yes/No)  - Char[]");

        puts("\033[1;32;40m 7 \033[0m- Cores           - Int");

        puts("\033[1;32;40m 8 \033[0m- Threads         - Int");

        puts("\033[1;32;40m 9 \033[0m- Socket          - Char[]");

        puts("\033[1;32;40m10 \033[0m- TDP             - Float");

        puts("\033[1;32;40m11 \033[0m- Price           - Float");

        /\* entering number of action \*/

        printf("\033[1;34;40mType the number of field: \033[0m");

        scanf("%d", &option\_select);

        if (option\_select>=1 && option\_select<=11)

        {   /\* entering field data \*/

            printf("\033[1;34;40mNow enter some suitable content: \033[0m");

            getchar();

            fgets(word[0], MAX\_LEN, stdin);

            slen = strlen(word[0]); word[0][slen - 1] = '\0';

            /\* selecting structures \*/

            printSelected(lst\_struct\_head, option\_select, word[0]);

        }

        else

        {

            printf("\033[1;31;40m\nError occurred! Your number should be in range of 1 to 11\033[0m\n");

        }

        puts("\nPress ENTER when ready");

        getchar();

        menuOpen(lst\_struct\_head, lst\_struct\_brand\_head);

    }

    /\* =================================================================================== \*/

    /\* EDIT BRAND LIST \*/

    /\* =================================================================================== \*/

    else if (option==7)

    {

        do {/\* console clearing \*/

            if (defOS()==0) system("cls");

            else system("clear");

            /\* about program \*/

            printf("\033[1;33;40m");

            puts("====================================================");

            puts("                  EDIT Brand list                   ");

            puts("====================================================");

            printf("\033[0m");

            puts("\033[1;34;40mEnter the number of the action you would like to do: \033[0m");

            puts("\033[1;32;40m 1 \033[0m- to PRINT list on screen");

            puts("\033[1;32;40m 2 \033[0m- to CREATE new structure");

            puts("\033[1;32;40m 3 \033[0m- to UPDATE single structure");

            puts("\033[1;32;40m 4 \033[0m- to DELETE single structure");

            puts("----------------------------------------------------");

            puts("\033[1;32;40m 5 \033[0m- to GO BACK");

            puts("----------------------------------------------------");

            /\* entering number of action \*/

            printf("\033[1;34;40mType the number of action: \033[0m");

            res = scanf("%d", &option);

            while (getchar() != '\n');

            if (res != 1) option = 0;

        } while(((option<1)||(option>5))||(res!=1));

        /\* =================================================================================== \*/

        /\* PRINT BRAND LIST \*/

        /\* =================================================================================== \*/

        if (option==1)

        {   /\* console clearing \*/

            if (defOS()==0) system("cls");

            else system("clear");

            printBrandStructure(lst\_struct\_brand\_head);

            puts("\nPress ENTER when ready");

            getchar();

            menuOpen(lst\_struct\_head, lst\_struct\_brand\_head);

        }

        /\* =================================================================================== \*/

        /\* CREATE BRAND LIST \*/

        /\* =================================================================================== \*/

        else if (option==2)

        {

            addStructureRecordBrand(lst\_struct\_brand\_head);

            puts("\nPress ENTER when ready");

            getchar();

            menuOpen(lst\_struct\_head, lst\_struct\_brand\_head);

        }

        /\* =================================================================================== \*/

        /\* UPDATE BRAND LIST \*/

        /\* =================================================================================== \*/

        else if (option==3)

        {   /\* console clearing \*/

            if (defOS()==0) system("cls");

            else system("clear");

            l=0;

            /\* print structure list on screen \*/

            puts("\033[1;34;40mUpdating element with chosen ID: \033[0m");

            printBrandStructure(lst\_struct\_brand\_head);

            /\* entering field data \*/

            printf("\033[1;34;40m\nType the number of brand ID: \033[0m");

            scanf("%d", &id);

            getchar();

            /\* selecting fields \*/

            lst\_struct\_brand = lst\_struct\_brand\_head->first;

            lst\_struct\_brand = selectBrandListElement(lst\_struct\_brand\_head, id);

            /\* update \*/

            if (lst\_struct\_brand != NULL)

            {

                printf("\033[1;32;40m 1 \033[0m- Brand name: ");

                fgets(word[0],MAX\_LEN,stdin);

                slen = strlen(word[0]); word[0][slen-1] = '\0';

                lst\_struct\_brand->name = word[0];

                l++;

            }

            if (l != 0)

            {

                printf("\033[1;32;40m\nFound item was successfully updated!\033[0m\n");

                /\* print structure list on screen \*/

                printBrandStructure(lst\_struct\_brand\_head);

                /\* print structure list on screen \*/

                printStructure(lst\_struct\_head);

                /\* confirming changes \*/

                printf("\033[1;31;40m \nWrite changes to a file? \033[0m");

                printf("\033[1;31;40m \nBe very careful! \033[0m");

                printf("\033[1;31;40m \nUpdating a particular brand will also update \nall related records in the main table!\033[0m \n");

                printf("\nType \033[1;32;40m1\033[0m to confirm changes: \033[0m ");

                scanf("%d", &option\_delete\_confirm);

                if (option\_delete\_confirm == 1)

                {

                    writeIntoFile(lst\_struct\_head);

                    writeIntoFileBrand(lst\_struct\_brand\_head);

                    printf("\033[1;32;40mChanges successfully written to a file.\n \033[0m");

                }

                else printf("\033[1;31;40mChanges not saved.\n \033[0m");

            }

            else printf("\033[1;31;40m\nNo matching result found!\033[0m\n");

            puts("\nPress ENTER when ready");

            getchar(); getchar();

            menuOpen(lst\_struct\_head, lst\_struct\_brand\_head);

        }

        /\* =================================================================================== \*/

        /\* DELETE BRAND LIST \*/

        /\* =================================================================================== \*/

        else if (option==4)

        {   /\* print structure list on screen \*/

            printBrandStructure(lst\_struct\_brand\_head);

            /\* entering field data \*/

            printf("\033[1;34;40m\nType the number of brand ID: \033[0m");

            scanf("%d", &id);

            getchar();

            /\* selecting fields \*/

            lst\_struct\_brand = lst\_struct\_brand\_head->first;

            lst\_struct\_brand = selectBrandListElement(lst\_struct\_brand\_head, id);

            if (lst\_struct\_brand != NULL)

            {

                deleteSelectedBrand(lst\_struct\_brand\_head, lst\_struct\_brand, lst\_struct\_head);

                lst\_struct\_brand\_head->cnt--;

                l++;

            }

            if (l != 0) printf("\033[1;32;40m\nFound brand were successfully deleted!\033[0m\n");

            else printf("\033[1;31;40m\nNo matching results found!\033[0m\n");

            /\* print structure list on screen \*/

            printBrandStructure(lst\_struct\_brand\_head);

            /\* print structure list on screen \*/

            printStructure(lst\_struct\_head);

            /\* confirming changes \*/

            printf("\033[1;31;40m \nWrite changes to a file? \033[0m");

            printf("\033[1;31;40m \nBe very careful! \033[0m");

            printf("\033[1;31;40m \nDeleting a particular brand will also delete \nall related records in the main table!\033[0m \n");

            printf("\nType \033[1;32;40m1\033[0m to confirm changes: \033[0m ");

            scanf("%d", &option\_delete\_confirm);

            if (option\_delete\_confirm == 1)

            {

                writeIntoFile(lst\_struct\_head);

                writeIntoFileBrand(lst\_struct\_brand\_head);

                printf("\033[1;32;40mChanges successfully written to a file.\n \033[0m");

            }

            else printf("\033[1;31;40mChanges not saved.\n \033[0m");

            puts("\nPress ENTER when ready");

            getchar();

            menuOpen(lst\_struct\_head, lst\_struct\_brand\_head);

        }

        /\* =================================================================================== \*/

        /\* GO BACK \*/

        /\* =================================================================================== \*/

        else if (option==5)

        {

            menuOpen(lst\_struct\_head, lst\_struct\_brand\_head);

        }

    }

    /\* =================================================================================== \*/

    /\* EXIT \*/

    /\* =================================================================================== \*/

    else if (option==8) printf("\033[1;31;40m\nThe program is quiting ... \n\033[0m");

}

/\* ===================================================================================

 \* Reading the initial data from CSV file

 \* ===================================================================================

 \* NAME: readFromFile(Head\* lst\_struct\_head, B\_Head\* lst\_struct\_brand\_head)

 \* TYPE: Head\*

 \*

 \* Returns structure list head

 \* =================================================================================== \*/

Head\* readFromFile(Head\* lst\_struct\_head, B\_Head\* lst\_struct\_brand\_head)

{

    FILE\* file\_data;

    Node\* lst\_struct, \*lst\_struct1;

    char str[MAX\_LEN];

    int s, nstring;

    int nos = 0, \*sep = NULL;

    file\_data = fopen("Lepov-coursework-data.csv", "r");

    if (file\_data)

    {

        nstring = fileLinesCounting(file\_data, "Lepov-coursework-data.csv");

        fclose(file\_data);

        file\_data = fopen("Lepov-coursework-data.csv", "r");

        lst\_struct\_head = createHead();

        if (lst\_struct\_head)

        {

            fgets(str, MAX\_LEN, file\_data);

            nos = separatorsCounting(str);

            sep = (int\*)calloc(nos, sizeof(int));

            if (sep)

            {

                separatorCoordinates(str, sep);

                /\* Creating a first list element \*/

                lst\_struct = createNode(str, sep, nos, lst\_struct\_brand\_head);

                if (lst\_struct)

                {

                    /\* Adding the first element to the head \*/

                    addFirst(lst\_struct\_head, lst\_struct);

                    s = 0;

                    if (sep != NULL) { free(sep); sep = NULL; }

                    lst\_struct = lst\_struct\_head->first;

                    /\* Creating other list structures \*/

                    while (s < nstring - 1)

                    {

                        fgets(str, MAX\_LEN, file\_data);

                        nos = separatorsCounting(str);

                        sep = (int\*)calloc(nos, sizeof(int));

                        if (sep)

                        {

                            separatorCoordinates(str, sep);

                            lst\_struct1 = createNode(str, sep, nos, lst\_struct\_brand\_head);

                            connectLast(lst\_struct\_head, lst\_struct, lst\_struct1);

                            lst\_struct = lst\_struct1;

                            s++;

                        }

                        else printf("Error at memory allocation!");

                        if (sep != NULL) { free(sep); sep = NULL; }

                    }

                }

                else printf("Error at memory allocation!");

            }

            else printf("Error at memory allocation!");

        }

        else printf("Error at memory allocation!");

    }

    else printf("File is not found!");

    fclose(file\_data);

    return lst\_struct\_head;

}

/\* ===================================================================================

 \* Reading the initial data from CSV file

 \* ===================================================================================

 \* NAME: readBrandFromFile(B\_Head\* lst\_struct\_brand\_head)

 \* TYPE: B\_Head\*

 \*

 \* Returns structure list head

 \* =================================================================================== \*/

B\_Head\* readBrandFromFile(B\_Head\* lst\_struct\_brand\_head)

{

    FILE\* file\_data;

    B\_Node\* lst\_struct, \*lst\_struct1;

    char str[MAX\_LEN];

    int s, nstring;

    int nos = 0, \*sep = NULL;

    file\_data = fopen("Lepov-coursework-data-brands.csv", "r");

    if (file\_data)

    {

        nstring = fileLinesCounting(file\_data, "Lepov-coursework-data-brands.csv");

        fclose(file\_data);

        file\_data = fopen("Lepov-coursework-data-brands.csv", "r");

        lst\_struct\_brand\_head = createBrandHead();

        if (lst\_struct\_brand\_head)

        {

            fgets(str, MAX\_LEN, file\_data);

            nos = separatorsCounting(str);

            sep = (int\*)calloc(nos, sizeof(int));

            if (sep)

            {

                separatorCoordinates(str, sep);

                /\* Creating a first list element \*/

                lst\_struct = createBrandNode(str, sep, nos);

                if (lst\_struct)

                {

                    /\* Adding the first element to the head \*/

                    addFirstBrand(lst\_struct\_brand\_head, lst\_struct);

                    s = 0;

                    if (sep != NULL) { free(sep); sep = NULL; }

                    lst\_struct = lst\_struct\_brand\_head->first;

                    /\* Creating other list structures \*/

                    while (s < nstring - 1)

                    {

                        fgets(str, MAX\_LEN, file\_data);

                        nos = separatorsCounting(str);

                        sep = (int\*)calloc(nos, sizeof(int));

                        if (sep)

                        {

                            separatorCoordinates(str, sep);

                            lst\_struct1 = createBrandNode(str, sep, nos);

                            connectLastPrev(lst\_struct\_brand\_head, lst\_struct, lst\_struct1);

                            lst\_struct = lst\_struct1;

                            s++;

                        }

                        else printf("Error at memory allocation!");

                        if (sep != NULL) { free(sep); sep = NULL; }

                    }

                }

                else printf("Error at memory allocation!");

            }

            else printf("Error at memory allocation!");

        }

        else printf("Error at memory allocation!");

    }

    else printf("File is not found!");

    fclose(file\_data);

    return lst\_struct\_brand\_head;

}

/\* ===================================================================================

 \* Selecting element from list

 \* ===================================================================================

 \* NAME: selectListElement(Head\* my\_head, int option, char\* word)

 \* TYPE: Node\*

 \*

 \* Returns structure that satisfies user conditions

 \* =================================================================================== \*/

Node\* selectListElement(Head\* my\_head, int word)

{

    int i;

    Node\* q = my\_head->first;

    for (i=1; i<my\_head->cnt; i++)

    {

        if ((q->id) == word) i=my\_head->cnt;

        else q = q->next;

    }

    return q;

}

/\* ===================================================================================

 \* Selecting element from list

 \* ===================================================================================

 \* NAME: selectBrandListElement(B\_Head\* my\_head, int id)

 \* TYPE: Node\*

 \*

 \* Returns structure that satisfies user conditions

 \* =================================================================================== \*/

B\_Node\* selectBrandListElement(B\_Head\* my\_head, int id)

{

    int i;

    B\_Node\* q = my\_head->first;

    for (i=0; i<my\_head->cnt; i++)

    {

        if ((q->id) == id)

        {

            i=my\_head->cnt;

        }

        else q = q->next;

    }

    return q;

}

/\* ===================================================================================

 \* Deleting selected structure from list

 \* ===================================================================================

 \* NAME: deleteSelected(Head\* my\_head, Node\* current\_node)

 \* TYPE: void

 \*

 \* Removes and clears selected structure

 \* =================================================================================== \*/

void deleteSelected(Head\* my\_head, Node\* current\_node)

{

    Node\* q = NULL, \*q1 = NULL;

    if (current\_node != NULL)

    {

        q = my\_head->first;

        q1 = my\_head->last;

        if (current\_node == q)

        {

            /\* for deleting the first element as well \*/

            my\_head->first = current\_node->next;

            current\_node->next = NULL;

            free(current\_node);

        }

        else

        {

            while (q != NULL)

            {   /\* delete the connections to next elements \*/

                if (q->next == current\_node)

                {

                    if (current\_node == q1) my\_head->last = q;

                    q->next = current\_node->next;

                    q = current\_node->next;

                    /\* clear element \*/

                    current\_node->next = NULL;

                    free(current\_node);

                }

                else q = q->next;

            }

        }

    }

}

/\* ===================================================================================

 \* Deleting selected structure from list

 \* ===================================================================================

 \* NAME: deleteSelectedBrand(B\_Head\* my\_head, B\_Node\* current\_node, Head\* lst\_struct\_head)

 \* TYPE: void

 \*

 \* Removes and clears selected structure

 \* =================================================================================== \*/

void deleteSelectedBrand(B\_Head\* my\_head, B\_Node\* current\_node, Head\* lst\_struct\_head)

{

    int i;

    B\_Node\* q = NULL, \*q1 = NULL;

    Node\* q2 = NULL;

    if (current\_node != NULL)

    {

        q = my\_head->first;

        q1 = my\_head->last;

        for (i=0; i<lst\_struct\_head->cnt; i++)

        {

            q2 = lst\_struct\_head->first;

            while (q2 != NULL)

            {

                if (strcmp(current\_node->name, q2->brand->name) == 0)

                {

                    deleteSelected(lst\_struct\_head, q2);

                    lst\_struct\_head->cnt--;

                }

                q2 = q2->next;

            }

        }

        if (current\_node == my\_head->last)

        {   /\* for deleting the last element as well \*/

            my\_head->last = current\_node->prev;

            my\_head->cnt--;

            my\_head->last->next = NULL;

            current\_node->next = NULL;

            current\_node->prev = NULL;

            free(current\_node);

        }

        else while (q != NULL)

        {   /\* delete the connections to next and previous elements \*/

            if (q->next == current\_node)

            {

                if (current\_node == q1) my\_head->last = q;

                q->next = current\_node->next;

                q = current\_node->next;

                q->prev = current\_node->prev;

                /\* clear element \*/

                current\_node->next = NULL;

                current\_node->prev = NULL;

                free(current\_node);

            }

            else q = q->next;

        }

    }

}

/\* ===================================================================================

 \* Creating structure without next and previous fields

 \* ===================================================================================

 \* NAME: createNode(char str[], int sep[], int nos, B\_Head\* lst\_struct\_brand\_head)

 \* TYPE: Node\*

 \*

 \* Returns new structure

 \* =================================================================================== \*/

Node\* createNode(char str[], int sep[], int nos, B\_Head\* lst\_struct\_brand\_head)

{

    Node\* p = NULL;

    B\_Node\* p\_brand = NULL;

    char \*word;

    p = (Node\*)malloc(sizeof(Node));

    p\_brand = (B\_Node\*)malloc(sizeof(B\_Node));

    word = wordSepAndStr(sep[0] + 1, sep[1], str);

    if (p && p\_brand && word)

    {

        p->id = 1;

        p->model = wordSepAndStr(0, sep[0], str);

        p\_brand = lst\_struct\_brand\_head->first;

        if (strcmp(word, lst\_struct\_brand\_head->first->name) == 0)

        {

            p->brand = lst\_struct\_brand\_head->first;

        }

        else while (p\_brand->next != NULL)

        {

            if (strcmp(word, p\_brand->name) == 0)

            {

                p->brand = p\_brand;

            }

            p\_brand = p\_brand->next;

        }

        if (strcmp(word, lst\_struct\_brand\_head->last->name) == 0)

        {

            p->brand = lst\_struct\_brand\_head->last;

        }

        p->frequency[0] = atof(wordSepAndStr(sep[1] + 1, sep[2], str));

        p->frequency[1] = atof(wordSepAndStr(sep[2] + 1, sep[3], str));

        p->boost = wordSepAndStr(sep[3] + 1, sep[4], str);

        p->cores = atof(wordSepAndStr(sep[4] + 1, sep[5], str));

        p->threads = atof(wordSepAndStr(sep[5] + 1, sep[6], str));

        p->socket = wordSepAndStr(sep[6] + 1, sep[7], str);

        p->tdp = atof(wordSepAndStr(sep[7] + 1, sep[8], str));

        p->price = atof(wordSepAndStr(sep[8] + 1, sep[9], str));

        p->next = NULL;

        if (p->model==NULL && p->boost==NULL && p->socket==NULL) p = NULL;

    }

    else p = NULL;

    return p;

}

/\* ===================================================================================

 \* Creating structure without next and previous fields

 \* ===================================================================================

 \* NAME: createBrandNode(char str[], int sep[], int nos)

 \* TYPE: Node\*

 \*

 \* Returns new structure

 \* =================================================================================== \*/

B\_Node\* createBrandNode(char str[], int sep[], int nos)

{

    B\_Node \*p=NULL;

    p = (B\_Node\*)malloc(sizeof(B\_Node));

    if (p)

    {

        p->id = 1;

        p->name = wordSepAndStr(0, sep[0], str);

        p->next = NULL;

        p->prev = NULL;

    }

    else p = NULL;

    return p;

}

/\* ===================================================================================

 \* Creating empty head

 \* ===================================================================================

 \* NAME: createHead()

 \* TYPE: Head\*

 \*

 \* Returns empty structure list head

 \* =================================================================================== \*/

Head\* createHead()

{

    Head\* ph = NULL;

    ph = (Head\*)malloc(sizeof(Head));

    if (ph)

    {

        ph->first = NULL;

        ph->last = NULL;

        ph->cnt = 0;

    }

    else ph = NULL;

    return ph;

}

/\* ===================================================================================

 \* Creating empty brand head

 \* ===================================================================================

 \* NAME: createHead()

 \* TYPE: B\_Head\*

 \*

 \* Returns empty structure list head

 \* =================================================================================== \*/

B\_Head\* createBrandHead()

{

    B\_Head\* ph = NULL;

    ph = (B\_Head\*)malloc(sizeof(B\_Head));

    if (ph)

    {

        ph->first = NULL;

        ph->last = NULL;

        ph->cnt = 0;

    }

    else ph = NULL;

    return ph;

}

/\* ===================================================================================

 \* Parsing file line to character array

 \* ===================================================================================

 \* NAME: wordSepAndStr(int x1, int x2, char C[])

 \* TYPE: char\*

 \*

 \* Returns array of character that was made of file lines and separators

 \* =================================================================================== \*/

char\* wordSepAndStr(int x1, int x2, char C[])

{

    char\* word = NULL;

    int i, m = 0;

    word = (char\*)malloc((x2 - x1 + 1) \* sizeof(char));

    memset(word, '\0', x2 - x1 + 1);

    if (word)

    {

        for (i = x1; i < x2; i++) { word[m] = C[i]; m++; }

        word[m] = '\0';

    }

    else word = NULL;

    return word;

}

/\* ===================================================================================

 \* Count lines if file

 \* ===================================================================================

 \* NAME: fileLinesCounting(FILE\* f, char FN[])

 \* TYPE: int

 \*

 \* Counts lines in data file

 \* =================================================================================== \*/

int fileLinesCounting(FILE\* f, char FN[])

{

    int n = 0;

    f = fopen(FN, "r");

    while (!feof(f)) if (fgetc(f) == '\n') n++;

    fclose(f);

    return (n);

}

/\* ===================================================================================

 \* Saving the position of separators

 \* ===================================================================================

 \* NAME: separatorCoordinates(char\* P, int sep[])

 \* TYPE: void

 \*

 \* This function saves the position of separators

 \* =================================================================================== \*/

void separatorCoordinates(char\* P, int sep[])

{

    int i, m = 0;

    for (i = 0; i < strlen(P); i++)

        if (P[i] == ';') { sep[m] = i; m++; }

    sep[m] = strlen(P);

}

/\* ===================================================================================

 \* Counting separators in line of file

 \* ===================================================================================

 \* NAME: separatorsCounting(char C[])

 \* TYPE: int

 \*

 \* Returns separators count

 \* =================================================================================== \*/

int separatorsCounting(char C[])

{

    int m = 0, i;

    for (i = 0; i < strlen(C); i++) if (C[i] == ';') m++;

    return m + 1;

}

/\* ===================================================================================

 \* Adding first structure to the head

 \* ===================================================================================

 \* NAME: addFirst(Head\* p0, Node\* p)

 \* TYPE: void

 \*

 \* Adds first structure to the head

 \* =================================================================================== \*/

void addFirst(Head\* p0, Node\* p)

{

    if (p0 && p)

    {

        p0->first = p;

        p0->last = p;

        p0->cnt++;

    }

}

/\* ===================================================================================

 \* Adding first structure to the head

 \* ===================================================================================

 \* NAME: addFirst(Head\* p0, Node\* p)

 \* TYPE: void

 \*

 \* Adds first structure to the head

 \* =================================================================================== \*/

void addFirstBrand(B\_Head\* p0, B\_Node\* p)

{

    if (p0 && p)

    {

        p0->first = p;

        p0->last = p;

        p0->cnt++;

    }

}

/\* ===================================================================================

 \* Adding next and previous structures to the list

 \* ===================================================================================

 \* NAME: connectLast(Head\* head, Node\* lst\_cur, Node\* lst\_new)

 \* TYPE: void

 \*

 \* Adds next and previous structures to the list

 \* =================================================================================== \*/

void connectLast(Head\* head, Node\* lst\_cur, Node\* lst\_new)

{

    if (lst\_cur && lst\_new && head)

    {

        /\* edit current structure \*/

        lst\_cur->next = lst\_new;

        /\* edit head structure \*/

        head->last = lst\_new;

        head->cnt++;

        /\* edit new structure \*/

        lst\_new->id = head->cnt;

    }

}

/\* ===================================================================================

 \* Adding next and previous structures to the list

 \* ===================================================================================

 \* NAME: connectLastPrev(B\_Head\* head, B\_Node\* lst\_cur, B\_Node\* lst\_new)

 \* TYPE: void

 \*

 \* Adds next and previous structures to the list

 \* =================================================================================== \*/

void connectLastPrev(B\_Head\* head, B\_Node\* lst\_cur, B\_Node\* lst\_new)

{

    if (lst\_cur && lst\_new && head)

    {

        /\* edit current structure \*/

        lst\_cur->next = lst\_new;

        /\* edit head structure \*/

        head->last = lst\_new;

        head->cnt++;

        /\* edit new structure \*/

        lst\_new->prev = lst\_cur;

        lst\_new->id = head->cnt;

    }

}

/\* ===================================================================================

 \* Define operating system function

 \* ===================================================================================

 \* NAME: defOS()

 \* TYPE: int

 \*

 \* Returns the type of current operating system (Windows or Linux)

 \* =================================================================================== \*/

int defOS()

{

    int os = 0; /\* Windows OS is selected as default \*/

    FILE \*f\_wind = fopen("/usr/lib/systemd", "r");

    if (f\_wind == NULL) os = 0; /\* if OS is Windows \*/

    else os = 1; /\* if OS is Unix based \*/

    fclose(f\_wind);

    return os;

}

/\* ===================================================================================

 \* Print structures

 \* ===================================================================================

 \* NAME: printArray(proc mass[], int n)

 \* TYPE: void

 \*

 \* Prints on screen list of structures

 \* =================================================================================== \*/

void printStructure(Head\* p0)

{

    Node\* p;

    printf("\033[1;32;40m==================================================================================================================\n");

    printf("|\033[0m%2s|%27s|%5s|%9s|%15s|%5s|%5s|%7s|%12s|%6s|%9s\033[1;32;40m|\n", "ID", "Model", "Brand", "Frequency", "Boost Frequency", "Boost", "Cores", "Threads", "Socket", "TDP", "Price");

    printf("==================================================================================================================\n");

    p = p0->first;

    while (p != NULL)

    {

        printf("|\033[0m%2d|%27s|%5s|%9d|%15d|%5s|%5d|%7d|%12s|%6.2f|%9.2f\033[1;32;40m|\n", p->id, p->model, p->brand->name,

        p->frequency[0], p->frequency[1], p->boost, p->cores, p->threads, p->socket, p->tdp, p->price);

        p = p->next;

    }

    printf("==================================================================================================================\033[0m\n");

}

/\* ===================================================================================

 \* Print structures

 \* ===================================================================================

 \* NAME: printArray(proc mass[], int n)

 \* TYPE: void

 \*

 \* Prints on screen list of structures

 \* =================================================================================== \*/

void printBrandStructure(B\_Head\* p0)

{

    B\_Node\* p;

    printf("\033[1;32;40m===========\n");

    printf("|\033[0m%2s|%6s\033[1;32;40m|\n", "ID", "Name");

    printf("===========\n");

    p = p0->first;

    while (p != NULL)

    {

        printf("|\033[0m%2d|%6s\033[1;32;40m| \n", p->id, p->name);

        p = p->next;

    }

    printf("===========\033[0m\n");

}

/\* ===================================================================================

 \* Add new element to array of structures

 \* ===================================================================================

 \* NAME: addStructureRecord(Head\* lst\_struct\_head, B\_Head\* lst\_struct\_brand\_head)

 \* TYPE: void

 \*

 \* This function adds new element to array of structures

 \* =================================================================================== \*/

void addStructureRecord(Head\* lst\_struct\_head, B\_Head\* lst\_struct\_brand\_head)

{

    FILE \*fp1=NULL;

    char \*\*str=NULL;

    int i,len,brand\_option;

    B\_Node \*lst\_struct\_brand=NULL;

    /\* open file for adding new record \*/

    fp1 = fopen("Lepov-coursework-data.csv", "a+");

    if (fp1!=NULL)

    {

        /\* Memory allocation for the string array \*/

        str=(char\*\*)malloc(10\*sizeof(char\*));

        lst\_struct\_brand=(B\_Node\*)malloc(sizeof(B\_Node));

        if(str!=NULL && lst\_struct\_brand!=NULL)

        {

            /\* Memory allocation for each element \*/

            for(i=0; i<10; i++)

            {

                str[i]=(char\*)malloc(MAX\_LEN\*sizeof(char));

                if(str[i]==NULL)

                {

                    printf("\033[1;31;40m\nMemory allocation trouble! \nProgram is going to terminate ... \n\033[0m");

                    exit(0);

                }

            }

            printf("\033[1;34;40m\nPlease enter new structure element: \033[0m\n");

            printf("\033[1;32;40m 1 \033[0m- Model:           ");

            fgets(str[0],MAX\_LEN,stdin);

            len = strlen(str[0]); str[0][len-1] = '\0';

            printBrandStructure(lst\_struct\_brand\_head);

            printf("\033[1;32;40m 2 \033[0m- Choose brand ID: ");

            fgets(str[1],MAX\_LEN,stdin);

            len = strlen(str[1]); str[1][len-1] = '\0';

            brand\_option = atoi(str[1]);

            lst\_struct\_brand = lst\_struct\_brand\_head->first;

            if (brand\_option>0 && brand\_option<=lst\_struct\_brand\_head->cnt)

            {

                for(i=0; i<lst\_struct\_brand\_head->cnt; i++)

                {

                    if (brand\_option==lst\_struct\_brand->id)

                    {

                        str[1] = lst\_struct\_brand->name;

                        len = strlen(str[1]); str[1][len] = '\0';

                    }

                    lst\_struct\_brand = lst\_struct\_brand->next;

                }

            }

            else

            {   /\* Closing file and clearing memory \*/

                for(i=0; i<10; i++)

                {

                    free(str[i]); str[i]=NULL;

                }

                free(str); str=NULL;

                fclose(fp1);

                printf("\033[1;31;40m\nValue error occurred! \nNeeds to be in range of %d to %d \n\033[0m",1,lst\_struct\_brand\_head->cnt);

                getchar();

                menuOpen(lst\_struct\_head, lst\_struct\_brand\_head);

            }

            printf("\033[1;32;40m 3 \033[0m- Base frequency:  ");

            fgets(str[2],MAX\_LEN,stdin);

            len = strlen(str[2]); str[2][len-1] = '\0';

            printf("\033[1;32;40m 4 \033[0m- Boost frequency: ");

            fgets(str[3],MAX\_LEN,stdin);

            len = strlen(str[3]); str[3][len-1] = '\0';

            printf("\033[1;32;40m 5 \033[0m- Boost (Yes/No):  ");

            fgets(str[4],MAX\_LEN,stdin);

            len = strlen(str[4]); str[4][len-1] = '\0';

            printf("\033[1;32;40m 6 \033[0m- Cores:           ");

            fgets(str[5],MAX\_LEN,stdin);

            len = strlen(str[5]); str[5][len-1] = '\0';

            printf("\033[1;32;40m 7 \033[0m- Threads:         ");

            fgets(str[6],MAX\_LEN,stdin);

            len = strlen(str[6]); str[6][len-1] = '\0';

            printf("\033[1;32;40m 8 \033[0m- Socket:          ");

            fgets(str[7],MAX\_LEN,stdin);

            len = strlen(str[7]); str[7][len-1] = '\0';

            printf("\033[1;32;40m 9 \033[0m- TDP:             ");

            fgets(str[8],MAX\_LEN,stdin);

            len = strlen(str[8]); str[8][len-1] = '\0';

            printf("\033[1;32;40m10 \033[0m- Price:           ");

            fgets(str[9],MAX\_LEN,stdin);

            len = strlen(str[9]); str[9][len-1] = '\0';

            fprintf(fp1,"%s;%s;%s;%s;%s;%s;%s;%s;%s;%s\n",str[0],str[1],str[2],str[3],str[4],str[5],str[6],str[7],str[8],str[9]);

        }

        else

        {   /\* If memory not allocated \*/

            printf("\033[1;31;40m\nMemory allocation trouble! \nProgram is going to terminate ... \n\033[0m");

            exit(0);

        }

    }

    else

    {   /\* If files not found \*/

        printf("\033[1;31;40m\nTrouble reading file! \nProgram is going to terminate ... \n\033[0m");

        fclose(fp1);

        exit(0);

    }

    /\* Closing file and clearing memory \*/

    for(i=0; i<10; i++)

    {

        free(str[i]); str[i]=NULL;

    }

    free(lst\_struct\_brand); lst\_struct\_brand=NULL;

    free(str); str=NULL;

    fclose(fp1);

    printf("\033[1;32;40m \nNew structure element successfully added.\n \033[0m");

}

/\* ===================================================================================

 \* Add new element to array of structures

 \* ===================================================================================

 \* NAME: addStructureRecordBrand(B\_Head\* lst\_struct\_brand\_head)

 \* TYPE: void

 \*

 \* This function adds new element to array of structures

 \* =================================================================================== \*/

void addStructureRecordBrand(B\_Head\* lst\_struct\_brand\_head)

{

    FILE \*fp1=NULL;

    char \*\*str=NULL;

    int i,len;

    /\* open file for adding new record \*/

    fp1 = fopen("Lepov-coursework-data-brands.csv", "a+");

    if (fp1!=NULL)

    {   /\* Memory allocation for the string array \*/

        str=(char\*\*)malloc(2\*sizeof(char\*));

        if(str!=NULL)

        {   /\* Memory allocation for each element \*/

            for(i=0; i<10; i++)

            {

                str[i]=(char\*)malloc(MAX\_LEN\*sizeof(char));

                if(str[i]==NULL)

                {

                    printf("\033[1;31;40m\nMemory allocation trouble! \nProgram is going to terminate ... \n\033[0m");

                    exit(0);

                }

            }

            printf("\033[1;34;40m\nPlease enter new structure element: \033[0m\n");

            printf("\033[1;32;40m 1 \033[0m- Brand name:  ");

            fgets(str[0],MAX\_LEN,stdin);

            len = strlen(str[0]); str[0][len-1] = '\0';

            fprintf(fp1,"%s;\n",str[0]);

        }

        else

        {   /\* If memory not allocated \*/

            printf("\033[1;31;40m\nMemory allocation trouble! \nProgram is going to terminate ... \n\033[0m");

            exit(0);

        }

    }

    else

    {   /\* If files not found \*/

        printf("\033[1;31;40m\nTrouble reading file! \nProgram is going to terminate ... \n\033[0m");

        fclose(fp1);

        exit(0);

    }

    /\* Closing file and clearing memory \*/

    for(i=0; i<10; i++)

    {

        free(str[i]); str[i]=NULL;

    }

    free(str); str=NULL;

    fclose(fp1);

    printf("\033[1;32;40m \nNew structure element successfully added.\n \033[0m");

}

/\* ===================================================================================

 \* Write structure elements to the file

 \* ===================================================================================

 \* NAME: writeIntoFile(Head\* lst\_struct\_head)

 \* TYPE: void

 \*

 \* Prints on file structure elements

 \* =================================================================================== \*/

void writeIntoFile(Head\* lst\_struct\_head)

{

    FILE \*fp1=NULL;

    Node\* lst\_struct=NULL;

    fp1 = fopen("Lepov-coursework-data.csv", "w");

    lst\_struct=(Node\*)malloc(sizeof(Node));

    if (lst\_struct)

    {

        lst\_struct = lst\_struct\_head->first;

    }

    else

    {   /\* If memory not allocated \*/

        printf("\033[1;31;40m\nMemory allocation trouble! \nProgram is going to terminate ... \n\033[0m");

        exit(0);

    }

    while(lst\_struct!=NULL)

    {

        fprintf(fp1,"%s;%s;%d;%d;%s;%d;%d;%s;%.2f;%.2f\n",

        lst\_struct->model,

        lst\_struct->brand->name,

        lst\_struct->frequency[0],

        lst\_struct->frequency[1],

        lst\_struct->boost,

        lst\_struct->cores,

        lst\_struct->threads,

        lst\_struct->socket,

        lst\_struct->tdp,

        lst\_struct->price);

        lst\_struct = lst\_struct->next;

    }

    free(lst\_struct); lst\_struct=NULL;

    fclose(fp1);

}

/\* ===================================================================================

 \* Write structure elements to the file

 \* ===================================================================================

 \* NAME: writeIntoFileBrand(Head\* lst\_struct\_brand\_head)

 \* TYPE: void

 \*

 \* Prints on file structure elements

 \* =================================================================================== \*/

void writeIntoFileBrand(B\_Head\* lst\_struct\_brand\_head)

{

    FILE \*fp1=NULL;

    B\_Node\* lst\_struct\_brand=NULL;

    fp1 = fopen("Lepov-coursework-data-brands.csv", "w");

    lst\_struct\_brand=(B\_Node\*)malloc(sizeof(B\_Node));

    if (lst\_struct\_brand)

    {

        lst\_struct\_brand = lst\_struct\_brand\_head->first;

    }

    else

    {   /\* If memory not allocated \*/

        printf("\033[1;31;40m\nMemory allocation trouble! \nProgram is going to terminate ... \n\033[0m");

        exit(0);

    }

    while(lst\_struct\_brand!=NULL)

    {

        fprintf(fp1,"%s;\n",

        lst\_struct\_brand->name);

        lst\_struct\_brand = lst\_struct\_brand->next;

    }

    free(lst\_struct\_brand); lst\_struct\_brand=NULL;

    fclose(fp1);

}

/\* ===================================================================================

 \* Print selected elements

 \* ===================================================================================

 \* NAME: printSelected(Head\* head, int option, char\* word)

 \* TYPE: void

 \*

 \* Prints on screen structure elements less than average

 \* =================================================================================== \*/

void printSelected(Head\* head, int option, char\* word)

{

    int i=0, l=0;

    Node\* p = head->first;

    int i\_word = atoi(word);

    float f\_word = atof(word);

    printf("\033[1;32;40m==================================================================================================================\n");

    printf("|\033[0m%2s|%27s|%5s|%9s|%15s|%5s|%5s|%7s|%12s|%6s|%9s\033[1;32;40m|\n", "ID", "Model", "Brand", "Frequency", "Boost Frequency", "Boost", "Cores", "Threads", "Socket", "TDP", "Price");

    printf("==================================================================================================================\n");

    for (i=0; i<head->cnt; i++)

    {

        if (option==1)

        {

            if((p->id) == i\_word) /\* ID \*/

            {

                printf("|\033[0m%2d|%27s|%5s|%9d|%15d|%5s|%5d|%7d|%12s|%6.2f|%9.2f\033[1;32;40m|\n", p->id, p->model, p->brand->name,

                p->frequency[0], p->frequency[1], p->boost, p->cores, p->threads, p->socket, p->tdp, p->price);l++;

            }

        }

        else if (option==2)

        {

            if(strcmp(p->model, word) == 0) /\* MODEL \*/

            {

                printf("|\033[0m%2d|%27s|%5s|%9d|%15d|%5s|%5d|%7d|%12s|%6.2f|%9.2f\033[1;32;40m|\n", p->id, p->model, p->brand->name,

                p->frequency[0], p->frequency[1], p->boost, p->cores, p->threads, p->socket, p->tdp, p->price);l++;

            }

        }

        else if (option==3)

        {

            if(strcmp(p->brand->name, word) == 0) /\* BRAND \*/

            {

                printf("|\033[0m%2d|%27s|%5s|%9d|%15d|%5s|%5d|%7d|%12s|%6.2f|%9.2f\033[1;32;40m|\n", p->id, p->model, p->brand->name,

                p->frequency[0], p->frequency[1], p->boost, p->cores, p->threads, p->socket, p->tdp, p->price);l++;

            }

        }

        else if (option==4)

        {

            if((p->frequency[0]) == i\_word) /\* BASE FREAQUENCY \*/

            {

                printf("|\033[0m%2d|%27s|%5s|%9d|%15d|%5s|%5d|%7d|%12s|%6.2f|%9.2f\033[1;32;40m|\n", p->id, p->model, p->brand->name,

                p->frequency[0], p->frequency[1], p->boost, p->cores, p->threads, p->socket, p->tdp, p->price);l++;

            }

        }

        else if (option==5)

        {

            if((p->frequency[1]) == i\_word) /\* BOOST FREAQUENCY \*/

            {

                printf("|\033[0m%2d|%27s|%5s|%9d|%15d|%5s|%5d|%7d|%12s|%6.2f|%9.2f\033[1;32;40m|\n", p->id, p->model, p->brand->name,

                p->frequency[0], p->frequency[1], p->boost, p->cores, p->threads, p->socket, p->tdp, p->price);l++;

            }

        }

        else if (option==6)

        {

            if(strcmp(p->boost, word) == 0) /\* BOOST \*/

            {

                printf("|\033[0m%2d|%27s|%5s|%9d|%15d|%5s|%5d|%7d|%12s|%6.2f|%9.2f\033[1;32;40m|\n", p->id, p->model, p->brand->name,

                p->frequency[0], p->frequency[1], p->boost, p->cores, p->threads, p->socket, p->tdp, p->price);l++;

            }

        }

        else if (option==7)

        {

            if((p->cores) == i\_word) /\* CORES \*/

            {

                printf("|\033[0m%2d|%27s|%5s|%9d|%15d|%5s|%5d|%7d|%12s|%6.2f|%9.2f\033[1;32;40m|\n", p->id, p->model, p->brand->name,

                p->frequency[0], p->frequency[1], p->boost, p->cores, p->threads, p->socket, p->tdp, p->price);l++;

            }

        }

        else if (option==8)

        {

            if((p->threads) == i\_word) /\* THREADS \*/

            {

                printf("|\033[0m%2d|%27s|%5s|%9d|%15d|%5s|%5d|%7d|%12s|%6.2f|%9.2f\033[1;32;40m|\n", p->id, p->model, p->brand->name,

                p->frequency[0], p->frequency[1], p->boost, p->cores, p->threads, p->socket, p->tdp, p->price);l++;

            }

        }

        else if (option==9)

        {

            if(strcmp(p->socket, word) == 0) /\* SOCKET \*/

            {

                printf("|\033[0m%2d|%27s|%5s|%9d|%15d|%5s|%5d|%7d|%12s|%6.2f|%9.2f\033[1;32;40m|\n", p->id, p->model, p->brand->name,

                p->frequency[0], p->frequency[1], p->boost, p->cores, p->threads, p->socket, p->tdp, p->price);l++;

            }

        }

        else if (option==10) /\* TDP \*/

        {

            if((p->tdp) == f\_word)

            {

                printf("|\033[0m%2d|%27s|%5s|%9d|%15d|%5s|%5d|%7d|%12s|%6.2f|%9.2f\033[1;32;40m|\n", p->id, p->model, p->brand->name,

                p->frequency[0], p->frequency[1], p->boost, p->cores, p->threads, p->socket, p->tdp, p->price);l++;

            }

        }

        else if (option==11) /\* PRICE \*/

        {

            if((p->price) == f\_word)

            {

                printf("|\033[0m%2d|%27s|%5s|%9d|%15d|%5s|%5d|%7d|%12s|%6.2f|%9.2f\033[1;32;40m|\n", p->id, p->model, p->brand->name,

                p->frequency[0], p->frequency[1], p->boost, p->cores, p->threads, p->socket, p->tdp, p->price);l++;

            }

        }

        p = p->next;

    }

    if(l==0)

    {

        printf("| No element found                                                                                               |\n");

    }

    printf("==================================================================================================================\033[0m\n");

}

/\* ===================================================================================

 \* Sort structures

 \* ===================================================================================

 \* NAME: sortStructures(Head\* head, char option\_sort)

 \* TYPE: void

 \*

 \* Sorts structures

 \* =================================================================================== \*/

void sortStructures(Head\* head, char option\_sort)

{

    int i, j;

    proc \*arr=NULL, arr\_temp;

    FILE \*fp1=NULL;

    arr = readFromFileArray(arr);

    fp1=fopen("Lepov-coursework-data.csv", "w");

    for (i=1; i<head->cnt; i++)

    {

        for (j=0; j<head->cnt-1; j++)

        {

            if (((option\_sort==1) && strcmp(arr[j].model, arr[j+1].model)>0)

            || ((option\_sort==2)  && strcmp(arr[j].brand, arr[j+1].brand)>0)

            || ((option\_sort==3)  && (arr[j].frequency[0] > arr[j+1].frequency[0]))

            || ((option\_sort==4)  && (arr[j].frequency[1] > arr[j+1].frequency[1]))

            || ((option\_sort==5)  && strcmp(arr[j].boost, arr[j+1].boost)>0)

            || ((option\_sort==6)  && (arr[j].cores > arr[j+1].cores))

            || ((option\_sort==7)  && (arr[j].threads > arr[j+1].threads))

            || ((option\_sort==8)  && strcmp(arr[j].socket, arr[j+1].socket)>0)

            || ((option\_sort==9)  && (arr[j].tdp > arr[j+1].tdp))

            || ((option\_sort==10) && (arr[j].price > arr[j+1].price)))

            {

                arr\_temp=arr[j];

                arr[j]=arr[j+1];

                arr[j+1]=arr\_temp;

            }

        }

    }

    for (i=0; i<head->cnt; i++)

    {

        fprintf(fp1,"%s;%s;%d;%d;%s;%d;%d;%s;%.2f;%.2f\n",

        arr[i].model,

        arr[i].brand,

        arr[i].frequency[0],

        arr[i].frequency[1],

        arr[i].boost,

        arr[i].cores,

        arr[i].threads,

        arr[i].socket,

        arr[i].tdp,

        arr[i].price);

    }

    fclose(fp1);

}

/\* ===================================================================================

 \* Read from file

 \* ===================================================================================

 \* NAME: readFromFile(proc \*arr)

 \* TYPE: proc

 \*

 \* This function reads structures from data file

 \* =================================================================================== \*/

proc\* readFromFileArray(proc \*arr)

{

    FILE \*fp;

    char buff[MAX\_LEN\*4];

    char \*field;

    int field\_count=0;

    int row\_count=0;

    int i=1;

    /\* Opening file \*/

    fp = fopen("Lepov-coursework-data.csv", "r");

    if (fp == NULL)

    {

        printf("\033[1;31;40m\nTrouble reading file! \nProgram is going to terminate ... \n\033[0m");

        fclose(fp);

        exit(0);

    }

    /\* counting lines for memory allocation \*/

    while(!feof(fp)) if(fgetc(fp) == '\n') i++;

    /\* returning to the beginning of file \*/

    fseek(fp,0,SEEK\_SET);

    /\* Memory allocation for the array (structure CAR) \*/

    arr = (proc\*)malloc(i\*sizeof(proc));

    if(arr==NULL)

    {

        printf("\033[1;31;40m\nMemory allocation trouble! \nProgram is going to terminate ... \n\033[0m");

        fclose(fp);

        exit(0);

    }

    /\* Filling the array \*/

    i = 0;

    while(fgets(buff, MAX\_LEN\*4, fp))

    {

        /\* for debugging \*/ /\* printf("%s\n", buff); \*/

        field\_count = 0;

        row\_count++;

        /\* setting a separator \*/

        field = strtok(buff, ";");

        while(field)

        {

            if(field\_count == 0) strcpy(arr[i].model, field);

            if(field\_count == 1) strcpy(arr[i].brand, field);

            if(field\_count == 2) arr[i].frequency[0]=atof(field);

            if(field\_count == 3) arr[i].frequency[1]=atof(field);

            if(field\_count == 4) strcpy(arr[i].boost, field);

            if(field\_count == 5) arr[i].cores=atof(field);

            if(field\_count == 6) arr[i].threads=atof(field);

            if(field\_count == 7) strcpy(arr[i].socket, field);

            if(field\_count == 8) arr[i].tdp=atof(field);

            if(field\_count == 9) arr[i].price=atof(field);

            field = strtok(NULL, ";");

            field\_count++;

        }

        i++;

    }

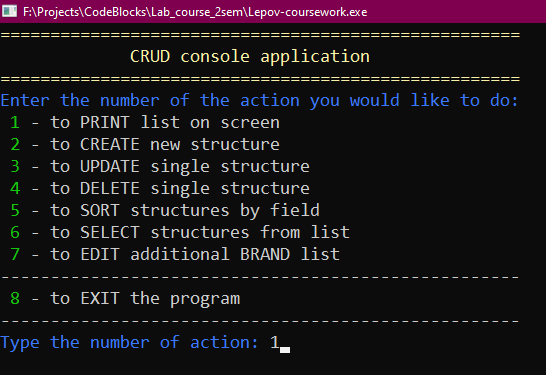
    /\* Closing file \*/

    fclose(fp);

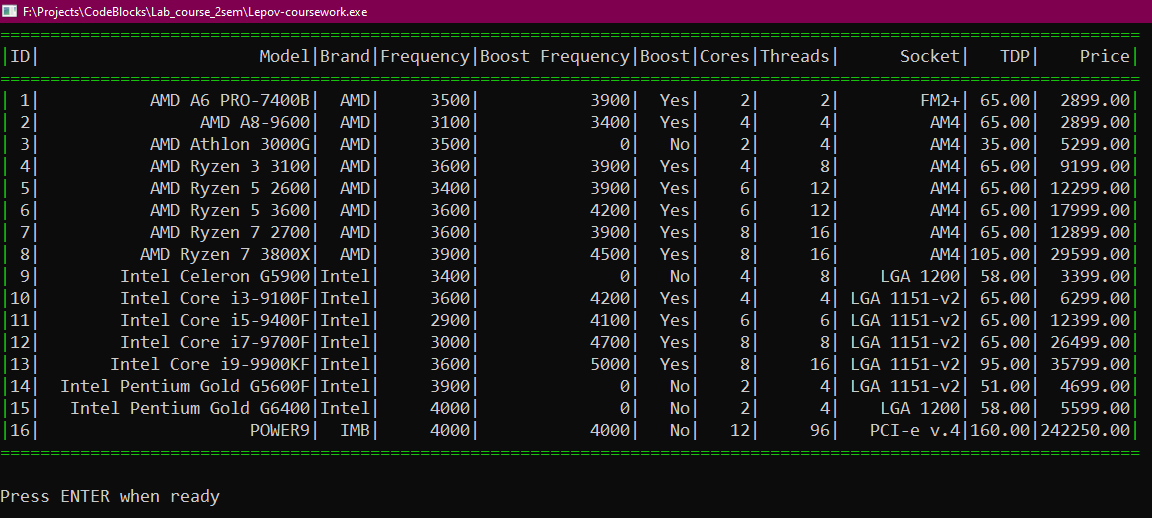
    return arr;

}

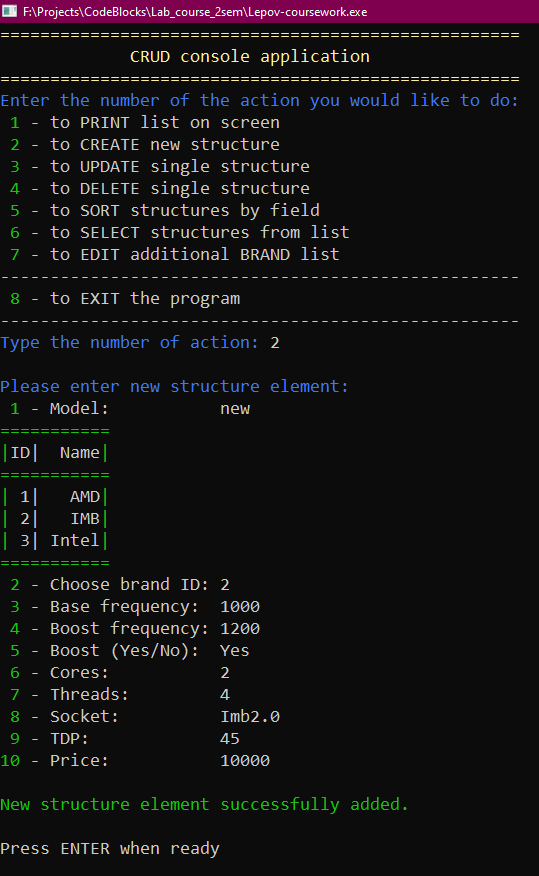
**Примеры выполнения программы.**



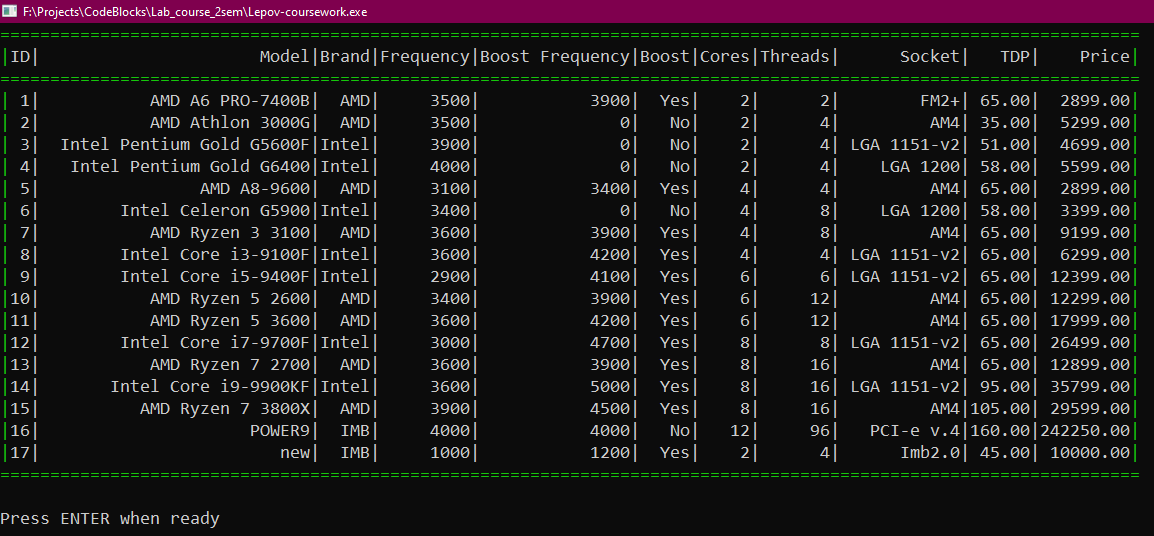
1. *Результат компиляции программного кода*



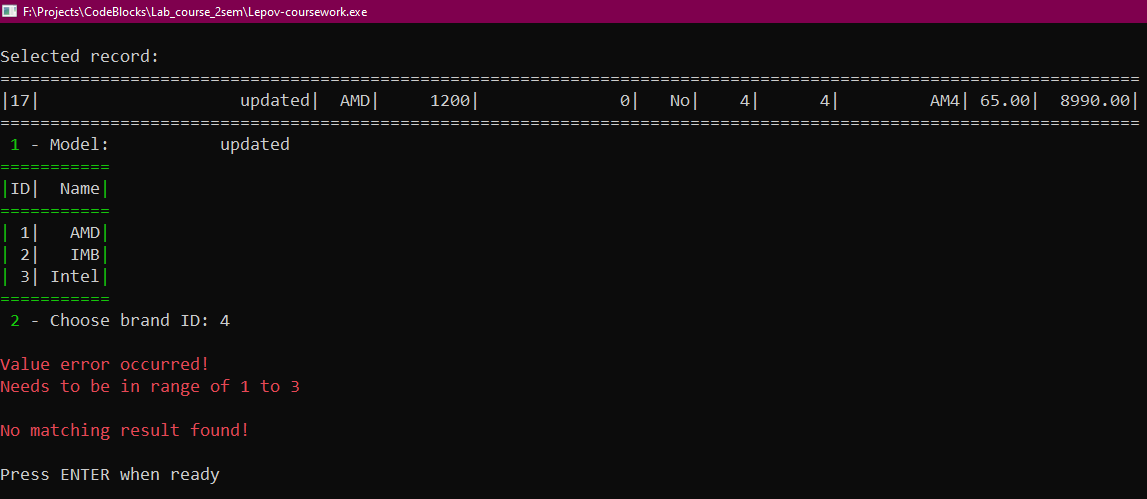
1. *Результат компиляции программного кода*



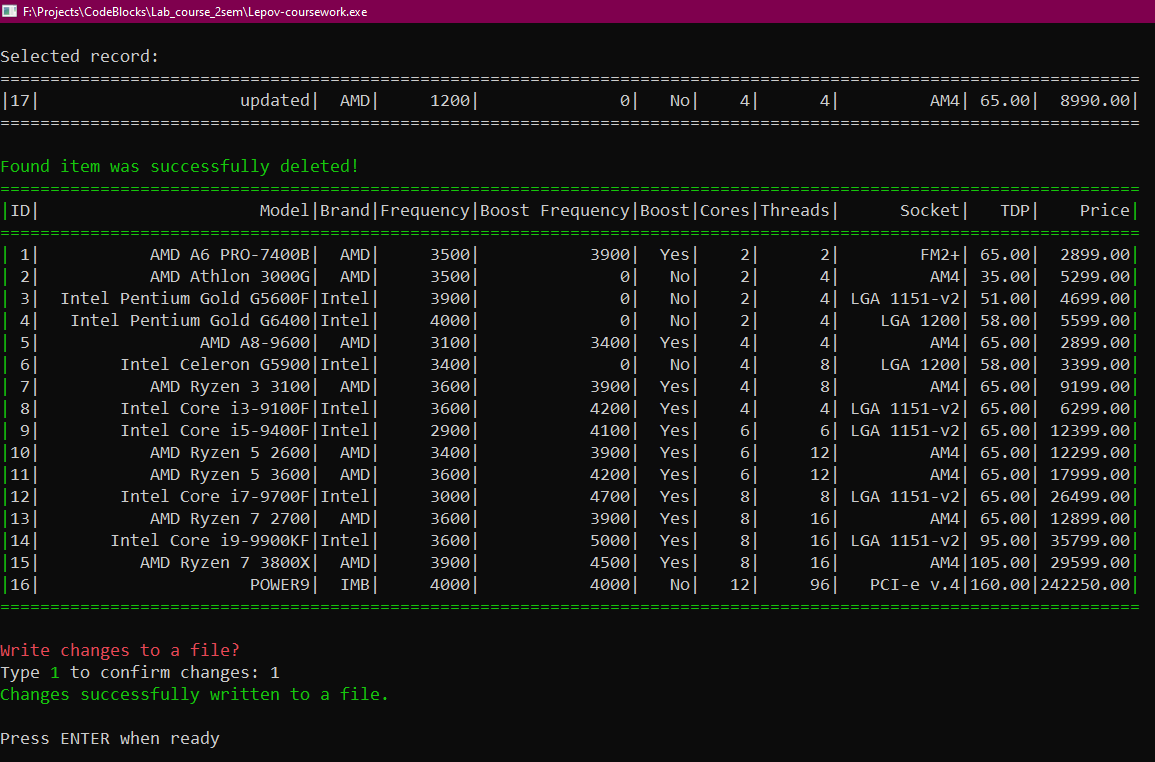
1. *Результат компиляции программного кода*



1. *Результат компиляции программного кода*



1. *Результат компиляции программного кода*



1. *Результат компиляции программного кода*

**Выводы.**

В результате работы над курсовой работой удалось выполнить составление электронной картотеки для предметной области «процессоры».