**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 |  |  |
| Преподаватель |  | Хахаев И.А. |

Санкт-Петербург 2022

**ВВЕДЕНИЕ**

**Цель:** Полное решение содержательной задачи (содержательная и формальная постановка задачи, спецификация, включая описание диалога, выбор метода решения и структур данных, разработка алгоритма, программная реализация, тестирование и отладка, документирование).

**Формулировка задачи:** Создание электронной картотеки материнских плат

**ЗАДАНИЕ**

Создать интерактивную картотеку материнских плат. При работе с программой пользователь сможет использовать удобное меню, пункты меню следующие:

0. Info - посмотреть инструкции к программе

1. Output - отобразить картотеку в виде таблицы на экране

2. Add - добавить новую запись в картотеку

3. Delete - удалить записи картотеки

4. Edit - редактировать записи картотеки

5. Search - поиск по картотеке

6. Sort - сортировка по картотеке

7. Save – сохранить в файл все изменения

8. Exit - сохранить все изменения на диске и выйти из программы

**ОПИСАНИЕ ОБЩЕЙ АРХИТЕКТУРЫ ДАННЫХ**

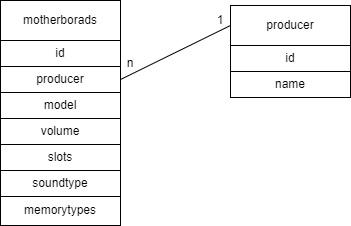
Архитектура данных построена на 2 структурах — списках.

Один список — двусвязный, то есть имеют связь между элементами, как на следующий так и на предыдущий элемент.

Эти списки используются для сохранения производителя платы.

Оставшейся же список — главная структура, со всеми остальными данными о материнских платах. Далее приведена ER – схема показывающая связь этих сущностей.

Для удобства написания кода программы, структуры и прототипы функций вынесены в отдельный h файл (файл - библиотека).



**ОПИСАНИЕ СТРУКТУР**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| **1** | **Producers** | **struct PRODUCER** | **Двусвязный список для моделей** |
|  | name | char | Производитель |
|  | id | int | Поле id |
|  | next | Model\* | Указатель на следующий элемент списка |
|  | prev | Model\* | Указатель на предыдущий элемент списка |

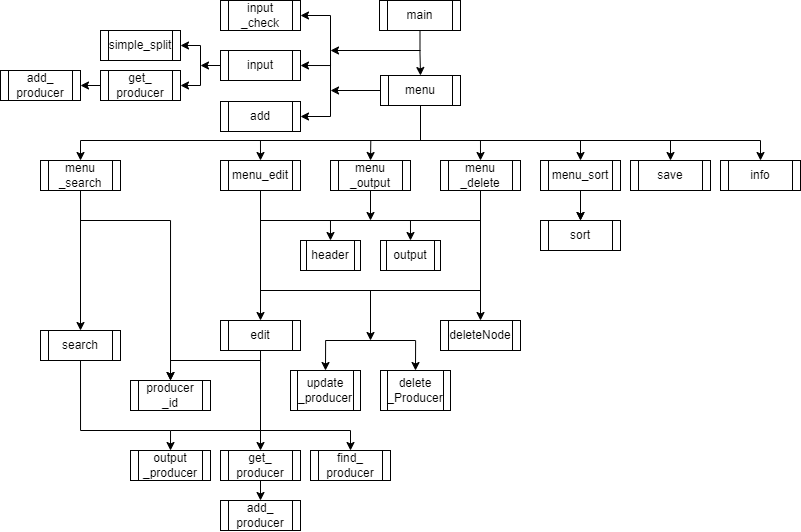
|  |  |  |  |
| --- | --- | --- | --- |
| **2** | **motherboards** | **struct BOARD** | **Односвязный главный список картотеки** |
|  | id | int | Поле id |
|  | producer | Producers | Производитель |
|  | model | char | Модель |
|  | volume | int | Объем допустимой памяти ОЗУ |
|  | slots | int | Кол-во слотов под ОЗУ |
|  | pci | float | PCI |
|  | soundtype | float | Тип звукового драйвера |
|  | memorytypes | int [3] | Поддерживаемые частоты ОЗУ |
|  | next | Producers\* | Указатель на следующий элемент списка |

**ОПИСАНИЕ ФУНКЦИЙ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Тип | Название | Входные данные | Назначение функции |
| 1 | int | main | - | Входная точка программы |
| 2 | motherboards\* | add | motherboards \*board, motherboards \*tmp | Добавление записи в список |
| 3 | void | add\_producer | Producers \*\*producer, char \*name | Добавление производителя в список |
| 4 | void | deleteNode | motherboards\*\* board, int key | Удаление записей картотеки |
| 5 | void | deleteProducer | Producers\*\* producer, Producers\* del | Удаление производителя |
| 6 | motherboards \* | edit | motherboards \*current, Producers \*producer, int status | Редактирование записей |
| 7 | bool | find\_producer | Producers\* producer, int id | Проверка на наличие введенного производителя в списке |
| 8 | Producers | get\_producer | Producers \*\*producer, char \*name | Ввод производителей |
| 9 | void | header | - | Вывод таблицы |
| 10 | void | info |  | Вывод информации о программе |
| 11 | int | input\_check | char line[MAXLEN] | Проверка на правильный ввод |
| 12 | void | menu | motherboards \*board, Producers \*producer, int count | Меню |
| 13 | motherboards\* | menu\_delete | motherboards \*board, int count | Меню для удаления |
| 14 | void | menu\_edit | motherboards \*board, Producers \*producer, int count | Меню для редактирования |
| 15 | void | menu\_output | motherboards \*board | Меню для вывода списка |
| 16 | void | menu\_search | motherboards \*board, Producers\* producer | Меню для поиска |
| 17 | motherboards\* | menu\_sort | motherboards \*board | Меню для сортировки |
| 18 | void | output | motherboards \*board | Вывод списка |
| 19 | void | output\_producer | Producers\* producer | Вывод производителей |
| 20 | void | producer\_id | Producers\* producer | Счетчик id для производителей |
| 21 | motherboards\* | input | char \*s, char sep, Producers \*\*producer, int count | Ввод данных в список |
| 22 | void | save | motherboards \*board | Сохранение в файл |
| 23 | void | search | motherboards \*board, Producers\* producer, int search\_status | Поиск |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Тип | Название | Входные данные | Назначение функции |
| 24 | char \*\* | simple\_split | char \*line, const char sep | Считывание со строки с разделителями |
| 25 | motherboards \* | sort | motherboards \*root, bool (\*comp)(motherboards\*, motherboards\*, int), int n | Сортировка |
| 26 | Producers\* | update\_producer | motherboards\* board, Producers\* producer | Обновление связей для списка производителей |
| 27 | bool | sort\_id | motherboards\* a, motherboards\* b, int n | Сортировки по всем полям списка |
| 28 | bool | sort\_producer |
| 29 | bool | sort\_model |
| 30 | bool | sort\_volume |
| 31 | bool | sort\_slots |
| 32 | bool | sort\_pci |
| 33 | bool | sort\_soundtype |
| 34 | bool | sort\_types0 |
| 35 | bool | sort\_types1 |
| 36 | bool | sort\_types2 |

**Схема вызова функций**

****

**ОПИСАНИЕ ПЕРЕМЕННЫХ**

**Функция main**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| 1 | producers | Producers \* | Массив структур моделей |
| 2 | board | motherboards \* | Массив структур материнских плат |
| 3 | tmp | motherboards \* | Временный массив структур материнских плат |
| 4 | file | FILE\* | Файл |
| 5 | s | char | Входная строка |
| 6 | count | int | Счетчик строк |

**Функция input**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | count | int | Счетчик |
| 2 | producer | Producers \* | Массив структур производителей |
| 3 | tmp | motherboards \* | Временный массив структур материнских плат |
| 4 | i | int | Cчетчик для цикла |
| 5 | s | char\* | Входная строка |
| 6 | s1 | char\*\* | Временная строка |
| 7 | sep | char\* | Строка разделителей |

**Функция menu**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | board | motherboards \* | Массив структур материнских плат |
| 2 | producer | Producers \* | Массив структур производителей |
| 3 | tmp | motherboards \* | Временный массив структур материнских плат |
| 4 | count | int | Счетчик |
| 5 | s | char | Входная строка |
| 6 | menu\_status | int | Статус меню |
| 7 | input\_status | int | Статус ввода |

**Функция add\_producer**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | producers | Producers \*\* | Массив структур производителей |
| 2 | name | char\* | Строка модели |
| 3 | new\_model | Producers\* | Новая запись о производителе |
| 4 | current | Producers\* | Текущий элемент |

**Функция get\_producer**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | producer | Producers \*\* | Массив структур производителей |
| 2 | name | char\* | Входная строка |
| 3 | current | Producers\* | Текущий элемент |

**Функция add**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | board | motherboards \* | Массив структур материнских плат |
| 2 | tmp | motherboards \* | Временный массив структур материнских плат |
| 3 | current | motherboards \* | Текущий элемент |

**Функция menu\_output**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | board | motherboards \* | Массив структур материнских плат |
| 2 | menu\_status | int | Статус меню |

**Функция menu\_delete**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | board | motherboards \* | Массив структур материнских плат |
| 2 | tmp | motherboards \* | Временный массив структур материнских плат |
| 3 | count | int | Счетчик |
| 4 | id | int | Id элемента для удаления |

**Функция menu\_edit**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | board | motherboards \* | Массив структур материнских плат |
| 2 | producer | Producers \* | Массив структур производителей |
| 3 | current | Producers\* | Текущий элемент |
| 4 | status | int | Статус выбора редактирования |
| 5 | status1 | int | Статус выбора редактирования |
| 6 | count | int | Счетчик |

**Функция menu\_search**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | board | motherboards \* | Массив структур материнских плат |
| 2 | choice | int | Статус выбора поиска |

**Функция menu\_sort**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | board | motherboards \* | Массив структур материнских плат |
| 2 | choice | int | Статус выбора сортировки |
| 3 | tеmp | motherboards \* | Временный массив структур материнских плат |
| 4 | choice1 | int | Статус выбора сортировки |

**Функция save**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | board | motherboards \* | Массив структур материнских плат |
| 2 | tmp | motherboards \* | Временный массив структур материнских плат |
| 3 | save | FILE\* | Файл |

**Функция output**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | board | motherboards \* | Массив структур материнских плат |
| 2 | tmp | motherboards \* | Временный массив структур материнских плат |

**Функция input\_check**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | i | int | Cчетчик для цикла |
| 2 | line | char | Входная строка |
| 3 | count | int | Cчетчик |

**Функция deleteNode**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | board | motherboards \*\* | Массив структур материнских плат |
| 2 | key | int | Id для удаления |
| 3 | tmp | motherboards \* | Временный массив структур материнских плат |
| 4 | prev | motherboards \* | Предыдущий элемент списка |

**Функция deleteProducers**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | producer | Producers \*\* | Массив структур производителей |
| 2 | s | char\* | Входная строка производителя |
| 3 | current | Producers\* | Текущий элемент |

**Функция edit**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | producers | Producers \* | Массив структур производителей |
| 2 | status | int | Статус редактирования |
| 3 | input | int | Статус ввода |
| 4 | current | motherboards \* | Текущий элемент |
| 5 | s | char | Входная строка |
| 6 | old\_name | char | Старая строка |

**Функция producer\_id**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | producer | Producers \* | Массив структур производителей |
| 2 | count | int | Счетчик |

**Функция output\_producer**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | producer | Producers \* | Массив структур производителей |
| 2 | tmp | motherboards \* | Временный массив структур материнских плат |

**Функция update\_producer**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | producer | Producers \* | Массив структур производителей |
| 2 | board | motherboards \*\* | Массив структур материнских плат |

**Функция find\_producer**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | producer | Producers \* | Массив структур производителей |
| 2 | word | char | Входная строка |

**Функция search**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | board | motherboards \* | Массив структур материнских плат |
| 2 | current | Producers\* | Текущий элемент |
| 3 | s2 | int | Строчка в формате float |
| 4 | tmp | motherboards \* | Временный массив структур материнских плат |
| 5 | count | int | Счетчик |
| 6 | s1 | int | Строчка в формате int |
| 7 | s | char | Входная строка |
| 8 | search\_status | int | Статус выбора поиска |
| 9 | menu\_status | int | Статус меню |

**Функция sort**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | root | motherboards \* | Массив структур материнских плат |
| 2 | current | motherboards \* | Текущий элемент |
| 3 | new\_root | motherboards \* | Новый указатель |
| 4 | comp | bool\* | Компаратор по всем сортировкам |
| 5 | n | int | Статус сортировки |

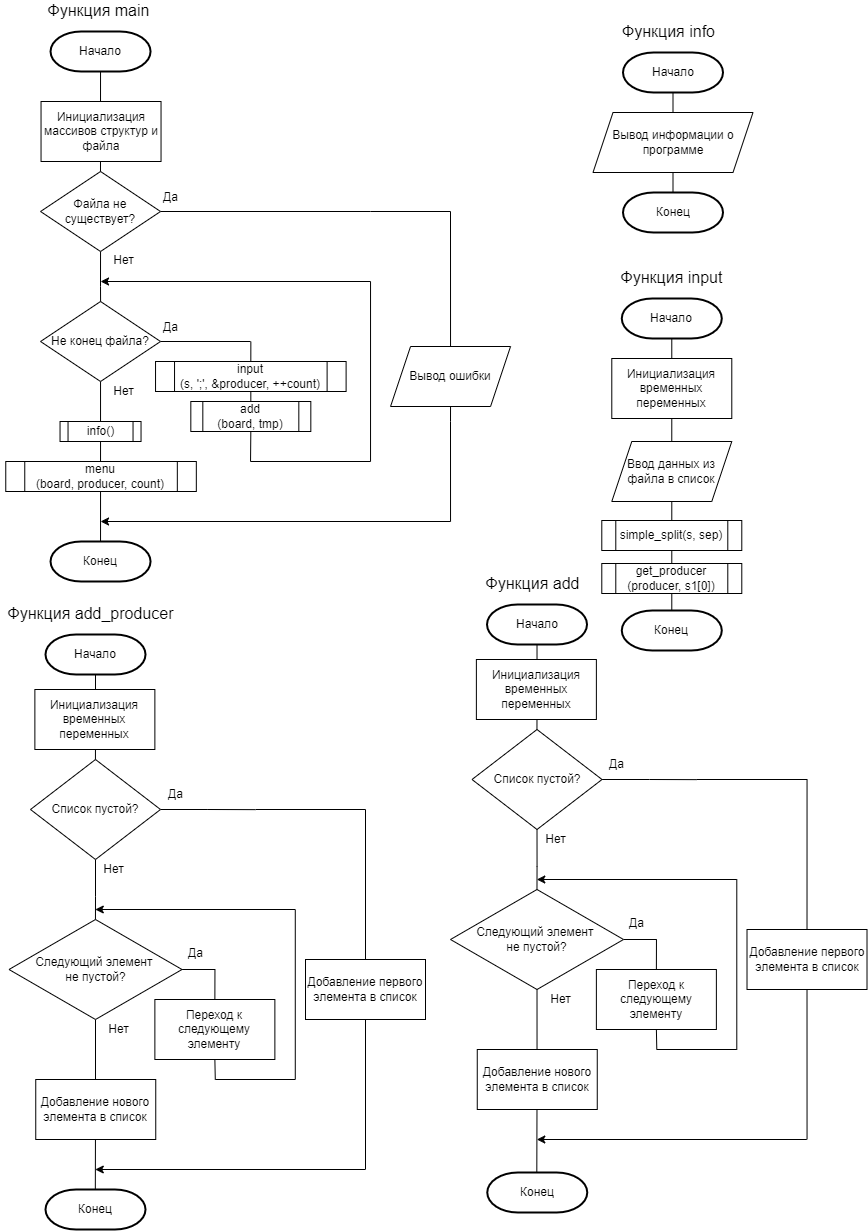
**Функции sort\_….**

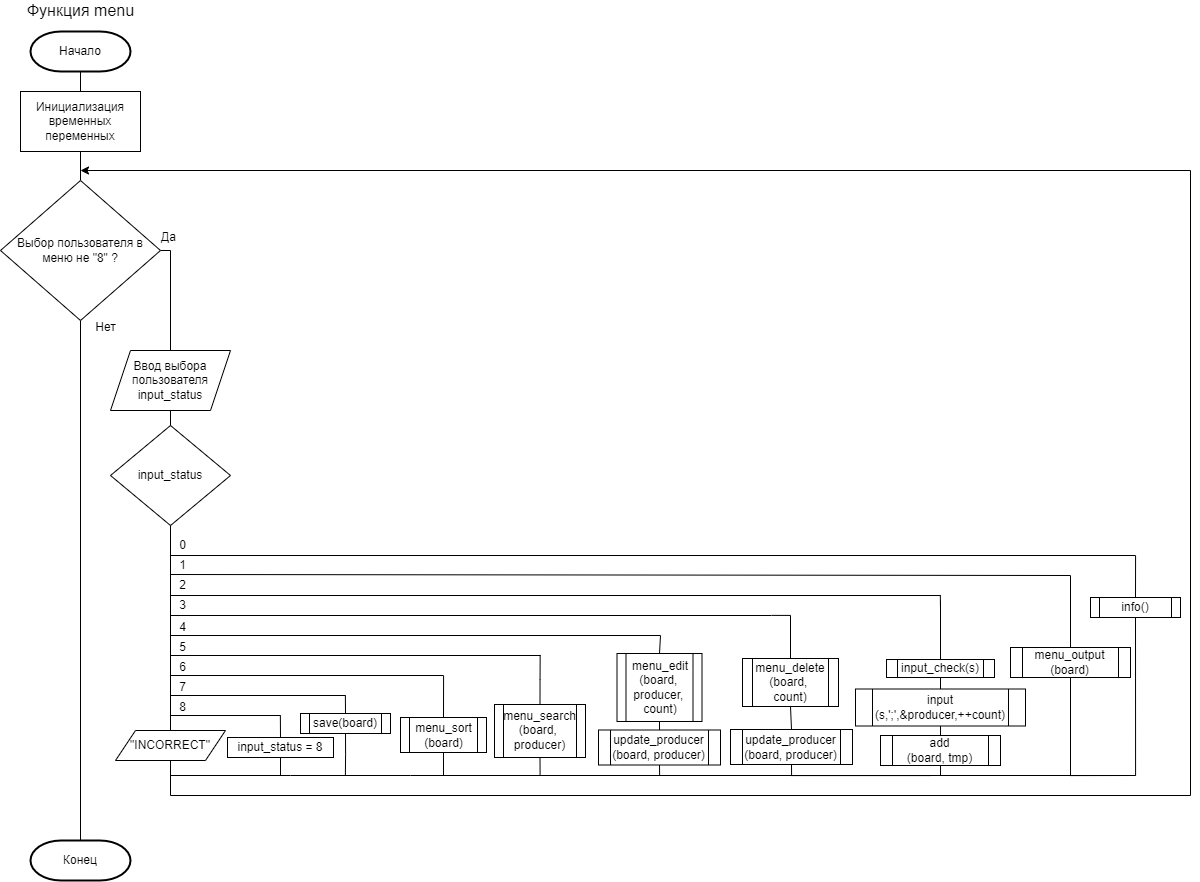
|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | а | motherboards \* | Массив структур материнских плат |
| 2 | n | int | Статус сортировки |

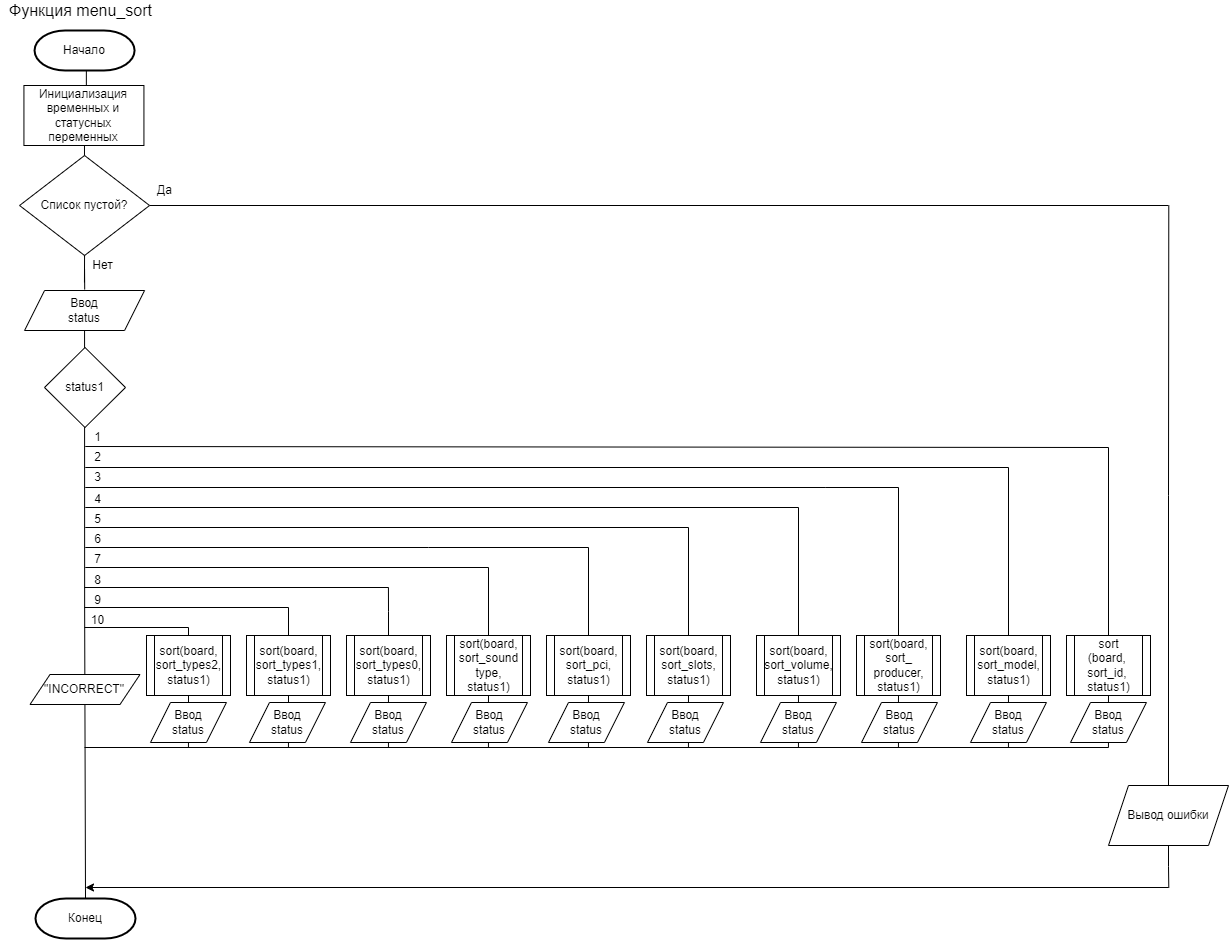
**Функция simple\_split**

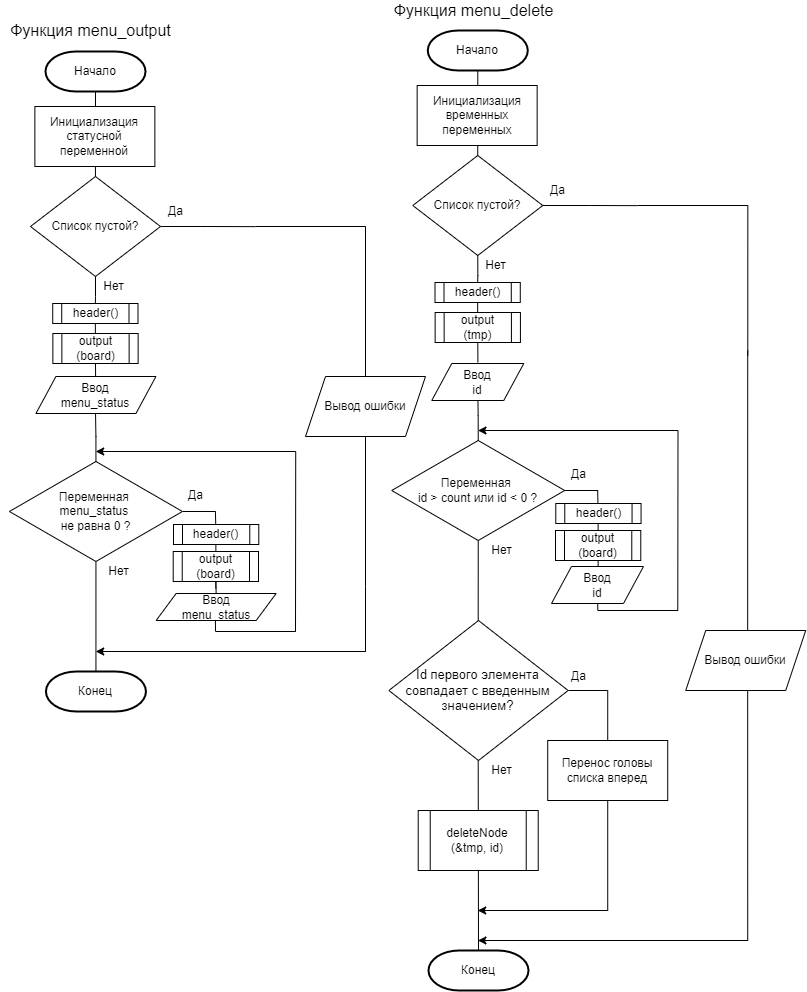
|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменно | Тип | Назначение |
| 1 | line | char\* | Входная строка |
| 2 | sep | const char | Разделитель |

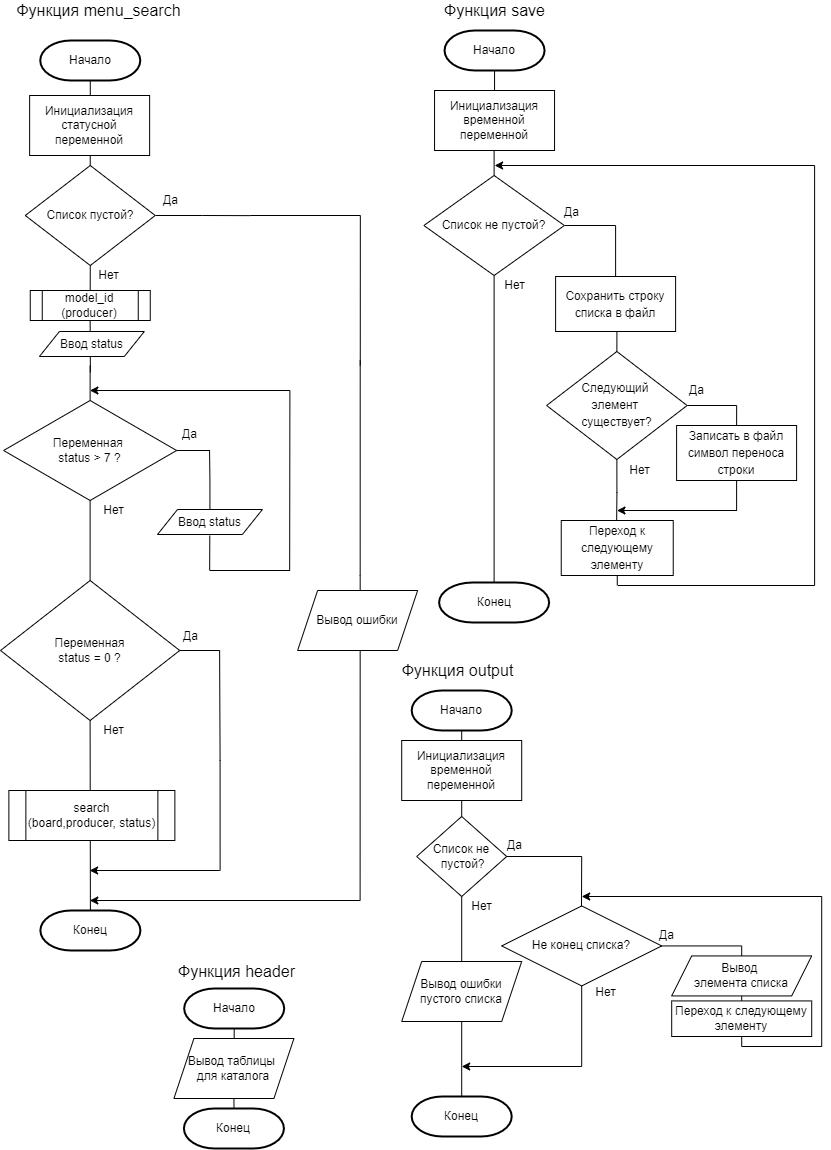
**Схемы алгоритмов:**

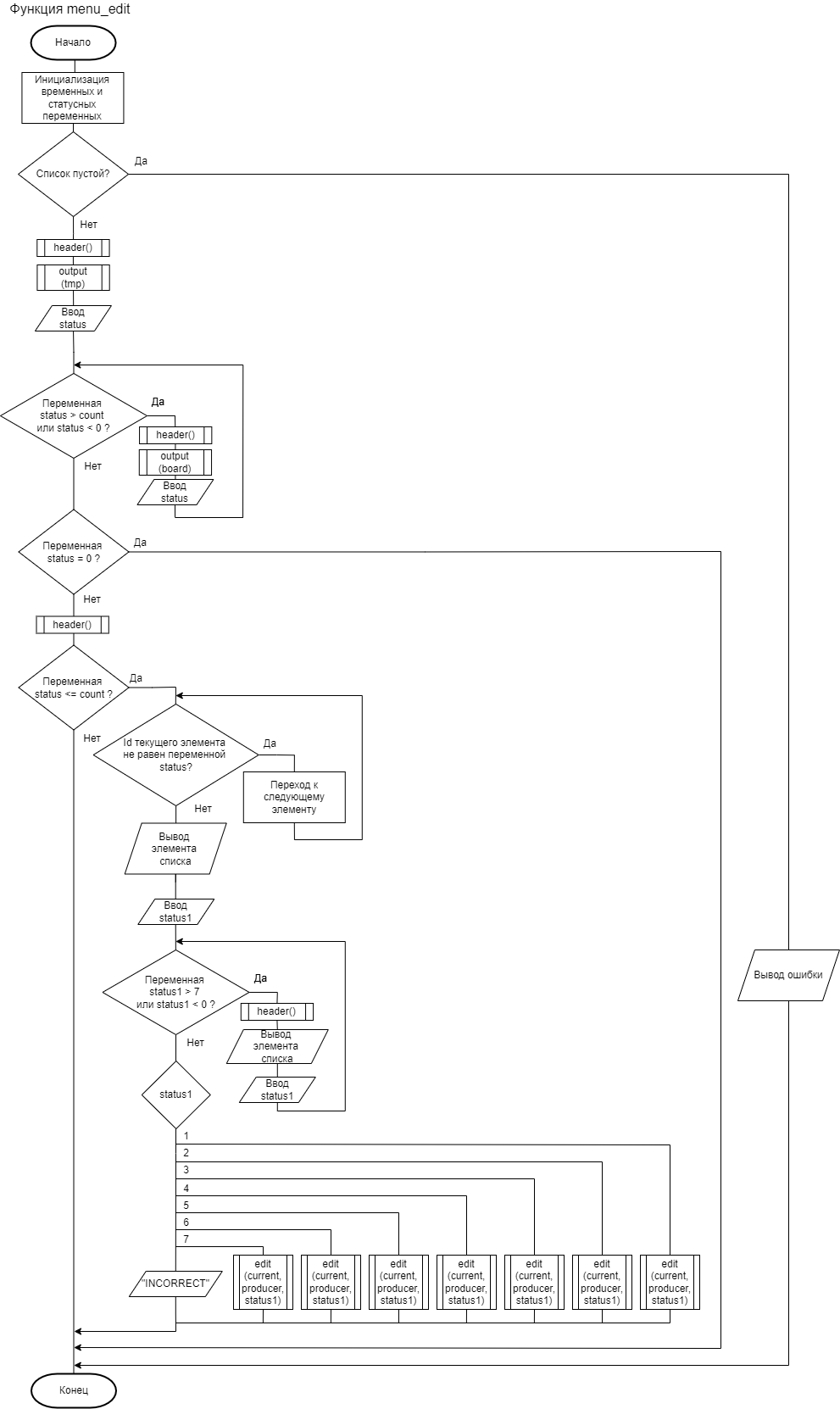
****

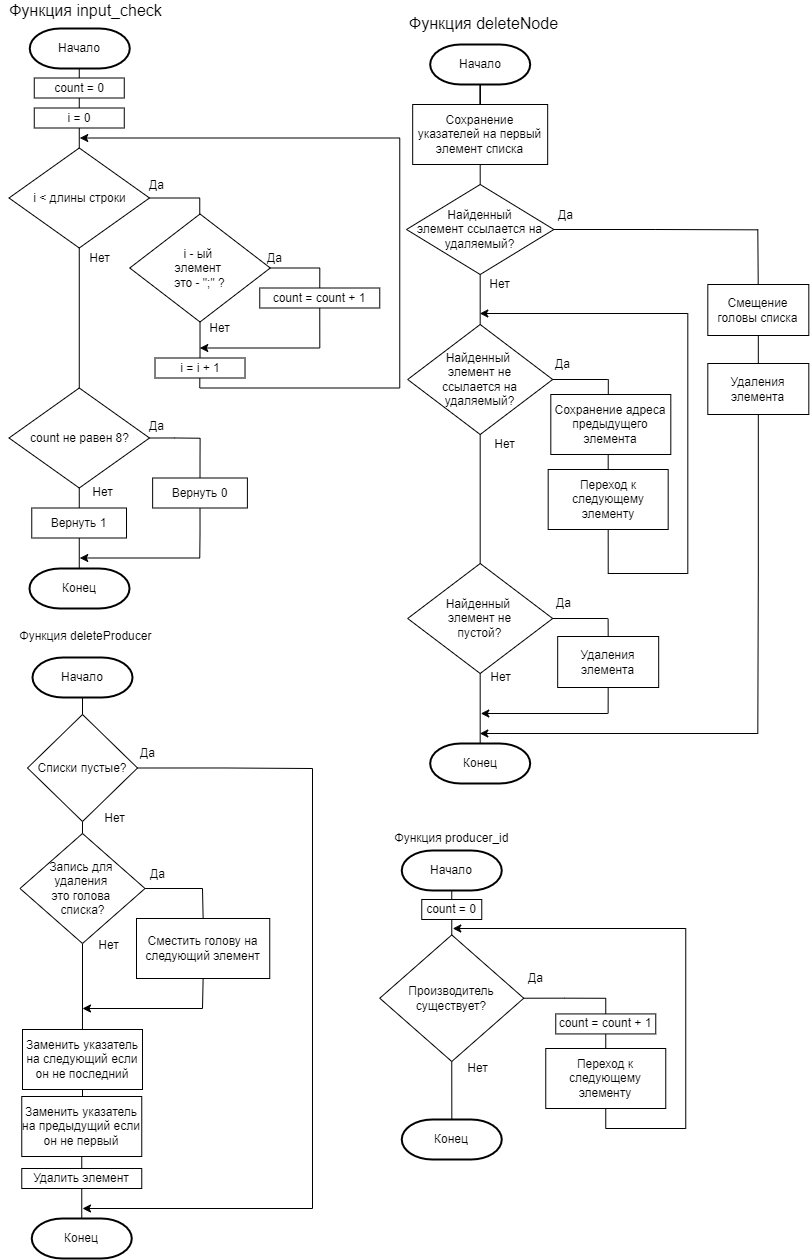
****

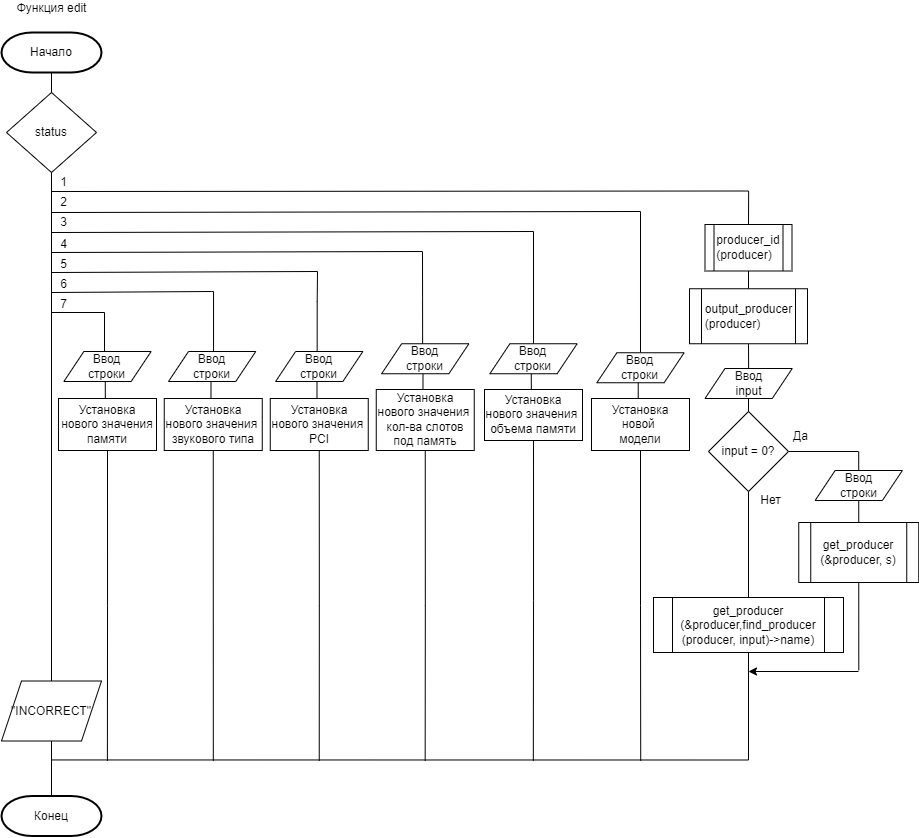
****

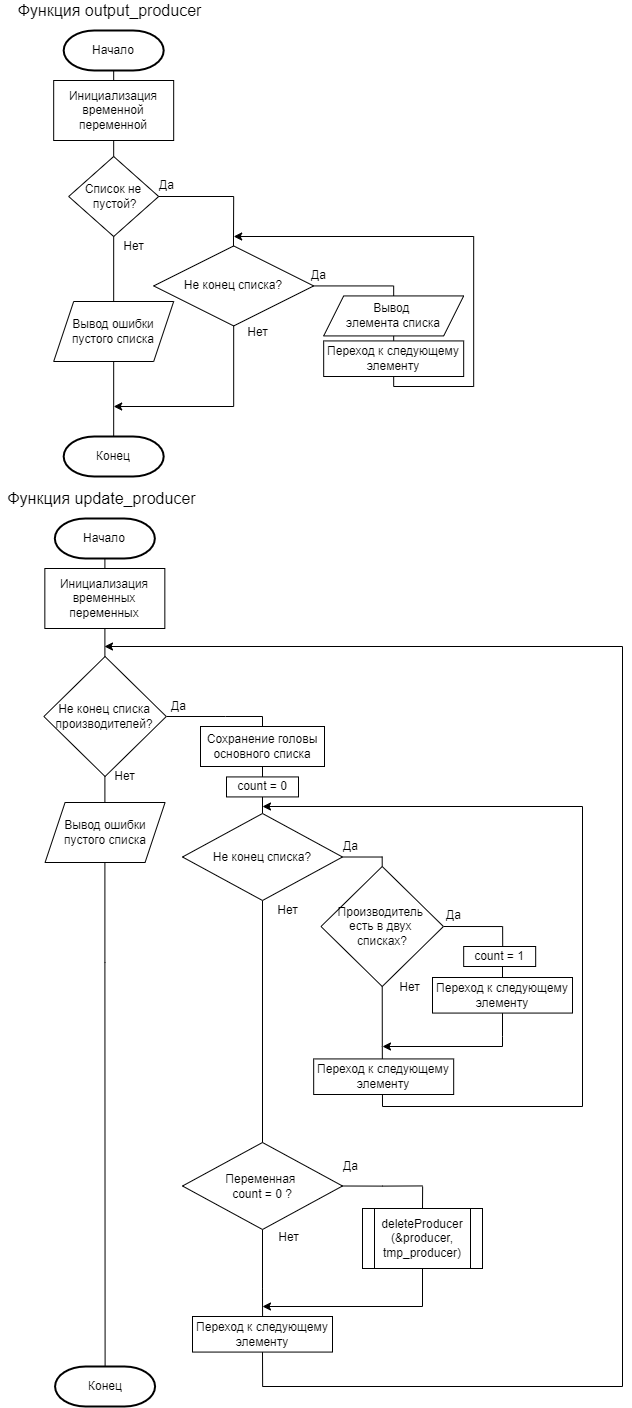
****

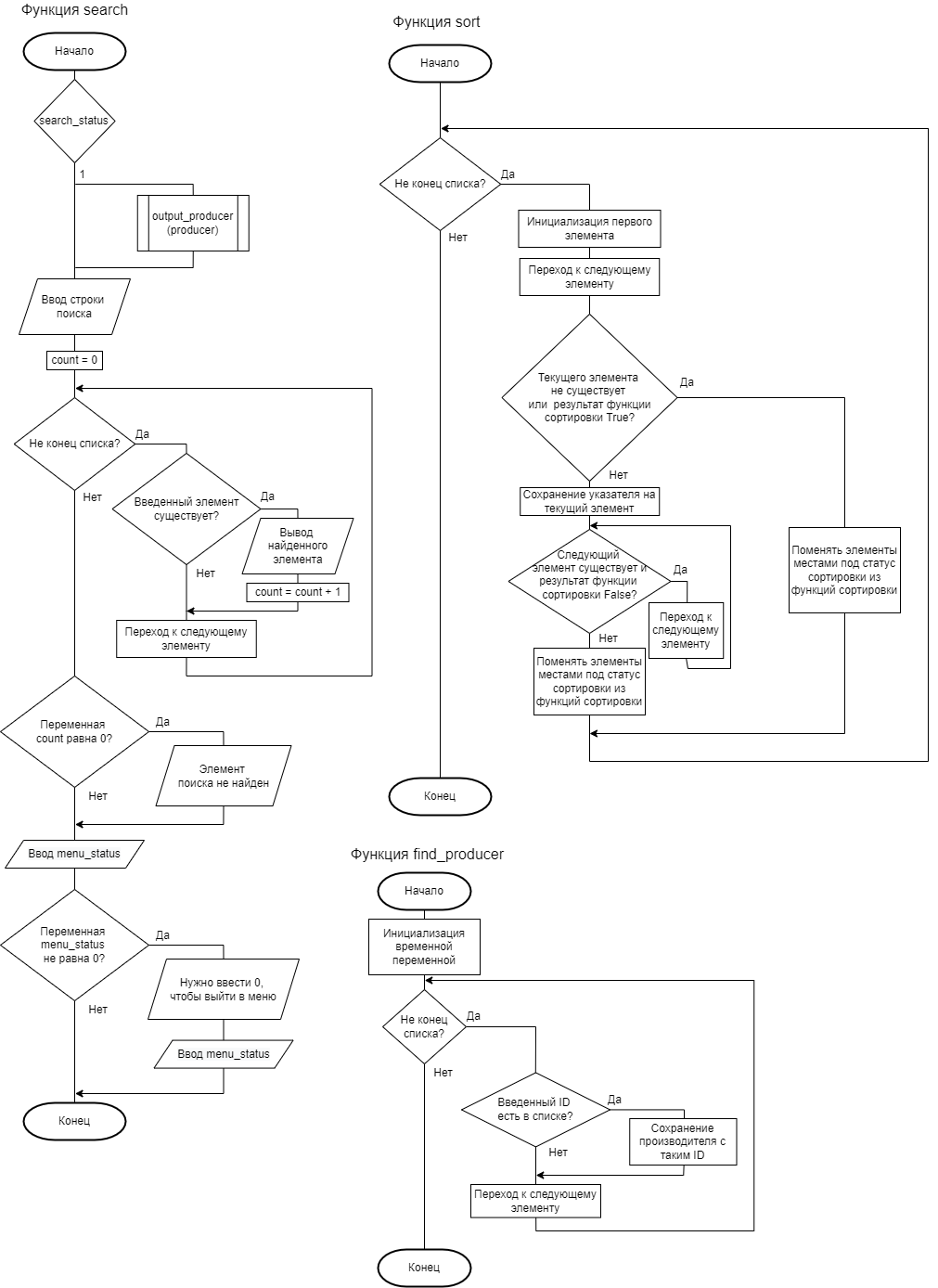
****

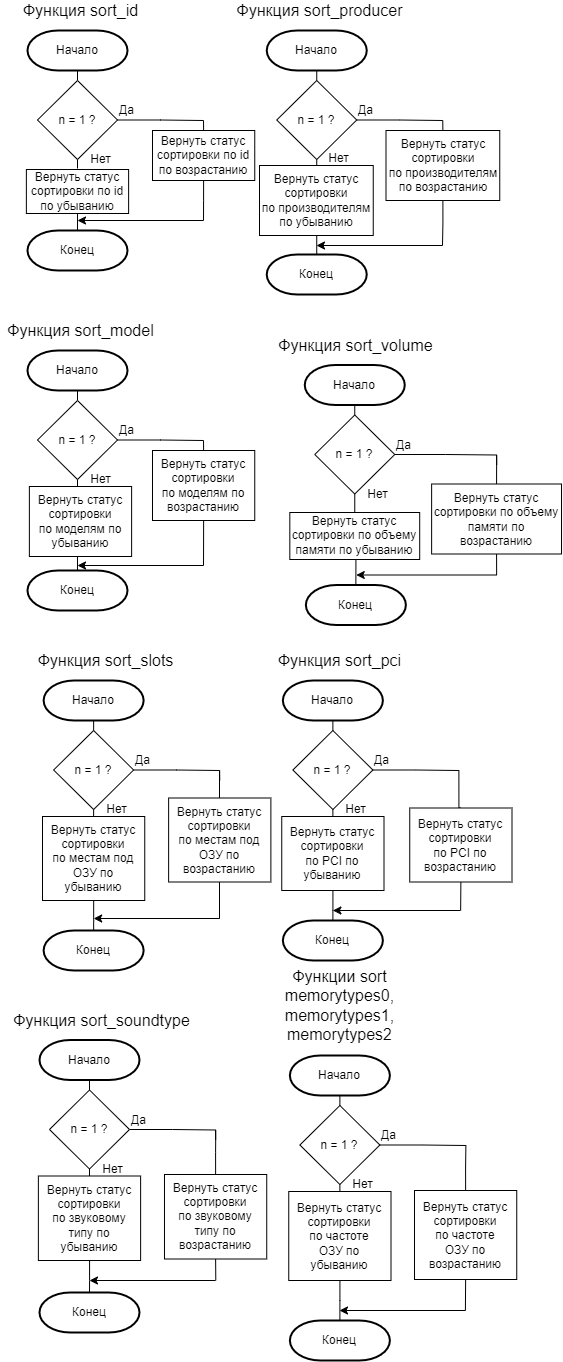
****

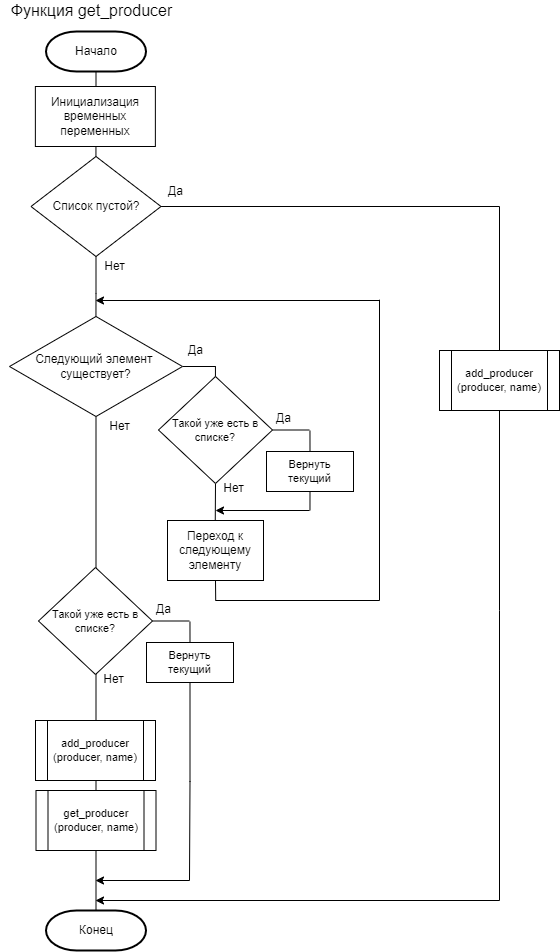
****

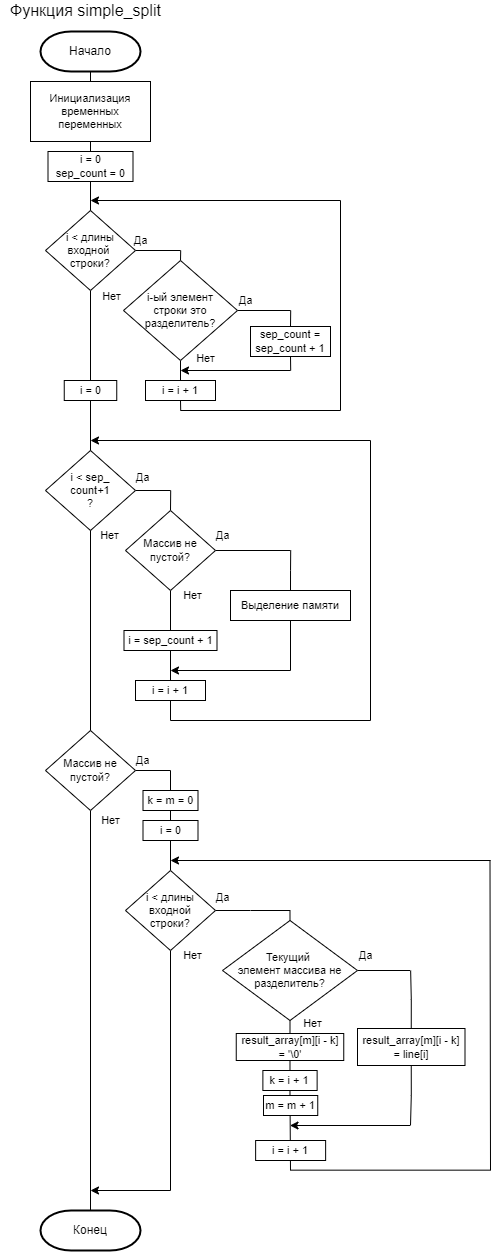
****

****

****

****

****

****

**Заключение.**

В ходе работы были использованы следующие заголовочные файлы и функции из них:

Библиотека <stdio.h>

• FILE

• fopen

• fprintf

• fclose

• fgets

• scanf

• stdout

• stdin

• getchar

• feof

Библиотека <stdlib.h>

• system(“clear||@cls”)

Библиотека <string.h>

• strlen

• strcmp

В результате выполнения работы научился работать со строками, функциями и файлами, улучшил навыки работы с обычными циклами, такими как while и for. Были изучены способы обработки текстовой информации, представленной в виде динамических массивов символов (строк), получаемых как из стандартного потока ввода, так и из текстового файла.

**ТЕКСТ ПРОГРАММЫ**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

/\*Structures and function prototypes\*/

#include "s&f.h"

/\*Program entry point\*/

int main()

{

Producers \*producer=NULL;

motherboards \*board=NULL;

motherboards \*tmp=NULL;

FILE \*file;

char s[MAXLEN];

int count;

/\*INPUT from file\*/

if ((file = fopen("file-win-.csv", "r")) == NULL) /\*Error - if the file don't exist\*/

{

clear

puts("Could not open file.csv\nRestart the program with a right file");

exit(0);

}

while ((fgets(s, MAXLEN, file)) != NULL) /\*Input from file\*/

{

tmp = input(s, ';', &producer, ++count);

board = add(board, tmp);

}

/\*INPUT from file\*/

/\*Start of the program\*/

info();

menu(board, producer, count);

clear

puts("That's all!\n\nCoursework was made by UGRUMOV MIHAIL group 1307\n\nHave a nice day!\n");

fclose(file);

return 0;

}

/\*information of program, help for user\*/

void info()

{

puts("Hello! This coursework was made by UGRUMOV MIHAIL group 1307\n");

puts("Below you can read about my program\n\n");

puts("Coursework: 'Development of an electronic catalogue'\n");

puts("0. Info - to see instruction");

puts("1. Output - to output catalogue");

puts("2. Add - to adding cards about objects in the subject area");

puts("3. Delete - to delete card by id from catalogue");

puts("4. Edit - to edit card, you must choose id and edit this card by any parameter");

puts("5. Search - to search trough catalogue");

puts("6. Sort - to sort catalogue by any field, increasing and decreasing");

puts("7. Save to file - to save all changes to file");

puts("8. Exit - to exit\n");

puts("\n\nPress ENTER to go to menu\n");

getchar();

}

/\*Input\*/

motherboards \*input(char \*s, char sep, Producers \*\*producer, int count)

{

motherboards \*tmp;

char \*\*s1 = NULL;

int i;

tmp = (motherboards \*) malloc(sizeof(motherboards));

if(!tmp)

{

printf("MEMORY ERROR\n");

exit(0);

}

s1 = simple\_split(s, sep);

tmp->id=count;

tmp->producer = get\_producer(producer, s1[0]);

tmp->model = s1[1];

tmp->volume = atoi(s1[2]);

tmp->slots = atoi(s1[3]);

tmp->pci = atof(s1[4]);

tmp->soundtype = atof(s1[5]);

for (i=0; i<3; i++) tmp->memorytypes[i] = atoi(s1[6+i]);

tmp->next = NULL;

return tmp;

}

/\*main menu of the program\*/

void menu(motherboards \*board, Producers \*producer, int count)

{

motherboards \*tmp=NULL;

char s[MAXLEN];

int menu\_status, input\_status;

while (input\_status != 8) /\*Mane MENU\*/

{

clear

menu\_status=1;

printf("0. Info\n1. Output catalogue \n2. Adding cards \n3. Deleting cards\n4. Editing cards\n5. Search through catalogue\n6. Sorting the catalogue\n7. Save to file\n8. Exit\n\nChoice: ");

scanf("%d", &input\_status);

clear

switch(input\_status)

{

/\* INFO \*/

case 0:

info();

getchar();

break;

/\* Catalogue output \*/

case 1:

menu\_output(board);

break;

/\* Adding cards about objects in the subject area \*/

case 2:

while (menu\_status==1)

{

clear

puts("Enter new card\n(Example: MSI;H310M;32;2;2.0;7.1;2133;2400;2666):\n");

scanf("%s", s);

menu\_status = input\_check(s);

if (menu\_status == 1)

{

tmp = input(s, ';', &producer, ++count);

board = add(board, tmp);

clear

puts("Print\n1 - to add more\n0 - to go to MENU\nChoice:");

scanf("%d", &menu\_status);

}

else

{

printf("\nInput error!\nPrint\n1 - to retry input\n0 - to return to MENU\n");

scanf("%d", &menu\_status);

}

}

break;

/\* Deleting cards \*/

case 3:

board = menu\_delete(board, count);

producer = update\_producer(board, producer);

if (board==NULL) free(board);

break;

/\* Editing cards \*/

case 4:

menu\_edit(board, producer, count);

producer = update\_producer(board, producer);

break;

/\* Search cards by parameter \*/

case 5:

menu\_search(board, producer);

break;

/\* Sorting the card index by parameter \*/

case 6:

board = menu\_sort(board);

break;

/\* Save to file \*/

case 7:

save(board);

break;

/\* Exit \*/

case 8:

input\_status = 8;

break;

default:

printf("Incorrect\n\n");

}

}

}

/\*Add producer to double linked list\*/

void add\_producer(Producers \*\*producer, char \*name)

{

Producers \*new\_model, \*current;

new\_model = (Producers \*) malloc(sizeof(Producers));

if(!new\_model)

{

printf("MEMORY ERROR\n");

exit(1);

}

strcpy(new\_model->name, name);

new\_model->next = NULL;

if (\*producer == NULL)

{

new\_model->prev = NULL;

\*producer = new\_model;

}

else

{

current = \*producer;

while (current->next != NULL)

{

current = current->next;

}

new\_model->prev = current;

current->next = new\_model;

}

}

/\*Get producer from input\*/

Producers \*get\_producer(Producers \*\*producer, char \*name)

{

Producers \*current;

if (\*producer == NULL)

{

add\_producer(producer, name);

return \*producer;

}

current = \*producer;

while (current->next != NULL)

{

if (strcmp(current->name, name) == 0)

{

return current;

}

current = current->next;

}

if (strcmp(current->name, name) == 0)

{

return current;

}

add\_producer(producer, name);

return get\_producer(producer, name);

}

/\*Add note from file to singly linked list\*/

motherboards \*add(motherboards \*board, motherboards \*tmp)

{

motherboards \*current;

if (board == NULL)

{

board = tmp;

return board;

}

current = board;

while (current->next != NULL)

{

current = current->next;

}

current->next = tmp;

return board;

}

/\*menu of output catalogue\*/

void menu\_output(motherboards \*board)

{

int menu\_status;

if (board==NULL)

{

puts("List is empty\n\n");

puts("Press enter to return to Menu\n");

getchar();

getchar();

return;

}

header();

output(board);

printf("\nPrint 0 to return menu\n");

scanf("%d", &menu\_status);

while (menu\_status!=0)

{

clear

header();

output(board);

printf("\nError! You must print 0 to return menu\n");

scanf("%d", &menu\_status);

}

}

/\*menu of delete node in catalogue\*/

motherboards\* menu\_delete(motherboards \*board, int count)

{

motherboards\* tmp;

int id;

tmp=board;

if (tmp==NULL) /\*Error input\*/

{

puts("List is empty\n\n");

puts("Press enter to return to Menu\n");

getchar();

getchar();

return tmp;

}

header();

output(tmp);

puts("Input element id to delete or 0 to return to MENU: "); /\*Choice of deleting item\*/

scanf("%d", &id);

while (id > count || id<0)

{

clear

header();

output(tmp);

puts("\nInput error! Retry!\n");

puts("Input element id to delete or 0 to return to MENU: "); /\*Error input\*/

scanf("%d", &id);

clear

}

if (tmp->id==id) tmp=tmp->next;

else deleteNode(&tmp, id);

return tmp;

}

/\*menu of edit any node in catalogue\*/

void menu\_edit(motherboards \*board, Producers \*producer, int count)

{

motherboards \*current;

int status, status1;

status = 1;

current = board;

if (board==NULL) /\*Error if the list is empty\*/

{

puts("\n\nList is empty\n\n");

puts("Press enter to return to Menu\n");

getchar();

getchar();

return;

}

header();

output(board);

puts("\nSelect element by id to edit\nOr print 0 to go to MENU\nChoice: ");

scanf("%d", &status);

clear

while (status > count || status < 0)

{

header();

output(board);

puts("Input error! Retry!\n"); /\*Error input\*/

puts("\nSelect element by id to edit\nOr print 0 to go to MENU\nChoice: ");

scanf("%d", &status);

clear

}

if (status == 0) return;

header();

if (status <= count)

{

while (current->id!=status) current = current->next;

printf("| %3d| %16s| %16s| %7d| %7d| %5.1f| %12.1f| %4d| %4d| %4d|\n",current->id,current->producer->name,current->model,current->volume,

current->slots,current->pci,current->soundtype,current->memorytypes[0],current->memorytypes[1],current->memorytypes[2]);

puts("\nSelect list field to edit:\n1 - Producer\n2 - Model\n3 - Volume\n4 - Slots\n5 - PCI\n6 - Sound type\n7 - Memory types\n\nChoice: ");

scanf("%d", &status1); /\*Choice of editing especially field\*/

while (status1 > 7 || status1 < 0)

{

clear

header();

printf("| %3d| %16s| %16s| %7d| %7d| %5.1f| %12.1f| %4d| %4d| %4d|\n",current->id,current->producer->name,current->model,current->volume,

current->slots,current->pci,current->soundtype,current->memorytypes[0],current->memorytypes[1],current->memorytypes[2]);

puts("\nInput error! Retry!\n"); /\*Error input\*/

puts("Select list field to edit:\n1 - Producer\n2 - Model\n3 - Volume\n4 - Slots\n5 - PCI\n6 - Sound type\n7 - Memory types\n\nChoice: ");

scanf("%d", &status1);

clear

}

switch(status1) /\*Editing fields\*/

{

case 1:

current = edit(current,producer,status1);

break;

case 2:

printf("Print new model:\n");

current = edit(current,producer,status1);

break;

case 3:

printf("Print new volume:\n");

current = edit(current,producer,status1);

break;

case 4:

printf("Print new slots:\n");

current = edit(current,producer,status1);

break;

case 5:

printf("Print new PCI:\n");

current = edit(current,producer,status1);

break;

case 6:

printf("Print new Sound type:\n");

current = edit(current,producer,status1);

break;

case 7:

printf("What memory type do you want to edit?\nPrint\n1 - to edit first\n2 - to edit second\n3 - to edit third");

current = edit(current,producer,status1);

break;

default:

printf("Incorrect\n\n");

}

}

else

{

clear

printf("No such element in catalogue!\n\nRetry!\n\n"); /\*Error if the chosen element doesn't exist\*/

}

}

/\*menu of search in catalogue\*/

void menu\_search(motherboards \*board, Producers\* producer)

{

int status;

if (board==NULL) /\*Error if the catalogue is empty\*/

{

puts("List is empty\n\n");

puts("Press enter to return to Menu\n");

getchar();

getchar();

return;

}

puts("Select an option to search:\n1 - Producer\n2 - Model\n3 - Volume\n4 - Slots\n5 - PCI\n6 - Sound type\n7 - Memory types \nor 0 to return to MENU\n\nChoice: ");

producer\_id(producer);

scanf("%d", &status); /\*Choice of searching item\*/

clear

while (status > 7)

{

puts("Input error! Retry!\n"); /\*Error input\*/

puts("Select an option to search:\n1 - Producer\n2 - Model\n3 - Volume\n4 - Slots\n5 - PCI\n6 - Sound type\n7 - Memory types \nor 0 to return to MENU\n\nChoice: ");

scanf("%d", &status);

clear

}

if (status == 0) return;

else search(board,producer, status);

}

/\*menu of sort catalogue\*/

motherboards \*menu\_sort(motherboards \*board)

{

motherboards \*tmp=NULL;

int status, status1;

status = 0;

status1 = 0;

if (board==NULL) /\*Error if the catalogue is empty\*/

{

puts("List is empty\n\n");

puts("Press enter to return to Menu\n");

getchar();

getchar();

return tmp;

}

puts("Select an option to sort:\n1 - Id\n2 - Producer\n2 - Model\n3 - Volume\n4 - Slots\n5 - PCI\n6 - Sound type\n7 - Memory type 1\n9 - Memory type 2\n10 - Memory type 3\nor 0 to return to MENU\n\nChoice: ");

scanf("%d", &status); /\*Choice of sorting field\*/

clear

switch(status)

{

default: /\*Error input\*/

printf("Input error! Print any sign to return MENU\n\n");

scanf("%d", &status);

getchar();

return board;

case 0: /\*Exit\*/

break;

case 1:

puts("Sort selection:\n1 - Increasing\n2 - Decreasing\n\nChoice: ");

scanf("%d", &status1);

tmp = sort(board, sort\_id, status1);

break;

case 2:

puts("Sort selection:\n1 - Increasing\n2 - Decreasing\n\nChoice: ");

scanf("%d", &status1);

tmp = sort(board, sort\_model, status1);

break;

case 3:

puts("Sort selection:\n1 - Increasing\n2 - Decreasing\n\nChoice: ");

scanf("%d", &status1);

tmp = sort(board, sort\_producer, status1);

break;

case 4:

puts("Sort selection:\n1 - Increasing\n2 - Decreasing\n\nChoice: ");

scanf("%d", &status1);

tmp = sort(board, sort\_volume, status1);

break;

case 5:

puts("Sort selection:\n1 - Increasing\n2 - Decreasing\n\nChoice: ");

scanf("%d", &status1);

tmp = sort(board, sort\_slots, status1);

break;

case 6:

puts("Sort selection:\n1 - Increasing\n2 - Decreasing\n\nChoice: ");

scanf("%d", &status1);

tmp = sort(board, sort\_pci, status1);

break;

case 7:

puts("Sort selection:\n1 - Increasing\n2 - Decreasing\n\nChoice: ");

scanf("%d", &status1);

tmp = sort(board, sort\_soundtype, status1);

break;

case 8:

puts("Sort selection:\n1 - Increasing\n2 - Decreasing\n\nChoice: ");

scanf("%d", &status1);

tmp = sort(board, sort\_types0, status1);

break;

case 9:

puts("Sort selection:\n1 - Increasing\n2 - Decreasing\n\nChoice: ");

scanf("%d", &status1);

tmp = sort(board, sort\_types1, status1);

break;

case 10:

puts("Sort selection:\n1 - Increasing\n2 - Decreasing\n\nChoice: ");

scanf("%d", &status1);

tmp = sort(board, sort\_types2, status1);

break;

}

return tmp;

}

/\*Save to file after the program finished\*/

void save(motherboards \*board)

{

motherboards \*tmp;

FILE \*save;

save = fopen("save-win-.csv", "w");

for (tmp = board; tmp != NULL; tmp = tmp->next) {

fprintf(save, "%s;%s;%d;%d;%.1f;%.1f;%d;%d;%d",

tmp->producer->name, tmp->model, tmp->volume,

tmp->slots, tmp->pci,tmp->soundtype,

tmp->memorytypes[0], tmp->memorytypes[1], tmp->memorytypes[2]);

if (tmp->next != NULL) fprintf(save, "\n");

}

fclose(save);

}

/\*header for output catalogue\*/

void header()

{

printf("Motherboards catalogue:\n\n");

printf("+----+-----------------+-----------------+--------+--------+------+-------------+-----------------+\n");

printf("| ID | %15s | %15s | %4s | %6s | %4s | %11s | %15s |\n","Producer","Model","Volume","Slots","PCI","Sound type","Memory types");

printf("+----+-----------------+-----------------+--------+--------+------+-------------+-----------------+\n");

}

/\*output catalogue\*/

void output(motherboards \*board)

{

motherboards \*tmp=NULL;

tmp = board;

if (tmp==NULL) puts("\n List is Empty");

else

{

while(tmp!=NULL)

{

printf("| %3d| %16s| %16s| %7d| %7d| %5.1f| %12.1f| %4d| %4d| %4d|\n",tmp->id,tmp->producer->name,tmp->model,tmp->volume,

tmp->slots,tmp->pci,tmp->soundtype,tmp->memorytypes[0],tmp->memorytypes[1],tmp->memorytypes[2]);

tmp = tmp->next;

}

printf("+----+-----------------+-----------------+--------+--------+------+-------------+-----------------+\n");

}

free(tmp);

tmp=NULL;

}

/\*Check for correct input from keyboard\*/

int input\_check(char line[MAXLEN])

{

int i, count;

count = 0;

for (i=0;i<strlen(line);i++)

{

if (line[i] == ';') count++;

}

if (count != 8) return 0;

else return 1;

}

/\*delete node by id from catalogue\*/

void deleteNode(motherboards\*\* board, int key)

{

motherboards \*tmp, \*prev;

tmp = \*board;

if (tmp != NULL && tmp->id == key)

{

\*board = tmp->next;

free(tmp);

return;

}

while (tmp != NULL && tmp->id != key)

{

prev = tmp;

tmp = tmp->next;

}

if (tmp == NULL) return;

prev->next = tmp->next;

free(tmp);

}

/\*delete node from list of model\*/

void deleteProducer(Producers\*\* producer, Producers\* del)

{

/\* base case \*/

if (\*producer == NULL || del == NULL)

return;

/\* If node to be deleted is head node \*/

if (\*producer == del)

\*producer = del->next;

/\* Change next only if node to be deleted is NOT the last node \*/

if (del->next != NULL)

del->next->prev = del->prev;

/\* Change prev only if node to be deleted is NOT the first node \*/

if (del->prev != NULL)

del->prev->next = del->next;

/\* Finally, free the memory occupied by element to delete\*/

free(del);

return;

}

/\*edit any node in catalogue by any parameter\*/

motherboards \*edit(motherboards \*current, Producers \*producer, int status)

{

int input;

char s[MAXLEN];

switch (status)

{

case 1: /\*Editing producer\*/

producer\_id(producer);

clear

output\_producer(producer);

printf("Enter the number of an existing element or enter 0 to add new:\nChoice: ");

scanf("%d", &input);

if (input==0)

{

scanf("%s",s);

current->producer = get\_producer(&producer, s);

}

else

{

current->producer = get\_producer(&producer, find\_producer(producer, input)->name);

}

break;

case 2: /\*Editing model\*/

scanf("%s",s);

strcpy(current->model, s);

break;

case 3: /\*Editing volume\*/

scanf("%s",s);

current->volume = atoi(s);

break;

case 4: /\*Editing slots\*/

scanf("%s",s);

current->slots = atoi(s);

break;

case 5: /\*Editing pci\*/

scanf("%s",s);

current->pci = atof(s);

break;

case 6: /\*Editing sound type\*/

scanf("%s",s);

current->soundtype = atof(s);

break;

case 7:/\*Editing memory types\*/

scanf("%d",&input);

printf("Print new memory type:\n");

scanf("%s",s);

current->memorytypes[input-1] = atoi(s);

break;

default:

puts("INCORRECT");

break;

}

clear

return current;

}

/\*Producer id count\*/

void producer\_id(Producers\* producer)

{

int count;

count=0;

while (producer!=NULL)

{

producer->id=++count;

producer=producer->next;

}

}

/\*Output producer to edit\*/

void output\_producer(Producers\* producer)

{

Producers\* tmp; tmp=producer;

if (producer==NULL)

{

puts("List is empty");

return;

}

while (tmp!=NULL)

{

printf("%d %s\n", tmp->id, tmp->name);

tmp=tmp->next;

}

}

/\*Update producer list\*/

Producers\* update\_producer(motherboards\* board, Producers\* producer)

{

motherboards\* tmp; Producers\* tmp\_producer;

int count;

tmp\_producer=producer;

while (tmp\_producer!=NULL)

{

tmp=board;

count =0;

while (tmp!=NULL)

{

if (strcmp(tmp\_producer->name, tmp->producer->name)==0)

{

count=1;

tmp=tmp->next;

break;

}

tmp=tmp->next;

}

if (count==0)

{

deleteProducer(&producer, tmp\_producer);

return producer;

}

tmp\_producer=tmp\_producer->next;

}

return producer;

}

/\*search nodes by any parameter\*/

void search(motherboards \*board, Producers\* producer, int search\_status)

{

motherboards \*current;

char s[MAXLEN];

int menu\_status, count, s1;

float s2;

s1 = 0;

s2 = 0;

count = 0;

current = board;

switch (search\_status)

{

case 1:

output\_producer(producer);

printf("Enter the number of an existing element: ");

break;

case 2:

printf("Enter searching model (implemented substring search): ");

break;

case 3:

printf("Enter searching volume: ");

break;

case 4:

printf("Enter searching slots: ");

break;

case 5:

printf("Enter searching PCI: ");

break;

case 6:

printf("Enter searching sound type: ");

break;

case 7:

printf("Enter searching memory type: ");

break;

default:

break;

}

scanf("%s", s);

clear

s1 = atoi(s);

s2 = atof(s);

header();

while (current != NULL) /\*search nodes by any parameter\*/

{

if ((strcmp(current->producer->name, find\_producer(producer, atoi(s))->name) == 0 && search\_status == 1) ||

(strstr(current->model,s) != NULL && search\_status == 2) ||

(s1 == current->volume && search\_status == 3) ||

(s1 == current->slots && search\_status == 4) ||

(s2 == current->pci && search\_status == 5) ||

(s2 == current->soundtype && search\_status == 6) ||

(s1 == current->memorytypes[0] && search\_status == 7)||

(s1 == current->memorytypes[1] && search\_status == 7)||

(s1 == current->memorytypes[2] && search\_status == 7))

{

count++;

printf("| %3d| %16s| %16s| %7d| %7d| %5.1f| %12.1f| %4d| %4d| %4d|\n",current->id,current->producer->name,current->model,current->volume,

current->slots,current->pci,current->soundtype,current->memorytypes[0],current->memorytypes[1],current->memorytypes[2]);

}

current = current->next;

}

if (count == 0)

{

clear

printf("No such elements in catalogue!\n"); /\*Error if no elements of search exist\*/

}

printf("\nPrint 0 to return menu\n");

scanf("%d", &menu\_status);

while (menu\_status!=0)

{

clear

printf("\nError! You must print 0 to return menu\n"); /\*Error input\*/

scanf("%d", &menu\_status);

}

}

/\*comparator to to find out if a given word is a producer's name of current node\*/

Producers\* find\_producer(Producers\* producer, int id)

{

Producers\* tmp, \*tp1;

tmp=producer;

while (tmp!=NULL)

{

if (tmp->id==id) tp1=tmp;

tmp=tmp->next;

}

return tp1;

}

/\*sort increasing/decreasing catalogue by any parameter \*/

motherboards \*sort(motherboards \*root, bool (\*comp)(motherboards\*, motherboards\*, int), int n)

{

motherboards \*current,\*new\_root = NULL, \*node;

while ( root != NULL )

{

node = root;

root = root->next;

if ( new\_root == NULL || comp(node, new\_root, n))

{

node->next = new\_root;

new\_root = node;

}

else

{

current = new\_root;

while ( current->next != NULL && !comp(node, current->next, n))

{

current = current->next;

}

node->next = current->next;

current->next = node;

}

}

return new\_root;

}

/\*comparator to sort catalogue by id \*/

bool sort\_id(motherboards\* a, motherboards\* b, int n)

{

if (n==1) return a->id <= b->id;

else return a->id >= b->id;

}

/\*comparator to sort catalogue by producer \*/

bool sort\_producer(motherboards\* a, motherboards\* b, int n)

{

if (n==1) return strcmp(a->producer->name, b->producer->name) <= 0 ? 1 : 0;

else return strcmp(a->producer->name, b->producer->name) >= 0 ? 1 : 0;

}

/\*comparator to sort catalogue by model \*/

bool sort\_model(motherboards\* a, motherboards\* b, int n)

{

if (n==1) return strcmp(a->model, b->model) <= 0 ? 1 : 0;

else return strcmp(a->model, b->model) >= 0 ? 1 : 0;

}

/\*comparator to sort catalogue by volume \*/

bool sort\_volume(motherboards\* a, motherboards\* b, int n)

{

if (n==1) return a->volume <= b->volume;

else return a->volume >= b->volume;

}

/\*comparator to sort catalogue by slots \*/

bool sort\_slots(motherboards\* a, motherboards\* b, int n)

{

if (n==1) return a->slots <= b->slots;

else return a->slots >= b->slots;

}

/\*comparator to sort catalogue by pci \*/

bool sort\_pci(motherboards\* a, motherboards\* b, int n)

{

if (n==1) return a->pci <= b->pci;

else return a->pci >= b->pci;

}

/\*comparator to sort catalogue by sound type \*/

bool sort\_soundtype(motherboards\* a, motherboards\* b, int n)

{

if (n==1) return a->soundtype <= b->soundtype;

else return a->soundtype >= b->soundtype;

}

/\*comparator to sort catalogue by memory type 1 \*/

bool sort\_types0(motherboards\* a, motherboards\* b, int n)

{

if (n==1) return a->memorytypes[0] <= b->memorytypes[0];

else return a->memorytypes[0] >= b->memorytypes[0];

}

/\*comparator to sort catalogue by memory type 2 \*/

bool sort\_types1(motherboards\* a, motherboards\* b, int n)

{

if (n==1) return a->memorytypes[1] <= b->memorytypes[1];

else return a->memorytypes[1] >= b->memorytypes[1];

}

/\*comparator to sort catalogue by memory type 3 \*/

bool sort\_types2(motherboards\* a, motherboards\* b, int n)

{

if (n==1) return a->memorytypes[2] <= b->memorytypes[2];

else return a->memorytypes[2] >= b->memorytypes[2];

}

/\*splitting the input string by delimiter character\*/

char \*\*simple\_split(char \*line, const char sep)

{

int sep\_count, i, k, m;

char \*\*result\_array = NULL;

for (i = 0, sep\_count = 0; i < strlen(line); i++)

if (line[i] == sep) sep\_count++;

result\_array = (char \*\*) malloc((sep\_count + 1) \* sizeof(char \*));

if(!result\_array)

{

printf("MEMORY ERROR\n");

exit(0);

}

for (i = 0; i < sep\_count + 1; i++)

{

if (result\_array != NULL)

{

result\_array[i] = (char \*) malloc(strlen(line) \* sizeof(char));

if(!result\_array[i])

{

printf("MEMORY ERROR\n");

exit(0);

}

if (result\_array[i] == NULL) result\_array = NULL;

}

else i = sep\_count + 1;

}

if (result\_array != NULL)

{

k = m = 0;

for (i = 0; i < strlen(line); i++)

{

if (line[i] != sep) result\_array[m][i - k] = line[i];

else

{

result\_array[m][i - k] = '\0';

k = i + 1;

m++;

}

}

}

return result\_array;

}