

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра вычислительной техники

Отчет по лабораторной работе № 9
по дисциплине «Программирование»
Тема: указатели на структуры и функции

Студент гр. 9305

Есин А.Ю

Преподаватель

Перязева Ю.В

Санкт-Петербург

2019

Содержание

Цель	3
Задание.....	3
Постановка задачи и описание решения	3
Описание структуры.....	4
Структура вызова функций	5
Текст программы	27
Заключение	68

Цель

Получить практические навыки в разработке алгоритмов с использованием указателей на структуры и функции на языке Си.

Задание

Для выбранной предметной области создать динамический массив структур, содержащих характеристики объектов предметной области.

Обязательный набор полей:

- динамический массив символов, включая пробелы (name)
- произвольный динамический массив символов
- числовые поля типов `int` и `float` (не менее двух полей каждого типа)
- поле с числовым массивом.

Написать программу, обеспечивающую начальное формирование массива структур при чтении из файла (текст с разделителями — CSV) с последующим возможным дополнением элементов массива при вводе с клавиатуры. Создать функции сортировки\выбора: выбор записей по значению последнего слова поля name, сортировка результата по убыванию значений любого из числовых полей (выбор поля для сортировки — из меню). Во всех случаях, когда при поиске записей результат отсутствует, следует вывести сообщение.

Постановка задачи и описание решения

Дано: CSV файл, содержащий структуры.

Требуется получить: структуры, удовлетворяющие требованиям сортировки.

В качестве предметной области были выбраны гитары (различные их характеристики позволяют задействовать все необходимые типы полей).

Сначала считываем данные (строка/строки, из которых формируются структуры) из файла. Затем возможно ввести доп.данные с клавиатуры, вывести все структуры на экран, а также продолжить и перейти к сортировке.

Выбор записей по значению последнего слова поля name заключается в следующем: пользователь вводит слово, оно сравнивается с последним

словом, перед которым был пробел (если такового нет, то с первым и единственным словом), не учитывая регистр, и если эти слова совпадают, то структура, в которой было данное имя, выводится на экран.

Сортировка по числовым полям осуществляется через метафункцию следующим образом: вызывается функция сортировки выбранного пользователем поля, в которой значения выбранного поля каждого нового элемента массива структур, начиная с первого, сравниваются со всеми предыдущими, и если предыдущее значение больше, то они меняются местами. После этого цикл повторяется для следующего элемента массива структур. Когда сортировка закончена, отсортированный массив структур выводится на экран.

Описание структуры

Таблица 1. Описание структуры (typedef struct unit {...} guitar).

Имя поля	Тип	Назначение
name	char*	Указатель на массив символов с именем элемента
info	char*	Указатель на строку с информацией об элементе (произвольный массив символов)
numOfPickups	int	Количество звукоснимателей
numOfFrets	int	Количество ладов
numOfStrings	int	Количество струн
menzureLength	float	Длина мензуры
neckRadius	float	Радиус грифа
stringsWidth	int*	Указатель на массив целых чисел, хранящий толщину струн

Структура вызова функций

1.Main

Описание:

Является точкой входа в программу

Прототип:

```
Int main();
```

Пример вызова:

```
main()
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Локальная	table	guitar**	Указатель на указатель на массив структур типа guitar
Локальная	flag	int	Выбор пункта меню
Локальная	csvReadFlag	int	Флаг, указывающий был ли файл открыт
Локальная	i	int	Счетчик цикла
Локальная	trash	int	Переменная для сбора мусора при ошибках считывания выбора пункта меню
Локальная	N	int	Количество элементов массива структур
Локальная	sortCount	int	Сборщик мусора
Локальная	search	char*	Указатель на строку для выбора по последнему слову поля name

Возвращаемое значение:

2.MainMenu

Описание:

Вывод главного меню

Прототип:

```
Void MainMenu();
```

Пример вызова:

```
MainMenu();
```

Возвращаемое значение:

3.fileInput

Описание:

Считывание данных из файла

Прототип:

```
guitar **fileInput(struct unit **table, int *N);
```

Пример вызова:

```
table = fileInput(table, &N);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	table	guitar**	Указатель на указатель на массив структур типа guitar
Формальный аргумент	N	int*	Количество элементов массива структур
Локальная	df	FILE*	Указатель на файл с именем filename
Локальная	filename	char*	Массив символов длиной nameLength

Локальная	length	int	Фактическая длина filename
Локальная	i	int	Вспомогательный флаг для уборки мусора
Локальная	size	int	Количество строк (элементов массива структур) в файле

Возвращаемое значение: возвращает указатель на указатель на массив структур типа guitar.

4.AddMenu()

Описание:

Вывод меню добавления элементов

Прототип:

Void MainMenu();

Пример вызова:

AddMenu();

Возвращаемое значение:

5.keyboardInput

Описание:

Ввод данных с клавиатуры

Прототип:

void keyboardInput(struct unit ***table,int *N);

Пример вызова:

keyboardInput(&table,&N);

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	table	guitar***	Тройной указатель на массив структур типа guitar
Формальный аргумент	N	int*	Количество элементов массива структур
Локальная	data	char*	Указатель на строку в которую сохраняются введенные данные

Возвращаемое значение:

6.mainSortMenu();

Описание:

Вывод главного меню сортировки

Прототип:

Void mainSortMenu();

Пример вызова:

mainSortMenu();

Возвращаемое значение:

7.strRead

Описание:

Считывание данных из *df и их запись в строку dest

Прототип:

void strRead(FILE *df, char **dest);

Пример вызова:

strRead(df, &str);

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	df	FILE*	Указатель на файл
Формальный аргумент	dest	char**	Указатель на строку, в которую сохраняются введенные данные
Локальная	symbol	char	Считываемый в данный момент символ
Локальная	multiplier	int	Множитель, используемый для расширения строки
Локальная	length	int	Длина строки
Локальная	errCount	int	Счетчик «мусорного» ввода

Возвращаемое значение:

8.nameSort

Описание:

Сортировка по последнему слову поля name

Прототип:

```
void nameSort(struct unit **table,int N,char *search);
```

Пример вызова:

```
nameSort(table, N, search);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	table	guitar**	Двойной указатель на массив структур типа guitat
Формальный аргумент	N	int	Количество элементов массива структур
Формальный аргумент	search	char*	Слово-запрос, введенное пользователем
Локальная	i	int	Счетчик цикла
Локальная	j	int	Счетчик цикла
Локальная	len	int	Длина имени элемента массива структуры
Локальная	startpoint	int	Начало последнего слова в поле name
Локальная	printed	int	Счетчик количества подходящих элементов массива структур
Локальная	scanFlag	int	Флаг необходимости приведения запроса пользователя в верхний регистр
Локальная	temp	char*	Строка в которую копируется последнее слово поля name

Возвращаемое значение:

9.numberSortMenu();

Описание:

Вывод меню сортировки числовых полей

Прототип:

Void numberSortMenu();

Пример вызова:

numberSortMenu();

Возвращаемое значение:

10.numberSearch

Описание:

Метафункция сортировки числовых полей

Прототип:

void numberSearch(struct unit **table, int N, void (*funcName)(guitar**, int));

Пример вызова:

numberSearch(table,N,pickupsSort);

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	table	guitar**	Двойной указатель на массив структур типа guitar
Формальный аргумент	N	int	Количество элементов массива структур
Формальный аргумент	funcName	void*	Указатель на функцию funcName

Возвращаемое значение:

11.PrintAll

Описание:

Вывод всех элементов массива структур на экран

Прототип:

```
void printAll(struct unit **table, int N);
```

Пример вызова:

```
printAll(table,N);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	table	guitar**	Двойной указатель на массив структур типа guitar
Формальный аргумент	N	int	Количество элементов массива структур

Возвращаемое значение:

12.InfoMenu

Описание:

Вывод меню помощи.

Прототип:

```
Void InfoMenu();
```

Пример вызова:

```
InfoMenu();
```

Возвращаемое значение:

13.StructPrint

Описание:

Вывод определенной строки структуры.

Прототип:

```
void structPrint(struct unit *table,int current);
```

Пример вызова:

```
structPrint(table[i],i);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	table	guitar*	Указатель на элемент массива структур типа guitar
Локальная	j	int	Счетчик цикла

Возвращаемое значение:

14.csvCount

Описание:

Подсчитывает количество строк в файле (кол-во элементов массива структур).

Прототип:

```
int csvCount(FILE *df);
```

Пример вызова:

```
size = csvCount(df);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	df	FILE*	Указатель на файл
Локальная	count	int	Счетчик количества строк в файле

Локальная	symbol	char	Считываемый в данный момент символ
-----------	--------	------	------------------------------------

Возвращаемое значение: количество строк в файле

15.unitFill

Описание:

Заполняет один элемент (строку) массива структур.

Прототип:

```
void unitFill(FILE *df, struct unit **table,int i);
```

Пример вызова:

```
unitFill(df,table,i);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	df	FILE*	Указатель на файл
Формальный аргумент	table	guitar**	Двойной указатель на массив структур типа guitar.
Формальный аргумент	i	int	Номер заполняемой строки массива структур
Локальная	str	char*	Указатель на строку, в которую считываются данные из файла
Локальная	j	int	Счетчик цикла
Локальная	endFlag	int	Флаг итераций считывания

Возвращаемое значение:

16.StringDelete

Описание:

Удаление из строки первых count символов.

Прототип:

```
void StringDelete(char *s,int count);
```

Пример вызова:

```
StringDelete(temp, prevCount);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	s	char*	Указатель на удаляемую строку
Формальный аргумент	count	int	Количество удаляемых символов
Локальная	k	int	Счетчик цикла
Локальная	j	int	Счетчик цикла

Возвращаемое значение:

17.clearUnit

Описание:

Очистка строки массива структур.

Прототип:

```
void clearUnit(struct unit *table);
```

Пример вызова:

```
clearUnit(table[i]);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	table	guitar*	Указатель на элемент массива структур

Возвращаемое значение:

18.nameInp

Описание:

Ввод поля name в структуру под номером count в массиве структур.

Прототип:

```
void nameInp(int count, char **source, struct unit *table);
```

Пример вызова:

```
nameInp(j,&str,table[i]);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	count	int	Номер элемента в массиве структур
Формальный аргумент	source	char**	Указатель на строку, из которой берутся данные для ввода
Формальный аргумент	table	guitar*	Указатель на элемент массива структур
Локальная	temp	char*	Временная строка в которую копируется name из исходной строки

Возвращаемое значение:

19.infoInp

Описание:

Ввод поля info в структуру под номером count в массиве структур.

Прототип:

```
void infoInp(int count, char **source, struct unit *table);
```

Пример вызова:

```
infoInp(j,&str,table[i]);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	count	int	Номер элемента в массиве структур
Формальный аргумент	source	char**	Указатель на строку, из которой берутся данные для ввода
Формальный аргумент	table	guitar*	Указатель на элемент массива структур
Локальная	temp	char*	Временная строка в которую копируется name из исходной строки
Локальная	prevCount	int	Длина уже введенного фрагмента в исходной строке

Возвращаемое значение:

20.pickupsInp

Описание:

Ввод поля numOfPickups в структуру под номером count в массиве структур.

Прототип:

```
void pickupsInp(int count, char **source, struct unit *table);
```

Пример вызова:

```
pickupsInp(j,&str,table[i]);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	count	int	Номер элемента в массиве структур
Формальный аргумент	source	char**	Указатель на строку, из которой берутся данные для ввода
Формальный аргумент	table	guitar*	Указатель на элемент массива структур
Локальная	tempStr	char*	Временная строка в которую копируется name из исходной строки
Локальная	prevCount	int	Длина уже введенного фрагмента в исходной строке
Локальная	tempInt	int	Вспомогательная переменная для подсчета длины считанной части исходной строки

Возвращаемое значение:

21.fretsInp

Описание:

Ввод поля numOfFrets в структуру под номером count в массиве структур.

Прототип:

```
void fretsInp(int count, char **source, struct unit *table);
```

Пример вызова:

```
fretsInp(j,&str,table[i]);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	count	int	Номер элемента в массиве структур
Формальный аргумент	source	char**	Указатель на строку, из которой берутся данные для ввода
Формальный аргумент	table	guitar*	Указатель на элемент массива структур
Локальная	tempStr	char*	Временная строка в которую копируется name из исходной строки
Локальная	prevCount	int	Длина уже введенного фрагмента в исходной строке
Локальная	tempInt	int	Вспомогательная переменная для подсчета длины считанной части исходной строки
Локальная	pickupsLen	int	Длина области исходной строки с numOfPickups

Возвращаемое значение:

22.numStringsInp

Описание:

Ввод поля numOfStrings в структуру под номером count в массиве структур.

Прототип:

```
void numStringsInp(int count, char **source, struct unit *table);
```

Пример вызова:

```
numStringsInp(j,&str,table[i]);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	count	int	Номер элемента в массиве структур
Формальный аргумент	source	char**	Указатель на строку, из которой берутся данные для ввода
Формальный аргумент	table	guitar*	Указатель на элемент массива структур
Локальная	tempStr	char*	Временная строка в которую копируется name из исходной строки
Локальная	prevCount	int	Длина уже введенного фрагмента в исходной строке
Локальная	tempInt	int	Вспомогательная переменная для подсчета длины считанной части исходной строки
Локальная	intFieldsLength	int	Длина области исходной строки, содержащей целочисленные значения

Возвращаемое значение:

23.menzureInp

Описание:

Ввод поля menzureLength в структуру под номером count в массиве структур.

Прототип:

```
void menzureInp(int count, char **source, struct unit *table);
```

Пример вызова:

```
menzureInp(j,&str,table[i]);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	count	int	Номер элемента в массиве структур
Формальный аргумент	source	char**	Указатель на строку, из которой берутся данные для ввода
Формальный аргумент	table	guitar*	Указатель на элемент массива структур
Локальная	tempStr	char*	Временная строка в которую копируется name из исходной строки
Локальная	prevCount	int	Длина уже введенного фрагмента в исходной строке
Локальная	tempInt	int	Вспомогательная переменная для подсчета длины считанной части исходной строки
Локальная	intFieldsLength	int	Длина области исходной строки, содержащей целочисленные значения

Возвращаемое значение:

24.radiusInp

Описание:

Ввод поля neckRadius в структуру под номером count в массиве структур.

Прототип:

```
void radiusInp(int count, char **source, struct unit *table);
```

Пример вызова:

```
radiusInp(j,&str,table[i]);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	count	int	Номер элемента в массиве структур
Формальный аргумент	source	char**	Указатель на строку, из которой берутся данные для ввода
Формальный аргумент	table	guitar*	Указатель на элемент массива структур
Локальная	tempStr	char*	Временная строка в которую копируется name из исходной строки
Локальная	temp2	char*	Вспомогательная строка
Локальная	tempFloat	float	Временная переменная
Локальная	prevCount	int	Длина уже введенного фрагмента в исходной строке
Локальная	tempInt	int	Вспомогательная переменная для подсчета длины

			считанной части исходной строки
Локальная	numberFieldsLength	int	Длина области исходной строки, содержащей числовые значения
Локальная	i	int	Счетчик цикла

Возвращаемое значение:

25.stringsWidthInp

Описание:

Ввод массива stringsWidth в структуру под номером count в массиве структур.

Прототип:

```
void stringsWidthInp(int count, char **source, struct unit *table);
```

Пример вызова:

```
stringsWidthInp(j,&str,table[i]);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	count	int	Номер элемента в массиве структур
Формальный аргумент	source	char**	Указатель на строку, из которой берутся данные для ввода
Формальный аргумент	table	guitar*	Указатель на элемент массива структур
Локальная	tempStr	char*	Временная строка в которую копируется name из исходной строки

Локальная	i	int	Счетчик цикла
Локальная	iteration	int	Номер вводимого элемента массива stringsWidth
Локальная	temp	int	Временная переменная

Возвращаемое значение:

26.keyboardUnitFill

Описание:

Заполнение полей структуры из строки, полученной с клавиатуры

Прототип:

```
void keyboardUnitFill(struct unit *table, char *str);
```

Пример вызова:

```
keyboardUnitFill((*table)[(*N)-1],data);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	str	char*	Строка с входными данными
Формальный аргумент	table	guitar*	Указатель на элемент массива структур
Локальная	j	int	Счетчик цикла
Локальная	endFlag	int	Флаг итераций считывания

Возвращаемое значение:

27.scanField

Описание:

Считывание данных и их соединение в одну строку про вводе с клавиатуры.

Прототип:

```
void scanField(char **dest,char **source,int *mult,int mod);
```

Пример вызова:

```
scanField(&inp,&temp,&mult,0);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	dest	char**	Строка со всеми данными нового элемента массива структур
Формальный аргумент	source	char**	Строка в которую считывается конкретное поле нового элемента массива
Формальный аргумент	mult	int*	Множитель, используемый для расширения строки
Формальный аргумент	mod	int	Флаг добавления ';' после добавления dest в source
Локальная	len	int	Длина строки
Локальная	split	char*	Строка с символом разделителем

Возвращаемое значение:

28.DataScan

Описание:

Основная функция для ввода данных с клавиатуры.

Прототип:

```
char *DataScan();
```

Пример вызова:

```
data = DataScan();
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Локальная	inp	char*	Строка со всеми данными нового элемента массива структур
Локальная	temp	char*	Строка в которую считывается конкретное поле нового элемента массива
Локальная	split	char*	Множитель, используемый для расширения строки
Локальная	multiplier	int	Множитель, используемый для расширения строки (для «внешней» функции)
Локальная	numOfStrings	int	Количество струн
Локальная	mult	int	Множитель, используемый для расширения строки (для «внутренней» функции)

Возвращаемое значение: строка с введенными данными.

29.pickupsSort

Описание:

Сортировка элементов массива структур по убыванию поля numOfPickups.

Прототип:

```
void pickupsSort(struct unit **table,int N);
```

Пример вызова:

```
numberSearch(table,N,pickupsSort);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Локальная	i	int	Счетчик цикла
Локальная	j	int	Счетчик цикла
Локальная	tempStruct	guitar*	Временная структура, используемая в сортировке
Формальный аргумент	table	unit**	Указатель на массив структур
Формальный аргумент	N	int	Количество элементов массива структур

Возвращаемое значение:

Функции fretsSort, numberOfStringsSort, menzureSort, radiusSort и stringWidthSort аналогичны функции pickupsSort (за исключением того, что сортировка производится по разным полям).

Текст программы

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>
```

```
#include <ctype.h>
```

```
typedef struct unit {  
    char *name;  
    char *info;  
    int numOfPickups;  
    int numOfFrets;  
    int numOfString;  
    float menzureLength;  
    float neckRadius;  
    int *stringsWidth;  
} guitar;
```

```
enum values {  
    nameLength = 35,  
    inputLength = 31,  
    readStep = 15,  
    keyboardReadStep = 70,  
    splitSymbolCompensation = 1,  
    sep = ','  
};
```

```
void MainMenu(); //вывод главного меню @@
```

```
void AddMenu(); //вывод меню добавления элемента @@
```

```
void InfoMenu(); //вывод меню помощи @@
```

```
void numberSortMenu(); //вывод меню сортировки числовых полей @@
```

```
void mainSortMenu(); //вывод главного меню сортировки @@
```

```
void printAll(struct unit **table, int N); //вывод всех строк структуры @@
```

void structPrint(struct unit *table); //вывод определенной строки структуры
@@

guitar **fileInput(struct unit **table, int *N); //организовывает ввод данных
из файла @@

int csvCount(FILE *df); //подсчитывает количество строк в файле (кол-во
элементов массива структур) @@

void unitFill(FILE *df, struct unit **table,int i); //Заполняет один элемент
(строку) массива структур @@

void strRead(FILE *df, char **dest); //считывает одну строку из файла @@

void StringDelete(char *s,int count); //удаление из строки первых count
символов @@

void clearUnit(struct unit *table); //очистка строки структуры @@

void nameInp(int count, char **source, struct unit *table); //вводы различных
полей структуры из строки в массив структур @@

void infoInp(int count, char **source, struct unit *table);

void pickupsInp(int count, char **source, struct unit *table);

void fretsInp(int count, char **source, struct unit *table);

void numStringsInp(int count, char **source, struct unit *table);

void menzureInp(int count, char **source, struct unit *table);

void radiusInp(int count, char **source, struct unit *table);

void stringsWidthInp(int count, char **source, struct unit *table);

void keyboardInput(struct unit ***table,int *N); //формирование строки
входных данных с клавиатуры @@

void keyboardUnitFill(struct unit *table, char *str); //заполнение полей
структуры из строки, полученной с клавиатуры @@

void scanField(char **dest,char **source,int *mult,int mod); //соединение
данных полей в строку (ввод с клавиатуры) @@

```
char *DataScan(); //основная функция для ввода данных с клавиатуры @@
```

```
void nameSort(struct unit **table,int N,char *search); //сортировка по  
последнему слову поля name @@
```

```
void numberSearch(struct unit **table, int N, void (*funcName)(guitar**,  
int)); //метафункция сортировки числовых полей @@
```

```
void pickupsSort(struct unit **table,int N); //сортировка по numOfPickups  
@@
```

```
void fretsSort(struct unit **table, int N); //сортировка по numOfFrets
```

```
void numberOfStringsSort(struct unit **table, int N); //сортировка по  
numOfStrings
```

```
void menzureSort(struct unit **table, int N); //сортировка по menzureLength
```

```
void radiusSort(struct unit **table, int N); //сортировка по neckRadius
```

```
void stringWidthSort(struct unit **table, int N); //сортировка по stringsWidth
```

```
void clearUnit(struct unit *table)
```

```
{  
    free(table->name);  
    free(table->info);  
    free(table->stringsWidth);  
    table->name = NULL;  
    table->info = NULL;  
    table->stringsWidth = NULL;  
    free(table);  
    table = NULL;  
}
```

```
void StringDelete(char *s,int count)
```

```
{
```

```

int j,k;
if (strlen(s) != 1) {
    for (j = 0; j <= count; j++) {
        for (k = 0; k < strlen(s) - 1; k++) {
            s[k] = s[k + 1];
        }
        s[k] = '\0';
    }
} else s[0] = '\0';
}

```

```

void nameInp(int count, char **source, struct unit *table)
{
    char *temp = NULL;

    temp = (char*)malloc(sizeof(char) * (count + 1));
    if (temp != NULL) {
        strncpy(temp,*source,count);
        temp[count] = '\0';
        table->name = (char*)malloc(sizeof(char) * (count + 1));
        if (table->name != NULL) {
            strcpy(table->name, temp);
        } else {
            perror("Can't copy a string, error at memory allocation!");
            exit(3);
        }
        free(temp);
        temp = NULL;
    }
}

```

```

    } else {
        perror("Ooops, seems like something's wrong with your memory!");
        exit(5);
    }

}

```

```

void infoInp(int count, char **source, struct unit *table)
{
    char *temp = NULL;
    int prevCount;

    prevCount = strlen(table->name);
    temp = (char*)malloc(sizeof(char) * (count + 1));
    if (temp != NULL) {
        strncpy(temp, *source, count);
        temp[count] = '\0';
        StringDelete(temp, prevCount);
        table->info = (char *) malloc(sizeof(char) * (strlen(temp) + 1));
        if (table->info != NULL) {
            strcpy(table->info, temp);
        } else {
            perror("Can't copy a string, error at memory allocation!");
            exit(3);
        }
    } else {
        perror("Ooops, seems like something's wrong with your memory!");
        exit(4);
    }
}

```



```

    }
    free(temp);
    temp = NULL;
}

```

```

void pickupsInp(int count, char **source, struct unit *table)
{
    int prevCount,tempInt;
    char *tempStr = NULL;

    prevCount    =    strlen(table->name)    +    strlen(table->info)    +
splitSymbolCompensation;
    tempStr = (char*)malloc(sizeof(char) * (count + 1));
    if (tempStr != NULL) {
        strncpy(tempStr,*source,count);
        tempStr[count] = '\0';
        StringDelete(tempStr,prevCount);
        tempInt = atoi(tempStr);
        table->numOfPickups = tempInt;
    } else {
        perror("Ooops, seems like something's wrong with your memory!");
        exit(6);
    }
    free(tempStr);
    tempStr = NULL;
}

```

```

void fretsInp(int count, char **source, struct unit *table)

```

```

{
    int prevCount,tempInt,pickupsLen = 0;
    char *tempStr = NULL;

    tempStr = (char*)malloc(sizeof(char) * (count + 1));
    if (tempStr != NULL) {
        strncpy(tempStr,*source,count);
        tempStr[count] = '\0';

        tempInt = table->numOfPickups;
        while (tempInt != 0) {
            pickupsLen++;
            tempInt /= 10;
        }

        prevCount = strlen(table->name) + strlen(table->info) + pickupsLen +
2*splitSymbolCompensation;
        StringDelete(tempStr,prevCount);
        tempInt = atoi(tempStr);
        table->numOfFrets = tempInt;
    } else {
        perror("Ooops, seems like something's wrong with your memory!");
        exit(7);
    }
    free(tempStr);
    tempStr = NULL;
}

```

```

void numStringsInp(int count, char **source, struct unit *table)
{
    int prevCount,tempInt,intFieldsLength = 0;
    char *tempStr = NULL;

    tempStr = (char*)malloc(sizeof(char) * (count + 1));
    if (tempStr != NULL) {
        strncpy(tempStr,*source,count);
        tempStr[count] = '\0';

        tempInt = table->numOfPickups;
        while (tempInt != 0) {
            intFieldsLength++;
            tempInt /= 10;
        }
        tempInt = table->numOfFrets;
        while (tempInt != 0) {
            intFieldsLength++;
            tempInt /= 10;
        }

        prevCount = strlen(table->name) + strlen(table->info) + intFieldsLength
+ 3*splitSymbolCompensation;
        StringDelete(tempStr,prevCount);
        tempInt = atoi(tempStr);
        table->numOfString = tempInt;
    } else {
        perror("Ooops, seems like something's wrong with your memory!");
    }
}

```

```

        exit(8);
    }
    free(tempStr);
    tempStr = NULL;
}

```

```

void menzureInp(int count, char **source, struct unit *table)

```

```

{
    int prevCount,tempInt,intFieldsLength = 0;
    float tempFloat;
    char *tempStr = NULL;

    tempStr = (char*)malloc(sizeof(char) * (count + 1));
    if (tempStr != NULL) {
        strncpy(tempStr,*source,count);
        tempStr[count] = '\0';

        tempInt = table->numOfPickups;
        while (tempInt != 0) {
            intFieldsLength++;
            tempInt /= 10;
        }
        tempInt = table->numOfFrets;
        while (tempInt != 0) {
            intFieldsLength++;
            tempInt /= 10;
        }
        tempInt = table->numOfString;
    }
}

```

```

while (tempInt != 0){
    intFieldsLength++;
    tempInt /= 10;
}

```

```

    prevCount = strlen(table->name) + strlen(table->info) + intFieldsLength
+ 4*splitSymbolCompensation;
    StringDelete(tempStr,prevCount);
    tempFloat = atof(tempStr);
    table->menzureLength = tempFloat;
} else {
    perror("Ooops, seems like something's wrong with your memory!");
    exit(9);
}
free(tempStr);
tempStr = NULL;
}

```

```

void radiusInp(int count, char **source, struct unit *table)
{
    int prevCount,tempInt,i,numberFieldsLength = 0;
    float tempFloat;
    char *tempStr = NULL,*temp2 = NULL;

    tempStr = (char*)malloc(sizeof(char) * (count + 1));
    if (tempStr != NULL) {
        strncpy(tempStr,*source,count);
        tempStr[count] = '\0';
    }
}

```

```

tempInt = table->numOfPickups;
while (tempInt != 0) {
    numberFieldsLength++;
    tempInt /= 10;
}
tempInt = table->numOfFrets;
while (tempInt != 0) {
    numberFieldsLength++;
    tempInt /= 10;
}
tempInt = table->numOfString;
while (tempInt != 0) {
    numberFieldsLength++;
    tempInt /= 10;
}
prevCount    =    strlen(table->name)    +    strlen(table->info)    +
numberFieldsLength + 4 * splitSymbolCompensation;
temp2 = (char*)malloc(sizeof(char) * (count + 1));
if (temp2 != NULL) {
    strncpy(temp2, *source, count);
    temp2[count] = '\0';
    StringDelete(temp2, prevCount);

    for (i = 0; temp2[i] != sep; i++, numberFieldsLength++);

    prevCount    =    strlen(table->name)    +    strlen(table->info)    +
numberFieldsLength + 5 * splitSymbolCompensation;

```

```

        StringDelete(tempStr, prevCount);
        tempFloat = atof(tempStr);
        table->neckRadius = tempFloat;
    } else {
        perror("Oops, seems like something's wrong with your memory!");
        exit(10);
    }
} else {
    perror("Oops, seems like something's wrong with your memory!");
    exit(11);
}
free(tempStr);
free(temp2);
tempStr = NULL;
temp2 = NULL;
}

```

```

void stringsWidthInp(int count, char **source, struct unit *table)
{
    char *tempStr = NULL;
    int i, iteration = 0, temp;

    tempStr = (char *) malloc(sizeof(char) * (strlen(*source) + 1));
    if (tempStr != NULL) {
        strcpy(tempStr, *source);
        StringDelete(tempStr, count);
        table->stringsWidth = (int*)malloc(sizeof(int) * (table->numOfString));
    }
}

```

```

if (table->stringsWidth != NULL) {

    for (i = 0; iteration < table->numOfString; i++) {
        if (tempStr[i] == sep || tempStr[i + 1] == '\0') {
            table->stringsWidth[iteration] = atoi(tempStr);
            temp = atoi(tempStr);
            StringDelete(tempStr,i);
            i = 0;
            iteration++;
        }
    }

} else {
    perror("Oops, seems like something's wrong with your memory!");
    exit(13);
}

} else {
    perror("Oops, seems like something's wrong with your memory!");
    exit(12);
}

free(tempStr);
tempStr = NULL;
}

void strRead(FILE *df, char **dest)
{
    char symbol = '\0';

```



```

int length = 0, multiplier = 1, errCount = 0;

(*dest) = (char *)malloc(sizeof(char) * (readStep + 1));
while (symbol != '\n') {
    symbol = fgetc(df);
    if (symbol >= 32) {
        if ((length % readStep == 0) && (length != 0)) {
            multiplier++;
            (*dest) = (char *) realloc((*dest), sizeof(char) * (readStep *
multiplier + 1));
            if ((*dest) == NULL) {
                perror("Ooops, looks like there's an error with memory
reassignment...");
                exit(1);
            }
        }
        (*dest)[length] = symbol;
        length++;
    } else errCount++;
}
(*dest)[length] = '\0';
}

```

```

int csvCount(FILE *df)
{
    char symbol = '\0';
    int count = 0;

```

```

while (symbol != EOF) {
    symbol = fgetc(df);
    if (symbol == '\n') {
        count++;
    }
}
return count;
}

```

```

void unitFill(FILE *df, struct unit **table,int i)
{
    char *str = NULL;
    int j,endFlag = 0;
    table[i] = (guitar*)malloc(sizeof(guitar));

    strRead(df, &str);
    for (j = 0;endFlag < 7;j++) {
        if (str[j] == sep || str[j + 1] == '\0') {
            if (endFlag == 0) {
                nameInp(j,&str,table[i]);
                endFlag++;
            } else if (endFlag == 1) {
                infoInp(j,&str,table[i]);
                endFlag++;
            } else if (endFlag == 2) {
                pickupsInp(j,&str,table[i]);
                endFlag++;
            } else if (endFlag == 3) {

```

```

        fretsInp(j,&str,table[i]);
        endFlag++;
    } else if (endFlag == 4) {
        numStringsInp(j,&str,table[i]);
        endFlag++;
    } else if (endFlag == 5) {
        menzureInp(j,&str,table[i]);
        endFlag++;
    } else if (endFlag == 6) {
        radiusInp(j,&str,table[i]);
        stringsWidthInp(j,&str,table[i]);
        endFlag++;
    }
}
}
}
free(str);
str = NULL;
}

```

```

void keyboardUnitFill(struct unit *table,char *str)

```

```

{
    int j,endFlag = 0;

    for (j = 0;endFlag < 7;j++) {
        if (str[j] == sep || str[j + 1] == '\0') {
            if (endFlag == 0) {
                nameInp(j, &str, table);
                endFlag++;
            }
        }
    }
}

```

```

    } else if (endFlag == 1) {
        infoInp(j, &str, table);
        endFlag++;
    } else if (endFlag == 2) {
        pickupsInp(j, &str, table);
        endFlag++;
    } else if (endFlag == 3) {
        fretsInp(j, &str, table);
        endFlag++;
    } else if (endFlag == 4) {
        numStringsInp(j, &str, table);
        endFlag++;
    } else if (endFlag == 5) {
        menzureInp(j, &str, table);
        endFlag++;
    } else if (endFlag == 6) {
        radiusInp(j, &str, table);
        stringsWidthInp(j, &str, table);
        endFlag++;
    }
}
}
}
}

```

```

guitar **fileInput(struct unit **table, int *N) {
    FILE *df;
    char filename[nameLength];
    int length,i = 0;

```

```

int size;

do {

    puts("Enter the name of the file from which you want to read the data
(without the .csv reference):\n"

        "WARNING!! MAX length of the file name is 30!");
    if (i == 0) {
        getchar();
        i++;
    }
    fgets(filename, inputLength, stdin);
    while (filename[0] == '\n') {
        puts("Empty string cannot be a filename!\nPlease try again:");
        fgets(filename, inputLength, stdin);
    }
    length = strlen(filename);
    if (filename[length - 1] == '\n') {
        filename[length - 1] = '\0';
    }
    strcat(filename, ".csv");
    df = fopen(filename, "r");

    if (!df) {
        puts("File does not exist, please enter another name!");
        fclose(df);
        getchar();
        system("cls");
    }
}

```

```

    }

} while (!df);

size = csvCount(df);
table = (guitar**)malloc(sizeof(guitar*) * size);
if ((*table) != NULL) {
    fseek(df,0,SEEK_SET);
    for (i = 0;i < size;i++) {
        unitFill(df,table,i);
        (*N)++;
    }
} else {
    perror("Ooops, something went wrong with the memory...");
    exit(2);
}
fclose(df);

return table;
}

void scanField(char **dest,char **source,int *mult,int mod)
{
    char split[] = ";";
    int len;

    strRead(stdin, source);
    if (*dest == NULL) {

```

```

        *dest = (char*)malloc(sizeof(char) * (keyboardReadStep + 1));
        strcpy(*dest,*source);
        strcat(*dest, split);
    } else {
        len = strlen(*dest) + strlen(*source);
        if (len / (keyboardReadStep * (*mult)) >= 1) {
            (*mult)++;
            realloc(*dest,sizeof(char) * ((*mult) * keyboardReadStep + 1));
        }
        strcat(*dest, *source);
        if (mod==0) {
            strcat(*dest, split);
        }
    }
    free(*source);
    *source = NULL;
}

```

```

char *DataScan()
{
    char *inp = NULL,*temp = NULL,split[] = "";
    int multiplier = 0,numOfStrings,mult = 1;

    puts("Enter unit name:");
    scanField(&inp,&temp,&mult,0);

    puts("Enter description or special info:");
    scanField(&inp,&temp,&mult,0);
}

```

```
puts("Enter number of pickups:");
scanfField(&inp,&temp,&mult,0);
```

```
puts("Enter number of frets:");
scanfField(&inp,&temp,&mult,0);
```

```
puts("Enter number of strings:");
strRead(stdin,&temp);
if ((strlen(inp) + strlen(temp)) / (keyboardReadStep * mult) >= 1) {
    mult++;
    realloc(inp,sizeof(char) * (mult * keyboardReadStep + 1));
}
strcat(inp,temp);
strcat(inp,split);
numOfStrings = atoi(temp);
```

```
puts("Enter menzure length:");
scanfField(&inp,&temp,&mult,0);
```

```
puts("Enter neck radius:");
scanfField(&inp,&temp,&mult,0);
```

```
puts("Enter strings width");
while (multiplier < numOfStrings) {
    if (multiplier == numOfStrings - 1) {
        scanfField(&inp,&temp,&mult,1);
    } else {
```



```

        scanField(&inp,&temp,&mult,0);
    }
    multiplier++;
}

return inp;
}

void keyboardInput(struct unit ***table,int *N)
{
    char *data = NULL;

    data = DataScan();
    (*N)++;
    (*table) = (guitar**)realloc((*table),sizeof(guitar*) * (*N));
    (*table)[(*N)-1] = (guitar*)malloc(sizeof(guitar));

    if (table != NULL) {
        keyboardUnitFill((*table)[(*N)-1],data);
    } else {
        perror("Ooops, something went wrong with the memory...");
        exit(2);
    }

    free(data);
    data = NULL;
}

```

```

void nameSort(struct unit **table,int N,char *search)
{
    int i,j,len,startpoint = 0,printed = 0,scanFlag = 0;
    char *temp = NULL;

    for (i = 0;i < N;i++) {
        len = strlen(table[i]->name);
        for (j = 0;j < len;j++) {
            if (table[i]->name[j] == ' ') {
                startpoint = j;
            }
        }
        temp = (char*)malloc(sizeof(char) * (len + 1));
        strcpy(temp,table[i]->name);
        if (temp[0] != 32 && startpoint == 0) {} else {
            StringDelete(temp, startpoint);
        }
        startpoint = 0;

        for (j = 0;j < strlen(temp);j++) {
            temp[j] = toupper(temp[j]);
        }
        if (scanFlag == 0) {
            for (j = 0; j < strlen(search); j++) {
                search[j] = toupper(search[j]);
            }
            scanFlag++;
        }
    }
}

```

```

        if (strcmp(temp,search) == 0) {
            structPrint(table[i]);
            printed++;
        }
    }
    if (printed == 0) {
        printf("No results!\n\n");
    }
    free(temp);
    temp = NULL;
}

```

```

void pickupsSort(struct unit **table,int N)
{
    int i,j;
    guitar *tempStruct = NULL;

    for (i = 1; i < N; i++) {
        tempStruct = table[i];
        for (j = i - 1;(j >= 0) && (table[j]->numOfPickups < tempStruct-
>numOfPickups);j--) {
            table[j + 1] = table[j];
            table[j] = tempStruct;
        }
    }
    tempStruct = NULL;
}

```

```
}
```

```
void fretsSort(struct unit **table, int N)
```

```
{
```

```
    int i,j;
```

```
    guitar *tempStruct = NULL;
```

```
    for (i = 1;i < N;i++) {
```

```
        tempStruct = table[i];
```

```
        for (j = i - 1;j >= 0) && (table[j]->numOfFrets < tempStruct-  
>numOfFrets);j--) {
```

```
            table[j + 1] = table[j];
```

```
            table[j] = tempStruct;
```

```
        }
```

```
    }
```

```
    tempStruct = NULL;
```

```
}
```

```
void numberOfStringsSort(struct unit **table, int N)
```

```
{
```

```
    int i,j;
```

```
    guitar *tempStruct = NULL;
```

```
    for (i = 1;i < N;i++) {
```

```
        tempStruct = table[i];
```

```
        for (j = i - 1;j >= 0) && (table[j]->numOfString < tempStruct-  
>numOfString);j--) {
```

```
            table[j + 1] = table[j];
```

```

        table[j] = tempStruct;
    }
}
tempStruct = NULL;
}

```

```

void menzureSort(struct unit **table, int N)
{
    int i,j;
    guitar *tempStruct = NULL;

    for (i = 1;i < N;i++) {
        tempStruct = table[i];
        for (j = i - 1;(j >= 0) && (table[j]->menzureLength < tempStruct-
>menzureLength);j--) {
            table[j + 1] = table[j];
            table[j] = tempStruct;
        }
    }
    tempStruct = NULL;
}

```

```

void radiusSort(struct unit **table, int N)
{
    int i, j;
    guitar *tempStruct = NULL;

    for (i = 1; i < N; i++) {

```

```

        tempStruct = table[i];

        for (j = i - 1; (j >= 0) && (table[j]->neckRadius < tempStruct-
>neckRadius); j--) {
            table[j + 1] = table[j];
            table[j] = tempStruct;
        }
    }
    tempStruct = NULL;
}

```

```

void stringWidthSort(struct unit **table, int N)
{
    int i,j;
    guitar *tempStruct = NULL;

    for (i = 1;i < N;i++) {
        tempStruct = table[i];

        for (j = i - 1;(j >= 0) && (table[j]->stringsWidth[0] < tempStruct-
>stringsWidth[0]);j--) {
            table[j + 1] = table[j];
            table[j] = tempStruct;
        }
    }
    tempStruct = NULL;
}

```

```

void numberSearch(struct unit **table, int N, void (*funcName)(guitar**, int))
{

```

```
funcName(table,N);  
printAll(table,N);  
}
```

```
int main() {
```

```
    guitar **table = NULL;
```

```
    int flag, csvReadFlag = 0, i, trash, N = 0, sortCount = 0;
```

```
    char *search = NULL;
```

```
    do {
```

```
        system("cls");
```

```
        MainMenu();
```

```
        flag = getchar();
```

```
        i = 0;
```

```
        if (flag == '\n') while ((trash = getchar()) != '\n') i++;
```

```
        if (i) {
```

```
            flag = '';
```

```
        }
```

```
        switch (flag) {
```

```
            case '1':
```

```
                if (csvReadFlag == 0) {
```

```
                    system("cls");
```

```
                    table = fileInput(table, &N);
```

```
                }
```

```
                do {
```

```
                    system("cls");
```

```
                    AddMenu();
```

```

    if (csvReadFlag != 0 && flag != 51 && flag != 56 && flag != 50)
{
    getchar();
} else if (csvReadFlag == 0) csvReadFlag++;
flag = getchar();
i = 0;
if (flag != '\n') while ((trash = getchar()) != '\n') i++;
if (i) {
    flag = '';
}
switch (flag) {
    case '1':
        do {
            //system("cls");
            keyboardInput(&table,&N);
            puts("1 - Enter one more line");
            puts("8 - Stop and proceed to sorting");
            flag = getchar();
            i = 0;
            if (flag != '\n') while ((trash = getchar()) != '\n') i++;
            if (i) {
                flag = '';
            }
        } while (flag != '8');
        break;
    case '2':
        do {
            system("cls");

```



```

mainSortMenu();

flag = getchar();

i = 0;

if (flag == '\n') while ((trash = getchar()) != '\n') i++;

if (i) {
    flag = '';
}

switch (flag) {
    case '1':
        system("cls");
        printf("Enter searching word (non case-sensitive): ");
        getchar();
        strRead(stdin,&search);
        printf("-----\n");
        nameSort(table, N, search);
        free(search);
        search = NULL;
        puts("Press ENTER to return");
        i = 0;
        if (flag != '\n') while ((trash = getchar()) != '\n') i++;
        if (i) {
            flag = '';
        }
        break;
    case '2':
        do {
            system("cls");
            numberSortMenu();

```

```

        if (sortCount == 0) {
            getchar();
            sortCount++;
        }
        flag = getchar();
        i = 0;
        if (flag == '\n') while ((trash = getchar()) != '\n') i++;
        if (i) {
            flag = '';
        }
        switch (flag) {
            case '1':
                system("cls");
                numberSearch(table,N,pickupsSort);
                puts("Press ENTER to return");
                i = 0;
                if (flag != '\n') while ((trash = getchar()) != '\n')
i++;

                if (i) {
                    flag = '';
                }
                getchar();
                break;
            case '2':
                system("cls");
                numberSearch(table,N,fretsSort);
                puts("Press ENTER to return");
                i = 0;

```

```

i++;

if (flag != '\n') while ((trash = getchar()) != '\n')

if (i) {
    flag = '';
}
getchar();
break;
case '3':
    system("cls");

numberSearch(table,N,numberOfStringsSort);

puts("Press ENTER to return");
i = 0;
if (flag != '\n') while ((trash = getchar()) != '\n')

i++;

if (i) {
    flag = '';
}
getchar();
break;
case '4':
    system("cls");
    numberSearch(table,N,menzureSort);
    puts("Press ENTER to return");
    i = 0;
    if (flag != '\n') while ((trash = getchar()) != '\n')

i++;

if (i) {
    flag = '';

```

```

    }
    getchar();
    break;
case '5':
    system("cls");
    numberSearch(table,N,radiusSort);
    puts("Press ENTER to return");
    i = 0;
    if (flag != '\n') while ((trash = getchar()) != '\n')
i++;

    if (i) {
        flag = "";
    }
    getchar();
    break;
case '6':
    system("cls");
    numberSearch(table,N,stringWidthSort);
    puts("Press ENTER to return");
    i = 0;
    if (flag != '\n') while ((trash = getchar()) != '\n')
i++;

    if (i) {
        flag = "";
    }
    getchar();
    break;
case '7':

```

```

        sortCount = 0;

        getchar();

        break;
default:
        system("cls");

        puts("There's no such paragraph!");

        puts("Press ENTER when ready...");

        do {

                flag = getchar();

        } while (flag != '\n');

    }

    } while (flag != '7');

    //flag = '';

    break;
case '8':

    break;
default:

    system("cls");

    puts("There's no such paragraph!");

    puts("Press ENTER when ready...");

    do {

            flag = getchar();

    } while (flag != '\n');

    }

    } while (flag != '8');

    flag = '';

    break;
case '3':

```

```

        system("cls");
        printAll(table, N);
        puts("Press ENTER to return");
        i = 0;
        if (flag != '\n') while ((trash = getchar()) != '\n') i++;
        if (i) {
            flag = '\n';
        }
        break;
    case '9':
        break;
    default:
        system("cls");
        puts("There's no such paragraph!");
        puts("Press ENTER when ready...");
        do {
            flag = getchar();
        } while (flag != '\n');
    }
} while (flag != '9');
flag = '\n';
break;
case '2':
    system("cls");
    InfoMenu();
    puts("Press ENTER to return");
    i = 0;
    if (flag != '\n') while ((trash = getchar()) != '\n') i++;

```

```

        if (i) {
            flag = '\n';
        }
        getchar();
        break;
    case '0':
        for (i = 0; i < N; i++) {
            clearUnit(table[i]);
        }
        free(table);
        table = NULL;
        break;
    default:
        system("cls");
        puts("There's no such paragraph!");
        puts("Press ENTER when ready...");
        do {
            flag = getchar();
        } while (flag != '\n');
    } while (flag != '0');

    return 0;
}

void MainMenu()
{
    puts("1 - Start");

```

```
    puts("2 - Info");
    puts("0 - Exit");
}
```

```
void AddMenu()
{
    puts("1 - Add custom lines from keyboard");
    puts("2 - Proceed to sorting");
    puts("3 - Print current table");
    puts("9 - Back");
}
```

```
void InfoMenu()
{
    puts("NOTE!!! The 'pick lines with serain word' sorting prints all the
structure lines with the chosen word\nequal the last word in line's name");
}
```

```
void numberSortMenu()
{
    puts("Pick the number field by which you want to sort:");
    puts("1 - Number of pickups\n2 - Number of frets\n3 - Number of strings\n4
- Menzure length\n5 - Neck radius");
    puts("6 - Strings width (sorted by the thinnest string)\n7 - Exit");
    //getchar();
}
```

```
void mainSortMenu()
```



```

{
    printf("1 - Pick lines with certain word\n");
    printf("2 - Number fields sorting\n");
    printf("8 - Back\n");
}

```

```

void printAll(struct unit **table, int N)

```

```

{
    int i,j;

    for (i = 0;i < N;i++) {
        printf("Name: %s;\n", table[i]->name);
        printf("Description/info: %s;\n",table[i]->info);
        printf("Number of pickups: %d;\n", table[i]->numOfPickups);
        printf("Number of frets: %d;\n",table[i]->numOfFrets);
        printf("Number of strings: %d;\n",table[i]->numOfString);
        printf("Menzure length: %.2f;\n", table[i]->menzureLength);
        printf("Neck radius: %.2f;\n", table[i]->neckRadius);
        printf("Strings width: ");
        for (j = 0;j < table[i]->numOfString;j++) {
            printf("%d ",table[i]->stringsWidth[j]);
        }
        printf("\n\n");
    }
}

```

```

void structPrint(struct unit *table)

```

```

{

```

```

int j;

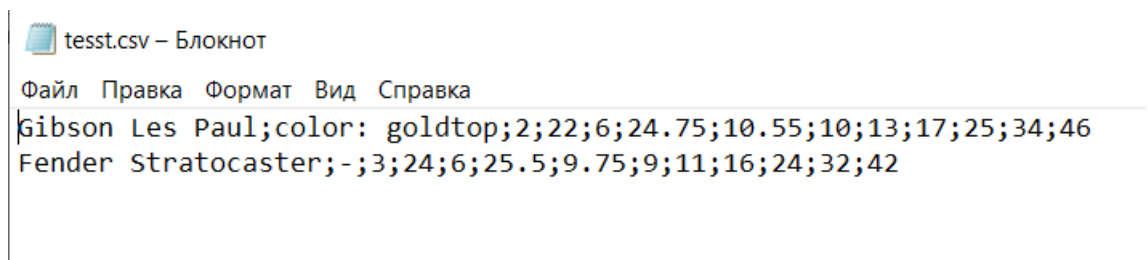
printf("Name: %s;\n", table->name);
printf("Description/info: %s;\n",table->info);
printf("Number of pickups: %d;\n", table->numOfPickups);
printf("Number of frets: %d;\n",table->numOfFrets);
printf("Number of strings: %d;\n",table->numOfString);
printf("Menzure length: %.2f;\n", table->menzureLength);
printf("Neck radius: %.2f;\n", table->neckRadius);
printf("Strings width: ");
for (j = 0;j < table->numOfString;j++) {
    printf("%d ",table->stringsWidth[j]);
}
printf("\n\n");
}

```

Пример работы программы

Исходные данные:

Filename = "tesst"



```

tesst.csv – Блокнот
Файл  Правка  Формат  Вид  Справка
Gibson Les Paul;color: goldtop;2;22;6;24.75;10.55;10;13;17;25;34;46
Fender Stratocaster;-;3;24;6;25.5;9.75;9;11;16;24;32;42

```

Вывод программы:

C:\Users\Artem\CLionProjects\StructLab9\cmake-build-debug\StructLab9.exe

Name: Gibson Les Paul;
Description/info: color: goldtop;
Number of pickups: 2;
Number of frets: 22;
Number of strings: 6;
Menzure length: 24.75;
Neck radius: 10.55;
Strings width: 10 13 17 25 34 46

Name: Fender Stratocaster;
Description/info: -;
Number of pickups: 3;
Number of frets: 24;
Number of strings: 6;
Menzure length: 25.50;
Neck radius: 9.75;
Strings width: 9 11 16 24 32 42

Press ENTER to return

C:\Users\Artem\CLionProjects\StructLab9\cmake-build-debug\StructLab9.exe

1 - Add custom lines from keyboard
2 - Proceed to sorting
3 - Print current table
0 - Back

1
Enter unit name:
PRS Custom 24
Enter description or special info:
Costs A LOT
Enter number of pickups:
2
Enter number of frets:
24
Enter number of strings:
6
Enter menzure length:
25
Enter neck radius:
10.7
Enter strings width
10
20
30
40
50
60

1 - Enter one more line
3 - Stop and proceed to sorting

```
C:\Users\Artem\CLionProjects\StructLab9\cmake-build-debug\StructLab9.exe
Enter searching word (non case-sensitive): Paul
-----
Name: Gibson Les Paul;
Description/info: color: goldtop;
Number of pickups: 2;
Number of frets: 22;
Number of strings: 6;
Menzure length: 24.75;
Neck radius: 10.55;
Strings width: 10 13 17 25 34 46

Press ENTER to return
```

Сортировка по значению поля menzure:

```
C:\Users\Artem\CLionProjects\StructLab9\cmake-build-debug\StructLab9.exe
Name: Fender Stratocaster;
Description/info: -;
Number of pickups: 3;
Number of frets: 24;
Number of strings: 6;
Menzure length: 25.50;
Neck radius: 9.75;
Strings width: 9 11 16 24 32 42

Name: PRS Custom 24;
Description/info: Costs A LOT;
Number of pickups: 2;
Number of frets: 24;
Number of strings: 6;
Menzure length: 25.00;
Neck radius: 10.70;
Strings width: 10 20 30 40 50 60

Name: Gibson Les Paul;
Description/info: color: goldtop;
Number of pickups: 2;
Number of frets: 22;
Number of strings: 6;
Menzure length: 24.75;
Neck radius: 10.55;
Strings width: 10 13 17 25 34 46

Press ENTER to return
```

Заключение

Выводы:

При выполнении лабораторной работы были получены практические навыки в разработке алгоритмов с использованием указателей на структуры и функции посредством написания программы на языке Си.