

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

Кафедра вычислительной техники

Отчет по лабораторной работе № 12
по дисциплине «Программирование»
Тема: кольцевые списки

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	19
Пример работы программы	38
Заключение	41

Цель

Получить практические навыки в разработке алгоритмов с использованием кольцевых списков на языке Си.

Задание

На основе линейного списка, созданного при выполнении лабораторной работы №11, создать кольцевой односвязный список и разработать подалгоритм, а также написать функцию, перемещающую «голову» списка в заданную позицию (после элемента с указанным номером).

Постановка задачи и описание решения

Дано: CSV файл, содержащий структуры.

Требуется получить: кольцевой список, «голова» которого была установлена после элемента с номером, выбранным пользователем.

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

В программе реализовано два различных способа удаления элемента: классическое для линейного списка удаление (“pop”), а также удаление элемента с номером, введенным пользователем.

Перемещение «головы» списка осуществляется следующим образом: новым элементом, следующим за «хвостом» становится элемент, следующий за «головой». Затем следующим за «головой» элементом становится элемент, следующий за выбранным пользователем элементом. После этого «голова» «вклинивается» на место элемента, следующего за выбранным пользователем. Таким образом указатель, до этого момента указывавший на элемент, идущий следом за выбранным пользователем, теперь указывает на перемещенную «голову»

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

Таблица 2.Описание структуры (typedef struct node_unit {...} node)

Имя поля	Тип	Назначение
guitar	guitar*	Указатель на информационную структуру
next	struct node_unit*	Указатель на следующий элемент списка

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	name	char*	Имя файла, из которого считываются данные
Локальная	head	node*	Указатель на начало списка
Локальная	tail	node*	Указатель на конец списка
Локальная	df	FILE*	Указатель на файл с именем name
Локальная	i	int	Счетчик цикла
Локальная	length	int	Длина линейного списка
Локальная	deleteNumber	int	Номер удаляемого элемента
Локальная	moveNumber	int	Номер элемента, после которого ставится голова
Локальная	count	int	Количество элементов списка в файле
Локальная	flag	char	Выбор пункта меню
Локальная	trash	char	Переменная для сбора мусора при ошибках

			считывания выбора пункта меню
--	--	--	----------------------------------

Возвращаемое значение:

2.MainMenu

Описание:

Вывод главного меню.

Прототип:

Void MainMenu();

Пример вызова:

MainMenu();

Возвращаемое значение:

3.Info

Описание:

Вывод меню помощи.

Прототип:

Void info();

Пример вызова:

info();

Возвращаемое значение:

4.startMenu

Описание:

Вывод меню основных действий.

Прототип:

```
void startMenu();
```

Пример вызова:

```
startMenu();
```

Возвращаемое значение:

5.deleteMenu

Описание:

Вывод меню удаления.

Прототип:

```
void deleteMenu();
```

Пример вызова:

```
deleteMenu();
```

Возвращаемое значение:

6.getData

Описание:

Ввод данных из строки и создание двумерного массива с этими данными для создания элемента списка.

Прототип:

```
char **getData(FILE *source);
```

Пример вызова:

```
data = getData(source);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	source	FILE*	Указатель на файл

Локальная	arrData	char**	Указатель на двумерный массив
Локальная	input	char*	Входная строка с данными
Локальная	shift	int	Сдвиг для ввода данных в строку двумерного массива
Локальная	i	int	Счетчик цикла
Локальная	count	int	Счетчик строки двумерного массива
Локальная	len	int	Длина строки с входными данными
Локальная	errorCount	int	Счетчик для очистки массива в случае ошибки
Локальная	trigger	char	Флаг ошибки выделения памяти

Возвращаемое значение: указатель на двумерный массив данных.

7.strRead

Описание:

Считывание строки из файла, на который указывает df.

Прототип:

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

Пример вызова:

```
strRead(source,&input);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	df	FILE*	Указатель на файл

Формальный аргумент	dest	char**	Указатель на строку, в которую производится считывание
Локальная	symbol	char	Символ, считываемый в данный момент
Локальная	length	int	Длина строки с данными
Локальная	multiplier	int	Множитель для увеличения размера строки
Локальная	errCount	int	Счетчик «мусорного» ввода

Возвращаемое значение:

8.clearStrArray

Описание:

Очищает память, выделенную под массив arr.

Прототип:

```
void clearStrArray(char **arr,int count);
```

Пример вызова:

```
clearStrArray(arrData,errorCount);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	count	int	Количество строк в массиве (удаляемых строк)
Формальный аргумент	arr	char**	Двумерный массив символов
Локальная	i	int	Счетчик цикла

Возвращаемое значение:

9.getInput

Описание:

Выбор источника ввода.

Прототип:

```
char *getInput();
```

Пример вызова:

```
name = getInput();
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Локальная	name	char*	Строка в которую записывается имя файла

Возвращаемое значение: строка, содержащая имя файла.

10.lenCount

Описание:

Подсчет количества строк в файле с указателем filename.

Прототип:

```
int lenCount(FILE *filename);
```

Пример вызова:

```
count = lenCount(df);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	filename	FILE*	Указатель на файл
Локальная	symbol	char	Считываемый из файла в данный момент символ
Локальная	i	int	Счетчик цикла

Возвращаемое значение: количество строк в файле.

11.create_node

Описание:

Создание элемента списка.

Прототип:

```
node *create_node (FILE *source,int *length);
```

Пример вызова:

```
temp = create_node(source,length);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	source	FILE*	Указатель на файл
Формальный аргумент	length	int*	Указатель на длину списка
Локальная	i	int	Счетчик цикла
Локальная	data	char**	Двумерный массив данных, которыми заполняется элемент списка

Возвращаемое значение: указатель на элемент списка.

12.print_from_head

Описание:

Вывод линейного списка, началом которого является head, на экран (в прямом порядке).

Прототип:

```
void print_from_head(node **head);
```

Пример вызова:

```
print_from_head(&head);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	head	node**	Двойной указатель на голову списка
Локальная	i	int	Счетчик цикла
Локальная	p	node*	Временный указатель на элемент списка

Возвращаемое значение:

13.headMoves

Описание:

Перемещение «головы» на место после элемента с индексом index.

Прототип:

```
void headMoves(node **head,node **tail,int index);
```

Пример вызова:

```
headMoves(&head,&tail,moveNumber);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	head	node**	Двойной указатель на голову списка
Формальный аргумент	tail	node**	Двойной указатель на конец списка
Формальный аргумент	index	int	Номер элемента, после которого устанавливается «голова»
Локальная	i	int	Счетчик цикла
Локальная	temp	node*	Временный указатель на элемент списка

Возвращаемое значение:

14.add_first

Описание:

Добавление элемента в список на место head.

Прототип:

```
void add_first(node **head,node **tail,FILE *source,int *length);
```

Пример вызова:

```
add_first(&head,&tail,df,&length);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	head	node**	Двойной указатель на голову списка
Формальный аргумент	tail	node**	Двойной указатель на конец списка

Формальный аргумент	source	FILE*	Указатель на файл
Формальный аргумент	length	int*	Указатель на длину списка
Локальная	temp	node*	Временный указатель на элемент списка

Возвращаемое значение:

15.add_last

Описание:

Добавление элемента в список на место tail.

Прототип:

```
void add_last(node **tail, FILE *source, int *length);
```

Пример вызова:

```
add_last(&tail, df, &length);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	tail	node**	Двойной указатель на конец списка
Формальный аргумент	source	FILE*	Указатель на файл
Формальный аргумент	length	int*	Указатель на длину списка
Локальная	temp	node*	Временный указатель на элемент списка

Возвращаемое значение:

16.insert

Описание:

Добавление в список элемента на указанную позицию index.

Прототип:

```
void insert(node **head, node **tail, int index, FILE *source, int *length);
```

Пример вызова:

```
insert(&head, &tail, i, source, &length);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	head	node**	Двойной указатель на начало списка
Формальный аргумент	tail	node**	Двойной указатель на конец списка
Формальный аргумент	index	int	Позиция добавляемого элемента
Формальный аргумент	source	FILE*	Указатель на файл
Формальный аргумент	length	int*	Указатель на длину списка
Локальная	p	node*	Вспомогательный указатель
Локальная	temp	node*	Вспомогательный указатель
Локальная	i	int	Счетчик цикла

Возвращаемое значение:

17.pop

Описание:

Удаление первого добавленного в список элемента.

Прототип:

```
void pop(node **tail,int *length);
```

Пример вызова:

```
pop(&tail,&length);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	tail	node**	Двойной указатель на конец списка
Формальный аргумент	length	int*	Указатель на длину списка
Локальная	temp	node*	Вспомогательный указатель

Возвращаемое значение:

18.removeNode

Описание:

Удаляет элемент перед элементом с номером i.

Прототип:

```
void removeNode(node **head,int i,int *length);
```

Пример вызова:

```
removeNode(&head,deleteNumber,&length);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение

Формальный аргумент	head	node**	Двойной указатель на начало списка
Формальный аргумент	i	int	Номер удаляемого элемента +1
Формальный аргумент	length	int*	Указатель на длину списка
Локальная	temp	node*	Вспомогательный указатель на элемент списка
Локальная	mem	node*	Вспомогательный указатель на элемент списка
Локальная	j	int	Счетчик цикла

Возвращаемое значение:

19.delete

Описание:

Удаление конкретного элемента unit списка.

Прототип:

```
void delete(node **unit);
```

Пример вызова:

```
delete(&temp);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	unit	node**	Двойной указатель на удаляемый элемент

Возвращаемое значение:

20.clearList

Описание:

Очистка памяти, выделенной под список.

Прототип:

```
void clearList(node **head,int length);
```

Пример вызова:

```
clearList(&head,length);
```

Описание переменных:

Вид переменной	Имя поля	Тип	Назначение
Формальный аргумент	head	node**	Двойной указатель на начало списка
Формальный аргумент	length	int	Указатель на длину списка
Локальная	temp	node*	Вспомогательный указатель
Локальная	i	int	Счетчик цикла

Возвращаемое значение:

Текст программы

1) *Файл main.c*

```
#include <stdio.h>
#include <stdlib.h>
#include "dataRead.h"
#include "list.h"
#include "menu.h"
```

```
int main()
```

```
{
```

```
    char *name = NULL;
```

```
    node *head = NULL, *tail = NULL;
```

```
    FILE *df = NULL;
```

```
    int i, length, deleteNumber, moveNumber, count = 0;
```

```
    char flag, trash;
```

```
    do {
```

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

```
        mainMenu();
```

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

```
        getchar();
```

```
        i = 0;
```

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

```
    i++;
```

```
        if (i) {
```

```
            flag = ' ';
```

```
        }
```

```
        switch (flag) {
```

```
            case '1':
```

```
                //ввод значений и работа
```

```
                do {
```

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

```
                    startMenu();
```

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

```
                    getchar();
```

```

        i = 0;
        if (flag == '\n') while ((trash =
getchar()) != '\n') i++;
        if (i) {
            flag = '`';
        }
        switch (flag) {
            case '1':
                //file inp
                while (df == NULL) {
                    name = getInput();
                    df = fopen(name, "r");
                    if (df == NULL) {
                        printf("File %s is not
found!\n\n", name);

                    }
                }
                puts("File                opened
successfully!");

                puts("Press ENTER to continue!");
                getchar();
                system("cls");
                count = lenCount(df);
                if (count >= 1) {
                    length = 0;

                    add_first(&head, &tail, df, &length);

                    tail = head;
                    head->next = head;
                    for (i = 0; i < (count -
1); i++) {

                        add_last(&tail, df, &length);

                    }
                    puts("File                read
successfully!");

                } else {

```

```

        puts("The file is empty!");
    }
    puts("Press ENTER to continue");
    fseek(df,0,SEEK_SET);
    getchar();
    fclose(df);
    break;
case '2':
    //keyboard inp
    do {
        system("cls");
        puts("Enter dataline with
split symbols:");

        df = stdin;
        if (count == 0) {    //Если это
первый ввод

add_first(&head,&tail,df,&length);

        tail = head;
        head->next = tail;
        tail->next = NULL;
        count++;
    } else {

add_last(&tail,df,&length);

        count++;
    }
    puts("Enter one more line? (1
- Yes/2 - No)");

    flag = getchar();
    getchar();
    } while (flag == '1');
    break;
case '3':
    //delete node
    do {

```

```

system("cls");
deleteMenu();
flag = getchar();
getchar();
i = 0;
if (flag == '\n') while
((trash = getchar()) != '\n') i++;

if (i) {
    flag = '`';
}
switch (flag) {
    case '1':
        //pop
        pop(&tail,&length);
        puts("\nPress ENTER
to return...");

        getchar();
        break;
    case '2':
        //task delete
        puts("Enter number of
node you want to delete:");

        scanf("%d",&deleteNumber);

        getchar();

        removeNode(&head,deleteNumber,&length);

        puts("Press ENTER to
return...");

        getchar();
        break;
    case '8':
        break;
    default:
        system("cls");
        puts("There's no such
paragraph!");

```

```

when ready...");

puts("Press ENTER

do {
    flag = getchar();
} while (flag !=

'\n');

    }
} while (flag != '8');
break;
case '4':
    //print list
    print_from_head(&head);
    puts("Press ENTER to return...");
    getchar();
    break;
case '5':
    puts("Enter the number of
element, after which you want to set head");
    scanf("%d",&moveNumber);

headMoves(&head,&tail,moveNumber);
    getchar();
    puts("Press ENTER to return...");
    getchar();
    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');
        break;
    case '2':
        //вывод справки
        system("cls");
        info();
        i = 0;
        getchar();
        break;
    case '0':
        //выход из программы
        if (name != NULL) {
            free(name);
            name = NULL;
        }
        clearList(&head, length);
        head->next = NULL;
        tail->next = NULL;
        head = NULL;
        tail = NULL;
        df = 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;
}

```


2) *Файл menu.c*

```
#include "menu.h"
#include <stdio.h>

void mainMenu()
{
    puts("welcome to the linear list lab!\n");
    puts("1 - enter data");
    puts("2 - info");
    puts("0 - exit");
}

void info()
{
    puts("This program reads data from .csv file (or entered manually from keyboard) and creates a bidirectional circular list from it.\n");
    puts("All the data should be entered in line with split symbols\nEXAMPLE: 'name;info;number;2;5;123;23' (no split symbol at the end of line)\n");
    puts("To delete node you need to enter number of unit (count starts with '1', NOT '0').\n");
    puts("Press ENTER to return...");
}

void startMenu()
{
    puts("1 - enter from file");
    puts("2 - enter from keyboard");
    puts("3 - delete node");
    puts("4 - print from head");
    puts("5 - move head AFTER element with entered number");
    puts("9 - return");
}
```

```

void deleteMenu()
{
    puts("1 - classic delete ('pop')");
    puts("2 - delete unit with entered number");
    puts("8 - return");
}

```

3) *Файл menu.h*

```

#ifndef LAB10_MENU_H
#define LAB10_MENU_H

void mainMenu();    //Вывод главного меню    @@
void info();        //вывод справки            @@
void startMenu();   //меню ввода информации и удаления  @@
void deleteMenu();  //меню выбора режима удаления    @@

#endif //LAB10_MENU_H

```

4) *Файл dataRead.c*

```

#include "dataRead.h"
#include "list.h"
#include <stdlib.h>
#include <string.h>

char **getData(FILE *source)
{
    char **arrData = NULL;
    char *input = NULL;
    int i, shift, count, len, errorCount, trigger = 0;

    strRead(source, &input);

```

```

len = strlen(input);
for (i = 0, count = 0; i < len; i++) {
    if (input[i] == split) {
        count++;
    }
}

arrData = (char**)malloc(sizeof(char*) * (count + 1));
if (arrData != NULL) {
    for (i = 0, errorCount = 0; i <= count; i++, errorCount++)
    {
        arrData[i] = (char*)malloc(sizeof(char) * len);
        if (arrData[i] != NULL) trigger = 1;
        else {
            trigger = 0;
            i = count;
        }
    }
    if (trigger) {
        count = 0;
        shift = 0;
        for (i = 0; i < len; i++) {
            if (input[i] != split) {
                arrData[count][i - shift] = input[i];
            } else {
                arrData[count][i - shift] = '\\0';
                count++;
                shift = i + 1;
            }
        }
        arrData[count][i - shift] = '\\0';
    } else {
        clearStrArray(arrData, errorCount);
    }
}

```

```

    free(input);
    input = NULL;
    return arrData;
}

```

```

void clearStrArray(char **arr,int count)
{
    int i;

    for (i = 0;i < count;i++) {
        free(arr[i]);
        arr[i] = NULL;
    }
    free(arr);
    arr = 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';
}

```

```

char *getInput()
{
    char *name = NULL;

    name = (char*)malloc(sizeof(char) * maxlen);
    puts("Enter filename:");
    fgets(name, (maxlen - referenceBuff), stdin);
    name[strlen(name) - 1] = '\0';
    if (strcmp(name, "standard") == 0) {
        name = "stdin";
    } else {
        strcat(name, ".csv");
    }
    return name;
}

```

```

int lenCount(FILE *filename)
{
    int i = 0;
    char symbol = '\0';

    while (symbol != EOF) {
        symbol = fgetc(filename);
        if (symbol == '\n') {
            i++;
        }
    }
}

```

```

    }
    fseek(filename,0,SEEK_SET);
    return i;
}

```

5) *Файл dataRead.h*

```

#ifndef LAB10_DATAREAD_H
#define LAB10_DATAREAD_H
#include <stdio.h>

enum values {
    readStep = 15,
    maxlen = 45,
    referenceBuff = 5,
    constFields = 7,
    split = ';',
};

char **getData(FILE *source);      //формирование из строки
массива данных для структуры @@

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

void clearStrArray(char **arr,int count); //очистка массива
данных @@

char *getInput();    //выбор источника ввода @@

int lenCount(FILE *filename); //подсчет количества строк в
файле @@

#endif //LAB10_DATAREAD_H

```

6) *Файл list.c*

```

#include "list.h"
#include "dataRead.h"

```

```

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

node *create_node(FILE *source,int *length)
{
    int i;
    char **data = NULL;

    data = getData(source);

    node *temp;
    temp = (node*) malloc (sizeof(node));
    temp->guitar = (guitar*)malloc(sizeof(guitar));
    temp->guitar->name      =      (char*)malloc(sizeof(char) *
strlen(data[0]));
    strcpy(temp->guitar->name,data[0]);
    temp->guitar->info      =      (char*)malloc(sizeof(char) *
strlen(data[1]));
    strcpy((temp->guitar->info),data[1]);
    temp->guitar->numOfPickups = atoi(data[2]);
    temp->guitar->numOfFrets  = atoi(data[3]);
    temp->guitar->numOfString = atoi(data[4]);
    temp->guitar->menzureLength = atof(data[5]);
    temp->guitar->neckRadius  = atof(data[6]);
    temp->guitar->stringswidth = (int*)malloc(sizeof(int) *
temp->guitar->numOfString);
    for (i = 0;i < (temp->guitar->numOfString);i++) {
        temp->guitar->stringswidth[i] = atoi(data[7 + i]);
    }
    temp -> next = NULL;
    clearStrArray(data,(i + 7));
    (*length)++;
    return temp;
}

```

```

void print_from_head(node **head)
{
    node *p;
    int i;

    p = *head;
    system("cls");
    if (p == NULL) {
        puts("There's nothing to print");
    }
    do {
        printf("Name: %s\n", p->guitar->name);
        printf("Description/info: %s\n",p->guitar->info);
        printf("Number of Pickups: %d\n", p->guitar-
>numOfPickups);
        printf("Number of frets: %d\n",p->guitar-
>numOfFrets);
        printf("Number of strings: %d\n",p->guitar-
>numOfString);
        printf("Menzure length: %.2f\n",p->guitar-
>menzureLength);
        printf("Neck radius: %.2f\n",p->guitar->neckRadius);
        printf("Strings width: ");
        for (i = 0;i < (p->guitar->numOfString);i++) {
            printf("%d ",p->guitar->stringswidth[i]);
        }
        p = p->next;
        printf("\n\n");
    } while (p != (*head));
}

```

```

void add_first(node **head,node **tail,FILE *source,int
*length)
{
    node *temp;

```



```

    temp = create_node(source,length);
    temp->next = *head;
    if (*length != 1) {
        (*tail)->next = temp;
    }
    *head = temp;
}

```

```

void add_last(node **tail,FILE *source,int *length)
{
    node *temp;

    temp = create_node(source,length);
    temp->next = (*tail)->next;
    (*tail)->next = temp;
    *tail = temp;
}

```

```

void pop(node **tail,int *length)
{
    node *temp = NULL;

    if (*tail != NULL) {
        temp = (*tail)->next;
        (*tail)->next = (*tail)->next->next;
        (*length)--;
        delete(&temp);
        puts("Success!");
    } else puts("There's nothing to delete!");
}

```

```

void removeNode(node **head,int i,int *length) //удаляет
элемент с указанным номером
{
    node *temp = NULL,*mem = NULL;

```

```

int j = 0;

if (i != 0) {
    temp = *head;
    if (j == i - 1) {
        pop(head, length);
    } else {
        do {
            mem = temp;
            temp = temp->next;
            j++;
        } while (j != (i - 1));
        mem->next = temp->next;
        delete(&temp);
        (*length)--;
        puts("Deleted successfully!");
    }
} else {
    puts("You cannot delete the node before head! (node
does not exist!)");
}
if (temp != NULL) {
    temp = NULL;
}
}

void delete(node **unit)
{;
    (*unit)->guitar->name = NULL;
    (*unit)->guitar->info = NULL;
    (*unit)->guitar->stringswidth = NULL;
    free((*unit)->guitar);
    (*unit)->guitar = NULL;
    free((*unit));
    unit = NULL;
}

```

```
}
```

```
void insert(node **head,node **tail, int index,FILE  
*source,int *length)
```

```
{
```

```
    node *p,*temp;
```

```
    int i;
```

```
    if (index == 0) {
```

```
        add_first(head,tail,source,length);
```

```
    } else {
```

```
        p = *head;
```

```
        i = 0;
```

```
        while (i < index - 1 && p != NULL) {
```

```
            p = p->next;
```

```
            i++;
```

```
        }
```

```
        if (p == NULL) {
```

```
            puts("No such node!");
```

```
        } else {
```

```
            temp = create_node(source,length);
```

```
            temp->next = p->next->next;
```

```
            p->next = temp;
```

```
        }
```

```
    }
```

```
}
```

```
void clearList(node **head,int length)
```

```
{
```

```
    int i;
```

```
    node *temp = NULL;
```

```
    temp = *head;
```

```
    for(i = 0;i < length;i++) {
```

```
        if (temp->guitar->stringswidth != NULL) {
```

```

        free(temp->guitar->stringswidth);
        temp->guitar->stringswidth = NULL;
    }
    free(temp->guitar);
    temp->guitar = NULL;
    temp = temp->next;
}

}

void headMoves(node **head,node **tail,int index)
{
    node *temp;
    int i = 1;

    temp = *head;
    if (index != 1) {
        while (i != index) {
            temp = temp->next;
            i++;
        }
        (*tail)->next = (*head)->next;
        (*head)->next = temp->next;
        temp->next = (*head);
    }
}

```

7) *Файл list.h*

```

#ifndef LAB10_LIST_H
#define LAB10_LIST_H
#include <stdio.h>

typedef struct unit {

```

```

    char *name;
    char *info;
    int numOfPickups;
    int numOfFrets;
    int numOfString;
    float menzureLength;
    float neckRadius;
    int *stringsWidth;
} guitar;

typedef struct node_unit {
    guitar *guitar;
    struct node_unit *next;
} node;

node *create_node (FILE *source,int *length);    //создание
элемента списка    @@

void print_from_head(node **head);    //вывод списка с начала
@@

void add_first(node **head,node **tail,FILE *source,int
*length);    //добавление элемента на место head    @@

void add_last(node **tail,FILE *source,int *length);
//добавление элемента в конец списка    @@

void insert(node **head,node **tail, int index,FILE
*source,int *length);    //добавление в список элемента на
указанную позицию    @@

void pop(node **tail,int *length);    //удаление первого
элемента    @@

void removeNode(node **head,int i,int *length); //удаляет
элемент перед элементом с указанным номером    @@

void delete(node **unit);    //удаление node @@

void clearList(node **head,int length);    //очистка памяти,
выделенной под список    @@

void headMoves(node **head,node **tail,int index);
//перемещение головы на место после элемента с индексом index


#endif //LAB10_LIST_H

```

Пример работы программы

Исходные данные:


Name = "base"

 base.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
My first gibson les paul;hehe;2;24;4;25;10;10;20;30;40
blender mega stratocaster;avivaavivaavivaavivaaviva;3;22;3;25.75;9.55;10;20;30
PRS Custom 24;pls help me;2;24;6;25;10.7;10;13;17;30;42;52
Ibanez prestige;metal machine;3;24;7;25.5;11;9;11;16;24;32;42;54
Dean razorback;Dimebag Darrel's signature;2;24;6;25.5;10.5;24;34;44;56;72;84
ESP Mirage;With original floyd rose;2;24;6;24.75;12;9;11;16;24;32;42
Epiphone Les Paul;;;2;22;6;24.75;10.55;10;13;17;30;42;52
Inspector Arrow;Multimenzure;2;24;6;25.5;12;10;13;17;26;36;46
Gibson Flying V;70's reissue;2;22;6;24.75;10.55;12;16;20;34;46;60
Burny Les Paul;Good Japan copy;2;22;6;24.75;10.55;10;13;17;30;42;52
```

Вывод программы:

 C:\Users\Artem\CLionProjects\lab12\cmake-build-debug\lab12.exe

```
1 - enter from file
2 - enter from keyboard
3 - delete node
4 - print from head
5 - move head AFTER element with entered number
9 - return
5
Enter the number of element, after which you want to set head
4
Press ENTER to return...
```

C:\Users\Artem\CLionProjects\lab12\cmake-build-debug\lab

```
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: PRS Custom 24
Description/info: pls help me
Number of Pickups: 2
Number of frets: 24
Number of strings: 6
Menzure length: 25.00
Neck radius: 10.70
Strings width: 10 13 17 30 42 52

Name: Ibanez prestige
Description/info: metal machine
Number of Pickups: 3
Number of frets: 24
Number of strings: 7
Menzure length: 25.50
Neck radius: 11.00
Strings width: 9 11 16 24 32 42 54

Name: Dean razorback
Description/info: Dimebag Darrel's signature
Number of Pickups: 2
Number of frets: 24
Number of strings: 6
Menzure length: 25.50
Neck radius: 10.50
Strings width: 24 34 44 56 72 84

Name: ESP Mirage
Description/info: With original floyd rose
Number of Pickups: 2
Number of frets: 24
Number of strings: 6
Menzure length: 24.75
Neck radius: 12.00
Strings width: 9 11 16 24 32 42
```

Name: Epiphone Les Paul
Description/info:
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 30 42 52

Name: Inspector Arrow
Description/info: Multimenzura
Number of Pickups: 2
Number of frets: 24
Number of strings: 6
Menzure length: 25.50
Neck radius: 12.00
Strings width: 10 13 17 26 36 46

Name: Gibson Flying V
Description/info: 70's reissue
Number of Pickups: 2
Number of frets: 22
Number of strings: 6
Menzure length: 24.75
Neck radius: 10.55
Strings width: 12 16 20 34 46 60

Name: Burny Les Paul
Description/info: Good Japan copy
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 30 42 52

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: My first gibson les paul
Description/info: hehe
Number of Pickups: 2
Number of frets: 24
Number of strings: 4
Menzure length: 25.00
Neck radius: 10.00
Strings width: 10 20 30 40

Name: blender mega stratocaster
Description/info: avivaavivaavivaavivaaviva
Number of Pickups: 3
Number of frets: 22
Number of strings: 3
Menzure length: 25.75
Neck radius: 9.55
Strings width: 10 20 30

Press ENTER to return...
```

Заключение

Выводы:

При выполнении лабораторной работы были получены практические навыки в разработке алгоритмов с использованием кольцевых списков посредством написания программы на языке Си.