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

Кафедра вычислительной техники

Отчет по лабораторной работе № 11-12

по дисциплине «Программирование»

Тема: **ЛИНЕЙНЫЕ ДВУСВЯЗНЫЕ СПИСКИ. КОЛЬЦЕВЫЕ СПИСКИ.**

Студент гр. 9305 _____Ивашкин В.В.

Преподаватель _____Перязева Ю.В,

Санкт-Петербург

2020

Содержание

Введение

Получить практические навыки в разработке алгоритма и написании программы на языке Си для знакомства с синтаксисом и абстрактными типами данных, в частности, с линейными двусвязными списками, кольцевыми списками, а также правилами написания кода на языке Си.

Задание

Перепроектировать структуру, созданную при выполнении лабораторной работы №9 (по выбранной предметной области), так, чтобы одно из информационных полей, содержащих характеристику группы объектов (издательство, модель, тип и т. п.), стало ссылкой на элемент двусвязного линейного списка и выполнить задание в соответствии с вариантом.

Разработать подалгоритм создания двусвязного списка из имеющегося двусвязного списка путем удаления элементов с заданным значением указанного информационного поля. Порядок копируемых элементов должен соответствовать их порядку в исходном списке. В случае отсутствия подходящих элементов вывести сообщение.

Разработать подалгоритм и написать функцию, получающую кольцевой односвязный список и осуществляющую создание линейного списка со следующим расположением элементов. При четном количестве элементов в кольцевом списке (пример 1,2,3,4,5,6) получается список 3,2,1,6,5,4 , а при нечетном - (пример 1,2,3,4,5) получается 2,1,3,5,4 (средний элемент остается на своем месте).

Постановка задачи и описание решения

Дан файл, содержащий информацию о велосипедных деталях: наименование, тип, год выпуска, стоимость, количество отзывов и рейтинг на основе этих отзывов. Необходимо сформировать двусвязный линейный список, содержащий данные файла, реализовать функции вывода списка, добавления элемента и удаления элементов с заданным значением выбранного числового поля. Создать заголовочные файлы, в которых будут основные функции работы со структурами и со списками.

Отдельно: необходимо сформировать кольцевой односвязный список, содержащий данные файла, реализовать функцию создания линейного списка с таким расположением элементов: первая и вторая половины элементов по отдельности переставляются в обратном порядке. При нечетном количестве элементов, средний элемент остаётся на своём месте.

Пример: (1,2,3,4,5,6) в (3,2,1,6,5,4) и (1,2,3,4,5) в (2,1,3,5,4)

Сначала считывается каждая строка и с помощью функции simple_split каждый пункт данных сохраняется в массив строк. Массив строк сохраняется в структуру с помощью функции struct_fill. С помощью функции create_node создаётся элемент списка.

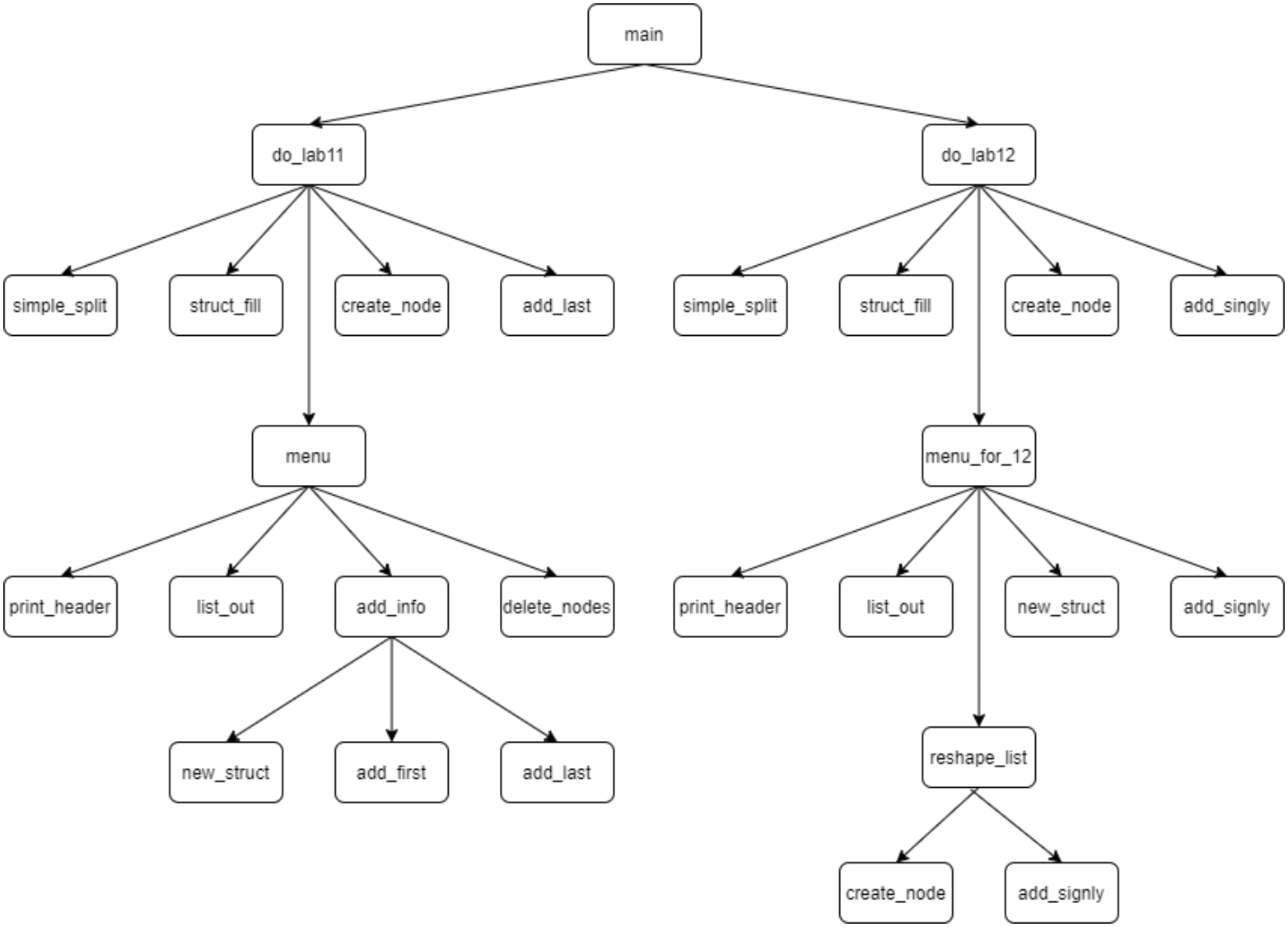
Далее с помощью меню пользователь может вывести список, добавить элемент и удалить определённые элементы.

Удаление происходит после того, как пользователь выберет числовое поле и введёт значение элементов, которые необходимо удалить. Алгоритм удаления работает таким образом, что сначала проверяется голова списка, и, если её необходимо удалить, проверка повторяется до тех пор, пока значение головы не будет совпадать с выбранным пользователем. После этого проверяются все элементы и удаляются.

При перестановке элементов, сначала так же происходит создание списка. Далее с помощью меню пользователь может либо сразу выполнить перестановку, либо добавить элементы.

Перестановка происходит таким образом: указатель доходит до середины списка. *Если количество элементов нечётное, элемент сохраняется и указатель переходит к следующему элементу.* Далее все элементы по порядку копируются в новый линейный список. Т.к. изначальный список кольцевой, копируются и элементы из первой половины. *Если количество элементов нечётное, сохранённый элемент вставляется в новый список в середине.*

Схема вызовов функций



Описание структур

1) struct bike_components

Имя поля	Тип поля	Назначение
name	char*	Наименование детали
type	char*	Тип детали
year	int	Год выпуска
cost	float	Стоимость
revievs	int	Количество отзывов
rating	float	Средняя оценка

2) struct node

Имя поля	Тип поля	Назначение
data	comps*	Информация, хранящаяся в элементе
next	struct node*	Указатель на следующий элемент
prev	struct node*	Указатель на предыдущий элемент

3) struct head

Имя поля	Тип поля	Назначение
cnt	int	Количество элементов в списке
first	struct node*	Указатель на первый элемент списка
last	struct node*	Указатель на последний элемент списка

Функции

1. Функция main

Описание:

Является точкой входа в программу. Открывает файл, выполняет 11 или 12 лабораторную работу.

Прототип:

int main()

Пример вызова:

main()

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Локальная	choice	int	Выбор пользователя

Возвращаемое значение:
0, если работа программы завершена корректно.

2. Функция do_lab11

Описание:

Выполняется 11 лабораторная работа. Считываются данные с файла.

Прототип:

void do_lab11()

Пример вызова:

do_lab11()

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Локальная	slen	int	Длина строки
Локальная	i	int	Счётчик цикла

Локальная	number_of_lines	int	Количество строк в файле
Локальная	s2	char**	Строка с разделённой по символу-разделителю информацией
Локальная	s3	comps*	Информация об 1 детали
Локальная	s1[256]	char	Строка
Локальная	sep	char	Символ-разделитель
Локальная	df	FILE*	Поток
Локальная	head	head*	Голова списка

3. Функция do_lab12

Описание:

Выполняется 12 лабораторная работа. Считываются данные с файла.

Прототип:

void do_lab12()

Пример вызова:

do_lab12()

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Локальная	slen	int	Длина строки
Локальная	i	int	Счётчик цикла
Локальная	number_of_lines	int	Количество строк в файле
Локальная	s2	char**	Строка с разделённой по символу-разделителю информацией
Локальная	s3	comps*	Информация об 1 детали
Локальная	s1[256]	char	Строка
Локальная	sep	char	Символ-разделитель
Локальная	df	FILE*	Поток
Локальная	head2	head*	Голова списка

Локальная	rehead	head*	Голова списка с перестановленными данными

4. Функция simple_split

Описание:

Функция разделяет строки по заданному разделителю. Подсчитывается количество разделителей, по количеству выделяется память двумерного массива и заполняется отдельными элементами.

Прототип:

char **simple_split(char *str, int length, char sep)

Пример вызова:

s2=simple_split(s1,slen,sep)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	str	char*	Строка, которую необходимо разделить
Формальный аргумент	length	int	Длина строки
Формальный аргумент	sep	char	Символ-разделитель
Локальная	str_array	char**	Массив строк
Локальная	i	int	Счётчик цикла
Локальная	j	int	Счётчик цикла
Локальная	k	int	Счётчик цикла
Локальная	m	int	Счётчик строки в массиве строк
Локальная	key	int	Проверка выделения памяти
Локальная	count	int	Проверка выделения памяти

Возвращаемое значение:
Массив строк

5. Функция ClearStringArray

Описание:

Функция очистки памяти для массива строк.

Прототип:

void ClearStringArray(char **str, int n)

Пример вызова:

ClearStringArray(str_array,count)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	str	char**	Массив строк
Формальный аргумент	n	int	Количество строк
Локальная	i	int	Счётчик цикла

6. Функция struct_fill

Описание:

Функция заполнения структуры данными массива строк. Создаёт структуру и по порядку сохраняет данные в неё.

Прототип:

comps *struct_fill(char **str)

Пример вызова:

s3 = struct_fill(s2)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	str	char**	Массив строк

Локальная	str0	comps*	Структура для заполнения

Возвращаемое значение:
Заполненная структура.

7. Функция new_struct

Описание:

Создаёт новую структуру, в которую пользователь вводит данные

Прототип:

comps *new_struct()

Пример вызова:

str0 = new_struct()

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Локальная	str0	comps	Новая структура

Возвращаемое значение:
Заполненная структура.

8. Функция create_node

Описание:

Функция создаёт новый узел связного списка. Отводится память под новую запись, устанавливаются значения полей, переданные в аргументах, ссылка на следующий елемент устанавливается в NULL.

Прототип:

node *create_node(comps *data)

Пример вызова:

p = create_node(s3)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	data	comps*	Структура, хранящаяся в узле
Локальная	temp	node*	Узел списка

Возвращаемое значение:
Возвращает ссылку на созданный узел связного списка.

9. Функция list_out

Описание:

Функция вывода списка. Выводит данные узла до тех пор, пока указатель не равен NULL

Прототип:

void list_out(node **head)

Пример вызова:

list_out(&head)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	head	node**	Указатель на голову списка
Локальная	p	node*	Указатель на текущий узел

10. Функция add_signly

Описание:

Функция добавляет структуру в начало односвязного списка.

Прототип:

void add_signly(head *head, comps *data)

Пример вызова:

add_signly(rehead,temp->data)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	head	node**	Указатель на голову списка
Формальный аргумент	data	comps*	Структура
Локальная	temp	node*	Временный узел

11. Функции add_first/add_last

Описание:

Функции добавляют структуру в начало/конец двусвязного списка.

Прототип:

void add_first(head *head, comps *data)

void add_last(head *head, comps *data);

Пример вызова:

add_first(head,str0)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	head	node**	Указатель на голову списка

Формальный аргумент	data	comps*	Структура
Локальная	temp	node*	Временный узел

12. Функция menu

Описание:

Функция выводит меню для 11 лабораторной работы.

Прототип:

void menu(head *head)

Пример вызова:

menu(head)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	head	head*	Указатель на голову для взаимодействия со списком
Локальная	choice	int	Выбор пользователя
Локальная	str0	comps*	Новая структура

13. Функция delete_nodes

Описание:

Функция удаляет элементы с заданным значением информационного поля, которое выберет пользователь. Сначала проверяется на совпадение со значением информации в голове списка. Если значения равны, текущий узел становится следующим по списку, память головы списка очищается и указатель приравнивается указателю на текущий узел. Это повторяется до тех пор, пока значение в голове равно введённому.

Далее вводится указатель на следующий узел относительно текущему. В цикле с пост-условием проверяется, равно-ли значение в следующем узле введённому значению. Если равно, указатель на следующий узел, хранящийся в текущем узле меняется на указатель на следующий узел, хранящийся в следующем узле.

(p1→next = p→next), где p1 – указатель на текущий узел списка, p – указатель на следующий узел списка, next – указатель на следующий узел списка, хранящийся в узле.

При совпадении значений хотя бы раз, переменная chk принимает значение 1. В конце алгоритма происходит проверка: если значение chk не равно 1, выводится сообщение о том, что введённого значения в данных нет.

Прототип:

void delete_nodes(node **head)

Пример вызова:

delete_nodes(&head)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	head	node**	Указатель на голову для взаимодействия со списком
Локальная	choice	int	Выбор пользователя
Локальная	chk	int	Проверка на присутствие данных для удаления
Локальная	value	float	Значение удаляемых элементов
Локальная	p1	node*	Указатель на текущий узел списка
Локальная	p	node*	Указатель на следующий узел списка

13. Функция menu_for_12

Описание:

Функция выводит меню для 12 лабораторной работы.

Прототип:

void menu_for_12(head *head2, head *rehead)

Пример вызова:

menu_for_12(head2, rehead)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	head2	head*	Указатель на голову для взаимодействия со списком
Формальный аргумент	rehead	head*	Указатель на голову нового списка
Локальная	choice	int	Выбор пользователя
Локальная	str0	comps*	Новая структура

14. Функция reshape_list

Описание:

Функция переставляет элементы списка. Указатель доходит до середины списка. Если количество элементов нечётное, элемент сохраняется, а указатель переходит к следующему элементу. Список полностью копируется. Если количество элементов нечётное, копирование разделяется на две части и между ними отдельно копируется сохранённый элемент из середины первоначального списка.

Прототип:

void reshape_list(head *head2, head *rehead)

Пример вызова:

reshape_list(head2,rehead)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	head2	head*	Указатель на голову для взаимодействия со списком
Формальный аргумент	rehead	head*	Указатель на голову нового списка
Локальная	center	node*	Элемент в середине списка

Локальная	temp	node*	Копируемый элемент
Локальная	i	int	Счётчик цикла
Локальная	half	int	Половина от количества элементов в списке

15. Функции free_head/free_node

Описание:

Функции отчищают голову и элементы списка.

Прототип:

void free_head(head *q)

void free_node(node *temp)

Пример вызова:

free_head(head2)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	q	head*	Указатель на голову списка
Формальный аргумент	temp	node*	Указатель на элемент списка

16. Функции free_list/free_nodes_only

Описание:

Функции отчищают список или только элементы списка.

Прототип:

void free_list(head *q)

void free_nodes_only(head *q)

Пример вызова:

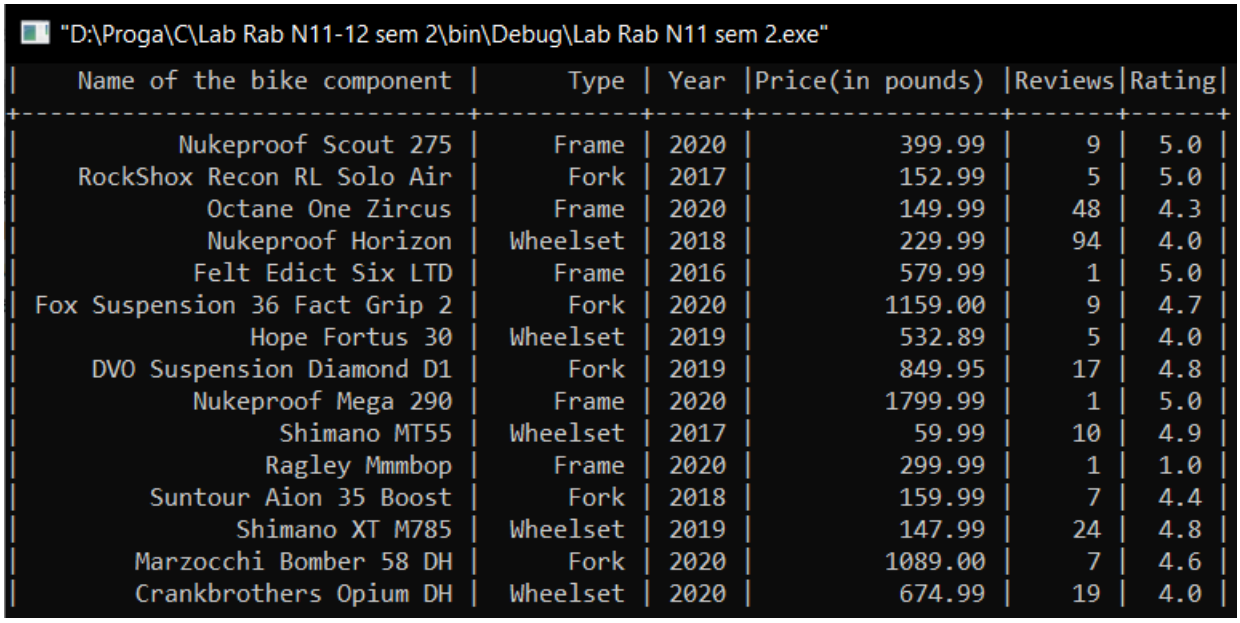
free_nodes_only(rehead)

Описание переменных:

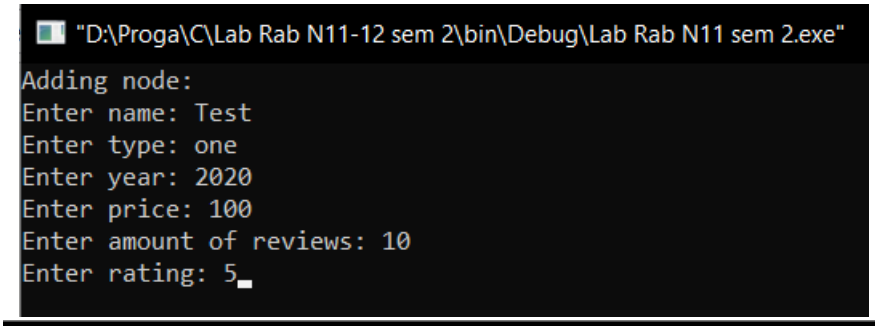
Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	q	head*	Указатель на голову списка
Локальная	temp	node*	Указатель на элемент списка
Локальная	p	node*	Указатель на элемент списка
Локальная	i	int	Счётчик цикла
Локальная	n	int	Количество элементов

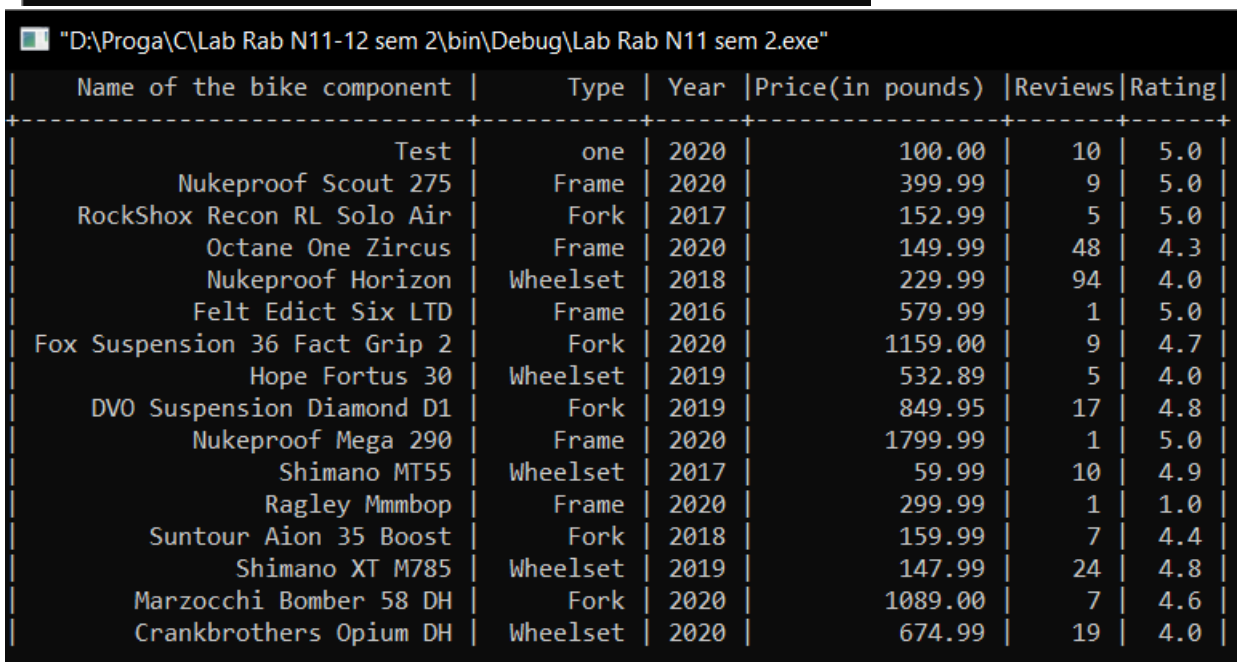
Контрольные примеры

Исходные данные:

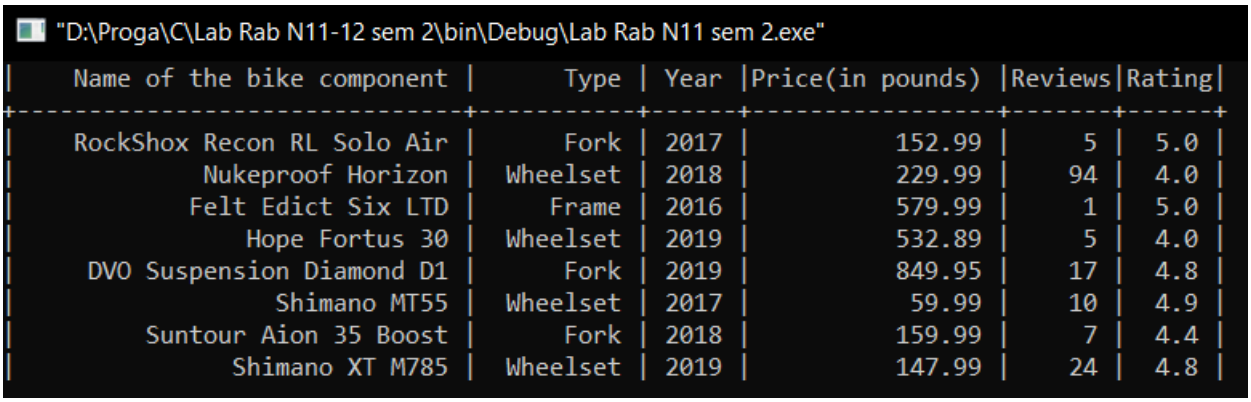


Добавление данных:

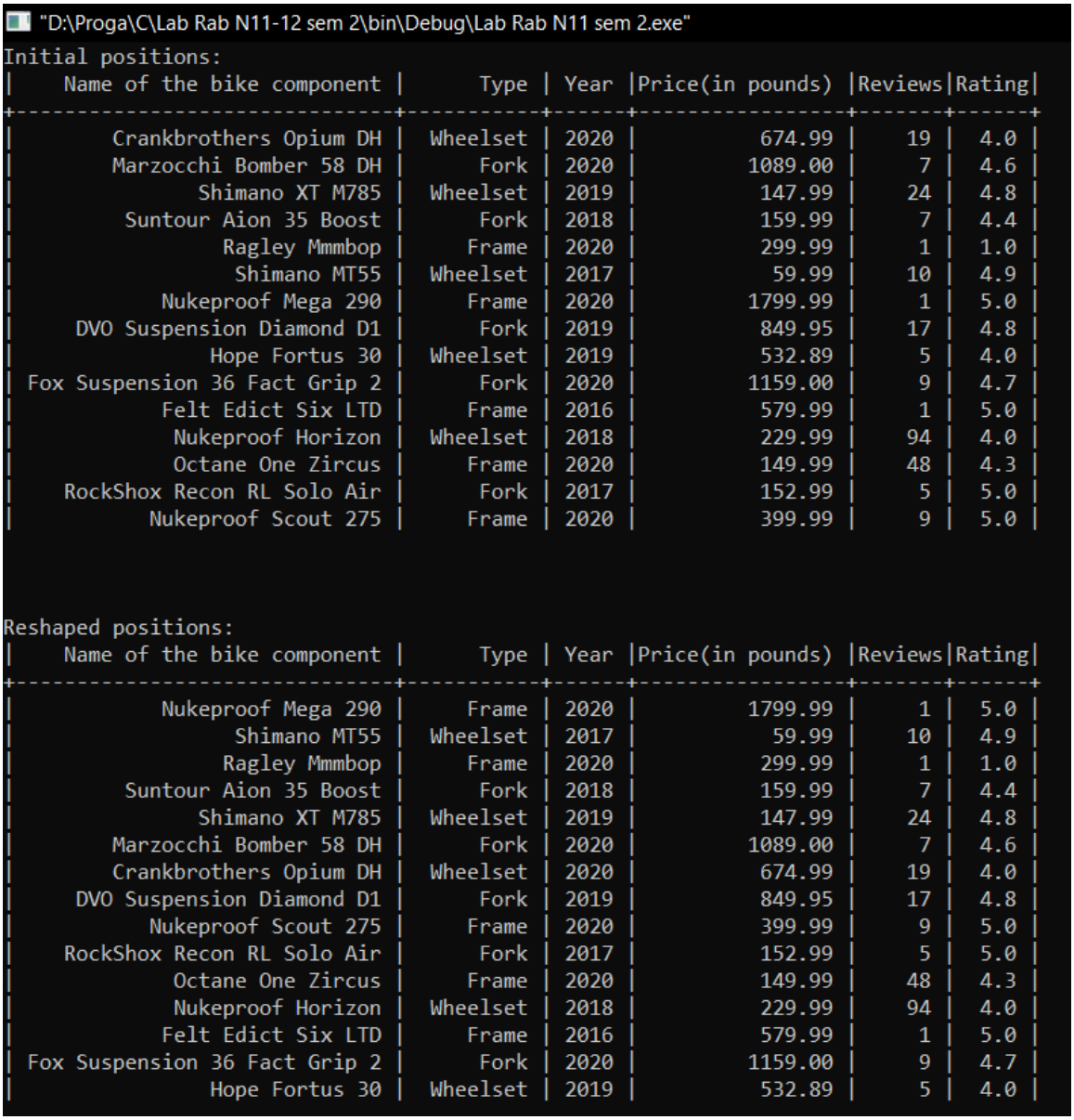




Удаление данных:

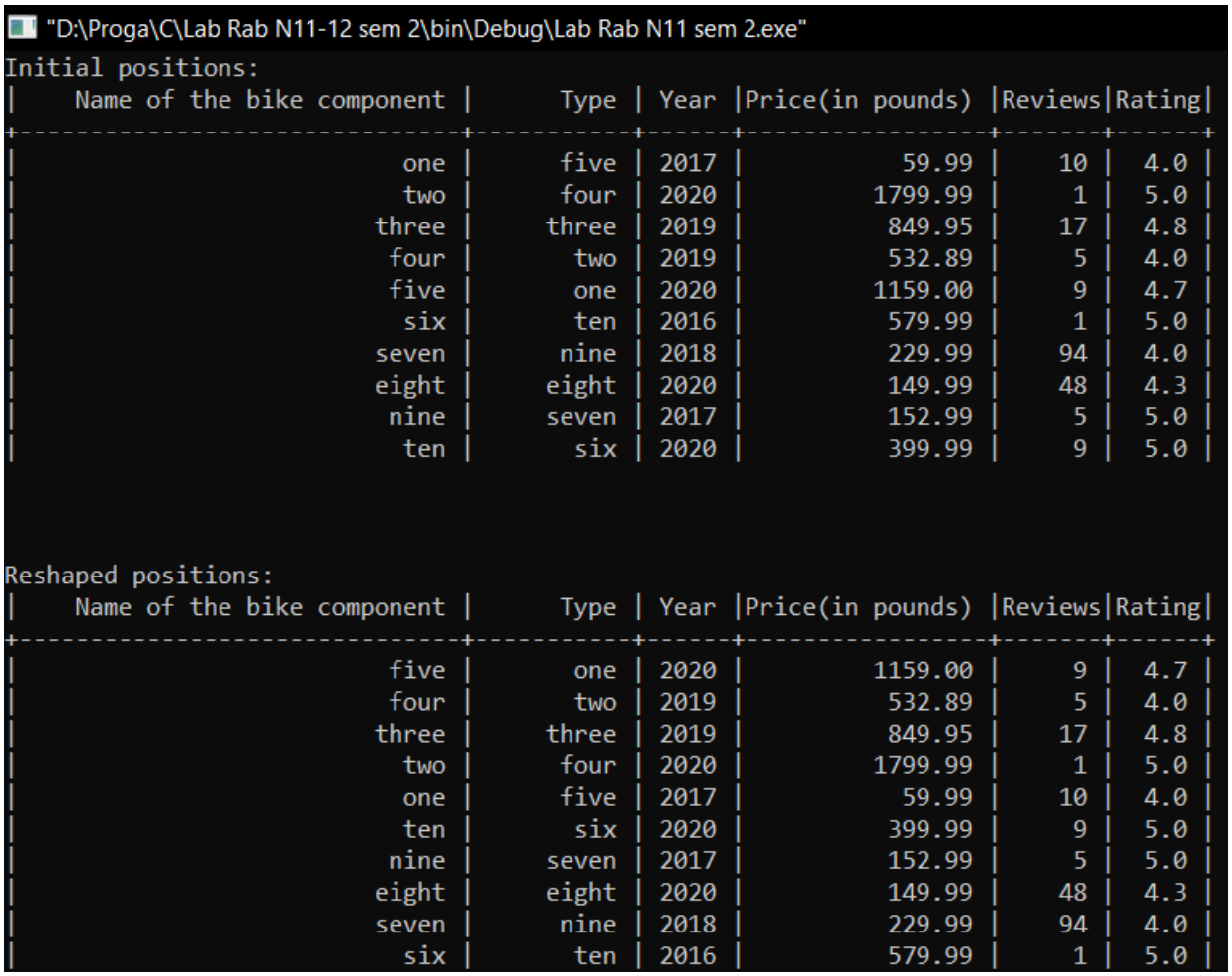


Перестановка данных:



Перестановка более лёгких для проверки данных:

(Первый столбец — изначальный порядок, второй столбец — порядковый номер элемента после перестановки)



Текст программы

main.c

```
#include "lab11.h"
#include "lab12.h"



int main()
{
    int choice;

    do
    {
        printf("| |Chose lab to run|\n");
        printf("+-+----------------+\n");
        printf("|1| - Lab N 11     |\n");
        printf("|2| - Lab N 12     |\n");
        printf("|0| - Exit         |\n");
        printf("Your choice: ");
        scanf("%d", &choice);
        getchar();

        switch(choice)
        {
        case 1:
            system("cls");
            do_lab11();
            break;
        case 2:
            system("cls");
            do_lab12();
            break;
        case 0:

            break;
        default:
            system("cls");
            puts("Incorrect input!");
            break;
        }

    }while(choice!=0);

    return 0;
}
```

lab11.h

```
#ifndef LAB11_H_INCLUDED
#define LAB11_H_INCLUDED
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "struct_csv.h"
#include "list.h"
#include "add_and_delete.h"
#include "clean_list.h"


void do_lab11();
void menu(head *head);


#endif // LAB11_H_INCLUDED
```

lab11.c

```
#include "lab11.h"


void do_lab11()
{
    int slen,i,number_of_lines;
    char **s2=NULL;
    comps *s3=NULL;
    char s1[256];
    char sep;
    FILE *df;
    head *head = NULL;

    head = create_head();
    sep=';';

    df=fopen("struct-data.csv","r");
    if(df!=NULL)
    {
        number_of_lines=0;
        while((fgets(s1,256,df))!=NULL) number_of_lines++;
        rewind(df);
        fgets(s1,256,df);
        slen=strlen(s1);
        s1[slen-1]='\0';
        slen=strlen(s1);

        s2=simple_split(s1,slen,sep);
        s3 = struct_fill(s2);
```

```
        create_node(s3,head); //Creating first node

      //Splitting other strings, creating//
      //structures and making list      //
        for(i=0;i<number_of_lines-1;i++)
        {
            fgets(s1,256,df);
            slen=strlen(s1);
            s1[slen-1]='\0';
            slen=strlen(s1);

            s2=simple_split(s1,slen,sep);
            s3 = struct_fill(s2);       //Adding node in the
            add_last(head,s3);          //end of the list
        }
        if (fclose(df)==EOF) printf ("Error with closing
file!");

      //Printing menu, where all other
      //main functions is being used
        menu(head);

      //Clearing memory
        if((head->cnt) == 1) free_head(head);
        else if((head->cnt) > 1) free_list(head);
        else if ((head->cnt) == 0) free(head);


    }
    else puts("File not found!");
}


void menu(head *head)
{
    int choice;

    do
    {
        printf("| |    Lab N 11   |\n");
        printf("+-+---------------+\n");
        printf("|1| - Show list   |\n");
        printf("|2| - Add node    |\n");
        printf("|3| - Delete nodes|\n");
        printf("|0| - Exit        |\n");
        printf("Your choice: ");
        scanf("%d", &choice);
        getchar();

        switch(choice)
        {
```

```
        case 1:     //Shows list as a table
            system("cls");
            print_header();
            list_out(head);
            break;
        case 2:     //User inputs data for new node
            system("cls");
            add_info(head);
            break;
        case 3:
            system("cls");
            delete_nodes(head);
            break;
        case 0:
            system("cls");
            break;
        default:
            system("cls");
            puts("Incorrect input!");
            break;
        }

    } while(choice!=0);
}
```

lab12.h

```
#ifndef LAB12_H_INCLUDED
#define LAB12_H_INCLUDED
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "struct_csv.h"
#include "list.h"
#include "add_and_delete.h"
#include "clean_list.h"


void do_lab12();
void menu_for_12(head *head2, head *rehead);
void reshape_list(head *head2, head *rehead);


#endif // LAB12_H_INCLUDED
```

lab12.c

```
#include "lab12.h"


void do_lab12()
{
    int slen,i,number_of_lines, c;
    char **s2=NULL;
    comps *s3=NULL;
    char s1[256];
    char sep;
    FILE *df;
    head *head2 = NULL;
    head *rehead = NULL;

    head2 = create_head();
    rehead = create_head();
    sep=';';

    printf("Make data easy to check?\n");
    printf("[1] - Yes\n");
    printf("[2] - No\n");
    scanf("%d",&c);
    //Uses simple data with numbers instead of names.
    //First column is the initial order
    //Second column is the reshaped order
    if (c==1) df=fopen("simple-data.csv","r");
    else df=fopen("struct-data.csv","r");
    //df=fopen("struct-data.csv","r");
    if(df!=NULL)
    {
        number_of_lines=0;
        while((fgets(s1,256,df))!=NULL) number_of_lines++;
        rewind(df);
        fgets(s1,256,df);
        slen=strlen(s1);
        s1[slen-1]='\0';
        slen=strlen(s1);

        s2=simple_split(s1,slen,sep);
        s3 = struct_fill(s2);
        create_node(s3,head2); //Creating first node

      //Splitting other strings, creating//
      //structures and making list      //
        for(i=0;i<number_of_lines-1;i++)
        {
            fgets(s1,256,df);
            slen=strlen(s1);
            s1[slen-1]='\0';
```

```
            slen=strlen(s1);

            s2=simple_split(s1,slen,sep);
            s3 = struct_fill(s2);
            add_signly(head2,s3);        //Adding node
(circularize too!)
        }
        if (fclose(df)==EOF) printf ("Error with closing
file!");

      //Printing menu for 12th lab
        menu_for_12(head2, rehead);

      //Clearing memory
        if((head2->cnt) == 1) free_head(head2);
        else if((head2->cnt) > 1) free_list(head2);
        else if ((head2->cnt) == 0) free(head2);


    }
    else puts("File not found!");
}

void menu_for_12(head *head2, head *rehead)
{
    int choice;
    comps *str0=NULL;

    do
    {
        printf("| |    Lab N 12   |\n");
        printf("+-+---------------+\n");
        printf("|1| - Reshape     |\n");
        printf("|2| - Add node    |\n");
        printf("|0| - Exit        |\n");
        printf("Your choice: ");
        scanf("%d", &choice);
        getchar();

        switch(choice)
        {
        case 1:
            system("cls");
            printf("Initial positions:\n");
            print_header();
            list_out(head2);

            reshape_list(head2,rehead);

            printf("\n\nReshaped positions:\n");
            print_header();
```

```
            list_out(rehead);
            free_nodes_only(rehead); //Clear to make program
reusable
            break;
        case 2:
            system("cls");
            printf("Adding node:\n");
            str0 = new_struct();
            add_signly(head2,str0);
            system("cls");
            printf("Node added\n");
            break;
        case 0:
            system("cls");
            break;
        default:
            system("cls");
            puts("Incorrect input!");
            break;
        }

    } while(choice!=0);
}


void reshape_list(head *head2,head *rehead)
{
    node *center=NULL;
    node *temp=NULL;
    int i, half;

    temp = head2->first;

    if(head2->cnt%2==0)
    {
        half = head2->cnt/2;                        //Example:
(1,2,3,4,5,6)
        for(i=0;i<half;i++) temp = temp->next;      //temp->data
== 4
        create_node(temp->data,rehead);             //First node
== 4
        for(i=1;i<head2->cnt;i++)                   //Makes list
(4,5,6,1,2,3)
        {                                           //So output
is (3,2,1,6,5,4) - correct!
            temp = temp->next;
            add_signly(rehead,temp->data);
        }
    }
    else
    {
```

```
        half = head2->cnt/2;                            //Example:
(1,2,3,4,5)
        for(i=0;i<half;i++) temp = temp->next;      //temp->data
== 3
        center = temp;                                  //Center
saves (its 3)
        temp = temp->next;                              //temp->data
== 4
        create_node(temp->data,rehead);             //First node
== 4
        for(i=1;i<half;i++)                             //Makes
first half of the list:
        {                                               //(4,5)
            temp = temp->next;
            add_signly(rehead,temp->data);
        }
        add_signly(rehead,center->data);            //Adds
center (4,5,3)
        for(i=0;i<half;i++)                             //Makes
second half of the list:
        {
//(4,5,3,1,2)
            temp = temp->next;                      //So output
is (2,1,3,5,4) - correct!
            add_signly(rehead,temp->data);
        }
    }


}
```

structs.h

```
#ifndef STRUCTS_H_INCLUDED
#define STRUCTS_H_INCLUDED

struct bike_components {
    char *name;
    char *type;
    int year;
    float cost;
    int reviews;
    float rating;
};
typedef struct bike_components comps;


struct node{

    comps *data;
```

```
    struct node *next;
    struct node *prev;
};
typedef struct node node;


struct head {
    int cnt;
    struct node *first;
    struct node *last;
};
typedef struct head head;


#endif // STRUCTS_H_INCLUDED
```

struct_csv.h

```
#ifndef STRUCT_CSV_H_INCLUDED
#define STRUCT_CSV_H_INCLUDED
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "structs.h"


char **simple_split(char *str, int length, char sep);
void ClearStringArray(char **str, int n);
comps *struct_fill(char **str);
comps *new_struct();
void print_header();


#endif // STRUCT_CSV_H_INCLUDED
```

struct_csv.c

```
#include "struct_csv.h"



char **simple_split(char *str, int length, char sep)
{
    char **str_array=NULL;
    int i,j,k,m;
    int key,count;
    for(j=0,m=0;j<length;j++)
    {
        if(str[j]==sep) m++;
```

```
    }

    key=0;
    str_array=(char**)malloc((m+1)*sizeof(char*));
    if(str_array!=NULL)
    {
        for(i=0,count=0;i<=m;i++,count++)
        {
            str_array[i]=(char*)malloc(length*sizeof(char));
            if(str_array[i]!=NULL) key=1;
            else
            {
                key=0;
                i=m;
            }
        }
        if(key)
        {
            k=0;
            m=0;
            for(j=0;j<length;j++)
            {
                if(str[j]!=sep) str_array[m][j-k]=str[j];
                else
                {
                    str_array[m][j-k]='\0';
                    k=j+1;
                    m++;
                }
            }
        }
        else
        {
            ClearStringArray(str_array,count);
        }
     }
     return str_array;
}


void ClearStringArray(char **str, int n)
{
    int i;

    for(i=0;i<n;i++)
    {
        free(str[i]);
        str[i]=NULL;
    }
    free(str);
    str=NULL;
```

```
}


comps *struct_fill(char **str)
{
    comps *str0=NULL;

    str0=(comps*)malloc(sizeof(comps));
    if(str0!=NULL)
    {
        str0->name=str[0];
        str0->type=str[1];
        str0->year=atoi(str[2]);
        str0->cost=atof(str[3]);
        str0->reviews=atoi(str[4]);
        str0->rating=atof(str[5]);
    }
    return str0;
}

comps *new_struct()
{
    comps *str0=NULL;

    str0=(comps*)malloc(sizeof(comps));
    str0 -> name = (char*)malloc(sizeof(char)*32);
    str0 -> type = (char*)malloc(sizeof(char)*32);
    if(str0!=NULL)
    {
        printf("Enter name: ");
        fgets((*str0).name,64,stdin);
        printf("Enter type: ");
        fgets((*str0).type,32,stdin);
        printf("Enter year: ");
        scanf("%d",&(*str0).year);
        printf("Enter price: ");
        scanf("%f",&(*str0).cost);
        printf("Enter amount of reviews: ");
        scanf("%d",&(*str0).reviews);
        printf("Enter rating: ");
        scanf("%f",&(*str0).rating);
        str0->name[strlen(str0->name)-1]='\0';
        str0->type[strlen(str0->type)-1]='\0';
    }
    return str0;
}


void print_header()
{
```

```
    printf("|%30s |%10s |%5s |%9s |%4s|%5s|\n","Name of the bike
component","Type","Year","Price(in pounds)","Reviews","Rating");
    printf("+-------------------------------+-----------+------
+-----------------+-------+------+\n");
}
```

list.h

```
#ifndef LIST_H_INCLUDED
#define LIST_H_INCLUDED
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "structs.h"
#include "struct_csv.h"
#include "add_and_delete.h"


head *create_head();
void create_node(comps *data, head *head);
void list_out(head *head);

#endif // LIST_H_INCLUDED
```

list.c

```
#include "list.h"


head *create_head()
{
    head *ph=NULL;

    ph=(head*)malloc(sizeof(head));
    if(ph)
    {
        ph->cnt=0;
        ph->first=NULL;
        ph->last=NULL;
    }
    return ph;
}



void create_node(comps *data, head *head)
{
      node *temp=NULL;
      temp = (node *)malloc(sizeof(node));
```

```
	temp->data = (comps*)malloc(sizeof(comps));

	temp->next = NULL;
	temp->prev = NULL;
	head->first = temp;
	head->last = temp;
	head->cnt += 1;
	temp->data = data;
}


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

	p = head->first;
	//while(p != NULL){  //Isn't used because of circular list
	for(i=0;i<head->cnt;i++){
		printf("|%30s |%10s |%5d |%16.2f |%6d |%5.1f |\n",
		p -> data -> name,
		p -> data -> type,
		p -> data -> year,
		p -> data -> cost,
		p -> data -> reviews,
		p -> data -> rating);
		p = p->next;
	}
	printf("\n");
}
```

add_and_delete.h

```
#ifndef ADD_AND_DELETE_H_INCLUDED
#define ADD_AND_DELETE_H_INCLUDED
#include <stdio.h>
#include <stdlib.h>
#include "structs.h"
#include "struct_csv.h"


void add_signly(head *head, comps *data);
void add_info(head *head);
void add_first(head *head, comps *data);
void add_last(head *head, comps *data);
void delete_nodes(head *head);

#endif // ADD_AND_DELETE_H_INCLUDED
```

add_and_delete.c

```
#include "add_and_delete.h"

//Used for SINGLY CURCULAR list
void add_signly(head *head, comps *data)
{
    node *temp;

    temp = (node *)malloc(sizeof(node));
    temp->data = (comps*)malloc(sizeof(comps));
    temp->next = head->first;
    head->first = temp;
    head->cnt += 1;
    temp->data = data;
    head->last->next = temp; //Making circular list
}


void add_info(head *head)
{
    int choice;
    comps *str0=NULL;

    do
    {
        printf("| |Position to add|\n");
        printf("+-+---------------+\n");
        printf("|1| - First       |\n");
        printf("|2| - Last        |\n");
        printf("|0| - Exit        |\n");
        printf("Your choice: ");
        scanf("%d", &choice);
        getchar();

        switch(choice)
        {
        case 1:
            system("cls");
            printf("Adding node:\n");
            str0 = new_struct();
            add_first(head,str0);
            system("cls");
            printf("Added the first\n");
            choice = 0;
            break;
        case 2:     //User inputs data for new node
            system("cls");
            printf("Adding node:\n");
            str0 = new_struct();
            add_last(head,str0);
```

```
            system("cls");
            printf("Added the last\n");
            choice = 0;
            break;
        case 0:

            break;
        default:
            system("cls");
            puts("Incorrect input!");
            break;
        }

    } while(choice!=0);

}


void add_first(head *head, comps *data)
{
    node *temp;

    temp = (node *)malloc(sizeof(node));
    temp->data = (comps*)malloc(sizeof(comps));
    temp->prev = NULL;
    temp->next = head->first;
    head->first->prev = temp;
    head->first = temp;
    head->cnt += 1;
    temp->data = data;
}


void add_last(head *head, comps *data)
{
    node *temp;

    temp = (node *)malloc(sizeof(node));
    temp->data = (comps*)malloc(sizeof(comps));
    temp->next = NULL;
    temp->prev = head->last;
    head->last->next = temp;
    head->last = temp;
    head->cnt += 1;
    temp->data = data;
}


void delete_nodes(head *head)
{
    int choice, chk;
```

```
float value;
node *p, *p1;


do
{
    printf("| | Choose field: |\n");
    printf("+-+---------------+\n");
    printf("|1| - Year        |\n");
    printf("|2| - Price       |\n");
    printf("|3| - Reviews     |\n");
    printf("|4| - Rating      |\n");
    printf("|0| - Back        |\n");
    printf("Your choice: ");
    scanf("%d", &choice);
    if(choice!=0)
    {
    printf("Enter value: ");
    scanf("%f", &value);
    }
    p1 = head->first;
    chk = 0;        //Check for appropriate data
    switch(choice)
    {
    case 1:
      //Deleting head until data isn't appropriate//
        while(p1->data->year==value)
        {
            p = p1->next;
            free(p1);
            p1 = p;
            p1->prev = NULL;
            head->first = p1;
            head->cnt -= 1;
            chk = 1;
        }

      //Goes through list and delete nodes//
      //With appropriate data             //
        do
        {
            p = p1->next;
            if(p->data->year==value)
            {
                chk = 1;
                p1->next = p->next;
                if(p->next!=NULL) p->next->prev = p1;
                else head->last = p1;
                head->cnt -= 1;
                free(p);
            }
```

```
            else
            {
                p1 = p1 -> next;
            }
        }while (p1->next!=NULL);
        if (chk==0) printf("There is no such value\n\n");
        else
        {
            system("cls");
            printf("Nodes have been deleted\n");
            choice = 0;
        }

        break;
    case 2:
      //Deleting head until data isn't appropriate//
        while(p1->data->cost==value)
        {
            p = p1->next;
            free(p1);
            p1 = p;
            p1->prev = NULL;
            head->first = p1;
            head->cnt -= 1;
            chk = 1;
        }

      //Goes through list and delete nodes//
      //With appropriate data             //
        do
        {
            p = p1->next;
            if(p->data->cost==value)
            {
                chk = 1;
                p1->next = p->next;
                if(p->next!=NULL) p->next->prev = p1;
                else head->last = p1;
                head->cnt -= 1;

                free(p);
            }
            else
            {
                p1 = p1 -> next;
            }
        }while (p1->next!=NULL);
        if (chk==0) printf("There is no such value\n\n");
        else
        {
            system("cls");
```

```
            printf("Nodes have been deleted\n");
            choice = 0;
        }
        break;
    case 3:
      //Deleting head until data isn't appropriate//
        while(p1->data->reviews==value)
        {
            p = p1->next;
            free(p1);
            p1 = p;
            p1->prev = NULL;
            head->first = p1;
            head->cnt -= 1;
            chk = 1;
        }

      //Goes through list and delete nodes//
      //With appropriate data             //
        do
        {
            p = p1->next;
            if(p->data->reviews==value)
            {
                chk = 1;
                p1->next = p->next;
                if(p->next!=NULL) p->next->prev = p1;
                else head->last = p1;
                head->cnt -= 1;

                free(p);
            }
            else
            {
                p1 = p1 -> next;
            }
        }while (p1->next!=NULL);
        if (chk==0) printf("There is no such value\n\n");
        else
        {
            system("cls");
            printf("Nodes have been deleted\n");
            choice = 0;
        }
        break;
    case 4:
      //Deleting head until data isn't appropriate//
        while(p1->data->rating==value)
        {
            p = p1->next;
            free(p1);
```

```
                p1 = p;
                p1->prev = NULL;
                head->first = p1;
                head->cnt -= 1;
                chk = 1;
            }

          //Goes through list and delete nodes//
          //With appropriate data             //
            do
            {
                p = p1->next;
                if(p->data->rating==value)
                {
                    chk = 1;
                    p1->next = p->next;
                    if(p->next!=NULL) p->next->prev = p1;
                    else head->last = p1;
                    head->cnt -= 1;

                    free(p);
                }
                else
                {
                    p1 = p1 -> next;
                }
            }while (p1->next!=NULL);
            if (chk==0) printf("There is no such value\n\n");
            else
            {
                system("cls");
                printf("Nodes have been deleted\n");
                choice = 0;
            }
        case 0:

            break;
        default:
            system("cls");
            puts("Incorrect input!");
            break;
        }
    }while(choice!=0);

}
```

clean_list.h

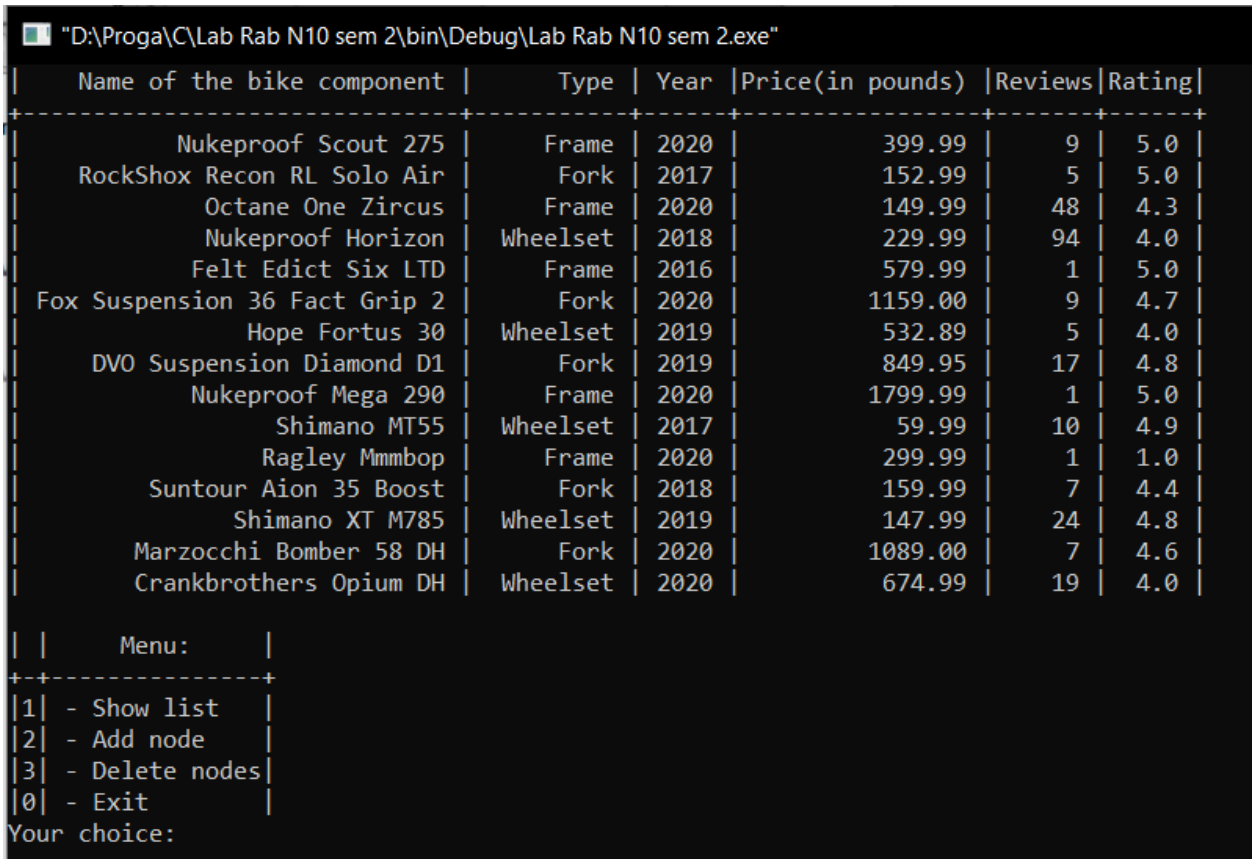
```
#ifndef CLEAN_LIST_H_INCLUDED
#define CLEAN_LIST_H_INCLUDED
#include <stdlib.h>
#include "structs.h"


void free_head(head *q);
void free_node(node *temp);
void free_list(head *q);
void free_nodes_only(head *q);

#endif // CLEAN_LIST_H_INCLUDED
```

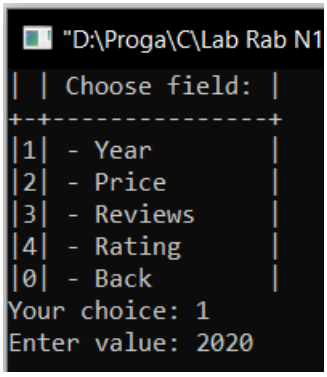
clean_list.c

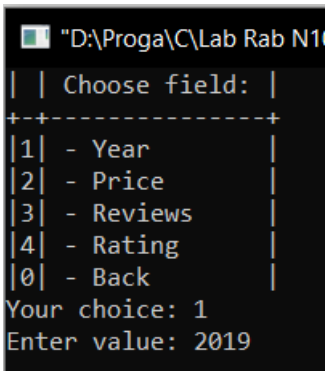
```
#include "clean_list.h"


void free_head(head *q){
    free_node(q->first);
    q->first = NULL;
    q->last = NULL;
    free(q);
}


void free_node(node *temp){
    if((temp->next) != NULL) temp->next = NULL;


    free(temp->data->name);
    free(temp->data->type);
    free(temp->data);
    free(temp);
}


void free_list(head *q){

    node *temp = NULL;
    node *p = NULL;
    int i,n;

    temp = q->first->next;
    n = q->cnt;
    free_head(q);

    //while((temp->next) != NULL){
    for(i=0;i<n-2;i++){
```

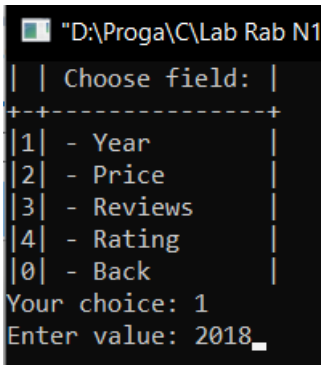
```
        p = temp;
        temp = temp -> next;
        free_node(p);
    }

    free_node(temp);
}


void free_nodes_only(head *q){

    node *temp = NULL;
    node *p = NULL;
    int i,n;

    temp = q->first->next;
    n = q->cnt;
    q->cnt = 0;


    for(i=0;i<n;i++){
        p = temp;
        temp = temp -> next;
        free_node(p);
    }

    free_node(temp);
   // q->first=NULL;
   // q->last=NULL;
}
```

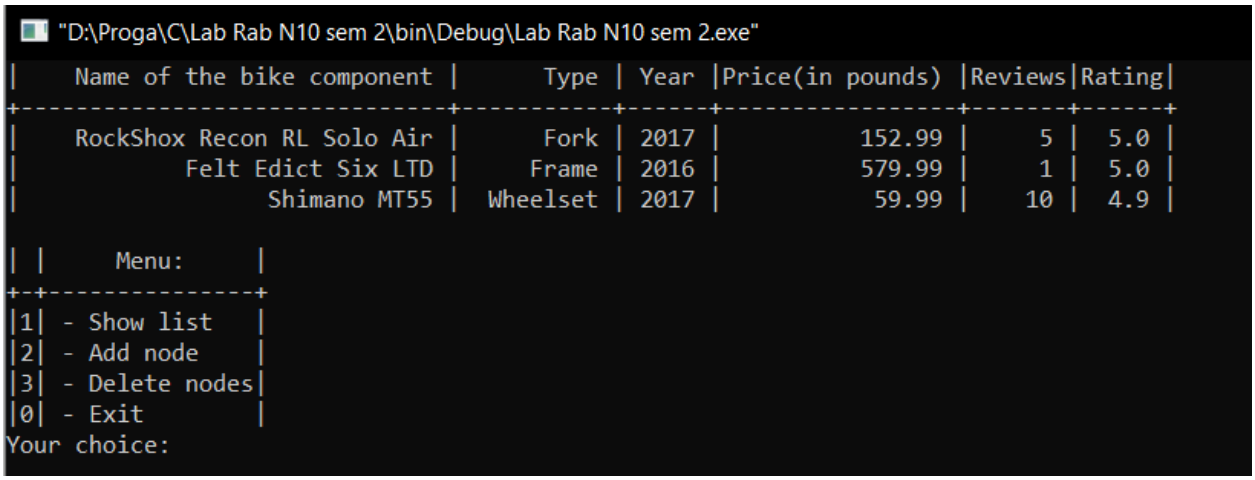
Пример работы программы

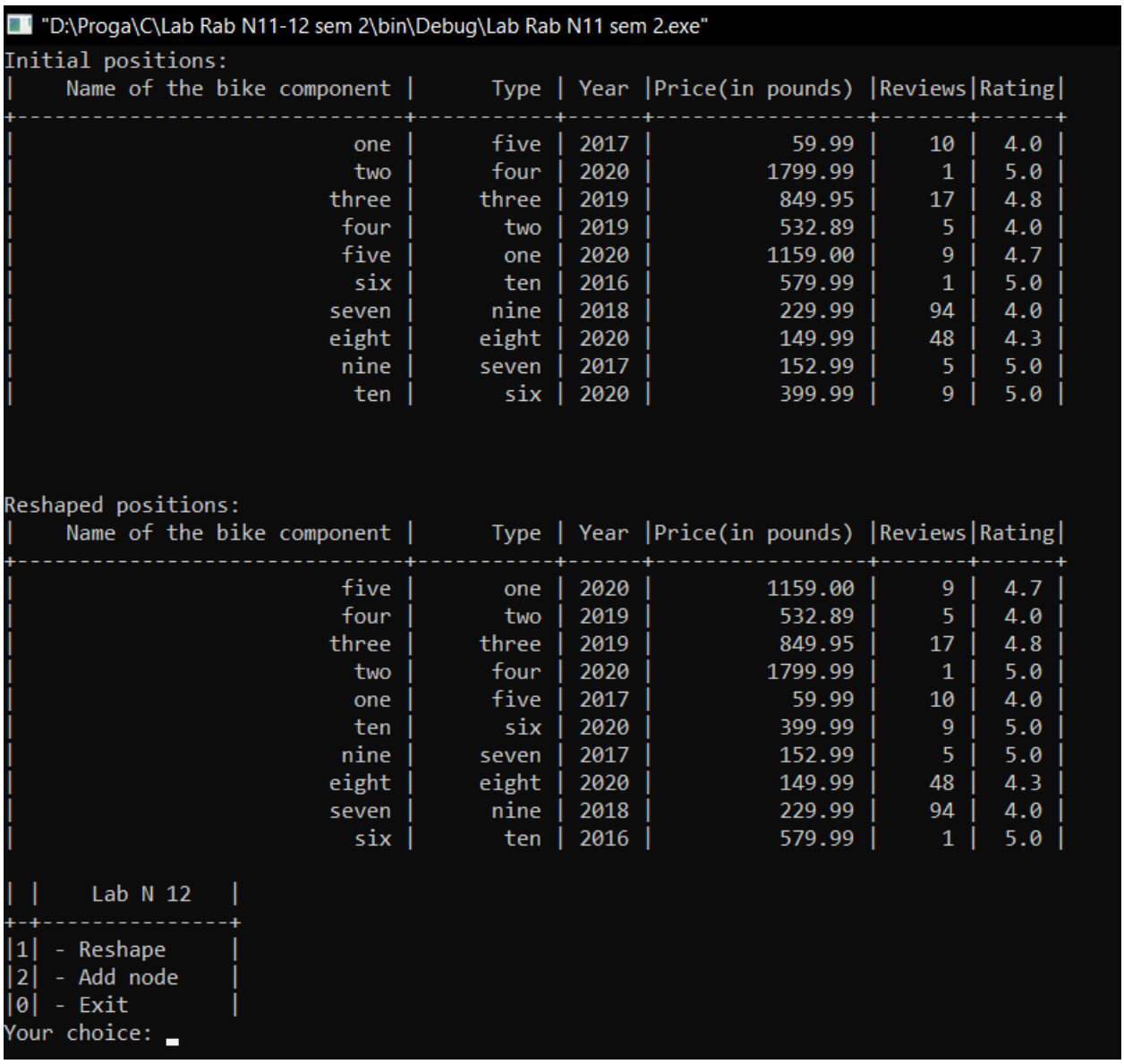











"D:\Proga\C\Lab Rab N11-12 sem 2\bin\Debug\Lab Rab N11 sem 2.exe"
Initial positions:
Name of the bike component	Type	Year	Price(in pounds)	Reviews	Rating
one	five	2017	59.99	10	4.0
two	four	2020	1799.99	1	5.0
three	three	2019	849.95	17	4.8
four	two	2019	532.89	5	4.0
five	one	2020	1159.00	9	4.7
six	ten	2016	579.99	1	5.0
seven	nine	2018	229.99	94	4.0
eight	eight	2020	149.99	48	4.3
nine	seven	2017	152.99	5	5.0
ten	six	2020	399.99	9	5.0
Reshaped positions:
Name of the bike component	Type	Year	Price(in pounds)	Reviews	Rating
five	one	2020	1159.00	9	4.7
four	two	2019	532.89	5	4.0
three	three	2019	849.95	17	4.8
two	four	2020	1799.99	1	5.0
one	five	2017	59.99	10	4.0
ten	six	2020	399.99	9	5.0
nine	seven	2017	152.99	5	5.0
eight	eight	2020	149.99	48	4.3
seven	nine	2018	229.99	94	4.0
six	ten	2016	579.99	1	5.0
	Lab N 12
1	- Reshape
2	- Add node
0	- Exit
Your choice:

Заключение

Выводы:

При выполнении лабораторной работы были получены практические навыки в разработке алгоритма и написании программы на языке Си. Были получены основные знания об абстрактных типах данных, в частности, о работе со списками, а также о разделении программы на заголовочные файлы на языке С.