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

Кафедра вычислительной техники

Отчет по лабораторной работе № 10
по дисциплине «Программирование»
Тема: **ЛИНЕЙНЫЕ ОДНОСВЯЗНЫЕ 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 односвязными списками, а также правилами написания кода на языке Си.

Задание

С использованием структуры, созданной при выполнении лабораторной работы №9, создать односвязный линейный список.

Разработать подалгоритм создания односвязного списка из имеющегося односвязного 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.

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 односвязный 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.

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.

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

Функции

1. Функция main

Описание:

Является точкой входа в программу. Открывает файл, считывает данные.

Прототип:

int main()

Пример вызова:

main()

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	node*	Голова списка
Локальная	temp	node*	Очередной элемент списка
Локальная	p	node*	Временный элемент

Возвращаемое значение:
0, если работа программы завершена корректно.

2. 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

3. 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

4. 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.

5. 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.

6. 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.

7. 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

8. Функция add

Описание:

Функция добавляет структуру в начало списка.

Прототип:

void add(node **head, comps *data)

Пример вызова:

add(&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

9. Функция menu

Описание:

Функция выводит меню

Прототип:

void menu(node *head)

Пример вызова:

menu(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	str0	comps*	Новая структура

10. 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:

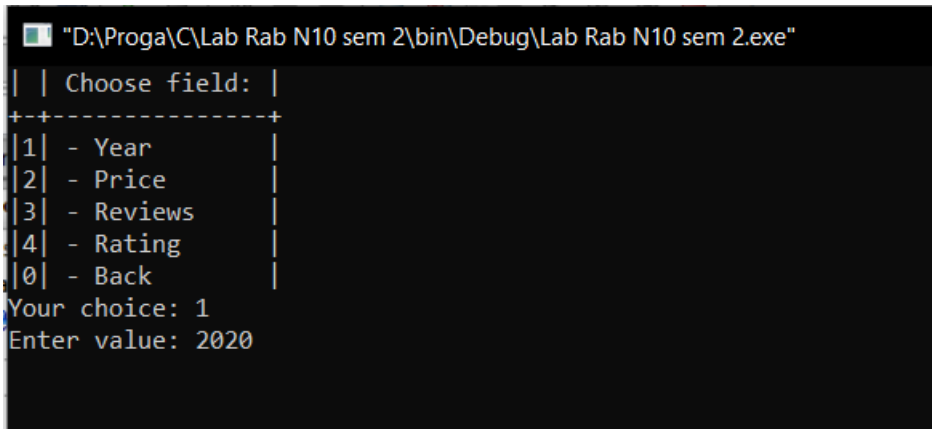
Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	head	node**	Указатель на голову для взаимодействия со списком
Локальная	choice	int	Выбор пользователя
Локальная	chk	int	Проверка на присутствие данных для удаления
Локальная	value	float	Значение удаляемых элементов
Локальная	p1	node*	Указатель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

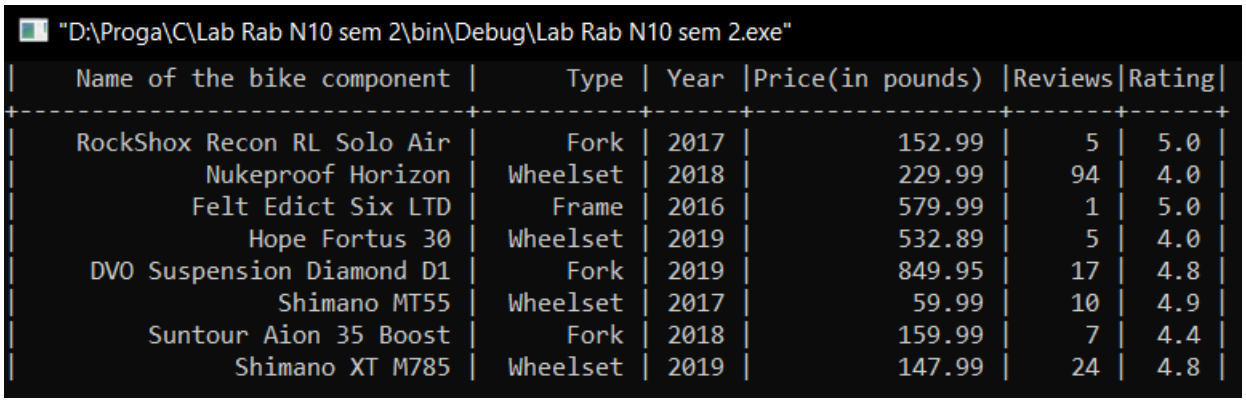
Контрольные примеры

Исходные данные:

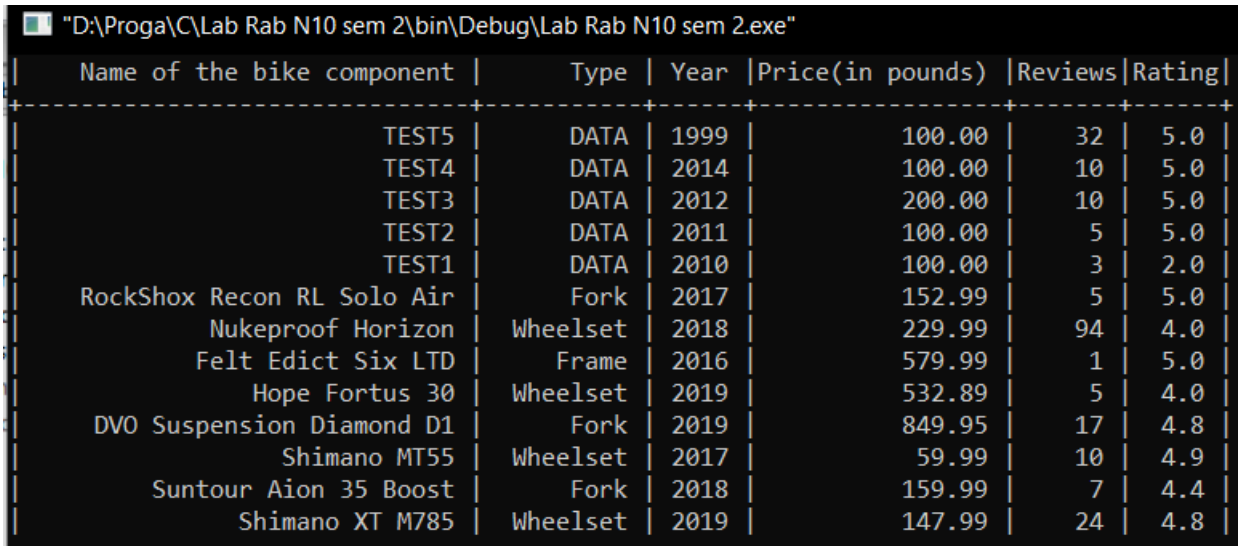
Name of the bike component	Type	Year	Price(in pounds)	Reviews	Rating
Nukeproof Scout 275	Frame	2020	399.99	9	5.0
RockShox Recon RL Solo Air	Fork	2017	152.99	5	5.0
Octane One Zircus	Frame	2020	149.99	48	4.3
Nukeproof Horizon	Wheelset	2018	229.99	94	4.0
Felt Edict Six LTD	Frame	2016	579.99	1	5.0
Fox Suspension 36 Fact Grip 2	Fork	2020	1159.00	9	4.7
Hope Fortus 30	Wheelset	2019	532.89	5	4.0
DVO Suspension Diamond D1	Fork	2019	849.95	17	4.8
Nukeproof Mega 290	Frame	2020	1799.99	1	5.0
Shimano MT55	Wheelset	2017	59.99	10	4.9
Ragley Mmmbop	Frame	2020	299.99	1	1.0
Suntour Aion 35 Boost	Fork	2018	159.99	7	4.4
Shimano XT M785	Wheelset	2019	147.99	24	4.8
Marzocchi Bomber 58 DH	Fork	2020	1089.00	7	4.6
Crankbrothers Opium DH	Wheelset	2020	674.99	19	4.0

При вводе в поле year значения 2020:

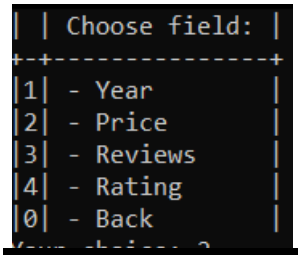


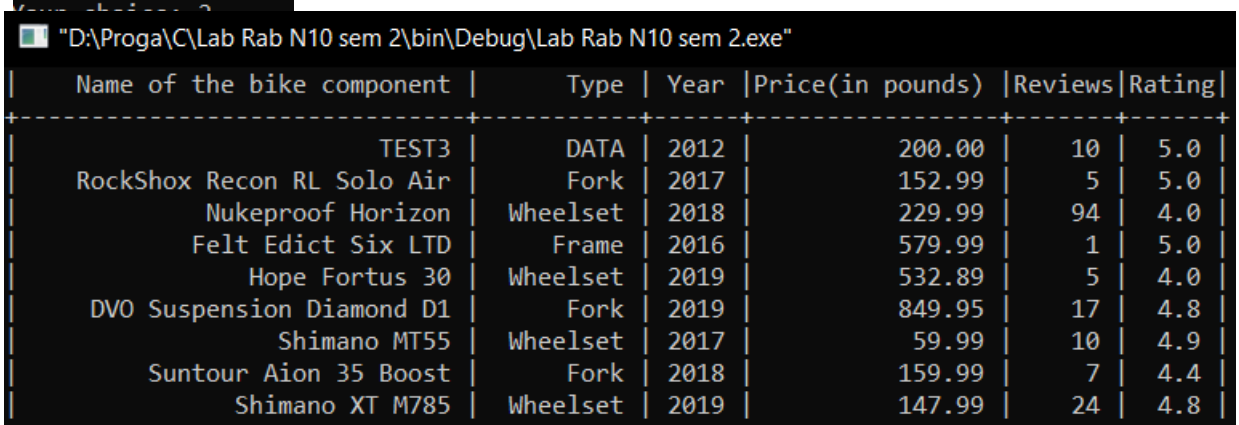


Добавление данных:



Удаление данных через другое поле: price = 100





Текст программы

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "struct_csv.h"
#include "list.h"



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

    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);
        temp = create_node(s3);  //Creating list
        head = temp;

      //Splitting other strings, crating//
      //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);
            p = create_node(s3);    //Adding node in the
            temp -> next = p;       //end of the list
            temp = p;
        }
        if (fclose(df)==EOF) printf ("Error with closing
file!");

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


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

    return 0;
}
```

struct_csv.h

```
#ifndef STRUCT_CSV_H_INCLUDED
#define STRUCT_CSV_H_INCLUDED

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


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();
void struct_out(comps *str0);       //Isn't used in the 10th Lab
void ClearStructure(comps *str0);   //Isn't used in the 10th Lab


#endif // STRUCT_CSV_H_INCLUDED
```

struct_csv.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#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");
}

//Isn't used in the 10th Lab//
void struct_out(comps *str0)
{
    printf("|%30s |%10s |%5d |%16.2f |%6d |%5.1f |\n",
        str0->name,str0->type,str0->year,str0->cost,str0-
>reviews,str0->rating);
}

//Isn't used in the 10th Lab//
void ClearStructure(comps *str0)
{
    free(str0->name);
    free(str0->type);
    str0->name=NULL;
    str0->type=NULL;
    free(str0);
    str0=NULL;
}
```

list.h

```
#ifndef LIST_H_INCLUDED
#define LIST_H_INCLUDED
```

```
#include "struct_csv.h" //contains struct comp


struct node{

    comps *data;
    struct node *next;
};
typedef struct node node;

node *create_node(comps *data);
void list_out(node **head);
void add(node **head, comps *data);
void menu(node *head);
void delete_nodes(node **head);

#endif // LIST_H_INCLUDED
```

list.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "list.h"



node *create_node(comps *data)
{
      node *temp;
      temp = (node *)malloc(sizeof(node));
      temp -> data = data;
      temp -> next = NULL;
      return temp;
}


void list_out(node **head)
{
      node *p;
      p = *head;
      while(p != NULL){
            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");
}


void add(node **head, comps *data)
{
    node *temp;
    temp = (node *)malloc(sizeof(node));
    temp -> data = (comps**)malloc(sizeof(comps*));
    temp -> next = *head;
    temp -> data = data;
    *head = temp;
}

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

    do
    {
        printf("| |     Menu:     |\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");
            printf("Adding node:\n");
            str0 = new_struct();
            add(&head,str0);
            system("cls");
            break;
        case 3:
            system("cls");
            delete_nodes(&head);
```

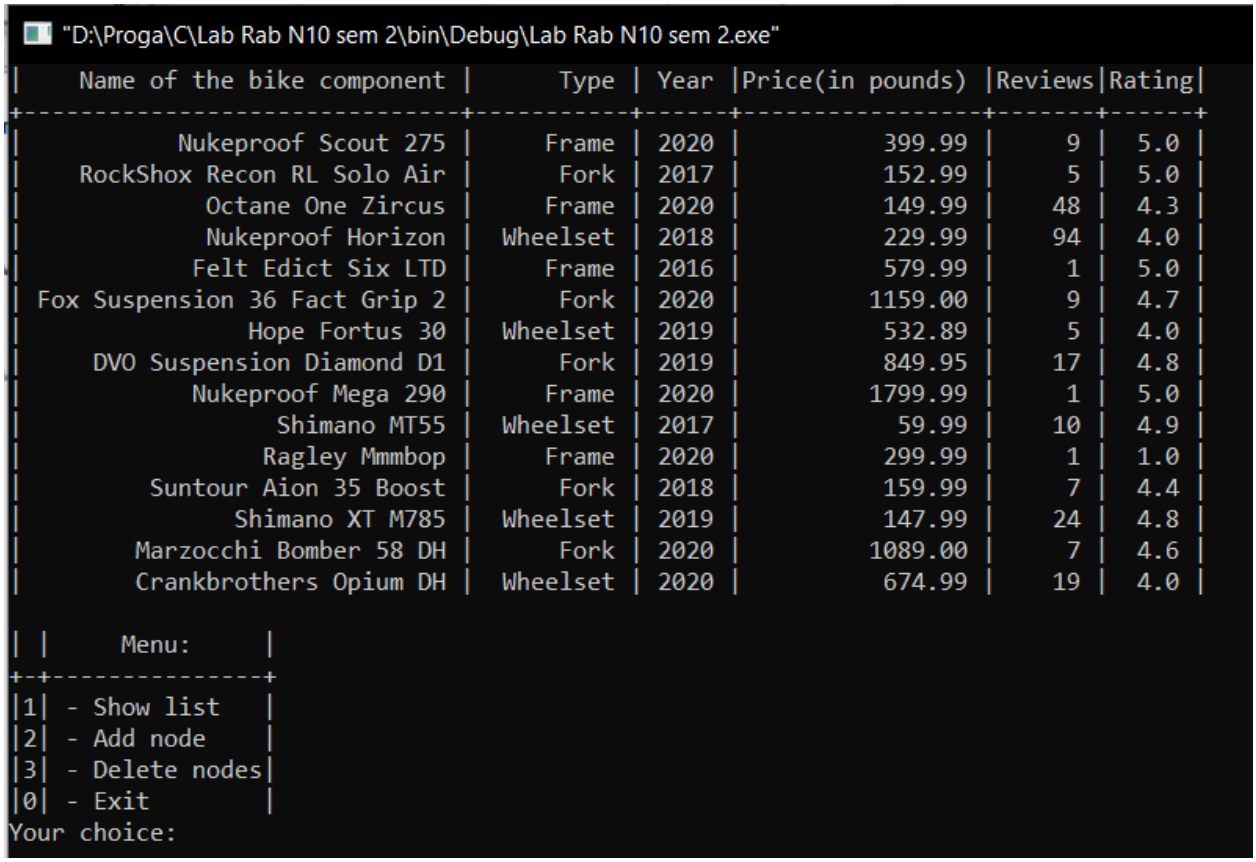
```
            break;
        case 0:

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

    } while(choice!=0);
}


void delete_nodes(node **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;
        chk = 0;        //Check for appropriate data
        switch(choice)
        {
        case 1:
          //Deleting head until data isn't appropriate//
            while((*head)->data->year==value)
            {   p1 = p1 -> next;
                free(head);
                        *head = p1;
                        chk = 1;
            }

          //Goes through list and delete nodes//
```

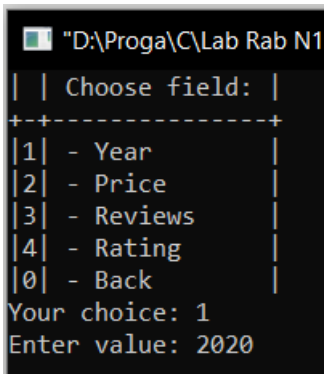
```
  //With appropriate data             //
    do
    {
        p = p1 -> next;
        if(p->data->year==value)
        {
            chk = 1;
            p1 -> next = p ->next;
            free(p);
        }
        else
        {
            p1 = p1 -> next;
        }
    }while (p1->next!=NULL);
    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((*head)->data->cost==value)
    {   p1 = p1 -> next;
        free(head);
                *head = p1;
                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;
            free(p);
        }
        else
        {
            p1 = p1 -> next;
        }
    }while (p1->next!=NULL);
    if (chk==0) printf("There is no such value\n\n");
    else
```

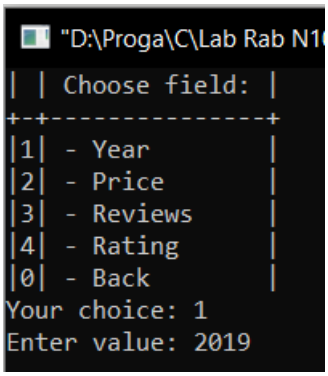
```
    {
        system("cls");
        printf("Nodes have been deleted\n");
        choice = 0;
    }
    break;
case 3:
  //Deleting head until data isn't appropriate//
    while((*head)->data->reviews==value)
    {   p1 = p1 -> next;
        free(head);
                *head = p1;
              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

  //Goes through list and delete nodes//
  //With appropriate data             //
```

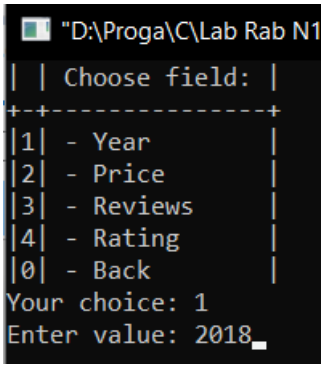
```
            do
            {
                p = p1 -> next;
                if(p->data->rating==value)
                {
                    chk = 1;
                    p1 -> next = p ->next;
                    free(p);
                }
                else
                {
                    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 0:

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

}
```

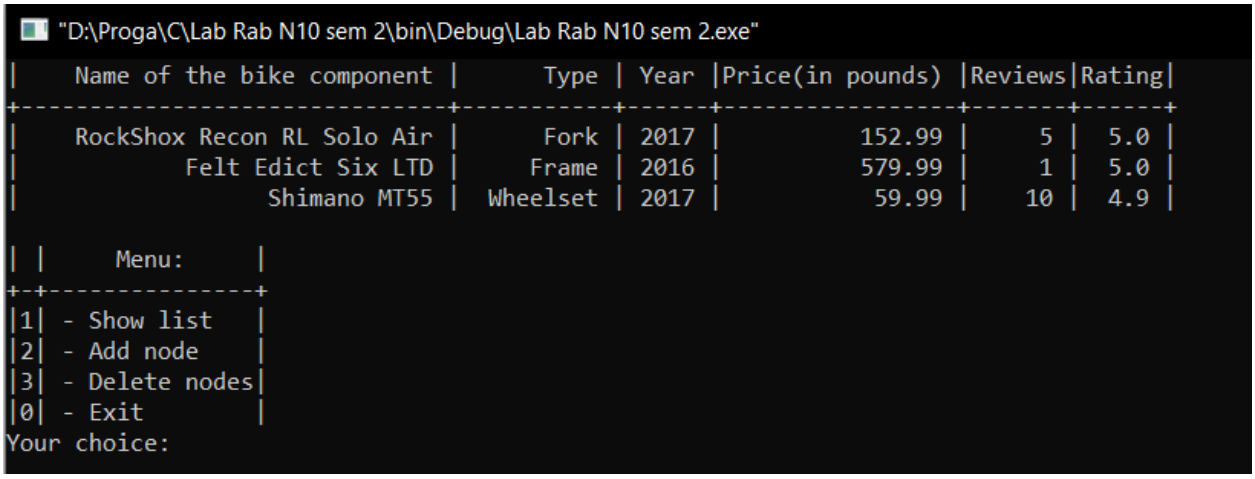
Пример работы программы











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.