МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

Государственное бюджетное образовательное учреждение

высшего образования

«ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПУТЕЙ СООБЩЕНИЯ ИМПЕРАТОРА АЛЕКСАНДРА I»

Кафедра «ИНФОРМАЦИОННЫЕ И ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ»

Дисциплина: «Программирование(C)»

ОТЧЕТ

по лабораторной работе № 6

Вариант *19*

Выполнил студент Шефнер А.

Факультета *АИТ*

Группы *ИВБ-211*

Санкт-Петербург

2023

**Постановка задачи**

1) Создать односвязный список. Данными списка должны быть структуры из задания № 4 соответствующего варианта. Создать функции для записи списка в файл и чтения списка из файла.

2) Реализовать минимум все функции, приведённые в лекции, предложить предложить свои варианты функций работы со списком.

3) Создать меню для управления работы со списком.

**Пояснения**

Мой список является отдельной структурой list, в которой находится количество элементов, ссылка на голову и ссылка на хвост листа. Данные хранятся в узлаъ node в виде указателей на void. Это позволяет Вам использовать данный список не только со структурой book, но и вообще с любыми другими структурами, если Вы работаете с ними через указатели. Для очистки списка сначала примените процедуру free для каждого элемента (это может сделать моя функция list\_apply), а зачем вызовате list\_delete.

**Код программы**

**c\_lab\_6.c** (точка входа программы и меню взаимодействия со списком)

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include "book.h"**

**#include "list.h"**

**#include "booksfile.h"**

**#define ACTION\_COUNT 14**

**list\* book\_list;**

**void(\*actions[ACTION\_COUNT])(void);**

**char\* action\_names[ACTION\_COUNT];**

**void action\_get\_head(void);**

**void action\_list\_get\_tail(void);**

**void action\_list\_get(void);**

**void action\_get\_from\_end(void);**

**void action\_show\_list(void);**

**void action\_push\_back(void);**

**void action\_push\_front(void);**

**void action\_insert(void);**

**void action\_pop\_back(void);**

**void action\_pop\_front(void);**

**void action\_remove\_at(void);**

**void action\_remove\_from\_end(void);**

**void action\_load\_from\_file(void);**

**void action\_write\_to\_file(void);**

***// void\*\* list\_to\_array(const list\* list);***

**void init\_actions(void);**

**void print\_actions(void);**

**int main(int argc, char\* argv[])**

**{**

**system("cls");**

**init\_actions();**

**book\_list = list\_create();**

**printf("Welcome to the book management tool!\n");**

**while (1)**

**{**

**system("pause");**

**system("cls");**

**print\_actions();**

**int action\_num;**

**printf("Enter action number: ");**

**scanf("%d", &action\_num);**

**if(action\_num == 0) break;**

**if(action\_num < 1 || action\_num > ACTION\_COUNT)**

**{**

**printf("Invalid action number.\n");**

**continue;**

**}**

**system("cls");**

**fseek(stdin,0,SEEK\_END);**

**actions[action\_num - 1]();**

**}**

**list\_apply(book\_list, free);**

**list\_delete(book\_list);**

**system("pause");**

**return 0;**

**}**

**void action\_get\_head(void)**

**{**

**printf("Head of the list:\n\n");**

**book\_print(list\_get\_head(book\_list));**

**}**

**void action\_list\_get\_tail(void)**

**{**

**printf("Tail of the list:\n\n");**

**book\_print(list\_get\_tail(book\_list));**

**}**

**void action\_list\_get(void)**

**{**

**size\_t index;**

**printf("Enter desired index: ");**

**scanf("%lld", &index);**

**book\* book = list\_get(book\_list, index);**

**if(book == NULL)**

**{**

**printf("Invalid index.\n");**

**return;**

**}**

**printf("Book at index %lld in the list:\n\n", index);**

**book\_print(book);**

**}**

**void action\_get\_from\_end(void)**

**{**

**size\_t index;**

**printf("Enter desired index: ");**

**scanf("%lld", &index);**

**book\* book = list\_get\_from\_end(book\_list, index);**

**if(book == NULL)**

**{**

**printf("Invalid index.\n");**

**return;**

**}**

**printf("Book at index %lld in the list:\n\n", index);**

**book\_print(book);**

**}**

**void action\_show\_list(void)**

**{**

**if(book\_list->count == 0)**

**{**

**printf("The list is empty.\n");**

**return;**

**}**

**printf("List of the books:\n\n");**

**list\_apply(book\_list, book\_print);**

**printf("\n\n");**

**}**

**void action\_push\_back(void)**

**{**

**printf(" Enter book:\n");**

**book\* new\_book = book\_get\_scanf();**

**list\_push\_back(book\_list, new\_book);**

**printf("\nBook added to list successfully.\n\n");**

**}**

**void action\_push\_front(void)**

**{**

**printf(" Enter book:\n");**

**book\* new\_book = book\_get\_scanf();**

**list\_push\_front(book\_list, new\_book);**

**printf("\nBook added to list successfully.\n\n");**

**}**

**void action\_insert(void)**

**{**

**size\_t index;**

**printf("Enter index (0 - %lld): ", book\_list->count);**

**scanf("%lld", &index);**

**if(index < 0 || index >= book\_list->count)**

**{**

**printf("Invalid index.\n");**

**}**

**printf(" Enter book:\n");**

**book\* new\_book = book\_get\_scanf();**

**list\_insert(book\_list, new\_book, index);**

**printf("\nBook added to list successfully.\n\n");**

**}**

**void action\_pop\_back(void)**

**{**

**book\* removed\_book = list\_pop\_back(book\_list);**

**if(removed\_book == NULL)**

**{**

**printf("The list is empty.\n");**

**return;**

**}**

**printf("The book, popped out from back:\n\n");**

**book\_print(removed\_book);**

**free(removed\_book);**

**}**

**void action\_pop\_front(void)**

**{**

**book\* removed\_book = list\_pop\_front(book\_list);**

**if(removed\_book == NULL)**

**{**

**printf("The list is empty.\n");**

**return;**

**}**

**printf("The book, popped out from front:\n\n");**

**book\_print(removed\_book);**

**free(removed\_book);**

**}**

**void action\_remove\_at(void)**

**{**

**size\_t index;**

**printf("Enter index (0 - %lld): ", book\_list->count);**

**scanf("%lld", &index);**

**if(index < 0 || index >= book\_list->count)**

**{**

**printf("Invalid index.\n");**

**}**

**book\* removed\_book = list\_get(book\_list, index);**

**list\_remove\_at(book\_list, index);**

**printf("\nThe book, removed from the list:\n\n");**

**book\_print(removed\_book);**

**free(removed\_book);**

**}**

**void action\_remove\_from\_end(void)**

**{**

**size\_t index;**

**printf("Enter index (0 - %lld): ", book\_list->count);**

**scanf("%lld", &index);**

**if(index < 0 || index >= book\_list->count)**

**{**

**printf("Invalid index.\n");**

**}**

**book\* removed\_book = list\_get(book\_list, index);**

**list\_remove\_at\_from\_end(book\_list, index);**

**printf("\nThe book, removed from the list:\n\n");**

**book\_print(removed\_book);**

**free(removed\_book);**

**}**

**void action\_load\_from\_file(void)**

**{**

**char filename[100];**

**printf("Enter file name: ");**

**scanf("%s", filename);**

**list\* tmp\_list = get\_book\_list\_from\_file(filename);**

**if(tmp\_list == NULL)**

**{**

**printf("Invalid file name.\n");**

**return;**

**}**

**list\_apply(book\_list, free);**

**free(book\_list);**

**book\_list = tmp\_list;**

**printf("Books loaded successfully\n");**

**}**

**void action\_write\_to\_file(void)**

**{**

**char filename[100];**

**printf("Enter file name: ");**

**scanf("%s", filename);**

**if(write\_book\_list\_to\_file(filename, book\_list))**

**{**

**printf("Data successfully written to the file.\n");**

**}**

**else**

**{**

**printf("Error during writing the list to file.\n");**

**}**

**}**

**void init\_actions(void)**

**{**

**actions[0] = action\_get\_head;**

**action\_names[0] = "Get head of the list.";**

**actions[1] = action\_list\_get\_tail;**

**action\_names[1] = "Get tail of the list.";**

**actions[2] = action\_list\_get;**

**action\_names[2] = "Get an element of the list at desired index.";**

**actions[3] = action\_get\_from\_end;**

**action\_names[3] = "Get an element of the list at desired index starting from the end.";**

**actions[4] = action\_show\_list;**

**action\_names[4] = "Print all elements of the list.";**

**actions[5] = action\_push\_back;**

**action\_names[5] = "Push back anew element to the list.";**

**actions[6] = action\_push\_front;**

**action\_names[6] = "Push front anew element to the list.";**

**actions[7] = action\_insert;**

**action\_names[7] = "Insert a new elenebt in the list at the desired index.";**

**actions[8] = action\_pop\_back;**

**action\_names[8] = "Pop back an element from the list and show it.";**

**actions[9] = action\_pop\_front;**

**action\_names[9] = "Pop front an element from the list and show it.";**

**actions[10] = action\_remove\_at;**

**action\_names[10] = "Remove an element from the list at the desired index.";**

**actions[11] = action\_remove\_from\_end;**

**action\_names[11] = "Remove an element from the list at the desired index starting from the end.";**

**actions[12] = action\_load\_from\_file;**

**action\_names[12] = "Load books from the file.";**

**actions[13] = action\_write\_to\_file;**

**action\_names[13] = "Write books to the file.";**

**}**

**void print\_actions(void)**

**{**

**for(int i = 0; i < ACTION\_COUNT; i++)**

**{**

**printf("%d - %s\n", i + 1, action\_names[i]);**

**}**

**printf("\n%d - Exit.\n\n", 0);**

**}**

**book.h** (струтура книги и основные функции)

**#pragma once**

**#define SURNAME\_CHAR\_NUMBER 20**

**#define THEME\_CHAR\_NUMBER 50**

**#define FULL\_CHAR\_NUMBER 83**

**#define SURNAME\_FORMAT "%20s"**

**#define THEME\_FORMAT "%50s"**

**#define YEAR\_FORMAT "%5hu"**

**#define PAGE\_FORMAT "%5hu"**

**#define BOOK\_FORMAT SURNAME\_FORMAT" "THEME\_FORMAT" "YEAR\_FORMAT" "PAGE\_FORMAT**

**typedef struct book**

**{**

**char surname[SURNAME\_CHAR\_NUMBER + 1];**

**char theme[THEME\_CHAR\_NUMBER + 1];**

**unsigned short year;**

**unsigned short page\_count;**

**} book;**

**void book\_print(book\* book);**

**book\* book\_get\_scanf(void);**

**book.с**

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**#include "book.h"**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**void write\_str\_without\_trailing\_spaces(char\* str\_ptr)**

**{**

**while (\*str\_ptr == ' ') str\_ptr++;**

**while (\*str\_ptr != '\0')**

**{**

**char tmp = \*str\_ptr == '\_' ? ' ' : (char)(\*str\_ptr);**

**putc(tmp, stdout);**

**str\_ptr++;**

**}**

**}**

**void book\_print(book\* book)**

**{**

**write\_str\_without\_trailing\_spaces(book->surname);**

**printf(" - ");**

**write\_str\_without\_trailing\_spaces(book->theme);**

**printf("\nyear: %d. %d pages.\n\n", book->year, book->page\_count);**

**}**

**book\* book\_get\_scanf(void)**

**{**

**fseek(stdin,0,SEEK\_END);**

**book\* book = malloc(sizeof(struct book));**

**printf("\nEnter surname:\n");**

**fgets(book->surname, SURNAME\_CHAR\_NUMBER, stdin);**

**book->surname[strcspn(book->surname, "\r\n")] = '\0';**

**printf("Enter theme:\n");**

**fgets(book->theme, THEME\_CHAR\_NUMBER, stdin);**

**book->theme[strcspn(book->theme, "\r\n")] = '\0';**

**printf("Enter year: ");**

**scanf("%hd", &book->year);**

**printf("Enter page count: ");**

**scanf("%hd", &book->page\_count);**

**return book;**

**}**

**list.h** (связный список)

**#pragma once**

**typedef** **unsigned** **long** **long** **size\_t**;

**typedef** **struct** **node**

{

**void**\* data;

**struct** **node**\* next;

} node;

**typedef** **struct** **list**

{

node\* head;

node\* tail;

**size\_t** count;

} list;

*// Creates a new list.*

list\* **list\_create**(**void**);

*// Returns a data of the lists head.*

**void**\* **list\_get\_head**(**const** list\* list);

*// Returns a data of the lists tail.*

**void**\* **list\_get\_tail**(**const** list\* list);

*// Returns a data from the list at index.*

**void**\* **list\_get**(**const** list\* list, **size\_t** index);

*// Returns a data from the list at index starting from end of list.*

**void**\* **list\_get\_from\_end**(**const** list\* list, **size\_t** index);

*// Converts a list to an array and returns a pointer to the array.*

*// Warning: a memory for an array is allocated with malloc.*

*// Don't forget to free the array after use.*

**void**\*\* **list\_to\_array**(**const** list\* list);

*// Adds a new element at the end of the list.*

**int** **list\_push\_back**(list\* list, **void**\* data);

*// Adds a new element at rhe start of the list*

**int** **list\_push\_front**(list\* list, **void**\* data);

*// Inserts a new element at a desired index.*

**int** **list\_insert**(list\* list, **void**\* data, **size\_t** index);

*// Removes an element from the end of the list and returns ins data*

**void**\* **list\_pop\_back**(list\* list);

*// Removes an element from the start of the list and returns ins data*

**void**\* **list\_pop\_front**(list\* list);

*// Removes an element from the list at index.*

**int** **list\_remove\_at**(list\* list, **size\_t** index);

*// Removes an element from the list at indext starting from the end.*

**int** **list\_remove\_at\_from\_end**(list\* list, **size\_t** index);

*// Applies a void function to every element of the list.*

**void** **list\_apply**(**const** list\* list, **void**(\*func)(**void**\*));

*// Applies a void function to every element of the list and*

*// saves the result to a new list.*

*// Warning: the new list will be allocated with malloc.*

list\* **list\_apply\_and\_save**(**const** list\* list, **void**\*(\*func)(**void**\*));

*// Deletes all the nodes of the list.*

*// Warning: data inside the nodes will not be deleted.*

**int** **list\_delete**(list\* list);

**list.с**

**#include "list.h"**

**#include <stdlib.h>**

list\* **list\_create**(**void**)

{

list\* list = malloc(sizeof(**struct** list));

list->count = 0;

list->head = NULL;

list->tail = NULL;

**return** list;

}

**void**\* **list\_get\_head**(**const** list\* list)

{

**return** list->head == NULL ? NULL : list->head->data;

}

**void**\* **list\_get\_tail**(**const** list\* list)

{

**return** list->tail == NULL ? NULL : list->tail->data;

}

**void**\* **list\_get**(**const** list\* list, **const** **size\_t** index)

{

**if**(index >= list->count || index < 0) **return** NULL;

**const** node\* tmp = list->head;

**for**(**size\_t** i = 0; i < index; i++)

{

tmp = tmp->next;

**if**(tmp == NULL) **return** NULL;

}

**return** tmp->data;

}

**void**\* **list\_get\_from\_end**(**const** list\* list, **const** **size\_t** index)

{

**return** list\_get(list, list->count - index);

}

**void**\*\* **list\_to\_array**(**const** list\* list)

{

**if**(list->count == 0) **return** NULL;

**void**\*\* array = malloc(sizeof(**void**\*) \* list->count);

**const** node\* tmp = list->head;

**for**(**size\_t** i = 0; i < list->count; i++)

{

array[i] = tmp->data;

tmp = tmp->next;

}

**return** array;

}

**int** **list\_push\_back**(list\* list, **void**\* data)

{

node\* node\_ptr = malloc(sizeof(node));

node\_ptr->data = data;

**if**(list->count == 0)

list->head = node\_ptr;

**else**

list->tail->next = node\_ptr;

list->tail = node\_ptr;

list->count++;

**return** 1;

}

**int** **list\_push\_front**(list\* list, **void**\* data)

{

**if**(list->count == 0)

{

**return** list\_push\_back(list, data);

}

node\* new\_node = malloc(sizeof(node));

new\_node->data = data;

new\_node->next = list->head;

list->head = new\_node;

**if**(list->count == 1)

{

list->tail = list->head->next;

}

list->count++;

**return** 1;

}

**int** **list\_insert**(list\* list, **void**\* data, **const** **size\_t** index)

{

**if**(index > list->count || index < 0) **return** 0;

**if**(index == list->count) **return** list\_push\_back(list, data);

**if**(index == 0) **return** list\_push\_front(list, data);

node\* tmp = list->head;

node\* new\_node = malloc(sizeof(node));

new\_node->data = data;

**for**(**size\_t** i = 0; i < index - 1; i++)

{

tmp = tmp->next;

}

new\_node->next = tmp->next;

tmp->next = new\_node;

list->count++;

**return** 1;

}

**void**\* **list\_pop\_back**(list\* list)

{

**if**(list->count == 0) **return** NULL;

**void**\* return\_data = list->tail->data;

**if**(list->count == 1)

{

free(list->tail);

list->tail = NULL;

list->head = NULL;

list->count--;

**return** return\_data;

}

node\* tmp = list->head;

**for**(**size\_t** i = 0; i < list->count - 2; i++)

{

tmp = tmp->next;

}

free(list->tail);

list->tail = tmp;

list->count--;

**return** return\_data;

}

**void**\* **list\_pop\_front**(list\* list)

{

**if**(list->count == 0) **return** NULL;

**if**(list->count == 1) **return** list\_pop\_back(list);

**void**\* data = list->head->data;

node\* tmp = list->head;

list->head = list->head->next;

free(tmp);

list->count--;

**return** data;

}

**int** **list\_remove\_at**(list\* list, **const** **size\_t** index)

{

**if**(list->count == 0 || index < 0) **return** 0;

**if**(index == 0)

{

list\_pop\_front(list);

**return** 1;

}

**if**(index == list->count - 1)

{

list\_pop\_back(list);

**return** 1;

}

**if**(list->count == 1)

{

free(list->head);

list->head = NULL;

list->tail = NULL;

**return** 1;

}

node\* tmp = list->head;

**for**(**size\_t** i = 0; i < index - 1; i++)

{

tmp = tmp->next;

}

node\* removed = tmp->next;

tmp->next = removed->next;

free(removed);

list->count--;

**return** 1;

}

**int** **list\_remove\_at\_from\_end**(list\* list, **size\_t** index)

{

**return** list\_remove\_at(list, list->count - index);

}

**void** **list\_apply**(**const** list\* list, **void**(\* func)(**void**\*))

{

**const** node\* tmp = list->head;

**for**(**size\_t** i = 0; i < list->count; i++)

{

func(tmp->data);

tmp = tmp->next;

}

}

list\* **list\_apply\_and\_save**(**const** list\* list, **void**\*(\* func)(**void**\*))

{

**struct** **list**\* new\_list = list\_create();

**if**(list->count == 0) **return** new\_list;

**const** node\* tmp = list->head;

node\* tmp\_new = malloc(sizeof(node));

new\_list->head = tmp\_new;

new\_list->head->data = func(list->head->data);

**for**(**size\_t** i = 0; i < list->count - 1; i++)

{

tmp\_new->next = malloc(sizeof(node));

tmp\_new = tmp\_new->next;

tmp = tmp->next;

tmp\_new->data = func(tmp->data);

}

new\_list->tail = tmp\_new;

new\_list->count = list->count;

**return** new\_list;

}

**int** **list\_delete**(list\* list)

{

**if**(list->count == 0)

{

free(list);

**return** 1;

}

**if**(list->count == 1)

{

free(list->head);

free(list);

**return** 1;

}

node\* tmp = list->head;

**for**(**size\_t** i = 0; i < list->count; i++)

{

node\* tmp\_next = tmp->next;

free(tmp);

tmp = tmp\_next;

}

free(list);

**return** 1;

}

**booksfile.h** (функции для чтения из файла и запись в файл книг)

**#pragma once**

**#include "list.h"**

list\* **get\_book\_list\_from\_file**(**const** **char**\* path);

**int** **write\_book\_list\_to\_file**(**char**\* path, list\* list);

**booksfile.с**

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**#include "booksfile.h"**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include "book.h"**

list\* **get\_book\_list\_from\_file**(**const** **char**\* path)

{

FILE\* file = fopen(path, "r");

**if**(!file) **return** NULL;

list\* book\_list = list\_create();

**char** buf[300];

**while**(fgets(buf, 300, file))

{

book\* new\_book = malloc(sizeof(book));

**int** idx = 0;

**for**(**int** i = 0; i < SURNAME\_CHAR\_NUMBER; i++, idx++)

{

new\_book->surname[i] = buf[idx];

}

new\_book->surname[SURNAME\_CHAR\_NUMBER] = '\0';

idx++;

**for**(**int** i = 0; i < THEME\_CHAR\_NUMBER; i++, idx++)

{

new\_book->theme[i] = buf[idx];

}

new\_book->theme[THEME\_CHAR\_NUMBER] = '\0';

idx++;

**char** num\_str[5];

**for**(**int** i = 0; i < 5; i++, idx++)

{

num\_str[i] = buf[idx];

}

idx++;

new\_book->year = atoi(num\_str);

**for**(**int** i = 0; i < 5; i++, idx++)

{

num\_str[i] = buf[idx];

}

new\_book->page\_count = atoi(num\_str);

list\_push\_back(book\_list, new\_book);

}

fclose(file);

**return** book\_list;

}

**int** **write\_book\_list\_to\_file**(**char**\* path, list\* list)

{

FILE\* file = fopen(path, "w");

**if**(!file) **return** 0;

book\*\* book\_arr = list\_to\_array(list);

**for**(**int** i = 0; i < list->count; i++)

{

fprintf(file, BOOK\_FORMAT"\n",

book\_arr[i]->surname,

book\_arr[i]->theme,

book\_arr[i]->year,

book\_arr[i]->page\_count

);

}

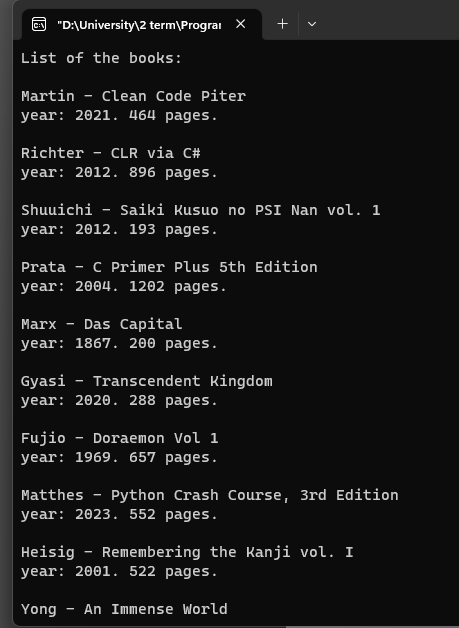
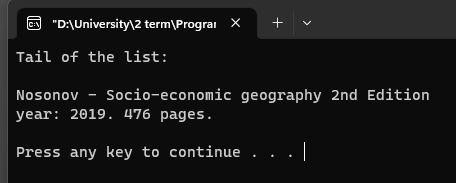
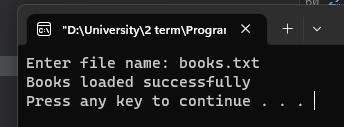
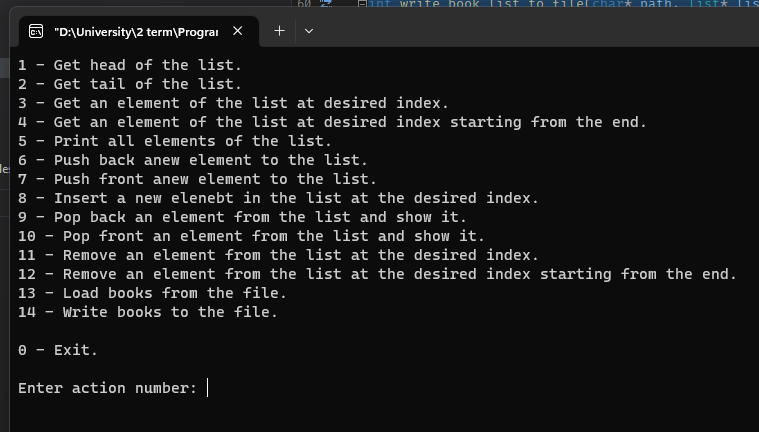
fclose(file);

free(book\_arr);

**return** 1;

}

**Отладка приложения**



**Содержимое файлов**

books.txt – исходный файл, из которого считываются книги.

