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(С)»

О Т Ч Е Т
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");
```

```c
        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);
```

```c
}

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);
```

```c
    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);
```

```c
}

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.c

```c
#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.c

```c
#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.c

```c
#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

```
"D:\University\2 term\Prograı

1 - Get head of the list.
2 - Get tail of the list.
3 - Get an element of the list at desired index.
4 - Get an element of the list at desired index starting from the end.
5 - Print all elements of the list.
6 - Push back anew element to the list.
7 - Push front anew element to the list.
8 - Insert a new elenebt in the list at the desired index.
9 - Pop back an element from the list and show it.
10 - Pop front an element from the list and show it.
11 - Remove an element from the list at the desired index.
12 - Remove an element from the list at the desired index starting from the end.
13 - Load books from the file.
14 - Write books to the file.

0 - Exit.

Enter action number:
```

```
"D:\University\2 term\Prograı

Enter file name: books.txt
Books loaded successfully
Press any key to continue . . .
```

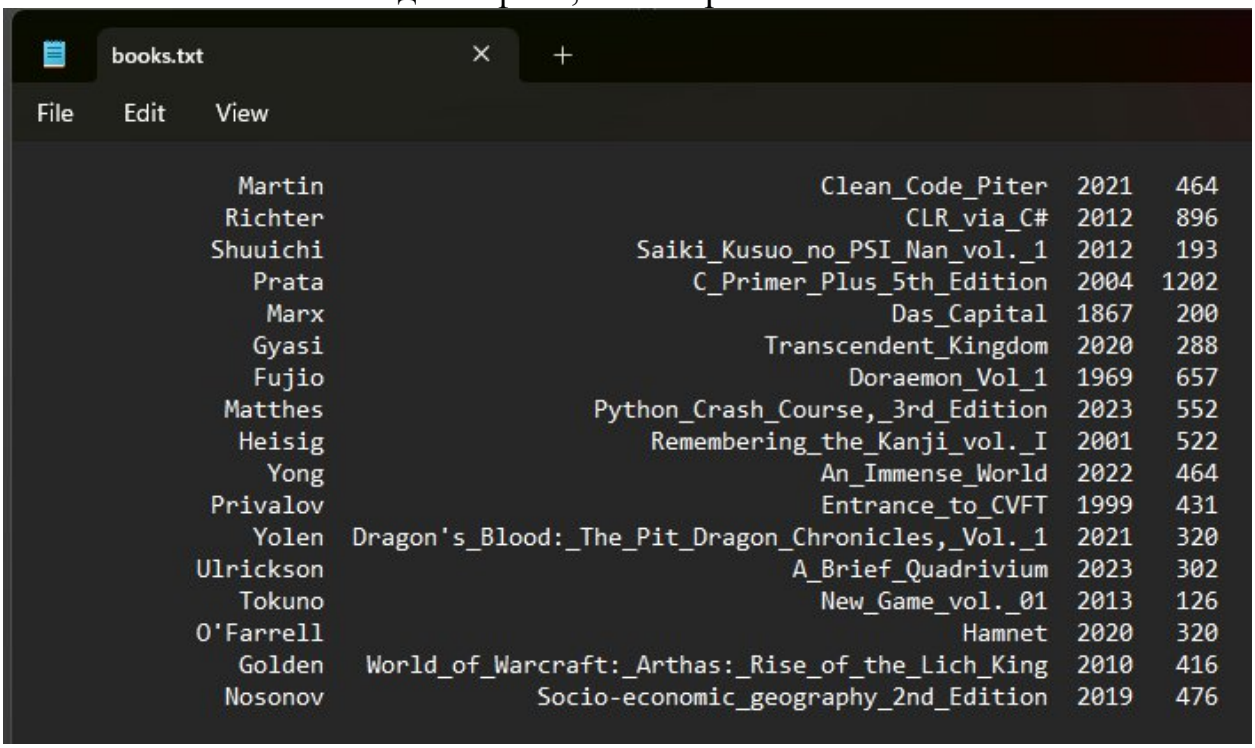
```
"D:\University\2 term\Prograı

Tail of the list:

Nosonov - Socio-economic geography 2nd Edition
year: 2019. 476 pages.

Press any key to continue . . .
```

"D:\University\2 term\Prograı

```
List of the books:

Martin - Clean Code Piter
year: 2021. 464 pages.

Richter - CLR via C#
year: 2012. 896 pages.

Shuuichi - Saiki Kusuo no PSI Nan vol. 1
year: 2012. 193 pages.

Prata - C Primer Plus 5th Edition
year: 2004. 1202 pages.

Marx - Das Capital
year: 1867. 200 pages.

Gyasi - Transcendent Kingdom
year: 2020. 288 pages.

Fujio - Doraemon Vol 1
year: 1969. 657 pages.

Matthes - Python Crash Course, 3rd Edition
year: 2023. 552 pages.

Heisig - Remembering the Kanji vol. I
year: 2001. 522 pages.

Yong - An Immense World
```

Содержимое файлов

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



```
Martin                                  Clean_Code_Piter  2021   464
Richter                                       CLR_via_C#  2012   896
Shuuichi                      Saiki_Kusuo_no_PSI_Nan_vol._1  2012   193
Prata                           C_Primer_Plus_5th_Edition  2004  1202
Marx                                          Das_Capital  1867   200
Gyasi                                Transcendent_Kingdom  2020   288
Fujio                                      Doraemon_Vol_1  1969   657
Matthes                    Python_Crash_Course,_3rd_Edition  2023   552
Heisig                         Remembering_the_Kanji_vol._I  2001   522
Yong                                     An_Immense_World  2022   464
Privalov                                Entrance_to_CVFT  1999   431
Yolen  Dragon's_Blood:_The_Pit_Dragon_Chronicles,_Vol._1  2021   320
Ulrickson                                A_Brief_Quadrivium  2023   302
Tokuno                                    New_Game_vol._01  2013   126
O'Farrell                                           Hamnet  2020   320
Golden   World_of_Warcraft:_Arthas:_Rise_of_the_Lich_King  2010   416
Nosonov                Socio-economic_geography_2nd_Edition  2019   476
```