Титульный лист

Оглавление

[1 Цель работы 3](#_Toc60239729)

[2 Задание №1 3](#_Toc60239730)

[2.1 Текст задания 3](#_Toc60239731)

[2.2 Текст программы 3](#_Toc60239732)

[2.3 Пример выполнения 4](#_Toc60239733)

[3 Задание №2 5](#_Toc60239734)

[3.1 Текст задания 5](#_Toc60239735)

[3.2 Блок-схема алгоритма 5](#_Toc60239736)

[3.3 Текст программы 5](#_Toc60239737)

[3.4 Пример выполнения 6](#_Toc60239738)

# Цель работы

Цель работы – создание компьютерной программы, позволяющей вводить, сохранять, анализировать и выводить информацию, хранящуюся в динамической структуре данных.

# Задачи

Для достижения указанной цели необходимо решить следующие задачи:

1. Разработать динамическую структуру хранения данных типа двунаправленный связанный список, содержащую данные о личной библиотеке, каждая книга в которой содержит следующие сведения:

* шифр книги;
* фамилию автора;
* название;
* год издания.

Количество записей заранее неизвестно.

2. Разработать программу, в которой реализуются следующие операции над данными:

* добавление новой записи в список;
* удаление записи из списка;
* редактирование записи в списке;
* сортировка списка;
* вывод на экран содержимого списка;
* загрузка данных из файла в список;
* запись списка в файл;
* извлечение информации из списка по указанным критериям.

# Листинг программы

**bilist.h**

**#ifndef** \_\_BIDIRECTIONAL\_LINKED\_LIST\_H\_\_

**#define** \_\_BIDIRECTIONAL\_LINKED\_LIST\_H\_\_

**#include** <stdio.h>

// функции обратного вызова для элементов списка

**typedef** **void** (\*callback\_t)(**void**\*); // для удаления

**typedef** **void** (\*writer\_t) (**void**\*, FILE\*); // для печати

**typedef** **void**\* (\*reader\_t) (FILE\*); // для чтения

**typedef** **int** (\*match\_t) (**const** **void**\*, **const** **void**\* value); // для поиска

**typedef** **int** (\*compare\_t) (**const** **void**\* lhs, **const** **void**\* rhs); // для сортировки

// элемент двунаправленного списка

**typedef** **struct** binode\_t {

**struct** binode\_t\* prev;

**struct** binode\_t\* next;

**void**\* data;

} binode\_t;

// структура двунаправленного списка

**typedef** **struct** bilist\_t {

**struct** binode\_t\* head;

**struct** binode\_t\* tail;

**int** count;

} bilist\_t;

// создаёт объект списка

bilist\_t\* **bilist\_create**();

// удаляет содержимое списка из памяти

**void** **bilist\_clear**(bilist\_t\* list, callback\_t release);

// удаляет содержимое и объект списка из памяти

**void** **bilist\_destroy**(bilist\_t\* books, callback\_t release);

// вставляет элемент в конец списка

**void** **bilist\_push\_back**(bilist\_t\* list, **void**\* data);

// вставляет элемент в список перед curr

**void** **bilist\_insert**(bilist\_t\* list, binode\_t\* curr, **void**\* data);

// удаляет элемент item из списка

**void** **bilist\_remove**(bilist\_t\* list, binode\_t\* item, callback\_t release);

// возвращает размер списка

**int** **bilist\_size**(**const** bilist\_t\* list);

// возвращает элемент по индексу

binode\_t\* **bilist\_item**(bilist\_t\* list, **int** index);

// возвращает данные по индексу

**void**\* **bilist\_get\_data**(bilist\_t\* list, **int** index);

// заменяет данные элемента

**void** **bilist\_set\_data**(bilist\_t\* list, binode\_t\* item, **void**\* data, callback\_t release);

// печатает список

**void** **bilist\_print**(**const** bilist\_t\* list, writer\_t printer, FILE\* stream, **const** **char**\* title);

// считывает список

**void** **bilist\_read**(bilist\_t\* list, reader\_t reader, FILE\* stream);

// ищет элемент

**void**\* **bilist\_find\_data**(**const** bilist\_t\* list, match\_t match, **const** **void**\* value);

// сортирует список

**void** **bilist\_bubble\_sort**(bilist\_t\* list, compare\_t compare);

**#endif** // \_\_BIDIRECTIONAL\_LINKED\_LIST\_H\_\_

**bilist.c**

**#include** "bilist.h"

**#include** <stdlib.h>

// создаёт объект списка

bilist\_t\* **bilist\_create**() {

bilist\_t\* list = (bilist\_t\*)**malloc**(**sizeof**(bilist\_t));

list->head = NULL;

list->tail = NULL;

list->count = 0;

**return** list;

}

// удаляет содержимое списка из памяти

**void** **bilist\_clear**(bilist\_t\* list, callback\_t release) {

binode\_t\* curr = list->head;

**while**(curr) {

binode\_t\* next = curr->next;

release(curr->data);

**free**(curr);

curr = next;

}

list->head = list->tail = NULL;

list->count = 0;

}

// удаляет содержимое и объект списка из памяти

**void** **bilist\_destroy**(bilist\_t\* books, callback\_t release) {

bilist\_clear(books, release);

**free**(books);

}

// вставляет элемент в конец списка

**void** **bilist\_push\_back**(bilist\_t\* list, **void**\* data) {

binode\_t\* item = (binode\_t\*)**malloc**(**sizeof**(binode\_t));

item->data = data;

item->next = NULL;

**if**(list->tail) { // список не пуст

item->prev = list->tail;

list->tail->next = item;

} **else** {

item->prev = NULL;

list->head = item;

}

list->tail = item;

list->count++;

}

// вставляет элемент в список перед curr

**void** **bilist\_insert**(bilist\_t\* list, binode\_t\* curr, **void**\* data) {

binode\_t\* item = (binode\_t\*)**malloc**(**sizeof**(binode\_t));

item->data = data;

item->next = curr;

**if**(curr) {

item->next->prev = curr->prev;

} **else** {

item->prev = NULL;

}

**if**(list->tail) { // список не пуст

item->prev = list->tail;

list->tail->next = item;

} **else** {

list->head = item;

list->tail = item;

}

list->count++;

}

// удаляет элемент item из списка

**void** **bilist\_remove**(bilist\_t\* list, binode\_t\* item, callback\_t release) {

**if**(!list->count)

**return**;

**if**(item->prev) {

item->prev->next = item->next;

}

**if**(item->next) {

item->next->prev = item->prev;

}

list->count--;

**if**(!list->count) {

list->head = NULL;

list->tail = NULL;

}

**if**(release)

release(item->data);

**free**(item);

}

// возвращает размер списка

**int** **bilist\_size**(**const** bilist\_t\* list) {

**return** list->count;

}

// возвращает элемент по индексу

binode\_t\* **bilist\_item**(bilist\_t\* list, **int** index) {

binode\_t\* curr = list->head;

**int** n = 0;

**while**(curr) {

**if**(index == n)

**return** curr;

curr = curr->next;

n++;

}

**return** NULL;

}

// возвращает данные по индексу

**void**\* **bilist\_get\_data**(bilist\_t\* list, **int** index) {

binode\_t\* curr = list->head;

**int** n = 0;

**while**(curr) {

**if**(index == n)

**return** curr->data;

curr = curr->next;

}

**return** NULL;

}

// заменяет данные элемента

**void** **bilist\_set\_data**(bilist\_t\* list, binode\_t\* item, **void**\* data, callback\_t release) {

**if**(!list->count)

**return**;

**if**(release)

release(item->data);

item->data = data;

**free**(item);

}

// печатает список

**void** **bilist\_print**(**const** bilist\_t\* list, writer\_t printer, FILE\* stream, **const** **char**\* title) {

**if**(title)

**fprintf**(stream, "%s:\n", title);

binode\_t\* curr = list->head;

**int** index = 0;

**while**(curr) {

**fprintf**(stream, "%d\t", index++);

printer(curr->data, stream);

curr = curr->next;

}

**fflush**(stream);

}

// считывает список

**void** **bilist\_read**(bilist\_t\* list, reader\_t reader, FILE\* stream) {

**void**\* data;

**while**((data = reader(stream))) {

bilist\_push\_back(list, data);

}

}

// ищет элемент

**void**\* **bilist\_find\_data**(**const** bilist\_t\* list, match\_t match, **const** **void**\* value) {

binode\_t\* curr = list->head;

**while**(curr) {

**if**(match(curr->data, value))

**return** curr->data;

curr = curr->next;

}

**return** NULL;

}

**static** **void** **bilist\_swap**(binode\_t \*a, binode\_t \*b)

{

**void**\* temp = a->data;

a->data = b->data;

b->data = temp;

}

// сортирует список

**void** **bilist\_bubble\_sort**(bilist\_t\* list, compare\_t compare) {

binode\_t \*left; // будет всегда указывать на начало списка

binode\_t \*right = NULL; // будет всегда указывать на начало списка

**int** swapped;

**do** {

swapped = 0;

left = list->head;

**while**(left->next != right) {

**if**(compare(left->data, left->next->data) > 0) {

bilist\_swap(left, left->next);

swapped = 1;

}

left = left->next;

}

// поскольку максимальный элемент уже в конце списка, присваимаем его указателю right,

// чтобы не проверять понапрасну уже отсортированный список

right = left;

} **while**(swapped);

}

**book.h**

**#ifndef** \_\_MY\_LIBRARY\_DATABASE\_H\_\_

**#define** \_\_MY\_LIBRARY\_DATABASE\_H\_\_

**#include** <stdio.h>

**#ifndef** TRUE

**# define** TRUE 1

**#endif**

**#ifndef** FALSE

**# define** FALSE 0

**#endif**

**#define** MAX\_STR\_SIZE 256

// структура, описывающая одну книгу

**typedef** **struct** book\_t {

**unsigned** year;

**char** code[MAX\_STR\_SIZE];

**char** author[MAX\_STR\_SIZE];

**char** name[MAX\_STR\_SIZE];

} book\_t;

// конструктор по умолчанию

book\_t\* **book\_create**();

// конструктор с параметрами

book\_t\* **book\_create\_init**(**unsigned** year, **const** **char**\* code, **const** **char**\* author, **const** **char**\* name);

// деструктор

**void** **book\_destroy**(**void**\* book);

// функции ввода

**void** **input\_book\_year**(book\_t\* book, FILE\* stream, **int** prompt);

**void** **input\_book\_code**(book\_t\* book, FILE\* stream, **int** prompt);

**void** **input\_book\_author**(book\_t\* book, FILE\* stream, **int** prompt);

**void** **input\_book\_name**(book\_t\* book, FILE\* stream, **int** prompt);

**void**\* **input\_book**(FILE\* stream, **int** prompt);

**void**\* **read\_book**(FILE\* stream);

// функция вывода

**void** **print\_book**(**void**\* ptr, FILE\* stream);

// функции обратного вызова для поиска

**int** **book\_compare\_year**(**const** **void**\* ptr, **const** **void**\* value);

**int** **book\_compare\_code**(**const** **void**\* ptr, **const** **void**\* value);

**int** **book\_compare\_author**(**const** **void**\* ptr, **const** **void**\* value);

**int** **book\_compare\_name**(**const** **void**\* ptr, **const** **void**\* value);

// функции обратного вызова для сортировки

**int** **book\_compare\_by\_year**(**const** **void**\* lhs, **const** **void**\* rhs);

**int** **book\_compare\_by\_code**(**const** **void**\* lhs, **const** **void**\* rhs);

**int** **book\_compare\_by\_author**(**const** **void**\* lhs, **const** **void**\* rhs);

**int** **book\_compare\_by\_name**(**const** **void**\* lhs, **const** **void**\* rhs);

**#endif** // \_\_MY\_LIBRARY\_DATABASE\_H\_\_

**book.c**

// Личная библиотека

**#include** "book.h"

**#include** <stdlib.h>

**#include** <string.h>

book\_t\* **book\_create**() {

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

book->year = 0;

book->code[0] = '\0';

book->author[0] = '\0';

book->name[0] = '\0';

**return** book;

}

book\_t\* **book\_create\_init**(**unsigned** year, **const** **char**\* code, **const** **char**\* author, **const** **char**\* name) {

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

book->year = year;

**strcpy**(book->code, code);

**strcpy**(book->author, author);

**strcpy**(book->name, name);

**return** book;

}

// функции ввода

**void** **book\_destroy**(**void**\* book) {

**free**(book);

}

**void** **input\_book\_year**(book\_t\* book, FILE\* stream, **int** prompt) {

**if**(prompt) {

**printf**("Введите год выпуска: ");

**fflush**(stdout);

}

**fscanf**(stream, "%u", &book->year);

}

**void** **input\_book\_code**(book\_t\* book, FILE\* stream, **int** prompt) {

**if**(prompt) {

**printf**("Введите шифр книги: ");

**fflush**(stdout);

}

**fscanf**(stream, "%255s", book->code);

}

**void** **input\_book\_author**(book\_t\* book, FILE\* stream, **int** prompt) {

**if**(prompt) {

**printf**("Введите фамилию автора: ");

**fflush**(stdout);

}

**fscanf**(stream, "%255s", book->author);

}

**void** **input\_book\_name**(book\_t\* book, FILE\* stream, **int** prompt) {

**if**(prompt) {

**printf**("Введите название книги: ");

**fflush**(stdout);

}

**fscanf**(stream, "%255s", book->name);

}

**void**\* **input\_book**(FILE\* stream, **int** prompt) {

book\_t\* book = book\_create();

input\_book\_year(book, stream, prompt);

input\_book\_code(book, stream, prompt);

input\_book\_author(book, stream, prompt);

input\_book\_name(book, stream, prompt);

**return** book;

}

**void**\* **read\_book**(FILE\* stream) {

book\_t\* book = book\_create();

**int** index;

**int** scan\_result = **fscanf**(stream, "%d\t%u\t%s\t%s\t%s\n", &index, &book->year, book->code, book->author, book->name);

**if**(scan\_result != 5) {

**free**(book);

book = NULL;

}

**return** book;

}

// функция вывода

**void** **print\_book**(**void**\* ptr, FILE\* stream) {

**const** book\_t\* book = (**const** book\_t\*)ptr;

**if**(book) {

**fprintf**(stream, "%u\t%s\t%s\t%s\n", book->year, book->code, book->author, book->name);

} **else** {

**fprintf**(stream, "Книга не найдена\n");

}

}

// функции обратного вызова для поиска

**int** **book\_compare\_year**(**const** **void**\* ptr, **const** **void**\* value) {

**const** book\_t\* book = (**const** book\_t\*)ptr;

**int** year = \*((**int**\*)value);

**return** book->year == year;

}

**int** **book\_compare\_code**(**const** **void**\* ptr, **const** **void**\* value) {

**const** book\_t\* book = (**const** book\_t\*)ptr;

**const** **char**\* code = value;

**return** (**strcmp**(book->code, code) == 0);

}

**int** **book\_compare\_author**(**const** **void**\* ptr, **const** **void**\* value) {

**const** book\_t\* book = (**const** book\_t\*)ptr;

**const** **char**\* author = value;

**return** (**strcmp**(book->author, author) == 0);

}

**int** **book\_compare\_name**(**const** **void**\* ptr, **const** **void**\* value) {

**const** book\_t\* book = (**const** book\_t\*)ptr;

**const** **char**\* name = value;

**return** (**strcmp**(book->name, name) == 0);

}

// функции обратного вызова для сортировки

**int** **book\_compare\_by\_year**(**const** **void**\* lhs, **const** **void**\* rhs) {

**const** book\_t\* a = (**const** book\_t\*)lhs;

**const** book\_t\* b = (**const** book\_t\*)rhs;

**return** a->year - b->year;

}

**int** **book\_compare\_by\_code**(**const** **void**\* lhs, **const** **void**\* rhs) {

**const** book\_t\* a = (**const** book\_t\*)lhs;

**const** book\_t\* b = (**const** book\_t\*)rhs;

**return** **strcmp**(a->code, b->code);

}

**int** **book\_compare\_by\_author**(**const** **void**\* lhs, **const** **void**\* rhs) {

**const** book\_t\* a = (**const** book\_t\*)lhs;

**const** book\_t\* b = (**const** book\_t\*)rhs;

**return** **strcmp**(a->author, b->author);

}

**int** **book\_compare\_by\_name**(**const** **void**\* lhs, **const** **void**\* rhs) {

**const** book\_t\* a = (**const** book\_t\*)lhs;

**const** book\_t\* b = (**const** book\_t\*)rhs;

**return** **strcmp**(a->name, b->name);

}

**main.c**

// Личная библиотека

**#include** "bilist.h"

**#include** "book.h"

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <string.h>

**#define** MENU\_CREATE 1 // добавление новой записи в список

**#define** MENU\_UPDATE 2 // удаление записи из списка

**#define** MENU\_DELETE 3 // редактирование записи в списке

**#define** MENU\_SORT 4 // сортировка списка

**#define** MENU\_OUTPUT 5 // вывод на экран содержимого списка

**#define** MENU\_LOAD 6 // загрузка данных из файла в список

**#define** MENU\_SAVE 7 // запись списка в файл

**#define** MENU\_SEARCH 8 // извлечение информации из списка по указанным критериям

**#define** MENU\_EXIT 9 // выход из программы

**#define** MAX\_FILE\_NAME 256

// проверка выхода индекса за границы диапазона

**int** **check\_index**(**unsigned** count, **unsigned** index) {

**if**(count == 0)

**return** 0;

**if**(index >= count) {

**printf**("Индекс может быть от 0 до %u\n", count - 1);

**fflush**(stdout);

**return** 0;

}

**return** 1;

}

// точка входа программы

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

{

**unsigned** i, menu;

book\_t book;

FILE\* file;

**char** file\_name[MAX\_FILE\_NAME];

bilist\_t\* books = bilist\_create();

**do** {

**printf**("[%d]\tДобавление новой записи в список\n", MENU\_CREATE);

**printf**("[%d]\tРедактирование записи в списке\n", MENU\_UPDATE);

**printf**("[%d]\tУдаление записи из списка\n", MENU\_DELETE);

**printf**("[%d]\tСортировка списка\n", MENU\_SORT);

**printf**("[%d]\tВывод на экран содержимого списка\n", MENU\_OUTPUT);

**printf**("[%d]\tЗагрузка данных из файла в список\n", MENU\_LOAD);

**printf**("[%d]\tЗапись списка в файл\n", MENU\_SAVE);

**printf**("[%d]\tИзвлечение информации из списка по указанным критериям\n", MENU\_SEARCH);

**printf**("[%d]\tВыход\n", MENU\_EXIT);

**printf**("Выберите действие: ");

**fflush**(stdout);

**scanf**("%u", &menu);

**switch**(menu) {

**case** MENU\_CREATE:

bilist\_push\_back(books, input\_book(stdin, TRUE));

**break**;

**case** MENU\_UPDATE:

**printf**("Введите индекс элемента: ");

**fflush**(stdout);

**scanf**("%u", &i);

**if**(!check\_index(bilist\_size(books), i))

**break**;

bilist\_set\_data(books, bilist\_item(books, i), input\_book(stdin, TRUE), book\_destroy);

**break**;

**case** MENU\_DELETE:

**printf**("Введите индекс элемента: ");

**fflush**(stdout);

**scanf**("%u", &i);

**if**(!check\_index(bilist\_size(books), i))

**break**;

bilist\_remove(books, bilist\_item(books, i), book\_destroy);

**break**;

**case** MENU\_SORT:

**printf**("Введите индекс поля, по которому будет выполняться поиск (0 - год, 1 - шифр, 2 - фамилия автора, 3 - название книги): ");

**fflush**(stdout);

**scanf**("%u", &i);

**switch**(i) {

**case** 0:

bilist\_bubble\_sort(books, book\_compare\_by\_year);

bilist\_print(books, print\_book, stdout, "Список книг");

**break**;

**case** 1:

bilist\_bubble\_sort(books, book\_compare\_by\_code);

bilist\_print(books, print\_book, stdout, "Список книг");

**break**;

**case** 2:

bilist\_bubble\_sort(books, book\_compare\_by\_author);

bilist\_print(books, print\_book, stdout, "Список книг");

**break**;

**case** 3:

bilist\_bubble\_sort(books, book\_compare\_by\_name);

bilist\_print(books, print\_book, stdout, "Список книг");

**break**;

**default**:

**printf**("Номер может быть от 0 до 3\n");

}

**fflush**(stdout);

**break**;

**case** MENU\_OUTPUT:

bilist\_print(books, print\_book, stdout, "Список книг");

**break**;

**case** MENU\_LOAD:

**printf**("Введите имя файла: ");

**fflush**(stdout);

**scanf** ("%255s", file\_name);

file = **fopen**(file\_name, "rt");

**if**(!file) {

**fprintf**(stderr, "Не удалось открыть файл \"%s\" для записи\n", file\_name);

**break**;

}

bilist\_clear(books, book\_destroy);

bilist\_read(books, read\_book, file);

**fclose**(file);

**break**;

**case** MENU\_SAVE:

**printf**("Введите имя файла: ");

**fflush**(stdout);

**scanf** ("%255s", file\_name);

file = **fopen**(file\_name, "wt");

**if**(!file) {

**fprintf**(stderr, "Не удалось открыть файл \"%s\" для записи\n", file\_name);

**break**;

}

bilist\_print(books, print\_book, file, NULL);

**fclose**(file);

**break**;

**case** MENU\_SEARCH:

**printf**("Введите индекс поля, по которому будет выполняться поиск (0 - год, 1 - шифр, 2 - фамилия автора, 3 - название книги): ");

**fflush**(stdout);

**scanf**("%u", &i);

**switch**(i) {

**case** 0:

input\_book\_year(&book, stdin, TRUE);

print\_book(bilist\_find\_data(books, book\_compare\_year, &book.year), stdout);

**break**;

**case** 1:

input\_book\_code(&book, stdin, TRUE);

print\_book(bilist\_find\_data(books, book\_compare\_code, book.code), stdout);

**break**;

**case** 2:

input\_book\_author(&book, stdin, TRUE);

print\_book(bilist\_find\_data(books, book\_compare\_author, book.author), stdout);

**break**;

**case** 3:

input\_book\_name(&book, stdin, TRUE);

print\_book(bilist\_find\_data(books, book\_compare\_name, book.name), stdout);

**break**;

**default**:

**printf**("Номер может быть от 0 до 3\n");

}

**break**;

**case** MENU\_EXIT:

**printf**("Завершение работы\n");

**break**;

**default**:

**printf**("Неверное значение\n");

}

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

} **while**(menu != MENU\_EXIT);

bilist\_destroy(books, book\_destroy);

**return** 0;

}

# Описание работы программы

Разработанная программа позволяет управлять информацией о коллекции книг, содержащихся в домашней библиотеке. Каждая книга представлена её шифром, названием, годом выпуска, а также фамилией автора.

После запуска программы - из под у правления среды разработки либо из командной строки - на экран будет выведено пользовательское меню, состоящее из 9 пунктов, каждый из которых представляет собой команду по управлению библиотекой:

[1] Добавление новой записи в список

[2] Редактирование записи в списке

[3] Удаление записи из списка

[4] Сортировка списка

[5] Вывод на экран содержимого списка

[6] Загрузка данных из файла в список

[7] Запись списка в файл

[8] Извлечение информации из списка по указанным критериям

[9] Выход

Выберите действие: \_

Сразу после списка команд расположен запрос на ввод номера команды.

После выполнения любого из действия, программа повторяет на экране пользовательское меню и запрашивает следующее действие, и так до завершения работы.

Как следует из названия команд, программа позволяет добавлять, редактировать и удалять информацию о книгах, хранить всю эту информацию в файлах, а также осуществлять поиск записей и сортировку коллекции.

# Результаты работы программы

Начнём работу с загрузки данных из файла, в который для удобства уже занесены тестовые данные. Для этого после запуска программы выберем команду №6 и введём имя файла с данными - db.txt, как показано на рисунке 1. Файл db.txt содержит записи о четырёх книгах, которые можно после загрузки вывести на экран командой №5.

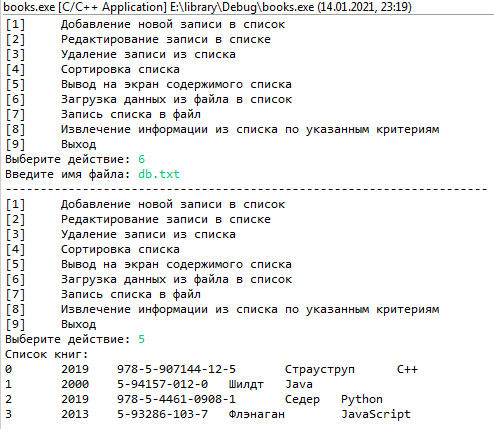


Рисунок 1 - загрузка данных из файла.

В дальнейшем, для отображения всех сделанных изменений, будем также пользоваться командой №5.

Следующим действием продемонстрируем работу команды №1 - добавление записи. При выборе этого пункта программа попросит ввести все поля добавляемой записи - год выпуска, шифр, автора и название, как показано на рисунке 3.

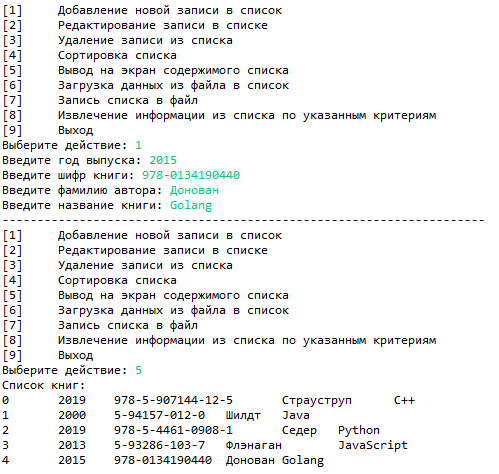


Рисунок 3 - добавление записи

Теперь база данных изменена с момента последней загрузки, так что сохраним её в файл командой №7. Для сохранения выберем тот же фай - db.txt - из которого она была загружена, рисунок 4.

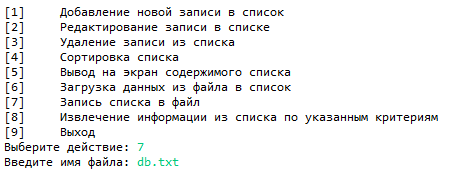


Рисунок 4- сохранение в файл

На рисунке 5 показано содержимое только что сохранённого файла. На последней строке расположена добавленная ранее запись.

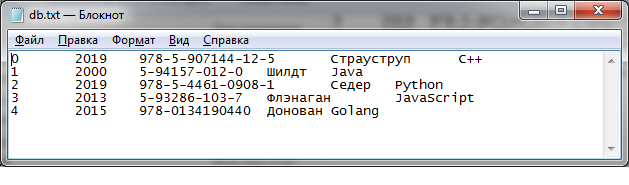


Рисунок 5- содержимое файла данных

За редактирование записи отвечает команда №2. При выборе этого пункта, программа сперва запросит индекс записи, данные которой необходимо отредактировать, а затем нужно будет ввести новые значения всех полей выбранной записи, как и при добавлении. Действия по редактированию записи с индексом 2 показаны на рисунке 6.

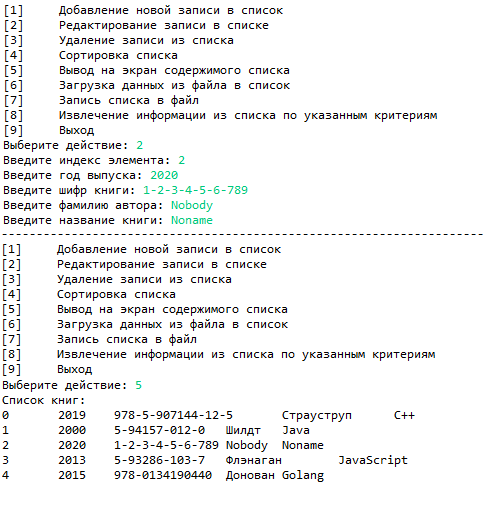


Рисунок 6 - редактирование записи

Следующим пунктом идёт команда № 3, отвечающая за удаление записей. Для выполнения этого действия, кроме номера команды, потребуется также ввести индекс удаляемой записи, как показано на рисунке 7.

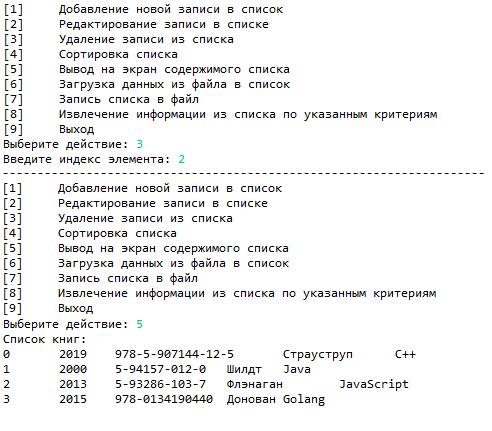


Рисунок 7 - удаление записи

Программа также позволяет упорядочить содержимое базы данных, причём сортировка может быть произведена по любому из имеющихся полей данных. Для этого предназначена команда №4. На рисунке 8 изображены две последовательные сортировки списка - сначала по шифру книги, затем по году выпуска.

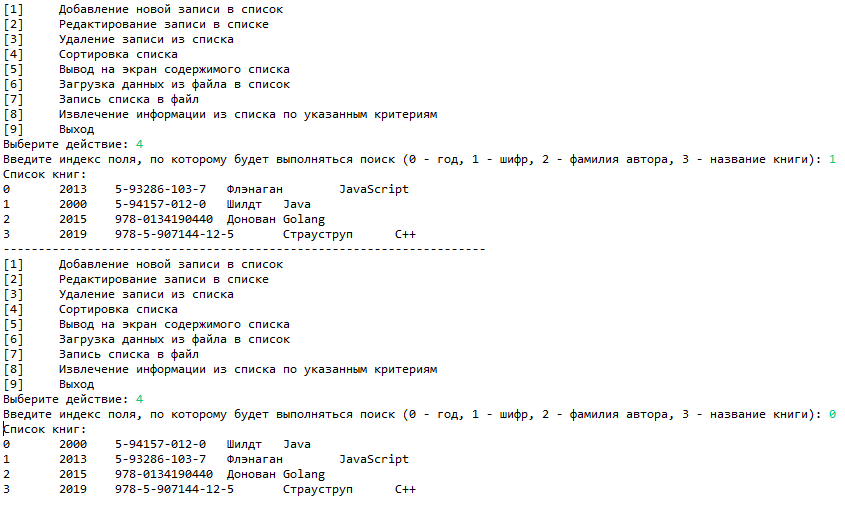


Рисунок 8 - сортировка и вывод упорядоченных данных.

Команда №8 позволяет найти в базе данных запись по любому из её полей. На рисунке 9 продемонстрирован поиск записи по фамилии автора книги.

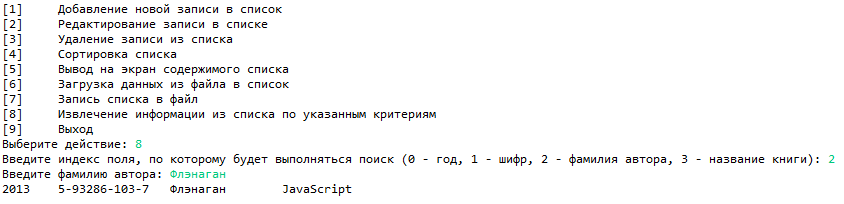


Рисунок 9 - Поиск записей по критерию

Команда №9 предназначена для завершения работы программы.