МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет “Львівська політехніка”

Інститут ІКНІ

Кафедра ПЗ



**ЗВІТ**

До лабораторної роботи № 2

З дисципліни : “Вступ до інженерії програмного забезпечення ”

**Лектор:**

ст. викл. каф. ПЗ

Левус Є.В.

**Виконав:**

ст. ПЗ-15

Марущак А.С.

**Прийняв**:

ассист. каф. ПЗ

Cамбір А. А.

“ ” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_

Львів-2022

**Тема.** Документування етапів проектування та кодування програми

**Мета.** Навчитися документувати основні результати етапів проектування та кодування найпростіших програм

**Теоретичні відомості**

**7.** **Які структури даних використовуються у вашій програмі? Які є альтернативні структури даних?**

У моїй програмі використовуються такі структури даних як зв'язний список та масив. Також є власне структури(одиниці мови С), які описують такі сутності, як команда, аругменти командного рядка, список, книгу, предмет списку. В якості альтернативи, можна було б використати таку стркутуру даних, як черга, для збереження списку команд, що йдуть на виконання. Для зберігання списків чудово б підійшла така структура даних як хеш-таблиця, проте враховуючи стиль написання коду мовою С, це було досить важко реалізувати.

**20.** **Які правила форматування конструкцій умов та циклів?**

Першочергово, для підвищення читабельності коду треба застосовувати відступи та табуляцію.

Форматування умов if-else - кожна команда починається після відступу у 2 пробіли, else пишеться в тому ж рядку, що і закриваюча дужка.

Форматування умов switch - кожна команда починається після відступу у 4 пробіли, а case – після відступу у 2 пробіли.

Форматування циклів - кожна команда починається після відступу у два пробіли, відкриваюча дужка пишеться у рядку заголовку циклу.

**36.** Що таке рефакторинг коду? Навіщо його виконувати?

Рефакторинг коду – один з типових процесів, що полягає у перетворенні програмного коду, зміні внутрішньої структури програмного забезпечення для полегшення розуміння коду і легшого внесення подальших змін без зміни зовнішньої поведінки самої системи. Цей процес полегшує роботу з кодом у майбутьному за рахунок зменшення його складності.

**Постановка завдання**

**Частина І.** У розробленій раніше програмі до лабораторної роботи з дисципліни «Основи програмування» внести зміни – привести її до модульної структури, де модуль – окрема функція-підпрограма. У якості таких функцій запрограмувати алгоритми зчитування та запису у файл, сортування, пошуку, редагування, видалення елементів та решта функцій згідно варіанту.

Частина ІІ. Сформувати пакет документів до розробленої раніше власної програми:

1. схематичне зображення структур даних, які використовуються для збереження інформації ;

2. блок-схема алгоритмів – основної функції й двох окремих функцій-підпрограм (наприклад, сортування та редагування);

3. текст програми з коментарями та оформлений згідно вище наведених рекомендацій щодо забезпечення читабельності й зрозумілості. Для схематичного зображення структур даних, блок-схеми алгоритму можна використати редактор MS-Visio або інший редактор інженерної та ділової графіки.

**Виконання роботи**

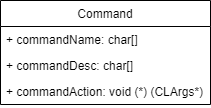
1. **Схематичне зображення структур даних, що використовуються у програмі.**



**Рис 2.1 Зв’язний список**

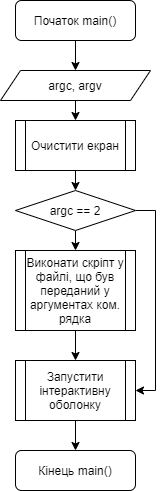


**Рис 2.2 Звичайний масив**



**Рис 2.3 Структура**

1. **Блок-схеми алгоритмів:**
   1. **Основної функції**

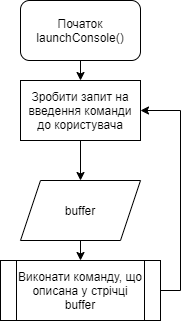


**Рис 2.4 Блок-схема основної функції**

* 1. **Двох додаткових функцій-підпрограм:**



**Рис 2.5 Блок-схема функції обробки скріптового файлу.**



**Рис 2.6 Блок-схема функції обробки інтерактивної оболонки.**

1. **Код програми після рефакторингу за правилами, вказаними у методичних вказівках.**

**lab10\_lib.h:**

#ifndef LAB10

#define LAB10

#include <stdarg.h>

#include <stddef.h>

//Макрос для знеаходження максимуму.

#define max(a, b) a > b ? a : b

//Макрос для створення стрічок з певною к-тю однакових симовлів.

#define REPEAT\_CHAR(ch, i) \

    for(int z = 0; z < (i); z++){\

        printf("%c", ch);\

    }\

//Макрос, що друкує стрічку за певним правилом.

#define MAKE\_LINE(plainCh, nodeCh, autS, titS, yearS, pageS, priceS)\

    printf("%c", nodeCh);\

    REPEAT\_CHAR(plainCh, (autS));\

    printf("%c", nodeCh);\

    REPEAT\_CHAR(plainCh, (titS));\

    printf("%c", nodeCh);\

    REPEAT\_CHAR(plainCh, (yearS));\

    printf("%c", nodeCh);\

    REPEAT\_CHAR(plainCh, (pageS));\

    printf("%c", nodeCh);\

    REPEAT\_CHAR(plainCh, (priceS));\

    printf("%c\n", nodeCh);

#define MAX\_NAME\_LEN 20

#define MAX\_SURNAME\_LEN 20

#define MAX\_COMMAND\_LEN 20

#define MAX\_ARGS\_COUNT 20

#define MAX\_DESCRIPTION\_LEN 1024

#define MAX\_LISTNAME\_LEN 20

#define MAX\_TITLE\_LEN 50

#define MAX\_LINE\_LEN 256

//Структура, що представляє дані про книгу

typedef struct Book

{

    char title[MAX\_TITLE\_LEN];

    struct Author

    {

        char name[MAX\_NAME\_LEN];

        char surname[MAX\_SURNAME\_LEN];

    }   author;

    double price;

    int pubYear;

    int pageCount;

} Book;

//Структура, що представляє дані про елемент списку.

typedef struct ListItem{

    Book data;

    struct ListItem \*next;

} ListItem;

//Структура, що представляє дані про список в цілому

typedef struct List{

    char name[MAX\_LISTNAME\_LEN];

    ListItem \*head;

} List;

//Структура, що представляє дані про аргументи команди: їх к-ть та значення.

typedef struct CLArgs{

    int argc;

    char \*argv[MAX\_ARGS\_COUNT];

} CLArgs;

//Структура, що представляє дані про команду.

typedef struct Command{

    char commandName[MAX\_COMMAND\_LEN];

    char commandDesc[MAX\_DESCRIPTION\_LEN];

    void (\*commandAction)(const CLArgs \*const args);

} Command;

typedef int (\*BookComparer)(Book a, Book b);

typedef int (\*BookPredicate)(Book a, va\_list va);

typedef double (\*BookSelector)(Book a, va\_list va);

CLArgs\* parseCommandLine(char \*str, char \*\*command);

void executeLine(char \*line);

void executeFile(char \*fileName);

void launchConsole();

void cls();

#endif

**lab10\_lib.c:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <ctype.h>

#include "lab10\_lib.h"

#pragma region Utils

// Функція, що повертає к-ть цифр у заданому числі.

int numLen(int num)

{

    int result = 0;

    while (num > 0)

    {

        num /= 10;

        result++;

    }

    return result;

}

//Функція, що повертає загальну довжину імені та прізвища автора з урахуванням пробілу між ними.

int getAuthorFullNameLen(const Book book)

{

    return strlen(book.author.name) + strlen(book.author.surname) + 1;

}

// Функція, що знаходить максимальну довжину імені та прізвища автора

// з урахуванням пробілу між ними з-поміж всіх книг у списку

int longestAuthorFieldSize(const List \*const books)

{

    if (!books)

        return 0;

    ListItem \*curr = books->head;

    int result = 0;

    while (curr)

    {

        int currLen = getAuthorFullNameLen(curr->data);

        result = currLen > result ? currLen : result;

        curr = curr->next;

    }

    return result;

}

//// Функція, що знаходить максимальну довжину назви книги з-поміж всіх книг у списку

int longestBookTitleSize(const List \*const books)

{

    if (!books)

        return 0;

    ListItem \*curr = books->head;

    int result = 0;

    while (curr)

    {

        int currLen = strlen(curr->data.title);

        if (currLen > result)

            result = currLen;

        curr = curr->next;

    }

    return result;

}

// Функція, що повертає максимальну довжину стрічки, що представляє рік видання,

// к-ті сторінок та ціни книги з-поміж всіх книг у списку

void longestNumberFieldsSize(const List \*const books, int \*const yearSize, int \*const pageSize, int \*const priceSize)

{

    if (!books)

    {

        \*yearSize = 0;

        \*pageSize = 0;

        \*priceSize = 0;

        return;

    }

    int

        maxYearLen = 0,

        maxPageLen = 0,

        maxPriceLen = 0,

        yearLen = 0,

        pageLen = 0,

        priceLen = 0;

    ListItem \*curr = books->head;

    while (curr)

    {

        yearLen = numLen(curr->data.pubYear);

        pageLen = numLen(curr->data.pageCount);

        priceLen = numLen((int)(curr->data.price)) + 3;

        if (yearLen > maxYearLen)

            maxYearLen = yearLen;

        if (pageLen > maxPageLen)

            maxPageLen = pageLen;

        if (priceLen > maxPriceLen)

            maxPriceLen = priceLen;

        curr = curr->next;

    }

    \*yearSize = maxYearLen;

    \*pageSize = maxPageLen;

    \*priceSize = maxPriceLen;

}

// Функція, що обраховує оптимальні розміри стовпців таблиці на основі отриманого списку.

void getSizes(const List \*const books, int \*const authorColumnWidth, int \*const titleColumnWidth, int \*const yearColumnWidth, int \*const pageColumnWidth, int \*const priceColumnWidth)

{

    \*authorColumnWidth = max(longestAuthorFieldSize(books), strlen("Author"));

    \*titleColumnWidth = max(longestBookTitleSize(books), strlen("Title"));

    longestNumberFieldsSize(books, yearColumnWidth, pageColumnWidth, priceColumnWidth);

    \*yearColumnWidth = max(\*yearColumnWidth, strlen("Pub. year"));

    \*pageColumnWidth = max(\*pageColumnWidth, strlen("Pages"));

    \*priceColumnWidth = max(\*priceColumnWidth, strlen("Price"));

}

// Функція, що обрізає всі пробіли з правого боку стрічки.

char \*rtrim(char \*s)

{

    char \*back = s + strlen(s);

    while (isspace(\*--back))

        ;

    \*(back + 1) = '\0';

    return s;

}

//Функція, що обрізає всі пробіли з лівого боку стрічки.

char \*ltrim(char \*s)

{

    while (isspace(\*s))

        s++;

    return s;

}

//Функція, що обрізає всі пробіли з обох кінців стрічки.

char \*trim(char \*s)

{

    return ltrim(rtrim(s));

}

//Функція що обрізає всі знаки лапок з обох кінців стрічки.

char \*qtrim(char \*s)

{

    char \*last = s + strlen(s) - 1;

    if (\*last == '"')

        \*last = 0;

    return \*s == '"' ? s + 1 : s;

}

//Функція, що запрошує користувача ввести певний текст, перед цим вивівши певне повідомлення

void prompt(const char const \*msg, char \*const buffer)

{

    printf(msg);

    gets(buffer);

    fflush(stdin);

}

// Перевірка існування файлу

int fexists(const char \*const fileName)

{

    FILE \*f = fopen(fileName, "r");

    if (f)

    {

        fclose(f);

        return 1;

    }

    return 0;

}

// Функція парсингу стрічки в структуру книги.

Book strToBook(char str[])

{

    Book result;

    char \*pPart = strtok(str, ",");

    sscanf(pPart, "%s %s", &result.author.name, &result.author.surname);

    pPart = strtok(NULL, ",");

    strcpy(result.title, pPart);

    pPart = strtok(NULL, ",");

    result.pubYear = atoi(pPart);

    pPart = strtok(NULL, ",");

    result.pageCount = atoi(pPart);

    pPart = strtok(NULL, ",");

    result.price = strtod(pPart, NULL);

    return result;

}

// Функція введення книги з консолі.

Book getBookFromConsole()

{

    char authorsName[MAX\_NAME\_LEN];

    char authorsSurname[MAX\_SURNAME\_LEN];

    char title[MAX\_TITLE\_LEN];

    int pubYear;

    int pageCount;

    double price;

    printf("Enter author(name surname): ");

    scanf("%s %s", authorsName, authorsSurname);

    fflush(stdin);

    printf("Enter book's title: ");

    gets(title);

    printf("Enter publication year: ");

    scanf("%d", &pubYear);

    printf("Enter page count: ");

    scanf("%d", &pageCount);

    printf("Enter price: ");

    scanf("%lf", &price);

    Book newBook;

    strcpy(newBook.title, trim(title));

    strcpy(newBook.author.name, authorsName);

    strcpy(newBook.author.surname, authorsSurname);

    newBook.price = price;

    newBook.pageCount = pageCount;

    newBook.pubYear = pubYear;

    return newBook;

}

// функція-перевірка, чи подана стрічка представляє собою число.

int isNumber(char \*str)

{

    str = trim(str);

    for (char \*p = str; \*p != 0; p++)

    {

        if (!isdigit(\*p) && \*p != '.')

            return 0;

    }

    return 1;

}

// Функція, яка перевіряє, чи стрічка починається з заданої підстрічки.

int startsWith(char \*str, char \*begining)

{

    if (strlen(begining) > strlen(str))

        return 0;

    for (int i = 0; i < strlen(begining); i++)

    {

        if (tolower(str[i]) != tolower(begining[i]))

            return 0;

    }

    return 1;

}

// Перевірка валідності формату файлу.

int rightFileFormat(char \*fileName)

{

    char \*extension = strrchr(fileName, '.');

    if (extension)

    {

        if (!strcmp(extension, ".lsexe"))

            return 1;

    }

    return 0;

}

//Перевірка валідності формату і існування файлу.

int isCorrectScriptFile(char \*fileName)

{

    return fexists(fileName) && rightFileFormat(fileName);

}

//Функція очистки екрану.

void cls()

{

    system("cls");

}

#pragma endregion

#pragma region Comparers

//Функція-компаратор, що порівнює книги за ім'ям автора.

int authorsNameBookComparer(Book a, Book b)

{

    return strcmp(a.author.name, b.author.name);

}

//Функція-компаратор, що порівнює книги за прізвищем автора.

int authorsSurnameBookComparer(Book a, Book b)

{

    return strcmp(a.author.surname, b.author.surname);

}

//Функція-компаратор, що порівнює книги за назвою книги.

int titleBookComaprer(Book a, Book b)

{

    return strcmp(a.title, b.title);

}

//Функція-компаратор, що порівнює книги за роком видання.

int yearBookComparer(Book a, Book b)

{

    return a.pubYear == b.pubYear ? 0 : a.pubYear > b.pubYear ? 1

                                                              : -1;

}

//Функція-компаратор, що порівнює книги за к-тю сторінок.

int pageBookComparer(Book a, Book b)

{

    return a.pageCount == b.pageCount ? 0 : a.pageCount > b.pageCount ? 1

                                                                      : -1;

}

//Функція-компаратор, що порівнює книги за ціною.

int priceBookComparer(Book a, Book b)

{

    return a.price == b.price ? 0 : a.price > b.price ? 1

                                                      : -1;

}

#pragma endregion

#pragma region FiltraionPredicatesAndSelectors

//Функція, що повертає одне з полів книги в залежності від заданих параметрів.

double GetBooksField(Book a, va\_list va)

{

    char \*field = va\_arg(va, char \*);

    if (!strcmp(field, "price"))

    {

        return a.price;

    }

    if (!strcmp(field, "pages"))

    {

        return (double)a.pageCount;

    }

    if (!strcmp(field, "year"))

    {

        return (double)a.pubYear;

    }

    return 0;

}

// Функція-компаратор для порівнняня, чи число менше іншого числа, що представлено стрічкою

int lessThanS(Book a, va\_list va)

{

    double value = GetBooksField(a, va);

    va\_arg(va, char \*);

    char \*limStr = va\_arg(va, char \*);

    double lim = strtod(limStr, NULL);

    return value < lim;

}

// Функція-компаратор для порівнняня, чи число більше іншого числа, що представлено стрічкою

int moreThanS(Book a, va\_list va)

{

    double value = GetBooksField(a, va);

    va\_arg(va, char \*);

    char \*limStr = va\_arg(va, char \*);

    double lim = strtod(limStr, NULL);

    return value > lim;

}

// Функція, що повертає середнє значення ознаки книги, описаної селектором.

double getListAverage(List \*list, BookSelector selector, ...)

{

    if (!list || !list->head)

        return 0;

    va\_list va;

    va\_start(va, selector);

    ListItem \*iter = list->head;

    double res = 0;

    int count = 0;

    while (iter)

    {

        res += selector(iter->data, va);

        iter = iter->next;

        count++;

    }

    return res / count;

}

// Функція-компаратор для порівнняня, чи число менше іншого числа

int lessThanD(Book a, va\_list va)

{

    double value = GetBooksField(a, va);

    va\_arg(va, char \*);

    double lim = va\_arg(va, double);

    return value < lim;

}

// Функція-компаратор для порівнняня, чи число більше іншого числа

int moreThanD(Book a, va\_list va)

{

    double value = GetBooksField(a, va);

    va\_arg(va, char \*);

    double lim = va\_arg(va, double);

    return value > lim;

}

// Функція, що перевіряє, чи деяке поле книги починається з потрібної підстрічки.

int startsWithS(Book a, va\_list va)

{

    char \*field = va\_arg(va, char \*);

    char \*starting = va\_arg(va, char \*);

    char \*str = "";

    if (!strcmp(field, "title"))

        str = a.title;

    else if (!strcmp(field, "name"))

        str = a.author.name;

    else if (!strcmp(field, "surname"))

        str = a.author.surname;

    return startsWith(str, starting);

}

// Функція, що перевіряє, чи деяке поле книги не починається з потрібної підстрічки.

int notStartsWithS(Book a, va\_list va)

{

    return !startsWithS(a, va);

}

#pragma endregion

#pragma region CommandsDeclarations

void exitProgram(const CLArgs \*const args);

void clearConsole(const CLArgs \*const args);

void getHelp(const CLArgs \*const args);

void createList(const CLArgs \*const args);

void getLists(const CLArgs \*const args);

void addElement(const CLArgs \*const args);

void switchList(const CLArgs \*const args);

void formTable(const CLArgs \*const args);

void loadFromFile(const CLArgs \*const args);

void countList(const CLArgs \*const args);

void deleteList(const CLArgs \*const args);

void sortList(const CLArgs \*const args);

void filterList(const CLArgs \*const args);

void limitList(const CLArgs \*const args);

void insertElement(const CLArgs \*const args);

void saveToFile(const CLArgs \*const args);

#pragma endregion

#pragma region CommandsAndListsStuff

//Змінна для зберігання к-ті списків, якими керує програма

static int listCount = 0;

 //Вказівник на початок масиву списків.

static List \*lists = NULL;

//Індекс поточного списку в масиві.

static int currentListIndex = -1;

//Вказівник на поточний список в масиві.

static List \*currentList = NULL;

//Масив з декларацією команд, їх описом та вказівниками на потрібні функції-обробники.

static Command commands[] = {

    {"exit",

     "Closes the console.\n\n"

     "Syntax: exit [<modifiers>]\n\n"

     "Modifiers:\n\n"

     "\t-s, --success - to set exit code to SUCCESS(default).\n"

     "\t-f, --fail - to set exit code to FAILURE.\n\n",

     exitProgram},

    {"cls",

     "Clears the console.\n\n",

     clearConsole},

    {"help",

     "Prints an information about selected command.\n\n"

     "Syntax: help [<command>]\n\n"

     "\t<command> - command we need info about\n\n",

     getHelp},

    {"create",

     "Creates a new list with specified name.\n\n"

     "Syntax: create [<modifiers>] [<name>]\n\n"

     "\t<name> - name of the list that will be created.\n\n"

     "Modifiers:\n\n"

     "\t-s, --select - selects created list as current.\n\n",

     createList},

    {"lists",

     "Prints off all available lists.\n\n",

     getLists},

    {"append",

     "Adds element to the current list.\n\n",

     addElement},

    {"show",

     "Shows the current list in table view.\n\n",

     formTable},

    {"switch",

     "Switches the current list to another with specified name.\n\n"

     "Syntax: switch [<name>]\n\n"

     "\t<name> - name of the list to switch to.\n\n",

     switchList},

    {"load",

     "Loads elements from specified file to the current list.\n\n"

     "Syntax: load [<modifiers>] [<file>]\n\n"

     "\t<file> - name of the file to load data from.\n\n"

     "Modifiers:\n\n"

     "\t-t, --text - to specify that the file type is text(default).\n"

     "\t-b, --binary - to specify that the file type is binary.\n\n",

     loadFromFile},

    {"count",

     "Counts the elements in the current list.\n\n",

     countList},

    {"delete",

     "Deletes the list with specified name.\n\n"

     "Syntax: delete [<name>]\n\n"

     "\t<name> - name of the list to be deleted.\n\n",

     deleteList},

    {"sortby",

     "Sorts list by the given parameters.\n\n"

     "Syntax: sortby [<modifiers>] <target>\n\n"

     "\t<target> - field of book which will be compared. \n"

     "\tCan be one from this:\n"

     "\tname\n\tsurname\n\ttitle\n\tyear\n\tpages\n\tprice\n\n"

     "Modifiers:\n\n"

     "\t-d, --descending - to specify that the list must be sorted in descending order.\n\n",

     sortList},

    {"filter",

     "Removes elements by specified criteria.\n\n"

     "Syntax: filter <mode> <target>\n\n"

     "\t<target> - field of book which will be checked. \n"

     "\tCan be one from this:\n"

     "\tname\n\tsurname\n\ttitle\n\tyear\n\tpages\n\tprice\n\n"

     "Modes:\n\n"

     "\tFor name, surname, title:\n"

     "\t\t--startswith <arg> - if element's target field's string starts with <arg> it will be deleted.\n"

     "\t\t--notstartswith <arg> - if element's target field's string not starts with <arg> it will be deleted.\n\n"

     "\tFor year, pages, price:\n"

     "\t\t--lessthan <arg> - if element's target field's value less than <arg> it will be deleted.\n"

     "\t\t--morethan <arg> - if element's target field's value more than <arg> it will be deleted.\n"

     "\t\t--belowaverage - if element's target field's value less than average it will be deleted.\n"

     "\t\t--aboveaverage - if element's target field's value more than average it will be deleted.\n\n",

     filterList},

    {"limit",

     "Limits current list to specified number of elements.\n\n"

     "Syntax: limit <number>\n\n"

     "\t<number> - number of elements in resulting list.\n\n",

     limitList},

    {"insert",

     "Inserts element at the specified position.\n\n",

     insertElement},

    {"save",

     "Saves elements from the current list to the specified file.\n\n"

     "Syntax: save [<modifiers>] [<file>]\n\n"

     "\t<file> - name of the file to save data to.\n\n"

     "Modifiers:\n\n"

     "\t-t, --text - to specify that the file type is text(default).\n"

     "\t-b, --binary - to specify that the file type is binary.\n\n",

     saveToFile}};

//Функція, що повертає вказівник на потрібну команду зі списку команд.

Command \*findCommand(const char \*const commandName)

{

    for (int i = 0; i < sizeof(commands) / sizeof(commands[0]); i++)

    {

        if (strcmp(commands[i].commandName, commandName) == 0)

            return commands + i;

    }

    return NULL;

}

//Функція, що перевіряє наявність заданого параметру у списку параметрів.

int findParam(const CLArgs \*const args, const char \*const param)

{

    if (!args)

        return -1;

    for (int i = 0; i < args->argc; i++)

    {

        if (strcmp(args->argv[i], param) == 0)

            return i;

    }

    return -1;

}

//Функція, що додає новий елемент на вказану позицію списку.

void addElementToList(List \*list, ListItem \*listItem, int pos)

{

    ListItem \*iter = list->head;

    if (!iter)

        list->head = listItem;

    else if (pos == 0)

    {

        listItem->next = list->head;

        list->head = listItem;

    }

    else

    {

        int currPos = 0;

        while (iter->next != NULL && currPos + 1 < pos)

        {

            iter = iter->next;

            currPos++;

        }

        listItem->next = iter->next;

        iter->next = listItem;

    }

}

//Функція, що повертає індекс списку в масиві за його іменем.

int findList(const char \*const listName)

{

    for (int i = 0; i < listCount; i++)

    {

        if (strcmp((lists + i)->name, listName) == 0)

        {

            return i;

        }

    }

    return -1;

}

//Функція, що видаляє список.

void deleteListRecursively(List \*list)

{

    ListItem \*iter = list->head;

    ListItem \*next = NULL;

    while (iter)

    {

        next = iter->next;

        free(iter);

        iter = next;

    }

}

//Функція, що підраховує к-ть елементів у списку.

int countListElements(List \*list)

{

    if (!list)

        return 0;

    int res = 0;

    for (ListItem \*iter = currentList->head; iter != NULL; iter = iter->next)

        res++;

    return res;

}

//Функція, що сортує список, використовуючи вказаний компаратор і у вказаному порядку.

void sortListUsingComparer(List \*list, BookComparer comparer, int descending)

{

    int size = countListElements(list);

    if (!list || size == 0 || size == 1)

        return;

    ListItem \*iter;

    Book temp;

    for (int i = 0; i < size; i++)

    {

        iter = list->head;

        while (iter->next)

        {

            if ((!descending && comparer(iter->data, iter->next->data) > 0) || (descending && comparer(iter->data, iter->next->data) < 0))

            {

                temp = iter->data;

                iter->data = iter->next->data;

                iter->next->data = temp;

            }

            iter = iter->next;

        }

    }

}

//Функція, що фільтрує список використовуючи певний предикат і вказані параметри.

void filterListBy(List \*list, BookPredicate predicate, ...)

{

    if (!list || !list->head)

        return;

    va\_list va;

    va\_start(va, predicate);

    ListItem \*iter = list->head, \*old, \*new;

    while (list->head && predicate(list->head->data, va))

    {

        old = list->head;

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

        free(old);

    }

    while (iter)

    {

        while (iter->next && predicate(iter->next->data, va))

        {

            old = iter->next;

            iter->next = iter->next->next;

            free(old);

        }

        iter = iter->next;

    }

}

#pragma endregion

#pragma region CommandsRealizations

//Функція-команда виходу з програми.

void exitProgram(const CLArgs \*const args)

{

    for (int i = 0; i < listCount; i++)

    {

        deleteListRecursively(lists + i);

    }

    free(lists);

    printf("Aborting a program..\n");

    if (!args || args->argc == 0 || findParam(args, "-s") != -1 || findParam(args, "--success") != -1)

        exit(EXIT\_SUCCESS);

    else if (findParam(args, "-f") != -1 || findParam(args, "--fail") != -1)

        exit(EXIT\_FAILURE);

}

//Функція-команда очищення консолі.

void clearConsole(const CLArgs \*const args)

{

    cls();

}

//Функція-команда отримання довідки про інші команди.

void getHelp(const CLArgs \*const args)

{

    if (!args || args->argc == 0)

    {

        printf("There are %d commands available:\n\n", sizeof(commands) / sizeof(commands[0]));

        for (int i = 0; i < sizeof(commands) / sizeof(commands[0]); i++)

        {

            printf("\t%s\n", commands[i].commandName);

        }

        printf("\nTo get the description of the command use \"help <command>\".\n\n");

        return;

    }

    if (args->argc == 1)

    {

        Command \*command = findCommand(args->argv[0]);

        if (command)

        {

            printf(command->commandDesc);

            printf("\n");

        }

        else

        {

            printf("Unknown command \"%s\"\n\n", args->argv[0]);

            return;

        }

    }

    else

    {

        printf("Only 1 parameter needed.\n\n");

        return;

    }

}

//Функція-команда створення нового списку.

void createList(const CLArgs \*const args)

{

    if (args && args->argc >= 3)

    {

        printf("Too many parameters.\n\n");

        return;

    }

    List newList;

    newList.head = NULL;

    int selPos = findParam(args, "-s") == -1 ? findParam(args, "--select") : findParam(args, "-s");

    int select = selPos != -1;

    if (!args || args->argc == 0 || (args->argc == 1 && select))

    {

        char buffer[MAX\_LISTNAME\_LEN];

        prompt("Enter list name: ", buffer);

        char \*listName = trim(buffer);

        if (strlen(listName) > 0)

            strcpy(newList.name, listName);

        else

        {

            printf("List name must contain text characters!\n\n");

            return;

        }

    }

    else

    {

        strcpy(newList.name, selPos == 0 ? args->argv[1] : args->argv[0]);

    }

    if (findList(newList.name) != -1)

    {

        printf("List with the name \"%s\" already exists.\n\n");

        return;

    }

    listCount++;

    List \*newListsArr = (List \*)realloc(lists, sizeof(List) \* listCount);

    if (!newListsArr)

    {

        printf("Error creating a list.\n");

        return;

    }

    newListsArr[listCount - 1] = newList;

    lists = newListsArr;

    printf("List \"%s\" successfully created.\n", newList.name);

    if (select || !currentList)

    {

        printf("Switching to new list.\n");

        currentListIndex = listCount - 1;

    }

    currentList = lists + currentListIndex;

    printf("\n");

}

//Функція-команда отримання назв всіх списків, наявних у програмі.

void getLists(const CLArgs \*const args)

{

    if (listCount == 0)

    {

        printf("There are no available lists. Create one by executing \"create\" command.\n\n");

        return;

    }

    printf("There are %d available lists: \n\n", listCount);

    for (int i = 0; i < listCount; i++)

    {

        printf("\t%s\n", lists[i].name);

    }

    printf("\n");

}

//Функція-команда додавання нової книги до списку з консолі.

void addElement(const CLArgs \*const args)

{

    if (!currentList)

    {

        printf("No list to add element to. Create one by executing \"create\" command.\n\n");

        return;

    }

    if (args->argc > 0)

    {

        printf("Unknown modifiers.\n\n");

        return;

    }

    ListItem \*newLI = (ListItem \*)malloc(sizeof(ListItem));

    if (!newLI)

    {

        printf("Error creating list item.\n\n");

        return;

    }

    Book newBook = getBookFromConsole();

    newLI->data = newBook;

    newLI->next = NULL;

    ListItem \*iter = currentList->head;

    addElementToList(currentList, newLI, countListElements(currentList));

    printf("Element successfuly added to list \"%s\"\n\n", currentList->name);

}

//Функція-команда виводу списку у вигляді таблиці.

void formTable(const CLArgs \*const args)

{

    if (!currentList)

    {

        printf("There are no list to display. Create one by executing \"create\" command.\n\n");

        return;

    }

    ListItem \*curr = currentList->head;

    if (!curr)

    {

        printf("No elements in the list \"%s\".\n\n", currentList->name);

        return;

    }

    int authorColumnWidth,

        titleColumnWidth,

        yearColumnWidth,

        pageColumnWidth,

        priceColumnWidth;

    getSizes(currentList, &authorColumnWidth, &titleColumnWidth,

             &yearColumnWidth, &pageColumnWidth, &priceColumnWidth);

    printf("\n");

    MAKE\_LINE('=', '+', authorColumnWidth + 2, titleColumnWidth + 2, yearColumnWidth + 2, pageColumnWidth + 2, priceColumnWidth + 2);

    printf("| %-\*s | %-\*s | %-\*s | %-\*s | %-\*s |\n",

           authorColumnWidth, "Author",

           titleColumnWidth, "Title",

           yearColumnWidth, "Pub. year",

           pageColumnWidth, "Pages",

           priceColumnWidth, "Price");

    MAKE\_LINE('=', '+', authorColumnWidth + 2, titleColumnWidth + 2, yearColumnWidth + 2, pageColumnWidth + 2, priceColumnWidth + 2);

    char fullName[MAX\_NAME\_LEN + MAX\_SURNAME\_LEN + 1];

    while (curr)

    {

        sprintf(fullName, "%s %s", curr->data.author.name, curr->data.author.surname);

        printf("| %-\*s | %-\*s | %-\*d | %-\*d | %-\*.2lf |\n",

               authorColumnWidth, fullName,

               titleColumnWidth, curr->data.title,

               yearColumnWidth, curr->data.pubYear,

               pageColumnWidth, curr->data.pageCount,

               priceColumnWidth, curr->data.price);

        curr = curr->next;

    }

    MAKE\_LINE('=', '+', authorColumnWidth + 2, titleColumnWidth + 2, yearColumnWidth + 2, pageColumnWidth + 2, priceColumnWidth + 2);

    printf("\n");

}

//Функція-команда переключення списків.

void switchList(const CLArgs \*const args)

{

    if (!currentList)

    {

        printf("There no lists to switch between. Create one by executing \"create\" command.\n\n");

        return;

    }

    if (args && args->argc >= 2)

    {

        printf("Too many parameters.\n\n");

        return;

    }

    char listName[MAX\_LISTNAME\_LEN];

    if (!args || args->argc == 0)

    {

        prompt("Enter list name to switch on: ", listName);

    }

    else if (args->argc == 1)

    {

        strcpy(listName, args->argv[0]);

    }

    char \*listNameTrimmed = trim(listName);

    if (strlen(listNameTrimmed) == 0)

    {

        printf("Name of list cannot be empty.\n\n");

        return;

    }

    if (!strcmp(listNameTrimmed, currentList->name))

    {

        printf("Already at this list.\n\n");

        return;

    }

    int listIndex = -1;

    if ((listIndex = findList(listName)) != -1)

    {

        printf("Switching to list \"%s\".\n\n", listName);

        currentListIndex = listIndex;

        currentList = lists + currentListIndex;

        return;

    }

    printf("No list with name \"%s\".\n\n", listName);

}

//Функція-команда завантаження списку з файлу.

void loadFromFile(const CLArgs \*const args)

{

    if (!currentList)

    {

        printf("No list to load data to. Create one by executing \"create\" command.\n\n");

        return;

    }

    char filename[MAX\_LINE\_LEN];

    int formatPointerPos = -1;

    int textFormat = 1;

    if (!args || args->argc == 0)

    {

        prompt("Enter name of file data will be loaded from: ", filename);

    }

    else if (args->argc >= 3)

    {

        printf("Too many parameters.\n\n");

        return;

    }

    else if (args->argc == 1)

    {

        if (((formatPointerPos = findParam(args, "-t")) != -1) || ((formatPointerPos = findParam(args, "--text")) != -1))

        {

            textFormat = 1;

            prompt("Enter name of file data will be loaded from: ", filename);

        }

        else if (((formatPointerPos = findParam(args, "-b")) != -1) || ((formatPointerPos = findParam(args, "--binary")) != -1))

        {

            textFormat = 0;

            prompt("Enter name of file data will be loaded from: ", filename);

        }

        else

        {

            strcpy(filename, args->argv[0]);

        }

    }

    else

    {

        if (((formatPointerPos = findParam(args, "-t")) != -1) || ((formatPointerPos = findParam(args, "--text")) != -1))

        {

            textFormat = 1;

        }

        else if (((formatPointerPos = findParam(args, "-b")) != -1) || ((formatPointerPos = findParam(args, "--binary")) != -1))

        {

            textFormat = 0;

        }

        else

        {

            printf("Unknown parameter.\n\n");

            return;

        }

        strcpy(filename, args->argv[formatPointerPos == 0 ? 1 : 0]);

    }

    char \*fileNameTrimmed = trim(filename);

    if (strlen(fileNameTrimmed) == 0)

    {

        printf("The name of file cannot be empty.\n\n");

        return;

    }

    if (fexists(fileNameTrimmed))

    {

        FILE \*fin = NULL;

        Book data;

        ListItem \*newLI = NULL;

        int elemCount = 0;

        if (textFormat)

        {

            fin = fopen(fileNameTrimmed, "r");

            char buffer[MAX\_LINE\_LEN];

            while (fgets(buffer, MAX\_LINE\_LEN, fin) != NULL)

            {

                if (strlen(buffer) <= 5)

                    continue;

                data = strToBook(buffer);

                newLI = (ListItem \*)malloc(sizeof(ListItem));

                newLI->data = data;

                newLI->next = NULL;

                addElementToList(currentList, newLI, elemCount);

                elemCount++;

            }

        }

        else

        {

            FILE \*fin = fopen(fileNameTrimmed, "rb");

            while (!feof(fin))

            {

                fread(&data, sizeof(Book), 1, fin);

                if (!feof(fin))

                {

                    newLI = (ListItem \*)malloc(sizeof(ListItem));

                    newLI->data = data;

                    newLI->next = NULL;

                    addElementToList(currentList, newLI, elemCount);

                    elemCount++;

                }

            }

        }

        if (elemCount)

        {

            printf("%d elements was successfully read from file \"%s\".\n\n", elemCount, fileNameTrimmed);

        }

        else

        {

            printf("File was opened, but there are no elements to read.\n\n");

        }

        if (fin)

            fclose(fin);

    }

    else

    {

        printf("File \"%s\" not exists.\n\n", fileNameTrimmed);

    }

}

//Функція-команда знаходженння к-ті елементів у списку.

void countList(const CLArgs \*const args)

{

    if (!currentList)

    {

        printf("There are no list. Create one by executing \"create\" command.\n\n");

        return;

    }

    int res = countListElements(currentList);

    if (res == 0)

        printf("There are no elements in list.\n\n");

    else

        printf("There are %d elements in list.\n\n", res);

}

//Функція-команда видалення списку.

void deleteList(const CLArgs \*const args)

{

    if (!currentList)

    {

        printf("No lists to delete. Create one by executing \"create\" command.\n\n");

        return;

    }

    if (args && args->argc >= 2)

    {

        printf("Too many parameters.\n\n");

        return;

    }

    char listName[MAX\_LISTNAME\_LEN];

    if (!args || args->argc == 0)

    {

        prompt("Enter name of the list to be deleted: ", listName);

    }

    else if (args->argc == 1)

    {

        strcpy(listName, args->argv[0]);

    }

    char \*listNameTrimmed = trim(listName);

    if (strlen(listNameTrimmed) == 0)

    {

        printf("List name cannot be empty.\n\n");

        return;

    }

    int listIndex = findList(listNameTrimmed);

    if (listIndex == -1)

    {

        printf("No list with name \"%s\".\n\n", listName);

        return;

    }

    deleteListRecursively(lists + listIndex);

    for (int i = listIndex + 1; i < listCount; i++)

    {

        lists[i - 1] = lists[i];

    }

    listCount--;

    List \*newListArr = listCount ? realloc(lists, listCount) : NULL;

    lists = newListArr;

    if (!lists)

        currentListIndex = -1;

    else if (currentListIndex == listIndex)

    {

        currentListIndex = 0;

    }

    else if (currentListIndex > listIndex)

        currentListIndex--;

    currentList = lists ? lists + currentListIndex : NULL;

    printf("Successfuly deleted list with name \"%s\".\n\n", listName);

}

//Функція-команда сортування списку.

void sortList(const CLArgs \*const args)

{

    if (!currentList)

    {

        printf("No list to sort. Create one by executing \"create\" command.\n\n");

        return;

    }

    if (!args || args->argc == 0)

    {

        printf("Specify sort's target. To get list of available targets execute \"help sortby\".\n\n");

        return;

    }

    if (args->argc >= 3)

    {

        printf("Too many parameters.\n\n");

        return;

    }

    char \*sortTarget;

    int descending = 0;

    if (args->argc == 1)

    {

        sortTarget = args->argv[0];

    }

    else if (args->argc == 2)

    {

        int desPos = -1;

        if ((desPos = findParam(args, "-d")) != -1 || (desPos = findParam(args, "--descending")) != -1)

        {

            descending = 1;

            sortTarget = args->argv[desPos == 0 ? 1 : 0];

        }

    }

    if (!strcmp(sortTarget, "price"))

        sortListUsingComparer(currentList, priceBookComparer, descending);

    else if (!strcmp(sortTarget, "name"))

        sortListUsingComparer(currentList, authorsNameBookComparer, descending);

    else if (!strcmp(sortTarget, "surname"))

        sortListUsingComparer(currentList, authorsSurnameBookComparer, descending);

    else if (!strcmp(sortTarget, "pages"))

        sortListUsingComparer(currentList, pageBookComparer, descending);

    else if (!strcmp(sortTarget, "year"))

        sortListUsingComparer(currentList, yearBookComparer, descending);

    else if (!strcmp(sortTarget, "title"))

        sortListUsingComparer(currentList, titleBookComaprer, descending);

    else

    {

        printf("Specify correct sort's target. To get list of available targets execute \"help sortby\".\n\n");

        return;

    }

    printf("List sorted successfully.\n\n");

}

//Функція-команда фільтрування списку.

void filterList(const CLArgs \*const args)

{

    if (!currentList)

    {

        printf("No list to filter. Create one by executing \"create\" command.\n\n");

        return;

    }

    if (!args || args->argc > 3 || args->argc < 2)

    {

        printf("Incorrect number of parameters.\n\n");

        return;

    }

    char \*target = "", \*mode = "", \*arg = "";

    int modePos;

    if ((modePos = findParam(args, "--lessthan")) != -1 ||

        (modePos = findParam(args, "--morethan")) != -1 ||

        (modePos = findParam(args, "--startswith")) != -1 ||

        (modePos = findParam(args, "--notstartswith")) != -1)

    {

        mode = args->argv[modePos];

        if (args->argc != 3)

        {

            printf("Parameter for \"%s\" mode needed.\n\n", mode);

            return;

        }

        if (modePos == 2)

        {

            printf("Parameter must be after mode.\n\n");

            return;

        }

        arg = args->argv[modePos == 0 ? 1 : 2];

        target = args->argv[modePos == 0 ? 2 : 0];

    }

    else if ((modePos = findParam(args, "--belowaverage")) != -1 ||

             (modePos = findParam(args, "--aboveaverage")) != -1)

    {

        mode = args->argv[modePos];

        if (args->argc != 2)

        {

            printf("No need parameter for \"%s\" mode.\n\n", mode);

            return;

        }

        target = args->argv[modePos == 0 ? 1 : 0];

    }

    else

    {

        printf("Need to specify correct filter mode. To get a list of possible modes execute \"help filter\".\n\n");

        return;

    }

    if (!strcmp(target, "price") || !strcmp(target, "year") || !strcmp(target, "pages"))

    {

        if (!strcmp(mode, "--lessthan"))

        {

            if (isNumber(arg))

            {

                filterListBy(currentList, lessThanS, target, arg);

            }

            else

            {

                printf("Illegal argument for \"%s\" mode.\n\n", mode);

                return;

            }

        }

        else if (!strcmp(mode, "--morethan"))

        {

            if (isNumber(arg))

            {

                filterListBy(currentList, moreThanS, target, arg);

            }

            else

            {

                printf("Illegal argument for \"%s\" mode.\n\n", mode);

                return;

            }

        }

        else if (!strcmp(mode, "--belowaverage"))

        {

            double average = getListAverage(currentList, GetBooksField, target);

            printf("Average: %lf\n", average);

            filterListBy(currentList, lessThanD, target, average);

        }

        else if (!strcmp(mode, "--aboveaverage"))

        {

            double average = getListAverage(currentList, GetBooksField, target);

            printf("Average: %lf\n", average);

            filterListBy(currentList, moreThanD, target, average);

        }

        else

        {

            printf("\"%s\" mode is not compatible with \"%s\" field.\n\n", mode, target);

            return;

        }

    }

    else if (!strcmp(target, "title") || !strcmp(target, "name") || !strcmp(target, "surname"))

    {

        if (!strcmp(mode, "--startswith"))

        {

            filterListBy(currentList, startsWithS, target, arg);

        }

        else if (!strcmp(mode, "--notstartswith"))

        {

            filterListBy(currentList, notStartsWithS, target, arg);

        }

        else

        {

            printf("\"%s\" mode is not compatible with \"%s\" field.\n\n", mode, target);

            return;

        }

    }

    printf("Filter was successful.\n\n");

}

//Функція-команда обрізання списку.

void limitList(const CLArgs \*const args)

{

    if (!currentList)

    {

        printf("No list to limit. Create one by executing \"create\" command.\n\n");

        return;

    }

    if (!args || args->argc != 1)

    {

        printf("Illegal number of parameters.\n\n");

        return;

    }

    int lim = atoi(args->argv[0]);

    int size = countListElements(currentList);

    if (lim <= 0)

    {

        printf("Illegal parameter for limit.\n\n");

        return;

    }

    if (size <= lim)

    {

        printf("There are already less elements in list.\n\n");

        return;

    }

    ListItem \*iter = currentList->head;

    for (int i = 0; i < lim - 1; i++)

    {

        iter = iter->next;

    }

    ListItem \*old = iter->next, \*temp = iter->next->next;

    iter->next = NULL;

    while (old)

    {

        temp = old->next;

        free(old);

        old = temp;

    }

    printf("List limited successfuly.\n\n");

}

//Функція-команда додавання елементу в масив з консолі.

void insertElement(const CLArgs \*const args)

{

    if (!currentList)

    {

        printf("No list to insert element to. Create one by executing \"create\" command.\n\n");

        return;

    }

    if (args && args->argc != 0)

    {

        printf("No parameters needed.\n\n");

        return;

    }

    ListItem \*newLI = (ListItem \*)malloc(sizeof(ListItem));

    if (!newLI)

    {

        printf("Error creating new list item.\n\n");

        return;

    }

    Book newBook = getBookFromConsole();

    int pos = 0;

    printf("Enter position of this item in list: ");

    scanf("%d", &pos);

    if (pos <= 0)

    {

        printf("Illegal position.\n\n");

        return;

    }

    newLI->data = newBook;

    newLI->next = NULL;

    addElementToList(currentList, newLI, pos - 1);

    printf("Element was added successfully.\n\n");

}

//Функція-команда збереження списку у файл.

void saveToFile(const CLArgs \*const args)

{

    if (!currentList)

    {

        printf("No list to save data from. Create one by executing \"create\" command.\n\n");

        return;

    }

    char filename[MAX\_LINE\_LEN];

    int formatPointerPos = -1;

    int textFormat = 1;

    if (!args || args->argc == 0)

    {

        prompt("Enter name of file data will be saved to: ", filename);

    }

    else if (args->argc >= 3)

    {

        printf("Too many parameters.\n\n");

        return;

    }

    else if (args->argc == 1)

    {

        if (((formatPointerPos = findParam(args, "-t")) != -1) || ((formatPointerPos = findParam(args, "--text")) != -1))

        {

            textFormat = 1;

            prompt("Enter name of file data will be saved to: ", filename);

        }

        else if (((formatPointerPos = findParam(args, "-b")) != -1) || ((formatPointerPos = findParam(args, "--binary")) != -1))

        {

            textFormat = 0;

            prompt("Enter name of file data will be saved to: ", filename);

        }

        else

        {

            strcpy(filename, args->argv[0]);

        }

    }

    else

    {

        if (((formatPointerPos = findParam(args, "-t")) != -1) || ((formatPointerPos = findParam(args, "--text")) != -1))

        {

            textFormat = 1;

        }

        else if (((formatPointerPos = findParam(args, "-b")) != -1) || ((formatPointerPos = findParam(args, "--binary")) != -1))

        {

            textFormat = 0;

        }

        else

        {

            printf("Unknown parameter.\n\n");

            return;

        }

        strcpy(filename, args->argv[formatPointerPos == 0 ? 1 : 0]);

    }

    char \*fileNameTrimmed = trim(filename);

    if (strlen(fileNameTrimmed) == 0)

    {

        printf("File name cannot be empty.\n\n");

        return;

    }

    if (fexists(fileNameTrimmed))

    {

        printf("The file \"%s\" already exists.\n", fileNameTrimmed);

        char choice[3];

        prompt("Do you want to overwrite it?(Y|N) ", choice);

        if (!(tolower(\*choice) == 'y'))

        {

            printf("\n\n");

            return;

        }

    }

    FILE \*fout = NULL;

    if (textFormat)

    {

        fout = fopen(fileNameTrimmed, "w");

        if (!fout)

        {

            printf("Error accessing the file.\n\n");

            return;

        }

        ListItem \*iter = currentList->head;

        Book currentData;

        while (iter)

        {

            currentData = iter->data;

            fprintf(fout, "%s %s,%s,%d,%d,%.2lf\n",

                    currentData.author.name,

                    currentData.author.surname,

                    currentData.title,

                    currentData.pubYear,

                    currentData.pageCount,

                    currentData.price);

            iter = iter->next;

        }

    }

    else

    {

        fout = fopen(fileNameTrimmed, "wb");

        ListItem \*iter = currentList->head;

        Book currentData;

        while (iter)

        {

            currentData = iter->data;

            fwrite(&currentData, sizeof(Book), 1, fout);

            iter = iter->next;

        }

    }

    if (fout)

        fclose(fout);

    printf("Saving was successful.\n\n");

}

#pragma endregion

#pragma region Console

//Функція, що парсить стрічку і розбиває її на команду і на параметри команди.

CLArgs \*parseCommandLine(char \*str, char \*\*command)

{

    CLArgs \*result = (CLArgs \*)malloc(sizeof(CLArgs));

    result->argc = 0;

    str = trim(str);

    char \*commandArgSepPos = strchr(str, ' ');

    \*command = str;

    //if no args(no space after command)

    if (!commandArgSepPos)

    {

        return result;

    }

    \*commandArgSepPos = 0;

    int inQuote = 0;

    char \*lexemStart = ltrim(commandArgSepPos + 1);

    char \*curr = lexemStart;

    while (\*curr != 0)

    {

        if (!inQuote && \*curr == '"' && strchr(curr + 1, '"'))

        {

            inQuote = 1;

        }

        else if (\*curr == '"' && inQuote)

        {

            inQuote = 0;

        }

        if (!inQuote && isspace(\*curr))

        {

            \*curr = 0;

            result->argv[result->argc++] = qtrim(trim(lexemStart));

            curr = ltrim(curr + 1);

            lexemStart = curr;

        }

        else

        {

            curr++;

            if (\*curr == 0)

            {

                result->argv[result->argc++] = qtrim(trim(lexemStart));

            }

        }

    }

    return result;

}

//Функція, що виконує команди, вказані у стрічці.

void executeLine(char \*line)

{

    char buffer[20][MAX\_LINE\_LEN];

    char \*part = strtok(line, ";");

    int commandCount = 0;

    while (part)

    {

        if (strlen(part) > 1)

            strcpy(buffer[commandCount++], part);

        part = strtok(NULL, ";");

    }

    for (int i = 0; i < commandCount; i++)

    {

        char \*commandName;

        CLArgs \*args = parseCommandLine(buffer[i], &commandName);

        Command \*command;

        if (command = findCommand(commandName))

        {

            command->commandAction(args);

            fflush(stdin);

        }

        else

        {

            printf("Unknown command \"%s\"\nType \"help\" to get list of all available commands.\n\n", commandName);

        }

    }

}

//Функція, що виконує команди, вказані у файлі.

void executeFile(char \*fileName)

{

    if (!rightFileFormat(fileName))

    {

        printf("Use only .lsexe files.\n\n");

        return;

    }

    FILE \*fin = fopen(fileName, "r");

    if (!fin)

    {

        printf("Error opening file \"%s\".", fileName);

        return;

    }

    char buffer[MAX\_LINE\_LEN];

    char consoleText[MAX\_LISTNAME\_LEN + 2];

    while (fgets(buffer, MAX\_LINE\_LEN, fin))

    {

        sprintf(consoleText, "%s> ", currentList ? currentList->name : "(no lists)");

        executeLine(buffer);

    }

}

//Функція, що запускає інтерактивну оболонку.

void launchConsole()

{

    char buffer[MAX\_LINE\_LEN];

    while (1)

    {

        char consoleText[MAX\_LISTNAME\_LEN + 2];

        sprintf(consoleText, "%s> ", currentList ? currentList->name : "(no lists)");

        prompt(consoleText, buffer);

        executeLine(buffer);

    }

}

#pragma endregion

**lab10.c**

#include "lab10\_lib.h"

//Головна функція програми.

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

{

    cls();

    if (argc == 2)

    {

        executeFile(argv[1]);

    }

    launchConsole();

    return 0;

}

**Висновок:** за допомогою виконання лабораторної роботи, я навчився документувати етапи проектування та кодування програми. В результаті мною були розроблені блок-схеми основних алгоритмів програми, схематичні зображення структур коду. Також, був здійснений рефакторинг коду.