

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення**



ЗВІТ

До лабораторної роботи №2

На тему: «Документування етапів проектування та кодування програми»

З дисципліни: «Вступ до інженерії програмного забезпечення»

Лектор:

доц. кафедри ПЗ

Левус Є. В.

Виконав:

ст. гр. ПЗ-16

Чаус О. М.

Прийняв:

викладач кафедри ПЗ Самбір А. А.

« ____ » _____ 2022 р.

Σ = ____

Тема роботи: Документування етапів проектування та кодування програми.

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

Теоретичні відомості

33. Що таке статичний аналіз коду?

Статичний аналіз коду – ретельний перегляд коду з метою виявлення семантичних помилок.

19. Як зробити текст програми читабельним?

Щоб зробити текст програми читабельним, необхідно правильно формувати код, зокрема:

- використовувати правильну табуляцію
- не перевищувати стандарту довжину рядків(80 символів)
- розподіляти функції та методи коментарями довжиною 80 символів
- називати функції та змінні відповідно до конвенцій про імена(угорська нотація і т. д.)

6. Що таке шаблони проектування? Яка мета використання патернів проектування?

Шаблони проектування – це архітектурна конструкція, яка надає рішення проблеми проектування в певному контексті. Це – опис того, як можна вирішити завдання. Патери проектування використовуються для вирішення поширених проблем, пришвидшуючи процес розробки.

Постановка завдання

Частина I. 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 II. 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 функційпідпрограм (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.

Отримані результати

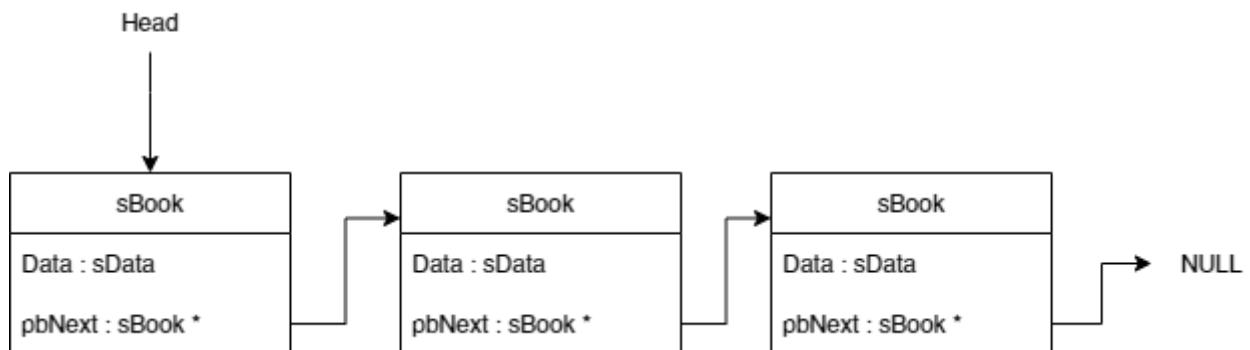
1. Схематичні зображення структур даних, які використовуються

1.1 Структури:

struct sData
m_szAuthorName : char[50]
m_szBookName : char[50]
m_szReleaseYear : char[50]
m_szPageCount : char[50]
m_szPrice : char[50]

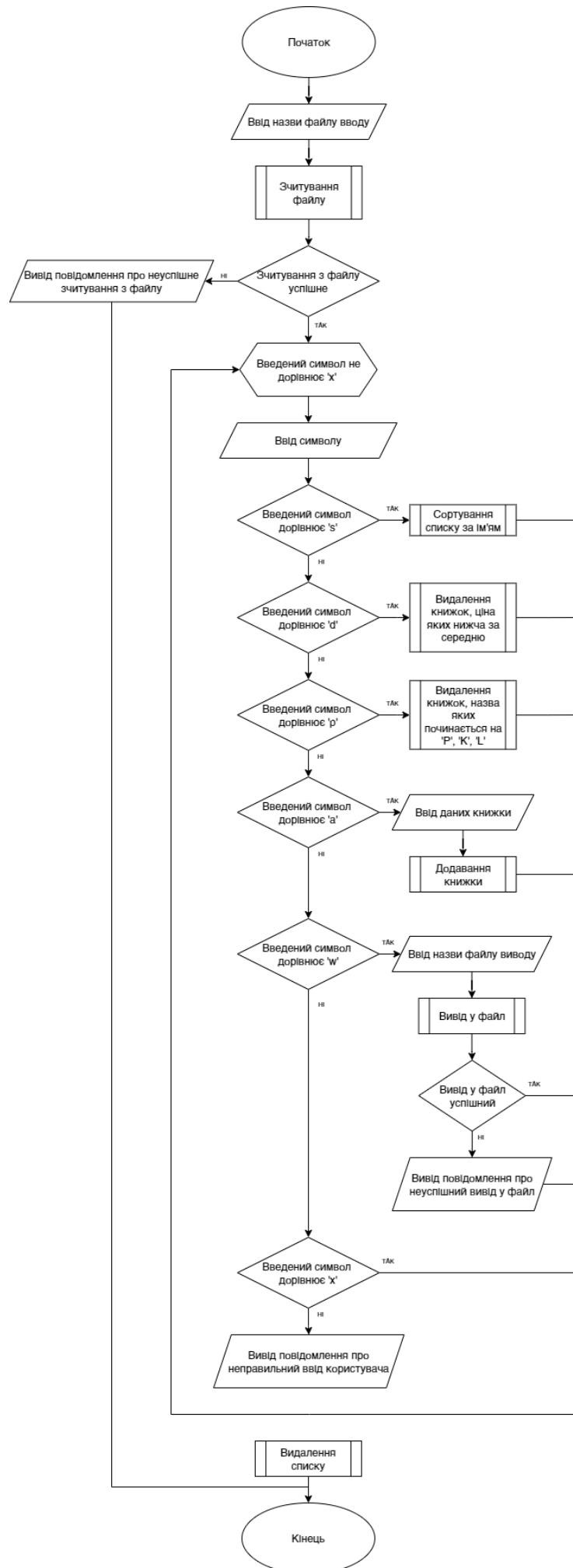
sBook
Data : sData
pbNext : sBook *

1.2 Однозв'язний список:

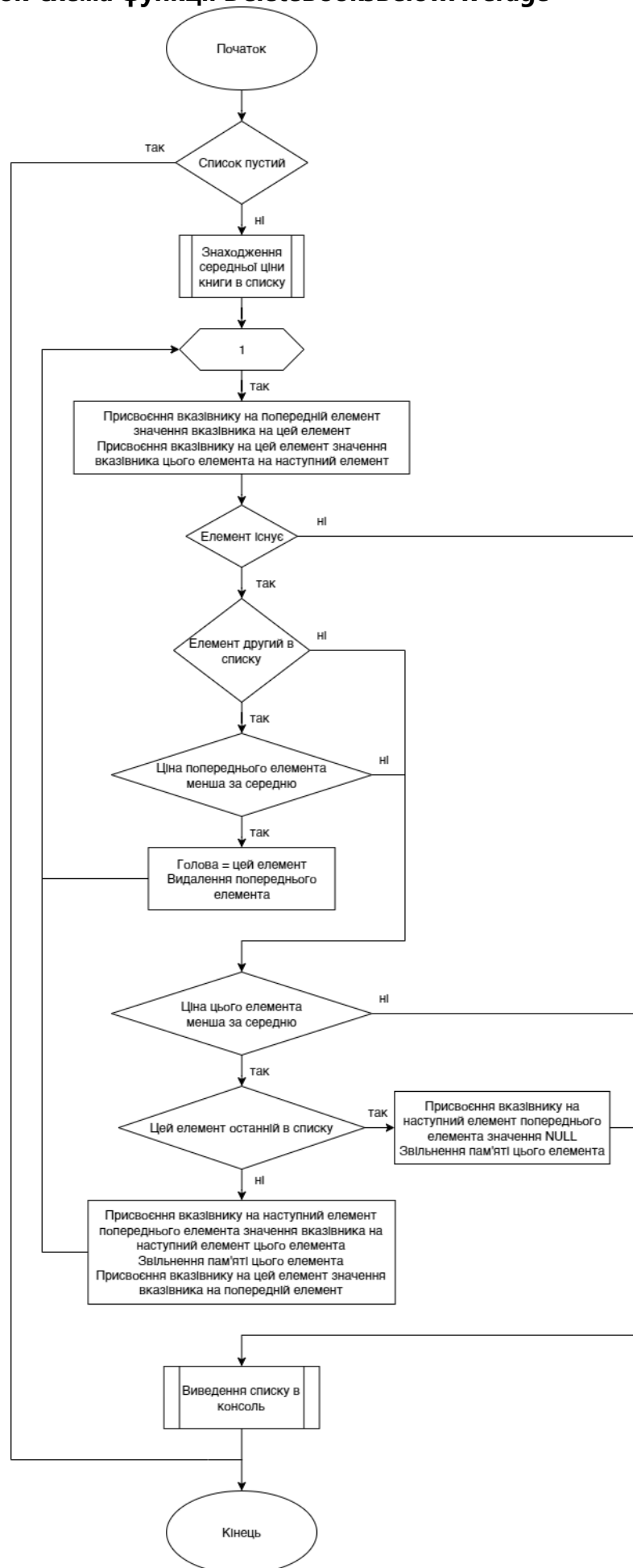


2. Блок-схема основної функції і двох окремих функцій-підпрограм

2.1. Блок-схема функції main



2.2. Блок-схема функції DeleteBooksBelowAverage



2.3 Блок-схема функції FindAveragePrice



2.3. Текст програми з коментарями та оформлений згідно рекомендацій

Файл functions.h:

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#include <stdlib.h>
#include <stdio.h>
//default size of string used in struct data
#define STR_SIZE 50
//default size of line used for read from file
#define LINE_SIZE 150

typedef struct
{
    char m_szAuthorName[STR_SIZE];
    char m_szBookName[STR_SIZE];
    char m_szReleaseYear[STR_SIZE];
    char m_szPageCount[STR_SIZE];
    char m_szPrice[STR_SIZE];
```

```

} sData;
typedef struct
{
    sData m_data;
    struct Book *m_pbNext;
} sBook;
//writes to file, returns 1 if successful, returns 0 if not successful
int WriteToFile(char * szFileName, sBook* head);
//reads from file, takes null pointer, returns 1 if successful, returns 0
//
    if not successful
int ReadFromFile(char *szFileName, sBook** head);
//prints out whole list
void PrintList(sBook *head);
//returns average price of elements in the list
double FindAveragePrice(sBook *head);
//deletes all books priced below average, takes pointer to head pointer
void DeleteBooksBelowAverage(sBook **head);
//sorts list by name of the book in descending order, takes pointer to head pointer
void SortByName(sBook **head);
//deletes every book containing "PKL" at the start of it's name, takes pointer
//
    to head pointer
void DeleteBooksStartingWithPKL(sBook **head);
//adds new book placed by name in descending order, takes pointer to head pointer
void AddBook(sBook **head);
//deletes list
void DeleteList(sBook** head);
#endif

```

Файл functions.c:

```

#pragma warning(disable:4996)
#include "functions.h"

```

```

//-----
int WriteToFile(char *szFileName, sBook *head)
{
    FILE *pfOutput = fopen(szFileName, "w");
    if (head)
    {
        sBook* pbCurrentItem = head;
        while (pbCurrentItem)
        {
            fprintf(pfOutput, "%s\t%s\t%s\t%s\t%s",
                pbCurrentItem->m_data.m_szAuthorName,
                pbCurrentItem->m_data.m_szBookName,
                pbCurrentItem->m_data.m_szReleaseYear,
                pbCurrentItem->m_data.m_szPageCount,
                pbCurrentItem->m_data.m_szPrice);
            pbCurrentItem = pbCurrentItem->m_pbNext;
            if (pbCurrentItem)
                fprintf(pfOutput, "\n");
        }
        return 1;
    }
    fclose(pfOutput);
    return 0;
}
//-----
int ReadFromFile(char *szFileName, sBook **head)
{
    FILE* pfInput;
    if (pfInput = fopen(szFileName, "r"))
    {
        char* szLim = ",\t\n";
        char cTemp = fgetc(pfInput);
        if(!cTemp)
        {

```

```

        return 0;
    }
    else { ungetc(cTemp, pfInput); }
    sBook* pbCurrentItem = (sBook*)malloc(sizeof(sBook));
    char line[LINE_SIZE];
    while (1)
    {
        if (!*head)
            *head = pbCurrentItem;
        fgets(line, sizeof(line), pfInput);
        strcpy(pbCurrentItem->m_data.m_szAuthorName, strtok(line, szLim));
        strcpy(pbCurrentItem->m_data.m_szBookName, strtok(NULL, szLim));
        strcpy(pbCurrentItem->m_data.m_szReleaseYear, strtok(NULL, szLim));
        strcpy(pbCurrentItem->m_data.m_szPageCount, strtok(NULL, szLim));
        strcpy(pbCurrentItem->m_data.m_szPrice, strtok(NULL, szLim));
        if (feof(pfInput))
        {
            pbCurrentItem->m_pbNext = NULL;
            break;
        }
        else
        {
            sBook* m_pbNextItem = (sBook*)malloc(sizeof(sBook));
            pbCurrentItem->m_pbNext = m_pbNextItem;
            pbCurrentItem = m_pbNextItem;
        }
    }
    fclose(pfInput);
    printf("\t\t\t\t\t\t\t[INITIAL LIST]\n\n");
    PrintList(*head);
    return 1;
}
return 0;
}
//-----
void PrintList(sBook *head)
{
    putchar('\n');
    for (int i = 0; i < 123; i++)
    {
        if (i == 0 || i == 122 || i == 33 || i == 73 || i == 89 || i == 105)
        {
            putchar('+');
        }
        else putchar('-');
    }
    putchar('\n');
    printf("| \tAUTHOR NAME\t\t | \tBOOK NAME\t\t\t | \tYEAR\t | \tPAGES\t"
        " | \tPRICE\t | \n");
    for (int i = 0; i < 123; i++)
    {
        if (i == 0 || i == 122 || i == 33 || i == 73 || i == 89 || i == 105)
        {
            putchar('+');
        }
        else putchar('-');
    }
    putchar('\n');
    while (head)
    {
        printf("| \t%-20s\t | \t%-24s\t | \t%-4s\t | \t%-4s\t | \t%-4s\t | \n",
            head->m_data.m_szAuthorName,
            head->m_data.m_szBookName,
            head->m_data.m_szReleaseYear,
            head->m_data.m_szPageCount,
            head->m_data.m_szPrice);
        head = head->m_pbNext;
    }
    for (int i = 0; i < 123; i++)
    {
        if (i == 0 || i == 122 || i == 33 || i == 73 || i == 89 || i == 105)

```



```

    {
        putchar('+');
    } else putchar('-');
}
printf("\n\n");
}
//-----
double FindAveragePrice(sBook *head)
{
    double dSum = 0;
    int i = 0;
    sBook* pbCurrentItem = head;
    while (pbCurrentItem)
    {
        dSum += atof(head->m_data.m_szPrice);
        i++;
        pbCurrentItem = pbCurrentItem->m_pbNext;
    }
    return dSum / i;
}
//-----
void DeleteBooksBelowAverage(sBook **head)
{
    if (head)
    {
        double avg = FindAveragePrice(*head);
        sBook* pbCurrentItem = *(head);
        while (1)
        {
            sBook* pbPreviousItem = pbCurrentItem;
            pbCurrentItem = pbCurrentItem->m_pbNext;
            if (!pbCurrentItem) break;
            if (pbPreviousItem == *(head) &&
                atof(pbPreviousItem->m_data.m_szPrice) < avg)
            {
                *head = pbPreviousItem->m_pbNext;
                free(pbPreviousItem);
            }
            else if (atof(pbCurrentItem->m_data.m_szPrice) < avg)
            {
                if (!pbCurrentItem->m_pbNext)
                {
                    pbPreviousItem->m_pbNext = NULL;
                    free(pbCurrentItem);
                    break;
                }
                pbPreviousItem->m_pbNext = pbCurrentItem->m_pbNext;
                pbCurrentItem->m_pbNext = NULL;
                free(pbCurrentItem);
                pbCurrentItem = pbPreviousItem;
            }
        }
        printf("\t\t\t\t\t\t\t[LIST AFTER DELETION]\n\n");
        PrintList(*(head));
    }
}
//-----
void SortByName(sBook **head)
{
    if (*head)
    {
        sBook* pbIterator = *head;
        while (pbIterator)
        {
            sBook* pbCurrentItem = *head;
            while (pbCurrentItem->m_pbNext)
            {
                sBook* prevP = pbCurrentItem;

```

```

        pbCurrentItem = pbCurrentItem->m_pbNext;
        if (strcmp(
            prevP->m_data.m_szBookName,
            pbCurrentItem->m_data.m_szBookName) > 0)
        {
            sData temp = prevP->m_data;
            prevP->m_data = pbCurrentItem->m_data;
            pbCurrentItem->m_data = temp;
        }
    }
    pbIterator = pbIterator->m_pbNext;
}
printf("\t\t\t\t\t\t\t[SORTED LIST]\n\n");
PrintList(*head);
}
}
//-----
void DeleteBooksStartingWithWithPKL(sBook **head)
{
    if (*head)
    {
        sBook* pbCurrentItem = *(head);
        while (1)
        {
            sBook* pbPreviousItem = pbCurrentItem;
            pbCurrentItem = pbCurrentItem->m_pbNext;
            if (!pbCurrentItem) break;
            char* szLimiter = "PKL";
            for (int i = 0; *(szLimiter + i) != 0; i++)
            {
                if (*(head) == pbPreviousItem)
                {
                    if (*(pbPreviousItem->m_data.m_szBookName) == *(szLimiter
+ i))
                    {
                        *(head) = pbPreviousItem->m_pbNext;
                        free(pbPreviousItem);
                        break;
                    }
                }
                if (*(pbCurrentItem->m_data.m_szBookName) == *(szLimiter + i))
                {
                    if (!pbCurrentItem->m_pbNext)
                    {
                        free(pbCurrentItem);
                        pbPreviousItem->m_pbNext = NULL;
                        pbCurrentItem = pbPreviousItem;
                        break;
                    } else
                    {
                        pbPreviousItem->m_pbNext = pbCurrentItem->m_pbNext;
                        pbCurrentItem->m_pbNext = NULL;
                        free(pbCurrentItem);
                        pbCurrentItem = pbPreviousItem;
                        break;
                    }
                }
            }
        }
    }
    printf("\t\t\t\t\t\t\t[LIST AFTER DELETION]\n\n");
    PrintList(*(head));
}
//-----
void AddBook(sBook **head)
{
    char* szLim = ",\t\n";
    char szNewBookm_data[LINE_SIZE];

```