

Міністерство освіти і науки України
Національний університет "Львівська політехніка"
Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення



Звіт

Про виконання лабораторної роботи №2

На тему:

«Документування етапів проектування та кодування програми»

з дисципліни

«Вступ до Інженерії Програмного Забезпечення»

Лектор:

Доцент каф. ПЗ

Левус Є. В.

Виконав:

ст. гр. ПЗ-11

Ясногородський Н.В.

Прийняла:

Доцент каф. ПЗ

Левус Є. В.

« __ » _____ 2022 р.

Σ = _____ .

Львів – 2022

Тема. Документування етапів проектування та кодування програми.

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

Теоретичні відомості







4. Які є рівні проектування ПЗ? Чим вони відрізняються?

Зазвичай складна задача проектування складається з двох етапів (рівнів):

- архітектурне проектування або високорівневий дизайн (Software Architectural Design/Top-level Design, англ.), протягом якого обумовлюються так звані компоненти високого рівня: зв'язок між найбільшими і загальними частинами програмного забезпечення
- детальне проектування компонентів (Software Detailed Design, англ.), протягом якого визначається деталізована архітектура, що описує кожну компоненту у тому обсязі, який необхідний для конструювання.

14. З чого складається блок-схема алгоритму?

Складається з геометричних фігур, які з'єднані напрямленими лініями

Основні конструкції блок-схем	
Оператор	Опис
	Термінатор. Початок/кінець
	Оператор вводу, виводу даних
	Процес обчислень, арифметична дія
	Розгалуження. Оператор умови.
	Підпрограми (функції)
	Оператор циклу

20. Які правила до найменування змінних? Навести три приклади.

- назва кожної змінної повинна починатися її типом, наприклад `int* pnArray`
- використовувати пари змінних `min/max`, `begin/end`, `open/close`
- додавати в кінець назви кваліфікатори `Min`, `Max`, `Avg`, `Index` при необхідності

Завдання

Частина I. У розробленій раніше програмі до лабораторної роботи з дисципліни «Основи програмування» внести зміни - привести 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

- схематичне зображення структур даних, які використовуються для збереження інформації;
- блок-схема алгоритмів - основної функції й двох окремих функцій підпрограм (наприклад, сортування та редагування);
- текст програми з коментарями та оформлений згідно наведених рекомендацій щодо забезпечення читабельності й зрозумілості.

Результат

Схематичне зображення структур даних:

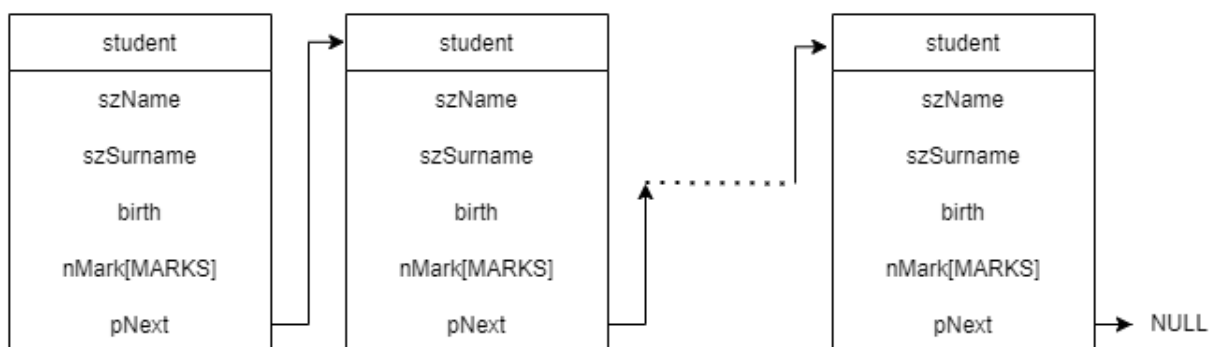


Рис.1. Однозв'язний список структур student



Рис.2. Структура sDate, що є полем birth структури student



Рис.3. Одновимірний масив

Блок-схеми:

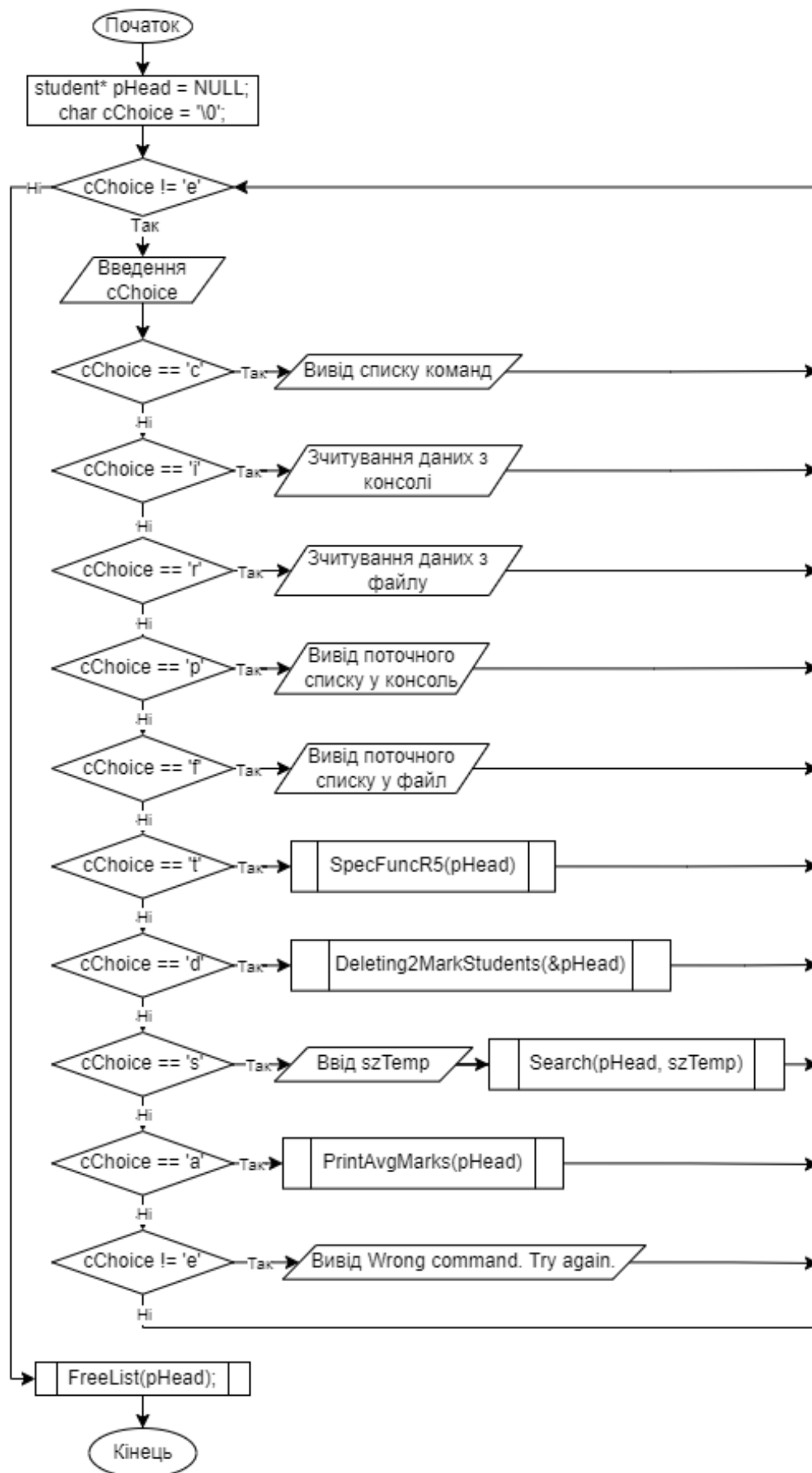


Рис.4. Функція main

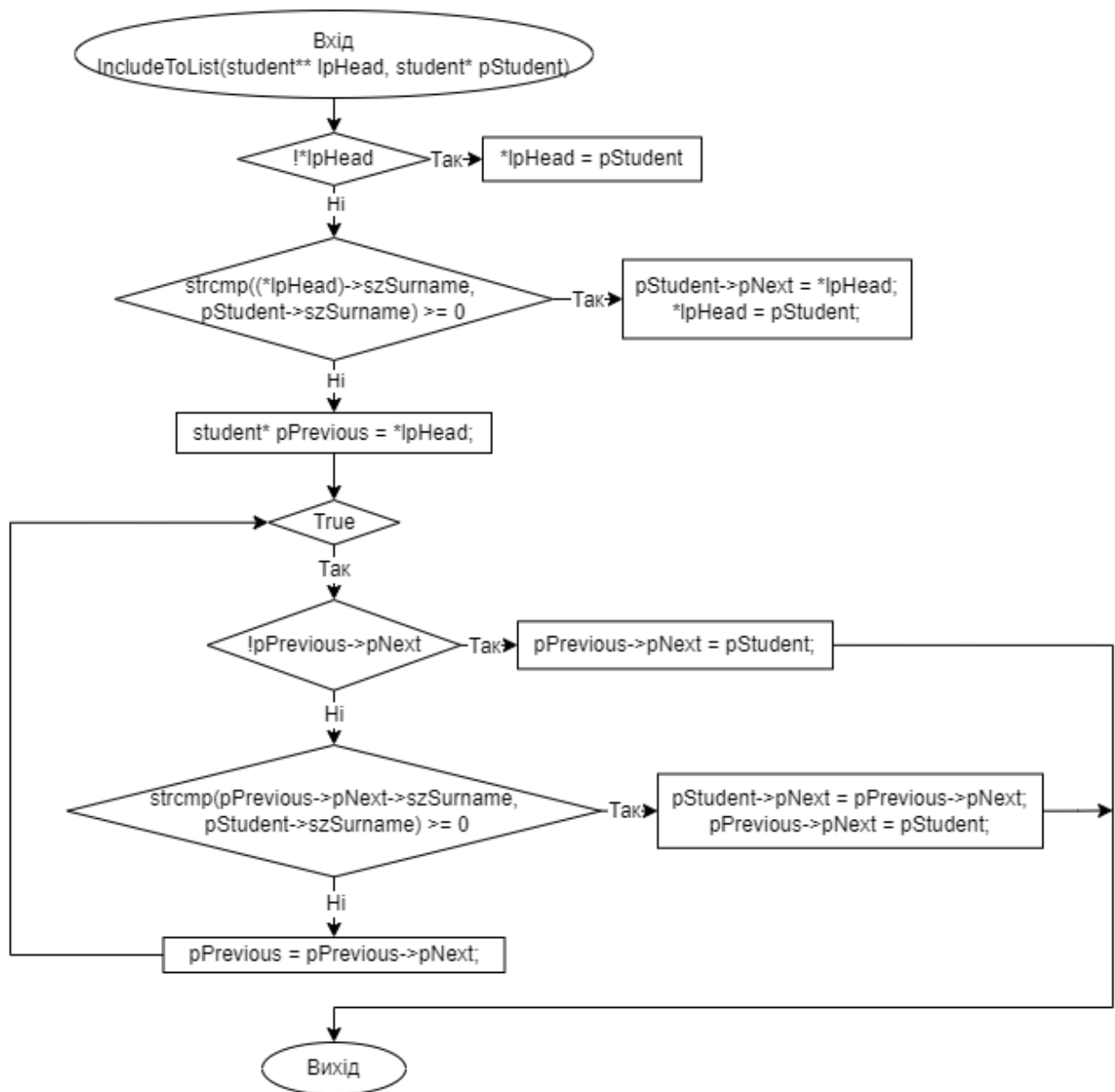


Рис.5. Функція IncludeToList

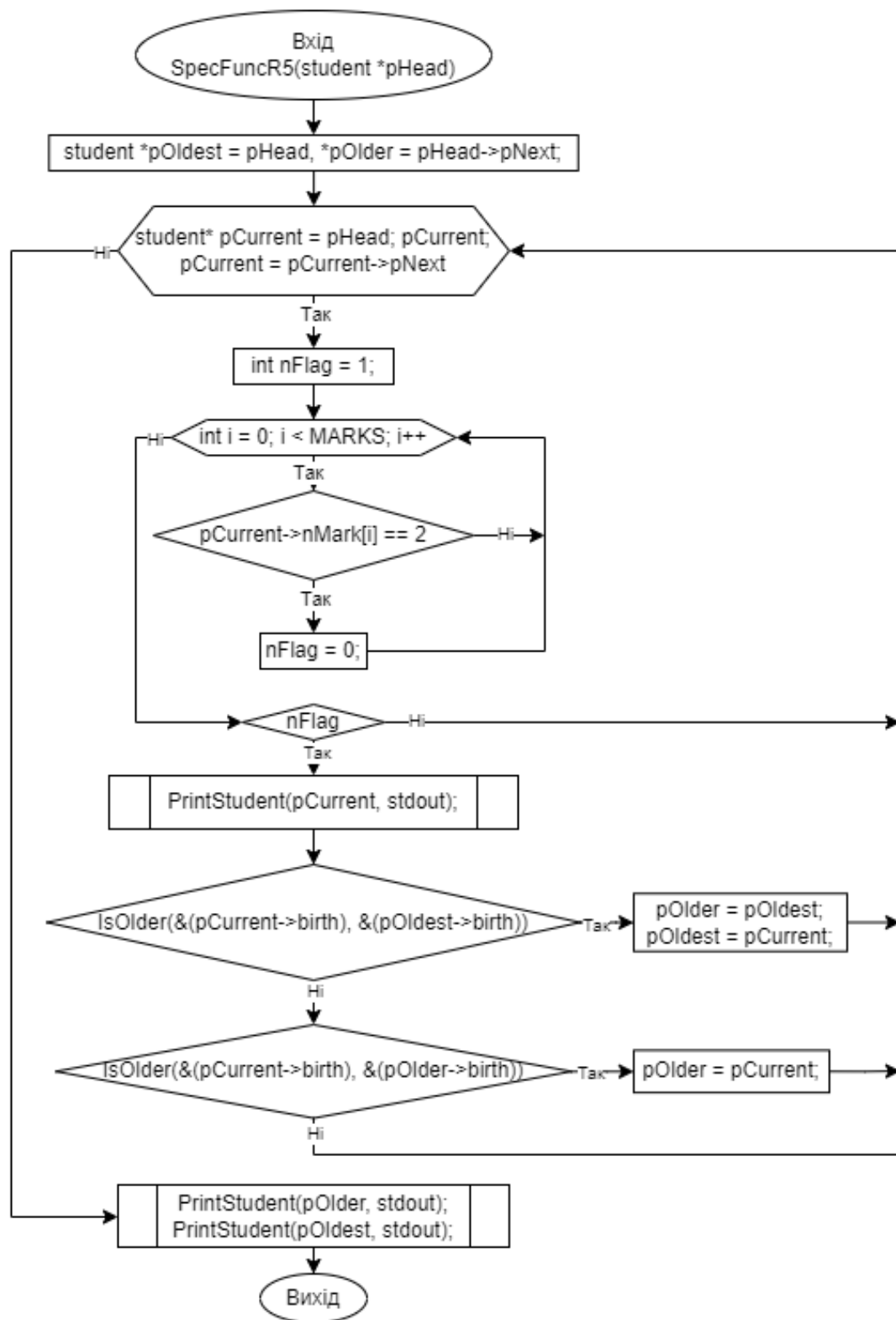


Рис.6. Функція SpecFuncR5

Текст програми:

func.h

```
#include <stdio.h>
#define MARKS 10 // exams count
#define NAME_SIZE 60 // maximum name and surname length

// represents info about the student (linked list)
typedef struct sStud {
    struct sStud *pNext;
    struct sDate {
        short nDay;
        short nMonth;
        short nYear;
    } birth;
    char szName[NAME_SIZE];
    char szSurname[NAME_SIZE];
    short nMark[MARKS];
} student;

void emptyList(student *pHead);
int printList(student *pHead, FILE *fStream);
int printStudent(student *pStudent, FILE *fStream);
int addToList(student **lpHead, student *pStudent);
int scanToList(student **lpHead, FILE *fStream);
int delete2MarkStudents(student **lpHead);
int isOlder(struct sDate* d1, struct sDate* d2);
int specFuncR5(student* pHead);
int searchByNameOrSurname(student* pHead, char szSegment[NAME_SIZE]);
int printAvgMarks(student* pHead);
```

func.c

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "func.h"

// free dynamic memory
void emptyList(student* pHead) {
    if (!pHead) return;
    emptyList(pHead->pNext);
    free(pHead);
}

// prints all students
int printList(student* pHead, FILE* fStream) {
    if (!fStream) return 0;
    if (pHead) {
        printStudent(pHead, fStream);
        printList(pHead->pNext, fStream);
        return 1;
    } else
        return 0;
}
```



```

int printStudent(student* pStudent, FILE* fStream) {
    if (!fStream) return 0;

    if (pStudent) {
        fprintf(fStream, "\n %10s %10s %02hd/%02hd/%hd ", pStudent->szName,
            pStudent->szSurname, pStudent->birth.nDay, pStudent->birth.nMonth,
            pStudent->birth.nYear);

        for (int i = 0; i < MARKS; i++)
            fprintf(fStream, "% hd", pStudent->nMark[i]);
        return 1;
    }

    return 0;
}

// adds with sorting by last name
int addToList(student** lpHead, student* pStudent) {
    if (!lpHead || !pStudent) return 0;

    if (!*lpHead) {
        *lpHead = pStudent;
        return 1
    }

    if (strcmp((*lpHead)->szSurname, pStudent->szSurname) >= 0) {
        pStudent->pNext = *lpHead;
        *lpHead = pStudent;
        return 1
    }

    student* pPrevious = *lpHead;
    while (1) {
        if (!pPrevious->pNext) {
            pPrevious->pNext = pStudent;
            break;
        } else if (strcmp(pPrevious->pNext->szSurname, pStudent->szSurname) >= 0) {
            pStudent->pNext = pPrevious->pNext;
            pPrevious->pNext = pStudent;
            break;
        }
        pPrevious = pPrevious->pNext;
    }

    return 1;
}

```

```

// scans student from stream and adds them
int scanToList(student** lpHead, FILE* fStream) {
    student* pStudent = (student*)malloc(sizeof(student));
    if (!pStudent) return 0;

    fscanf_s(fStream, "%s %s %hd/%hd/%hd", pStudent->szName, NAME_SIZE,
        pStudent->szSurname, NAME_SIZE, &pStudent->birth.nDay,
        &pStudent->birth.nMonth, &pStudent->birth.nYear);

    for (int i = 0; pStudent->szName[i] != '\0'; i++) {
        if (!isalpha(pStudent->szName[i])) {
            free(pStudent);
            return 0;
        }
    }
    for (int i = 0; pStudent->szSurname[i] != '\0'; i++) {
        if (!isalpha(pStudent->szSurname[i])) {
            free(pStudent);
            return 0;
        }
    }
    if (pStudent->birth.nDay < 1 || pStudent->birth.nDay > 31 ||
        pStudent->birth.nMonth < 1 || pStudent->birth.nMonth > 12 ||
        pStudent->birth.nYear < 1920 || pStudent->birth.nYear > 2020) {
        free(pStudent);
        return 0;
    }
    for (int j = 0; j < MARKS; j++) {
        if (!fscanf_s(fStream, " %hd", pStudent->nMark + j) ||
            pStudent->nMark[j] < 2 || pStudent->nMark[j] > 5) {
            free(pStudent);
            return 0;
        }
    }
    pStudent->pNext = NULL;
    if (!addToList(lpHead, pStudent)) {
        free(pStudent);
        return 0;
    }
    return 1;
}

// delete students with a grade "2" for the first exam
int delete2MarkStudents(student** lpHead) {
    if (!lpHead || !*lpHead) return 0;

    for (student *pPrevious = NULL, *pCurrent = *lpHead; pCurrent;) {
        if (pCurrent->nMark[0] != 2) {
            pPrevious = pCurrent;
            pCurrent = pCurrent->pNext;
            continue;
        }
        if (!pPrevious) {
            *lpHead = pCurrent->pNext;
            free(pCurrent);
            pCurrent = *lpHead;
        } else {
            pPrevious->pNext = pCurrent->pNext;
            free(pCurrent);
            pCurrent = pPrevious->pNext;
        }
    }
    return 1;
}

```

```

// compare dates

int isOlder(struct sDate* d1, struct sDate* d2) {
    if (d1->nYear < d2->nYear)
        return 1;
    else if (d1->nYear > d2->nYear)
        return 0;
    else {
        if (d1->nMonth < d2->nMonth)
            return 1;
        else if (d1->nMonth > d2->nMonth)
            return 0;
        else {
            if (d1->nDay < d2->nDay)
                return 1;
            else
                return 0;
        }
    }
}

// prints students without '2' mark and looks for two oldest of them
// returns 0 if the list contains less than 2 students, otherwise 1
int specFuncR5(student* pHead) {
    if (!pHead || !pHead->pNext) return 0;

    printf("\n Students that don't have mark '2':");
    student *pOldest = pHead, *pOlder = pHead->pNext;

    for (student* pCurrent = pHead; pCurrent; pCurrent = pCurrent->pNext) {
        int nFlag = 1;
        for (int i = 0; i < MARKS; i++) {
            if (pCurrent->nMark[i] == 2) nFlag = 0;
        }

        if (nFlag) {
            PrintStudent(pCurrent, stdout);
            if (IsOlder(&(pCurrent->birth), &(pOldest->birth))) {
                pOlder = pOldest;
                pOldest = pCurrent;
            } else if (IsOlder(&(pCurrent->birth), &(pOlder->birth))) {
                pOlder = pCurrent;
            }
        }
    }

    printf("\n\n The two oldest:");
    PrintStudent(pOlder, stdout);
    PrintStudent(pOldest, stdout);
    return 1;
}

```

```

// searches by name or last name and print them
int searchByNameOrSurname(student* pHead, char szSegment[NAME_SIZE]) {
    if (!pHead) return 0;
    int nFlag = 0;
    for (student* pCurrent = pHead; pCurrent; pCurrent = pCurrent->pNext) {
        if (strstr(pCurrent->szName, szSegment) ||
            strstr(pCurrent->szSurname, szSegment)) {
            PrintStudent(pCurrent, stdout);
            nFlag = 1;
        }
    }
    return nFlag;
}

// prints average marks
int printAvgMarks(student* pHead) {
    if (!pHead) return 0;

    double lfMarks[MARKS] = {0.};
    int nCounter = 0;

    for (student* pCurrent = pHead; pCurrent;
        pCurrent = pCurrent->pNext, nCounter++) {
        for (int i = 0; i < MARKS; i++) {
            lfMarks[i] += (double)pCurrent->nMark[i];
        }
    }

    for (int i = 0; i < MARKS; i++) printf(" %g", lfMarks[i] / nCounter);
    return 1;
}

```

Source.c

```

#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "func.h"

int main(void) {
    student* pHead = NULL;
    char cChoice = '\0', garbage;

    while (cChoice != 'e') {
        printf(
            "\n\n Enter the character that corresponds to the command, or c to get "
            "a list of commands: ");

        while (1) {
            garbage = cChoice;
            cChoice = getchar();
            if (cChoice == '\n') {
                cChoice = garbage;
                break;
            }
        }
    }
}

```

```

switch (cChoice) {
    case 'c': {
        printf(
            " Commands list : \n\
e - exit\n\
i - add student from console\n\
r - read students from file\n\
p - print the current list\n\
f - save the current list to a file\n\
t - print students without '2' marks and two oldest of them\n\
d - delete students with the first '2' mark\n\
s - search for students by name or last name\n\
a - print the average marks from all exams\n");
        break;
    }
    case 'i': {
        if (!ScanToList(&pHead, stdin)) {
            printf("\n Cannot include student. Check inputed data.");
        }
        while (getchar() != '\n') {
        }
        break;
    }
    case 'r': {
        FILE* fp;
        if (fopen_s(&fp, "data.txt", "r")) {
            printf("\n Cannot open file.");
            break;
        }
        for (int i = 1; !feof(fp); i++) {
            if (!ScanToList(&pHead, fp)) {
                printf("Cannot scan %d-th line of data. Check input file.", i);
            }
        }
        fclose(fp);
        break;
    }
    case 'p': {
        printf("\n %10s %10s %s\t %s\n", "Name", "Surname", "Birth", "Marks");
        if (!PrintList(pHead, stdout)) {
            printf("\n Cannot print list.");
        }
        break;
    }
    case 'f': {
        FILE* fp;
        if (fopen_s(&fp, "out.txt", "w")) {
            printf("\n Cannot open file.");
            break;
        }
        if (!PrintList(pHead, fp)) {
            printf("\n Cannot save list.");
        }
        fclose(fp);
        break;
    }
    case 't': {
        if (!SpecFuncR5(pHead)) {
            printf("\n Cannot find oldest students.");
        }
        break;
    }
    case 'd': {
        if (!Deleting2MarkStudents(&pHead)) {
            printf("\n Cannot delete students.");
        }
    }
}

```

```

    }
    break;
}
case 's': {
    char szTemp[NAME_SIZE];
    scanf_s("%s", szTemp, NAME_SIZE);
    if (!Search(pHead, szTemp)) {
        printf("\n Students not found.");
    }
    while (getchar() != '\n') {
    }
    break;
}
case 'a': {
    if (!PrintAvgMarks(pHead)) {
        printf("\n Cannot find average marks.");
    }
    break;
}
default: {
    if (cChoice != 'e') printf("\n Wrong command. Try again.");
    break;
}
}
}

emptyList(pHead);
return 0;
}

```

Висновок

Навчився документувати етапи проектування та кодування програми, склав схематичне зображення використаних структур даних, блок-схему алгоритмів та написав код програми.