МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Чернігівська політехніка»

ОСНОВИ ПРОГРАМУВАННЯ

МЕТОДИЧНІ ВКАЗІВКИ

до курсового проєктування
для здобувачів вищої освіти першого (бакалаврського) рівня
за освітньою програмою “Комп’ютерна інженерія”

Обговорено і рекомендовано
на засіданні кафедри
інформаційних і комп'ютерних
систем
Протокол № 6 від 30.05.23

Чернігів 2023

Основи програмування. Методичні вказівки до курсового проєктування для здобувачів вищої освіти першого (бакалаврського) рівня за освітньою програмою „Комп'ютерна інженерія". /Укл.: Бивойно П.Г., Бивойно Т.П., Бичко В.А. – Чернігів: НУ «Чернігівська політехніка», 2023. – 67 с.

Укладачі: Бивойно Павло Георгійович, канд. техн. наук, доцент
Бивойно Тарас Павлович, ст. викладач
Бичко Володимир Анатолійович, канд. ф.м. наук, доцент

Відповідальний за випуск: Базилевич В.М., зав. кафедрою інформаційних и комп'ютерних систем, канд. економ. наук

Рецензент: Пріла О.А., канд. техн. наук, доцент кафедри інформаційних та комп'ютерних систем НУ «Чернігівська політехніка»

ЗМІСТ

ВСТУП

У курсі «Основи програмування» виконання курсового проекту є частиною самостійної роботи студентів над дисципліною. Виконання проекту передбачає проведення аналізу деякої предметної області і подальшу побудову програми, що вирішує проблеми зберігання та обробки інформації, що пов’язана з цією областю.

Окрім аналізу предметної області студент має проаналізувати типові структури даних та оцінити можливість і доцільність реалізації їх у проекті.

На етапі розробки проекту студент має визначитися із функціоналом майбутньої системи та структурою меню. Необхідно також розробити схему збереження даних у файлі та вибрати, або розробити алгоритми опрацювання даних.

Розроблені засоби мають дозволити обробку даних як безпосередньо через програму, так і через консольний інтерфейс користувача.

Робота над проєктом виконується поетапно.

У розділах цих методичних вказівках докладно пояснюється, що потрібно зробити на кожному етапі і наводяться зразки функцій, які потрібно реалізувати в межах етапу.

Результатом роботи є працездатний програмний консольний додаток та звіт у вигляді друкованого тексту оформленого відповідно до вимог розділу 7 стандарту ДСТУ3008:2015. Бали за роботу виставляються з урахуванням своєчасності та якості виконання як програмної частини, так і звіту.

Варіант завдання, який має виконувати студент, визначається номером студента у списку групи

Студент може запропонувати іншу предметну область для реалізації, але завдання обов'язково має бути погоджено з викладачем.

ЗАВДАННЯ ДО КУРСОВОГО ПРОЄКТУ

Варіант	Корінь дерева	1-й рівень	2-й рівень
1.Залізниця	місто	вокзал	рейс
2.Автобус	вокзал	напрям	рейс
3. Аеропорт	місто	аеропорт	рейс
4. Комп'ютер	країна	фірма	комп'ютер
5. Оргтехніка	місто	магазин	товар
6. Торгова база	склад	група товарів	товар
7. Демографія	регіон	область	насел.пункт
8. Екологія	область	район	екопроблеми
9. Місто	вулиця	будинок	квартира
10. Комун.хоз.	жред	професія	робітник
11. МОК	олімпіада	вид спорту	спортсмен
12. УПЛ	клуб	амплуа	спортсмен
13. Ігрові чемпіонати	що, де, коли	команди	учасники
14. Комендант	корпус	аудиторія	мебель
15. Фарм. фірма	країна	відділ	ліки
16.Торг. мережа	фірма	постачальники	товари
17. Виробництво	вироби	вузли	деталі
18. Бібліотека	кафедра	предмет	література
19. Деканат	спеціальність	група	студент
20. Кафедра	предмет	викладачі	студенти
21. Ресторан	кухня	блюдо	продукти
22. Лікарня	відділення	хворий	показники
23. Космос	зірки	планети	супутники
24. Мої витрати	дата(3 поля)	вид, ліміт	чек
25. Географія	континенти	регіони	держави
26. Гуртожитоки	гуртожиток	кімната	мешканець
27. Торгівля	продавець	товар	реалізація
28. Поліклініка	лікар	пацієнт	рецепт
29. Музика	жанр	група	пісня
30. Мої подорожі	континент	країна	друзі
31. Суд	суддя	справа	підсудні
32.Турфірма	вид туризму	країна	пропозиція
33. Відпочинок	країна	курорт	готель
34. Бронь готелів	країна	місто	готель
35. Військ. допомога	країна	період	зброя
36. Біженці	країна	місто	біженець

ВИМОГИ ДО РОБОТИ

Курсовий проект передбачає створення динамічної інформаційної структури даних у вигляді консольного Qt-додатку, що забезпечить роботу із багаторівневою ієрархічною структурою даних у вигляді розгалуженого дерева з єдиним коренем.

Корінь та компоненти кожного рівня мають бути структурами, що містять не менше двох інформаційних полів. Поля структури не можуть бути одного типу. Принаймні на двох рівнях у структур мають бути числові поля.

Нащадки кореня (перший рівень) мають зберігатися у масиві вказівників на відповідні структури.

Структури другого рівня мають зберігатися у лінійному списку.

Текстові поля всіх структур мають зберігати вказівники на рядки символів, що розташовані в динамічній пам’яті.

Нащадки кожного з елементів мають бути впорядковані за простими або складними правилами. Складне правило має бути реалізовано принаймні на одному рівні.

Усі операції додавання, редагування і вилучення вузлів, операції збереження і відновлення дерева у файлі структур треба реєструвати у текстовому файлі протоколу.

Додаток має забезпечити реалізацію таких функцій:

– відобразити дерево на консолі цілком (у текстовому режимі);
– відобразити на консолі файл протоколу;
– відобразити дані вузла;
– відобразити перелік нащадків вузла;
– змінити дані існуючого вузла;
– додати вузол до дерева;
– вилучити будь який вузол дерева;
– вилучити усіх нащадків вузла;
– зберегти структуру даних у файлі;
– відновити структуру даних із файлу;
– вивести на консоль перелік частини вузлів за певною ознакою;
– обчислити якусь числову характеристику групи елементів;
– отримати повну інформацію про вузол пойого назві (можливо не повній)

Перелік функцій додатку має забезпечити роботу з деревом як програмно так і через консольне меню.

Детальні вимоги до проекту наводяться в індивідуальному завданні на проект, яке складає студент і узгоджує із керівником проекту.

На захист студент має представити працюючий додаток і звіт оформлену відповідно до ДСТУ3008:2015 і вимог кафедри.

ГРАФІК ВИКОНАННЯ РОБОТИ

Робота над проектом починається з першого тижня семестру і захист має відбутися до початку залікового тижня. Графік роботи наведено у таблиці 1.

Таблиця 1 – Графік роботи над проектом

№	Етапи роботи	Звітність	Дедлайн (тиждень семестру)	Оцінка в балах за етап
1	Розробка моделі структури даних	.docx	1-й	до 10
2	Початкова реалізація проєкту	архів проєкту	3-й	до 10
3	Створення дефолтного дерева		5-й	до 10
4	Реалізація роботи через меню		7-й	до 10
5	Редагування та видалення вузлів		9-й	до 10
6	Робота з деревом		11-й	до 10
7	Представити текстову частину	.docx	13-й	до 10
8	Захист	проєкт із змінами	14-й	до 30

Проект оцінюється за 100 бальною шкалою. За роботу у семестрі студент може отримати до 70 балів. Бали виставляються за кожний етап в залежності від якості виконання та додержання графіку виконання.

Етап, за який виставлено оцінку не менше 6 балів, вважається завершеним.

Після «дедлайну» результати виконання етапу не приймаються, але студент може завантажити свої наробки як результат наступного етапу і оцінку буде виставлено і за попередній (попередні) етапи, а також і за поточний, якщо завдання поточного етапу виконано.

Якщо результати виконання якогось етапу були представлені невчасно, оцінка не може бути вищою за 8 балів, кожен наступний пропущений «дедлайн» зменшує максимальну оцінку ще на 1 бал. Якщо недоліки, за які оцінку було знижено на попередньому етапі, не були виправлені, то отримати максимальну оцінку за наступний етап студент не може.

Захисти роботи проводяться на останньому тижні семестру. Під час захисту студент має виконати практичне завдання з доопрацювання свого проекту та відповісти на питання, що стосуються програмування і структур даних, якщо викладач вважає це необхідним.

За результатами захисту роботи студент може отримати до 30 балів.

Результати виконання етапів студент викладає у вигляді архіву папки з qt-проектом (папка …-build-desktop не потрібна). Текстові етапи надсилаються у вигляді .docx файлів. Назви файлів мають починатися з прізвища студента.

1 РОЗРОБКА МОДЕЛІ СТРУКТУРИ ДАНИХ

1.1 Побудова логічної моделі

Логічна модель описує структуру даних у термінах предметної області і не пов’язана з програмною реалізацією структури. В нашому проекті логічна модель являє собою сильно розгалужене дерево, де кожен вузол може мати будь-яку кількість нащадків.

Коренем дерева є структура «Університет». Ця структура, як корінь дерева, існує у проекті в одному екземплярі. «Університет» будемо характеризувати такими атрибутами:

– назва університету;

– рівень акредитації.

«Університет» має нащадків, які представлені структурами «Кафедра». Визначимо атрибути для структури «Кафедра»:

– назва кафедри;

– обсяг навчального навантаження (у годинах);

– випускаюча, чи ні.

Кожна «Кафедра» має певну кількість викладачів, які представлені структурами «Викладач». Структура «Викладач» буде мати такі атрибути:

– ім’я викладача;

– посада;

– рік народження.

Схематичне представлення деревовидної структури даних для нашого проекту показано на рисунку

Рисунок 1 – Логічна модель деревовидної структури даних

1.2 Розробка фізичної моделі структури даних

Фізична модель описує структуру даних у термінах мови програмування. Тут перш за все визначаються ідентифікатори для шаблонів структур та назви полів структур і типи даних для цих полів.

Далі визначається перелік службових полів структури, що використовуються для зв'язування вузлів дерева між собою. Існує багато способів реалізації таких зв'язків. У даному проекті, відповідно до завдання, зв'язки між структурами, які утворюють дерево будуть реалізовані таким чином.

Структура «Університет», яку ми назвемо Univer, буде містити вказівник на масив вказівників на структури типу «Кафедра». Для забезпечення роботи з цим масивом у структурі потрібні ще два поля – поле для зберігання довжини масиву і поле для зберігання кількості кафедр, що знаходяться у масиві.

Структура «Кафедра» з назвою Dept (скорочення від слова Department) буде містити вказівник на початок списку викладачів цієї кафедри.

Структура «Викладач» (Teacher), буде містити вказівник на викладача, який є наступним у списку. Тобто список буде односпрямованим.

Результати розробки структур для вузлів дерева наведені у таблицях 2..4.

Таблиця 2 – Поля структури Univer (Університет)

Призначення поля	Ідентифікатор	Тип
Назва університету	name	char *
Рівень акредитації	level	int
Вказівник на масив вказівників на кафедри	deptArr	Dept * *
Розмір масиву вказівників на кафедри	arrLength	unit
Кількість вказівників у масиві (кількість кафедр)	deptCount	unit

Таблиця 3 – Поля структури Dept (Кафедра)

Призначення поля	Ідентифікатор	Тип
Назва кафедри	name	char *
Обсяг навчального навантаження (у годинах)	hours	unit
Випускаюча кафедра, чи ні (1 або 0)	spec	int
Вказівник на першого викладача у списку викладачів кафедри	firstTeacher	Teacher*
Вказівник на батьківську структуру (університет)	parent	Univer *

Таблиця 4 – Поля структури Teacher (Викладач)

Призначення поля	Ідентифікатор	Тип
Ім’я викладача	name	char *
Посада	post	char *
Рік народження	birthYear	uint
Вказівник на наступного викладача у списку викладачів кафедри	nextTeacher	Teacher *
Вказівник на батьківську структуру (кафедра)	parent	Dept *

Фізична модель нашої структури даних у вигляді схеми показана на рисунку 2.

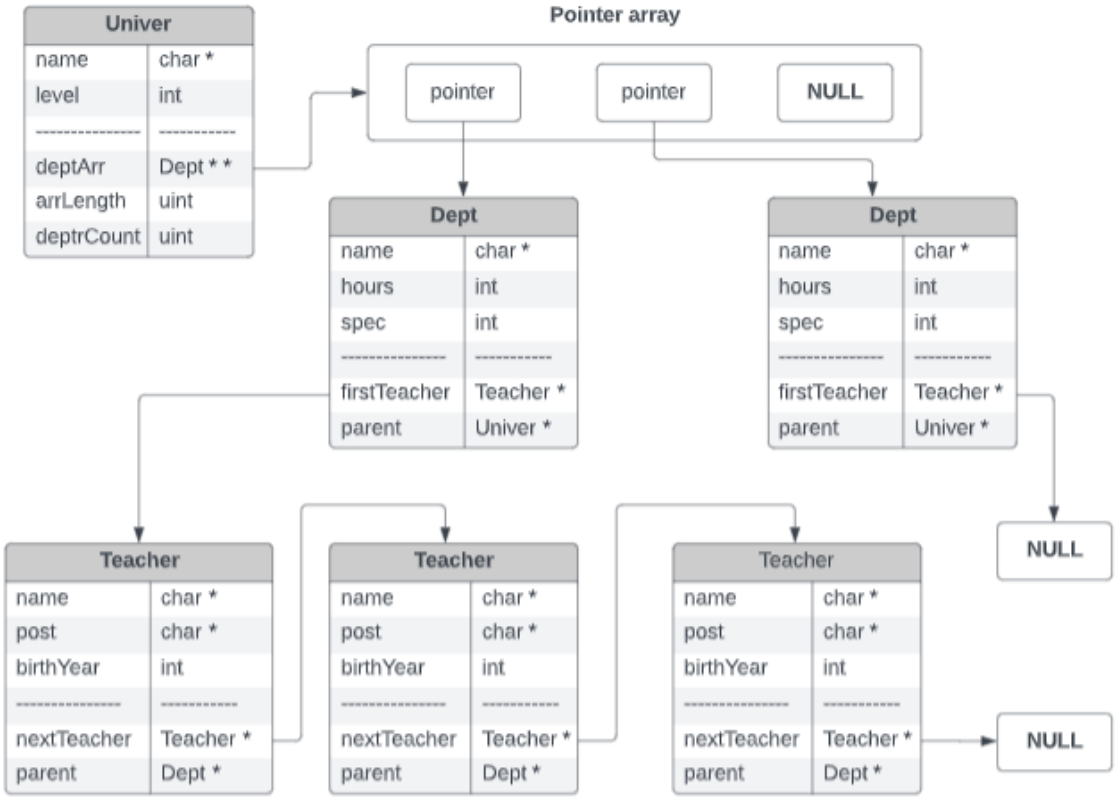

Рисунок 2 – Фізична модель структури даних

На рисунку показано, що структура Univer через поле deptArr має посилання на масив вказівників, який на рисунку умовно зображено у вигляді прямокутника з назвою Pointer array. Довжина масиву дозволяє розмістити три вказівника, тобто значення поля arrLength структури Univer дорівнює 3. Але в масиві вказівниками на структури Dept зайнято тільки дві комірки. Третя комірка вільна і значення вказівника дорівнює NULL. У цьому випадку значення поля deptCount структури Univer дорівнює 2.

Перша із двох структур Dept, які показані на рисунку, має у полі firstTeacher посилання на список викладачів кафедри. У другої це посилання дорівнює NULL, тобто викладачі на цій кафедрі поки що відсутні.

Кожна структура Teacher має посилання на наступну структуру у списку через поле nextTeacher. У структури, яка є останньою у списку, значення вказівника nextTeacher дорівнює NULL.

2 ПОЧАТКОВА РЕАЛІЗАЦІЯ ПРОЄКТУ

На цьому етапі слід реалізувати такі завдання;

– створити проєкт, заголовні файли та деякі файли .cpp для зберігання коду функцій;

– оголосити структури для вузлів дерева у файлі type.h;

– оголосити зовнішні змінні для кореня дерева і файлу для протоколу роботи із застосунком (лог-файл) у файлі prototype.h;

– оголосити і створити допоміжні функції;

– написати і оголосити функцію виведення вмісту лог-файлу на консоль;

– написати і оголосити функцію виведення інформації про проєкт;

– створити і оголосити заготовки функцій створення дефолтного дерева, виведення дерева на консоль та функції головного меню;

– написати функцію main().

2.1 Реалізація файлової структури проєкту

Створімо консольний проєкт. Для назви проєкту будемо використовувати прізвище розробника та тему. У прикладі це проєкт ByvoinoUniver.

Проєкт, буде містити значну кількість функцій а також оголошення, тому доцільно їх розмістити у різних файлах, щоб спростити доступ і роботу з ними. Об’єднувати елементи проєкту можна за різними ознаками і кожний розробник може робити це на свій розсуд. Але розташовувати весь код або велику кількість його у файлі main.cpp забороняється.

У прикладі, що розглядається, файлова структура проєкту після реалізації даного етапу виглядає так, як показано на рисунку 3.

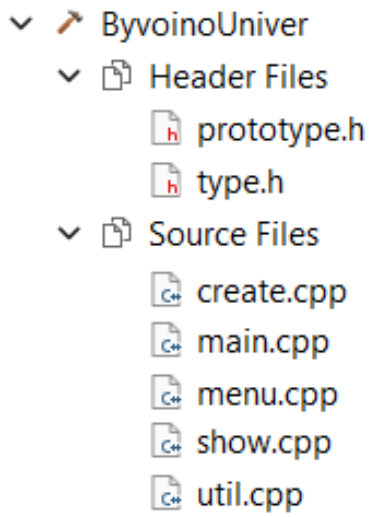

Рисунок 3 – Файлова структура проєкту

2.1.1 Додавання файлів до проєкту

В якості прикладу розглянемо додавання до проєкту заголовного файлу type.h, який буде містити оголошення шаблонів структур, що будуть використовуватися в проєкті.

Щоб створити такий файл і додати його до складу проєкту скористаймося функцією контекстного меню Add New… для теки з файлами проєкту (виклик правою кнопкою миші), рисунок 4. Далі вибираємо діалог створення файлу C++Header File (або C++Source File для створення .cpp файлу). У наступному діалозі вводимо ім’я файлу (розширення можна не вводити) і натискаємо кнопку Next. Після цього на екрані з’являється текст файлу CMakeList.txt. До розділу addExecutable треба додати рядок з ім’ям створеного файлу, яке знаходиться в буфері обміну. Після збереження файлу CMakeList.txt (Ctrl+S) новий файл потрапить до складу проєкту.

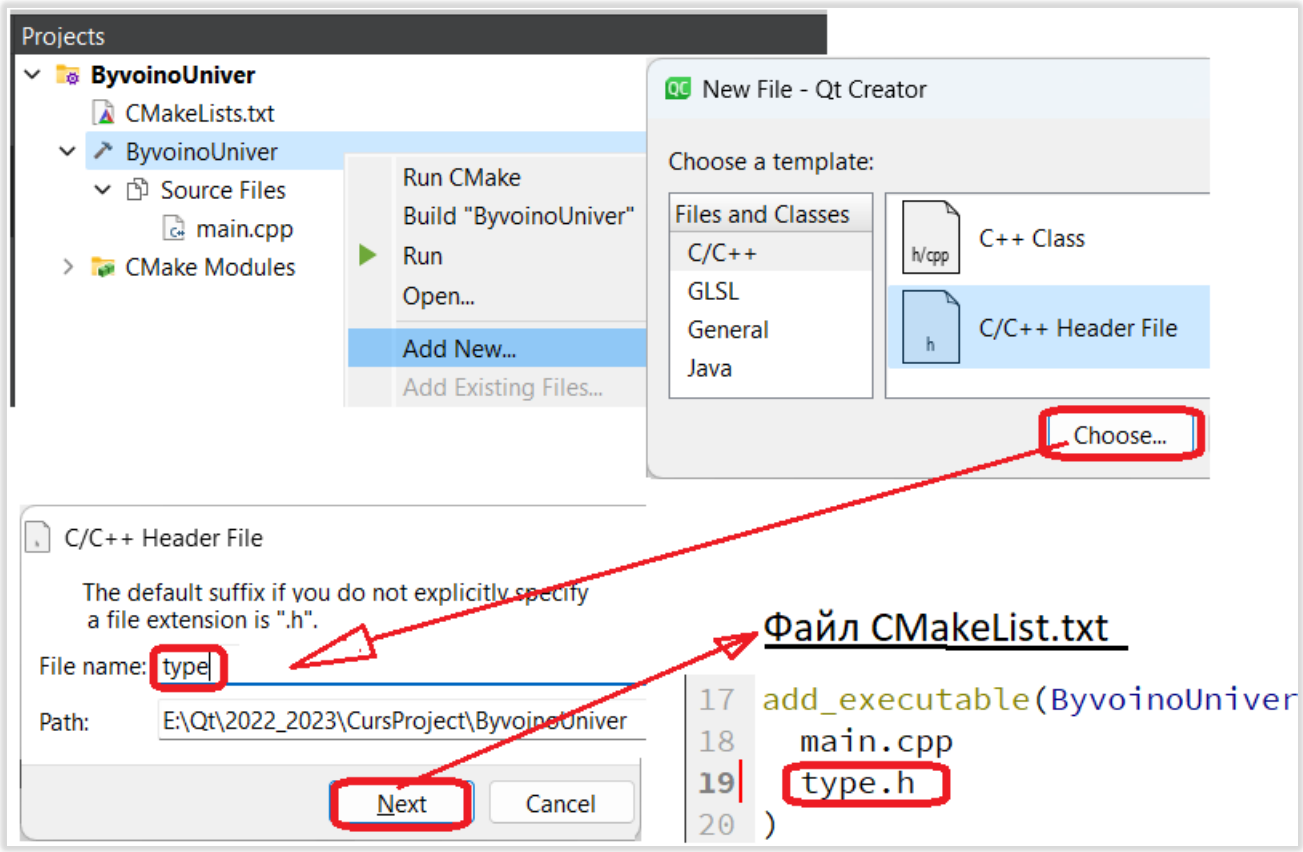

Рисунок 4 – Послідовність діалогових вікон для створення type.h

2.1.2 Оголошення структур у файлі type.h

Початковий вигляд файлу наведено на рисунку 5.

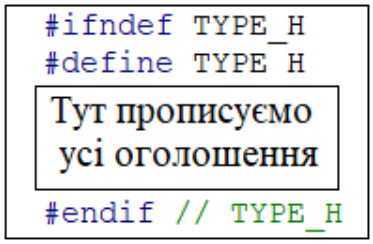

Рисунок 5 – Початковий вигляд файлу type.h

У файлі вже прописано директиви умовної компіляції, які захищають програму від повторного включення заголовних файлів. Докладно про ці директиви можна почитати у розділі 17.3.2 підручнику, сторінка 372. Їх не треба вилучати, а оголошення типів робити після другої директиви, перед останньою.

Можна також цю класичну конструкцію замінити викликом директиви #pragma once на початку файлу.

Шаблони структур оголосимо відповідно до таблиць 2, 3 та 4. Виходячи з того, що структури мають перехресні посилання, зробимо попереднє оголошення їх імен, лістинг 1.

Підключаємо також заголовний файл <QTypeInfo> для того, щоб можна було користуватися скороченими назвами типів з модіфікаторами, наприклад uint замість unsigned int.

Лістинг 1 – Оголошення шаблонів структур

```
#include <QTypeInfo>

struct Univer;
struct Dept;
struct Teacher;

struct Univer{
    //information fields
    char *name;
    int level;
    //service fields
    Dept **deptArray;
    uint arrSize;
    uint deptCount;
};

struct Dept{
    //information fields
    char *name;
    uint hours;
    int spec;
    //service fields
    Teacher *firstTeacher;
    Univer *parent;
```

```
};

struct Teacher{
    //information fields
    char* name;
    char* post;
    uint birthYear;
    //service fields
    Teacher *nextTeacher;
    Dept *parent;
};
```

2.1.3 Створення файлу для прототипів функцій

Додаймо до проєкту ще один заголовний файл prototype.h де будемо зберігати директиви прототипи функцій.

До цього файлу ми також додаємо усі потрібні директиви **#include** та **using namespace.** Також додаймо включення файлу type.h. Це позбавить нас необхідності підключати ці директиви у кожному з файлів .cpp. Достатньо буде **#include “prototype.h”.**

Оголосимо також дві зовнішні глобальні змінні – root та logFileName, які будуть доступні у файлах, де буде підключено prototype.h. Перша з цих змінних буде містити вказівник на корінь дерева, друга – ім’я файлу протоколу роботи програми. Пам’ять для цих змінних буде виділено пізніше, у файлі main.cpp. Там же ці змінні будуть проініціалізовані.

Фрагмент цього файлу наведено у лістингу 2. В ньому наведено прототипи функцій, які створюються на даному етапі розробки проєкту.

Прототипи доцільно групувати відповідно до файлів, де будуть знаходитися їх реалізації.

Лістинг 2 – Початковий варіант файлу prototype.h

```
#pragma once

#include <cstring>
#include <iostream>
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include "type.h"
using namespace std;

//Зовнішні глобальні змінні
extern Univer *root; //корінь дерева
extern FILE *logFile ; //ім'я файлу протоколу

//функції файлу create.cpp
Univer* createDefaultTree();

//функції файлу menu.cpp
int mainMenu(void*);

//функції файлу show.cpp
int showProjectInfo(void *);
int showTree(void*);
int showLogFile(void *);

//функції файлу util.cpp
void pause();
int rootExist();
int getInt(const char* prompt, int min, int max);
char* getStr(const char* prompt);
```

2.1.4 Реалізація допоміжних функцій

Особливість цих функцій полягає в тому, що вони будуть використовуватися іншими функціями в різних режимах роботи. Зокрема, це організація зупинок виконання програми, виведення повідомлень до протоколу роботи програми, введення числової та текстової інформації, перевірка існування дерева.

Зберігати ці функції будемо у файлі util.cpp, тож потрібно створити такий файл і включити його у проєкт.

До створеного файлу слід додати директиву #include "prototype.h". Цю директиву слід буде додавати і до інших файлів .cpp.

2.1.4.1 Функція pause()

Ця допоміжна функція зупиняє виконання програми до натискання будь-якої клавіші. Така функція є і у стандартній бібліотеці операційної системи і, якщо використовується англомовний або україномовний варіант ОС, тоді функцію можна і не створювати. Але студенти часто використовують неліцензійні російськомовні версії ОС. В цьому випадку функцію треба створити обов'язково і саме її використовувати для організації паузи у виконанні програми. Використання мови агресора і окупанта у застосунку буде, як мінімум, неетичним.

Код функції pause наведено у лістингу 3. Її слід розташувати у файлі util.cpp і додати її прототип до файлу prototype.h.

Лістинг 3 – Код функції pause

```
void pause(){
    cout << "\nДля продовження натисніть будь-яку клавішу\n";
    getch();  //Потребує #include <conio.h>
}
```

2.1.4.2 Функція rootExiste()

Багато операцій з деревом, які ми далі будемо реалізовувати, можливі тільки в тому випадку, якщо дерево існує. Ця допоміжна функція буде перевіряти, чи існує корінь.

Якщо корінь не існує, то на консоль виводиться повідомлення про це і програма зупиняється, щоб користувач мав можливість прочитати повідомлення.

Цю функцію доцільно викликати перед виконанням операцій з деревом, щоб уникнути збоїв у програміол.

Код функції наведено у лістингу 4. Її слід розташувати у файлі util.cpp і додати прототип до файлу prototype.h.

Лістинг 4 – Код функції printToLog

```
int rootExist(){
    if(root == NULL){
        cout << "Tree does not exist!\n";
        pause();
    }
    return root != NULL;
}
```

2.1.4.3 Функція введення рядка символів

Користувачеві додатку, що розробляється, у процесі спілкування з програмою доведеться вводити числову і текстову інформацію через консоль. Використання для цього об'єкта cin та методу getline() цього об'єкта, а також функції gets() або gets_s, у деяких ситуаціях є причиною виникнення збоїв у програмі. Для того, щоб уникнути таких небажаних випадків і спростити введення даних, створімо функцію для введення рядка символів. Ця функція в якості параметра приймає рядок, який буде показано на екрані в якості запрошення для введення даних. Функція виділяє необхідну пам'ять для введеного рядка і повертає вказівник на нього. Її можна просто скопіювати, лістинг 5.

Лістинг 5 – Допоміжна функція для введення рядка символів

```
char* getStr(const char* prompt){
    cout << prompt;
    string s;
    fflush(stdin);
    getline(cin,s);
    char* cStr = new char[s.length() +1];
    strncpy(cStr, s.c_str(),s.length() +1);
    return cStr;
}
```

2.1.4.4 Функція введення цілого числа в заданому діапазоні

Для введення цілих чисел ми будемо використовувати ще одну допоміжну функцію. Ця функція в якості параметра також приймає рядок, який буде показано на екрані в якості запрошення для введення даних. Крім тексту запрошення функція приймає мінімальне та максимальне допустимі значення числа, що вводиться. Якщо число, що введено, не потрапляє в заданий діапазон – запит на введення повторюється.

Код функції наведено в лістингу 6 і його можна скопіювати.

Лістинг 6 – Допоміжна функція для введення цілого числа

```
int getInt(const char* prompt, int min, int max){
    int num; string s;
    do{
        printf("%s(%d..%d) ", prompt,min,max);
        fflush(stdin);
        getline(cin,s);
        num = atoi(s.c_str());
    }while(num < min || num > max);
    return num;
}
```

2.1.5 Створення функцій для роботи з деревом

Тут мі підготуємо функції проєкту, які будуть викликатися функцією main. Не забуваймо прототипи функцій додавати до файлу prototype.h.

2.1.5.1 Функція showProjectInfo

Ця функція, лістинг 7, буде виводити на консоль інформацію про проєкт. Вона буде викликатися одразу після запуску програми. Також до неї можна буде звертатися через меню користувача, тому її заголовок створімо відповідно з вимогами до функцій меню проєкта, про що мова буде йти пізніше.

Розташуємо функцію у файлі show.cpp, який потрібно створити.

Лістинг 7 – Код функції showProjectInfo

```
int showProjectInfo(void *){
    system("cls");
    cout << "Проєкт 'Університет'.\n";
    cout << "Розробник Бивойно П.Г., кафедра ІКС.\n";
    cout << "\nКорінь: Університет "
            "\n\tполя: назва, рівень акредитації.\n";
    cout << "1-й рівень: Кафедра"
            "\n\tполя: назва, навантаження в годинах, чи випускаюча.\n";
    cout << "2-й рівень: Викладач"
            "\n\tполя: ім'я, посада, рік народження.\n";
    pause();
    return 0;
};
```

2.1.5.2 Функція createDefaultTree

На даному етапі ми підготуємо тільки заготовку для цієї функції, код якої наведено у лістингу 8. Функцію розташуємо у файлі create.cpp, який потрібно створити і до якого треба додати директиву #include "prototype.h"

На даному етапі функція буде тільки виводити повідомлення до лог-файлу.

Лістинг 8 – Заготовка функції createDefaultTree

```
Univer* createDefaultTree(){
    fprintf_s(logFile,"\n\nПочинаємо створювати дефолтне дерево.");
    Univer *univer;
    fprintf_s(logFile,"\nДефолтне дерево створено\n");
    return univer;
};
```

2.1.5.3 Функція showTree

На даному етапі ми підготуємо тільки заготовку для цієї функції, код якої наведено у лістингу 9, і її створено відповідно з вимогами до функцій меню. На даному етапі тут тільки виводиться повідомлення до лог-файлу.

Лістинг 9 – Заготовка функції showTree

```
int showTree(void*){
    if(!rootExist()) return 0;
    fprintf_s(logFile,"\nДемонстрація дерева.");
    system("cls");
    cout << "\n    --- Tree structure ---\n";
    pause();
    fprintf_s(logFile,"\nДемонстрацію дерева  завершено.");
    return 0;
}
```

2.1.5.4 Функція showLogFile

Ця функція, лістинг 10, забезпечує виведення лог-файлу на консоль.

Лістинг 10 – Код функції showLogFile

```
int showLogFile(void *){
    system("cls");
    char buf[512];
    rewind(logFile);
    while(fgets(buf,512,logFile)!=NULL){
        cout << buf;
    }
    pause();
    return 0;
}
```

2.1.6 Функція активізації головного меню

На даному етапі створюємо тільки заготовку цієї функції, яка теж буде викликатися у функції main. Поки що тут тільки виводиться повідомлення до лог-файлу, лістинг 11.

Функцію розташуємо у файлі menu.cpp. Не забудьте додати до цього файлу директиву #include "prototype.h".

Лістинг 11 – Заготовка функції mainMenu

```
//Головне меню застосунку
int mainMenu(void *ptr){
    fprintf_s(logFile,"\nВиклик головного меню застосунку");
    return 0;
}
```

2.1.7 Реалізація функції main

Заготовка функції main з’явилась разом із файлом main.cpp після створення проєкту. Тепер ми наповнимо її кодом.

Перш за все перед функцією оголошуємо глобальні змінні для кореня дерева і файлу протоколу. Саме тут для цих змінних буде виділено пам’ять. Нагадаємо, що у заголовному файлі prototype.h оголошено, що ці змінні є зовнішніми і завдяки цьому вони доступні у всіх файлах .cpp, де підключено цей заголовний файл.

У самій функції main ми реалізуємо сценарій роботи із застосунком, а саме:

– відкриваємо файл протоколу і виводимо туди перше повідомлення. До назви файлу протоколу слід включити прізвище студента;

– після запуску програми на екрані має з’явитися інформація про проєкт;

– далі створюємо дефолтне дерево та ініціалізуємо змінну root;

– наступним кроком має бути виведення на консоль дефолтного дерева;

– далі на консолі має з’явитися головне меню, через яке користувач буде працювати із застосунком;

– після завершення операцій з деревом і виходом із головного меню на екрані має з’явитися протокол роботи з деревом.

– Після цього файл протоколу зак4ривається і робота програми завершується.

Відповідний код функції main наведено у лістингу 12.

Сценарій роботи з проєктом має бути саме таким і студенти в своїх проєктах мають цього дотримуватися.

Нічого іншого у файлі main.cpp s в функції main не має бути!

Лістинг 12 – Текст файлу main.cpp

```
#include "prototype.h"

Univer *root;
FILE *logFile;

int main(int argc, char *argv[]){
    logFile = fopen("logByvoino", "wt+");
    fprintf(logFile,"\nПроєкт 'Університет' стартував.");
    showProjectInfo(NULL);
    root = createDefaultTree();
    showTree(NULL);
    mainMenu(NULL);
    fprintf(logFile, "\nРоботу з проєктом 'Університет' завершено.\n");
    showLogFile(NULL);
    fclose(logFile);
    return 0;
}
```

Після реалізації функції main проект можна протестувати. На консолі мають з’явитися повідомлення про етапи виконання проєкту і протокол.

3 СТВОРЕННЯ ДЕФОЛТНОГО ДЕРЕВА

На цьому етапі треба створити початкове (дефолтне) дерево і продемонструвати його на консолі. Для реалізації цього завдання треба зробити наступне:

– оголосити і реалізувати функції створення вузлів дерева у програмному режимі, без участі користувача;

– написати функцію розширення масиву вказівників, яка може знадобитися в разі розширення переліку кафедр університету;

– оголосити і реалізувати функції додавання нащадків до вузлів дерева у програмному режимі, без участі користувача;

– наповнити кодом функцію створення дефолтного дерева;

– наповнити кодом функцію виведення дерева на консоль.

3.1 Функції створення вузлів дерева у програмному режимі

Таких функцій буде три, відповідно до кількості різновидів структур для вузлів дерева.

Ці функції мають сформувати відповідну структуру і повернути вказівник на неї.

Параметрами функції будуть значення полів структури, які будуть використані для ініціалізації цих полів.

Крім того, у функції мають бути проініціалізовані службові поля.

Функції розташуємо у файлі create.cpp.

Також слід оголосити створені функції у файлі prototype.h.

3.1.1 Функція створення структури «Університет»

Код функції наведено у лістингу 13.

Лістинг 13 – Код функції створення структури «Університет»

```
Univer *createUniver(char *name, int level){
    Univer *univer = new Univer();
    univer->name = name;
    univer->level = level;
    univer->arrSize = 2;
    univer->deptArray = new Dept*[univer->arrSize];
    univer->deptCount = 0;
    fprintf_s(logFile,"\nСтворено університет %s ",univer->name);
    return univer;
}
```

3.1.2 Функція створення структури «Кафедра»

Код функції наведено у лістингу 14.

Лістинг 14 – Код функції створення структури «Кафедра»

```
Dept *createDept(char *name, int hours, int spec){
    Dept *dept = new Dept();
    dept->name = name;
    dept->hours = hours;
    dept->spec = spec;
    dept->firstTeacher = NULL;
    dept->parent = NULL;
    fprintf_s(logFile,"\nСтворено кафедру %s", dept->name);
    return dept;
}
```

3.1.3 Функція створення структури «Викладач»

Код функції наведено у лістингу 15.

Лістинг 15 – Код функції створення структури «Викладач»

```
Teacher *createTeacher(char *name, char *post, int birthYear){
    Teacher *teacher = new Teacher();
    teacher->name = name;
    teacher->post = post;
    teacher->birthYear = birthYear;
    teacher->parent = NULL;
    teacher->nextTeacher = NULL;
    fprintf_s(logFile,"\nСтворено викладача %s", teacher->name);
    return teacher;
}
```

3.2 Додавання нащадків до вузлів дерева у програмному режимі

Для реалізації цього завдання, перш за все, знадобиться дві функції додавання. Перша - для додавання кафедр до університету, друга – для додавання викладачів до кафедри. Але оскільки структури мають бути розташовані в певному порядку, то для порівняння структур потрібні відповідні функції, які теж треба створити. Крім того, може виникнути необхідність розширення масиву вказівників у разі збільшення кількості кафедр. Тому потрібна ще функція розширення масиву вказівників.

Створімо файл add.cpp для збереження цих функцій. Нагадаємо, що до файлу додати директиву #include "prototype.h".

Також слід оголосити створені функції у файлі prototype.h.

3.2.1 Функції порівняння структур

3.2.1.1 Функція порівняння кафедр

Кафедри будемо впорядковувати перш з все за ознакою випускаюча чи звичайна. Першими у переліку будуть випускаючі. Якщо за цією ознакою кафедри не розрізняються, тоді вони впорядковуються за назвою. Код функції порівняння наведено у лістингу 16.

Лістинг 16 – Код функції порівняння кафедр

```
int cmpDept(Dept *d1, Dept *d2){
    if(d1->spec != d2->spec)
        return d2->spec;
    return strcmp(d1->name, d2->name);
}
```

3.2.1.2 Функція порівняння викладачів

Викладачів будемо впорядковувати за прізвищем. Функція, що допоможе вирішити це завдання представлена у лістингу 17.

Лістинг 17 –Код функції порівняння викладачів

```
int cmpTeacher(Teacher *f1, Teacher *f2){
    return strcmp(f1->name, f2->name);
}
```

3.2.2 Функція розширення масиву вказівників

Ця функція буде викликатися функцією додавання кафедри у випадку, якщо обсяг пам’яті, що була виділена для масиву, вичерпався. Код функції наведено у лістингу 1. Зверніть увагу, що для виведення повідомлення у протокол, тут використовується інший спосіб.

Лістинг 18 – Код функції розширення масиву вказівників

```
void growDeptAr(Univer *pU){
    int newSize = 1.5 * pU->arrSize + 0.5;
    Dept** newAr = new Dept*[newSize];
    // Copy old array to new
    memcpy(newAr, pU->deptArray, pU->arrSize*sizeof(Dept*));
    pU->arrSize = newSize;
    Dept **oldAr = pU->deptArray;
    pU->deptArray = newAr;
    delete[] oldAr;
    fprintf_s(logFile,"\nРозмір масиву кафедр збільшено до %d",newSize);
}
```

3.2.3 Функціїї додавання нащадків

Кожна з цих функцій має свою специфіку, яка пов’язана з особливостями способу зберігання даних у відповідних структурах.

3.2.3.1 Функція додавання кафедри до університету

Особливість зберігання даних в масиві вимагає постійного контролю за обсягом пам’яті, що використовується для зберігання даних. Якщо обсяг пам’яті, що було виділено, вже використано, потрібно виділяти нову ділянку пам’яті більшого розміру і переписувати туди масив, а стару ділянку звільняти.

Для збереження порядку розташування даних після додавання нового елемента потрібно шукати місце для вставки цього елемента, після чого робити зсув елементів масиву праворуч від знайденого місця, щоб звільнити місце для вставки.

Код функції, що реалізує ці завдання наведено в лістингу 19.

Лістинг 19 – Код функції додавання кафедри

```
void addDept(Univer *pU, Dept*pD){
    pD->parent = pU;
    if(pU->arrSize == pU->deptCount)
        growDeptAr(pU);
    int n = pU->deptCount;
    while(n > 0 && cmpDept(pU->deptArray[n - 1], pD) >  0){
        pU->deptArray[n] =  pU->deptArray[n - 1];
        n--;
    }
    //Put new department into ordered array
    pU->deptArray[n] = pD;
    pU->deptCount++;
    fprintf_s(logFile,"\nДодано кафедру %s.",pD->name);
}
```

3.2.3.2 Функція додавання викладача до кафедри

Ця функція відрізняється від попередньої, тому що нащадки зберігаються не у масиві, а у списку.

Алгоритм додавання розпадається на три гілки.

Якщо список пустий, то посилання на новий елемент записується у поле firstTeacher структури кафедра.

Якщо новий елемент менший ніж елемент, на який вказує firstTeacher, то новий елемент має стати першим, а firstTeacher наступним.

В іншому випадку доводиться йти списком і шукати місце для нового елемента і вставити його туди.

Код функції наведено у лістингу 20.

Лістинг 20 – Код функції додавання викладача до кафедри

```
void addTeacher(Dept *pD, Teacher *pT){
    pT->parent = pD;
    //Insert new department
    if(pD->firstTeacher == NULL)// List is empty
        pD->firstTeacher = pT;
    else if(cmpTeacher(pT, pD->firstTeacher) < 0){
        // New element is less then first element
        pT->nextTeacher = pD->firstTeacher;
        pD->firstTeacher = pT;
    }
    else{
        Teacher *p = pD->firstTeacher;
        // search place for new element
        while(p->nextTeacher != NULL
              && cmpTeacher(pT, p->nextTeacher) >= 0)
            p = p->nextTeacher ;
        // Inserting new elemet
        pT->nextTeacher = p->nextTeacher;
        p->nextTeacher = pT;
    }
     fprintf_s(logFile,"\nДодано викладача %s до кафедри %s.",
              pT->name,pD->name);
}
```

3.2.4 Створення дефолтного дерева

Створюючи це дерево ми тестуємо розроблені функції, тому бажано налаштовувати параметри нащадків, вибирати їх кількість і порядок додавання таким чином, щоб працювали всі функції і гілки алгоритмів.

На попередньому етапі заготовку цієї функції у файлі create.cpp вже було створено, тепер треба наповнити її кодом. У лістингу 21 наведено dfhsfyn реалізації цієї функції.

Лістинг 21 – Функція створення дефолтного дерева

```
Univer* createDefaultTree(){
    fprintf_s(logFile,"\n\nПочинаємо створювати дефолтне дерево.");
    Univer *univer = createUniver(new char[]{
        "НУ'Чернівська політехніка'"}, 4);
    Dept *dept1 = createDept(new char[]{"ІКС"}, 6000, 1);
    Dept *dept2 = createDept(new char[]{"Історії"}, 4000, 0);
    Dept *dept3 = createDept(new char[]{"ПІ"}, 2000, 1);
    addDept(univer, dept1);
    addDept(univer, dept2);
    addDept(univer, dept3);

    Teacher *t1 = createTeacher(new char[]{"Казимир В.В"},
    new char[]{"професор"}, 1953);
    Teacher *t2 = createTeacher(new char[]{"Бивойно П.Г"},
    new char[]{"доцент"}, 1943);
    Teacher *t3 = createTeacher(new char[]{"Базилевич В.М"},
    new char[]{"завідувач кафедрою"}, 1985);
    Teacher *t4 = createTeacher(new char[]{"Білоус І.В"},
    new char[]{"завідувач кафедрою"}, 1983);
    Teacher *t5 = createTeacher(new char[]{"Денисов Ю.О"},
    new char[]{"завідувач кафедрою"}, 1975);
    addTeacher(dept2,t5);
    addTeacher(dept1,t2);
    addTeacher(dept1,t1);
    addTeacher(dept3,t4);
    addTeacher(dept1,t3);
    fprintf_s(logFile,"\nДефолтне дерево створено\n");
    return univer;
};
```

3.2.5 Відображення дерева на консолі

Ця функція дозволяє переглянути структуру дерева на екрані. Оскільки кількість інформаційних полів невелика, то можна виводити всю інформацію про структури (окрім службової).

Нагадаємо, що заготовку цієї функії ми вже створювали, тепер треба наповнити її кодом.

Для виведення дерева потрібно два цикли. Зовнішній цикл забезпечує обробку масиву вказівників, а внутрішній – обробку списку викладачів.

Кожний наступний рівень дерева виділяється додатковою табуляцією.

Код функції наведено у лістингу 22.

Лістинг 22 – Код демонстрації дерева на консолі

```
int showTree(void*){
    if(!rootExist()) return 0;
    fprintf_s(logFile,"\nДемонстрація дерева.");
    system("cls");
    cout << "\n    --- Tree structure ---\n";
    cout << "\nУніверситет " << root->name << ", рівень акредитації "
         << root->level;
    //Department loop
    Dept ** dAr = root->deptArray;
    for(int i = 0; i < root->deptCount; i++){
        cout <<  "\n\tКафедра " << dAr[i]->name
             << ", нававнтаження " << dAr[i]->hours << " годин, "
             << (dAr[i]->spec ? "випускаюча.":"загальна.");
       // Teacher loop
       Teacher *pT = dAr[i]->firstTeacher;
       while(pT != NULL){
           cout << "\n\t\t" << pT->name << ",  "<< pT->post
                << ", рік народження " << pT->birthYear;
           pT = pT->nextTeacher;
       }
    }
    pause();
    fprintf_s(logFile,"\nДемонстрацію дерева  завершено.");
    return 0;
}
```

Після реалізації цієї функції можна запустити проект на виконання і побачити на екрані створене дерево, або почати роботу з пошуку та виправлення помилок.

4 РЕАЛІЗАЦІЯ РОБОТИ З ДЕРЕВОМ ЧЕРЕЗ МЕНЮ

На цьому етапі ми маємо забезпечити доступ до створених функцій через консольне меню. Кількість функцій, які забезпечують роботу з деревом чимала. Їх кількість ще збільшиться, коли доведеться розширювати перелік функцій під час захисту проєкту. Але спеціалісти з розробки інтерфейсів користувача наголошують, що людина нормально сприймає меню, коли кількість пунктів в цьому меню не перевищує від 5 до 7. Тому доцільно зробити меню багаторівневим.

Також слід врахувати, що меню, яке буде розроблено на даному етапі, в подальшому (на наступних етапах і під час захисту) буде розширюватися. Тому доцільно зробити його простим для розширення, а для цього бажано, щоб елементи меню були стандартними.

На цьому етапі ми вирішимо наступні задачі:

– оголосимо тип функції меню;

– оголосимо структуру для елемента меню;

– напишемо функцію активізації меню;

– завершимо реалізацію функції, яка буде формувати вміст головного меню;

– сформуємо функції для формування вмісту меню другого рівня;

– напишемо функції виклику операцій додавання нащадків до вибраного вузла через меню;

– напишемо функції виведення на консоль переліку нащадків вибраного вузла.

4.1 Оголошення типу функції меню

Оголошуючи цей тип, ми стандартизуємо функції, які можуть викликатися через меню. Це дозволить створити стандартний шаблон структури елемента меню, де функція буде полем цієї структури.

Функція меню буде повертати ціле число, яке можна використовувати для аналізу результатів виконання і прийняття рішення про подальші дії.

В якості параметра функція приймає вказівник типу void. Зазвичай, це буде вказівник на структуру вузла дерева, для якої викликається меню.

Виходячи з того, що тип функції меню потрібен тільки для формування елементів меню, розташуємо це оголошення у файлі menu.cpp після директиви include перед реалізацією функцій. Код оголошення наведено в лістингу 23.

Лістинг 23 – Оголошення типу функції меню

```
//Тип для функцій, що викликаються через меню
typedef int MenuFunc(void*);
```

4.2 Оголошення структури для пункту меню

Ця структура містить два поля – назву пункту меню, яка буде виводитися на консоль під певним номером, і поле типу функції меню, яке буде містити вказівник на функцію (фактично ім’я функції), яка відповідає вибраному пункту меню.

Код оголошення наведено в лістингу 24.

Цей шаблон розташуємо також у файлі menu.cpp після оголошення типу функції меню.

Лістинг 24 – Шаблон структури для пункту меню

```
//Структура для пункиу меню
struct MenuUnit{
    //Текст пункту меню
    const char *text;
    //Вказівник на функцію, яка викликається цим пунктом меню
    MenuFunc *func;
};
```

4.3 Функція для активізації меню

Ця функція використовується для виведення меню на консоль, лістинг 25. Функцію теж розташуємо у файлі menu.cpp, після оголошення шаблону структури елемента меню. Тоді її навіть можна не прописувати у файлі prototytpe.h.

Вміст меню передається до функції runMenu у вигляді масиву пунктів меню. Кожен елемент цього масиву має бути структурою типу MenuUnit. Окрім того параметрами функціє є назва меню і вказівник, який буде передаватися функції, що пов’язана з даним пунктом меню.

Ознакою кінця масиву пунктів меню є значення NULL для вказівника на функцію і звернення до цього пункту буде означати, що роботу з меню завершено. Вихід з меню відбувається також у тому випадку, коли фунція цього пункту меню повертає не нульове значення.

Нумерація пунктів меню на консолі починається з 1. Це пов’язано з тим, що функції введення чисел з консолі можуть повертати 0, якщо введено не цифру.

Лістинг 25 – Код функції для активізації меню

```
void runMenu(MenuUnit menu[], const char *title, void* ptr){
    while(true){
        system("cls");
        printf("    --- %s ---\n", title);
        //Виводимо перелік функцій меню з номерами, що починаються з 1
        //Цикл завершується, якщо вказівик на функцію дорівнює NULL
        int cnt = 0;
```

```
        for(;; cnt++){
            printf("%2d. %s\n", cnt+1, menu[cnt].text);
            if(menu[cnt].func == NULL)
                break;
        }
        //Запит на вибір варіанта меню
        int v = getInt(" Enter your choice number: ",
                       1, cnt + 1);
        //Аналіз вибраного номера
        if(menu[v - 1].func == NULL)
            return; //Завершення роботи меню
        // Виклик вибраної функції
        int result = menu[v - 1].func(ptr);
        //Аналіз результату виконання функції
        if( result != 0){
            return; //Завершення роботи меню
        }
    }
}
```

4.4 Функція формування головного меню

Завдання цієї функцій – сформувати масив потрібних елементів меню першого рівня і активізувати це меню за допомогою функції runMenu. В якості вказівника, який передається до функції, логічно передавати вказівник на корінь дерева, але зважаючи на те, що корінь дерева оголошено як глобальну змінну, то можна передавати і NULL.

Код цієї функції наведено в лістингу 26. Нагадаємо, що заготовку цієї функції було вже створено раніше, але там тільки виводилося повідомлення до лог-файлу.

Лістинг 26 – Функція формування головного меню

```
//Головне меню застосунку
int mainMenu(void *ptr){
    fprintf_s(logFile,"\nВиклик головного меню застосунку");
    MenuUnit menu[] = {
        {"University operation", menuUniverOperation},
        {"Department operation", menuDeptOperation},
        {"Teacher operation", menuTeacherOperation},
        {"Tree operation", menuTreeOperation},
        {"Query operation", menuQueryOperation},
        {"Exit from project",NULL}
    };
    runMenu(menu, "Main menu", ptr);
    return 0;
}
```

Тут ми в якості значень полів funk елементів масиву використовуємо ідентифікатори функцій, які мають викликатися після вибору відповідного пункту меню. Ці функції ще не створено, тому компілятор буде повідомляти про помилки.

Щоб уникнути повідомлень про помилки можна створити шаблони цих функцій у файлі menu.cpp відповідно з типом функцій меню і оголосити їх у файлі prototype.h.

Після цього можна протестувати проєкт і побачити на консолі головне меню.

Як бачимо, завдання головного меню полягає у виклику функцій меню наступного рівня. Тому далі переходимо до формування цих функцій.

4.5 Функції формування меню другого рівня

На даному етапі ми реалізуємо тільки деякі з цих функцій і відповідні меню сформуємо не в повному обсязі. Завершимо їх розробку на наступних етапах. Розташувати ці функції теж доцільно у файлі menu.cpp

4.5.1 Меню для операцій з деревом

Заготовку відповідної функції, можливо, вже було створено. Включимо, поки що, до цього меню тільки ті функції, які вже реалізовано, а саме – демонстрацію дерева і файлу протоколу. Вказівник, що передається у функцію, не використовується.

Код функції формування цього меню наведено у лістингу 27.

Лістинг 27 – Функція формування меню для операцій з деревом

```
int menuTreeOperation(void* ptr){
     fprintf_s(logFile,"\nВиклик меню операцій з деревом");
     MenuUnit menu[] = {
        {"Show project information", showProjectInfo},
        {"Show tree", showTree},
        {"Show log file", showLogFile},
        {"Return to main menu",NULL}};
    runMenu(menu, "\n    --- Tree operation ---",ptr);
    return 0;
}
```

Тепер можна протестувати цю функцію і викликати для перегляду інформацію про проєкт, дефолтне дерево і файл протоколу.

4.5.2 Реалізація меню для операцій з університетом

4.5.2.1 Функція формування меню операцій з університетом

Заготовку відповідної функції, можливо, також було створено. Включимо, поки що, до цього меню тільки дві функції – додавання нової кафедри через меню та виведення переліку кафедр університету.

Код функції формування цього меню наведено у лістингу 28.

Зверніть увагу, що меню можна активізувати тільки у випадку існування дерева.

Лістинг 28 – Функція формування меню для операцій з університетом

```
int menuUniverOperation(void*){
    if(!rootExist()) return 0;
    fprintf_s(logFile,"\nВиклик меню операцій з університетом");
    MenuUnit menu[] = {
        {"Show list of department", showDeptListByMenu},
        {"Add department", addDeptByMenu},
        {"Return to main menu", NULL},
    };
    runMenu(menu,"Univesity menu", root);
    return 0;
}
```

Сформоване меню посилається на функції, яких ще нема, тож оголосимо ці функції та створімо заготовки для них

Після цього можна протестувати роботу меню і якщо все працює, можна перейти до реалізації завдань меню.

4.5.2.2 Функції для виведення переліку кафедр

Завдання виведення переліку кафедр реалізуємо за допомогою двух функцій. Розташуємо їх у файлі show.cpp.

Перша з них буде виводити на консоль перелік кафедр без заголовка і паузи після виведення. Ця функція буде використовуватися і в інших функціях, які будуть створюватися пізніше, і це дозволить уникнути дублювання коду.

У цій функції для виведення переліку кафедр організовано цикл по масиву вказівників. Код функції наведено в лістингу 29.

Друга функція, на яку посилається меню, буде використовувати попередню і сформує заголовок та паузу після виведення переліку.

Код функції наведено в лістингу 30,

Лістинг 29 – Функція виведення переліку кафедр

```
int showDeptList(void*){
    int n = root->deptCount;
    Dept **dAr = root->deptArray;
    cout <<"\n\tПерелік кафедр університету " << root->name << ":";
    for(int i = 0; i < n; i++){
        printf("\n Кафедра %s, навантаження %d годин, %s.",
               dAr[i]->name, dAr[i]->hours,
               dAr[i]->spec ? "випускаюча":"загальна");
    }
    printf("\n");
    pause();
    return 0;
}
```

Лістинг 30 – Функція меню для перегляду переліку кафедр

```
int showDeptListByMenu(void*){
    cout <<"\n\tПерелік кафедр університету " << root->name << ":";
    showDeptList(NULL);
    printf("\n");
    pause();
    return 0;
}
```

Після реалізації наведених функцій доцільно протестувати їх взаємодію між собою і з меню.

4.5.2.3 Функція додавання кафедри через меню

У цій функції треба реалізувати діалог, через який користувач зможе ввести інформацію, що необхідна для створення структури «кафедра».

Далі реалізується наступний алгоритм:

– отриману інформацію передаємо функції createDept(), яку було створенотна попередньому етапі і яка використовувалася для побудови дефолтного дерева. Функція createDept() повертає вказівник на створену структуру;

– отриманий вказівник передаємо функції addDept(), яку теж було створено раніше, і яка теж використовувалася.

– виводимо на консоль новий перелік кафедр для контролю виконання операції додавання.

Функцію розташуємо у файлі add.cpp.

Реалізацію цієї функції наведено у лістингу 31.

Лістинг 31 – Функція додавання кафедри через меню

```
int addDeptByMenu(void *ptr){
    system("cls");
    cout << " --- New department adding ---\n";
    if(root == NULL){
        cout << " Tree was not created!\n";
        pause();
        return 1;
    }
    char* name = getStr(" Department name is: ");
    int hours = getInt(" Amount of hours is: ", 1000, 40000);
    int spec = getInt(" Is department special? 1-yes, 0-no ",0,1);
    Dept *dept = createDept(name,hours, spec);
    addDept(root, dept);
    cout << "\n Department list after adding";
    showDeptList(NULL);
    pause();
    return 0;
}
```

4.5.3 Реалізація меню для операцій з кафедрою

Особливість цієї задачі полягає в тому, що спочатку потрібно вибирати кафедру, а вже потім вибирати із меню, що робити з цією кафедрою.

Тому перш за все реалізуємо функцію вибору кафедри.

4.5.3.1 Функція вибору кафедри

Ця функція має вивести на консоль перенумерований перелік назв кафедр, надати користувачеві можливість вибрати потрібну кафедру і повернути вказівник на вибрану структуру. Код функції наведено в лістингу 32.

Фактично, ця функція теж формує меню, тому розташуємо її у файлі menu.cpp.

Лістинг 32 – Код функції вибору кафедри

```
Dept *selectDept(){
    system("cls");
    printf("\n   --- Department selection ---\n");
    int nDept = root->deptCount;
    Dept **dAr = root->deptArray;
    for(int i = 0; i < nDept; i++){
        printf("\n %2d. Кафедра %s", i+1, dAr[i]->name);
    }
    printf("\n %2d. Exit\n", nDept + 1);
    int num = getInt(" Enter your choice: ", 1, nDept + 1);
    if(num == nDept + 1)
        return NULL;
    return root->deptArray[num - 1];
}
```

4.5.3.2 Функція створення меню для роботи з кафедрою

Меню формується за два кроки. Перший крок – це вибір кафедри. Другий крок – виведення переліку функцій для роботи з кафедрою. До вибраної функції буде передано вказівник на вибрану кафедру.

Якщо дерево не існує, меню працювати не буде.

На даному етапі до меню додаємо тільки ті функції, які будемо реалізовувати на даному етапі, лістинг 33.

Лістинг 33 – Функція створення меню для роботи з кафедрою

```
int menuDeptOperation(void* ){
    if(!rootExist()) return 0;
    Dept *pD = selectDept();
    if(pD == NULL) return 0;
    fprintf_s(logFile,"\nВиклик меню операцій з кафедрою %s ", pD->name);
    MenuUnit menu[] = {
        {"Show teachers list", showTeacherListByMenu},
        {"Add teacher", addTeacherByMenu},
        {"Return to main menu", NULL}
    };
    runMenu(menu, pD->name, pD);
    return 0;
}
```

Створімо тепер функції, на яке посилається сформоване меню.

4.5.3.3 Функції виведення списку викладачів кафедри

Так само, як і у випадку виведення списку кафедр, реалізуємо це завдання через дві функції. Розташуємо ці функції також у файлі show.cpp.

Код першої функції функції наведено в лістингу 34.

Лістинг 34 – Функція виведення списку викладачів

```
int showTeacherList(Dept* ptr){
    Dept *dept = (Dept*)ptr;
    Teacher * teacher = dept->firstTeacher;
    while(teacher != NULL){
        printf("\n\tTeacher %s, post %s, %d b.y.",
               teacher->name, teacher->post, teacher->birthYear);
        teacher = teacher->nextTeacher;
    }
    return 0;
}
```

Друга функція, лістинг 35, використовує попередню і додає заголовок для переліку і паузу після виведення.

Лістинг 35 – Функція меню для виведення переліку викладачів кафедри

```
int showTeacherListByMenu(void* ptr){
    Dept *dept = (Dept*)ptr;
    printf("\n TeacherList for department %s:", dept->name);
    showTeacherList(dept);
    pause();
    return 0;
}
```

4.5.3.4 Функція додавання викладача до кафедри

Функцію розташуємо у файлі add.cpp. Код функції наведено у лістингу 36.

Лістинг 36 – Функція додавання викладача до кафедри через меню

```
int addTeacherByMenu(void *ptr){
    Dept * dept = (Dept*)ptr;
    char* name = getStr(" Teacher's name is: ");
    char* post = getStr(" Teacher's post is: ");
    int birthYear = getInt(" Teacher's year of birth: ", 1930, 2020);
    Teacher *teacher = createTeacher(name, post, birthYear);
    addTeacher(dept, teacher);
    cout << "\n Teacher list after adding";
    showTeacherList(dept);
    pause();
    return 0;
}
```

Після цього можна протестувати створені функції активізуючи відповідні пункти меню.

4.5.4 Реалізація меню для роботи з викладачами

Це буде найпростіше меню, але перед виведенням цього меню потрібно ще вибирати кафедру і викладача. Функцію вибору кафедри вже реалізовано, тепер реалізуємо функцію вибору викладача.

4.5.4.1 Функція для вибору викладача

Код функції наведено в лістингу 37.

Лістинг 37 – Функція вибору пвикладача

```
Teacher *selectTeacher(Dept *pD){
    system("cls");
    printf("\n   --- %s ---", pD->name );
    printf("\n   --- Teacher selection ---\n");
   //Виводимо нумерований перелік викладачів
    int n = 1;
    Teacher *pT = pD->firstTeacher;
    for(; pT != NULL; n++){
        printf("\n %2d. Викладач %s", n, pT->name);
        pT = pT->nextTeacher;
    }
    printf("\n %2d. Exit\n", n);
    //Запит і введення потрібного номера
    int num = getInt(" Enter your choice: ", 1, n);
    //Пошук вказівника по номеру
    if(num == n) return NULL;
    pT = pD->firstTeacher;
    for(int i = 1; i < num && pT!= NULL; i++)
        pT = pT->nextTeacher;
    return pT;
}
```

4.5.4.2 Функція реалізації меню для роботи з викладачем

Меню формується за три кроки. Перший крок – це вибір кафедри. Другий крок – вибір викладача. Третій крок – виведення переліку функцій для роботи з викладачем. До вибраної функції буде передано вказівник на вибраного викладача.

Якщо дерево не існує меню активізуватися не буде.

На даному етапі до меню додаємо тільки вихід з меню. Всі функції для роботи з викладачем додамо до меню пізніше.

Текст функції наведено в лістингу 38.

Лістинг 38 – Функція виклику меню для роботи з викладачем

```
int menuTeacherOperation(void* ){
    if(!rootExist()) return 0;
    Dept *pD = selectDept();
    if(pD == NULL) return 0;
    Teacher *pT = selectTeacher(pD);
    if(pT == NULL) return 0;
    fprintf_s(logFile,"\nВиклик меню операцій з викладачем %s ", pT->name);
    MenuUnit menu[] = {
        {"Return to main menu", NULL}
    };
    runMenu(menu, pT->name, pT);
    return 0;
}
```

Після реалізації функцій слід протестувати протестувати їх шляхом активізації відповідних пунктів меню.

5 РЕДАГУВАННЯ ТА ВИДАЛЕННЯ ВУЗЛІВ ДЕРЕВА

Функції редагування і видалення вузлів дерева через меню потребують використання фунцій виведення інформації на консоль перед редагуванням та після видалення для того, щоб операції не виконувалися наосліп. Тому спочатку ми реалізуємо функції виведення інформації про вузли. Крім того, забезпечимо доступ до цих функцій через меню.

Таким чином на цьому етапі ми вирішимо наступні задачі:

– додамо до меню вузлів функції перегляду, видалення та редагування;

– оголосимо та реалізуємо функції перегляду інформації про вузли та функції перегляду інформації через меню;

– оголосимо та реалізуємо функції видалення вузлів та функції видалення вузлів через меню;

– оголосимо та реалізуємо функції редагування вузлів та функції редагування через меню.

5.1 Розширення переліку функцій меню для вузлів

Перелік функцій потрібно розширити для меню усіх трьох вузлів.

Як приклад, розглянемо розширене меню вузла «Університет». Код відповідної функції наведено в лістингу 39.

Лістинг 39 – Функція формування меню для вузла «Університет»

```
int menuUniverOperation(void*){
    if(!rootExist()) return 0;
    fprintf_s(logFile,"\nВиклик меню операцій з університетом");
    MenuUnit menu[] = {
        {"Edit University information", editUniverByMenu},
        {"Delete University", delUniverByMenu},
        {"Delete all department", delAllDeptByMenu},
        {"Show University information", showUniverInfoByMenu},
        {"Show list of department", showDeptListByMenu},
        {"Add department", addDeptByMenu},
        {"Return to main menu", NULL},
    };
    runMenu(menu,"Univesity menu", root);
    return 0;
}
```

Виходячи з того, у коді використовуються посилання на функції, які ще не було створено (їх обведено у лістингу), слід оголосити ці функції і створити заготовки для них відповідно до типу для функцій меню.

Меню для вузла «Кафедра» має бути розширено аналогічними функціями.

Меню вузлів останнього рівня простіше. Там випадають групові

операції, бо нащадків у структур цього рівня нема.

Функції виведення інформації рекомендуємо розташувати у файлі show.cpp, вилучення – у файлі del.cpp, редагування – у файлі edit.cpp.

5.2 Виведення інформації про вузли

На цьому кроці слід створити по дві функції для кожного вузла – функцію, яку будуть використовувати інші програми і функцію виклику цієї операції через меню.

У функціях для інших програм будуть виводитися тільки ті поля, які доступні для редагування.

У функціях, які будуть викликатися через меню буде виводитися і інша інформація, яка характеризує структуру. Для отримання такої інформації доведеться написати допоміжні функції для обчислення деяких узагальнених характеристик вузлів

В якості прикладу розглянемо функції для структури «Кафедра».

5.2.1 Приклади функцій виведення інформації

5.2.1.1 Допоміжна функція підрахунку кількості викладачів кафедри

Цю функцію, лістинг 40, будемо використовувати у функції виведення інформації про кафедру. Її можна розташувати у файлі util.cpp.

Лістинг 40 – Функція підрахунку кількості викладачів кафедри

```
int calcTeachDept(Dept *ptr){
    int count = 0;
    Teacher *pT = ptr->firstTeacher;
    while(pT != NULL){
        count++;
        pT = pT->nextTeacher;
    }
    return count;
}
```

5.2.1.2 Функція виведення інформації про кафедру

Реалізація функції дуже проста і не потребує пояснень. Код функції наведено в лістингу 41.

Лістинг 41 – Функція виведення інформації про кафедру

```
void showDeptInfo(Dept *ptr){
    cout << " Назва кафедри: " << ptr->name << endl;
    cout << " Навантаження: " << ptr->hours << " годин" << endl;
    cout << " Кафедра " << (ptr->spec? "випускаюча" : "загальна") << endl;
}
```

5.2.1.3 Функція меню для виведення інформації про кафедру

Особливість функції виведення інформації через меню пролягає в наявності заголовка і виклику функції pause() після завершення виведення інформації. Крім інформації, що виводить функція showDeptInfo, виводиться додаткова інформація про кафедру. Код функції наведено в лістингу 42.

Лістинг 42 – Функція меню для отримання інформації про кафедру

```
int showDeptInfoByMenu(void *ptr){
    Dept *pDpt  = (Dept*)ptr;
    system("cls");
    cout << "__Інформація про кафедру__\n";
    cout << " Кафедра університету " << pDpt->parent->name << endl;
    showDeptInfo(pDpt);
    cout << " На кафедрі працює " << calcTeachDept(pDpt) << "викладачів\n";
    pause();
    return 0;
}
```

Після реалізації цієї функції доцільно протестувати її і пов’язані з нею функції шляхом виклику відповідного пункту меню

5.3 Видалення вузлів дерева

Видалення вузлів дерева має дві характерні особливості.

Перша полягає в в тім, що видаляючи вузол ми маємо видалити і усіх його нащадків.

Друга особлівість – необхідно звільняти пам’ять, яка відводилася для зберігання вузлів.

Наведені особливості вимагають, щоб видалення починалось з нащадків останнього рівня дерева, так званих листів дерева.

Зважаючи на це ми почнемо розглядати функції видалення зі структури останнього рівня.

Для кожного рівня спочатку треба звільнити пам’ять яку використовували поля структури, а потім вде вивільняти пам’ять, що займала сама структура.

5.3.1 Видалення інформації про викладачів

Так само, як і у випадку виведення інформації, завдання будемо вирішувати за допомогою двох функцій. Перша з них буде безпосередньо вирішувати завдання видалення з урахуванням особливостей структури, а друга забезпечить виклик цієї функції через меню та виведення додаткової інформації.

5.3.1.1 Функція видалення викладача

Інформація про викладачів кафедри зберігається в односпрямованому списку. Вказівник на початок списку зберігається у полі firstTeacher структури Dept, тому алгоритм видалення має дві гілки.

Перша гілка активізується, якщо видаляється структура, яка є першою у списку. У цьому випадку вказівник на першого викладача у списку міняємо на вказівник, який вказував на наступного.

Друга гілка працює для інших структур. Тут в циклі ідемо по списку і знаходимо елемент, після якого йде елемент, який потрібно видалити. Після цього змінюємо у цього елемента вказівник на наступного беручи значення вказівника на наступного у елемента, який видаляємо.

Після маніпуляцій зі списком звільняємо пам’ять, яку використовував елемент, який видалено із списку.

Текст функції наведено в лістингу 43.

Лістинг 43 – Функція видалення структури Teacher із списку

```
int delTeacher(Teacher *delTch){
    Dept *dept = delTch->parent;
    if(dept->firstTeacher == delTch){
        dept->firstTeacher = dept->firstTeacher->nextTeacher;
    }
    else{
        Teacher *current = dept->firstTeacher;
        while(current != NULL && current->nextTeacher != delTch){
            current = current->nextTeacher;
        }
        if(current == NULL)
            return 1; //Не знайдено
        //Видаляємо із списку
        current->nextTeacher = current->nextTeacher->nextTeacher;
    }
    //Звільняємо пам'ять
    delete[] delTch->name;
    delete[] delTch->post;
    delete delTch;
    return 0; //Видалено
}
```

5.3.1.2 Функція видалення викладача через меню

У функції, лістинг 44, перед видаленням виводиться інформація про викладача і запит на підтвердження видалення.

Після видалення виводиться список викладачів, що залишились.

Лістинг 44 – Текст функції видалення викладача черех меню

```
int delTeacherByMenu(void *ptr){
    Teacher *delTch = (Teacher*)ptr;
    Dept * parent = delTch->parent;
    system("cls");
    cout << "Інформація про викладача, що видаляється:\n";
    showTeacherInfo(delTch);
    cout << " Якщо Ви дійсно хочете видалити цю структуру, натисніть 0,\n";
    cout << " інакше будь-який інший символ.\n ";
    if(getch() != '0'){
        cout << "Ви відмовилися від видалення.";
        pause();
        return 0; //Залишимося в меню викладача
    }
    //Видалення
    fprintf_s(logFile,"\nВидалення викладача %s", delTch->name);
    delTeacher(delTch);
    fprintf_s(logFile,"\nВикладача видалено");
    cout << " Викладача видалено." << endl;
    cout << " Список викладачів, що залишились:";
    showTeacherList(parent);
    pause();
    return 1; //Повернення в попереднє меню
}
```

5.3.1.3 Функція видалення всіх викладачів кафедри

У цій функції, лістинг 45, організовано цикл послідовного доступу до елементів списку і на кожному кроці цикла видаляється одна структура за допомогою функції видалення викладача, яка була написана раніше.

В якості параметра функція приймає вказівник на кафедру.

Лістинг 45 – Функція видалення всіх викладачів кафедри

```
int delAllTeacher(Dept *dept){
    Teacher *current = dept->firstTeacher;
    int count =0;
    while(current != NULL){
        Teacher *delTch = current;
        current = current->nextTeacher;
        delTeacher(delTch);
        count++;
    }
    return count;
}
```

5.3.1.4 Видалення всіх викладачів кафедри через меню

Ця функція стосується кафедри, хоча відбувається видалення викладачів. Тому виклик її треба робити через меню кафедри.

У функції, лістинг 46, перед видаленням виводиться список викладачів і запит на підтвердження видалення. Для видалення викликається функція, яка була розгляута в попередньому підпункті.

Лістинг 46 – Функція видалення всіх викладачів кафедри через меню

```
int delAllTeacherByMenu(void *ptr){
    Dept *dept = (Dept*)ptr;
    cout << " Викладачі кафедри " << dept->name;
    showTeacherList(dept);
    cout << " \n Якщо Ви дійсно хочете видалити всіх, натисніть 0,\n";
    cout << " інакше будь-який інший символ.\n ";
    if(getch() != '0'){
        cout << "Ви відмовилися від видалення.";
        pause();
        return 0; //Залишимося в меню кафедри
    }
    fprintf_s(logFile,"\nВидалення всіх викладачів кафедри %s",dept->name);
    delAllTeacher(dept);
    fprintf_s(logFile,"\nВсіх викладачів кафедри видалено");
    cout << " Викладачів видалено.\n";
    pause();
    return 0; //Залишимося в меню кафедри
}
```

5.3.2 Видалення інформації про кафедри

Реалізовувати це завдання будемо так само, як і видалення викладачів. Тобто для видалення кафедри або всіх кафедр будемо використовувати по дві функції. Перша – для безпосереднього видалення, друга – для меню.

5.3.2.1 Функція видалення кафедри

В якості параметра до цієї функції, лістинг 47, передається вказівник на кафедру, яку потрібно видалити.

Функція у циклі переглядає масив і шукає вказівник на потрібну кафедру. Для знайденого елемента звільняється пам’ять, яка виділялась для назви кафедри, і видаляються всі викладачі кафедри. Після чого виконується зсув ліворуч усіх елементів масиву, що були розташовані після знайденого, і лічильник кафедр зменшується на 1.

Лістинг 47 – Текст функції видалення кафедри

```
int delDept(Dept *dpt){
    Univer *parent = dpt->parent;
    Dept ** dptAr =  parent->deptArray;
    int n = parent->deptCount;
    //Пошук елемента, що теба видалити
    for(int i = 0; i <n; i++){
        if(dptAr[i] == dpt){
            //Звільнення пам'яті
            delete[] dptAr[i]->name;
            delAllTeacher(dptAr[i]);
            //Видалення шляхом зсуву частини масиву ліворуч
            for(int j = i; j < n-1; j++){
                dptAr[j] = dptAr[j+1];
            }
            //Оновлення лічильника кафедр
            parent->deptCount = n - 1;
            break;
        }
    }
    return 1; //Повернення в попереднє меню
}
```

5.3.2.2 Видалення кафедри через меню

Ця функція, лістинг 48, дуже схожа на функцію видалення викладача через меню.

Лістинг 48 – Функція видалення кафедри через меню

```
int delDeptByMenu(void *ptr){
    Dept *delDp = (Dept*)ptr;
    system("cls");
    cout << "Інформація про кафедру, що видаляється:\n";
    showDeptInfo(delDp);
    cout << " Якщо Ви дійсно хочете видалити цю структуру, натисніть 0,\n";
    cout << " інакше будь-який інший символ.\n ";
    if(getch() != '0'){
        cout << "Ви відмовилися від видалення.";
        pause();
        return 0; //Залишимося в меню викладача
    }
    //Видалення
    fprintf_s(logFile,"\nВидалення кафедри %s", delDp->name);
    delDept(delDp);
```

```
    fprintf_s(logFile,"\nКафедру видалено");
    cout << " Кафедру видалено." << endl;
    cout << " Список кафедр, що залишились:";
    showDeptList(root);
    pause();
    return 1; //Повернення в попереднє меню
}
```

5.3.2.3 Функція видалення всіх кафедр

У функції, лістинг 49, в циклі послідовно видаляються елементи масиву кафедр, після чого лічильник кафедр обнуляється.

Лістинг 49 – Текст функції видалення всіх кафедр

```
int delAllDept(Univer *pU){
    int count =0;
    int n = root->deptCount;
    Dept ** dptAr =  root->deptArray;
    //Цикл по кафедрам для видалення
    for(int i = 0; i <n; i++){
        delDept(dptAr[i]);
    }
    //Оновлення лічильника кафедр
    root->deptCount = 0;
    return count;
}
```

5.3.2.4 Видалення всіх кафедр через меню

Хоч функція стосується кафедр, але її слід викликати через меню університету. Вона дуже схожа на функцію видалення всіх викладачів через функцію меню, яка розглядалась раніше, тому наводити її тут не будемо.

5.3.3 Видалення інформації про університет

У цьому пункті потрібно реалізувати тільки видалення університету. Це завдання, так само як і раніше, будемо вирішувати за допомогою двох функцій.

5.3.3.1 Функція видалення університету

У цій функції, лістинг 50, треба звільнити пам’ять, яку займає назва університету, потім видалити всі кафедри. Далі звільнити пам’ять, що займав масив кафедр, і після цього присвоїти вказівнику на корінь дерева значення NULL.

Лістинг 50 – Текст функції видалення університету

```
void delUniver(){
    delete[] root->name;
    delAllDept(root);
    delete[] root->deptArray;
    delete root;
    root = NULL;
}
```

5.3.3.2 Видалення університета через меню

Текст функції наведено в лістингу 51

Лістинг 51 – Текст функції видалення університету через меню

```
int delUniverByMenu(void *ptr){
    system("cls");
    cout << "\n Інформація про університет:\n";
    showUniverInfo(root);
    cout << "\n Якщо Ви дійсно хочете видалити дерево,"
            " натисніть 0,\n інакше будь-який інший символ.\n ";
    if(getch() != '0'){
        cout << "Ви відмовилися від видалення.";
        pause();
        return 0; //Залишимося в поточному меню
    }
    fprintf_s(logFile,"\nВидалення дерева");
    delUniver();
    fprintf_s(logFile,"\nДерево видалено");
    cout << " Дерево видалено.\n";
    pause();
    return 1;
}
```

5.4 Редагування інформації про вузли

Редагування підлягає тільки та інформація, яка вводиться користувачем під час створення вузла.

Алгоритм редагування залежить від рівня структури. Найпростіше редагувати корінь дерева, тому що корінь не є елементом масиву або списку і тому нема проблеми впорядкування.

Тому спочатку розглянемо функцію редагування кореневої структури.

5.4.1 Редагування інформації у кореневій структурі

Для редагування кореневої структури будемо використовувати наступний алгоритм:

– виводимо наявні дані про структуру;
– запитуємо користувача, чи потрібно редагування;
– вводимо нові дані про структуру;
– оновлюємо поля існуючої структури;
– виводимо на перегляд інформацію про відкореговану структуру.

Код функції, що реалізує цей алгоритм наведено в лістингу 52.

Лістинг 52 – Код функції редагування інформації про університет

```
int editUniverByMenu(void *ptr){
    system("cls");
    cout << "\n __Редагування інформації про університет __\n";
    cout << "\n Дані про структуру, які можна змінити:\n";
    showUniverInfo(root);
    cout << "\n Якщо Ви будете редагувати ці дані натисніть 0,\n";
    cout << " інакше будь-який інший символ.\n ";
    if(getch() != '0'){
        cout << "\n Ви відмовилися від редагування.";
        pause();
        return 0; //Залишимося в меню університету
    }
    fprintf_s(logFile,"\nРедагування університету %s",root->name);
    cout << "\n Введіть нові дані\n";
    char* name = getStr(" Назва університету: ");
    int level = getInt(" Рівень акредитації: ", 1, 4);
    delete[] root->name;
    root->name = name;
    root->level = level;
    cout << "\n Дані структури після редагування:\n";
    fprintf_s(logFile, "\nНова назва %s, новий рівень %d",
              root->name, root->level);
    showUniverInfo(root);
    pause();
    return 0; //Залишимося в меню університету
}
```

5.4.2 Редагування інформації у структурах наступних рівнів

Зміна інформації у структурах не кореневого рівня може порушити впорядкованість елементів масиву або списка. Для того, щоб відновити порядок, можна відсортувати змінену групу елементів.

Можна також на основі нових даних створити нову структуру, видалити стару структуру і додати нову. У цьому випадку функція сортування не потрібна, а функції створення, видалення та додавання структур ми вже маємо.

Доаткова перевага такого способу ще і в тім, що він не залежить від організації групи даних.

Тож реалізуємо цей спосіб.

У лістингу 53, в якості зразку, наведено код функції для редагування інформації про кафедру.

Лістинг 53 – Текст функції редагування інформації про кафедру

```
int editDeptByMenu(void *ptr){
    Dept *oldDpt = (Dept*)ptr;
    Univer *parent = oldDpt->parent;
    system("cls");
    cout << "\n __Редагування інформації про кафедру __\n";
    cout << "\n Дані про структуру, які можна змінити:\n";
    showDeptInfo(oldDpt);
    cout << "\n Якщо Ви будете редагувати ці дані натисніть 0,\n";
    cout << " інакше будь-який інший символ.\n ";
    if(getch() != '0'){
        cout << "\n Ви відмовилися від редагування.";
        pause();
        return 0; //Залишимося в меню кафедри
    }
    fprintf_s(logFile, "\nРедагування кафедри %s",oldDpt->name);
    cout << "\n Введіть нові дані\n";
    char* name = getStr(" Назва кафедри: ");
    int hours = getInt(" Навантаження, годин: ", 1000, 40000);
    int spec = getInt(" Кафедра випускаюча? 1-так, 0-ні ",0,1);
    Dept *newDpt = createDept(name,hours, spec);
    delDept(oldDpt);
    addDept(parent, newDpt);
    fprintf_s(logFile,"/nНова назва %s, навантаження %d, %s",newDpt->name,
              newDpt->hours, newDpt->spec ? "випускаюча": "загальна");
    cout << "\n Перелік кафедр після редагування";
    showDeptList(root);
    pause();
    return 1; //Ідемо в попереднє меню, бо вказівник на кафедру змінився
}
```

Після реалізації цієї функції доцільно її протестувати. Після цього можна створити функцію редагування інформації про викладача. Код функції редагування інформації про викладача за структурою такий самий.

6 РОБОТА З ДЕРЕВОМ (СТВОРЕННЯ, ЗБЕРЕЖЕННЯ, ВІДНОВЛЕННЯ, ЗАПИТИ)

На цьому етапі до проєкту додамо можливість створення дерева через меню, що стане в нагоді після видалення дерева. Це буде або знов дефолтне дерево, або новий корінь, до якого можна буде додати будь які вузли.

Реалізуємо також можливість зберегти дерево у файлі і відновити його з файлу.

Крім того, на цьому етапі ми реалізуємо запити до дерева, які будуть повертати деяку узагальнену інформацію, що стосується дерева в цілому.

Визначені завдання мають багато спільного. Головне полягає в тім, що для вирішення більшості з цих завдань доводиться працювати з усіма, або частиною вузлів на різних рівнях дерева. Тобто доводиться «обходити» дерево так само, як це ми робили для демонстрації дерева на екрані.

Тож будемо вирішувати наступні задачі:

– розширимо перелік функцій меню для операцій з деревом і реалізуємо підменю запитів до дерева;

– реалізуємо функції створення кореня дерева та виклику функції створення дефолтного дерева через меню;

– реалізуємо запит на отримання інформації про дерево через меню;

– реалізуємо функції для збереження дерева у файлі;

– реалізуємо функції для відновлення дерева з файлу.

– реалізуємо запити на отримання узагальненої інформації про дерево.

Реалізація останнього пункта залежить від переліку завдань, що були визначені у завданні на проєктування. У нашому прикладі будуть реалізовані такі запити:

– пошук наймолодшого викладача;

– підрахунок сумарного навчального навантаження по університету;

– виведення переліку випускаючих кафедр.

6.1 Доопрацювання меню

Реалізуймо функцію menuQuery, яка буде виводити на консоль підменю, через яке можна буде активізувати запити до дерева. В якості зразка можна взяти функції створення інших підменю. А функції запитів мають відповідати варіанту студента.

До вже існуючої функції menuTree додаймо ще елементи, для створення кореня та дефолтного дерева, виведення інформації про проєкт і виклику функцій збереження та відновлення дерева з файлу. Після цього функція має вглядати так, як показано в лістингу 54.

Зауважимо, що меню має працювати незалежно від того чи існує дерево, чи ні.

Доцільно разом з цією функцією створити і заготовки функцій, що викликаються через це меню.

Лістинг 54 – Фунція створення меню для роботи з деревом

```
int menuTreeOperation(void* ptr){
     fprintf_s(logFile,"\nВиклик меню операцій з деревом");
     MenuUnit menu[] = {
        {"Create root", createRootByMenu},
        {"Create default tree", createDefaultTreeByMenu},
        {"Show project information", showProjectInfo},
        {"Show tree", showTree},
        {"Show log file", showLogFile},
        {"Store to file", storeTreeByMenu},
        {"Restore from file", restoreTreeByMenu},
        {"Return to main menu",NULL}};
    runMenu(menu, "\n    --- Tree operation ---",ptr);
    return 0;
}
```

6.2 Реалізація функцій створення дерева

Після того, як ми створили функцію видалення дерева, потрібно надати можливість відновлення його. Реалізуймо це завдання. Іідповідні функції розташуємо у файлі create.cpp/

6.2.1 Функція меню для створення дефолтного дерева.

Функція створення дефолтного дерева вже є. Реалізуймо її виклик через меню, лістинг 55.

Лістинг 55 – Функція меню для створення дефолтного дерева

```
int createDefaultTreeByMenu(void*){
    system("cls");
    cout << " --- Default tree creating ---\n";
    if(root != NULL){
        cout << " Tree already exist!\n";
        cout << "\n Якщо Ви дійсно хочете видалити існуюче дерево,"
                " натисніть 0,\n інакше будь-який інший символ.\n ";
        if(getch() == '0')
            delUniver();
        else{
            cout << "Ви відмовилися від створення дерева.";
            pause();
            return 0; //Залишимося в поточному меню
        }
     }
    root = createDefaultTree();
    showTree(root);
    return 0; //Залишаємось в меню дерева
}
```

6.2.2 Функція меню для створення кореня нового дерева

Код функції наведено в лістингу 56.

Лістинг 56 – Функція меню для створення кореня дерева.

```
int createRootByMenu(void*){
    system("cls");
    cout << " --- New tree creating ---\n";
    if(root != NULL){
        cout << " Tree already exist!\n";
        cout << "\n Якщо Ви дійсно хочете видалити існуюче дерево,"
                " натисніть 0,\n інакше будь-який інший символ.\n ";
        if(getch() == '0')
            delUniver();
        else{
            cout << "Ви відмовилися від створення нового дерева.";
            pause();
            return 0; //Залишимося в поточному меню
        }
     }
    char* name = getStr(" Назва університету: ");
    int level = getInt(" Рівень акредитації: ", 1, 4);
    root = createUniver(name, level);
    showTree(root);
    return 0;
}
```

Створені функції треба протестувати. Спочатку слід видалити дерево, потім створити нове, використовуючі обидва варіанти.

6.3 Збереження дерева у файлі та відновлення його з файлу

Дерево, яке ми створили, зберігається тільки в пам’яті комп’ютера. Якщо ж ми хочемо зберегти дерево, з яким працювали, або хочемо перенести його на інший комп’ютер, треба зберегти його у файлі і надати можливість відновити його з файлу. Дерево – це не текст, тому доцільно його зберігати в бінарному файлі.

Усі функції, що пов’язані з реалізацією цього завдання розташуємо у файлі tree.cpp.

6.3.1 Запис інформації про дерево у файл

Для вирішення цього завдання напишемо декілька функцій.

Перша буде безпосередньо вирішувати задачу запису даних про дерево у файл. В якості параметра функція буде приймати вказівник на структуру FILE. В реалізації цієї функції будемо використовувати допоміжну функцію запису текстових полів до файлу, що скоротить код основної функціїі зменшіть імовірність виникнення помилок.

Ще одна функція забезпечить діалог з користувачем, через яку користувач повідомить програму, де він хоче зберігати дерево.

6.3.1.1 Допоміжна функція для запису текстових полів у файл

В нашому проекті використовуються структури, які містять вказівники на текстові поля. Самі тексти зберігаються в інших ділянках динамічної пам'яті. Тому до файлу, крім структури, треба записати і текст.

Текст у файл будемо зписувати у два кроки. Спочатку записуємо число, значення якого відповідає кількості символів у тексті, включаючи і ознаку кінця рядка. Після цього записуємо сам текст.

Код відповідної функції наведено в лістингу 57.

Лістинг 57 – Функція зпису текста у файл

```
void storeText(char* text, FILE* file){
    int len = strlen(text) + 1;
    fwrite(&len, sizeof(int), 1, file);
    fwrite(text, sizeof(char),len,file);
}
```

6.3.1.2 Функція запису даних про дерево у файл

Інформацію, що пов’язана з деревом, будемо записувати до файлу у такій самій послідовності як і під час виведення його на екран.

Спочатку записуємо інформацію про корінь.

Далі організуємо цикл по масиву вказівників на кафедри і записуємо у файл інформацію про поточну кафедру.

Для поточної кафедри організуємо цикл по викладачам і послідовно записуємо у файл інформацію про всіх викладачів кафедри.

Далі повертаємось до циклу роботи з кафедрами і записуємо інформацію про наступну кафедру.

Зберігання у файлі інформації про вузли дерева, які є структурами, має свої особливості. Перш за все у файлі зберігаємо саму структуру. Але, якщо структура має поля з вказівниками, то у файлі потрібно буде зберегти і дані, на які посилаються ці вказівники. Зокрема, для запису даних, на які посилаються вказівники типу char*, будемо використовувати допоміжну функцію storeText.

Код функції для збереження інформації про дерево, що розглядається в якості приклада, наведено в лістингу 58.

Лістинг 58 – Функція збереження дерева у файлі

```
void storeTree(FILE* file){
    //Store root structure
    fwrite(root,sizeof(Univer),1,file);
    //Store university's name
    storeText(root->name, file);
    //loop for departments storing
    Dept ** deptArr = root->deptArray;
    for(int i = 0; i < root->deptCount; i++){
        //Store current department structure
        fwrite(deptArr[i],sizeof(Dept),1,file);
        //Store department's name
        storeText(deptArr[i]->name, file);

        //loop for teachers storing
        Teacher *teacher = deptArr[i]->firstTeacher;
        while(teacher != NULL){
            //Store current teacher structure
            fwrite(teacher,sizeof(Teacher), 1, file);
            //Store teacher's name and post
            storeText(teacher->name, file);
            storeText(teacher->post, file);
            //Move on to the next teacher
            teacher = teacher->nextTeacher;
        }
    }
}
```

6.3.1.3 Функція запису дерева до файлу через меню

Ця функція реалізує діалог з користувачем, через який користувач повідомить програму, де він хоче зберігати дерево. Текст функції наведено у лістингу 59.

Лістинг 59 – Функція запису дерева до файлу через меню

```
int storeTreeByMenu(void*){
    system("cls");
    if(!rootExist()) return 0;
    cout << "\t--- Saving the tree to a file ---\n";
    char* fileName = getStr("Enter file name: ");
    FILE *f = fopen(fileName, "wb");
    if(f == NULL)
        cout << " Error opening file " << fileName << endl;
```

```
    else{
        storeTree(f);
        cout << " The tree was saved in file " << fileName << endl;
        fprintf_s(logFile,"/nДерево збережено у файлі %s ", fileName);
        fclose(f);
    }
    pause();
    return 0;
}
```

6.3.2 Реалізація відновлення дерева з файлу

Відновлення дерева з файлу відбувається в такій самій послідовності як і збереження. Головна особливість цього процесу полягає в необхідності виділяти пам’ять для записів, що відновлюються.

Для спрощення основної функції відновлення ми напишемо ще дві допоміжних функції.

Крім того знадобиться функція меню для відновлення дерева з файлу.

6.3.2.1 Допоміжна функція для считування даних типу char*

Тексти будемо відновлювати відповідно до того, як ми записували їх до файлу, у два кроки. Спочатку читаємо число, значення якого відповідає кількості символів у тексті, включаючи і ознаку кінця рядка. Після цього зчитуємо відповідну кількість символів.

Код відповідної функції наведено в лістингу 60.

Лістинг 60 – Функція зчитування з файлу даних типу char*

```
char* restoreText(FILE* file){
    int len;
    fread(&len, sizeof(int), 1, file);
    char* text = new char[len];
    fread(text, sizeof(char), len, file);
    return text;
}
```

6.3.2.2 Допоміжна функція зчитування даних про викладача

Операція, яку виконує ця функція, буде потрібна в основній програмі у двох місцях, тому доцільно виділити відповідний фрагмент коду в окрему функцію. Це скоротить код основної програми і покращить її читабельність.

Текст функції наведено в лістингу 61

Лістинг 61 – Функція відновлення даних про викладача з файлу

```
Teacher *restoreTeacher(FILE* file, Dept* parent){
    Teacher *teacher = new Teacher();
    teacher->parent = parent;
    fread(teacher,sizeof(Teacher),1,file);
    teacher->name = restoreText(file);
    teacher->post = restoreText(file);
    return teacher;
}
```

6.3.2.3 Функція відновлення дерева з файлу

Як вже наголошувалося, відновлення дерева з файлу відбувається в такій самій послідовності як і збереження. Головна особливість цього процесу полягає в необхідності виділяти пам’ять для записів, що відновлюються.

Текст функції наведено в лістигу 62.

Лістинг 62 – Функція відновлення дерева з файлу

```
void restoreTree(FILE* file){
    //Restore root structure
    root = new Univer();
    fread(root,sizeof(Univer),1,file);
    //restore university's name
    root->name = restoreText(file);
    //Create array for departments
    Dept ** deptArr = new Dept*[root->arrSize];
    root->deptArray  = deptArr;
    //loop for departments restoring
    for(int i = 0; i < root->deptCount; i++){
        //Restore current department structure
        deptArr[i] = new Dept();
        fread(deptArr[i],sizeof(Dept),1,file);
        //Restore department's name
        deptArr[i]->name = restoreText(file);
        //Teachers restoring
        if(deptArr[i]->firstTeacher != NULL){
            //First teacher restoring
            deptArr[i]->firstTeacher = restoreTeacher(file, deptArr[i]);
            //loop for other teachers restoring
            Teacher *teacher = deptArr[i]->firstTeacher;
```

```
            while(teacher->nextTeacher != NULL){
                //Restore next teacher
                teacher->nextTeacher = restoreTeacher(file, deptArr[i]);
                //Move on to the next teacher
                teacher = teacher->nextTeacher;
            }
        }
    }
}
```

6.3.2.4 Функція меню для відновлення дерева з файлу

Ця функція реалізує діалог з користувачем, через який користувач може ввести ім'я файлу, де зберігається дерево.

Текст функції наведено в лістингу 63.

Лістинг 63 – Функція меню для відновлення дерева з файлу

```
int restoreTreeByMenu(void*){
    system("cls");
    cout << "\t--- Restoring the tree from a file ---\n";
    char* fileName = getStr("Enter file name: ");
    FILE *f = fopen(fileName, "rb");
    if(f == NULL){
        cout << " Error opening file " << fileName << endl;
        pause();
        return 1;
    }
    restoreTree(f);
    fclose(f);
    cout << " The tree restored from file " << fileName << endl;
    //Send message to log file
    fprintf_s(logFile,"/nДерево відновлено з файлу %s ", fileName);
    pause();
    showTree(NULL);
    return 0;
}
```

6.4 Реалізація запитів до дерева

Задачі реалізації запитів до дерева залежать від теми проєкту. Тому наводити багато прикладів у цьому підрозділі ми не будемо.

Але студенти мають реалізувати, як мінімум, три запити. У цих запитах можна, наприклад, знаходити вузол з якимось умовно найкращим показником, обчислювати загальну суму або кількість якогось показника, знаходити якийсь

вузол за назвою, але в цьому випадку користувачеві достатньо ввести тільки початкові літери назви.

Як вже наголошувалося, для реалізації запитів доводиться «обходити» дерево. Приклади обходу вже розглядалися в багатьох завданнях, що вирішувалися раніше.

В якості приклада розглянемо реалізацію запиту на пошук наймолодшого викладача.

6.4.1 Пошук наймолодшого викладача

Для вирішення цього завдання напишемо дві функції.

Перша буде безпосередньо вирішувати задачу пошуку наймолодшого викладача і повертати казівник на структуру Teacher.

Друга сформує текст відповіді на запит.

6.4.1.1 Функція пошуку наймолодшого викладача

У функції два цикли. Зовнішній забезпечує перегляд кафедр, а внутрішній – перегляд викладачів. Параметром для порівняння є рік народження викладача. В якості результату запам’ятовується посилання на структуру Teacher.

Код функції наведено в лістингу 64.

Лістинг 64 – Функція пошуку наймолодшого викладача

```
Teacher* findYoungest(){
    Teacher *young = NULL;
    Dept ** deptArr = root->deptArray;
    for(int i = 0; i < root->deptCount; i++){
        Teacher *teacher = deptArr[i]->firstTeacher;
        while(teacher != NULL){
            if(young == NULL || teacher->birthYear > young->birthYear)
                young = teacher;
            teacher = teacher->nextTeacher;
        }
    }
     return young;
}
```

6.4.1.2 Функція меню для пошуку наймолодшого викладача

У цій функції спочатку отримуємо посилання на наймолодшого викладача, обчислюємо його вік і виводимо інформацію на консоль.

Для того, щоб користувач побачив результат, виклиуаємо функцію pause().

Код функції наведено в лістингу 65.

Лістинг 65 – Реалізація пошуку наймолодшого викладача через меню

```
int findYoungestByMenu(void*){
    Teacher *young = findYoungest();
    if(young == NULL) {
        cout << "\n Не знайдено.";
        pause();
        return 1;
    }
    SYSTEMTIME st;
    GetSystemTime(&st);
    int age = st.wYear - young->birthYear;
    system("cls");
    cout << "The youngest teacher is:\n";
    showTeacherInfo(young);
    cout << "Teacher's age is "<< age << " years\n";
    pause();
    return 0;
}
```

Реалізація функцій запитів до дерева завершує роботу над проєктом у семестрі. Але під час захисту студенту доведеться розширити можливості створеного проєкту.

Розширення може полягати в реалізації додаткових запитів, модифікації значень деяких полів структур. Можливо також завдання на внесення змін до існуючих структур або навіть додання ще одного рівня ієрархії у дереві.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Шпак З.Я. Програмування мовою С / З.Я. Шпак. – Львів : вид-во НУ «Львівська політехніка», 2011. – 436 с.
2. Kernighan B. W., Ritchie D. M. C Programming Language / http://www.amazon.com/C-Programming-Language-2nd-Edition/dp/0131103628Dennis M. Ritchie, Brian W. Kernighan,http://www.amazon.com/C-Programming-Language-2nd-Edition/dp/0131103628 – 2nd ed. – Prentce-hall, inc., 1988. – 263 p.
3. Kochan S.G. Programming in C/– 3nd ed. – Sams Publishing, 2004. – 505 p.
4. C Programming Tutorial. – Режим доступу: https://www.guru99.com/c-programming-tutorial.html
5. Дисципліна «Курсовий проект з основ програмування». – Режим доступу https://eln.stu.cn.ua/course/view.php?id=4815