МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чернігівський національний технологічний університет

РЕАЛІЗАЦІЯ СТРУКТУР ДАНИХ МОВОЮ ПРОГРАМУВАННЯ С

МЕТОДИЧНІ ВКАЗІВКИ
до виконання курсового проекту з дисципліни
«Основи програмування»
для студентів спеціальностей
121 – "Інженерія програмного забезпечення"
123 – "Комп'ютерна інженерія"

ЗАТВЕРДЖЕНО
на засіданні кафедри
інформаційних та комп'ютерних систем
протокол № 6 від 30.01.20

Чернігів ЧНТУ 2020

Реалізація структур даних мовою програмування С. Методичні вказівки до виконання курсового проекту з дисципліни «Основи програмування» для студентів спеціальностей 121 – “Інженерія програмного забезпечення”, 123 – “Комп’ютерна інженерія”. /Укл.: Бивойно П.Г. – Чернігів: ЧНТУ, 2020. – 51с.

Укладач: Бивойно Павло Георгійович, канд. техн. наук, доцент;

Відповідальний за випуск: В.М. Базилевич, зав. кафедрою інформаційних та комп’ютерних систем, канд. економ. наук, доцент.

Рецензент: Білоус І.В., канд. техн. наук, доцент кафедри інформаційних технологій та програмної інженерії Чернігівського національного технологічного університету

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 деякі проблеми цієї області. Ніяких розрахунків у проекті не передбачається, але має буди програмна реалізація додатку.

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.

Результатом роботи є працездатний програмний консольний додаток та звіт обсягом приблизно 30 сторінок друкованого тексту оформленого відповідно до стандартів кафедри. Бали за роботу виставляються з урахуванням своєчасності та якості виконання.

За звичай номер варіанту завдання для проекту призначається викладачем, та студент може і сам обрати предметну область для реалізації, але завдання обов'язково має бути погоджено з викладачем

ЗАВДАННЯ ДО КУРСОВОГО ПРОЕКТУ

Варіант			
1.Залізниця	місто	вокзал	рейс
2.Автобусні перевезення	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	город
8. Екологія	область	район	екопроблеми
9. Місто	вулиця	будинок	квартира
10. Комунальні служби	жек	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. Фармацевтична фірма	країна	відділ	ліки
16.Торгова 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	сума, що, де
25. Географія	континенти	регіони	держави
26. Гуртожиток	блок	кімната	мешканець
27. Мережева торгівля	продавець	товар	реалізація
28. Поліклініка	лікар	пацієнт	рецепт
29. Музика	жанр	група	пісня
30. «Орел чи решка»	континент	країна	учасники

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.

Кількість рівнів нащадків кореня дерева має бути не менше трьох. Корінь та компоненти кожного рівня мають бути структурами, що містять не менше двох 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 структур.

Доступ до нащадків другого рівня має бути реалізовано через масив вказівників на відповідні структури.

Структури третього рівня мають зберігатися у лінійному списку.

Нащадки кожного з елементів мають бути упорядковані за простими та складними правилами.

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

Додаток має забезпечити реалізацію таких функцій:

- відобразити дані вузла;
- змінити дані існуючого компоненту;
- додати елемент до дерева;
- відобразити перелік нащадків компоненту;
- вилучити елемент із дерева;
- відобразити дерево на консолі цілком (у текстовому режимі);
- зберегти структуру даних у файлі;
- відновити структуру даних із файлу.
- відшукати компонент за якоюсь ознакою на кожному рівні;
- обчислити якусь числову характеристику групи елементів;
- відобразити на консолі файл протоколу .

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

Предметна область проекту має відповідати одному з варіантів із наведеного нижче переліку. Номер варіанту студент отримує у керівника проекту або самостійно вибирає з переліку, наведеного на сайті дистанційного навчання. Студент може запропонувати керівникові проекту і свій варіант.

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 проектну документацію, оформлену відповідно до стандартів кафедри.

ОБОВ’ЯЗКОВІ РОЗДІЛИ ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Пояснювальна записка до роботи створюється відповідно до вимог оформлення випускних кваліфікаційних робіт. Наведені нижче пункти можна вважати доповненням до цих вимог.

1.1 Індивідуальне завдання

У завданні має бути наведено опис системи, для якої створюється модель. Зокрема, мають бути перелічені поля структур на усіх рівнях. Також мають бути сформульовані завдання для пошуку у дереві та завдання для визначення числових характеристик.

1.2 Аналіз та розробка системи що підлягає реалізації

У цьому розділі слід проаналізувати предметну область та навести її схематичне зображення у довільному вигляді і короткі пояснення до схеми.

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

1.3 Розробка системи

У цьому розділі 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.

1.4 Реалізація системи

У цьому розділі слід навести реалізації функцій систем. Для кожної функції має бути коментар про її призначення та особливості застосування.

Для функцій реалізації пунктів меню слід навести також результати їх виконання у вигляді копій екрану.

ГАФІК ВИКОНАННЯ РОБОТИ

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

Проект оцінюється за 100 бальною шкалою. За роботу у семестрі студент може отримати до 60 балів. Бали виставляються за кожний етап в залежності від якості виконання.

Етап, за який виставлено оцінку, вважається завершеним.

За етап, результати виконання якого були представлені невчасно, оцінка не може бути вищою за 8 балів. Якщо недоліки, за які оцінку було знижено на попередньому етапі, не були виправлені, то отримати максимальну оцінку за наступний етап студент не може. Мінімальна оцінка за етап не може бути меншою за 6 балів.

Захисти роботи проводяться за графіком на двох останніх тижнях семестру. Під час захисту студент має виконати практичне завдання з доопрацювання свого проекту та відповісти на питання, що стосуються програмування і структур даних.

За результатами захисту роботи студент може отримати до 40 балів.

Таблиця 1 – Графік роботи над проектом

Етапи роботи	Кінцевий термін (тиждень семестру)	Оцінка в балах за етап
Представити для затвердження індивідуальне завдання	1-й	
Доопрацювати індивідуальне завдання. Структури даних та прототипи функцій у файлі проекту	2-й	до 10
Реалізація функцій створення і додавання вузлів у та відображення дерева	4-й	до 10
Реалізація операцій редагування та вилучення вузлів дерева у програмному режимі.	6-й	до 10
Реалізація функцій збереження дерева у файлі, та відновлення його з файлу.	8-й	до 10
Представити остаточну версію проекту на перевірку	10-й	до 10
Представити текстову частину на перевірку	12-й	до 10

Результати виконання етапів студент викладає у вигляді архіву папки з qt-проектом (папка …-build-desktop не потрібна). Текстова частина надсилається у вигляді .docx файлу.

ПРИКЛАД ПОБУДОВИ СИСТЕМИ

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на виконання курсового проекту
з дисципліни "Основи програмування"
студента Іванчука М. П., група КІ-203

Тема роботи: Інформаційна система "Міністерство освіти".

Очікувані технічні та експлуатаційні результати роботи:

Консольний Qt проект, який забезпечує роботу з інформацією про університет (факультети, кафедри, викладачі) згідно варіанта завдання 0.

Корінь – «міністерство освіти» містить назву міністерства та його бюджет.

Рівень 1 - «університет», має містити назву університету та рівень акредитації. Університети впорядковуються за рівнем акредитації та назвою.

Рівень 2 - «факультет», зберігає інформацію про назву факультету та ім’я декана. Факультети впорядковуються відповідно до назви.

Рівень 3 - «кафедра», містить назву кафедри, ознаку - чи є кафедра випускаючою, та кількість викладачів на кафедрі.

Окрім типових операцій введення даних, їх корегування, вилучення та перегляду, мають бути реалізовані такі специфічні операції:

- пошук кафедри з найбільшою кількістю викладачів;
- пошук факультету за назвою і виведення відповідної інформації;
- виведення на консоль кількості випускаючих кафедр для кожного університету.

Планова трудомісткість роботи: 1 кредит **(**30 годин).

Обсяг текстової документації:

Пояснювальна записка до проекту об'ємом ~30 сторінок друкованого тексту формату А4.

Проект оформляється згідно вимогам до оформлення випускних робіт.

Плановий термін захисту проекту: кінець травня 2020 року, відповідно до графіку засідань комісії.

Виконавець роботи: _____ Іванчук М.П.

Керівник роботи: _____ Бивойно П.Г.

Дата видачі завдання: " 10 " лютого 2020р.

Дата узгодження завдання: " 24 " лютого 2020р.

АНАЛІЗ ЗАДАЧІ
СТВОРЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Аналіз опису інформаційної системи, що наведено в індивідуальному завданні свідчіть про те, систему можна представити у вигляді сильно розгалуженого дерева. Представлення такої інформаційної системи у вигляді графу наведено на рисунку 1.1.

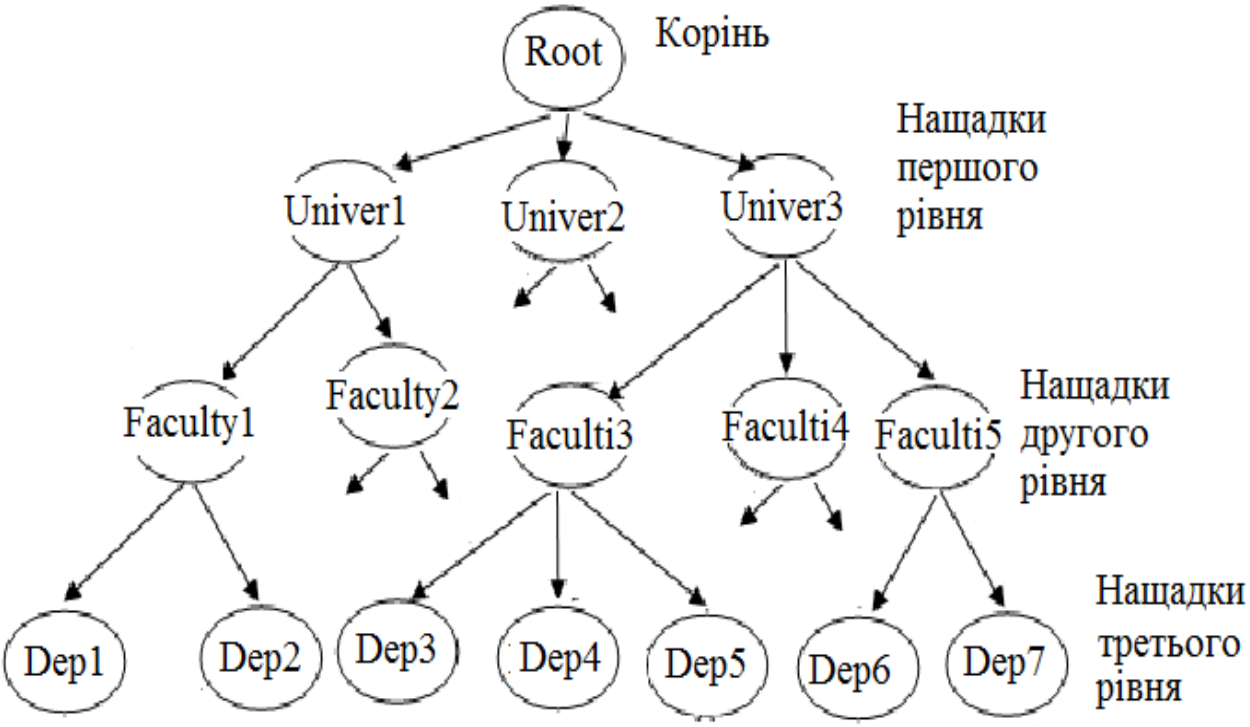

Рисунок 1.1 – Логічна структура інформаційної системи

Програмна реалізація графу, який наведено на рисунку 1.1 ускладняється тим, що кожний вузол може мати будь яку кількість нащадків. Існують різні підходи до програмної реалізації таких логічних структур різною, але у будь якому випадку структура має бути динамічною, бо наперед невідомо, скільки нащадків буде у кожного вузла і їх кількість може змінюватися у часі..

Далі розглядаються можливі реалізації такої структури та оцінюються їх переваги та недоліки .

1.1 Аналіз варіантів реалізації сильно розгалуженого дерева

У цьому підрозділі слід докладно розглянути типові структури даних та оцінити їх переваги та недоліки.

Слід враховувати, що робота обов’язково буде перевірятися на наявність плагіату. У разі виявлення плагіату робота буде відправлятися на переробку.

1.1.1 Розгалужене дерево на базі масиву структур

1.1.2 Розгалужене дерево на базі масиву вказівників

1.1.3 Розгалужене дерево на базі однонаправлених списків

1.1.4 Розгалужене дерево на базі двунаправлених списків

1.1.5 Розгалужене дерево із збереженням дітей вузла у бінарному дереві

1.1.6 Розгалужене дерево із збереженням дітей вузла у хеш таблиці

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

1.2 Структура інформаційної системи "Міністерство освіти"

Відповідно до вимог, нащадки кореня (перший рівень) мають зберігатися у масиві структур. Доступ до елементів другого рівня має бути реалізовано через масив вказівників на відповідні структури. Елементи третього рівня мають зберігатися у лінійному списку.

Фізичну структуру інформаційної система «Міністерство освіти», що відповідає наведеним вимогам, схематично показана на рисунку 2.1.

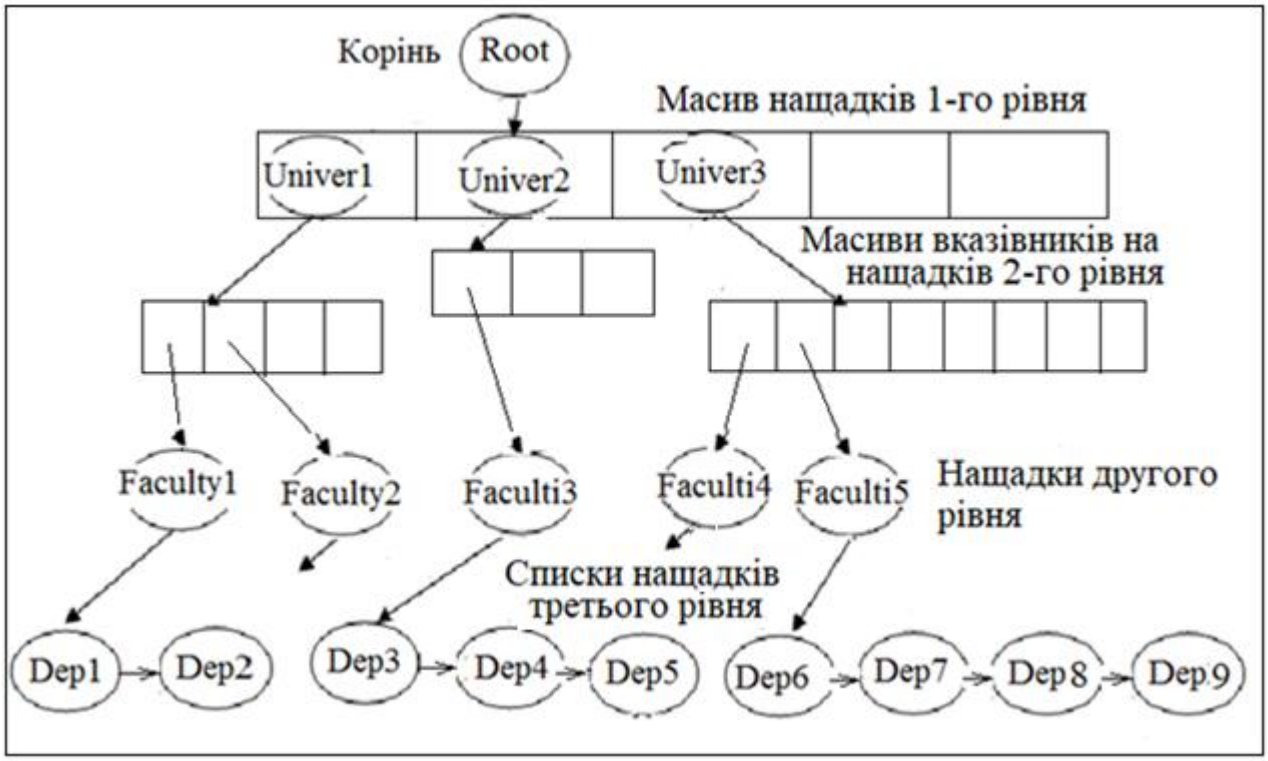

Рисунок 2.1 – Фізична структура інформаційної системи

Як видно з рисунку, корінь має посилання на масив структур, що описують університети. Кожен університет має посилання на масив вказівників, елементи якого містять посилання на структури з інформацією

про факультет. Кожен факультет має посилання на початок списку кафедр. А кожна кафедра має посилання на свого сусіда у списку.

Таким чином, кожен вузол має містити деяку кількість інформаційних полів, а також мати службові поля, які реалізують зв'язки між вузлами дерева.

Кожний вузол є структурою. Вузли одного рівня мають однаковий тип, але типи вузлів різних рівнів відрізняються. Розглянемо докладно ці структури.

1.2.1 Структура «корінь»

Для шаблону структури «корінь» приймемо назву «Ministry». Перелік полів та їх характеристики наведемо у таблиці 2.1.

Таблиця 2.1 – Перелік полів структури Ministry

Поле структури	Ідентифікатор	Тип	Обмеження
Назва міністерства	name	char []	40 символів
Бюджет, млн. грн.	budget	int	до 1 000 000
Масив університетів	childArray	Univer []	
Кількість університетів	childCount	int	до 100
Розмір масиву для університетів	arSize	int	

Особливість цієї структури полягає в тому, що назва університету зберігається як масив символів у самій структурі. Так само і університети зберігаються в масиві, як структури. Це є вимогою технічного завдання.

1.2.2 Структура «університет»

Для шаблону структури «університет» приймемо назву «Univer». Перелік полів та їх характеристики наведемо у таблиці 2.2.

Таблиця 2.2 – Перелік полів структури Univer

Поле структури	Ідентифікатор	Тип	Обмеження
Назва університету	name	char []	40 символів
Рівень акредитації.	level	int	до 5
Масив вказівників на факультети	pFacultyArray	Faculty* *	
Кількість факультетів	facultyCount;	int	до 10
Розмір масиву вказівників	arSize	int	
Вказівник на батьківський вузол	parent	Ministry*	

У структурі університет, відповідно до технічного завдання, масив факультетів містить вказівники на структури «факультет». Але назва університету зберігається у структурі як рядок символів.

У наступних структурах назва буде зберігатися у структурі у вигляді вказівника.

1.2.3 Структура «факультет»

Для шаблону структури «факультет» приймемо назву «Faculty». Перелік полів та їх характеристики наведемо у таблиці 2.3.

Як бачимо, у структурі Faculty усі поля є вказівниками.

Таблиця 2.3 – Перелік полів структури Faculty

Поле структури	Ідентифікатор	Тип	Обмеження
Назва факультету	name	char *	40 символів
Ім’я декану	dean	char *	40 символів
Вказівник на початок списку кафедр	firstDept	Dept *	
Вказівник на батьківський вузол	parent	Univer*	

1.2.4 Структура «кафедра»

Для шаблону структури «кафедра» приймемо назву «Dept». Перелік полів та їх характеристики наведемо у таблиці 2.4.

Таблиця 2.4 – Перелік полів структури Dept

Поле структури	Ідентифікатор	Тип	Обмеження
Назва кафедри	name	char *	40 символів
Кількість викладачів	size	int	від 5 до 50
Випускаюча, чи ні.	spec	int	1 або 0
Вказівник на наступну кафедру	nextDept	Dept*	
Вказівник на батьківський вузол	parent	Faculty*	

1.3 Розробка функціоналу системи

Функціональні можливості системи можна розділити на дві групи.

Перша група – це функції, що використовуються для маніпуляцій з деревом та його вузлами у програмному режимі.

Друга група функцій забезпечує роботу з деревом через консольне меню. Функції цієї групи реалізуються на основі функцій першої групи. Тому, перш за все слід визначитися з функціями першої групи.

1.3.1 Функції для маніпуляцій з деревом у програмному режимі

Розробимо сигнатури цих функцій, об’єднавши їх за призначенням.

1.3.1.1 Функції для створення вузлів дерева

```
Ministry *createMinistry(char* name, int budget);
Univer *createUniver(char* name, int level);
Faculty *createFaculty(char *name, char *dean);
Dept *createDept(char *name, int size, int spec);
```

1.3.1.2 Функції для додавання вузлів дерева

```
void addUniver(Ministry *ministry, Univer* univer);
void addFaculty(Univer *pU, Faculty *pF);
void addDept(Faculty *pF, Dept *pD);
```

1.3.1.3 Функції для редагування інформації у вузлах дерева

Ці функції дозволяють змінювати значення інформаційних полів структур.

```
void editMinistry(Ministry *pM, char *newName, int budget );
void editUniver(Univer *pU, char *newName, int level);
void editFaculty(Faculty *pF, char *newName, char *newDean);
void editDept(Dept *pD, char *newName, int size, int spec);
```

1.3.1.4 Функції для видалення вузлів дерева

Це функції, які дозволяють вилучати окремі структури із переліку нащадків вузла, Інформація про нащадка, якого треба вилучити, передається у вигляді посилання на нього.

```
Univer* delUniver(Univer *pU);
Faculty* delFac(Faculty *pF);
Dept* delDept(Dept*);
```

Передбачимо також видалення усього переліку нащадків разом. В якості параметра передаємо посилання на батьківський вузол.

```
void delAllUniver(Ministry *pM);
void delAllFac(Univer *pU);
void delAllDept(Faculty *pF);
```

1.3.1.5 Функції для операцій з деревом в цілому

Перш за все, це функція створення тестового дерева. Наявність тестового дерева суттєво пришвидшує тестування інших функцій і, відповідно, розробку проекту.

```
int createDefaultTree(void*);
```

Окрім того потрібні функції для відображення дерева на екрані, видалення дерева, функції збереження дерева у файлі та відновлення його з файлу

```
int showTree(void*);
```

int delTree(void*);
void storeToFile(FILE *f);
void restoreFromFile(FILE *f);

Перші три функції можна буде використовувати як у програмному режимі, так і через меню, тому їх сигнатури записані у відповідності до вимог типу функцій меню, хоча потреби передавати туди вказівник нема.

1.3.1.6 Функції реалізації запитів до дерева

Перелік цих функцій визначається вимогами технічного завдання.
Dept* findBiggestDept();
Faculty* findFacultyName(char*);
int listSpecDept();

1.3.1.7 Допоміжні функції

Визначимо функцію, яка буде утримувати консоль від закриття. Вона буде використовуватися функцією showTree().
void pause();

1.4 Розробка консольного інтерфейсу користувача

Консольний інтерфейс користувача має дві складові.

Перша складова формує перелік дій, які можливі на даному етапі спілкування з додатком, виводить цей перелік на екран і очікує на вибір користувача. У найпростішому варіанті користувач відповідає на запит введенням з клавіатури номеру вибраного варіанту. Саме так буде і у нашому додатку. Таку складову інтерфейсу найчастіше називають консольним меню.

Друга складова інтерфейсу користувача забезпечує реалізацію вибраної функції меню та повернення результату.

1.4.1 Розробка переліку опцій консольного меню

Наше додаток має реалізувати такі функції :

- створити корінь;
- створити дерево за замовчуванням;
- додати нащадка до вибраного вузла;
- показати перелік нащадків вибраного вузла;
- показати інформаційне наповнення вузла;
- редагувати інформаційне наповнення вузла;
- вилучити вузол;
- видалити усіх нащадків вузла;
- показати усе дерево;
- зберегти дерево у файлі;
- завантажити дерево із файлу;
- видалити дерево;

– показати на консолі файл протоколу;
– виконати запит пошук найбільшої кафедри;
– виконати пошук факультету за назвою;
– вивести на консоль кількість випускаючих кафедр для кожного університету.

Бажано, щоб інтерфейс користувача був швидким та зручним.

Теоретично можливі різні схеми меню і кожен розробник може використовувати таку, яку вважає найзручнішою. Ми ж пропонуємо наступну схему.

1.4.2 Розробка складових частин консольного меню

Перш за все розділимо наведений перелік функцій на групи. Підставою для цього є настанови з розробки інтерфейсів користувача, які рекомендують формувати кількість опцій меню приблизно 5, але не більше 7.

Для поділу на групи скористаймося тим фактом, що частина з перелічених функцій працює з усім деревом, починаючи з кореня, або тільки з коренем. Сформуємо з цих функцій три групи:

– функції для роботи з кореневим вузлом;
– функції обробки дерева в цілому;
– функції реалізації запитів.

Інші функції потребують визначення конкретного вузла. Особливість роботи з деревом у консольному режимі полягає в тому, що для роботи з конкретним вузлом доводиться визначатися з усіма його предками. Тобто кількість послідовних уточнень залежить від рівня. Враховуючи цю обставину сформуємо ще три групи функцій:

– функції опрацювання вузлів рівня «університет»;
– функції опрацювання вузлів рівня «факультет»;
– функції опрацювання вузлів рівня «кафедра».

Враховуючи вищенаведене, сформуємо меню першого рівня так, як показано на рисунку 2.2.

```
   --- Main menu ---
1. Ministry operation
2. University operation
3. Faculty operation
4. Department operation
5. Tree operation
6. Query operation
7. Exit
Enter your choice number:
```

Рисунок 2.2 – Вигляд меню першого рівня

Пункти 1, 5, 6 меню не потребують уточнень і одразу викликають меню другого рівня, вигляд яких представлено на рисунках 2.3, 2.4, 2.5.

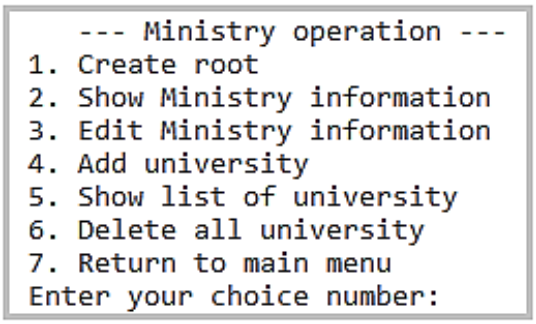

Рисунок 2.3 – Перелік функцій меню «Ministry operation»

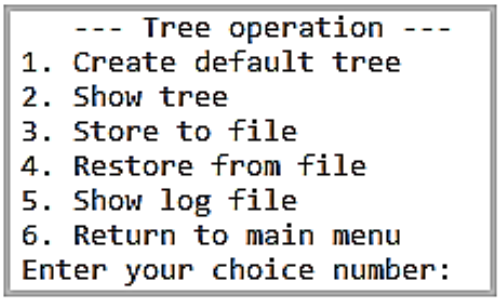

Рисунок 2.4 – Перелік функцій меню «Tree operation»

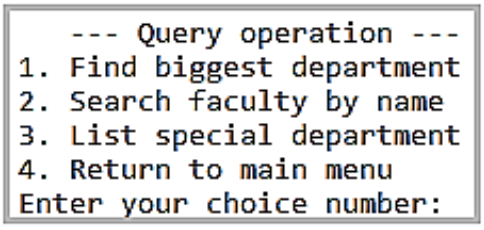

Рисунок 2.5 – Перелік функцій меню «Query operation»

Кожен пункт наведених меню другого рівня пов’язаний з виконанням конкретної операції.

Натомість пункти 2, 3, 4 головного меню потребують подальших уточнень.

Для пункту 2 потрібно тільки одне уточнення – вибір університету, рисунок 2.6

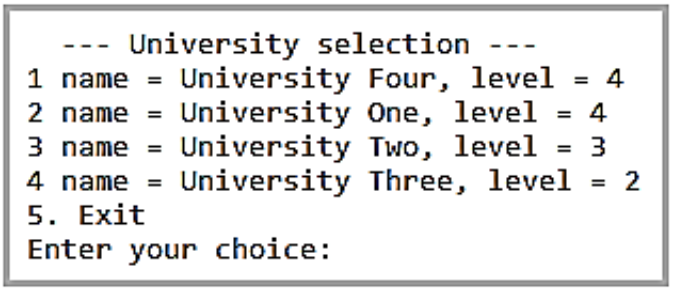

Рисунок 2.6 – Вигляд меню вибору університету

Пункт 3 головного меню потребує не тільки вибору університету, але й вибору факультету, рисунок 2.7.

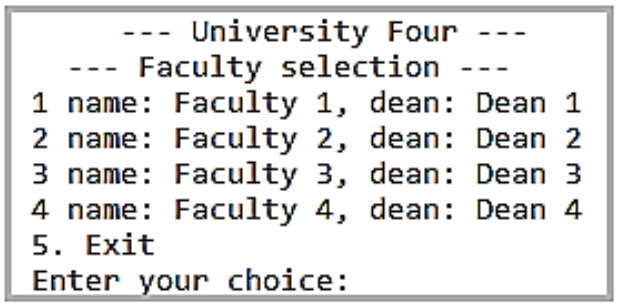

Рисунок 2.7 – Вигляд меню вибору факультету

А пункт 4 потребує ще й вибору кафедри, рисунок 2.8.

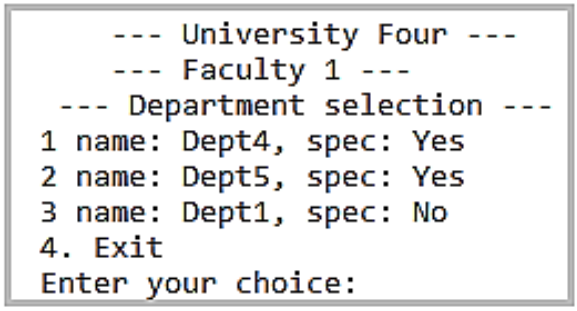

Рисунок 2.8 – Вигляд меню вибору кафедри

Особливість меню наведених на рисунках 2.6, 2.7, 2.8 полягає в тому, що вони не пов’язані з якимись діями, а просто повертають посилання на вибраний вузол. Це посилання має бути передано до меню вибору операції над вузлом. Саме тому функції обробки вузлів мають бутив визначені, як функції, що приймають вказівник, як параметр. А оскільки вказівники можуть бути різних типів (університет, факультет, кафедра), то цей вказівник у сигнатурі функції має бути типу void*.

Далі розглянемо меню обробки вибраного вузла.

Вигляд цих меню для обробки вузлів «університет», «факультет», «кафедра» наведено на рисунках 2.9, 2.10, 2.11.

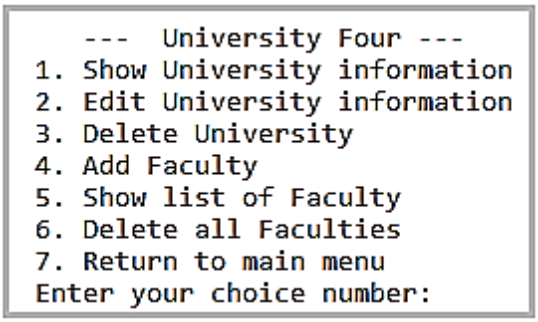

Рисунок 2.9 – Вигляд меню обробки вузла «університет»

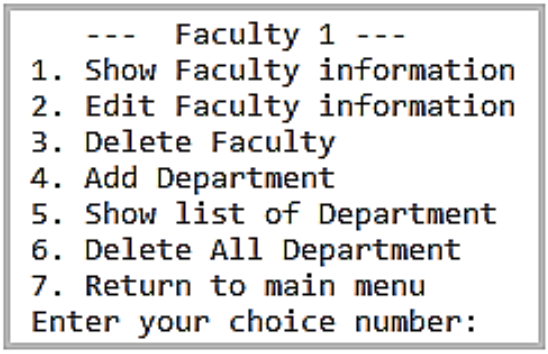

Рисунок 2.10 – Вигляд меню обробки вузла «факультет»

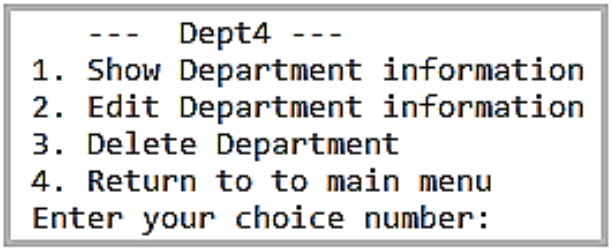

Рисунок 2.11 – Вигляд меню обробки вузла «кафедра»

Наведені меню обробки вузлів також мають приймати вказівник, як параметр.

1.4.3 Визначення типу для функцій обробки вузлів та дерева

Для усіх функцій обробки вузлів та дерева в цілому, які викликаються через пункти доцільно визначити єдиний тип.

Приймемо, що усі ці функції мають приймати в якості параметра вказівник типу void*, а повертати значення типу int. Значення, що повертається, будемо розглядати як ознаку коректності виконання функції, або якусь іншу ознаку. Вказівник, що передається як параметр, буде надавати доступ до вузла, що опрацьовується.

Приймемо назву MenuFunc для цього типу функції.

1.4.4 Функції створення меню

Для формування меню, які зображено на рисунках 2.2 - 2.11, необхідні відповідні функції.

Розробимо сигнатури цих функцій, об’єднавши їх за призначенням.

1.4.4.1 Функції головного меню

```
int mainMenu(void* );
int menuTree(void*);
int menuQueries(void*);
int menuMinistry(void* );
int menuQuery(void*);
```

int menuUniver(void*);
int menuFaculty(void*);
int menuDept(void*);

1.4.4.2 Функції вибору вузла із списку нащадків

Univer* selectUniver();
Faculty* selectFaculty(Univer*);
Dept* selectDept(Faculty *);

1.4.4.3 Розробка шаблону для структури «меню»

Працювати з меню буде зручно, якщо об'єднати назву пункту меню та посилання на відповідну функцію у структурі. Це дозволить для кожного меню створити масив таких структур і опрацьовувати його.

Для шаблону структури «меню» приймемо назву «MenuUnit». Перелік полів та їх характеристики наведемо у таблиці 2.5.

Таблиця 2.5 – Перелік полів структури MenuUnit

Поле структури	Ідентифікатор	Тип	Обмеження
Назва пункту меню	text	string	
Вказівник на функцію реалізації пункту меню	func	MenuFunc*	

1.4.4.4 Розробка алгоритму обробки масиву структур MenuUnit

Відповідна функція буде розглянута у розділі «Реалізація». Тут студенти мають навести схему алгоритму роботи цієї функції, та навести опис її роботи.

1.4.5 Функції для маніпуляцій з деревом через меню

Ці функції реалізують другу складову інтерфейсу користувача, яка полягає у реалізації замовленої операції, підготовці та виведенні на екран інформації, яку замовив користувач, або у повідомленні про успішне виконання операції.

Розробимо сигнатури цих функцій, об'єднавши їх за призначенням.

1.4.5.1 Функції для додавання вузлів дерева

Ці функції мають організувати діалог з користувачем для отримання даних про вузол, що створюється, після цього будуть використовувати вже розглянуті функції додавання у програмному режимі.

int addUniverByMenu(void*);
int addFacultyByMenu(void*);
int addDeptByMenu(void*);

1.4.5.2 Функції для відображення інформації про вузли дерева

Тут є дві групи функцій. Перша група забезпечує виведення значень інформаційних полів структури.

```
int showMinistryInfo(void*);
int showUniverInfo(void*);
int showFacultyInfo(void*);
int showDeptInfo(void*);
```

Друга група виводить перелік нащадків вузла.

```
int showUniverList(void*);
int showFacultyList(void*);
int showDeptList(void*);
```

1.4.5.3 Функції для редагування інформації вузлів дерева

Ці функції мають організувати діалог з користувачем у якому виводять інформацію про існуючі значення параметрів вузла і надають можливість ввести нові значення.

```
int editMinistryByMenu(void*);
int editUniverByMenu(void*);
int editFacultyByMenu(void*);
int editDeptByMenu(void*);
```

Після введення нових даних ці функції будуть використовувати вже розглянуті функції додавання у програмному режимі.

1.4.5.4 Функції для видалення вузлів

Розробимо два різновиди цих функцій. Перша група функцій буде використовуватися для видалення окремого вузла.

```
int delUniverByMenu(void*);
int delFacultyByMenu(void*);
int delDeptByMenu(void*);
```

Друга група бде забезпечувати видалення усіх нащадків.

```
int delAllUniverByMenu(void* ptr);
int delAllFacultyByMenu(void* ptr);
int delAllDeptByMenu(void* ptr);
```

Функції обоїх груп мають забезпечувати каскадне видалення вузлів і звільняти пам’ять, яку використовували видалені елементи.

1.4.5.5 Функції для операцій з деревом

Деякі з цих функцій вже було розглянуто як функції, що використовуються в програмному режимі. Це наступні функції.

```
int showTree(void*);
int delTree(void*);
```

Окрім них розробимо ще функції для маніпуляцій з файлами.

```
int storeTreeByMenu(void* );
int restoreTreeByMenu(void*);
int showLogFile(void*);
```

1.5 Розробка схеми збереження дерева у файлі

1.5.1 Структура файлу з даними про дерево

Інформацію про дерево будемо записувати у файл блоками таким чином, щоб її було зручно використовувати під час відновлення дерева з файлу.

Схему розташування інформації у файлі зображено на рисунку 2.2. У схемі, для скорочення не показано поле parent.

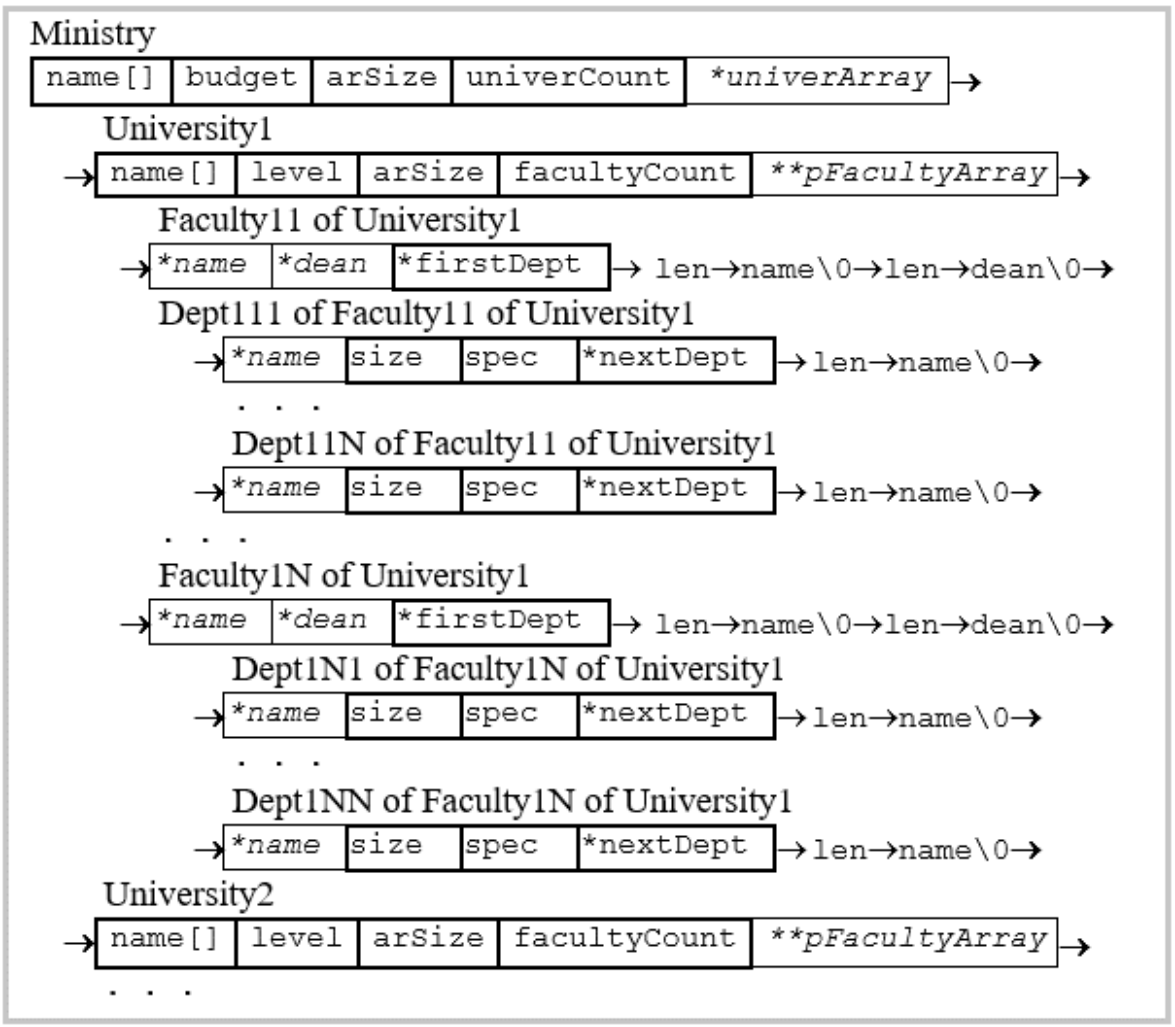

Рисунок 2.2 – Схема збереження дерева у файлі

Структури на схемі показані у вигляді фрагментів таблиці. Товсті поля комірок означають, що інформація з цих комірок буде використовуватися без змін під час відновлення дерева з файлу. Тонкі поля означають, що ця інформація не потрібна під час відновлення. За звичай у цих полях знаходяться вказівники.

Дані, на які посилаються ці вказівники, розташовуються у файлі блоками, що йдуть за структурою. Це стосується елементів масивів та імен, якщо у структурі зберігається не масив символів, а вказівник на це ім’я. В останньому випадку після структури записується довжина рядка символів, а після цього сам рядок символів.

Єдиний вказівник, що зберігається у файлі і використовується для відновлення дерева – це вказівних *nextDep. Але і тут нас цікавить лише, чи дорівнює він NULL, чи ні.

Першим блоком у файлі іде коренева структура – один блок. Далі йдуть одиночні блоки структур «університет».

Після кожного «університету» ідуть одиночні блоки структур «факультет». Оскільки назва факультету та ім’я декана у структурі зберігаються через вказівники, то значення цих полів записуються у файлі після структури у вигляді пар – довжина рядка і сам рядок.

А після кожного блоку «факультет» послідовно йдуть структури усіх кафедр факультету. Але після кожної структури записується пара – довжина назви кафедри і сама назва.

Запропонована структура зберігання не ідеальна, бо у файлі зберігається зайва інформація (частини структур, що виділено тонкими лініями і курсивом. Але це надає змогу зчитувати структури цілком, без фрагментації. Це спрощує функцію відновлення дерева з файлу.

1.5.2 Схема алгоритму функції збереження дерева у файлі

1.5.3 Схема алгоритму відновлення дерева з файлу

1.6 Формування протоколу роботи з деревом

Протокол буде створюватися у вигляді текстового файлу, у якому будуть фіксуватися усі маніпуляції користувача з деревом через меню.

Файл буде створюватися разом із запуском додатку у поточній директорії із стандартним ім’ям log.txt.

Першим повідомленням у файлі буде інформація про назву проекту та його розробника.

Кожне нове повідомлення у файлі починається з нового рядка і містить інформацію про дату і час операції.

У повідомленнях про створення, видалення, або про зміну стану вузлів наводиться повна інформацію про вузол.

У повідомленнях про збереження дерева у файлі або його відновлення з файлу вказується ім’я файлу.

Ім’я файлу будемо зберігати у глобальній змінній.

Для формування протоколу буде створиено дві допоміжні функції - для відкриття файлу протоколу і для додавання повідомлення у протокол.

```
void openLogFile(string message);
void printToLog(string message);
```

1.7 Розробка файлової структури проекту

Оголошення шаблонів структур розташуємо у заголовному файлі. type.h.

Оскільки кількість шаблонів невелика, то у цьому файлі напишемо і прототипи функцій.

Файл main.cpp також не будемо перевантажувати. До нього включимо лише код, що пов’язаний із запуском додатку. Усі функції, що були перелічені у попередньому пункті розташуємо у декількох файлах .cpp. Розміщення функцій у різних файлах полегшить пошук потрібних функцій під час роботи з проектом. У цьому прикладі ми скористуємося функціональною ознакою для створення файлів. Але це не обов’язково, можливі також інші рішення..

Перелік файлів проекту наведено у таблиці 2.6.

Таблиця 2.6 – Перелік файлів проекту

Призначення	Назва
Шаблони структур, константи та прототипи функцій	type.h
Функції реалізації меню	menu.cpp
Функції для створення вузлів	create.cpp
Функції додавання вузлів та пов’язані з ними	add.cpp
Функції редагування вузлів та пов’язані з ними	edit.cpp
Функції виведення інформації про вузли	show.spp
Функції видалення вузлів	del.cpp
Функції опрацювання дерева в цілому	tree.cpp
Функції реалізації запитів	query.cpp
Допоміжні функції	util.cpp

Докладно про програми, які складаються з багатьох файлів, можна почитати у підрозділі 12.3 підручника, сторінка 242.

РЕАЛІЗАЦІЯ ПРОЕКТУ

2.1 Реалізація файлової структури проекту

Створімо консольний проект. У прикладі це проект ср2020. Одночасно з проектом створюється файл main.cpp.

Далі через функцію контекстного меню проекту Add New… викликаємо діалоги створення нових файлів C++Header File та C++Source File і створюємо файл type.h та усі файли .cpp, відповідно до таблиці 2.6.

Після цього файлова структура нашого проекту має виглядати так, як показано на рисунку 3.1.

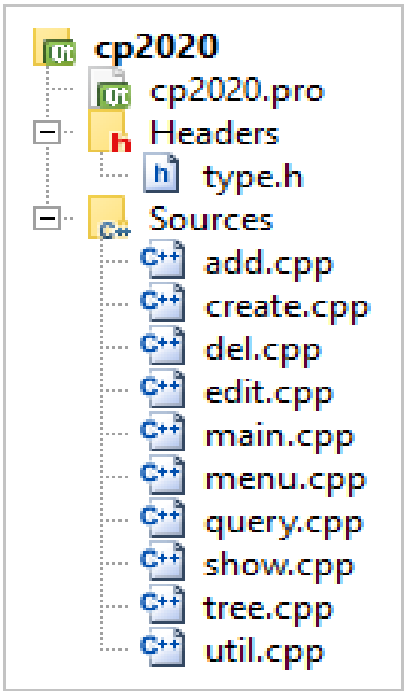

Рисунок 3.1 – Файлова структура проекту

2.2 Оголошення шаблонів структур, констант і прототипів функцій

2.2.1 Визначення шаблонів структур

Шаблони структур оголошуємо у файлі type.h. Початковий вигляд файлу наведено на рисунку 3.2.

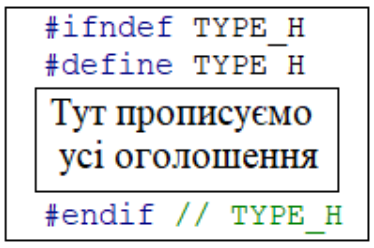

Рисунок 3.2 – Початковий вигляд файлу type.h

Після створення 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.

Шаблони структур оголосимо відповідно до таблиць 2.1, .. 2.4. Виходячи з того, що структури мають перехресні посилання, зробимо їх попереднє оголошення, лістинг 3.1.

Зверніть увагу! До цього файлу ми також будемо додавати усі потрібні директиви **#include** та **using namespace.** Це позбавить нас необхідності підключати ці директиви у кожному з файлів .cpp. Тепер достатньо буде підключати тільки директиву **#include “type.h”.**

Лістинг 3.1 – Оголошення шаблонів структур

```
#include <iostream>
#include <windows.h>
#include <conio.h>
#include <stdio.h>
using namespace std;
#define SIZE_NAME 40
struct Dept;
struct Faculty;
struct Univer;
struct Ministry{
    char  name[SIZE_NAME];
    int budget;
    int arSize;
    int univerCount;
    Univer *univerArray;
};
struct Univer{
    char name[SIZE_NAME];
    int level;
    int arSize;
    int facultyCount;
    Faculty* *pFacultyArray;
    Ministry *parent;
};
struct Faculty{
    char* name;
    char* dean;
    Dept *firstDept;
    Univer *parent;
};
struct Dept{
    char* name;
    int size;
    int spec;
    Dept *nextDept;
    Faculty *parent;
};
```

2.2.2 Оголошення типів, що пов’язані з меню, та глобальних змінних

Оголосимо далі у файлі type.h тип для функцій обробки пунктів меню та структуру MenuUnit.

Оголосимо також дві зовнішні глобальні змінні root – корінь дерева, та logFileName – ім’я файлу протоколу. Ці змінні будуть пізніше прописані у

файлі main. Включення цих оголошень у файл type.h дозволить звертатися до відповідних змінних у будь якому файлі проекту, до якого додано директиву #include “type.h”.

Лістинг 3.2 – Оголошення типів для меню та зовнішніх глобальних зиінних

```
// Тип функцій для пунктів меню
typedef int MenuFunc(void*);
//Структура елементу меню
struct MenuUnit{
    string text;
    MenuFunc *func;
};
// Зовнішні глобальні змінні
extern Ministry *root;
extern char* logFileName;
```

2.2.3 Оголошення прототипів функцій

Прототипи функцій також будемо прописувати у файлі type.h. Фактично вони вже написані у розділі «Розробка». Тому у лістингу 3.3 ми покажемо тільки фрагменти їх оголошення. Головна мета цього лістингу показати, що прототипи ми згрупували у відповідності з файлами .cpp, де будуть знаходитися їх реалізації.

Лістинг 3.3 – Фрагмент голошення прототипів функцій у файлі type.h

```
//add.cpp
void addUniver(Ministry *ministry, Univer* univer);
void addFaculty(Univer *pU, Faculty *pF);
void addDept(Faculty *pF, Dept *pD);

void growUniverAr(Ministry *pM);
void growFacultyrAr(Univer *pU);

int cmpUniver(Univer *u1, Univer *u2);
int cmpFaculty(Faculty *f1, Faculty *f2);
int cmpDept(Dept *d1, Dept *d2);

int addUniverByMenu(void*);
int addFacultyByMenu(void*);
int addDeptByMenu(void*);
```

2.3 Реалізація функцій для маніпуляцій з деревом у програмному режимі

2.3.1 Реалізація функцій створення вузлів дерева

Ці функції розташуємо у файлі create.cpp. На початку файлу напишемо директиву #include “type.h”. Далі можна писати реалізації функцій.

Функції створення вузлів мають схожу структуру.

Кожна з них повертає посилання на вузол, який вона створює.

В якості параметрів ці функції приймають значення інформаційних полів, або посилання на них.

Алгоритм роботи функції полягає у виконанні таких етапів:

– виділити пам’ять для структури і отримати посилання на неї;

– сформувати інформаційні поля структури відповідно до переданих параметрів;

– проініціалізувати службові поля структури;

– повернути посилання на сформовану структуру.

Під час написання коду функцій слід взяти до уваги наступне. Якщо інформаційні поля структури оголошені як масиви символів, а параметр передається як посилання на рядок, то необхідно використовувати функцію копіювання рядків.

Розглянемо ці функції.

2.3.1.1 Функція створення «міністерства»

Код функції наведено у лістингу 3.4.

Лістинг 3.4 – Код функції створення вузла «міністерство»

```
Ministry *createMinistry(char* name, int budget){
    Ministry *ministry = new Ministry();
    strncpy(ministry->name, name, SIZE_NAME) ;
    ministry->budget = budget;
    ministry->arSize = 2;
    ministry->univerArray = new Univer[ministry->arSize];
    ministry->univerCount = 0;
    return ministry;
}
```

Як бачимо, ім’я міністерства копіюється у саму структуру, для зберігання нащадків типу Univer створюється масив univerArray.

2.3.1.2 Функція створення «університету»

Код функції наведено у лістингу 3.5.

Лістинг 3.5 – Код функції створення вузла «університет»

```
Univer *createUniver(char *name, int level){
    Univer *univer = new Univer();
    strncpy(univer->name, name, SIZE_NAME);
```

```
        univer->level = level;
        univer->arSize = 2;
        univer->pFacultyArray = new Faculty*[univer->arSize];
        univer->facultyCount = 0;
        return univer;
```

Тут так само, як і у попередній функції, ім’я університету копіюється у саму структуру, але масив pFacultyArray створюється для зберігання вказівників на структури Faculty.

2.3.1.3 Функція створення «факультету»

Код функції наведено у лістингу 3.6.

Лістинг 3.6 – Код функції створення вузла «факультет»

```
Faculty *createFaculty(char *name, char *dean){
    Faculty *faculty = new Faculty();
    char* facultyName = new char[strlen(name)+1];
    strncpy(facultyName, name, strlen(name)+1);
    faculty->name = facultyName;
    char *deanName = new char[strlen(dean)+1];
    strncpy(deanName, dean, strlen(dean)+1);
    faculty->dean = deanName;
    faculty->firstDept = NULL;
    return faculty;
}
```

Ця структура зберігає тільки посилання на ім’я факультету та декана. Самі імена знаходяться в пам’яті поза межами структури і займають саме стільки пам’яті, скільки їм потрібно.

Так само ім’я кафедри зберігається і у наступній структурі.

2.3.1.4 Функція створення «кафедри»

Код функції наведено у лістингу 3.7.

Лістинг 3.7 – Код функції створення вузла «кафедра»

```
Dept *createDept(char *name, int size, int spec){
    Dept *dept = new Dept();
    char* deptName = new char[SIZE_NAME];
    strncpy(deptName, name, SIZE_NAME);
    dept->name =deptName;
    dept->size = size;
    dept->spec = spec;
    dept->nextDept = NULL;
    return dept;
}
```

2.3.2 Реалізація допоміжних функцій, що пов'язані з додавання вузлів дерева

Для успішної реалізації операцій додавання необхідно розробити декілька допоміжних функцій.

Для розширення масивів, в яких зберігаються нащадки міністерства і університетів, будуть використовуватися наступні функції.

void growUniverAr(Ministry *pM);
void growFacultyrAr(Univer *pU);

Зважаючи на вимогу завдання розташовувати нащадків у певному порядку, створюємо функції порівняння вузлів дерева, які теж будуть використовуватися у процесі додавання вузлів.

int cmpUniver(Univer *u1, Univer *u2);
int cmpFaculty(Faculty *f1, Faculty *f2);
int cmpDept(Dept *d1, Dept *d2);

Ці функції розташуємо у файлі add.cpp. На початку файлу напишемо директиву #include "type.h". Далі можна писати реалізації функцій.

2.3.2.1 Реалізація функцій збільшення масивів

Структура і алгоритми цих функцій дуже схожі.

В якості параметру вони приймають вказівник на структуру, у якої існує потреба у збільшенні місця для масиву.

Сам алгоритм полягає у виконанні наступних дій:

- створити новий масив більшого розміру;
- скопіювати елементи старого масиву у новий;
- замінити посилання на масив у структурі на нове;
- звільнити пам'ять, яку займав старий масив.

Різниця між функціями growUniverAr та growFacultyAr полягає тільки у типах параметрів функції та типі елементів масиву. Тому наводимо у якості прикладу одну з них, лістинг 3.8.

Лістинг 3.8 – Код функції для збільшення розміру масиву «університетів»

```
void growUniverAr(Ministry *pM){
    int newSize = 1.5 * pM->arSize + 0.5;
    Univer* newAr = new Univer[newSize];
    // Copy old array to new
    memcpy(newAr, pM->univerArray,
                    pM->arSize*sizeof(Univer));
    pM->arSize = newSize;
    Univer *oldAr = pM->univerArray;
    pM->univerArray = newAr;
    delete[] oldAr;
```

```
}
```

2.3.2.2 Реалізація функцій порівняння структур

Як вже наголошувалося, ці функції потрібні для пошуку місця нової структури серед існуючих нащадків вузла.

Структура функцій однакова. В якості параметрів вони приймають вказівники на дві структури, що порівнюються, а повертають ціле число. Це число дорівнює 0, якщо структури однакові. Число має бути від'ємним, якщо перша структура «менша» і додатнім, якщо «меншою» є друга структура. Поняття «менша структура» у даному контексті означає, що ця структура у переліку має знаходитися раніше.

Алгоритм функції визначається правилом порівняння. В якості прикладу наведемо функцію порівняння університетів, лістинг 3.9.

Лістинг 3.9 – Код функції cmpUniver()

```
int cmpUniver(Univer u1, Univer u2){
    if(u1.level != u2.level)
        return u1.leve2 – u1.level;
    return strcmp(u1.name, u2.name;
}
```

Наведена функція забезпечить розташування університетів за спаданням рівня акредитації. Університети з однаковим рівнем будуть розташовані відповідно з назвою у порядку абетки.

2.3.3 Реалізація функцій додавання вузлів дерева

Маючи допоміжні функції можемо перейти до функцій додавання вузлів до дерева. Ці функції теж розташуємо у файлі add.cpp.

Функції мають схожу структуру. Вони нічого не повертають. В якості параметрів ці функції приймають посилання на вузол, до якого додається новий нащадок, і посилання на цього нащадка.

Алгоритми роботи функцій дещо відрізняються, бо нащадки зберігаються по різному.

2.3.3.1 Реалізація функції додавання «університету

Лістинг цієї функції наведено у лістингу 3.10.

Лістинг 3.10 – Код функції додавання університету

```
void addUniver(Ministry *pM, Univer *pU){
    pU->parent = pM;
    if(pM->arSize == pM->univerCount)
        growUniverAr(pM);
```

```
        // Insert new univer in sorted order
        int n = pM->univerCount;
        while(n > 0 &&
              cmpUniver(&pM->univerArray[n - 1], pU) > 0 ){
            pM->univerArray[n] =  pM->univerArray[n - 1];
            n--;
        }
        //Put new university into array
        memcpy(&pM->univerArray[n], pU,sizeof(Univer));
        pM->univerCount++;
        delete pU;
    }
```

Алгоритм роботи функції наступний.

Спочатку аналізується наповненість масиву університетів і у разі потреби викликається функція growUniverAr для виділення додаткової пам'яті.

Далі у циклі, починаючи з кінця, визначається місце для нової структури відповідно із значеннями її полів так, щоб не порушити порядок сортування. Елементи масиву, що «більші» зсуваються праворуч.

Нова структура вставляється у масив шляхом копіювання параметру, посилання на який передається до функції.

Виходячи з того, що у масиві університетів зберігаються самі структури, у функціях cmpUniver і memcpy використовується операція & для отримання адреси структури.

Пам'ять, яку займав параметр, через який передавалася нова структура, вивільняється.

2.3.3.2 Реалізація функції додавання «факультету»

Код функції наведено у лістингу 3.11.

Лістинг 3.11 – Код функції додавання факультету

```
    void addFaculty(Univer *pU, Faculty*pF){
        pF->parent = pU;
        if(pU->arSize == pU->facultyCount)
            growFacultyAr(pU);
        int n = pU->facultyCount;
        while(n > 0 &&
              cmpFaculty(pU->pFacultyArray[n - 1], pF) > 0){
            pU->pFacultyArray[n] =  pU->pFacultyArray[n - 1];
            n--;
        }
        //Put new university into sorted array
        pU->pFacultyArray[n] = pF;
        pU->facultyCount++;
    }
```

Алгоритм роботи цієї функції такий самий, як і у попередньої. Але тут ми працюємо з посиланнями на структури, тому не потрібні операції

отримання адреси і замість операції копіювання використовується присвоєння.

2.3.3.3 Реалізація функції додавання кафедри

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

Код функції наведено у лістингу 3.12.

Алгоритм додавання розпадається на три гілки.

У першому випадку список пустий, тому посилання на новий елемент записується у поле firstDept структури факультет.

У другому варіанті новий елемент менший ніж елемент, на який вказує посилання firstDept, тому новий елемент має стати першим, а firstDept наступним.

У третьому випадку доводиться йти списком і шукати місце для нового елементу і знайшовши це місце, вставити туди новий елемент

Лістинг 3.12 – Код функції додавання кафедри

```
void addDept(Faculty *pF, Dept *pD){
    pD->parent = pU;
    pD->nextDept = NULL;
     //Insert new department
    if(pF->firstDept == NULL) // List is empty
        pF->firstDept = pD;
    else if(cmpDept(pD, pF->firstDept) < 0){
        // New element is less then first element
        pD->nextDept = pF->firstDept;
        pF->firstDept = pD;
    }
    else{
        Dept *p = pF->firstDept;
        // Search place for new element
        while(p->nextDept != NULL &&
                              cmpDept(pD, p->nextDept)>=0)
            p = p->nextDept ;
        // Inserting new elemet
        pD->nextDept = p->nextDept;
        p->nextDept = pD;
    }
}
```

2.3.4 Реалізація функції створення дерева

Головне призначення цієї функції – тестування реалізацій функцій. Код її може змінюватися в процесі розробки проекту. Туди, окрім функцій створення вузлів та їх додавання, можна буде добавляти виклики функцій редагування, вилучення вузлів та інші. До того ж зовсім не обов’язково на початку створювати і додавати вузли усіх рівнів. Можна створювати дерево поступово.

Функцію розташуємо у файлі tree.cpp. На початку файлу напишемо директиви #include “type.h”. Це, зокрема, дозволить використовувати зовнішню глобальну змінну root.

У реалізації функції, що наведена у лістингу 3.13, ми створюємо корінь, а також по декілька університетів, факультетів, кафедр та зв’язуємо їх між собою. Після створення викликаємо функцію showTree, яку розглянемо далі.

У функції використовується функція delTree, яка буде реалізована пізніше, тому зараз для неї можна створити заготовку з пустим вмістом.

Лістинг 3. 13 – Код функції створення дерева

```
int createDefaultTree(void*){
    if(root != NULL) delTree(NULL);
    root = createMinistry("Ministry", 1000);
    Univer *univer1 = createUniver("University One", 4);
    Univer *univer2 = createUniver("University Two", 3);
    Univer *univer3 = createUniver("University Three", 2);
    Univer *univer4 = createUniver("University Four", 4);
    addUniver(root, univer1);
    addUniver(root, univer2);
    addUniver(root, univer3);
    addUniver(root, univer4);
    Faculty *f1 = createFaculty("Faculty 1", "Dean 1");
    Faculty *f2 = createFaculty("Faculty 2", "Dean 2");
    Faculty *f3 = createFaculty("Faculty 3", "Dean 3");
    Faculty *f4 = createFaculty("Faculty 4", "Dean 4");
    addFaculty(&root->univerArray[0], f4);
    addFaculty(&root->univerArray[0], f3);
    addFaculty(&root->univerArray[0], f2);
    addFaculty(&root->univerArray[0], f1);
    Dept *d1 = createDept("Dept1", 5, 0);
    Dept *d2 = createDept("Dept2", 10, 1);
    Dept *d3 = createDept("Dept3", 6, 0);
    Dept *d4 = createDept("Dept4", 20, 1);
    Dept *d5 = createDept("Dept5", 15, 1);
    addDept(f1, d1);
    addDept(f1, d5);
    addDept(f1, d4);
    addDept(f2, d2);
    addDept(f3, d3);
    showTree(NULL);
    return 0;
}
```

Особливість створення дерева полягає в тому, що міністерство зберігає копії університетів, які передавалися до функції add через параметри. Як наслідок, структури університету, що оголошені у цій функції, вже не є структурами, що належать університету. Тому до університетів доводиться звертатися через корінь.

2.3.5 Реалізація функції відображення дерева

Ця функція дозволяє переглядати структуру дерева на консолі і суттєво допомагає під час тестування роботи додатку. Її ми теж розташуємо у файлі tree.cpp.

Код функції наведено у лістингу 3.14.

Лістинг 3.14 – Код функції відображення дерева

```
int showTree(void* ){
    system("cls");
    cout << " --- Tree structure ---\n";
    if(root == NULL){
        cout << " Tree does not exist\n";
        pause();
        return;
    }
    //Ministry *root = (Ministry*)ptr;
    cout << root->name << ", budget = "
            << root->budget << endl;
    //Univer loop
    Univer *uAr = root->univerArray;
    for(int i = 0; i < root->univerCount; i++){
        cout <<  "\t" << uAr[i].name << ", level = "
                << uAr[i].level <<endl;
        //Faculty loop
        Faculty ** fpAr = uAr[i].pFacultyArray;
        for(int j = 0; j < uAr[i].facultyCount; j++){
            cout <<  "\t\t" << fpAr[j]->name << ", dean "
                    << fpAr[j]->dean << endl;
            // Dept loop
            Dept *pD = fpAr[j]->firstDept;
            while(pD != NULL){
                cout << "\t\t\t" << pD->name
                     << ", size: "<< pD->size
                     << (pD->spec ? ", special" : "")<< endl;
                pD = pD->nextDept;
            }
        }
    }
    pause();
    return 0;
}
```

Функція завершується викликом допоміжної функції pause(), яка зупиняє виконання програми до натискання якої небудь клавіші.

Код функції pause() наведено у лістингу 3.15. Її слід розташувати у файлі util.cpp.

Лістинг 3.15 – Код функції pause()

```
#include "type.h"
void pause(){
    cout << " Press any key to continue.\n" ;
    getch();
}
```

2.3.6 Формування головного файлу проекту main.cpp

Після створення розглянутих вище функцій можна провести перше тестування додатку. Для цього скористаймося файлом main.cpp.

2.3.6.1 Оголошення глобальних змінних

Оголосимо у файлі, одразу після директиви #include “type.h”, глобальну змінну root, яка буду містити посилання на корінь дерева, та глобальну змінну logFileName з іменем файлу протоколу. Ці змінні будуть використовуватися також в інших файлах проекту:

Ministry *root; //корінь дерева.

char* logFileName = “log.txt”; // ім’я файлу протоколу.

В інших файлах, там де ці змінні будуть потрібні, вони будуть доступні через файл type.h, тому що там оголошені із специфікатором extern.

Зазначимо, що у файлі, де для такої змінної виділяється пам’ять, специфікатор extern не потрібен. Докладніше про роль цього специфікатору можна почитати у розділі 12.2 підручника, сторінка 238.

2.3.6.2 Реалізація функції main()

Функція main(), лістинг 3.3, на першому етапі нашого проекту буде вирішувати тільки одне завдання - створення дерева. Для цього активізується відповідна функція, яка формує дерево і виводить його на консоль.

Лістинг 3.16 – Текст функції main() проекту (перший варіант)

```
#include "type.h"
Ministry *root;
extern char* logFileName = "log.txt";
int main(int argc, char *argv[])
{
    SetConsoleTitleA("Project 'Education Ministry', "
                     "student Ivanchuk M.P., gr. KI-191");
    createDefaultTree(NULL);
    return 0;
}
```

Після запуску додатку на екрані має з’явитися зображення дерева у такому вигляді, як показано на рисунку 3.3

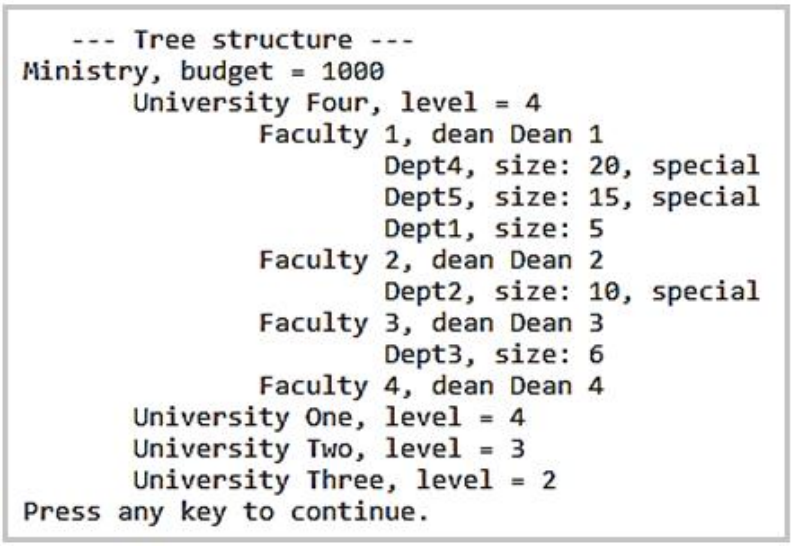

Рисунок 3.3 – Вигляд дерева на консолі

2.3.7 Реалізація функцій редагування вузлів

Ці функції розташуємо у файлі edit.cpp. На початку файлу напишемо директиву #include “type.h”. Далі можна писати реалізації функцій.

Але доводиться враховувати той факт, що функції редагування міняють вміст полів, і таким чином можуть порушувати впорядкованість переліку нащадків вузла. Тому після редагування потрібно впорядкувати перелік. Для вирішення цієї проблеми реалізуймо функції впорядкування нащадків.

void sortUniver(Ministry *pM);
void sortFac(Univer *pU);
void sortDept(Faculty *pF);

Найбільш прийнятним алгоритмом впорядкування переліків університетів і факультетів буде алгоритм сортування масивів вставкою.

Впорядкування списку кафедр краще виконати шляхом перезапису списку. Для цього у структурі факультет створити посилання на пустий список, після чого додати до факультету посилання на кафедри зі старого списку.

Усі ці функції будуть використовувати функції порівняння, що були наведені у розділі додавання.

В якості прикладу розглянемо реалізації деяких з них.

2.3.7.1 Функція сортування масиву університетів

Код функції наведено у лістингу 3.17.

Лістинг 3.17 – Код функції сортування масиву університетів

```
void sortUniver(Ministry *pM){
    int n = pM->univerCount;
    Univer* ar = pM->univerArray;
    for(int k = 1; k < n; k++){
        if(cmpUniver(&ar[k-1], &ar[k])>0){
```

```
                int j = k ; Univer b = ar[j];
                do{
                    ar[j] = ar[j - 1];
                    j--;
                }while(j >0 && cmpUniver(&ar[j], &b)>0);
                ar[j] = b;
            }
        }
    }
```

Функція сортування масиву факультетів буде відрізнятися від наведеної тільки тим, що там буде непотрібною операція визначення адреси структур, тому що університет містить масив вказівників на структури, а не масив структур.

2.3.7.2 Функція впорядкування списку кафедр

Код функції наведено у лістингу 3.18.

Лістинг 3.18 – Код функції впорядкування масиву кафедр

```
 void sortDept(Faculty *pF){
     Dept *pD = pF->firstDept;
     pF->firstDept = NULL;
     while(pD != NULL){
         Dept *oldNext = pD->nextDept;
         addDept(pF, pD);
         pD = oldNext;
     }
 }
```

2.3.7.3 Функція редагування інформаційних полів університету

Усі функції редагування мають схожу структуру. Вони нічого не повертають. В якості параметрів приймають посилання на вузол та перелік параметрів, що мають бути змінені. Як приклад наведемо код функції редагування університету, лістинг 3.19.

Лістинг 3.19 – Код функції редагування інформаційних полів університету

```
void editUniver(Univer *pU, char *newName, int level){
    strncpy(pU->name, newName, SIZE_NAME);
    pU->level = level;
    sortUniver(pU->parent);

}
```

2.3.8 Реалізація функцій видалення вузлів дерева

Ці функції розташуємо у файлі del.cpp. На початку файлу напишемо директиву #include “type.h”. Далі можна писати реалізації функцій.

У розділі «Розробка» розглядалося два варіанти таких функцій. Перший варіант передбачав видалення тільки одного нащадка, другий – видалення усіх.

Функції, що вилучають один елемент, приймають посилання цей елемент, а повертають або те саме посилання, або NULL, якщо елемент не знайдено. Звільняти пам’ять має функція, що викликає дану.

Друга група функцій приймає посилання на батьківський вузол, у якого потрібно видалити усіх нащадків. Ці функції нічого не повертають.

Особливість реалізації цих функцій полягає в необхідності звільняти пам'ять, яку займали елементи, що видаляються. До того ж, видалення елементів вищих рівнів потребує попереднього видалення усіх його нащадків. В якості прикладу розглянемо функцію видалення усіх кафедр факультету та функцію видалення одного факультету.

2.3.8.1 Функція видалення усіх кафедр факультету

Код функції наведено у лістингу 3.20.

Лістинг 3.20 – Код функції видалення усіх кафедр факультету

```
void delAllDept(Faculty *pFaculty){
    Dept *pD = pFaculty->firstDept;
    while(pD !=NULL){
        Dept *toDel = pD;
        pD = pD->nextDept;
        delete toDel;
    }
    pFaculty->firstDept = NULL;
}
```

2.3.8.2 Функція видалення факультету з університету

Код функції наведено у лістингу 3.21.

Лістинг 3.21 – Код функції видалення факультету з університету

```
Faculty* delFaculty (Faculty *pDel){
    Univer *pU = pDel->parent;
    delAllDept(pDel);
    for(int delIdx = 0; delIdx < pU->facultyCount; delIdx++){
        if(pU->pFacultyArray[delIdx] ==  pDel){
            for(int i = delIdx ; i < pU->facultyCount -1 ; i++)
                pU->pFacultyArray[i] = pU->pFacultyArray[i + 1];
            pU->facultyCount--;
            return pDel;
        }
```

```
    }
    return NULL;
}
```

У цій функції ми спочатку шукаємо індекс факультету, що підлягає вилученню. Після цього робимо зсув елементів масиву праворуч

2.3.9 Реалізація функцій збереження дерева у файлі та відновлення з файлу

Схему збереження дерева у файлі було розглянуто у розділі «Розробка». Тут ми наведемо реалізації функції, яка записує інформацію про дерево до файлу, та функції, що відновлює дерево з файлу. Параметром для цих функцій є вказівник на відкритий файл. Функції розташуємо у файлі tree.cpp.

2.3.9.1 Реалізація функції для збереження дерева у файлі

Код функції наведено у лістингу 3.22.

Лістинг 3. 22 – Код функції для збереження дерева у файлі

```
void storeTree(FILE *f ){
    //Store ministry structure
    fwrite(root, sizeof(Ministry),1, f);
    //Preparing for university loop
    Univer *uAr = root->univerArray;
    for(int i = 0; i < root->univerCount; i++){
        //Store current university structure
        fwrite(&uAr[i], sizeof(Univer),1, f);
        //Preparing for faculty loop
        Faculty ** pfAr = uAr[i].pFacultyArray;
        for(int j = 0; j < uAr[i].facultyCount; j++){
            //Store current faculty structure
            fwrite(pfAr[j], sizeof(Faculty), 1, f);
            //Store current faculty name with its length
            int len = strlen((pfAr[j])->name)+1;
            fwrite(&len, sizeof(int), 1, f);
            fwrite((pfAr[j]->name), sizeof(char),len, f);
            //Store current dean name with its length
            len = strlen((pfAr[j])->dean)+1;
            fwrite(&len, sizeof(int), 1, f);
            fwrite((pfAr[j])->dean, sizeof(char),len, f);
            //Preparing for department loop
            Dept *pD = pfAr[j]->firstDept;
            while(pD != NULL){
                //Store current department structure
                fwrite(pD, sizeof(Dept), 1, f);
                //Store department name
                int len = strlen(pD->name)+1;
                fwrite(&len, sizeof(int), 1, f);
                fwrite(pD->name, sizeof(char),len, f);
```

```
                pD = pD->nextDept;
            }
        }
    }
    fclose(f);
}
```

2.3.9.2 Реалізація функції відновлення дерева з файлу

Ця функція формує елементи нового дерева використовуючи інформацію з файлу.

Спочатку видаляється старе дерево, якщо воно існувало, після чого виділяється пам’ять для кореня дерева. Далі зчитується перший блок інформації з файлу і відповідна інформація заноситься у кореневу структуру. Після цього вже стає відомо, скільки потрібно виділити пам’яті для масиву університетів.

Наступний крок – виділення пам’яті під масив університетів і організація циклу заповнення цього масиву блоками інформації з файлу.

Після відновлення структури для першого університету виділяємо пам’ять для масиву вказівників на факультети і запускається цикл відновлення структур факультетів. У цьому циклі після зчитування структури факультету зчитуються також довжина і вміст назви факультету та імені декана.

Далі аналізується вказівник на початок списку кафедр і якщо він не NULL, відновлюється інформація про всі кафедри факультету.

Після цього продовжується виконання вкладених циклів.

Лістинг 3. 23 – Код функції відновлення дерева з файлу

```
void  restoreTree(FILE *f){
    if(root != NULL)
        delTree(NULL);
    //Create new root and initialise it from file
    root = new Ministry();
    fread(root, sizeof(Ministry),1, f);
    //Allocate memory for array of universities
    root->univerArray = new Univer[root->arSize];
    //University loop
    for(int i = 0; i < root->univerCount; i++){
        //Restore university structure into array
        fread(&root->univerArray[i], sizeof(Univer), 1, f);
        Univer *un = &root->univerArray[i];
        un->parent = root;
        //Allocate memory for  array of pointers to faculty
        un->pFacultyArray = new Faculty*[un->arSize];
        for(int j = 0; j < un->facultyCount; j++){
            //Create and restore faculty structure
            Faculty *fc = new Faculty();
            fread(fc, sizeof(Faculty), 1, f);
            fc->parent = un;
            un->pFacultyArray[j] = fc;
```

```
                    //Restore faculty name
                    int len;
                    fread(&len, sizeof(int), 1, f);
                    char *fName = new char[len];
                    fread(fName, sizeof(char), len,f);
                    fc->name = fName;
                    //Restore faculty dean name
                    fread(&len, sizeof(int), 1, f);
                    char *dName = new char[len];
                    fread(dName, sizeof(char), len,f);
                    fc->dean = dName;
                    if(fc->firstDept != NULL){
                        Dept *dp = new Dept();
                        fread(dp, sizeof(Dept), 1,  f);
                        dp->parent = fc;
                        fc->firstDept = dp;
                        //Read length of name
                        int len;
                        fread(&len, sizeof(int), 1, f);
                        //Read  name
                        char *dName = new char[len];
                        fread(dName, sizeof(char), len,f);
                        dp->name = dName;
                        while(dp->nextDept != NULL){
                            Dept *dpnxt = new Dept();
                            fread(dpnxt, sizeof(Dept), 1, f);
                            dp->parent = fc;
                            dp->nextDept = dpnxt;
                            dp = dpnxt;
                            //Read length of name
                            fread(&len, sizeof(int), 1, f);
                            //Read  name
                            char *dName = new char[len];
                            fread(dName, sizeof(char), len,f);
                            dp->name = dName;
                        }
                    }
                }
            }
            fclose(f);
        }
```

Після створення цих функцій слід модифікувати функцію main і спочатку спробувати записати дерево у файл, а потім його відновити.

2.4 Реалізація інтерфейсу користувача

2.4.1 Реалізація допоміжних функцій

Користувачеві додатку, що розробляється, у процесі спілкування з програмою доведеться вводити числову і текстову інформацію через консоль. Використання для цього об’єкту cin та методу getline() цього об’єкту, а також

функції gets(), у деяких ситуаціях є причиною виникнення збоїв у програмі. Для того, щоб уникнути таких небажаних випадків, і спростити введення даних у діалогах з програмою створімо спеціальні функції для введення рядка символів і цілого числа.

char*getStr(string prompt);

int getInt(string prompt, int min, int max);

Перша з цих функцій в якості параметру приймає рядок, який програма буде виводити на екран, запрошуючи користувача ввести дані.

Друга функція, окрім тексту запрошення, приймає мінімальне та максимальне допустимі значення числа, що вводитья.

2.4.1.1 Функція для введення цілого числа

Ця функція виводить на екран запит на введення даних та контролює відповідність цих даних заданому діапазону. Якщо дані вводяться невірно, запит повторюється.

Код функції наведено у лістингу 3.24.

Лістинг 3. 24 – Код функції введення цілого числа

```
int getInt(string prompt, int min, int max){
    int num; string s;
    do{
        cout << prompt;
        getline(cin,s);
        num = atoi(s.c_str());
    } while(num < min || num > max);
    return num;
}
```

2.4.1.2 Функція для введення рядка символів

Ця функція виводить запит на введення рядка символів, а після введення цього рядка виділяє пам’ять під нього і повертає посилання на відповідну ділянку пам’яті.

Код функції наведено у лістингу 3.25.

Лістинг 3. 25 – Код функції введення рядка символів

```
char* getStr(string prompt){
    cout << prompt;
    string s;
    getline(cin,s );
    //allocate memory for char array
    char* cStr = new char[s.length() +1];
    strcpy(cStr, s.c_str());
    return cStr;
}
```

2.4.2 Реалізація функцій для створення та активізації меню

Відповідні функції розташуємо у файлі menu.cpp.

Усі функції для створення консольних меню мають схожу структуру.

У тексті функції обов'язково прописується перелік опцій меню у вигляді масиву структур типу MenuUnit.

Функція може містити також підготовчі операції до запуску. Це можуть бути, наприклад, виклики списків вузлів, із яких вибирається потрібний вузол.

Останнім кроком є виклик функції запуску меню.

Ця функція циклічно виводить на консоль інформацію про меню і перенумерований перелік опцій меню. Завершується перелік опцією вихід. Для цієї опції значення вказівника на відповідну функцію має дорівнювати NULL. Це є ознака завершення списку опцій меню.

Якщо вибрано цю опцію, цикл переривається. Інакше викликається функція, що пов'язана з вибраним елементом масиву. Далі цикл повторюється.

Цикл переривається і у тому випадку, якщо функція реалізації опції меню повернула не нульове значення.

Функція приймає масив структур пунктів меню, назву меню і вказівник, який вона буде передавати функціям реалізації відповідного пункту.

2.4.2.1 Функція запуску меню

Код функції запуску меню наведено у лістингу 3.26.

Лістинг 3. 26 – Код функції запуску меню

```
void runMenu(MenuUnit menu[], string menuName, void* ptr){
    while(true){
        system("cls");
        cout << menuName << endl;
        //Show menu option list with 1-based number
        int cnt = 0;
        for(;; cnt++){
            cout << " " << cnt + 1 <<". "
                    << menu[cnt].text<<endl;
            if(menu[cnt].func == NULL)
                break;
        }
        // Get 1-based variant number
        int v = getInt(" Enter your choice number: ",
                       1, cnt + 1);
        //Call menu function
        if(menu[v - 1].func == NULL)
            break; // Exit
        int result = menu[v - 1].func(ptr);
        if( result != 0){
            // Signal "Exit" from func";
            break;
        }
    }
```

```
    }
```

Далі розглянемо декілька функцій створення консольних меню.

2.4.2.2 Функція створення головного меню

Код функції наведено у лістингу 3.27.

Лістинг 3. 27 – Код функції створення головного меню

```
int mainMenu(void* ptr){
    // Create and run main menu
    MenuUnit menu[] = {
        {"Ministry operation", menuMinistry},
        {"University operation", menuUniver},
        {"Faculty operation", menuFaculty},
        {"Department operation", menuDept},
        {"Tree operation", menuTree},
        {"Query operation", menuQuery},
        {"Exit",NULL} };
    runMenu(menu, "\n    --- Main menu ---", ptr);
    return 0;
}
```

Як бачимо, головне меню викликає функції створення меню другого рівня. Меню другого рівня можуть викликати меню третього рівня і так далі.

Далі розглянемо функцію меню «міністерства».

2.4.2.3 Функція створення меню міністерства

Код функції наведено у лістингу 3.28. У цьому меню викликаються функції обробки кореня. Потреби передавати вказівник на вузол, що обробляється нема, бо корінь глобальна змінна. Тому до функцій обробки кореня через функцію runMenu передається NULL.

Лістинг 3. 28 – Код функції створення меню міністерства

```
 int menuMinistry(void*){
     MenuUnit menu[] = {
         {"Create root", createRoot},
         {"Show Ministry information", showMinistryInfo},
         {"Edit Ministry information", editMinistryByMenu},
         {"Add university", addUniverByMenu},
         {"Show list of university", showUniverList},
         {"Delete all university", delAllUniverByMenu},
         {"Return to main menu", NULL} };
     runMenu(menu, "\n    --- Ministry operation ---",
NULL);
     return 0;
 }
```

Далі розглянемо функцію створення меню для роботи з кафедрою.

2.4.2.4 Функція створення меню для роботи з кафедрою

Код функції наведено у лістингу 3.28.

Лістинг 3. 28 – Код функції створення меню для роботи з кафедрою

```
int menuDept(void* ){
    Univer *pU = selectUniver();
    if(pU == NULL) return 1;
    Faculty *pF = selectFaculty(pU);
    if(pF == NULL) return 1;
    Dept *pD = selectDept(pF);
    if(pD == NULL) return 1;
    string menuName = "\n    ---  ";
    menuName.append(pD->name).append(" ---");
    MenuUnit menu[] = {
        {"Show Department information", showDeptInfo},
        {"Edit Department information", editDeptByMenu},
        {"Delete Department", delDeptByMenu},
        {"Return to to main menu", NULL}};
    runMenu(menu, menuName, pD);
    return 0;
}
```

Перед тим, як створити меню кафедри, наша функція має довідатися про кафедру, яка буде оброблятися. Тобто потрібно отримати вказівник на цю кафедру. Але перед цим ще треба отримати вказівник на університет, а потім на факультет.

Для вирішення цих задач наша функція послідовно звертається до функцій selectUniver, selectFaculty, selectDept, які повертають вказівники на елемент, що користувач вибирає із списку.

Для прикладу далі розглянемо код функції selectDept.

2.4.2.5 Функція вибору кафедри із списку

Код функції наведено у лістингу 3.29.

Як можна побачити, функція створює діалог з користувачем, але сама список не виводить, а користується для цього функцією printDeptList. Ця допоміжна функція розглядається далі.

Отримавши через діалог номер кафедри у списку, функція шукає відповідний вказівник. Для масиву ця операція буде реалізовуватися простіше.

Лістинг 3. 29 – Код функції вибору кафедри із списку

```
Dept *selectDept(Faculty *pF){
    system("cls");
    cout << "\n     --- " << pF->parent->name << " ---\n";
    cout << "     --- " << pF->name << " ---\n";
    cout << "   --- Department selection ---\n";
```

```
        int nDept = printDeptList(pF);
        cout << " " << nDept + 1 << ". Exit\n";
        int num = getInt(" Enter your choice: ",
                         1, nDept + 1);
        if(num == nDept + 1)
            return NULL;
        Dept *pD = pF->firstDept;
        for(int i = 1; i < num; i++)
            pD = pD->nextDept;
        return pD;
    }
```

2.4.2.6 Функції для виведення списків нащадків

Це допоміжні функції, які просто друкують перелік нащадків на консолі. Їх можна буде використовувати не тільки для вибору нащадка у меню, але й для реалізації інших функцій.

int printUniverList();
int printFacultyList(Univer*);
int printDeptList(Faculty*);

Як приклад, наведемо код функції для виведення списку факультетів, лістинг 3.30.

Лістинг 3.30 – Код функції виведення списку кафедр

```
int printFacultyList(Univer* pU){
    int n = pU->facultyCount;
    Faculty **pF = pU->pFacultyArray;
    for(int i = 1; i <= n; i++, pF++){
        cout << " " << i << " name: " << (*pF)->name
             << ", dean: " << (*pF)->dean << endl;
    }
    return n;
}
```

2.4.3 Реалізація функцій роботи з деревом через меню

Ці функції будемо реалізуємо у файлах .cpp відповідно до функціонального призначення цих функцій

Фактично, усі основні завдання по обробці дерева вже вирішуються функціями, що призначені для роботи у програмному режимі. У функціях цього розділу залишається реалізувати діалог з користувачем, отримувати від нього дані, передавати ці дані вже реалізованим функціям, отримувати від них результат і повертати його користувачеві.

Для прикладу розглянемо реалізацію декількох функцій роботи з деревом через меню.

2.4.3.1 Функція виведення списку кафедр

Код функції наведено у лістингу 3.31.

Лістинг 3.31 – Код функції додавання університету через меню

```
int addUniverByMenu(void *){
    system("cls");
    cout << " --- New university adding ---\n";
    if(root == NULL){
        cout << " Tree was not created!\n";
        pause();
        return 1;
    }
    char* name = getStr(" New university name is: ");
    if(strlen(name) == 0){
        cout << " New university has not been created.\n";
        pause();
        return 1;
    }
    int level = getInt(" New university level is: ", 1, 4);
    addUniver(root, createUniver(name, level));
    showUniverList(NULL);
    return 0;
}
```

2.4.3.2 Функція редагування інформації про кафедру через меню

Код функції наведено у лістингу 3.32.

Лістинг 3.32 – Код функції редагування інформації про кафедру через меню

```
int editDeptByMenu(void* ptr){
    if(ptr == NULL) return 1;
    Dept *pD = (Dept*)ptr;
    system("cls");
    cout << " --- Department informatin editing ---\n";
    cout << " OLd department name is: " << pD->name << endl;
    char* newName = getStr(" New department name is: ");
    if(strlen(newName) == 0){
        cout << " The name has not been changed.\n";
        newName = pD->name;
    }
    cout <<  " Old department stuff size is: "
            << pD->size << endl;
    int size = getInt(" New department stuff size is: ", 5, 25);
    cout <<  " Old department status is: "
           << (pD->spec ? "special" : "not special") << endl;
    int spec = getInt(" New department status"
                      "(1-special, 0-not special)is: ", 0, 1);
    editDept(pD,newName, size, spec);
    showDeptInfo(pD);
```

```
    return 0;
}
```

2.4.3.3 Функція видалення факультету через меню

Код функції наведено у лістингу 3.33.

Лістинг 3.33 – Код функції видалення факультету через меню

```
 int delFacultyByMenu(void *ptr){
     Faculty*pF = (Faculty*)ptr;
     Univer *pU = pF->parent;
     if(delFaculty(pF) != NULL){
         delete pF;
         pF = NULL;
     }
     showFacultyList(pU);
     return 1;
 }
```

2.4.3.4 Функція завантаження дерева з файлу через меню

Код функції наведено у лістингу 3.34.

Лістинг 3.34 – Код функції завантаження дерева з файлу через меню

```
int  restoreTreeByMenu(void*){
    system("cls");
    cout << "\t--- Restoring the tree from a file ---\n";
    char* fileName = getStr("Enter file name: ");
    FILE *f;
    if((f = fopen(fileName, "rb")) == NULL){
        cout << " Error opening file " << fileName << endl;
        pause();
        return 1;
    }
    restoreTree(f);
    showTree(NULL);
    //Send message to log file
    string s = "Tree saved in file ";
    s.append(fileName).append("\n");
    printToLog(s);
    return 0;
}
```

2.4.4 Остаточна реалізація функції main()

У остаточній реалізації функції main() передбачимо тільки відкриття файлу протоколу та запуск головного меню.

Код функції наведено у лістингу 3.35.

Лістинг 3.31 – Код функції додавання університету через меню

```
#include "type.h"
Ministry *root;
extern char* logFileName = "log.txt";
int main(int argc, char *argv[]){
   SetConsoleTitleA("Project 'Education Ministry', "
                    "student Ivanchuk M.P., gr. KI-191");
    //Send message to log file
    string s = "Project 'Education Ministry'\n";
    s.append("Student Ivanchuk M.P., gr.KI-203\n");
    openLogFile(s);
    mainMenu(NULL);
    return 0;
}
```

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 / Dennis M. Ritchie, Brian W. Kernighan, – 2nd ed. – Prentce-hall, inc., 1988. – 263 p.
3. Kochan S.G. Programming in C/– 3nd ed. – Sams Publishing, 2004. – 505 p.

ІНФОРМАЦІЙНІ РЕСУРСИ

1. https://www.guru99.com/c-programming-tutorial.html
2. https://eln.stu.cn.ua/login/index.php