ЗМІСТ

ВСТУП

У сучасному світі майже всі програмні застосунки містять такі елементи як кнопки, перемикачі, прапорці, радіокнопки та інші елементи, необхідні для інтерактиву користувача з застосунком. Кожен застосунок має фронтенд (Frontend) та бекенд (Backend). Фронтенд - це те, що бачить користувач, коли він користується застосунком, а бекенд - це внутрішня логіка застосунку, де організовано підвантаження та обробку даних, а також відбувається процес взаємодії фронтенду із бекендом.

Важливо правильно вміти організувати цей зв’язок, інакше або застосунок буде неправильно функціонувати, або можуть бути пошкоджені певні дані в ході роботи, або він буде надто уповільнено працювати у зв’язку з великим навантаженням оперативної пам’яті пристрою та використанням неефективних алгоритмів. Саме для запровадження ефективних алгоритмів, правильної організації процесів додатку та взаємозв’язку його компонентів.

Метою курсової роботи є розробка програми з використанням класів - абстрактного класу Command, похідних від нього CopyCommand, CutCommand, PasteCommand та UndoCommand, контейнерного класу збереження застосованих команд, класу редактора Editor, безпосередньо класу-відправника команд, використовуючи концепції ООП, а саме - інкапсуляцію, абстракцію та поліморфізм. Також для реалізації завдання були використані динамічні масиви даних стандартної бібліотеки шаблонів STL-контейнер stack та патерн “Команда”.

1 ПОСТАНОВКА ЗАДАЧІ

Розробити контейнерний клас для збереження та обробки масиву редакторів/додатків за паттерном «Команда». Для цього розробити:

– абстрактний клас - Команда (Command), та похідні від нього класи команд – копіювання(CopyCommand), вставки (PasteCommand), вирізання (CutCommand) та відміни(UndoCommand);

– контейнерний клас збереження застосованих команд;

– клас редактора - Editor, що містить безпосередні операції над текстом. Він відіграє роль одержувача – команди делегують йому свої дії;

– клас, який налаштовує об'єкти для спільної роботи. Він виступає у ролі відправника – створює команди, щоб виконати якісь дії.

Команди, які змінюють стан редактора (наприклад, команда вставки тексту з буфера обміну), зберігають копію стану редактора перед виконанням дії. Копії виконаних команд розміщуються в історії команд, звідки вони можуть бути доставлені, якщо потрібно буде скасувати виконану операцію.

Класи елементів інтерфейсу, історії команд та інші не залежать від конкретних класів команд, оскільки працюють з ними через загальний інтерфейс. Це дозволяє додавати до програми нові команди, не змінюючи наявний код.

Для роботи з масивом об'єктів використати контейнерний клас – stack.

Програма курсової роботи повинна зберігати та зчитувати дані контейнеру у зовнішньому файлі, мати меню для обробки контейнеру (відкриття файлу з контейнером, збереження, редагування, додання, видалення записів, сортування, пошук, друк та інше).

2 ОСНОВНА ЧАСТИНА

2.1 Аналіз вхідних та вихідних даних

Тестова програма розробленої ієрархії класів буде зберігати масив команд, які виконуються над текстовими файлами. Ці команди включають вставку, вирізання, видалення. Кожна команда зберігає стан редактора перед виконанням, що дозволяє скасувати попередні дії.

Вхідними даними розробленої тестової програми є значення елементів ієрархії класів - Копіювання: значення початкової та кінцевої позицій у тексті, Вставка: значення початкової та кінцевої позицій у тексті та текст для вставки, Вирізання: значення початкової та кінцевої позицій у тексті, Видалення: значення початкової та кінцевої позицій у тексті, Скасування: остання зроблена дія, Повторення: остання скасована дія; номер пункту меню для виконання дії над елементами контейнера – stack, введених з клавіатури або отримані з зовнішнього файлу.

Вихідні дані програми включають вивід: меню застосунку, даних ієрархії класів на екран, стану редактора після виконання команди, змісту текстового файлу, повідомлення про успішне виконання команди або помилку, якщо команда не може бути виконана.

2.2 Методи та засоби програмування

Для виконання поставленого завдання курсової роботи необхідні такі розділи програмування мовою C++, як:

- класи, інкапсуляція;
- дружні класи;
- зв’язки: асоціація, композиція, реалізація, залежність;

– поліморфізм віртуальних функцій;

– контейнер STL - stack;

– інструменти по роботі з файловими потоками вводу-виводу;

– патерн проектування “Команда”.

2.2.1 Об’єктно-орієнтована парадигма розвиває об’єктне (модульне) програмування засобами створення ієрархій об’єктів і класів, які керують складністю програмних проектів, відділяючи внутрішню складність конструкції від її зовнішнього використання.

Кожен об'єкт програми має тип, або клас, який визначає його поведінку. Класи об’єднують атрибути, необхідні для зберігання даних, разом із методами - функціями для їх обробки.

Члени класу, спільні для всіх об’єктів класу, – статичні атрибути та методи. Їх особливість - для їх виклику не потрібен об’єкт - немає поточного об’єкта, до якого їх застосовують. Тому для методів класу немає указника на поточний об’єкт this [1].

В об’єктно-орієнтованій парадигмі є декілька основних технологій: успадкування, модульність, поліморфізм та інкапсуляція.

Одна з цих технологій - інкапсуляція - забезпечує приховування даних: прямий доступ до даних є неможливим, тому вони приховані від зовнішніх дій, що захищає їх від випадкових змін [2].

2.2.2 Клас може надати виняткові права доступу до своєї закритої частини іншому класу, окремій функції іншого класу чи вільній функції, оголошення яких усередині класу позначено ключовим словом friend. Тоді відповідна програмна конструкція здобуває статус дружньої: у класу з’являються дружні класи (friend class) або дружні функції (friend func- tion). Надання виняткових прав доступу – серйозне порушенням інкапсуляції, оскільки воно розширює загальновизнаний набір конструкцій, яким надано доступ до об’єкта [1].

2.2.3 Існує п’ять різних типів зв’язків між класами: залежність, асоціація, агрегація, композиція та успадкування.

Відношення асоціації, наведене на рисунку 2.1, показує, що один клас (зверху на рисунку - poolMonitor) містить атрибут, тип якого є другим класом (зверху на рисунку - phClass).

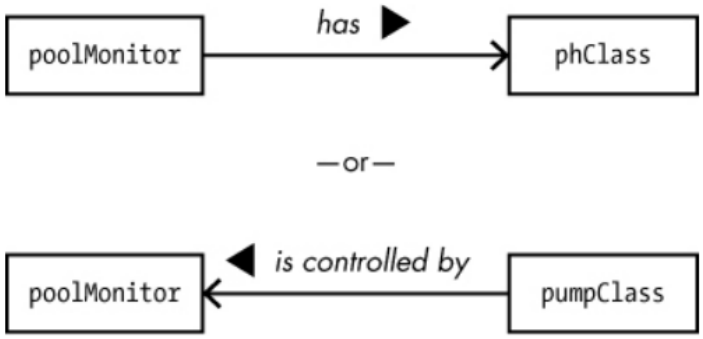

Рисунок 2.1 - Відношення асоціації

У відношеннях композиції менші класи, які містяться в більшому класі, не є самостійними класами: вони існують суто для підтримки класу, що містить або складає. Тривалість життя складового об’єкта та об’єктів-частин однакова. Компонуючий об’єкт відповідає за виділення та звільнення пам’яті, пов’язаної з частинами. Приклад наведено на рисунку 2.2.

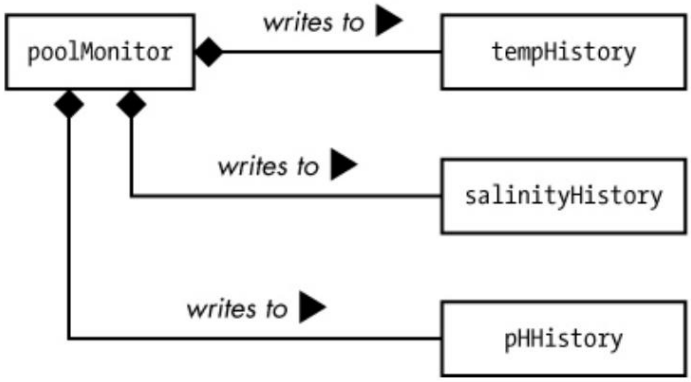

Рисунок 2.2 - Відношення композиції [3]

Два класи мають відношення залежності, коли об’єкти одного класу працюють з об’єктами іншого класу. У прикладі, наведеному на рисунку 2.3, класи userInterface і poolMonitor працюють разом, коли об’єкт userInterface хоче отримати дані для відображення (наприклад, коли ви передаєте об’єкт poolMonitor методу userInterface як параметр).

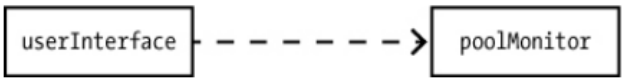

Рисунок 2.3 - Відношення залежності

Інтерфейс - це набір операцій, які очікуються від певних класів. Якщо клас реалізує інтерфейс, він фактично успадковує всі операції з цього інтерфейсу. Щоб показати, що клас реалізує даний інтерфейс, ви малюєте пунктирну лінію з порожнистою стрілкою від класу до діаграми інтерфейсу.

Відношення реалізації, наведене на рисунку 2.4, показує, що клас (phClass) реалізує даний інтерфейс (sensor) або абстрактний клас.

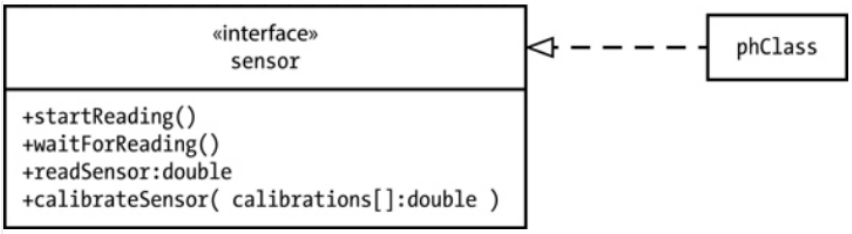

Рисунок 2.4 - Відношення реалізації [3]

2.2.4 Використовуючи механізми успадкування, можна розробити загальний клас, який визначає характеристики, що є властиві множині взаємопов'язаним між собою елементам. Цей клас потім може успадковуватися іншими, вузько спеціалізованими класами з додаванням у кожен з них своїх, властивих тільки їм унікальних особливостей.

Під терміном “поліморфізм” розуміють механізм реалізації функції, у якому різні результати можна отримати за допомогою одного її імені. З цієї причини поліморфізм іноді характеризується фразою "один інтерфейс, багато методів". Це означає, що до всіх функцій-членів класу можна отримати доступ одним і тим самим способом, незважаючи на можливу відмінність у конкретних діях, пов'язаних з кожною окремою операцією [2].

2.2.5 Стандартна бібліотека шаблонів (скор. “STL” від “Standard Template Library”) - це бібліотека, яка надає програмісту доступ до шаблонних класів і функцій загального призначення, які реалізують багато популярних і часто використовуваних алгоритмів і структур даних. Вона містить стеки, черги, вектори тощо [2].

Ядро стандартної бібліотеки шаблонів містить чотири основні елементи [4]:

– контейнери – спосіб організації зберігання даних вбудованих типів чи об’єктів власних класів;

– ітератори – вказують на елементи в контейнері. Збільшуємо ітератор - він переходить до наступного елементу;

– алгоритми – це процедури, які застосовуються до контейнерів для обробки їх даних різними способами;

– функціональні об’єкти – функції, які були загорнуті в класи, щоб мати змогу з ними взаємодіяти як з об’єктами.

У деяких контейнерах доступ до елементів відбувається за допомогою адресації, при цьому доступ до окремих елементів не обмежений. Але існують також і такі структури даних, в яких є обмеження доступу до елементів. Одним з представників таких спискових структур є стековий список або просто стек [5].

Стек (англ. Stack – стопка) – це структура даних, де нові елементи завжди записуються в її початок (вершину) і вибираються теж з початку. Метод доступу до елементів - LIFO (Last Input − First Output, "останнім прийшов − першим вийшов"; див. рис. 2.5). Найчастіше принцип роботи стека порівнюють зі стопкою тарілок: щоб взяти другу зверху, потрібно спочатку взяти верхню. Для отриман-

ня доступний лише один елемент - його вершина.

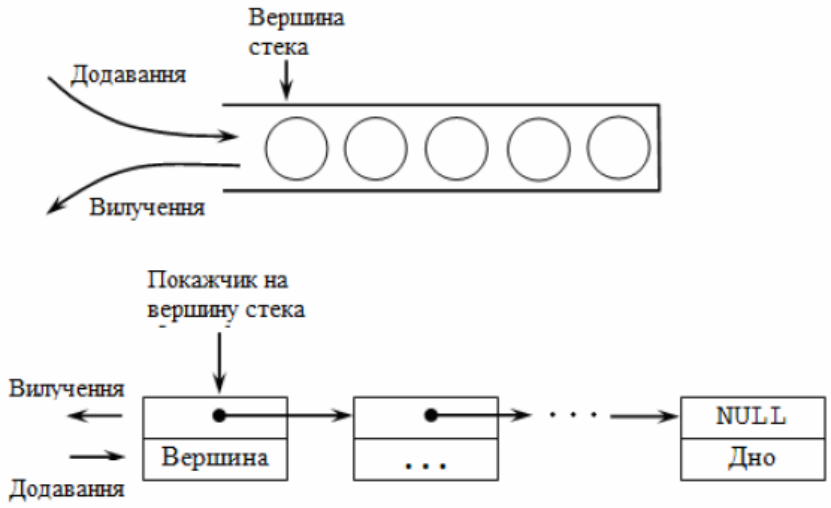

Рисунок 2.5 – Принцип організації посилань у стеку [5]

STL пропонує std::stack у заголовку <stack>. Шаблон класу stack приймає два параметри шаблону: базовий тип упакованого контейнера, наприклад int, а необов'язковий другий - тип упакованого контейнера, наприклад deque, list або vector [6].

Оголошення шаблону стека використовує такий синтаксис [7]:

template <class T, class Container = deque<T> > class stack;

Приклади оголошення стеків [8]:

stack <int, vector<int> > s2;

stack <int> i3;

Список операцій std::stack [6]:

– s.empty() - повертає true, якщо контейнер порожній:

if(s.empty()) { return false; }

– s.size() - повертає кількість елементів у контейнері:

for (int i = 0; i < s.size(); i++) {...}

– s.top() - повертає посилання на елемент у верхній частині стеку:

std::cout << s.top();

– s.push(t) - поміщає копію t наверх контейнера:

s.push(35);

– s.emplace(...) - створює T на місці, перенаправляючи ... відповідному конструктору:

s.emplace(42);

– s.pop() - видаляє елемент з вершини контейнера:

std::cout << s.top();

s.pop();

– s1.swap(s2) - змінює зміст s2 на s1:

s1.swap(myStack2);

2.2.6 У C++ передбачено інструменти для роботи з файлами у заголовку <fstream>. 3 типи потоків: вхідний, вихідний і введення-виведення. Для їх відкриття створюємо змінні:

– ifstream in; - вхідний потік;

– ofstream out; - вихідний потік;

– fstream both; - потік введення-виведення.

Відкриття файлу - за допомогою функції open(), яка повертає bool-значення на потік - потрібно завжди перед спробою отримання доступу до даних файла перевіряти результат її виклику. Наприклад: myStream.open("test");

Щоб закрити файл, використовується функція-член close(). Наприклад: myStream.close();

Запис/зчитування даних здійснюється за допомогою операторів "<<" і ">>".

Приклад запису та зчитування даних [2]:

ofstream out("test.txt");

out << 123.23 << endl;

ifstream in("test.txt");

float f;

in >> f;

2.2.7 Патерн проектування “Команда” інкапсулює запит як об’єкт, дозволяючи параметризувати клієнтам різні запити, запити черги або запити журналу, підтримувати операції, які можна скасувати (див. рис. 2.6).

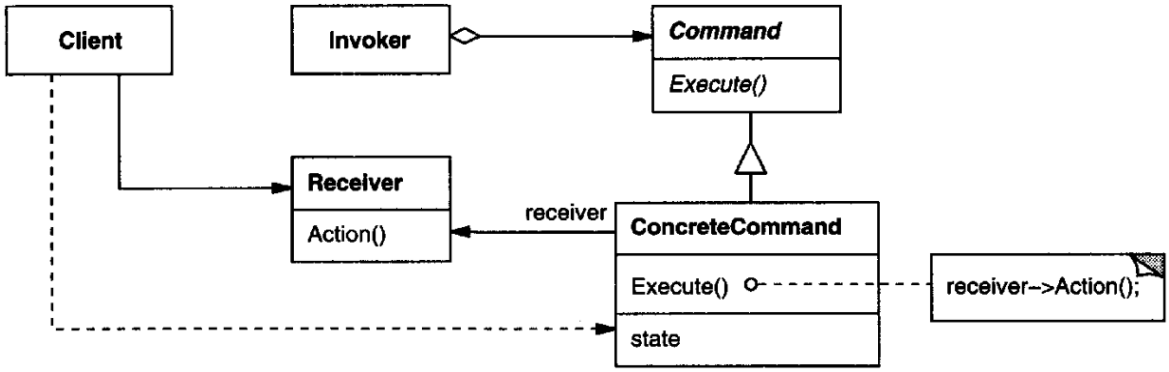

Рисунок 2.6 - Структура паттерну

Іноді необхідно надсилати запити об’єктам, не знаючи нічого про запитувану операцію чи одержувача запиту. Наприклад, набори інструментів інтерфейсу користувача включають такі об’єкти, як кнопки та меню, які виконують запит у відповідь на введення користувача. Але набір інструментів не може реалізувати запит явно в кнопці чи меню, тому що лише програми, які використовують набір інструментів, знають, що потрібно зробити з яким об’єктом.

Патерн Command дозволяє об’єктам набору інструментів робити запити до невизначених об’єктів програми, перетворюючи сам запит на об’єкт. Цей об’єкт можна зберігати та передавати, як і інші об’єкти. Ключем до цього шаблону є абстрактний клас Command, який оголошує інтерфейс для виконання операцій. У найпростішому вигляді цей інтерфейс містить абстрактну операцію Execute. Підкласи Concrete Command визначають пару приймач-дія, зберігаючи отримувача як змінну екземпляра та реалізовуючи Execute для виклику запиту. Одержувач має знання, необхідні для виконання запиту. Приклад реалізації наведено на рисунку 2.7 [9].

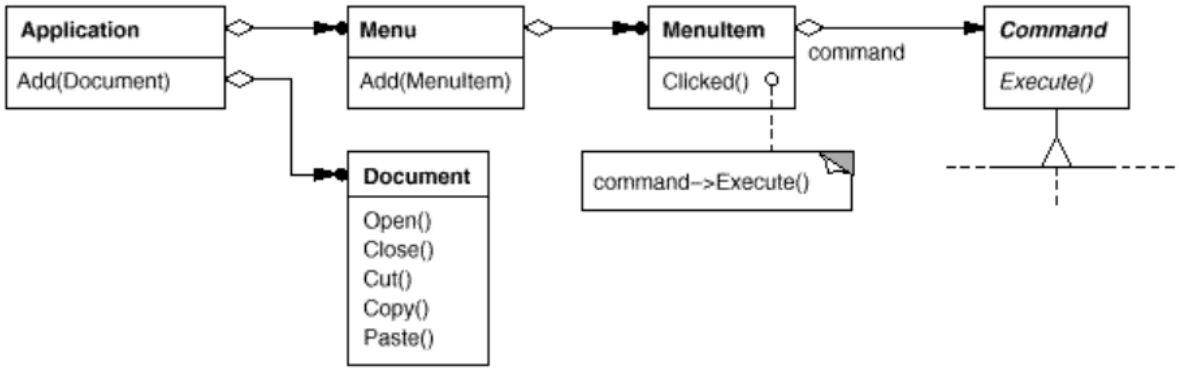

Рисунок 2.7 - Приклад реалізації набору інструментів інтерфейсу користувача [9]

2.3 Опис логічної структури програми

Для виконання завдання курсової роботи була спроектована та розроблена ієрархія класів: Session, SessionsHistory, Editor, Command, CopyCommand, PasteCommand, CutCommand, DeleteCommand, UndoCommand, RedoCommand, FilesManager, CommandsManager, Program. Діаграма класів та зв’язки між класами наведені на рисунку 2.8.

– Класи створюються для зберігання певної інформації. Розробленні класи мають таке призначення:

– Session - контейнерний клас, який містить історію команд у вигляді stack об’єктів Command, буфер обміну у вигляді stack текстових даних, ім’я та індекс команди, на якій користувач зупинився в момент работи з командами, в історії команд. Також в ньому визначено методи для проведення операцій над стеком історії команд та стеком буферу обміну;

– SessionsHistory - контейнерний клас, який містить історію сеансів у вигляді stack об’єктів Session. Також в ньому визначено методи для проведення операцій над стеком;

– Editor - клас, в якому міститься інформація про редактор текстових файлів, а саме - історія сеансів, сеанс, в якому в даний момент знаходиться користувач, текст, який в даному сеансі редагує користувач;

– Command - абстрактний клас, що описує загальну поведінку для класів команд, які його реалізують, а саме: виконання команди, її скасування, створення копії об’єкта команди. Також містить дані-параметри, необхідні безпосередньо для виконання команд: вказівник на об’єкт редактора, початковий та кінцевий індекси, змінна-копія тексту, який редагує користувач, текст для вставки, вказівник на минулу команду та вказівник на команду, яку потрібно скасувати або повторити;

– CopyCommand - клас команди копіювання, що реалізує абстрактний клас Command;

– PasteCommand - клас команди вставки, що реалізує абстрактний клас Command;

– CutCommand - клас команди вирізання, що реалізує абстрактний клас Command;

– DeleteCommand - клас команди видалення, що реалізує абстрактний клас Command;

– UndoCommand - клас команди скасування, що реалізує абстрактний клас Command;

– RedoCommand - клас команди повторення, що реалізує абстрактний клас Command;

– FilesManager - клас, що містить інформацію про директорії збереження метаданих та самих даних. Надає функціонал по запису-зчитуванні метаданих та даних;

– CommandsManager - клас, що містить набір усіх команд, які може виконати користувач над текстом, та надає зручний доступ до встановлення параметрів для них та їх виклику за допомогою stack пар типу текст-команда та відповідних методів;

– Program - клас, який представляє безпосередньо програму. Відповідає за рендерінг інтерфейсу, містить об’єкт CommandsManager та Editor й відповідає за коректне використання їх функціоналу.

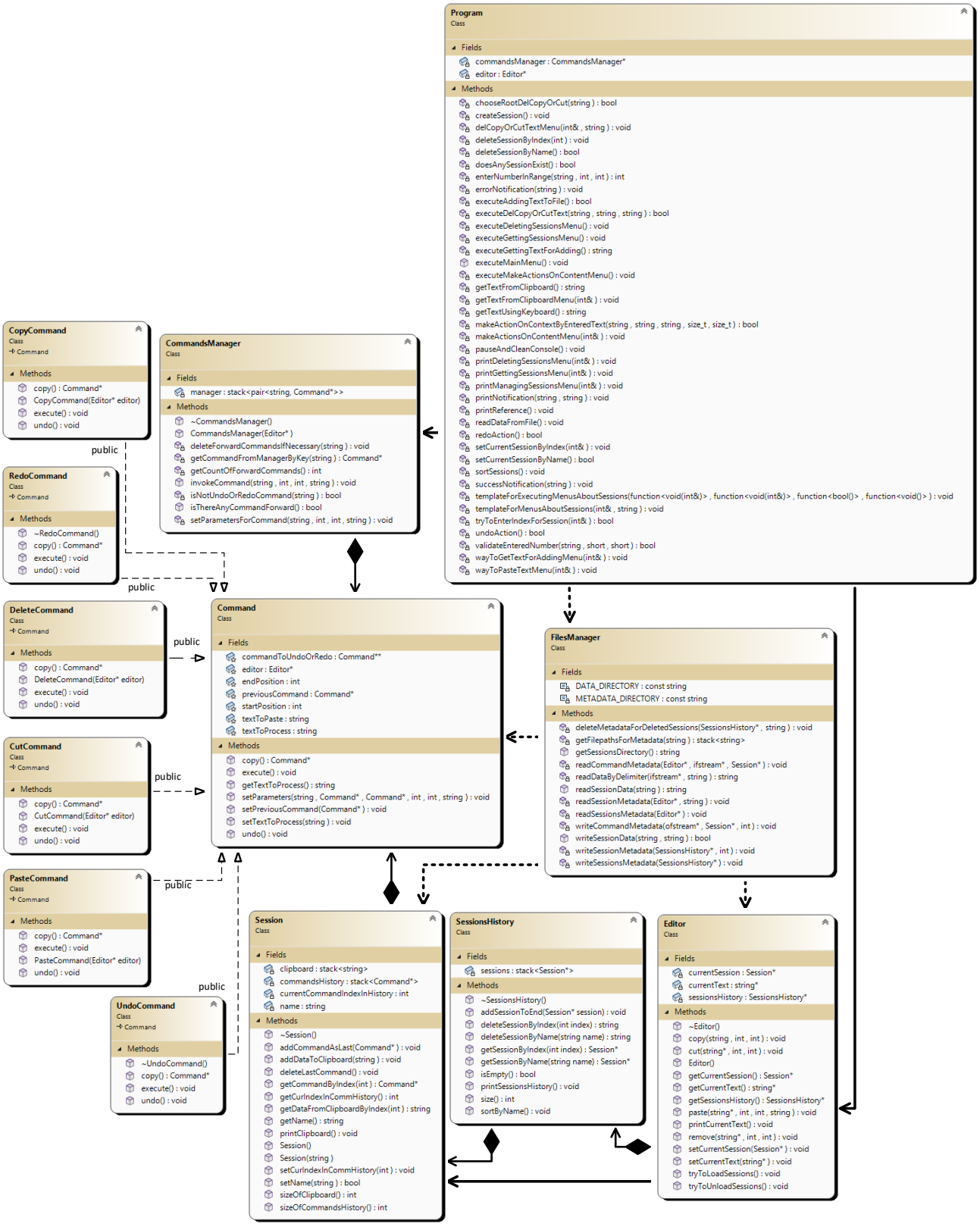

Рисунок 2.8 - Діаграма класів Session, SessionsHistory, Editor, Command, CopyCommand, PasteCommand, CutCommand, DeleteCommand, UndoCommand, RedoCommand, FilesManager, CommandsManager, Program

Структура визначення класів складається з властивостей/атрибутів та методів, які можуть мати рівні доступу (захисту), такі як: закриті (private), відкриті (public).

Атрибути класу Session:

- std::stack<Command*> commandsHistory - історія команд;
- std::stack<std::string> clipboard - буфер обміну;
- int currentCommandIndexInHistory - індекс на команді, на якій знаходиться користувач, бо, можливо, він скасував декілька команд або повторив,і це потрібно відслідковувати;
- std::string name - ім'я сеансу.

Методи класу Session:

- Session() - конструктор без параметрів;
- Session(std::string filename) - конструктор з параметрами;
- ~Session() - деструктор;
- void addCommandAsLast(Command* command) - додавання команди в історію команд;
- void addDataToClipboard(std::string data) - додавання даних до буферу обміну;
- void deleteLastCommand() - видалити останню команду з історії команд;
- int sizeOfCommandsHistory() - отримати розмір історії команд;
- int sizeOfClipboard() - отримати розмір буферу обміну;
- bool setName(std::string filename) - встановити атрибут name;
- void setCurIndexInCommHistory(int currentCommandIndexInHistory) - встановити атрибут currentCommandIndexInHistory;
- std::string getName() - отримати атрибут name;
- Command* getCommandByIndex(int index) - отримати команду з історії команд за індексом;
- int getCurIndexInCommHistory() - отримати атрибут currentCommandIndexInHistory;

– std::string getDataFromClipboardByIndex(int index) - отримати дані з буфе-ру обміну за індексом;

– void printClipboard() - вивести увесь буфер обміну.

Блок-схеми методів класу Session наведені у додатку А.

Атрибути класу SessionsHistory:

– std::stack<Session*> sessions - історія сеансів.

Методи класу SessionsHistory:

– ~SessionsHistory() - деструктор;

– void addSessionToEnd(Session* session) - додати сеанс в кінець стеку;

– Session* getSessionByIndex(int index) - отримати сеанс за індексом;

– Session* getSessionByName(std::string name) - отримати сеанс за іменем;

– std::string deleteSessionByIndex(int index) - видалити сеанс за індексом;

– std::string deleteSessionByName(std::string name) - видалити сеанс за іменем;

– bool isEmpty() - перевірка на те, чи історія сеансів пуста;

– int size() - отримати розмір історія сеансів;

– void printSessionsHistory() - вивести історію сеансів, а саме - імена;

– void sortByName() - відсортувати історію сеансів за іменем.

Блок-схеми методів класу SessionsHistory наведені у додатку А.

Атрибути класу Editor:

– static SessionsHistory* sessionsHistory - історія сеансів;

– static Session* currentSession - сеанс, з яким користувач працює в даний момент;

– static std::string* currentText - текст, який користувач редагує в даний мо-мент.

Методи класу Editor:

– Editor() - конструктор без параметрів;

– ~Editor() - деструктор;

– void tryToLoadSessions() - підвантаження метаданих;

– void tryToUnloadSessions() - відвантаження метаданих;

– void copy(std::string textToProcess, int startPosition, int endPosition) - копіювання частини текста;

– void paste(std::string* textToProcess, int startPosition, int endPosition, std::string textToPaste) - вставка текста textToPaste в текст файлу;

– void cut(std::string* textToProcess, int startPosition, int endPosition) - вирізання частини тексту;

– void remove(std::string* textToProcess, int startPosition, int endPosition) - видалення частини тексту;

– static Session* getCurrentSession() - отримання атрибуту currentSession;

– static std::string* getCurrentText() - отримання атрибуту currentText;

– static SessionsHistory* getSessionsHistory() - отримання атрибуту sessionsHistory;

– static void setCurrentSession(Session* session) - встановлення атрибуту currentSession;

– static void setCurrentText(std::string* text) - встановлення атрибуту currentText;

– static void printCurrentText() - вивід тексту сеансу, в якому в даний момент перебуває користувач.

Атрибути класу Command:

– Editor* editor - редактор;

– int startPosition, endPosition - початкова та кінцева позиції для вставки, заміни, видалення, копіювання, вирізання;

– std::string textToProcess, textToPaste - поля для тексту, який обробляємо, і для тексту, який вставляємо;

– Command* previousCommand, * commandToUndoOrRedo - вказівник на попередню команду в історії команда відносно цієї команди й вказівник на команду, яку збираємось скасувати або повторити.

Методи класу Command:

- virtual void execute() - віртуальна функція, роль якої - виконання команди;
- virtual void undo() - віртуальна функція, роль якої - скасування команди;
- virtual Command* copy() - віртуальна функція, роль якої - створення копії об’єкту команди;
- void setParameters(std::string typeOfCommand, Command* previousCommand, Command* commandToUndoOrRedo, int startPosition, int endPosition, std::string textToPaste) - встановлення параметрів для команди;
- std::string getTextToProcess() - отримання атрибуту textToProcess;
- void setTextToProcess(std::string textToProcess) - встановлення атрибуту textToProcess;
- void setPreviousCommand(Command* previousCommand) - встановлення атрибуту previousCommand.

Методи класу CopyCommand:

- CopyCommand(Editor* editor) - конструктор з параметрами;
- void execute() override - перевантажена функція виконання команди;
- void undo() override - перевантажена функція скасування команди;
- Command* copy() override - перевантажена функція створення копії об’єкту команди.

Методи класу PasteCommand:

- PasteCommand(Editor* editor) - конструктор з параметрами;
- void execute() override - перевантажена функція виконання команди;
- void undo() override - перевантажена функція скасування команди;
- Command* copy() override - перевантажена функція створення копії об’єкту команди.

Методи класу CutCommand:

- CutCommand(Editor* editor) - конструктор з параметрами;
- void execute() override - перевантажена функція виконання команди;
- void undo() override - перевантажена функція скасування команди;
- Command* copy() override - перевантажена функція створення копії

об’єкту команди.

Методи класу DeleteCommand:

- DeleteCommand(Editor* editor) - конструктор з параметрами;
- void execute() override - перевантажена функція виконання команди;
- void undo() override - перевантажена функція скасування команди;
- Command* copy() override - перевантажена функція створення копії об’єкту команди.

Методи класу UndoCommand:

- ~UndoCommand() - деструктор;
- void execute() override - перевантажена функція виконання команди;
- void undo() override - перевантажена функція скасування команди;
- Command* copy() override - перевантажена функція створення копії об’єкту команди.

Методи класу RedoCommand:

- ~RedoCommand() - деструктор;
- void execute() override - перевантажена функція виконання команди;
- void undo() override - перевантажена функція скасування команди;
- Command* copy() override - перевантажена функція створення копії об’єкту команди.

Атрибути класу FilesManager:

- static const std::string METADATA_DIRECTORY = "Metadata\\" - шлях директорії для збереження метаданих;
- static const std::string DATA_DIRECTORY = "Data\\" - шлях директорії для збереження даних.

Методи класу FilesManager:

- static std::stack<std::string> getFilepathsForMetadata (std::string directory) - отримання шляхів до файлів у вигляді стек-змінній з певної директорії;
- static std::string readDataByDelimiter (std::ifstream* ifs_session, std::string delimiter) - зчитати дані з файлу послідовно від розділювача до розділювача;

– static void deleteMetadataForDeletedSessions (SessionsHistory* sessionsHistory, std::string directory) - видалити метадані для сеансів, яких вже не існує;

– static void writeSessionsMetadata (SessionsHistory* sessionsHistory) - записати метадані усіх сеансів;

– static void writeSessionMetadata (SessionsHistory* sessionsHistory, int index) - записати метадані певного сеансу;

– static void writeCommandMetadata (std::ofstream* ofs_session, Session* session, int index) - записати метадані певної команди;

– static void readSessionsMetadata(Editor* editor) - зчитати метадані усіх доступних сеансів;

– static void readSessionMetadata(Editor* editor, std::string filepath) - зчитати метадані певного сеансу;

– static void readCommandMetadata(Editor* editor, std::ifstream* ifs_session, Session* session) - зчитати метадані певної команди;

– static std::string getSessionsDirectory() - отримати атрибут DATA_DIRECTORY;

– static std::string readSessionData(std::string fullFilepath) - зчитати текстові дані з певного файлу;

– static bool writeSessionData (std::string filename, std::string newData) - записати текстові дані в певний файл.

Атрибути класу CommandsManager:

– std::stack<std::pair<std::string, Command*>> manager - контейнер усіх команд, дозволяє зручно їми керувати за допомогою поліморфізму.

Методи класу CommandsManager:

– CommandsManager(Editor* editor) - конструктор без параметрів;

– ~CommandsManager() - деструктор;

– Command* getCommandFromManagerByKey(std::string typeOfCommand) - отримати команду з manager за допомогою текстового ключа;

– bool isNotUndoOrRedoCommand (std::string typeOfCommand) - перевірити,

чи тип команди текстом не є типом команди Скасувати або Повторити;

- void setParametersForCommand (std::string typeOfCommand, int startPosition, int endPosition, std::string textToPaste) - встановити параметри для команди;
- int getCountOfForwardCommands() - отримати кількість команд, які залишились спереду, якщо користувач скасував певну кількість команд;
- void deleteForwardCommandsIfNecessary (std::string typeOfCommand) - видалити команди, які залишились спереду, якщо користувач скасував певну кількість команд. Корисно у випадку, якщо користувач скасував кілька команд й виконав нову команду;
- bool isThereAnyCommandForward() - перевірити, чи залишились команди спереду, якщо користувач скасував певну кількість команд;
- void invokeCommand(std::string typeOfCommand, int startPosition = 0, int endPosition = 0, std::string textToPaste = "") - викликати команду певного типу.

Атрибути класу Program:

- Editor* editor - редактор;
- CommandsManager* commandsManager - менеджер команд, за допомогою якого й викликаються усі команди.

Методи класу Program:

- bool validateEnteredNumber (std::string option, short firstOption, short lastOption) - валідація введеного користувачем числа в певному діапазоні;
- int enterNumberInRange (std::string message, int firstOption, int lastOption) - організація логіки введення числа користувачем в певному діапазоні;
- void readDataFromFile() - організація логіки зчитування даних з файлу та присвоєння тексту атрибутові currentText редактора;
- void pauseAndCleanConsole() - зупинити консоль, а після натиснення будь-якої клавіші - очистити;
- void printNotification(std::string type, std::string msg) - вивід певного типу повідомлення;
- void successNotification(std::string msg) - вивід успішного повідомлення;

- void errorNotification(std::string msg) - вивід повідомлення про помилку;
- void printReference() - вивести довідку стосовно програми;
- void templateForMenusAboutSessions(int& choice, std::string action) - шаблон для підменю роботи із сеансами;
- void templateForExecutingMenusAboutSessions (std::function<void(int&)> mainFunc, std::function<void(int&)> menu, std::function<bool()> actionFuncByName, std::function<void()> additionalFunc = nullptr) - шаблон організації логіки роботи деяких підменю роботи із сеансами;
- std::string getTextUsingKeyboard() - отримати текст для додавання в файл через ввід з клавіатури;
- std::string getTextFromClipboard() - отримати текст для додавання в файл з буферу обміну;
- bool makeActionOnContextByEnteredText (std::string typeOfCommand, std::string actionInPast, std::string textToPaste = "", size_t startIndex = -2, size_t endIndex = -2) - організація виконання логіки якоїсь дії над текстом з файлу;
- bool undoAction() - організація виконання логіки скасування дії;
- bool redoAction() - організація виконання логіки повторення дії;
- void sortSessions() - організація виконання логіки сортування сеансів;
- void wayToGetTextForAddingMenu(int& choice) - вивід меню способів отримання текстів для додавання в файл;
- void getTextFromClipboardMenu(int& choice) - вивід меню отримання тексту з буферу обміну;
- void wayToPasteTextMenu(int& choice) - вивід меню способів вставки тексту в файл;
- void delCopyOrCutTextMenu(int& choice, std::string action) - вивід меню того, над яким обсягом тексту файлу ви хочете зробити дію видалення, копіювання або вирізання;
- void makeActionsOnContentMenu(int& choice) - вивід меню можливих дій над текстом файлу;

– void printGettingSessionsMenu(int& choice) - вивід меню для входу в певний сеанс;

– void printDeletingSessionsMenu(int& choice) - вивід меню для видалення певного сеансу;

– void printManagingSessionsMenu(int& choice) - вивід меню можливих дій над сеансами;

– std::string executeGettingTextForAdding() - організація виконання логіки пунктів меню способів отримання тексту для вставки;

– bool executeAddingTextToFile() - організація виконання логіки вставки тексту в файл;

– void executeMakeActionsOnContentMenu() - організація виконання логіки виконання дій над текстом файлу;

– void executeDeletingSessionsMenu() - організація виконання логіки видалення сеансів;

– void executeGettingSessionsMenu() - організація виконання логіки входу в певний сеанс програми;

– bool executeDelCopyOrCutText(std::string typeOfCommand, std::string actionForMenu, std::string actionInPast) - організація виконання логіки видалення, копіювання або вирізання тексту файлу;

– bool chooseRootDelCopyOrCut(std::string action) - організація виконання логіки вибору дії видалення, копіювання або вирізання в залежності від вибору користувача;

– bool tryToEnterIndexForSession(int& index) - організація виконання логіки вводу індексу для входу в конкретний сеанс;

– bool doesAnySessionExist() - перевірити, чи існують в даний момент в програмі якісь сеанси;

– void createSession() - організація виконання логіки створення сеансу;

– void setCurrentSessionByIndex(int& index) - організація виконання логіки входу в конкретний сеанс за індексом;

– bool setCurrentSessionByName() - організація виконання логіки входу в конкретний сеанс за іменем;

– void deleteSessionByIndex(int index = -1) - організація виконання логіки видалення конкретного сеансу за індексом;

– bool deleteSessionByName() - організація виконання логіки видалення конкретного сеансу за іменем;

– void executeMainMenu() - організація виконання логіки меню дій над сеансами.

Для тестування можливостей розробленої ієрархії класів складена програма, яка має виклики методів обробки контейнеру екземплярів класу Command, контейнеру екземплярів класу Session та меню для виконання для їх виклику. Схема меню тестової програми наведена на рисунку 2.9.

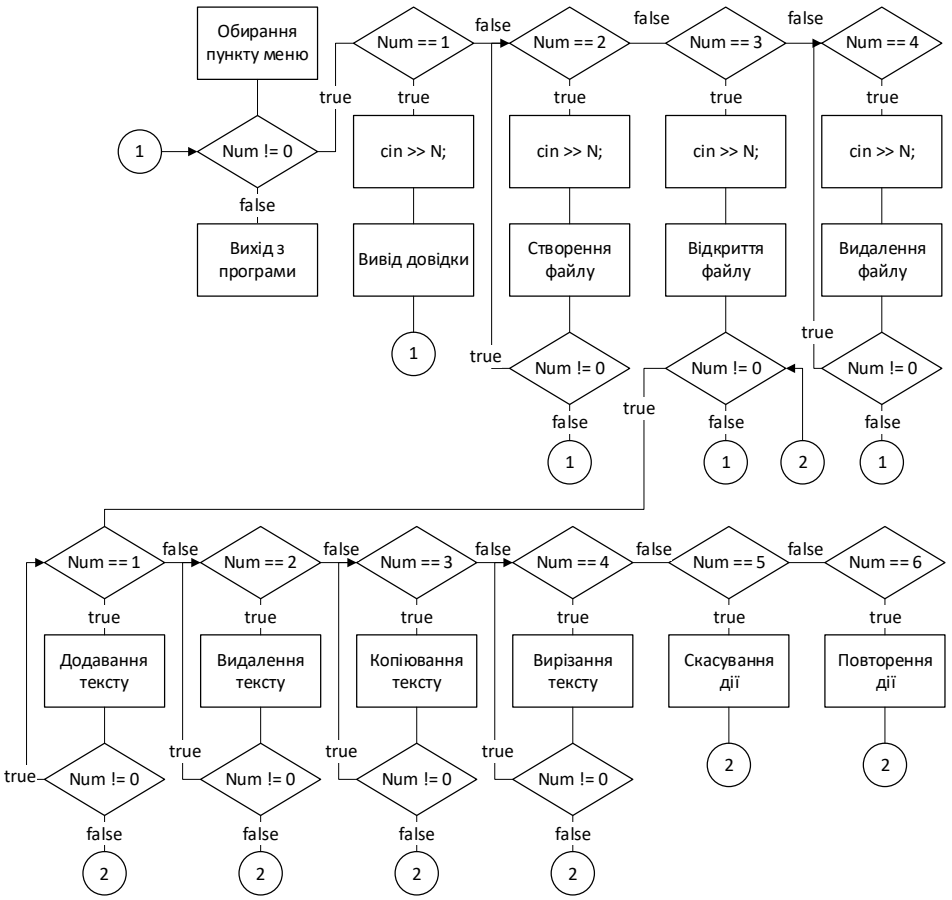

Рисунок 2.9 - Функціональна схема меню програми

2.4 Опис фізичної структури програми

Відповідно постановки задачі, діаграми класів та логічної структури розробленого головно меню програми було складено визначення методів класів: Session, SessionsHistory, Editor, Command, CopyCommand, PasteCommand, CutCommand, DeleteCommand, UndoCommand, RedoCommand, FilesManager, CommandsManager, Program і тестова програма можливостей класів Session і SessionsHistory.

Скомпільований файл тестування визначених класів знаходиться у файлі Program.exe.

Код програми складений на мові програмування C++ та складається з визначення абстрактних типів даних та функцій-членів класу. Для повноцінного функціонування методів класів та тестової програми необхідно підключення допоміжних бібліотечних файлів за допомогою директиви препроцесора #include, тобто бібліотек визначення типів, констант, вбудованих функцій і тд, які забезпечують:

- <iostream> - функції вводу-виводу мови C;
- <fstream> - інструменти файлового вводу-виводу мови C++;
- <filesystem> - надає засоби для роботи з файловою системою, включаючи створення, копіювання, переміщення та видалення файлів і каталогів, а також отримання інформації про них;
- <stack> - забезпечує методами обробки контейнеру типу stack STL;
- <functional> - надає засоби для створення та маніпулювання функціями, функціональними об'єктами та обгортками функцій;
- <windows.h> - забезпечує програмі функціонал який надається операційною средою.

Відповідно до списку атрибутів, методів класів, опис їх призначення наведених у підрозділі 2.3 складаємо їх визначення за алгоритмом виконання (Додаток А). Програмний код методів та функцій представлений у додатку Б.

2.4.1 Опис методів класу Session

Опис параметрів та змінних конструкторів та деструктора наведений у таблиці 2.1.

Таблиця 2.1 - Параметри та змінні конструкторів та деструктора класу Session

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
Session()	Параметр	void	-	-
Session(std::string filename)	Параметр	string	filename	Назва файлу/сеансу
~Session()	Параметр	void	-	-

У методах встановлення (set) значень даних членів класу Session, параметр метода привласнюється відповідній властивості (даному члену) класу. Опис методів setX представлений у таблиці 2.2.

Таблиця 2.2 - Параметри та змінні методів setX класу Session

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
void setCurIndexInCommHistory(int currentCommandIndexInHistory)	Параметр	int	currentCommandIndexInHistory	Індекс команди, на якій зупинився користувач, в історії команд
bool setName(std::string filename)	Параметр	string	filename	Назва файлу/сеансу
	Змінна	string	forbiddenCharacters	Заборонені символи в імені файлу
	Змінна	string	ch	Символьний ітератор

Методи отримання значення властивостей (get) класу Session, за допомогою оператору return, як результат роботи повертають значення відповідної властивості класу. Опис методів getX представлений у таблиці 2.3.

Таблиця 2.3 - Параметри та змінні методів getX класу Session

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
std::string getName()	Змінна	void	name	Назва файлу/сеансу
Command* getCommandByIndex(int index)	Параметр	int	index	Індекс команди
	Змінна	Command*	commandsHistory._Get_container()[index]	Вказівник на команду
int getCurIndexInComHistory()	Змінна	void	currentCommandIndexInHistory	Індекс команди, на якій зупинився користувач, в історії команд
std::string getDataFromClipboardByIndex(int index)	Параметр	int	index	Індекс даних в буфері обміну
	Змінна	string	clipboard._Get_container()[index]	Дані з буферу обміну

На основі розробленого класу Command складаємо клас Session по обробці масиву команд за допомогою контейнерного класу stack.

Опис методів класу Session представлений у таблиці 2.4.

Таблиця 2.4 - Параметри та змінні методів класу Session

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
void addCommandAsLast(Command* command)	Параметр	Command*	command	Виконана команда
void Clipboard(std::string data)	Параметр	string	data	Скопійовані дані
void deleteLastCommand()	Параметр	void	-	-
int sizeOfCommandsHistory()	Змінна	int	commandsHistory.size()	Розмір історії команд
int sizeOfClipboard()	Змінна	int	clipboard.size()	Розмір буферу обміну
printClipboard()	Змінна	int	i	Лічильник в циклі

2.4.2 Опис методів класу SessionsHistory

Опис параметрів та змінних деструктора наведений у таблиці 2.5.

Таблиця 2.5 - Параметри та змінні деструктора класу SessionsHistory

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
~SessionsHistory ()	Параметр	void	-	-

Методи отримання значення властивостей (get) класу SessionsHistory, за допомогою оператору return, як результат роботи повертають значення відповідної властивості класу. Опис методів getX представлений у таблиці 2.6.

Таблиця 2.6 - Параметри та змінні методів getX класу SessionsHistory

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
Session* getSessionByIndex(int index)	Параметр	int	index	Індекс сеансу
	Змінна	Session*	sessions._Get_container()[index]	Вказівник на сеанс
Session* sion- on- ByName(std::string name)	Параметр	string	name	Назва файлу/сеансу
	Змінна	deque<Session*>::const_iterator	sessionsIter	Ітератор, який вказує на знайдений за іменем сеанс, якщо такий існував
	Змінна	Session*	session	Вказівник на сеанс, параметр в lambda-функції, необхідний для пошуку сеансу за іменем

На основі розробленого класу Session складаємо клас SessionsHistory по обробці масиву команд за допомогою контейнерного класу stack.

Опис методів класу SessionsHistory представлений у таблиці 2.7.

Таблиця 2.7 - Параметри та змінні методів класу SessionsHistory

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
void addSessionToEnd(Session* session)	Параметр	Session*	session	Сеанс, який додаємо в історію
std::string deleteSessionByIndex(int index)	Параметр	int	index	Індекс сеансу
	Змінна	deque<Session*>*	ptrOnUnderlyingContainer	Вказівник на контейнер, на основі якого був створений стек
	Змінна	Session*	ptrOnRetiringSession	Вказівник на сеанс, який хочемо видалити
	Змінна	string	filename	Назва файлу/сеансу
	Змінна	deque<Session*>	notRetiringElements	Додатковий контейнер, необхідний для зберігання сеансів, які не видаляємо
std::string deleteSessionByName(std::string name)	Параметр	string	name	Назва файлу/сеансу
	Змінна	Session*	sessionToDelete	Вказівник на сеанс, який хочемо видалити
	Змінна	Session*	topSession	Вказівник на верхній елемент стеку сеансів
	Змінна	string	filename	Назва файлу/сеансу
	Змінна	stack<Session*>	notRetiringElements	Додатковий стек, необхідний для зберігання сеансів, які не видаляємо
bool isEmpty()	Змінна	bool	sessions.size() == 0	Булеве значення того, чи дорівнює розмір історії сеансів 0
int size()	Змінна	int	sessions.size()	Розмір історії сеансів
void printSessionsHistory()	Змінна	int	i	Лічильник циклу
void sortByName()	Змінна	deque<Session*>	tempSessions	Додатковий контейнер, необхідний для зберігання сеансів
	Змінна	Session*	session	Ітератор в циклі на елементи змінної tempSessions

2.4.3 Опис методів класу Editor

Опис параметрів та змінних конструкторів та деструктора наведений у таблиці 2.8.

Таблиця 2.8 - Параметри та змінні конструкторів та деструктора класу Editor

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
Editor()	Параметр	void	-	-
~Editor()	Параметр	void	-	-

У методах встановлення (set) значень даних членів класу Editor, параметр метода привласнюється відповідній властивості (даному члену) класу. Опис методів setX представлений у таблиці 2.9.

Таблиця 2.9 - Параметри та змінні методів setX класу Editor

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
static void setCurrentSession(Session* session)	Параметр	Session*	session	Сеанс, в якому користувач в даний момент
static void setCurrentText(std::string * text)	Параметр	string*	text	Вказівник на текст, з яким користувач працює в даний момент

Методи отримання значення властивостей (get) класу Editor, за допомогою оператору return, як результат роботи повертають значення відповідної властивості класу. Опис методів getX представлений у таблиці 2.10.

Таблиця 2.10 - Параметри та змінні методів getX класу Editor

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
static Session* getCurrentSession()	Змінна	void	currentSession	Сеанс, в якому користувач в даний момент
static SessionsHistory* getSessionsHistory()	Змінна	void	sessionsHistory	Історія сеансів
static std::string* getCurrentText()	Змінна	void	currentText	Вказівник на текст, з яким користувач працює в даний момент

Опис методів класу Editor представлений у таблиці 2.11.

Таблиця 2.11 - Параметри та змінні методів класу Editor

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
void tryToLoadSessions()	Параметр	void	-	-
void tryToUnloadSessions()	Параметр	void	-	-
void copy(std::string textToProcess, int startPosition, int endPosition)	Параметр	string	textToProcess	Текст, над яким потрібно виконати команду
	Параметр	int	startPosition	Стартова позиція, з якої потрібно виконувати команду
	Параметр	int	endPosition	Кінцева позиція, до якої потрібно виконувати команду
	Змінна	string	dataToCopy	Дані, які потрібно помістити в буфер обміну
void paste(std::string* textToProcess, int startPosition, int endPosition, std::string textToPaste)	Параметр	string	textToProcess	Текст, над яким потрібно виконати команду
	Параметр	int	startPosition	Стартова позиція, з якої потрібно виконувати команду
	Параметр	int	endPosition	Кінцева позиція, до якої потрібно виконувати команду
	Параметр	string	textToPaste	Текст, який необхідно вставити
void cut(std::string* textToProcess, int startPosition, int endPosition)	Параметр	string	textToProcess	Текст, над яким потрібно виконати команду
	Параметр	int	startPosition	Стартова позиція, з якої потрібно виконувати команду
	Параметр	int	endPosition	Кінцева позиція, до якої потрібно виконувати команду
void remove(std::string* textToProcess, int startPosition, int endPosition)	Параметр	string	textToProcess	Текст, над яким потрібно виконати команду
	Параметр	int	startPosition	Стартова позиція, з якої потрібно виконувати команду
	Параметр	int	endPosition	Кінцева позиція, до якої потрібно виконувати команду
static void printCurrentText()	Параметр	void	-	-

2.4.4 Опис методів абстрактного класу Command

У методах встановлення (set) значень даних членів класу Command, параметр метода привласнюється відповідній властивості (даному члену) класу. Опис методів setX представлений у таблиці 2.12.

Таблиця 2.12 - Параметри та змінні методів setX класу Command

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
void setTextToProcess(std::string textToProcess)	Параметр	string	textToProcess	Текст, над яким потрібно виконати команду
void setPreviousCommand(Command* previousCommand)	Параметр	Command*	previousCommand	Вказівник на минулу команду відносно в історії команд відносно об’єкту-нащадка цього класу

Методи отримання значення властивостей (get) класу Command, за допомогою оператору return, як результат роботи повертають значення відповідної властивості класу. Опис методів getX представлений у таблиці 2.13.

Таблиця 2.13 - Параметри та змінні методів getX класу Command

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
std::string getTextToProcess()	Змінна	void	textToProcess	Текст, над яким потрібно виконати команду

Опис методів класу Command представлений у таблиці 2.14.

Таблиця 2.14 - Параметри та змінні методів класу Command

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
void setParameters(std::string typeOfCommand, Command* previousCommand, Command* commandToUndoOrRedo, int startPosition, int endPosition, std::string textToPaste)	Параметр	string	typeOfCommand	Тип команди
	Параметр	Command*	previousCommand	Вказівник на минулу команду відносно в історії команд відносно об’єкту-нащадка цього класу
	Параметр	Command*	commandToUndoOrRedo	Команда, яку потрібно скасувати або повторити
	Параметр	int	startPosition	Стартова позиція, з якої потрібно виконувати команду
	Параметр	int	endPosition	Кінцева позиція, до якої потрібно виконувати команду
	Параметр	string	text	Текст, над яким потрібно виконати команду

Також цей клас використовує віртуальні функції:

virtual void execute() = 0; - для виконання команди.

virtual void undo() = 0; - для скасування команди.

virtual Command* copy() = 0; - для створення копії об’єкту команди.

2.4.5 Опис методів класу CopyCommand

Опис параметрів та змінних конструкторів наведений у таблиці 2.15.

Таблиця 2.15 - Параметри та змінні конструкторів класу CopyCommand

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
CopyCommand(Editor* editor)	Параметр	Editor*	editor	Текстовий редактор

Опис методів класу CopyCommand представлений у таблиці 2.16.

Таблиця 2.16 - Параметри та змінні методів класу CopyCommand

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
void execute() override	Параметр	void	-	-
void undo() override	Параметр	void	-	-
Command* copy() override	Параметр	void	-	-

2.4.6 Опис методів класу PasteCommand

Опис параметрів та змінних конструкторів наведений у таблиці 2.17.

Таблиця 2.17 - Параметри та змінні конструкторів класу PasteCommand

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
PasteCommand(Editor* editor)	Параметр	Editor*	editor	Текстовий редактор

Опис методів класу PasteCommand представлений у таблиці 2.18.

Таблиця 2.18 - Параметри та змінні методів класу PasteCommand

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
void execute() override	Параметр	void	-	-
void undo() override	Параметр	void	-	-
Command* copy() override	Змінна	Command *	new PasteCommand(*this)	Копія об’єкту

2.4.7 Опис методів класу CutCommand

Опис параметрів та змінних конструкторів наведений у таблиці 2.19.

Таблиця 2.19 - Параметри та змінні конструкторів класу CutCommand

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
CutCommand(Editor* editor)	Параметр	Editor*	editor	Текстовий редактор

Опис методів класу CutCommand представлений у таблиці 2.20.

Таблиця 2.20 - Параметри та змінні методів класу CutCommand

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
void execute() override	Параметр	void	-	-
void undo() override	Параметр	void	-	-
Command* copy() override	Змінна	Command *	new CutCommand(*this)	Копія об’єкту

2.4.8 Опис методів класу DeleteCommand

Опис параметрів та змінних конструкторів наведений у таблиці 2.21.

Таблиця 2.21 - Параметри та змінні конструкторів класу DeleteCommand

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
DeleteCommand(Editor* editor)	Параметр	Editor*	editor	Текстовий редактор

Опис методів класу DeleteCommand представлений у таблиці 2.22.

Таблиця 2.22 - Параметри та змінні методів класу DeleteCommand

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
void execute() override	Параметр	void	-	-
void undo() override	Параметр	void	-	-
Command* copy() override	Змінна	Command*	new DeleteCommand(*this)	Копія об’єкту

2.4.9 Опис методів класу UndoCommand

Опис параметрів та змінних деструктора наведений у таблиці 2.23.

Таблиця 2.23 - Параметри деструктора класу UndoCommand

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
~UndoCommand()	Параметр	void	-	-

Опис методів класу UndoCommand представлений у таблиці 2.24.

Таблиця 2.24 - Параметри та змінні методів класу UndoCommand

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
void execute() override	Параметр	void	-	-
void undo() override	Параметр	void	-	-
Command* copy() override	Параметр	void	-	-

2.4.10 Опис методів класу RedoCommand

Опис параметрів та змінних деструктора наведений у таблиці 2.25.

Таблиця 2.25 - Параметри деструктора класу RedoCommand

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
~RedoCommand()	Параметр	void	-	-

Опис методів класу RedoCommand представлений у таблиці 2.26.

Таблиця 2.26 - Параметри та змінні методів класу RedoCommand

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
void execute() override	Параметр	void	-	-
void undo() override	Параметр	void	-	-
Command* copy() override	Параметр	void	-	-

2.4.11 Опис методів класу FilesManager

Методи отримання значення властивостей (get) класу FilesManager, за допомогою оператору return, як результат роботи повертають значення відповідної властивості класу. Опис методів getX представлений у таблиці 2.27.

Таблиця 2.27 - Параметри та змінні методів getX класу FilesManager

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
static std::stack<std::string> getFilepathsForMetadata(std::string directory)	Параметр	string	directory	Директорія, імена всіх файлів якої потрібно отримати
	Змінна	stack<string>	filesFromMetadataDirectory	Стек імен файлів директорії
	Змінна	filesystem::directory_iterator	iteratorOnFiles	Ітератор для перегляду вмісту директорії

Продовження таблиці 2.27

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
	Змінна	filesystem::directory_entry&	entry	Об’єкт в директорії
static std::string getSessionsDirectory()	Змінна	void	DATA_DIRECTORY	Директорія збереження безпосередньо текстових файлів

Опис методів класу FilesManager представлений у таблиці 2.28.

Таблиця 2.28 - Параметри та змінні методів класу FilesManager

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
static std::string readDataByDelimiter(std::ifstream* ifs_session, std::string delimiter)	Параметр	ifstream*	ifs_session	Вказівник на потік зчитування
	Параметр	string	delimiter	Розділювач даних
	Змінна	string	line	Один рядок
	Змінна	string	text	Увесь зчитаний з файлу текст
	Змінна	int	counterOfLines	Кількість рядків
static void deleteMetadataForDeletedSessi-si-ons(SessionsHistory* sessionsHistory, std::string directory)	Параметр	SessionsHistory*	sessionsHistory	Історія сеансів
	Параметр	string	directory	Директорія для видалення зайвих метаданих
	Змінна	stack<string>	filesFromMetadataDirectory	Стек імен файлів директорії
	Змінна	string	sessionFilepath	Шлях метаданих для певного сеансу
	Змінна	int	i	Лічильник в циклі
	Змінна	int	j	Лічильник в циклі
static void writeSessionsMetadata(SessionsHistory* sessionsHistory)	Параметр	SessionsHistory*	sessionsHistory	Історія сеансів
	Змінна	int	i	Лічильник в циклі
static void readSessionsMetadata(Editor* editor)	Параметр	Editor*	editor	Текстовий редактор
	Змінна	stack<string>	available_sessions	Доступні для зчитування сеанси

Продовження таблиці 2.28

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
	Змінна	int	i	Лічильник в циклі
static void writeSessionMetadata(SessionsHistory* sessionsHistory, int index)	Параметр	SessionsHistory*	sessionsHistory	Історія сеансів
	Параметр	int	index	Індекс сеансу
	Змінна	Session*	session	Сеанс, отриманий за індексом
	Змінна	ofstream	ofs_session	Потік запису
	Змінна	int	j	Лічильник в циклі
static void writeCommandMetadata(std::ofstream* ofs_session, Session* session, int index)	Параметр	ofstream*	ofs_session	Вказівник на потік запису
	Параметр	Session*	session	Сеанс
	Параметр	int	index	Індекс команди
	Змінна	string	typeOfCommand	Тип команди
	Змінна	string	nameOfCommandClass	Ім’я класу команди
	Змінна	string	delimiter	Розділювач даних
	Змінна	Command*	command	Команда, отримана за індексом
static void readSessionMetadata(Editor* editor, std::string filepath)	Параметр	Editor*	editor	Текстовий редактор
	Параметр	string	filepath	Шлях до сеансу
	Змінна	string	line	Один рядок
	Змінна	Session*	session	Сеанс, метадані якого зчитуються
	Змінна	int	countOfCommands	Кількість команд в сеансі
	Змінна	ifstream	ifs_session	Потік зчитування
	Змінна	int	j	Лічильник в циклі
static void readCommandMetadata(Editor* editor, std::ifstream* ifs_session, Session* session)	Параметр	Editor*	editor	Текстовий редактор
	Параметр	ifstream*	ifs_session	Вказівник на потік зчитування
	Параметр	Session*	session	Сеанс, метадані якого зчитуються
	Змінна	string	typeOfCommand	Тип команди
	Змінна	string	text	Увесь зчитаний з файлу текст

Продовження таблиці 2.28

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
	Змінна	Command *	command	Команда, метадані якої зчитують-ся
	Змінна	Command *	previousCommand	Вказівник на минулу команду в історії команд
static std::string readSessionData(std::string fullFilepath)	Параметр	string	fullFilepath	Шлях до файлу
	Змінна	string	text	Увесь зчитаний з файлу текст
	Змінна	string	line	Один рядок
	Змінна	ifstream	file	Потік зчитування
	Змінна	int	counterOfLines	Кількість рядків
static bool writeSessionData(std::string filename, std::string newData)	Параметр	string	filename	Ім’я файлу
	Параметр	string	newData	Дані для запису
	Змінна	ofstream	file	Потік запису

2.4.12 Опис методів класу CommandsManager

Опис параметрів та змінних конструкторів та деструктора наведений у таблиці 2.29.

Таблиця 2.29 - Параметри та змінні конструкторів та деструктора класу CommandsManager

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
CommandsManager (Editor* editor)	Параметр	Editor*	editor	Текстовий редактор
~CommandsManager()	Параметр	void	-	-

Методи отримання значення властивостей (get) класу CommandsManager, за допомогою оператору return, як результат роботи повертають значення відповідної властивості класу. Опис методів getX представлений у таблиці 2.30.

Таблиця 2.30 - Параметри та змінні методів getX класу CommandsManager

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
Command* get-CommandFromMan Manager- ager- ByKey(std::string typeOfCommand)	Параметр	string	typeOfCommand	Тип команди
	Змінна	int	i	Лічильник в циклі
int getCountOfForward Commands()	Змінна	int	Edi- tor::getCurrentSes sion()- >sizeOfCom- mandsHistory() - 1 - Edi- tor::getCurrentSes sion()- >getCurIndexIn- CommHistory()	Кількість команд, які залишились спереду після скасування дій

Опис методів класу CommandsManager представлений у таблиці 2.31.

Таблиця 2.31 - Параметри та змінні методів класу CommandsManager

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
bool isNotUn- doOrRedoCom- mand(std::string typeOfCommand)	Параметр	string	typeOfCommand	Тип команди
void setParameters- ForCom- mand(std::string typeOfCommand, int startPosition, int endPosition, std::string textToPaste)	Параметр	string	typeOfCommand	Тип команди
	Параметр	int	startPosition	Стартова позиція, з якої потрібно виконувати команду
	Параметр	int	endPosition	Кінцева позиція, до якої потрібно виконувати команду
	Параметр	string	textToPaste	Текст, який необхідно вставити
	Змінна	Command*	previousCommand	Минула команда в історії команд
	Змінна	Command *	commandToUndo OrRedo	Команда, яку потрібно скасувати або повторити
bool isThereAnyComma ndForward()	Параметр	void	-	-

Продовження таблиці 2.31

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
void deleteForwardCommandsIfNecessary(std::string typeOfCommand)	Параметр	string	typeOfCommand	Тип команди
	Змінна	int	countOfForwardCommands	Кількість команд, які залишились спереду після скасування дій
	Змінна	int	i	Лічильник циклу
void invokeCommand(std::string typeOfCommand, int startPosition = 0, int endPosition = 0, std::string textToPaste = "")	Параметр	string	typeOfCommand	Тип команди
	Параметр	int	startPosition	Стартова позиція, з якої потрібно виконувати команду. За замовчуванням = 0
	Параметр	int	endPosition	Кінцева позиція, до якої потрібно виконувати команду. За замовчуванням = 0
	Параметр	string	textToPaste	Текст, який необхідно вставити. За замовчуванням = “0”

2.4.13 Опис методів класу Program

Опис методів класу Program представлений у таблиці 2.32.

Таблиця 2.32 - Параметри та змінні методів класу Program

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
bool validateEnteredNumber(std::string option, short firstOption, short lastOption)	Параметр	string	option	Введений користувачем варіант
	Параметр	short	firstOption	Початок діапазону числових варіантів вводу
	Параметр	short	lastOption	Кінець діапазону числових варіантів вводу
	Змінна	char	num	Ітератор в циклі на символи option
	Змінна	unsigned long long	convertedValue	Конвертоване значення option
int enterNumberInRange(std::string message, int firstOption, int lastOption)	Параметр	string	message	Повідомлення для виводу
	Параметр	short	firstOption	Початок діапазону числових варіантів вводу
	Параметр	short	lastOption	Кінець діапазону числових варіантів вводу
	Змінна	bool	isOptionVerified	Булеве значення того, чи потрібний варіант із запропонованих ввів користувач

Продовження таблиці 2.32

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
	Змінна	string	option	Введений користувачем варіант
void readDataFromFile()	Змінна	string	filepath	Шлях до файлу
	Змінна	string	textFromFile	Дані, зчитані з файлу
void pauseAndCleanConsole()	Параметр	void	-	-
void printNotification(std::string type, std::string msg)	Параметр	string	type	Тип повідомлення: помилка чи успіх
	Параметр	string	msg	Зміст повідомлення
void successNotification(std::string msg)	Параметр	string	msg	Зміст повідомлення
void errorNotification(std::string msg)	Параметр	string	msg	Зміст повідомлення
void printReference()	Параметр	void	-	-
void templateForMenusAboutSesusAboutSessions(int& choice, std::string action)	Параметр	int&	choice	Опція меню
	Параметр	string	action	Текстом дія над сеансами, яку виконує користувач
void templateForExecutingMenusAboutSessisions(std::function<void(int&)> mainFunc, std::function<void(int&)> menu, std::function<bool()> actionFuncByName, std::function<void()> additionalFunc = nullptr)	Параметр	function<void(int&)>	mainFunc	Функціональний об’єкт основної функції дії над сеансом, саме за індексом
	Параметр	function<void(int&)>	menu	Функціональний об’єкт меню дії над сеансами
	Параметр	function<bool()>	actionFuncByName	Функціональний об’єкт дії над сеансом за іменем
	Параметр	function<void()>	additionalFunc	Функціональний об’єкт функції по виконанню меню дій над змістом файлу. За замовчуванням дорівнює nullptr
	Змінна	int	choice	Опція меню
	Змінна	int	index	Індекс обраного сеансу
	Змінна	bool	isActionSuccessfull	Булеве значення успішності виконаної дії
bool undoAction()	Параметр	void	-	-
bool redoAction()	Змінна	bool	isThereAnyCommandForward	Булеве значення того, чи є спереді команди, щоб повторити

Продовження таблиці 2.32

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
std::string getTextUsingKeyboard()	Змінна	string	line	Один рядок
	Змінна	string	text	Весь введений текст
	Змінна	int	countOfLines	Кількість рядків
std::string getTextFromClipboard()	Змінна	int	choice	Опція меню
	Змінна	int	sizeOfClipboard	Розмір буферу обміну
bool makeActionOnContext-ByEntered-Text(std::string typeOfCommand, std::string actionInPast, std::string textToPaste = "", size_t startIndex = -2, size_t endIndex = -2)	Параметр	string	typeOfCommand	Тип команди
	Параметр	string	actionInPast	Дія, яку виконує користувач, в минулому часі
	Параметр	string	textToPaste	Текст для вставки. За замовчуванням дорівнює ""
	Параметр	size_t	startIndex	Стартова позиція, з якої потрібно виконувати команду. За замовчуванням дорівнює -2
	Параметр	size_t	endIndex	Кінцева позиція, до якої потрібно виконувати команду. За замовчуванням дорівнює -2
	Змінна	string	textForAction	Введений користувачем текст, який, якщо він є в файлі, то над ним й виконується команда
void sortSessions()	Параметр	void	-	-
void wayToGetTextForAddingMenu(int& choice)	Параметр	int&	choice	Опція меню
void getTextFromClipboardMenu(int& choice)	Параметр	int&	choice	Опція меню
void wayToPasteTextMenu(int& choice)	Параметр	int&	choice	Опція меню
void delCopyOrCutTextMenu(int& choice, std::string action)	Параметр	int&	choice	Опція меню
	Параметр	string	action	Текстом команда, яку виконує користувач
void makeActionsOnContentMenu(int& choice)	Параметр	int&	choice	Опція меню

Продовження таблиці 2.32

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
void printGettingSessionsMenu(int& choice)	Параметр	int&	choice	Опція меню
void printDeletingSessionsMenu(int& choice)	Параметр	int&	choice	Опція меню
void printManagingSessionsMenu(int& choice)	Параметр	int&	choice	Опція меню
std::string executeGettingTextForAdding()	Змінна	string	textToPaste	Текст для вставки
	Змінна	int	choice	Опція меню
bool executeAddingTextToFile()	Змінна	string	textToPaste	Текст для вставки
	Змінна	int	choice	Опція меню
void executeMakeActionsOnContentMenu()	Змінна	bool	wasTextSuccessfullyChanged	Булеве значення того, чи було текст якось змінено
	Змінна	int	choice	Опція меню
void executeDeletingSessionsMenu()	Змінна	function<void(int)>	mainFunc	Функціональний об’єкт функції видалення сеансу за індексом
	Змінна	function<void(int&)>	menu	Функціональний об’єкт меню видалення сеансів
	Змінна	function<bool()>	actionFuncByName	Функціональний об’єкт функції видалення сеансу за іменем
	Змінна	int	index	Індекс сеансу в історії сеансів
	Змінна	int&	choice	Опція меню
void executeGettingSessionsMenu()	Змінна	function<void(int&)>	mainFunc	Функціональний об’єкт функції отримання сеансу за індексом
	Змінна	function<void(int&)>	menu	Функціональний об’єкт меню отримання сеансів
	Змінна	function<bool()>	actionFuncByName	Функціональний об’єкт функції отримання сеансу за іменем
	Змінна	function<void()>	additionalFunc	Функціональний об’єкт функції виклику меню дій над змістом файлу
	Змінна	int	index	Індекс сеансу в історії сеансів
	Змінна	int&	choice	Опція меню

Продовження таблиці 2.32

Заголовок, опис результату методу	Роль змінної	Тип	Позначення	Опис
bool executeDelCopyOrCutText(std::string typeOfCommand, std::string actionForMenu, std::string actionInPast)	Параметр	string	typeOfCommand	Тип команди
	Параметр	string	actionForMenu	Текстом дія, виконувана над текстом
	Параметр	string	actionInPast	Текстом дія, виконувана над текстом, в минулому часі
	Змінна	int	choice	Опція меню
	Змінна	bool	wasOperationSuccessful	Булеве значення того, чи успішно була виконана операція
bool chooseRootDelCopyOrCut(std::string action)	Параметр	string	action	Текстом дія, виконувана над текстом
bool tryToEnterIndexForSession(int& index)	Параметр	int&	index	Індекс сеансу в історії сеансів
bool doesAnySessionExist()	Параметр	void	-	-
void createSession()	Змінна	string	filename	Ім’я сеансу/файлу
	Змінна	Session*	newSession	Створюваний сеанс
	Змінна	string	filepath	Шлях до файлу
	Змінна	ofstream	file	Потік запису
void setCurrentSessionByIndex(int& index)	Параметр	int&	index	Індекс сеансу в історії сеансів
bool setCurrentSessionByName()	Змінна	string	name	Ім’я сеансу/файлу
	Змінна	Session*	session	Сеанс, який встановлюємо як той, з яким зараз працює користувач
void deleteSessionByIndex(int index = -1)	Параметр	int	index	Індекс сеансу в історії сеансів. За замовчуванням дорівнює -1
bool deleteSessionByName()	Змінна	string	name	Ім’я сеансу/файлу
	Змінна	string	filename	Повне ім’я сеансу/файлу
	Змінна	string	pathToSession	Шлях до файлу сеансу
void executeMainMenu()	Змінна	int	choice	Опція меню

У головній функції void main() оголошуємо локальні зміні (див. Додаток Б):

Program program - екземпляр класу Program

Далі викликаємо його метод executeMainMenu():

program.executeMainMenu();

В якому за допомогою операторів do…while та switch керуємо операціями над елементами контейнера, тобто меню.

```
void printManagingSessionsMenu(int& choice) {
  system("cls");
  std::cout << "Головне меню:\n";
  std::cout << "0. Закрити програму\n";
  std::cout << "1. Довідка\n";
  std::cout << "2. Створити сеанс\n";
  std::cout << "3. Відкрити сеанс\n";
  std::cout << "4. Видалити сеанс\n";
  choice = enterNumberInRange("Ваш вибір: ", 0, 4);
}
```

2.5 Тестування програми

При тестуванні програми були перевірені та випробувані всі пункти меню тестованої програми розробленої ієрархії класів, що об’єднані у контейнер типу stack. Також випробувані різні типи даних при ініціалізації атрибутів розроблених класів.

Спочатку було протестоване головне меню, наведене на рисунку 2.10.

Якщо жодних сеансів ще не було створено, а спробувати обрати пункт “3” або “4”, то буде виведено помилку (див. рис. 2.11).

Якщо при виборі пункту меню спробувати ввести текст, якісь символи або цифри, які виходять за межі діапазону цифр, які надані для пунктів меню, то буде

виведено помилку (див. рис. 2.12). Це також працює для всіх підменю, які є в програмі.

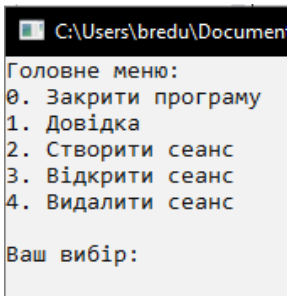

Рисунок 2.10 - Головне меню

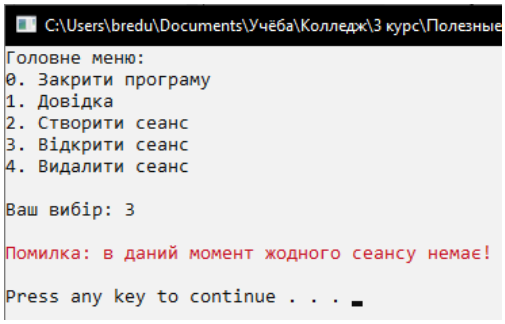

Рисунок 2.11 - Спроба відкрити або видалити сеанс при відсутності сеансів

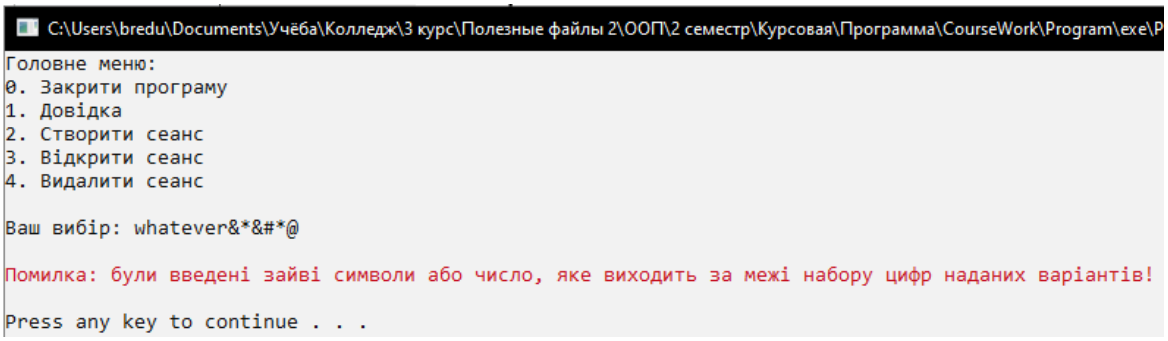

Рисунок 2.12 - Спроба ввести некоректний пункт меню

Якщо при створенні сеансу ввести в імені один із заборонених символів, буде виведено помилку (див. рис. 2.13).

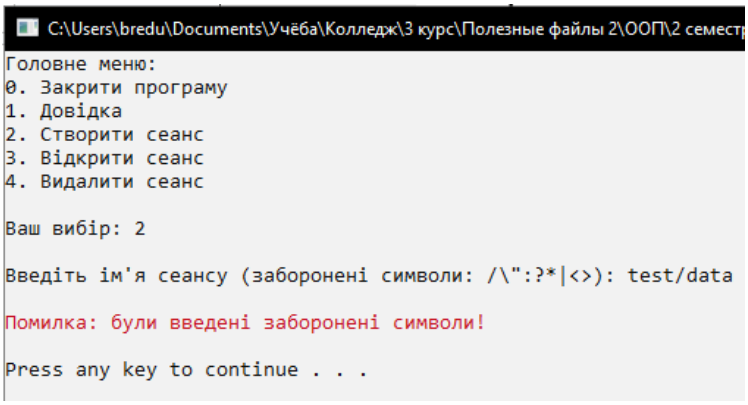

Рисунок 2.13 - Спроба ввести заборонений символ в імені сеансу

Якщо є декілька сеансів, зайти в підменю видалення сеансів й при вводі позиції сеансу, який хочемо видалити, спробувати ввести текст, якісь символи або цифри, які виходять за межі діапазону цифр, які надані для нумерації сеансів, то буде виведено помилку (див. рис. 2.14). Це працює в усіх місцях програми, де потрібно ввести позицію.

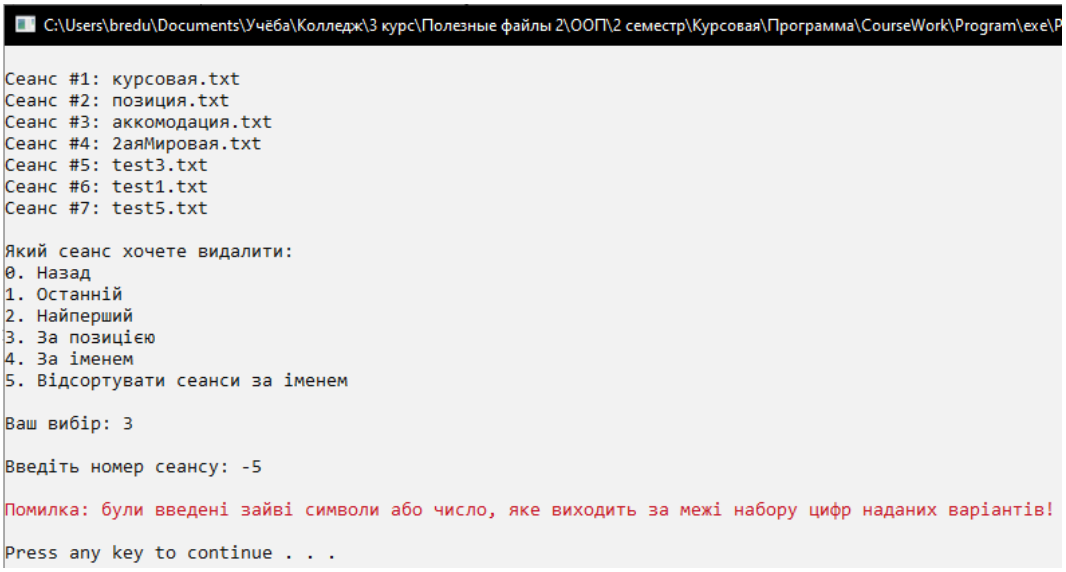

Рисунок 2.14 - Спроба ввести некоректну позицію

Якщо спробувати ввести ім’я, якого немає в списку, буде виведено помилку (див. рис. 2.15).

Рисунок 2.15 - Спроба вводу імені сеансу, якого немає в списку

Якщо видалити всі сеанси, після чого спробувати обрати якийсь пункт з підменю для видалення сеансів, то буде виведено помилку (див. рис. 2.16). Це працює для всіх пунктів підменю, якщо немає сеансів, доступних для видалення.

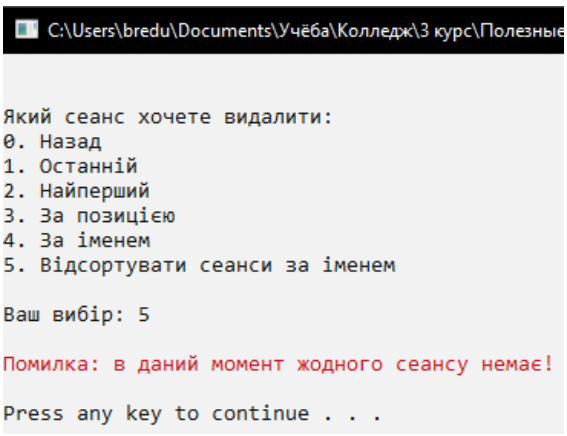

Рисунок 2.16 - Спроба обрати пункт підменю при відсутності сеансів

Відкривши файл, якщо він порожній і спробувати обрати пункти “2”, “3”, “4”, то буде виведено помилку (див. рис. 2.17).

Якщо немає дій, які можна було б скасувати, буде виведено помилку (див. рис. 2.18). Якщо немає дій, які можна було б повторити, буде виведено помилку (див. рис. 2.19).

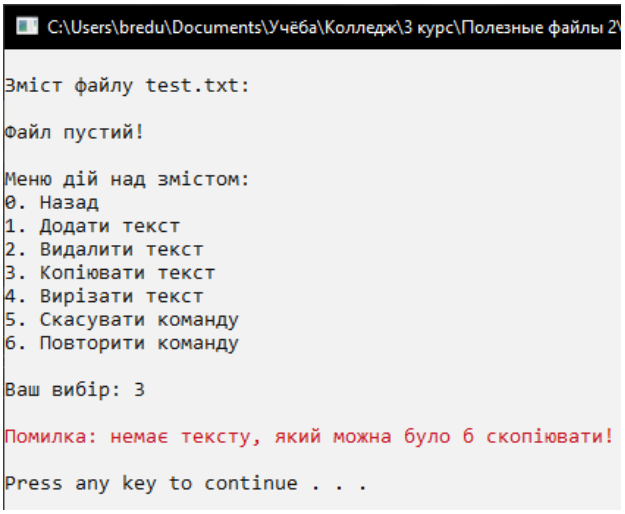

Рисунок 2.17 - Спроба зробити щось із змістом файлом при його відсутності

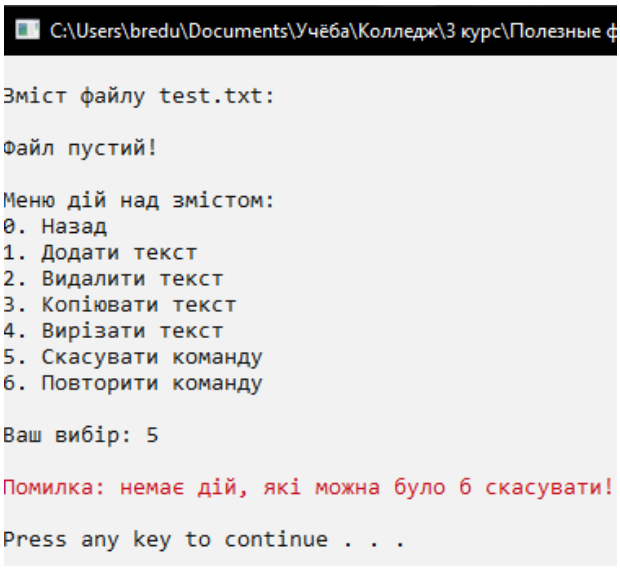

Рисунок 2.18 - Спроба скасувати дію

Якщо при додаванні тексту спробувати додати дані з буферу обміну, хоча він буде порожній, то буде виведено помилку (див. рис. 2.20).

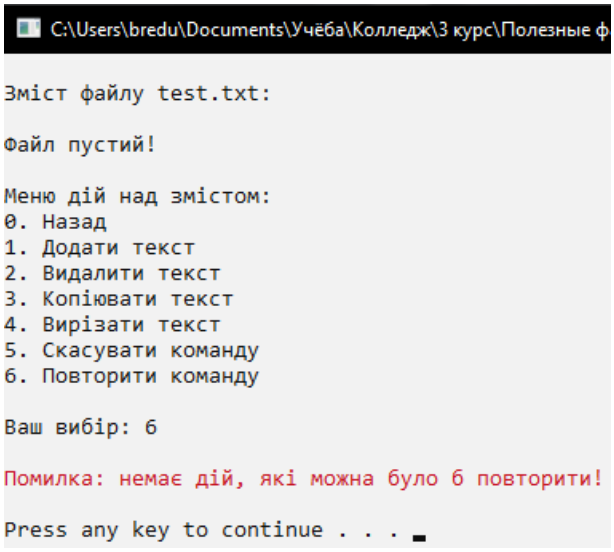

Рисунок 2.19 - Спроба повторити дію

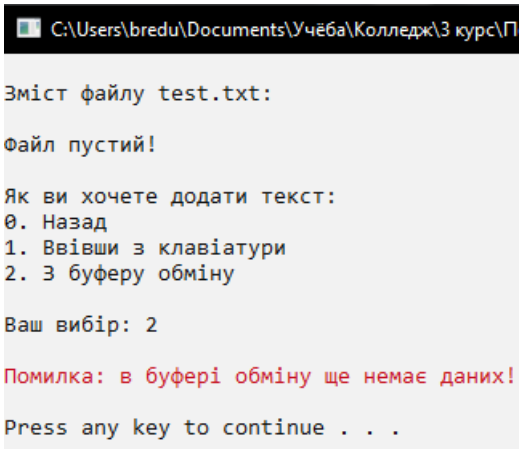

Рисунок 2.20 - Спроба додати дані з пустого буферу обміну

Якщо при додаванні тексту спробувати замінити цим текстом якусь частину з файлу, хоча файл сам буде порожній, буде виведено помилку (див. рис. 2.21).

Якщо в файлі все ж таки текст є, але при обранні пункту “3” ввести текст, якого в файлі немає, то буде виведено помилку (див. рис. 2.22). Це працює також для видалення, копіювання та вирізання, де можна ввести текст з клавіатури, який буде або не буде частиною змісту файлу і над яким потрібно буде виконати цю дію.

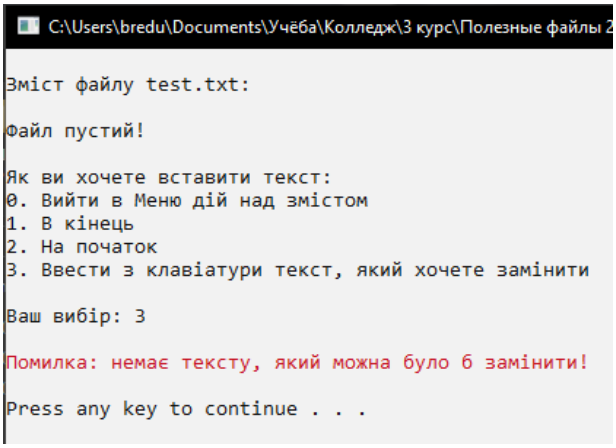

Рисунок 2.21 - Спроба замінити частину з файлу введеним текстом при тому, що файл порожній

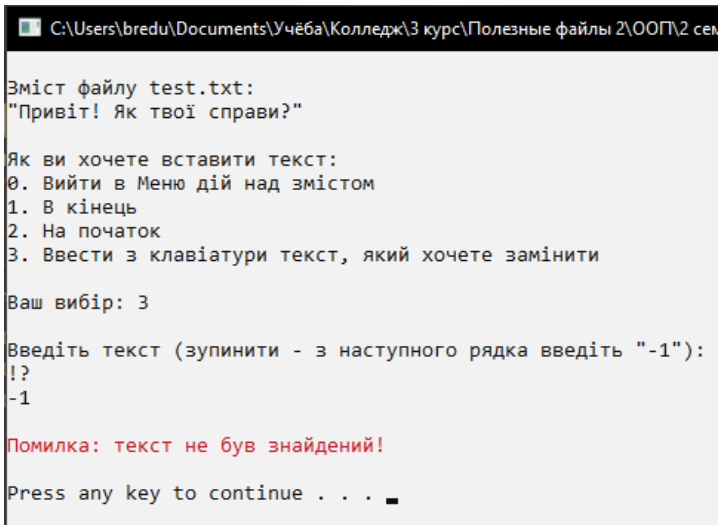

Рисунок 2.22 - Спроба введення частини тексту з файлу і виконання над нею дії при тому, що цей текст відсутній в файлі

2.6 Вимоги до складу та параметрів технічних та програмних засобів

Для повноцінної роботи програми необхідний персональний комп’ютер з 64-ох розрядною операційною системою Windows v.8.1/10/11.

Конфігурація комп’ютеру повинна бути такою:

– процесор із тактовою частотою не нижче 3.4 ГГц. Рекомендується використовувати як мінімум двоядерний процесор;

– відеоадаптер з мінімальною роздільною здатністю 720p (1280 на 720 пікселів);

– місце на жорсткому диску: до 232 ГБ (мінімум 1.4 ГБ) вільного місця, залежно від встановлених компонентів;

– 2 ГБ ОЗУ; рекомендується 8 ГБ ОЗУ (мінімум 2,5 ГБ при виконанні на віртуальній машині);

– монітор з мінімальним роздільною здатністю 720p (1280 на 720 пікселів); розмір - від 17 дюймів і більше;

– мишка з мінімальним роздільною здатністю на екрані в 800 DPI;

– будь-яка стандартна клавіатура USB або бездротова клавіатура, сумісна з операційною системою Windows.

2.7 Інструкція користувача програми

Запуск програми виконується за допомогою файлу Program.exe. Для збереження та читання даних необхідний зовнішній файл з формованими даними або порожній. При запуску програми на екрані командного вікна консольного вводу-виводу з’являється головне меню програми яке наведено на рис. 2.23.

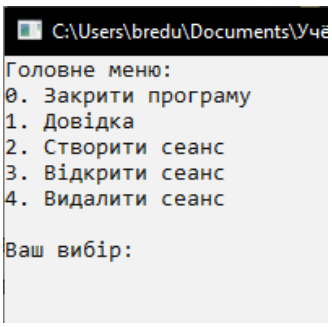

Рисунок 2.23 - Головне меню

Поняття “сеанс” в контексті програми означає деяке представлення того, що користувач відкриває та редагує певний файл. Це поняття можна вважати синонімом слова “файл”.

Завантаження метаданих в контейнер з файлу виконується автоматично після запуску програми. Завантаження ж самих даних з певного файлу здійснюється після обрання його в меню відкриття сеансів - пункт “3” головного меню - за індексом або за іменем (див. рис. 2.24). Якщо в імені файлу при обранні його за такий спосіб немає розширення “.txt”, то програма сама встановить потрібне розширення. Після обрання файлу виконується завантаження та вивід в консоль його даних (див. рис. 2.25). Якщо файл не має ніяких даних, програма про це повідомить (див. рис. 2.26).

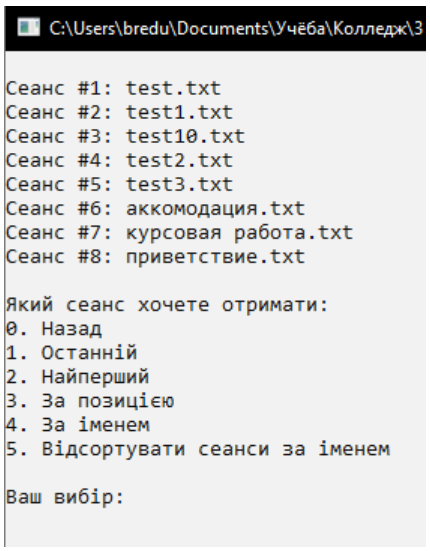

Рисунок 2.24 - Меню отримання сеансу

Для отримання довідки про програму потрібно обрати пункт “1” головного меню (див. рис. 2.27).

Для створення нового сеансу потрібно обрати пункт “2” головного меню, після чого користувач вводить ім’я файлу, яке в результаті стане й ім’ям сеансу (див. рис. 2.28).

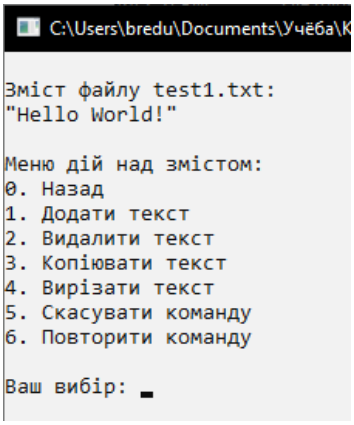

Рисунок 2.25 - Вивід даних файлу test1.txt в консоль

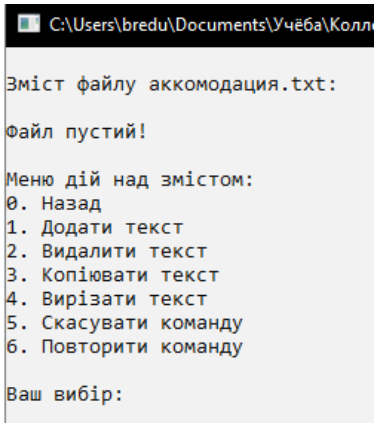

Рисунок 2.26 - Відкритий пустий файл

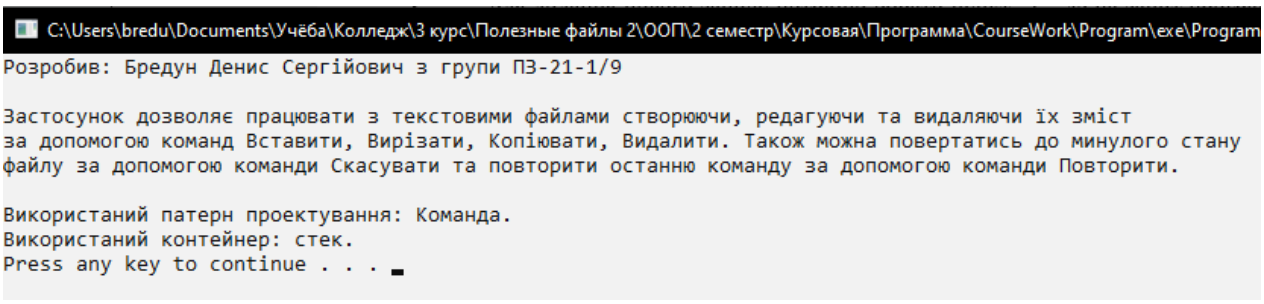

Рисунок 2.27 - Довідка про програму

Обравши пункт “4” головного меню, користувач може видалити сеанс за індексом або іменем (див. рис. 2.29).

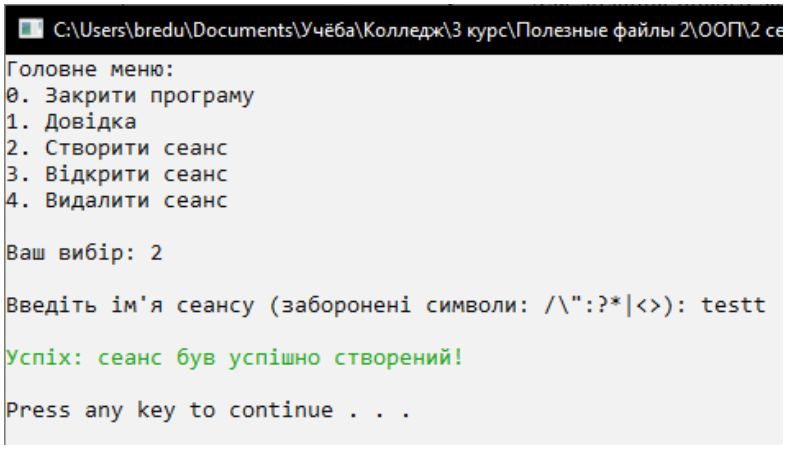

Рисунок 2.28 - Створення сеансу testt

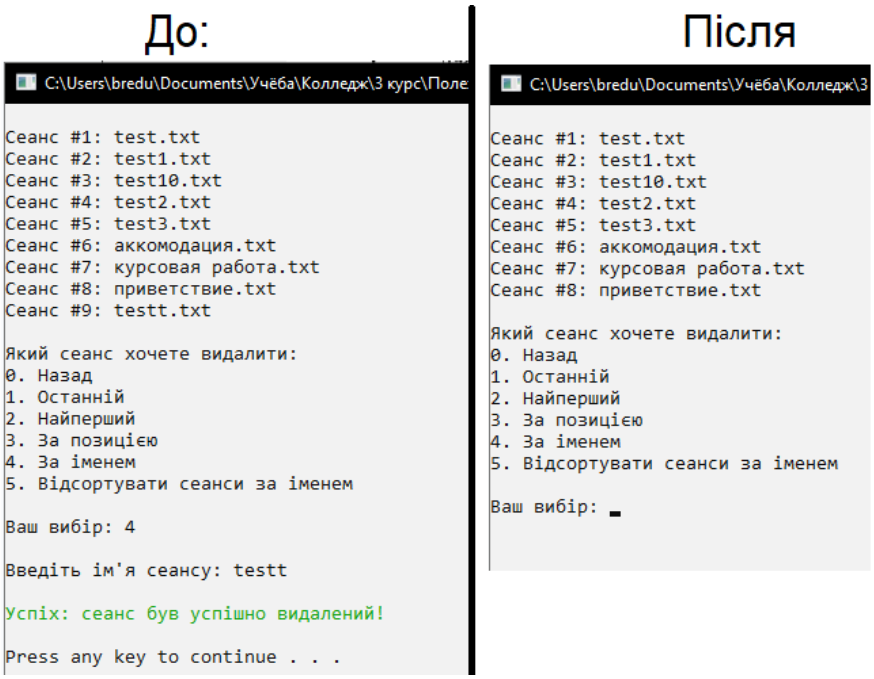

Рисунок 2.29 - Видалення сеансу

При відкритті або видаленні сеансів їх також можна відсортувати за іменем, обравши пункт “5” підменю (див. рис. 2.30).

Увійшовши в певний сеанс, іншими словами - обравши певний файл, користувач може працювати над змістом файлу за допомогою меню, наведеного на рисунку 2.31.

До:

```
C:\Users\bredu\Documents\Учёба\Колледж\3 курс\Поле

Сеанс #1: test.txt
Сеанс #2: test1.txt
Сеанс #3: test10.txt
Сеанс #4: test2.txt
Сеанс #5: курсовая работа.txt
Сеанс #6: пункт назначения.txt
Сеанс #7: Авдеевка.txt
Сеанс #8: accomondation.txt
Сеанс #9: test11.txt
Сеанс #10: test4.txt
Сеанс #11: test3.txt

Який сеанс хочете отримати:
0. Назад
1. Останній
2. Найперший
3. За позицією
4. За іменем
5. Відсортувати сеанси за іменем

Ваш вибір: 5

Успіх: сеанси були успішно відсортовані!

Press any key to continue . . .
```

Після:

```
C:\Users\bredu\Documents\Учёба\Колледж\3

Сеанс #1: accomondation.txt
Сеанс #2: test.txt
Сеанс #3: test1.txt
Сеанс #4: test10.txt
Сеанс #5: test11.txt
Сеанс #6: test2.txt
Сеанс #7: test3.txt
Сеанс #8: test4.txt
Сеанс #9: Авдеевка.txt
Сеанс #10: курсовая работа.txt
Сеанс #11: пункт назначения.txt

Який сеанс хочете отримати:
0. Назад
1. Останній
2. Найперший
3. За позицією
4. За іменем
5. Відсортувати сеанси за іменем

Ваш вибір:
```

Рисунок 2.30 - Сортування сеансів за іменами

```
C:\Users\bredu\Documents\Учёба\Коллед

Зміст файлу accomondation.txt:

Файл пустий!

Меню дій над змістом:
0. Назад
1. Додати текст
2. Видалити текст
3. Копіювати текст
4. Вирізати текст
5. Скасувати команду
6. Повторити команду

Ваш вибір:
```

Рисунок 2.31 - Меню редагування змісту файлу

Обравши пункт “1” цього меню, користувач зможе ввести текст з клавіатури або вставити з буферу обміну, якщо він не пустий (див. рис. 2.32).

Далі користувач може вставити текст в кінець або на початок тексту з файлу (див. рис. 2.33) або замінити якусь частину тексту з файлу тим текстом, який він тільки що ввів (див. рис. 2.34).

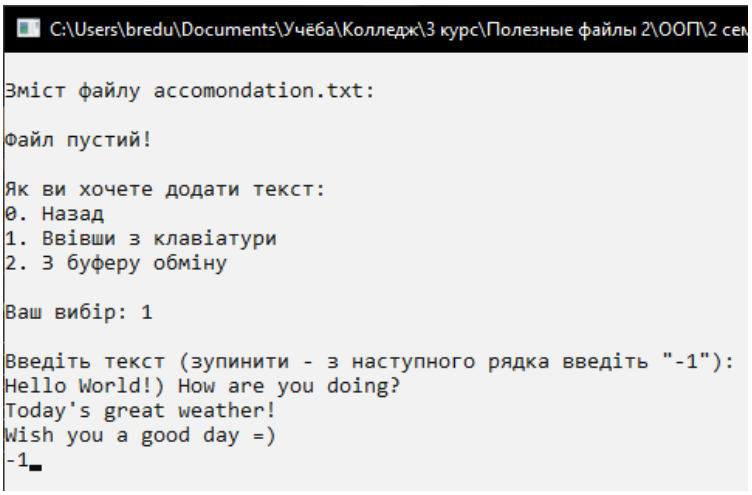

Рисунок 2.32 - Ввід тексту з клавіатури

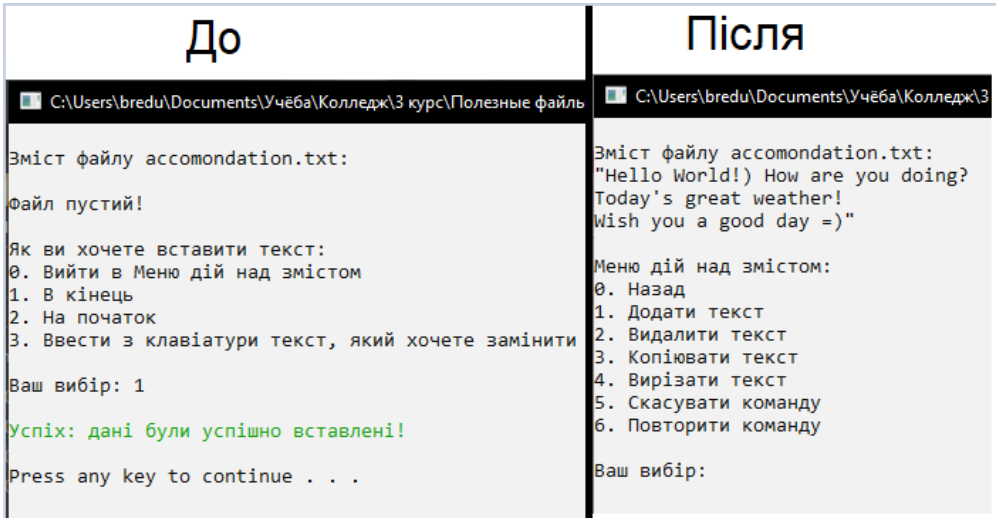

Рисунок 2.33 - Вставка тексту в файл за допомогою вводу з клавіатури

Обравши пункт “2” користувач може видалити весь або частину тексту (див. рис. 2.35).

Обравши пункт “3” користувач може копіювати весь або частину тексту (див. рис. 2.36).

Обравши пункт “4” користувач може вирізати весь або частину тексту (див. рис. 2.37).

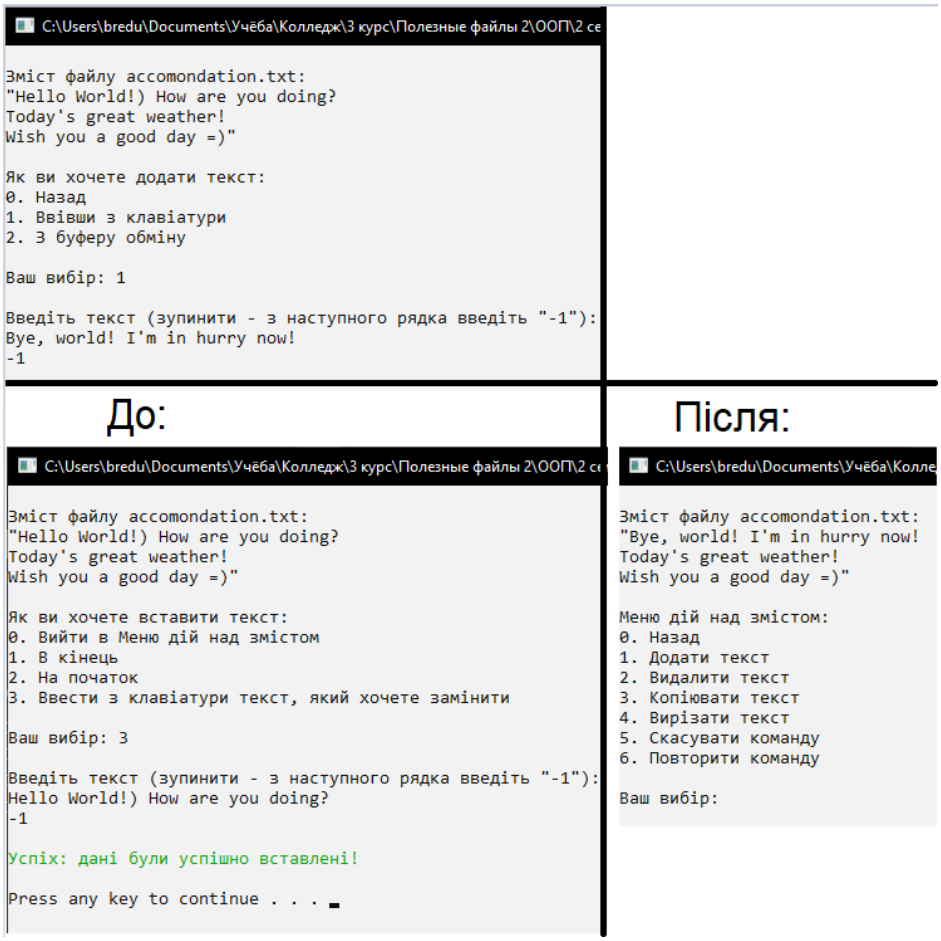

Рисунок 2.34 - Вставка тексту замість якоїсь частини вже існуючого тексту

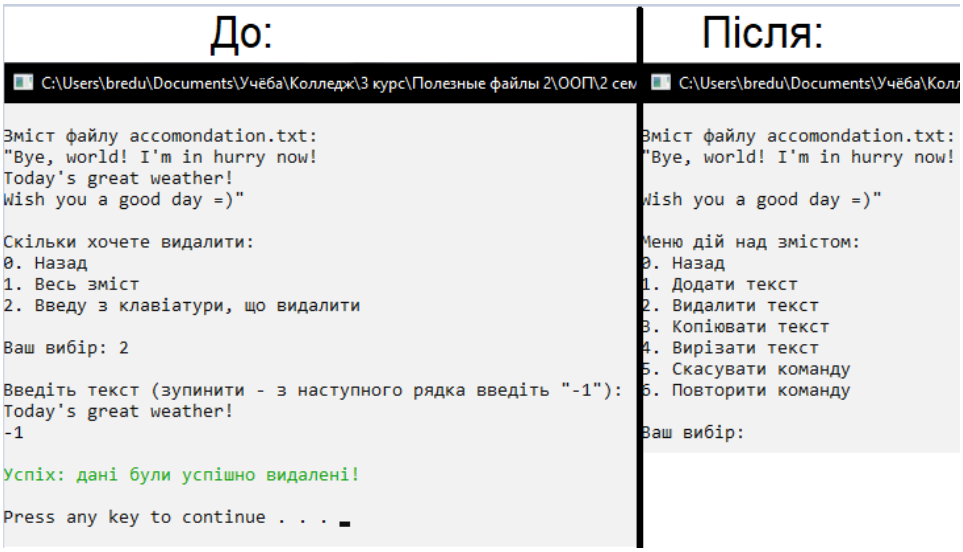

Рисунок 2.35 - Видалення тексту

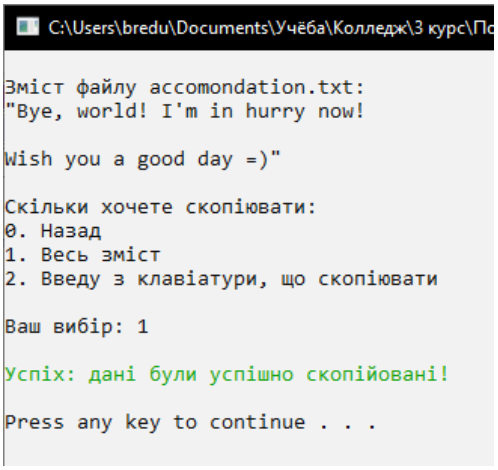

Рисунок 2.36 - Копіювання тексту

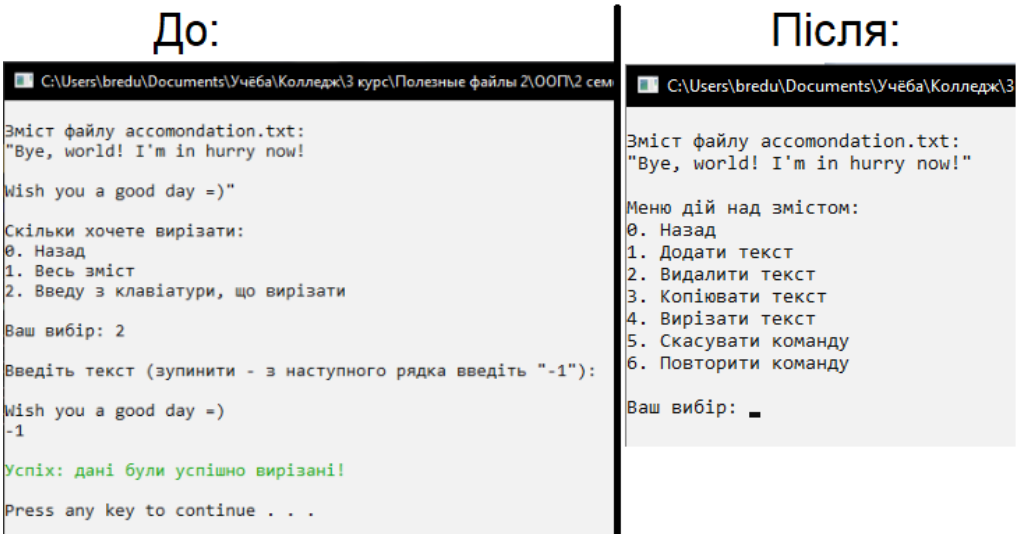

Рисунок 2.37 - Вирізання тексту

Маючи дані в буфері обміну, їх можна використати (див. рис. 2.38). Буфер обміну має дані тільки поки працює програма. При виході з програми він очищується.

Обравши пункт “5” користувач може скасувати останню дію (див. рис. 2.39).

Обравши пункт “6” користувач може повторити останню скасовану дію (див. рис. 2.40).

Для повернення з будь-якого підменю назад та для виходу з програми потрібно обрати пункт “0”. При виході з програми автоматично зберігаються мета-

дані, щоб користувач при наступному вході в цей сеанс мав теж можливість скасувати та повторити дії.

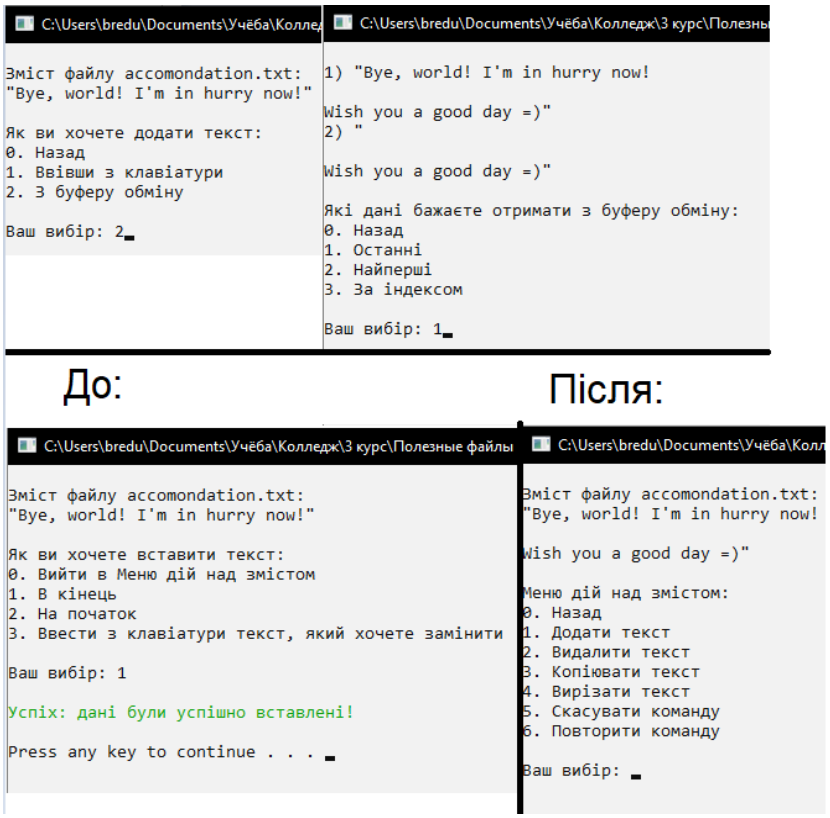

Рисунок 2.38 - Вставка даних з буферу обміну

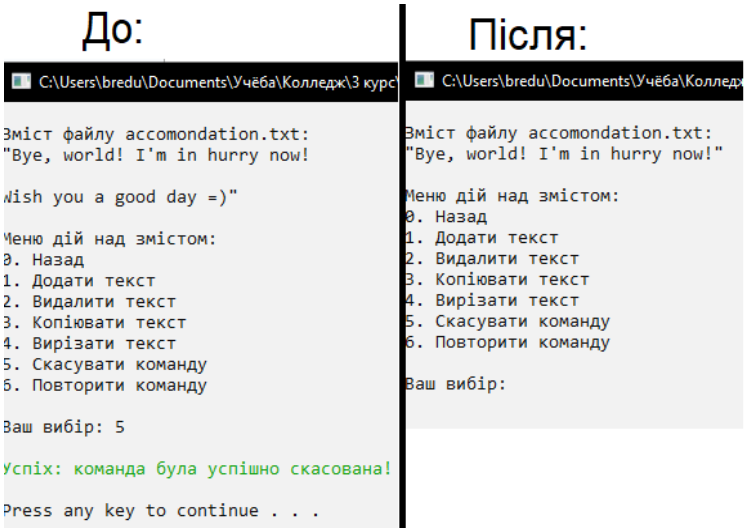

Рисунок 2.39 - Скасування останньої дії

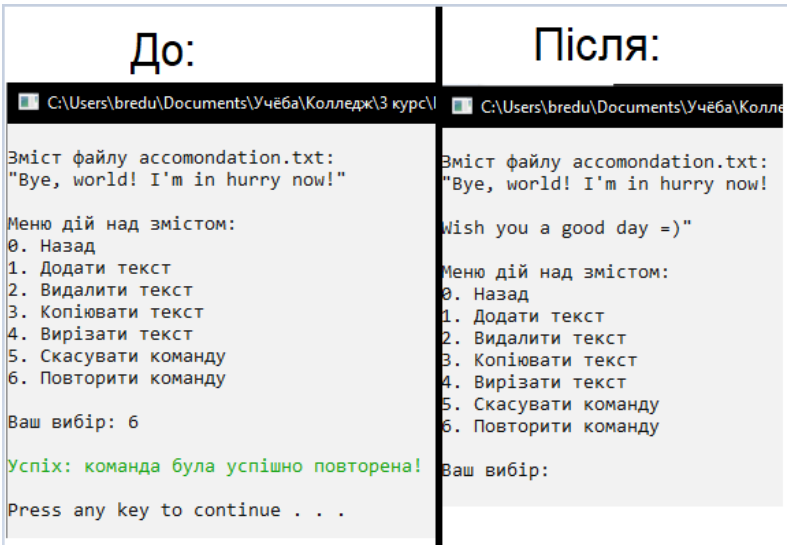

Рисунок 2.40 - Повторення останньої скасованої дії

Результат редагування файлу та метадані сеансу наведені на рисунку 2.41.

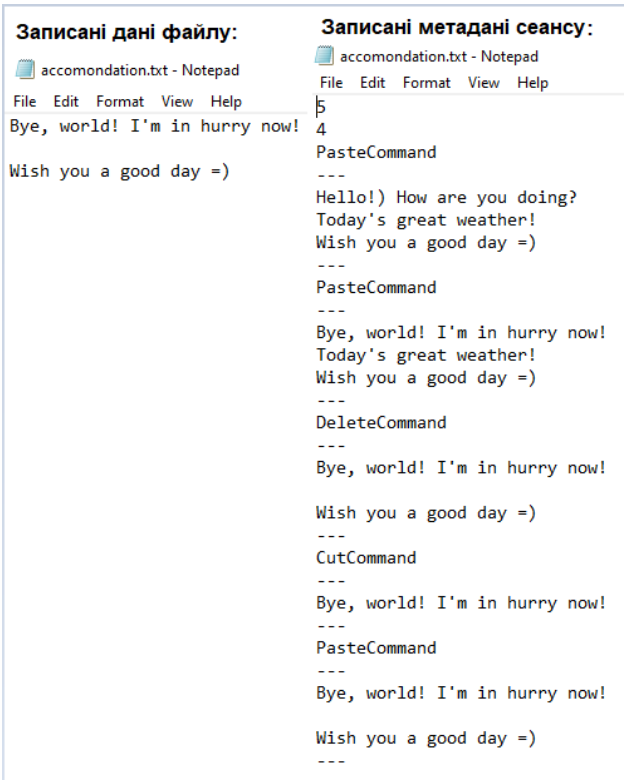

Рисунок 2.41 – Дані та метадані для файлу accomondation.txt

2.8 Результати реалізації програми

У прикладі розробленої ієрархії класів – Session, SessionsHistory, Editor, Command, CopyCommand, PasteCommand, CutCommand, DeleteCommand, UndoCommand, RedoCommand, FilesManager, CommandsManager, Program – застосовані концепції ООП та інструменти стандартної бібліотеки шаблонів, яка дозволить у майбутньому складати нові абстрактні типи даних, які будуть об’єднувати, зберігати та обробляти різні типи даних.

Дані, що вводяться і виводяться підтримують латинський алфавіт та кирилицю, тому інформація, що вводиться та виводиться не деформується.

Для усунення недоліків було б добре відцентрувати текст, зберігати буфер обміну для кожного сеансу в файл, щоб при наступному відкритті сеансу користувач мав можливість продовжити використовувати його, також переробити програму з використанням Windows UI, щоб користувач мав можливість безпосередньо в текстовому полі редагувати текст, додати гарячі клавіші, виділяти безпосередньо зміни в тексті певним кольором.

ВИСНОВКИ ПО РОБОТІ

В даній курсовій роботі була розроблена ієрархія класів відповідно до постанови задачі курсової роботи: Session, SessionsHistory, Editor, Command, CopyCommand, PasteCommand, CutCommand, DeleteCommand, UndoCommand, RedoCommand, FilesManager, CommandsManager, Program і тестова програма можливостей класів Session і SessionsHistory; визначені потрібні методи (сетери, гетери). Застосування при складанні класів динамічної структури stack дало змогу вдосконалити знання по їх створенню, обробці та ідентифікації.

За час виконання курсової роботи я досконально вивчив принципи концепцій ООП (створення класів, їх реалізація абстрактного класу), визначення методів, інструментів STL (cin, cout, fstream тощо) та файлових потоків.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бублик В. В. Об'єктно-орієнтоване програмування : підручник. Київ : ІТ-кн., 2015. 624 с.
2. Грицюк Ю., Рак Т. Об'єктно-орієнтоване програмування мовою С++ : навч. посіб. Львів : ЛДУ БЖД, 2011. 202 с.
3. Hyde R. Write great code : навчальний посібник / ed by.: B. Yien et al. 3th ed. San Francisco : No Starch Press, Inc, 2020. 1259 p.
4. Lafore R. Object-oriented programming in C++ : навчальний посібник / ed. by: M. Purcell et al. USA : Sams Publishing, 2001. 1038 p.
5. Воробйова О. Д., Глазунова Л. В. Алгоритми та структури даних. Частина 1. Структури даних : конспект лекцій. Одеса : ОНАЗ ім.О.С. Поп., 2017. 48 с.
6. Lospinoso J. C++ crash course. A fast-paced introduction. : навчальний посібник / ed. by: M. Sneeringer, R. Hoffman. San Francisco : No Starch Press, Inc, 2019. 794 p.
7. Green D., Guntheroth K., Mitchell S. The C++ workshop : навчальний посібник / ed. by M. Dhyani. Birmingham : Packt Publishing Ltd, 2020. 605 p.
8. Кравець П. О. Об'єктно-орієнтоване програмування : навч. посіб. Львів : Вид-во Львів. політехніки, 2012. 624 с.
9. Design Patterns: Elements of Reusable Object-Oriented Software : навчальний посібник / E. Gamma et al. Portland : Addison-Wesley, 1994. 395 p.

Додаток А

Блок-схеми

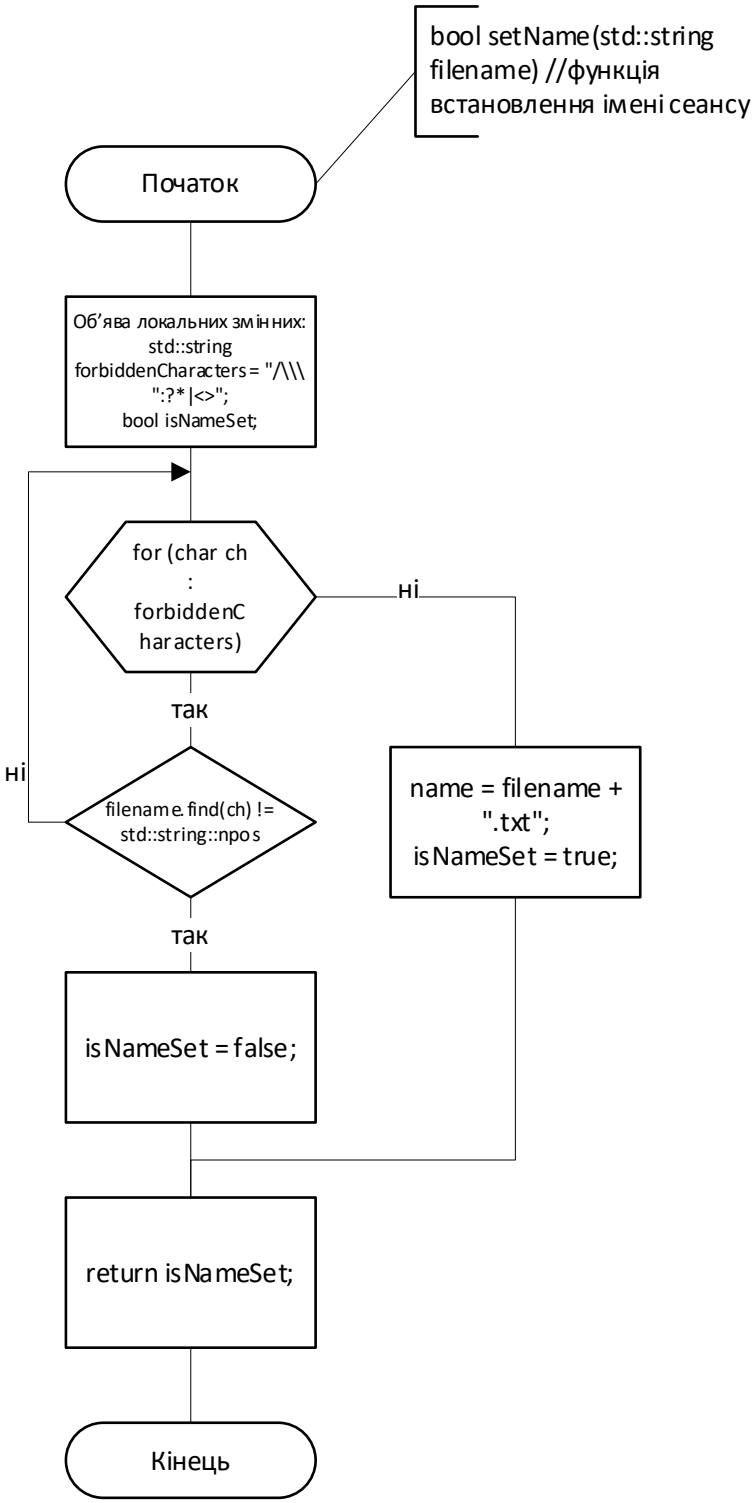

Рисунок А.1 – Блок-схема методу setName() класу Session

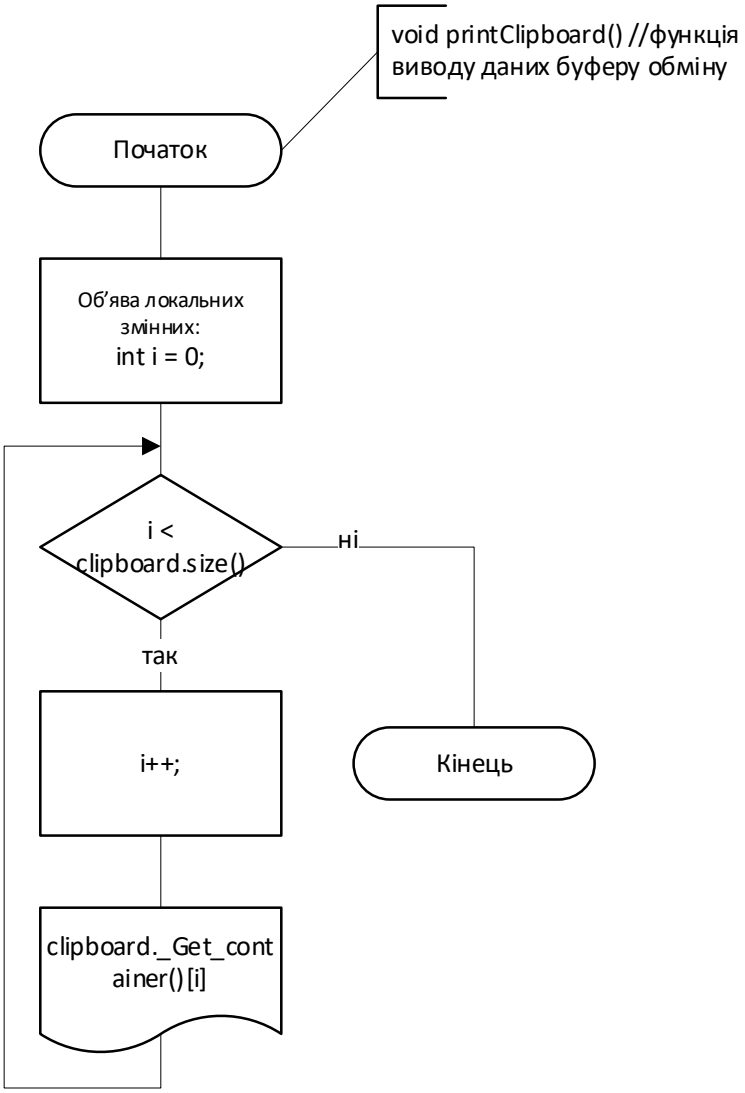

Рисунок А.2 – Блок-схема методу printClipboard() класу Session

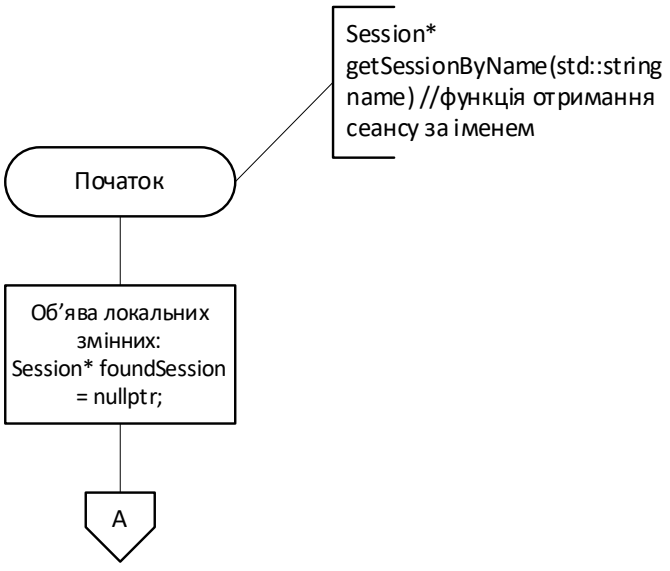

Рисунок А.3 – Блок-схема методу getSessionByName() класу SessionsHistory

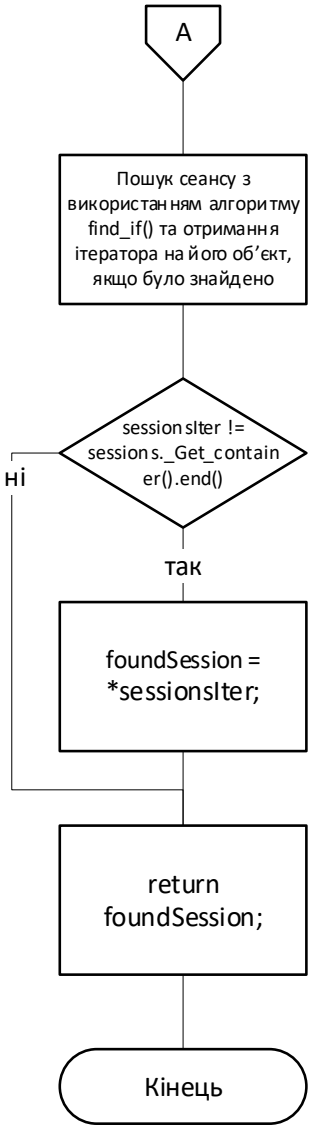

Рисунок А.3, аркуш 2

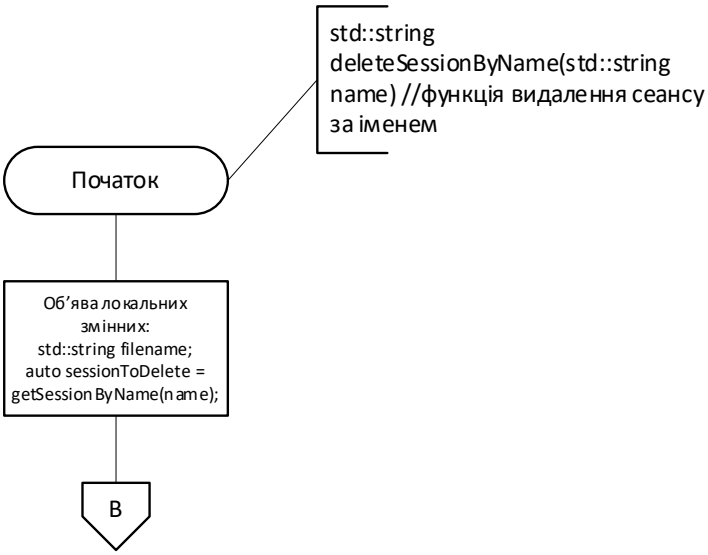

Рисунок А.4 – Блок-схема методу deleteSessionByName() класу SessionsHistory

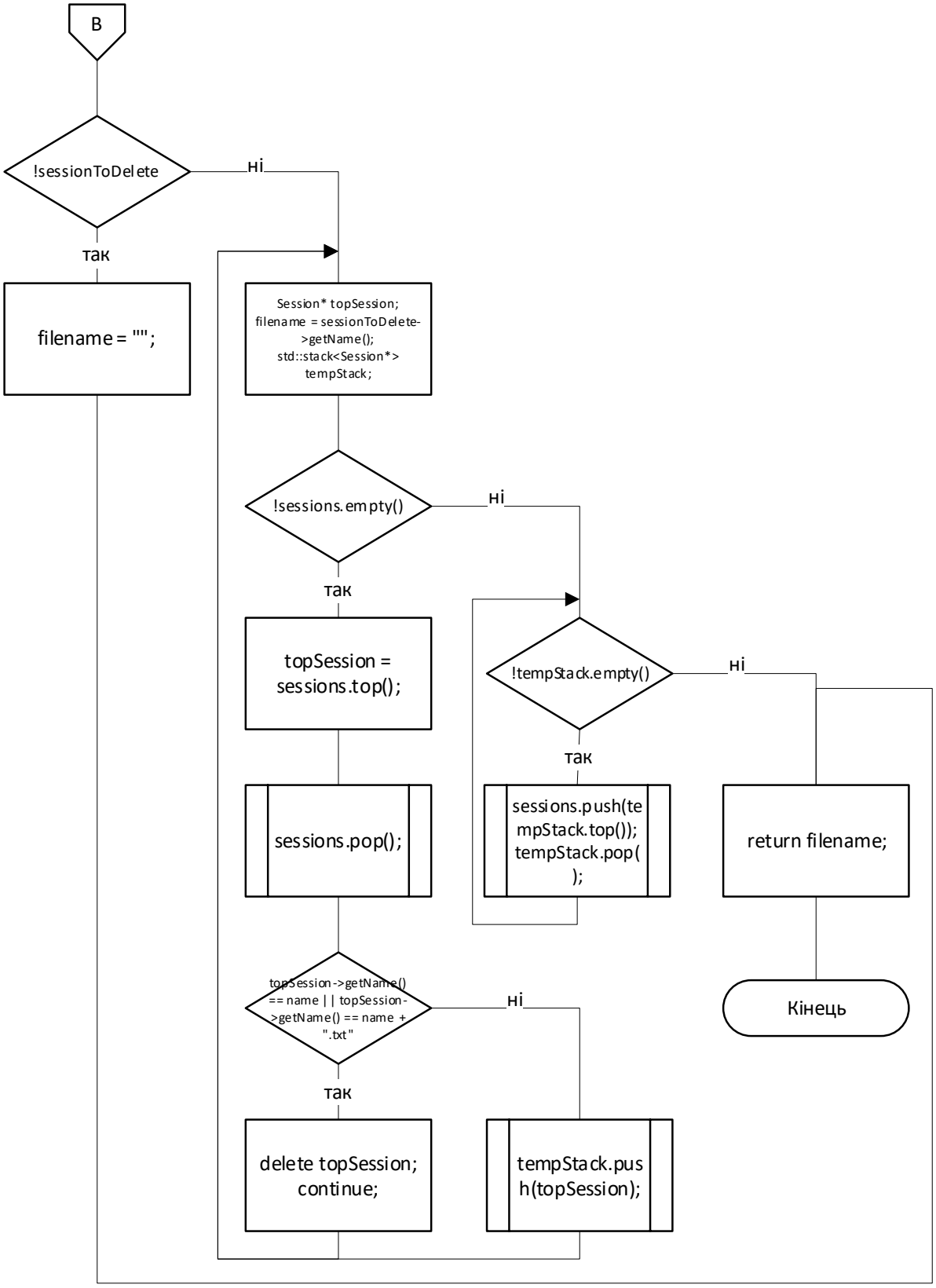

Рисунок А.4, аркуш 2

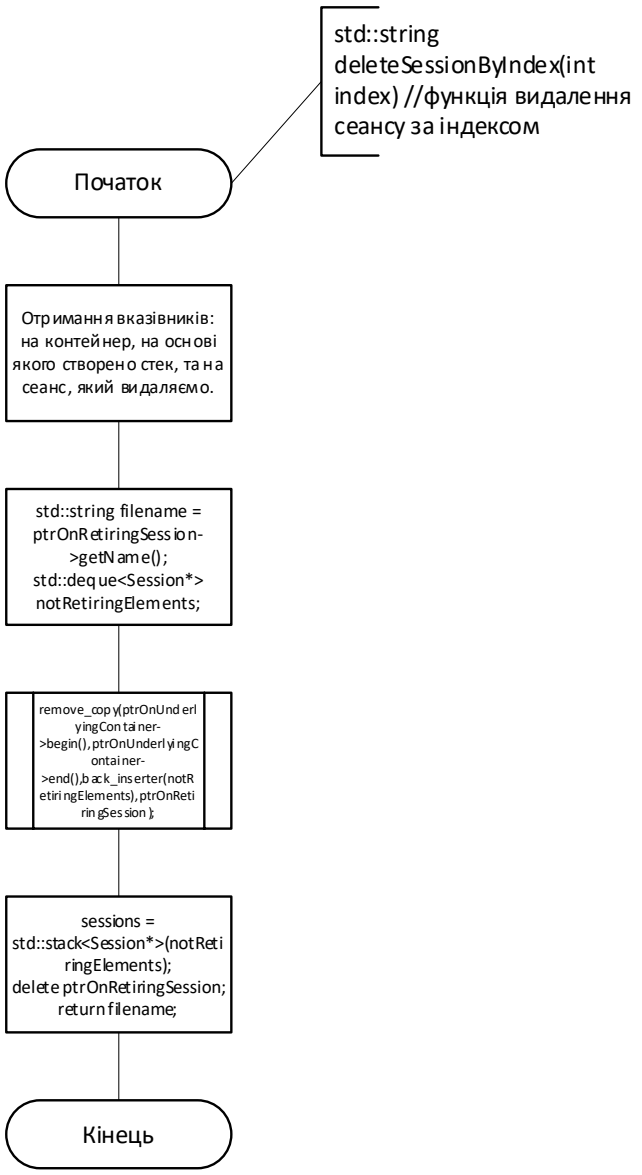

Рисунок А.5 – Блок-схема методу deleteSessionByIndex() класу SessionsHistory

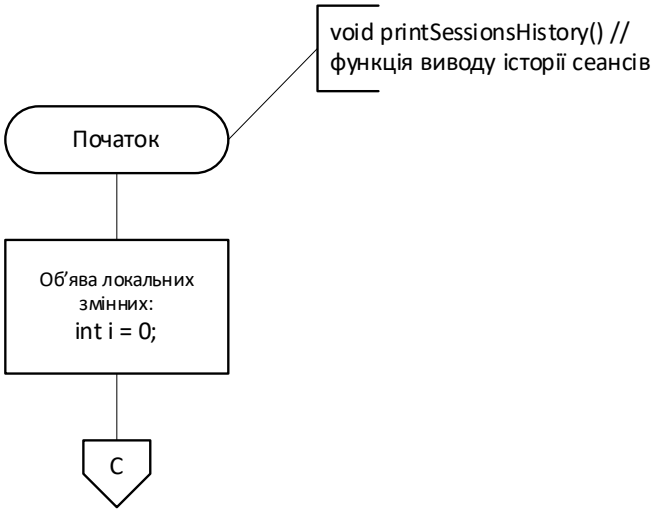

Рисунок А.6 – Блок-схема методу printSessionsHistory() класу SessionsHistory

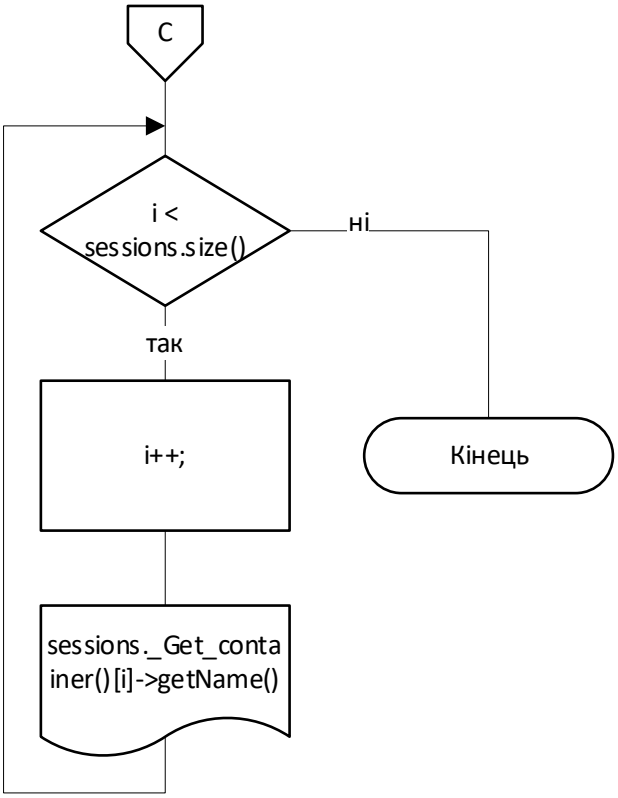

Рисунок А.6, аркуш 2

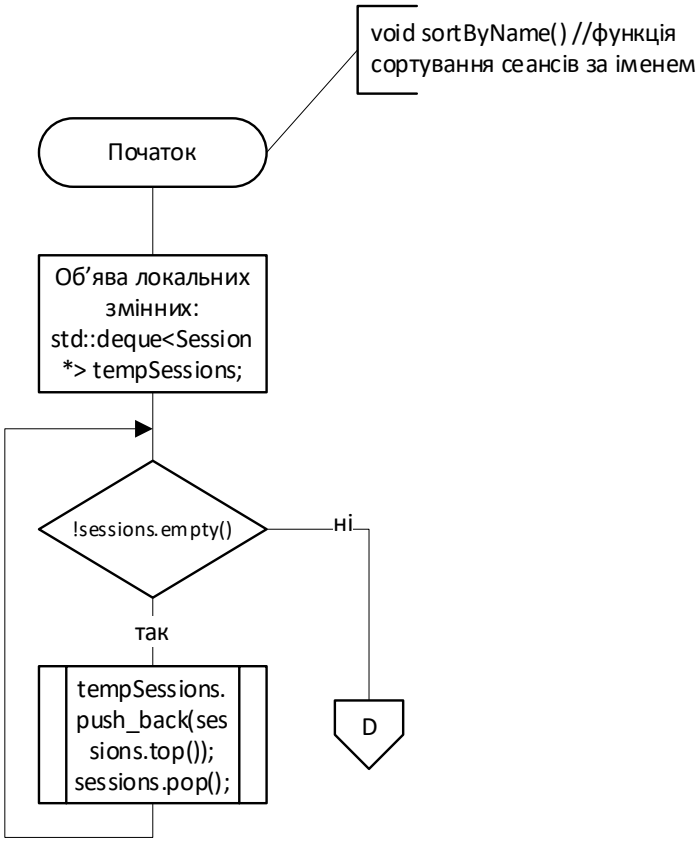

Рисунок А.7 – Блок-схема методу sortByName() класу SessionsHistory

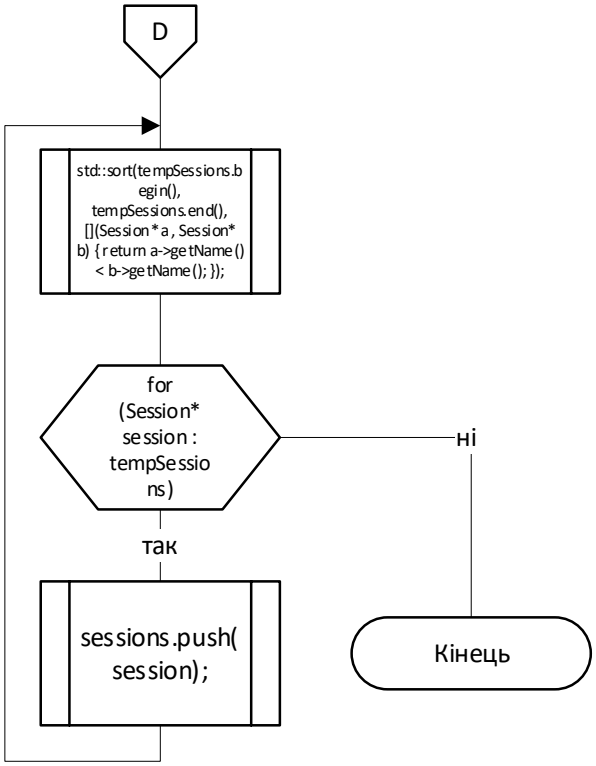

Рисунок А.7, аркуш 2

Додаток Б

Лістинг програми

////////////////////// Program.cpp //////////////////////

```
#include <iostream>
#include <fstream>
#include <filesystem>
#include <stack>
#include <functional>
#include <windows.h>

class Command;

class Session {
private:
	std::stack<Command*> commandsHistory; //історія команд
	std::stack<std::string> clipboard; //буфер обміну
	int currentCommandIndexInHistory; //індекс на команді, на якій знаходиться кори-
стувач, бо, можливо, він скасував декілька команд або повторив,
	//і це потрібно відслідковвувати
	std::string name; //ім'я сеансу

public:
	Session() { currentCommandIndexInHistory = -1; }
	Session(std::string filename) : Session() { name = filename; }
	~Session() {
		while (!commandsHistory.empty()) {
			if (commandsHistory.top())
				delete commandsHistory.top();
			commandsHistory.pop();
		}
	}

	void addCommandAsLast(Command* command) { commandsHistory.push(command); }
	void addDataToClipboard(std::string data) { clipboard.push(data); }
	void deleteLastCommand() {
		delete commandsHistory.top();
		commandsHistory.pop();
	}

	int sizeOfCommandsHistory() { return commandsHistory.size(); }
	int sizeOfClipboard() { return clipboard.size(); }

	bool setName(std::string filename) {
		std::string forbiddenCharacters = "/\\\":?*|<>";

		for (char ch : forbiddenCharacters)
			if (filename.find(ch) != std::string::npos)
				return false;

		name = filename + ".txt";
		return true;
	}
	void setCurIndexInCommHistory(int currentCommandIndexInHistory) {
		this->currentCommandIndexInHistory = currentCommandIndexInHistory;
	}

	std::string getName() { return name; }
	Command* getCommandByIndex(int index) { return commandsHisto-
ry._Get_container()[index]; }
	int getCurIndexInCommHistory() { return currentCommandIndexInHistory; }
```

```
	std::string getDataFromClipboardByIndex(int index) { return clip-
board._Get_container()[index]; }

	void printClipboard() {
		system("cls");
		for (int i = 0; i < clipboard.size(); i++)
			std::cout << "\n" << i + 1 << ") \"" << clipboard._Get_container()[i]
<< "\"";
		std::cout << std::endl;
	}
};

class SessionsHistory {
private:
	std::stack<Session*> sessions; //історія сеансів

public:
	~SessionsHistory() {
		while (!sessions.empty()) {
			delete sessions.top();
			sessions.pop();
		}
	}

	void addSessionToEnd(Session* session) { sessions.push(session); }
	Session* getSessionByIndex(int index) { return sessions._Get_container()[index]; }
	Session* getSessionByName(std::string name) {
		auto sessionsIter = std::find_if(sessions._Get_container().begin(), ses-
sions._Get_container().end(), [name](Session* session) {
			return session->getName() == name + ".txt" || session->getName() ==
name;
			});

		if (sessionsIter != sessions._Get_container().end())
			return *sessionsIter;
		else
			return nullptr;
	}
	std::string deleteSessionByIndex(int index) {
		auto ptrOnUnderlyingContainer = &sessions._Get_container();
		auto ptrOnRetiringSession = (*ptrOnUnderlyingContainer)[index];

		std::string filename = ptrOnRetiringSession->getName();

		std::deque<Session*> notRetiringElements;

		remove_copy(ptrOnUnderlyingContainer->begin(),
			ptrOnUnderlyingContainer->end(),
			back_inserter(notRetiringElements),
			ptrOnRetiringSession);

		sessions = std::stack<Session*>(notRetiringElements);

		delete ptrOnRetiringSession;

		return filename;
	}
	std::string deleteSessionByName(std::string name) {
		auto sessionToDelete = getSessionByName(name);
		if (!sessionToDelete)
			return "";

		Session* topSession;
		std::string filename = sessionToDelete->getName();
		std::stack<Session*> tempStack;
```

```
		while (!sessions.empty()) {
			topSession = sessions.top();
			sessions.pop();

			if (topSession->getName() == name || topSession->getName() == name +
".txt") {
				delete topSession;
				continue;
			}

			tempStack.push(topSession);
		}

		while (!tempStack.empty()) {
			sessions.push(tempStack.top());
			tempStack.pop();
		}

		return filename;
	}

	bool isEmpty() { return sessions.size() == 0; }
	int size() { return sessions.size(); }

	void printSessionsHistory() {
		system("cls");
		for (int i = 0; i < sessions.size(); i++)
			std::cout << "\nСеанс #" << i + 1 << ": " << ses-
sions._Get_container()[i]->getName();
		std::cout << std::endl;
	}

	void sortByName() {
		std::deque<Session*> tempSessions;

		while (!sessions.empty()) {
			tempSessions.push_back(sessions.top());
			sessions.pop();
		}

		std::sort(tempSessions.begin(), tempSessions.end(), [](Session* a, Session*
b) {
			return a->getName() < b->getName();
			});

		for (Session* session : tempSessions)
			sessions.push(session);
	}
};

class Editor {
private:
	static SessionsHistory* sessionsHistory; //історія сеансів
	static Session* currentSession; //сеанс, з яким користувач працює в даний момент
	static std::string* currentText; //текст, який користувач редагує в даний момент

public:
	Editor();

	~Editor() {
		delete sessionsHistory;
		if(currentText)
			delete (currentText);
	}
```

```
	void tryToLoadSessions();
	void tryToUnloadSessions();

	void copy(std::string textToProcess, int startPosition, int endPosition);
	void paste(std::string* textToProcess, int startPosition, int endPosition,
std::string textToPaste);
	void cut(std::string* textToProcess, int startPosition, int endPosition);
	void remove(std::string* textToProcess, int startPosition, int endPosition);

	static Session* getCurrentSession();
	static SessionsHistory* getSessionsHistory();
	static std::string* getCurrentText();
	static void setCurrentSession(Session* session);
	static void setCurrentText(std::string* text);

	static void printCurrentText();
};

class Command {
protected:
	Editor* editor; //редактор, в якому відбувається редагування тексту за допомогою
команд
	int startPosition, endPosition; //початкова та кінцева позиції для вставки,
заміни, видалення, копіювання, вирізання
	std::string textToProcess, textToPaste; //поля для тексту, який обробляємо і для
тексту, який вставляємо
	Command* previousCommand, * commandToUndoOrRedo; //вказівник на попередню команду
(в історії команд щось по типу однонапрямленого списка),
	//далі - вказівник на команду, яку збираємось скасувати або повторити

public:
	virtual void execute() = 0;
	virtual void undo() = 0;
	virtual Command* copy() = 0;

	void setParameters(std::string typeOfCommand, Command* previousCommand, Command*
commandToUndoOrRedo, int startPosition, int endPosition, std::string textToPaste) {
		if (typeOfCommand == "Undo" || typeOfCommand == "Redo")
		{
			this->commandToUndoOrRedo = commandToUndoOrRedo;
			return;
		}

		if (startPosition > endPosition && endPosition > -1 && startPosition < Edi-
tor::getCurrentText()->size())
			std::swap(startPosition, endPosition);

		if (startPosition == -1 && endPosition == -1) {
			startPosition = 0;
			endPosition = 0;
		}

		this->startPosition = startPosition;
		this->endPosition = endPosition;
		this->textToProcess = *(Editor::getCurrentText());
		this->previousCommand = previousCommand;

		if (typeOfCommand == "Paste")
			this->textToPaste = textToPaste;
	}

	std::string getTextToProcess() { return textToProcess; }
	void setTextToProcess(std::string textToProcess) { this->textToProcess = textTo-
Process; }
	void setPreviousCommand(Command* previousCommand) { this->previousCommand = previ-
ousCommand; }
```

```
};

class CopyCommand : public Command {
public:
	CopyCommand(Editor* editor);

	void execute() override;
	void undo() override;
	Command* copy() override;
};

class DeleteCommand : public Command {
public:
	DeleteCommand(Editor* editor);

	void execute() override;
	void undo() override;
	Command* copy() override;
};

class CutCommand : public Command {
public:
	CutCommand(Editor* editor);

	void execute() override;
	void undo() override;
	Command* copy() override;
};

class PasteCommand : public Command {
public:
	PasteCommand(Editor* editor);

	void execute() override;
	void undo() override;
	Command* copy() override;
};

class UndoCommand : public Command {
public:
	~UndoCommand() {
		if (commandToUndoOrRedo)
			delete (commandToUndoOrRedo);
	}

	void execute() override;
	void undo() override;
	Command* copy() override;
};

class RedoCommand : public Command {
public:
	~RedoCommand() {
		if (commandToUndoOrRedo)
			delete (commandToUndoOrRedo);
	}

	void execute() override;
	void undo() override;
	Command* copy() override;
};

class FilesManager {
private:
	friend class Editor;
```

```
	static const std::string METADATA_DIRECTORY, //директорія папки метаданих
		DATA_DIRECTORY; //директорія, де безпосередньо зберагаються текстові файли, які ми редагуємо в програмі

	static std::stack<std::string> getFilepathsForMetadata(std::string directory) {
		std::stack<std::string> filesFromMetadataDirectory;

		auto iteratorOnFiles = std::filesystem::directory_iterator(directory);

		for (const auto& entry : iteratorOnFiles)
			filesFromMetadataDirectory.push(entry.path().string());

		return filesFromMetadataDirectory;
	}
	static std::string readDataByDelimiter(std::ifstream* ifs_session, std::string delimiter) {
		std::string line, text;
		int counterOfLines = 0;

		getline(*ifs_session, line);
		while (getline(*ifs_session, line)) {
			if (line == delimiter)
				break;
			counterOfLines++;
			if (counterOfLines > 1)
				line = "\n" + line;
			text += line;
		}

		return text;
	}

	static void deleteMetadataForDeletedSessions(SessionsHistory* sessionsHistory, std::string directory) {
		if (!std::filesystem::exists(directory))
			return;

		std::stack<std::string> filesFromMetadataDirectory = getFilepathsForMetadata(directory);
		std::string sessionFilepath;
		bool isPartOfRealtimeSessions;

		for (int i = 0; i < filesFromMetadataDirectory.size(); i++)
		{
			isPartOfRealtimeSessions = false;
			for (int j = 0; j < sessionsHistory->size(); j++)
			{
				sessionFilepath = directory + sessionsHistory->getSessionByIndex(j)->getName();
				if (filesFromMetadataDirectory._Get_container()[i] == sessionFilepath)
					isPartOfRealtimeSessions = true;
			}

			if (!isPartOfRealtimeSessions)
				remove(filesFromMetadataDirectory._Get_container()[i].c_str());
		}

	}

	static void writeSessionsMetadata(SessionsHistory* sessionsHistory) {
		deleteMetadataForDeletedSessions(sessionsHistory, METADATA_DIRECTORY);

		if (!std::filesystem::exists(METADATA_DIRECTORY))
			std::filesystem::create_directories(METADATA_DIRECTORY);
```

```
		for (int i = 0; i < sessionsHistory->size(); i++)
			writeSessionMetadata(sessionsHistory, i);
	}
	static void writeSessionMetadata(SessionsHistory* sessionsHistory, int index) {
		Session* session = sessionsHistory->getSessionByIndex(index);

		std::ofstream ofs_session(METADATA_DIRECTORY + session->getName());

		ofs_session << session->sizeOfCommandsHistory() << std::endl;
		ofs_session << session->getCurIndexInCommHistory() << std::endl;

		for (int j = 0; j < session->sizeOfCommandsHistory(); j++)
			writeCommandMetadata(&ofs_session, session, j);

		ofs_session.close();
	}
	static void writeCommandMetadata(std::ofstream* ofs_session, Session* session, int
index) {
		std::string typeOfCommand, nameOfCommandClass, delimiter = "---\n";
		Command* command = session->getCommandByIndex(index);

		nameOfCommandClass = std::string(typeid(*command).name());

		if (nameOfCommandClass == "class PasteCommand")
			typeOfCommand = "PasteCommand";
		else if (nameOfCommandClass == "class CutCommand")
			typeOfCommand = "CutCommand";
		else
			typeOfCommand = "DeleteCommand";

		*ofs_session << typeOfCommand << std::endl << delimiter;

		if (command->getTextToProcess() != "")
			*ofs_session << command->getTextToProcess() << std::endl;

		*ofs_session << delimiter;
	}

	static void readSessionsMetadata(Editor* editor) {
		if (!std::filesystem::exists(METADATA_DIRECTORY))
			return;

		std::stack<std::string> available_sessions;

		available_sessions = getFilepathsForMetadata(METADATA_DIRECTORY);

		for (int i = 0; i < available_sessions.size(); i++)
			readSessionMetadata(editor, available_sessions._Get_container()[i]);
	}
	static void readSessionMetadata(Editor* editor, std::string filepath) {
		std::string line;
		filepath.erase(0, METADATA_DIRECTORY.size());
		Session* session = new Session(filepath);
		int countOfCommands;

		std::ifstream ifs_session(METADATA_DIRECTORY + filepath);

		getline(ifs_session, line);
		countOfCommands = stoi(line);

		getline(ifs_session, line);
		session->setCurIndexInCommHistory(stoi(line));

		for (int j = 0; j < countOfCommands; j++)
			readCommandMetadata(editor, &ifs_session, session);
```

```
            editor->getSessionsHistory()->addSessionToEnd(session);

            ifs_session.close();
      }
      static void readCommandMetadata(Editor* editor, std::ifstream* ifs_session, Ses-
sion* session) {
            std::string typeOfCommand, text;
            Command* command,* previousCommand;

            getline(*ifs_session, typeOfCommand);
            if (typeOfCommand == "CutCommand")
                  command = new CutCommand(editor);
            else if (typeOfCommand == "PasteCommand")
                  command = new PasteCommand(editor);
            else
                  command = new DeleteCommand(editor);

            if (session->sizeOfCommandsHistory() > 0)
            {
                  previousCommand = session->getCommandByIndex(session-
>sizeOfCommandsHistory() - 1);
                  command->setPreviousCommand(previousCommand);
            }

            text = readDataByDelimiter(ifs_session, "---");
            command->setTextToProcess(text);

            session->addCommandAsLast(command);
      }

public:
      static std::string getSessionsDirectory() {
            return DATA_DIRECTORY;
      }

      static std::string readSessionData(std::string fullFilepath) {
            std::string text, line;

            std::ifstream file(fullFilepath);

            int counterOfLines = 0;

            while (getline(file, line))
            {
                  counterOfLines++;
                  if (counterOfLines > 1)
                        line = "\n" + line;
                  text += line;
            }

            file.close();

            return text;
      }

      static bool writeSessionData(std::string filename, std::string newData) {
            std::ofstream file(DATA_DIRECTORY + filename);

            if (!file.is_open())
                  return false;

            file << newData;
            if(!newData.empty() && newData[newData.size() - 1] == '\n')
                  file << '\n';

            file.close();
```

```
            return true;
      }
};

const std::string FilesManager::METADATA_DIRECTORY = "Metadata\\",
FilesManager::DATA_DIRECTORY = "Data\\";

void Editor::tryToLoadSessions() { FilesManager::readSessionsMetadata(this); }
void Editor::tryToUnloadSessions() { FilesManag-
er::writeSessionsMetadata(sessionsHistory); }

Editor::Editor() { this->sessionsHistory = new SessionsHistory(); }

void Editor::copy(std::string textToProcess, int startPosition, int endPosition) {
      std::string dataToCopy = textToProcess.substr(startPosition, endPosition -
startPosition + 1);
      currentSession->addDataToClipboard(dataToCopy);
}
void Editor::paste(std::string* textToProcess, int startPosition, int endPosition,
std::string textToPaste) {
      if (startPosition == endPosition) {
            if(startPosition == 0)
                  *textToProcess = textToPaste + *textToProcess;
            else if(startPosition == textToProcess->size() - 1)
                  *textToProcess += textToPaste;
            else
                  (*textToProcess).replace(startPosition, endPosition - startPosition +
1, textToPaste);
      }
      else
      {
            if(endPosition == -1)
                  (*textToProcess).replace(0, 1, textToPaste);
            else if (endPosition == textToProcess->size())
                  (*textToProcess).replace(textToProcess->size() - 1, textToProcess-
>size() - 1, textToPaste);
            else
                  (*textToProcess).replace(startPosition, endPosition - startPosition +
1, textToPaste);
      }
      *currentText = *textToProcess;
}
void Editor::cut(std::string* textToProcess, int startPosition, int endPosition) {
      copy(*textToProcess, startPosition, endPosition);
      remove(textToProcess, startPosition, endPosition);
      *currentText = *textToProcess;
}
void Editor::remove(std::string* textToProcess, int startPosition, int endPosition) {
      (*textToProcess).erase(startPosition, endPosition - startPosition + 1);
      *currentText = *textToProcess;
}

Session* Editor::getCurrentSession() { return currentSession; }
std::string* Editor::getCurrentText() { return currentText; }
SessionsHistory* Editor::getSessionsHistory() { return sessionsHistory; }
void Editor::setCurrentSession(Session* session) { currentSession = session; }
void Editor::setCurrentText(std::string* text) { currentText = text; }

void Editor::printCurrentText() {
      system("cls");
      std::cout << "\nЗміст файлу " << currentSession->getName() << ":\n";
      *currentText != "" ?
            std::cout << "\"" << *currentText << "\"\n" :
            std::cout << "\nФайл пустий!\n";
}
```

```
SessionsHistory* Editor::sessionsHistory;
Session* Editor::currentSession;
std::string* Editor::currentText;

CopyCommand::CopyCommand(Editor* editor) { this->editor = editor; }

void CopyCommand::execute() { editor->copy(textToProcess, startPosition, endPosition); }
void CopyCommand::undo() { }
Command* CopyCommand::copy() { return nullptr; }

DeleteCommand::DeleteCommand(Editor* editor) { this->editor = editor; }

void DeleteCommand::execute() { editor->remove(&textToProcess, startPosition, endPosi-
tion); }
void DeleteCommand::undo() { *(Editor::getCurrentText()) = previousCommand-
>getTextToProcess(); }
Command* DeleteCommand::copy() { return new DeleteCommand(*this); }

CutCommand::CutCommand(Editor* editor) { this->editor = editor; }

void CutCommand::execute() { editor->cut(&textToProcess, startPosition, endPosition); }
void CutCommand::undo() { *(Editor::getCurrentText()) = previousCommand-
>getTextToProcess(); }
Command* CutCommand::copy() { return new CutCommand(*this); }

PasteCommand::PasteCommand(Editor* editor) { this->editor = editor; }

void PasteCommand::execute() { editor->paste(&textToProcess, startPosition, endPosition,
textToPaste); }
void PasteCommand::undo() {
	if (previousCommand)
		*(Editor::getCurrentText()) = previousCommand->getTextToProcess();
	else
		*(Editor::getCurrentText()) = "";
}
Command* PasteCommand::copy() { return new PasteCommand(*this); }

void UndoCommand::execute() { commandToUndoOrRedo->undo(); }
void UndoCommand::undo() { }
Command* UndoCommand::copy() { return nullptr; }

void RedoCommand::execute() { *(Editor::getCurrentText()) = commandToUndoOrRedo-
>getTextToProcess(); }
void RedoCommand::undo() { }
Command* RedoCommand::copy() { return nullptr; }

class CommandsManager {
private:
	std::stack<std::pair<std::string, Command*>> manager; //зберігач усіх команд, доз-
воляє зручно їми керувати за допомогою поліморфізму

	Command* getCommandFromManagerByKey(std::string typeOfCommand) {
		for (int i = 0; i < manager.size(); i++)
			if (manager._Get_container()[i].first == typeOfCommand)
				return manager._Get_container()[i].second;
	}
	bool isNotUndoOrRedoCommand(std::string typeOfCommand) { return typeOfCommand !=
"Undo" && typeOfCommand != "Redo"; }
	void setParametersForCommand(std::string typeOfCommand, int startPosition, int
endPosition, std::string textToPaste) {
		Command* commandToUndoOrRedo = nullptr, * previousCommand = nullptr;

		if (typeOfCommand == "Undo")
			commandToUndoOrRedo = Editor::getCurrentSession()-
>getCommandByIndex(Editor::getCurrentSession()->getCurIndexInCommHistory());
```

```
            if(typeOfCommand == "Redo")
                  commandToUndoOrRedo = Editor::getCurrentSession()-
>getCommandByIndex(Editor::getCurrentSession()->getCurIndexInCommHistory() + 1);

            if (Editor::getCurrentSession()->getCurIndexInCommHistory() != -1 && isNo-
tUndoOrRedoCommand(typeOfCommand))
                  previousCommand = Editor::getCurrentSession()-
>getCommandByIndex(Editor::getCurrentSession()->sizeOfCommandsHistory() - 1);



            getCommandFromManagerByKey(typeOfCommand)->setParameters(typeOfCommand, pre-
viousCommand, commandToUndoOrRedo, startPosition, endPosition, textToPaste);
      }
      int getCountOfForwardCommands() {
            return Editor::getCurrentSession()->sizeOfCommandsHistory() - 1 - Edi-
tor::getCurrentSession()->getCurIndexInCommHistory();
      }
      void deleteForwardCommandsIfNecessary(std::string typeOfCommand) {
            if (typeOfCommand != "Undo" && typeOfCommand != "Redo" && isThereAnyCommand-
Forward())
            {
                  int countOfForwardCommands = getCountOfForwardCommands();
                  for (int i = 0; i < countOfForwardCommands; i++)
                        Editor::getCurrentSession()->deleteLastCommand();
            }
      }

public:
      CommandsManager(Editor* editor) {
            manager.push(std::pair("Copy", new CopyCommand(editor)));
            manager.push(std::pair("Paste", new PasteCommand(editor)));
            manager.push(std::pair("Cut", new CutCommand(editor)));
            manager.push(std::pair("Delete", new DeleteCommand(editor)));
            manager.push(std::pair("Undo", new UndoCommand()));
            manager.push(std::pair("Redo", new RedoCommand()));
      }
      ~CommandsManager() {
            while (!manager.empty())
            {
                  delete manager.top().second;
                  manager.pop();
            }
      }

      bool isThereAnyCommandForward() {
            return Editor::getCurrentSession()->sizeOfCommandsHistory() != 0 &&
                  Editor::getCurrentSession()->getCurIndexInCommHistory() < Edi-
tor::getCurrentSession()->sizeOfCommandsHistory() - 1;
      }
      void invokeCommand(std::string typeOfCommand, int startPosition = 0, int endPosi-
tion = 0, std::string textToPaste = "") {

            deleteForwardCommandsIfNecessary(typeOfCommand);
            setParametersForCommand(typeOfCommand, startPosition, endPosition,
textToPaste);

            getCommandFromManagerByKey(typeOfCommand)->execute();

            if (isNotUndoOrRedoCommand(typeOfCommand) && typeOfCommand != "Copy")
                  Editor::getCurrentSession()-
>addCommandAsLast(getCommandFromManagerByKey(typeOfCommand)->copy());

            if(typeOfCommand == "Undo")
                  Editor::getCurrentSession()-
```

```
>setCurIndexInCommHistory(Editor::getCurrentSession()->getCurIndexInCommHistory() - 1);
		else
			if (typeOfCommand != "Copy")
				Editor::getCurrentSession()-
>setCurIndexInCommHistory(Editor::getCurrentSession()->getCurIndexInCommHistory() + 1);
	}
};

class Program {
private:
	Editor* editor; //редактор
	CommandsManager* commandsManager; //менеджер команд, за допомогою якого й виклика-
ються усі команди

	bool validateEnteredNumber(std::string option, short firstOption, short
lastOption) {
		if (option.empty())
			return false;

		for (char num : option)
			if (num < '0' || num > '9')
				return false;

		auto convertedValue = std::stoull(option);

		return firstOption <= convertedValue && convertedValue <= lastOption;
	}
	int enterNumberInRange(std::string message, int firstOption, int lastOption) {
		bool isOptionVerified = false;
		std::string option;

		std::cout << "\n" << message;
		getline(std::cin, option);

		isOptionVerified = validateEnteredNumber(option, firstOption, lastOption);

		if (!isOptionVerified)
			printNotification("error", "були введені зайві символи або число, яке
виходить за межі набору цифр наданих варіантів!");

		return isOptionVerified ? stoi(option) : -1;
	}
	void readDataFromFile() {
		std::string filepath = FilesManager::getSessionsDirectory() + editor-
>getCurrentSession()->getName();
		std::string textFromFile = FilesManager::readSessionData(filepath);
		editor->setCurrentText(new std::string(textFromFile));
	}
	void pauseAndCleanConsole() {
		system("pause");
		system("cls");
	}
	void printNotification(std::string type, std::string msg) {
		if (type == "success")
			successNotification(msg);
		else
			errorNotification(msg);
		pauseAndCleanConsole();
	}
	void successNotification(std::string msg) {
		std::cout << "\nУспіх: " << msg << "\n\n";
	}
	void errorNotification(std::string msg) {
		std::cout << "\nПомилка: " << msg << "\n\n";
	}
	void printReference() {
```

```
            system("cls");
            std::cout << "Розробив: Бредун Денис Сергійович з групи ПЗ-21-1/9\n\n";
            std::cout << "Застосунок дозволяє працювати з текстовими файлами створюючи,
редагуючи та видаляючи їх зміст\n";
            std::cout << "за допомогою команд Вставити, Вирізати, Копіювати, Видалити.
Також можна повертатись до минулого стану\n";
            std::cout << "файлу за допомогою команди Скасувати та повторити останню ко-
манду за допомогою команди Повторити.\n\n";
            std::cout << "Використаний патерн проектування: Команда.\n";
            std::cout << "Використаний контейнер: стек.\n";
            system("pause");
      }

      void templateForMenusAboutSessions(int& choice, std::string action) {
            std::cout << "\nЯкий сеанс хочете " << action << ":\n";
            std::cout << "0. Назад\n";
            std::cout << "1. Останній\n";
            std::cout << "2. Найперший\n";
            std::cout << "3. За позицією\n";
            std::cout << "4. За іменем\n";
            std::cout << "5. Відсортувати сеанси за іменем\n";
            choice = enterNumberInRange("Ваш вибір: ", 0, 5);
      }
      void templateForExecutingMenusAboutSessions(std::function<void(int&)> mainFunc,
std::function<void(int&)> menu,
            std::function<bool()> actionFuncByName, std::function<void()> additionalFunc
= nullptr) {
            int choice, index = -1;
            bool isActionSuccessfull = false;

            do
            {
                  editor->getSessionsHistory()->printSessionsHistory();
                  menu(choice);
                  switch (choice)
                  {
                  case -1: continue;
                  case 0:
                        std::cout << "\nПовернення до Головного меню.\n\n";
                        system("pause");
                        return;
                  default:
                        if (doesAnySessionExist())
                        {
                              switch (choice)
                              {
                              case 1:
                              case 2:
                              case 3:
                                    index = choice == 1 ? editor-
>getSessionsHistory()->size() : choice == 2 ? 1 : -1;
                                    mainFunc(index);
                                    if (index != -1 && additionalFunc)
                                          additionalFunc();
                                    continue;
                              case 4:
                                    isActionSuccessfull = actionFuncByName();
                                    if (isActionSuccessfull && additionalFunc)
                                          additionalFunc();
                                    continue;
                              case 5:
                                    sortSessions();
                              }
                        }
                  }
            } while (true);
```

```
	}

	std::string getTextUsingKeyboard() {
		std::string line, text;
		int countOfLines = 0;

		std::cout << "\nВведіть текст (зупинити - з наступного рядка введіть \"-
1\"): \n";
		while (getline(std::cin, line)) {
			if (line == "-1")
				break;
			else {
				countOfLines++;
				if (countOfLines > 1)
					text += "\n" + line;
				else
					if (line == "")
						text += "\n";
					else
						text += line;
			}
		}

		return text;
	}
	std::string getTextFromClipboard() {
		if (editor->getCurrentSession()->sizeOfClipboard() == 0) {
			printNotification("error", "в буфері обміну ще немає даних!");
			return "";
		}

		int choice, sizeOfClipboard = editor->getCurrentSession()-
>sizeOfClipboard();

		editor->getCurrentSession()->printClipboard();
		getTextFromClipboardMenu(choice);

		switch (choice)
		{
		case 0:
			std::cout << "\nПовернення до меню вибору способа додавання
текста.\n\n";
			system("pause");
			return "";
		case 1:
			return editor->getCurrentSession()-
>getDataFromClipboardByIndex(sizeOfClipboard - 1);
		case 2:
			return editor->getCurrentSession()->getDataFromClipboardByIndex(0);
		case 3:
			choice = enterNumberInRange("Введіть номер даних: ", 1, sizeOfClip-
board);
			if (choice != -1)
				return editor->getCurrentSession()-
>getDataFromClipboardByIndex(choice - 1);
			return "";
		default:
			return "";
		}
	}

	bool makeActionOnContextByEnteredText(std::string typeOfCommand, std::string ac-
tionInPast,
		std::string textToPaste = "", size_t startIndex = -2, size_t endIndex = -2)
{
		if (startIndex == -2 && endIndex == -2) {
```

```
                    if (editor->getCurrentText()->empty()) {
                          printNotification("error", "немає тексту, який можна було б
замінити!");
                          return false;
                    }

                    std::string textForAction;

                    textForAction = getTextUsingKeyboard();

                    if (textForAction.empty()) {
                          printNotification("error", "текст не був введений!");
                          return false;
                    }

                    startIndex = editor->getCurrentText()->find(textForAction);

                    if (startIndex == std::string::npos) {
                          printNotification("error", "текст не був знайдений!");
                          return false;
                    }

                    if (typeOfCommand == "Paste") {
                          if (startIndex == 0 && textForAction.size() == 1)
                                endIndex = -1;
                          else if (startIndex == editor->getCurrentText()->size() - 1 &&
textForAction.size() == 1)
                                endIndex = editor->getCurrentText()->size();
                          else
                                endIndex = startIndex + textForAction.size() - 1;
                    }
                    else {
                          if (textForAction.size() == 1)
                                endIndex = startIndex;
                          else
                                endIndex = startIndex + textForAction.size() - 1;
                    }
              }

              commandsManager->invokeCommand(typeOfCommand, startIndex, endIndex,
textToPaste);
              printNotification("success", "дані були успішно " + actionInPast + "!");
              return true;
        }

        bool undoAction() {
              if (editor->getCurrentSession()->sizeOfCommandsHistory() > 0 && editor-
>getCurrentSession()->getCurIndexInCommHistory() != -1)
              {
                    commandsManager->invokeCommand("Undo");
                    printNotification("success", "команда була успішно скасована!");
                    return true;
              }
              printNotification("error", "немає дій, які можна було б скасувати!");
              return false;
        }
        bool redoAction() {
              bool isThereAnyCommandForward = commandsManager->isThereAnyCommandForward();
              if (isThereAnyCommandForward)
              {
                    commandsManager->invokeCommand("Redo");
                    printNotification("success", "команда була успішно повторена!");

              }
              else
                    printNotification("error", "немає дій, які можна було б повторити!");
```

```
	return isThereAnyCommandForward;
}
void sortSessions() {
	editor->getSessionsHistory()->sortByName();
	printNotification("success", "сеанси були успішно відсортовані!");
}

void wayToGetTextForAddingMenu(int& choice) {
	std::cout << "\nЯк ви хочете додати текст:\n";
	std::cout << "0. Назад\n";
	std::cout << "1. Ввівши з клавіатури\n";
	std::cout << "2. З буферу обміну\n";
	choice = enterNumberInRange("Ваш вибір: ", 0, 2);
}
void getTextFromClipboardMenu(int& choice) {
	std::cout << "\nЯкі дані бажаєте отримати з буферу обміну:\n";
	std::cout << "0. Назад\n";
	std::cout << "1. Останні\n";
	std::cout << "2. Найперші\n";
	std::cout << "3. За індексом\n";
	choice = enterNumberInRange("Ваш вибір: ", 0, 3);
}
void wayToPasteTextMenu(int& choice) {
	std::cout << "\nЯк ви хочете вставити текст:\n";
	std::cout << "0. Вийти в Меню дій над змістом\n";
	std::cout << "1. В кінець\n";
	std::cout << "2. На початок\n";
	std::cout << "3. Ввести з клавіатури текст, який хочете замінити\n";
	choice = enterNumberInRange("Ваш вибір: ", 0, 3);
}
void delCopyOrCutTextMenu(int& choice, std::string action) {
	std::cout << "\nСкільки хочете " << action << ":\n";
	std::cout << "0. Назад\n";
	std::cout << "1. Весь зміст\n";
	std::cout << "2. Введу з клавіатури, що " << action << "\n";
	choice = enterNumberInRange("Ваш вибір: ", 0, 2);
}
void makeActionsOnContentMenu(int& choice) {
	std::cout << "\nМеню дій над змістом:\n";
	std::cout << "0. Назад\n";
	std::cout << "1. Додати текст\n";
	std::cout << "2. Видалити текст\n";
	std::cout << "3. Копіювати текст\n";
	std::cout << "4. Вирізати текст\n";
	std::cout << "5. Скасувати команду\n";
	std::cout << "6. Повторити команду\n";
	choice = enterNumberInRange("Ваш вибір: ", 0, 6);
}
void printGettingSessionsMenu(int& choice) {
	templateForMenusAboutSessions(choice, "отримати");
}
void printDeletingSessionsMenu(int& choice) {
	templateForMenusAboutSessions(choice, "видалити");
}
void printManagingSessionsMenu(int& choice) {
	system("cls");
	std::cout << "Головне меню:\n";
	std::cout << "0. Закрити програму\n";
	std::cout << "1. Довідка\n";
	std::cout << "2. Створити сеанс\n";
	std::cout << "3. Відкрити сеанс\n";
	std::cout << "4. Видалити сеанс\n";
	choice = enterNumberInRange("Ваш вибір: ", 0, 4);
}

std::string executeGettingTextForAdding() {
```

```
		std::string textToPaste;
		int choice;
		do
		{
			editor->printCurrentText();
			wayToGetTextForAddingMenu(choice);

			switch (choice)
			{
			case 0:
				std::cout << "\nПовернення до Меню дій над змістом.\n\n";
				system("pause");
				return "";
			case 1:
			case 2:
				textToPaste = choice == 1 ? getTextUsingKeyboard() :
getTextFromClipboard();

				if (textToPaste.empty())
					continue;
				else
					return textToPaste;
			}
		} while (true);
	}
	bool executeAddingTextToFile() {
		std::string textToPaste = executeGettingTextForAdding();
		if (textToPaste == "") return false;

		int choice;

		do {
			editor->printCurrentText();
			wayToPasteTextMenu(choice);
			switch (choice) {
			case 0:
				std::cout << "\nПовернення до Меню дій над змістом.\n\n";
				system("pause");
				return false;
			case 1:
				makeActionOnContextByEnteredText("Paste", "вставлені",
textToPaste, editor->getCurrentText()->size() - 1, editor->getCurrentText()->size() - 1);
				return true;
			case 2:
				makeActionOnContextByEnteredText("Paste", "вставлені",
textToPaste, 0, 0);
				return true;
			case 3:
				if (makeActionOnContextByEnteredText("Paste", "вставлені",
textToPaste))
					return true;
			}
		} while (true);
	}
	void executeMakeActionsOnContentMenu() {
		commandsManager = new CommandsManager(editor);
		bool wasTextSuccessfullyChanged;
		int choice;

		readDataFromFile();

		do
		{
			wasTextSuccessfullyChanged = false;
			editor->printCurrentText();
			makeActionsOnContentMenu(choice);
```

```
                    switch (choice)
                    {
                    case 0:
                          std::cout << "\nПовернення до Меню для отримання сеансу.\n\n";
                          system("pause");
                          delete (commandsManager);
                          return;
                    case 1:
                          wasTextSuccessfullyChanged = executeAddingTextToFile();
                          break;
                    case 2:
                          wasTextSuccessfullyChanged = chooseRootDelCopyOr-
Cut("видалити");
                          break;
                    case 3:
                          wasTextSuccessfullyChanged = chooseRootDelCopyOr-
Cut("скопіювати");
                          break;
                    case 4:
                          wasTextSuccessfullyChanged = chooseRootDelCopyOr-
Cut("вирізати");
                          break;
                    case 5:
                          wasTextSuccessfullyChanged = undoAction();
                          break;
                    case 6:
                          wasTextSuccessfullyChanged = redoAction();
                    }
                    if (wasTextSuccessfullyChanged)
                          FilesManager::writeSessionData(editor->getCurrentSession()-
>getName(), *(editor->getCurrentText()));
              } while (true);
       }
       void executeDeletingSessionsMenu() {
              std::function<void(int)> mainFunc = [this](int index) {
                    deleteSessionByIndex(index);
                    };
              std::function<void(int&)> menu = [this](int& choice) {
                    printDeletingSessionsMenu(choice);
                    };
              std::function<bool()> actionFuncByName = [this]() {
                    return deleteSessionByName();
                    };

              templateForExecutingMenusAboutSessions(mainFunc, menu, actionFuncByName);
       }
       void executeGettingSessionsMenu() {
              std::function<void(int&)> mainFunc = [this](int& index) {
                    setCurrentSessionByIndex(index);
                    };
              std::function<void(int&)> menu = [this](int& choice) {
                    printGettingSessionsMenu(choice);
                    };
              std::function<bool()> actionFuncByName = [this]() {
                    return setCurrentSessionByName();
                    };
              std::function<void()> additionalFunc = [this]() {
                    executeMakeActionsOnContentMenu();
                    };

              templateForExecutingMenusAboutSessions(mainFunc, menu, actionFuncByName, ad-
ditionalFunc);
       }
       bool executeDelCopyOrCutText(std::string typeOfCommand, std::string actionForMenu,
std::string actionInPast) {
```

```
		int choice;
		bool wasOperationSuccessful = false;

		editor->printCurrentText();
		delCopyOrCutTextMenu(choice, actionForMenu);

		switch (choice)
		{
		case 0:
			std::cout << "\nПовернення до Меню дій над змістом.\n\n";
			system("pause");
			return false;
		case 1:
			wasOperationSuccessful = makeActionOnContextByEn-
teredText(typeOfCommand, actionInPast, "", 0, editor->getCurrentText()->size() - 1);
			return wasOperationSuccessful;
		case 2:
			wasOperationSuccessful = makeActionOnContextByEn-
teredText(typeOfCommand, actionInPast);
			return wasOperationSuccessful;
		}
	}
	bool chooseRootDelCopyOrCut(std::string action) {
		if (editor->getCurrentText()->size() == 0)
		{
			printNotification("error", "немає тексту, який можна було б " + ac-
tion + "!");
			return false;
		}
		else
		{
			if (action == "видалити")
				return executeDelCopyOrCutText("Delete", action, "видалені");
			else if (action == "скопіювати")
				return executeDelCopyOrCutText("Copy", action, "скопійовані");
			else
				return executeDelCopyOrCutText("Cut", action, "вирізані");
		}
	}

	bool tryToEnterIndexForSession(int& index) {
		if (index == -1) {
			index = enterNumberInRange("Введіть номер сеансу: ", 1, editor-
>getSessionsHistory()->size());
			if (index == -1)
				return false;
		}
		return true;
	}
	bool doesAnySessionExist() {
		if (editor->getSessionsHistory()->isEmpty())
			printNotification("error", "в даний момент жодного сеансу немає!");
		return !editor->getSessionsHistory()->isEmpty();
	}

	void createSession() {
		std::string filename;

		std::cout << "\nВведіть ім'я сеансу (заборонені символи: /\\\":?*|<>): ";
		getline(std::cin, filename);

		Session* newSession = new Session();
		if (newSession->setName(filename))
		{
			if (editor->getSessionsHistory()->getSessionByName(filename) !=
nullptr) {
```

```
                        delete newSession;
                        printNotification("error", "сеанс з таким іменем вже існує!");
                        return;
                  }

                  if (!std::filesystem::exists(FilesManager::getSessionsDirectory()))

      std::filesystem::create_directories(FilesManager::getSessionsDirectory());

                  std::string filepath = FilesManager::getSessionsDirectory() + newS-
essionâ>getName();
                  std::ofstream file(filepath);
                  file.close();
                  editor->getSessionsHistory()->addSessionToEnd(newSession);
                  printNotification("success", "сеанс був успішно створений!");
            }
            else
            {
                  delete newSession;
                  printNotification("error", "були введені заборонені символи!");
            }
      }
      void setCurrentSessionByIndex(int& index) {
            if (tryToEnterIndexForSession(index))
                  editor->setCurrentSession(editor->getSessionsHistory()-
>getSessionByIndex(index - 1));
      }
      bool setCurrentSessionByName() {
            std::string name;
            std::cout << "\nВведіть ім'я сеансу: ";
            getline(std::cin, name);
            auto session = editor->getSessionsHistory()->getSessionByName(name);
            if (session == nullptr)
                  printNotification("error", "сеанса з таким іменем не існує!");
            else
                  editor->setCurrentSession(session);
            return session != nullptr;
      }
      void deleteSessionByIndex(int index = -1) {
            if (tryToEnterIndexForSession(index)) {
                  std::string nameOfSession = editor->getSessionsHistory()-
>deleteSessionByIndex(index - 1);
                  std::string pathToSession = FilesManager::getSessionsDirectory() +
nameOfSession;
                  remove(pathToSession.c_str());

                  printNotification("success", "сеанс був успішно видалений!");
            }
      }
      bool deleteSessionByName() {
            std::string name;
            std::cout << "\nВведіть ім'я сеансу: ";
            getline(std::cin, name);
            auto filename = editor->getSessionsHistory()->deleteSessionByName(name);
            if (filename.empty())
            {
                  printNotification("error", "сеанса з таким іменем не існує!");
                  return false;
            }
            std::string pathToSession = FilesManager::getSessionsDirectory() + filename;
            remove(pathToSession.c_str());

            printNotification("success", "сеанс був успішно видалений!");
            return true;
      }
public:
```

```
	void executeMainMenu() {
		int choice;
		editor = new Editor();
		editor->tryToLoadSessions();

		do
		{
			printManagingSessionsMenu(choice);
			switch (choice)
			{
			case 0:
				std::cout << "\nДо побачення!\n";
				editor->tryToUnloadSessions();
				delete editor;
				return;
			case 1:
				printReference();
				continue;
			case 2:
				createSession();
				continue;
			case 3:
			case 4:
				if (doesAnySessionExist())
					choice == 3 ? executeGettingSessionsMenu() :
					executeDeletingSessionsMenu();
			}

		} while (true);
	}
};

int main()
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	Program program;
	program.executeMainMenu();
}
```