**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

КУрСОВАЯ РАБОТА

**по дисциплине «Программирование»**

**Тема: Работа со списками.**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент гр. 4354 |  |  | И.В. Чучалин |
| Преподаватель |  |  | В.А. Калмычков |

# ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Чучалин И.В.

Группа 4354

Тема работы: Обработка текста.

Исходные данные: реализовать алгоритм на основе разбиения программы на набор модулей. Программа не должна содержать стандартных и библиотечных типов и связанных с ними функций. Должны быть учтены все составляющие формулировки задания и выполнены указанные действия, причем в качестве результата должны быть предложены все варианты, удовлетворяющие заданию. Программа должна предоставлять возможность неоднократного выполнения действий.

Содержание пояснительной записки:

Исходная формулировка задания, постановка задачи, описание алгоритма, текст программы, вывод программы, выводы.

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 17.02.25

Дата сдачи реферата:

Дата сдачи реферата:

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Чучалин И.В.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Калмычков В.А.

# АННОТАЦИЯ

Разработана программа на языке C++ для выполнения операций над текстом, представленным в виде последовательности отдельных предложений, содержащих слова и знаки пунктуации. Реализованы структуры данных для представления текста в виде иерархических списков, обеспечивающих эффективную обработку большого текста. Программа поддерживает ввод параметров из консоли, а также выполнение поиска подходящего предложения по последнему слову и проведение операций с этим предложением: вставки нового слова перед указанным, удаления указанного или всех знаков препинания по заданным пользователем командам. Программа протестирована на различных входных данных, демонстрируя корректность и устойчивость. Работа направлена на изучение алгоритмов обработки текста, хранящегося в виде иерархического списка, а также на развитие навыков проектирования программного обеспечения.

# SUMMARY

A C++ program has been developed to perform operations on text presented as a sequence of separate sentences containing words and punctuation marks. Data structures for representing text in the form of hierarchical lists are implemented, ensuring efficient processing of large text. The program supports entering parameters from the console, as well as searching for a suitable sentence by the last word and performing operations with this sentence: inserting a new word before the specified one, deleting the specified or all punctuation marks according to user-defined commands. The program has been tested on various input data, demonstrating correctness and stability. The work is aimed at studying algorithms for processing text stored in a hierarchical list, as well as developing software design skills.

# Оглавление

[ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ 2](#_Toc199683036)

[АННОТАЦИЯ 3](#_Toc199683037)

[SUMMARY 3](#_Toc199683038)

[Оглавление 4](#_Toc199683039)

[Исходная формулировка задания 4](#_Toc199683040)

[Постановка задачи 4](#_Toc199683041)

[Способ решения 4](#_Toc199683042)

[Организация интерфейса пользователя 5](#_Toc199683043)

[Организация входного файла 5](#_Toc199683044)

[Организация выходного файла 5](#_Toc199683045)

[Организация консоли 5](#_Toc199683046)

[Реализация ввода/вывода 5](#_Toc199683047)

[Модульное представление 5](#_Toc199683048)

[Внутреннее представление данных 6](#_Toc199683049)

[Описание внутренних функций 7](#_Toc199683050)

[Определение функций 7](#_Toc199683051)

[Назначение функций 8](#_Toc199683052)

[Описание алгоритма 9](#_Toc199683053)

[Общее описание алгоритма программы 9](#_Toc199683054)

[Принцип работы иерархического списка 9](#_Toc199683055)

[Блок-схемы 10](#_Toc199683056)

[Текст программы 15](#_Toc199683057)

[Результаты работы программы и контрольные примеры 23](#_Toc199683058)

[Выводы 29](#_Toc199683059)

# Исходная формулировка задания

Текст представляет собой последовательность отдельных предложений, содержащих слова и знаки пунктуации. Преобразовать текст в соответствии с последовательностью команд редактирования, которые должны позволять вставлять, удалять и заменять заданные слова в определенных предложениях.

Команды редактирования: вставить в предложении новое слово перед заданным словом; удалить в предложении знак пунктуации (указанный и/или все).

Указание определенного предложения: предложение, заканчивающееся на указанное слово.

Указание заданного слова: все в предложении, содержащее заданную последовательность символов.

# Постановка задачи

Дано

Документ с некоторым текстом, состоящим из предложений.

Задача

Обработать текст согласно командам пользователя.

# Способ решения

Перенести текст в сложный иерархический список, состоящий из блоков слов, пробелов, знаков пунктуации и проводить дальнейшую работу с ним.

# Организация интерфейса пользователя

## Организация входного файла

О1. Текст, представленный в виде строк.

## Организация выходного файла

Подробный:

О1. Изначальный текст.

О2. Найденные предложения.

О3. Произведенная операция.

О4. Измененный текст.

О5. Количество предложений.

О6. Последнее заданное слово.

Неподробный:

О1. Изначальный текст.

О2. Найденные предложения.

О3. Финальная версия текста.

## Организация консоли

О1. Вывод ошибок.

I1. Ввод слова для поиска предложения.

О2. Вывод подходящих предложений.

О3. Меню.

I2. Выбор функции.

I3. Ввод параметров.

О4. Вывод сообщения о завершении работы программы.

# Реализация ввода/вывода

Использованы потоки ввода (cin>>) и вывода (cout<<) библиотеки «iostream», а также функции объекта cin: getline, clear, eof.

Использованы потоки чтения из файла (>>) и записи в файл (<<) библиотеки «fstream», а также функции объекта типа fstream: open, is\_open, fail, seekg, eof, close. Для настройки ввода были использованы функции setiosflags, resetiosflags библиотеки «iomanip».

# Модульное представление

Изображение выглядит как текст, снимок экрана, Шрифт, Прямоугольник

Контент, сгенерированный ИИ, может содержать ошибки.

# Внутреннее представление данных

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Имя переменной** | | **Тип** | | **Назначение** | |
| c | | char | | Текущий символ из файла | |
| symbol | | Символ пунктуации | |
| lastWord | | char[100] | | Введённое слово | |
| char[256] | | Последнее слово предложения | |
| currentWord | | char[256] | | Текущее слово | |
| newWord | | char[100] | | Новое слово | |
| targetWord | | Целевое слово | |
| dest | | char\* | | Буфер назначения | |
| result | | Буфер для слова | |
| filename | | const char\* | | Имя входного файла | |
| newWord | | Новое слово | |
| str1 | | Первая строка | |
| str2 | | Вторая строка | |
| str | | Исходная строка | |
| substr | | Подстрока | |
| s | | Текущая позиция в str | |
| sub | | Текущая позиция в substr | |
| word | | Слово для поиска | |
| targetWord | | Целевое слово | |
| doc | | const Form\_V& | | Документ для печати | |
| Form\_V | | Документ | |
| Form\_V& | | Ссылка на документ для удаления | |
| found | | bool | | Флаг нахождения совпадений | |
| inWord | | Флаг нахождения внутри слова | |
| insertBeforeAll | | Флаг вставки перед всеми словами | |
| shouldInsert | | Флаг вставки | |
| validSentence | | Флаг корректности предложения | |
| choice | | int | | Выбор в меню | |
| cmpResult | | Результат сравнения | |
| currentSentence | | Идентификатор предложения | |
| i | | Счётчик | |
| len | | Длина слова | |
| level\_num | | Номер уровня | |
| matchCount | | Число совпадений | |
| sentenceId | | Идентификатор предложения | |
| sentenceNum | | Номер предложения | |
| spaceCount | | Счётчик пробелов | |
| targetLen | | Длина целевого слова | |
| newLen | | Длина нового слова | |
| maxLen | | Макс. длина копирования | |
| matchingSentences | | int\* | | Массив совпадений | |
| int[100] | |
| result | | int& | | Результат сравнения | |
| matchCount | | Число совпадений | |
| in | | std::ifstream | | Поток чтения файла | |
| conciseOut | | std::ofstream | | Файл краткого вывода | |
| std::ostream& | | Поток краткого вывода | |
| detailedOut | | std::ostream | | Файл детализированного вывода | |
| std::ostream& | | Поток детализированного вывода | |
|  | | | | | |
| **Тип** | **Структура/Поле** | | **Тип данных** | | **Назначение** |
| BlockType | - | | enum | | Тип блока |
| Comma | comma | | char | | Символ (;, ,, :) |
| EL\_Stroka | type | | BlockType | | Тип блока |
|  | content | | union | | Данные блока |
|  | sentence\_id | | int | | Идентификатор предложения |
|  | next | | EL\_Stroka\* | | Следующий блок в строке |
|  | next\_word\_block | | Следующая часть слова |
| EL\_V | line | | Line | | Строка (цепочка блоков) |
|  | next | | EL\_V\* | | Следующий уровень |
| Form\_V | head | | Начало списка уровней |
|  | total\_sentences | | int | | Общее число предложений |
| Letters | data[6] | | char[] | | Символы слова |
|  | size | | int | | Число символов |
|  | is\_word\_part | | bool | | Флаг части слова |
| Line | head | | EL\_Stroka\* | | Начало цепочки блоков |
| Punctuation | symbol | | char | | Символ (., !, ?) |
| Spaces | count | | int | | Число пробелов |

# Описание внутренних функций

## Определение функций

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Модуль** | **Имя функции** | **Возвращаемое значение** | **Входные параметры** | **Выходные параметры** | **Модифицируемые параметры** | **Транзитные параметры** |
| Del.cpp | DeleteDocument | void | Form\_V& doc | none | doc | None |
| InpS.cpp | ReadDocument | Form\_V& doc, const char\* filename | doc | filename |
| PrintDocument | const Form\_V& doc, std::ostream& out | None | doc, out |
| PrintDocumentConcise | const Form\_V& doc, std::ostream& out | None | doc, out |
| list.cpp | InitDocument | Form\_V& doc | doc | None |
| InitLevel | EL\_V\* level | level | None |
| AddLevelToDoc | Form\_V& doc, EL\_V\* level | doc | level |
| InitBlock | EL\_Stroka\*& block, BlockType type | block | type |
| AddBlockToLine | Line& line, EL\_Stroka\* block | line | block |
| listFunc.cpp | GetFullWord | EL\_Stroka\* block, char\* result | result | block |
| CompareStrings | const char\* str1, const char\* str2, int& result | result | str1, str2 |
| CopyString | char\* dest, const char\* src, int maxLen | dest | src, maxLen |
| IsSubstring | bool | const char\* str, const char\* substr | None | str, substr |
| main.cpp | ShowMenu | void | None | None | None |
| DisplayMatchingSentences | const Form\_V& doc, const char\* word, int\* matchingSentences, int& matchCount, std::ostream& detailedOut, std::ostream& conciseOut | matchingSentences, matchCount, detailedOut, conciseOut | doc, word |
| main | int | None | document, detailedOut, conciseOut, matchingSentences, matchCount, lastWord | None |
| Operate.cpp | InsertWord | void | Form\_V& doc, const char\* targetWord, const char\* newWord, int sentenceNum | doc | targetWord, newWord, sentenceNum |
| RemoveSpecificPunctuation | Form\_V& doc, int sentenceNum, char symbol, std::ostream& out | doc, out | sentenceNum, symbol |
| RemoveAllPunctuation | Form\_V& doc, int sentenceNum, std::ostream& out | doc, out | sentenceNum |
| PrintOperationResult | const Form\_V& doc, const char\* targetWord, const char\* newWord, int sentenceNum, std::ostream& out | None | doc, targetWord, newWord, sentenceNum, out |

## Назначение функций

**DeleteDocument (Del.cpp):**

Освобождает память, занятую документом, удаляя все уровни, блоки и их части.

**ReadDocument (InpS.cpp):**

Читает текст из файла, создает структуру документа, разбивая его на уровни и блоки (слова, пробелы, знаки препинания), присваивая идентификаторы предложениям.

**PrintDocument (InpS.cpp):**

Выводит подробное представление документа с уровнями, блоками и их содержимым в указанный поток.

**PrintDocumentConcise (InpS.cpp):**

Выводит краткое представление документа, отображая текст без детальной структуры.

**InitDocument (list.cpp):**

Инициализирует документ, устанавливая начальные значения для списка уровней и счетчика предложений.

**InitLevel (list.cpp):**

Инициализирует уровень документа, обнуляя указатели на строку и следующий уровень.

**AddLevelToDoc (list.cpp):**

Добавляет новый уровень в конец списка уровней документа.

**InitBlock (list.cpp):**

Создает и инициализирует блок (слово, пробелы, запятая, пунктуация) с заданным типом.

**AddBlockToLine (list.cpp):**

Добавляет блок в конец цепочки блоков строки.

**GetFullWord (listFunc.cpp):**

Собирает полное слово из цепочки блоков, записывая его в буфер.

**CompareStrings** (listFunc.cpp):

Сравнивает две строки, возвращая результат сравнения.

**CopyString (listFunc.cpp):**

Копирует строку в буфер с учетом максимальной длины.

**IsSubstring (listFunc.cpp):**

Проверяет, является ли одна строка подстрокой другой.

**ShowMenu (main.cpp):**

Выводит меню с опциями для взаимодействия с пользователем.

**DisplayMatchingSentences(main.cpp):**

Находит и отображает предложения, заканчивающиеся заданным словом, в подробном и кратком виде.

**main (main.cpp):**

Управляет программой, обеспечивая цикл взаимодействия с пользователем через меню, вызов операций и обработку ввода/вывода.

**InsertWord (Operate.cpp):**

Вставляет новое слово перед заданным словом или всеми словами в указанном предложении.

**RemoveSpecificPunctuation (Operate.cpp):**

Удаляет указанный символ пунктуации в заданном предложении и выводит результат.

**RemoveAllPunctuation (Operate.cpp):**

Удаляет все знаки препинания в заданном предложении и выводит результат.

**PrintOperationResult (Operate.cpp):**

Выводит результат операции вставки слова в документ.

# Описание алгоритма

## Общее описание алгоритма программы

Алгоритм, реализованный в предоставленных файлах C++, обрабатывает текстовый документ, разбивая его на уровни (строки) и блоки (слова, пробелы, знаки препинания), используя структуры данных `Form\_V`, `EL\_V`, `EL\_Stroka` с `union` для хранения различных типов данных. Основные функции включают чтение текста из файла, создание структуры документа с разделением на слова (до 5 символов в блоке, с цепочкой для длинных слов), пробелы, запятые и пунктуацию, а также управление предложениями с уникальными идентификаторами. Программа позволяет вставлять слова перед заданным словом или всеми словами в предложении, удалять определенные или все знаки препинания в указанном предложении, искать предложения по последнему слову и выводить документ в подробном или кратком формате. Пользователь взаимодействует через меню, выбирая операции и указывая параметры (слово, номер предложения, символ). Память освобождается при завершении работы.

## Принцип работы иерархического списка

Иерархический список в программе организован через структуры `Form\_V`, `EL\_V`, `Line` и `EL\_Stroka` для представления текста как многоуровневой структуры. `Form\_V` — корень документа, содержит указатель на список уровней (`head`) и счетчик предложений. Каждый уровень (`EL\_V`) представляет строку текста, включая цепочку блоков (`Line`) и указатель на следующий уровень (`next`). `Line` хранит указатель на начало цепочки блоков (`head`). `EL\_Stroka` — блок, содержащий тип (`BlockType`: слово, пробелы, запятая, пунктуация), данные в `union` (для экономии памяти), идентификатор предложения и указатели на следующий блок (`next`) и часть слова (`next\_word\_block`).

Текст разбивается на уровни (по строкам, разделенным `\n`), а каждая строка — на блоки: слова (до 5 символов, длинные слова разбиваются на цепочки блоков), пробелы, запятые и знаки пунктуации. Функции `InitDocument`, `InitLevel`, `InitBlock`, `AddLevelToDoc`, `AddBlockToLine` создают и связывают элементы списка. `ReadDocument` заполняет структуру, читая файл, а `DeleteDocument` освобождает память. Иерархия позволяет эффективно обрабатывать текст, поддерживая операции вставки, удаления и поиска, сохраняя структуру предложений и слов.

# Блок-схемы

InpS.cpp

ReadDocument

Изображение выглядит как диаграмма, оригами, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

PrintDocument PrintDocumentConcise

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как текст, диаграмма, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Del.cpp

deleteDocument

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

List.cpp

InitDocument InitLevel AddLevelToDoc AddBlockToLine

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.Изображение выглядит как текст, диаграмма, линия, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.Изображение выглядит как текст, диаграмма, линия, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

InitBlock

Изображение выглядит как текст, диаграмма, линия, План

Контент, сгенерированный ИИ, может содержать ошибки.

listFunc.cpp

GetFullWord CompareStrings CopyString

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как текст, диаграмма, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как текст, диаграмма, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

IsSubstring

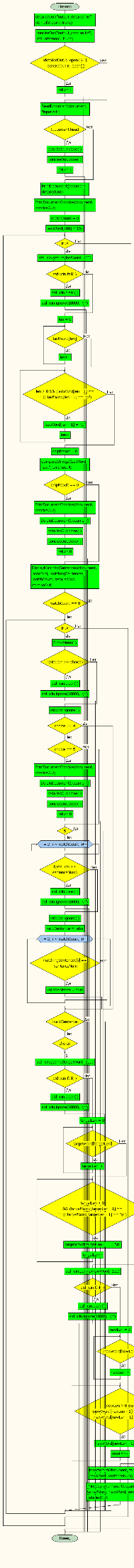
Изображение выглядит как текст, диаграмма, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Main.cpp

DisplayMatchingSentences main

Изображение выглядит как снимок экрана, Красочность, дизайн

Контент, сгенерированный ИИ, может содержать ошибки. 

ShowMenu Operate.cpp

PrintOperationResult

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как текст, Шрифт, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

InsertWord RemoveSpecificPunctuation RemoveAllPunctuation

Изображение выглядит как снимок экрана, диаграмма, Красочность, линия

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как текст, диаграмма, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки. Изображение выглядит как текст, диаграмма, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

# Текст программы

**Del.cpp**  
#include "Del.h"

#include "list.h"

void DeleteDocument(Form\_V& doc) {

EL\_V\* level = doc.head;

while (level) {

EL\_Stroka\* block = level->line.head;

while (block) {

EL\_Stroka\* nextBlock = block->next;

EL\_Stroka\* wordPart = block->next\_word\_block;

while (wordPart) {

EL\_Stroka\* nextPart = wordPart->next\_word\_block;

delete wordPart;

wordPart = nextPart;

}

delete block;

block = nextBlock;

}

EL\_V\* nextLevel = level->next;

delete level;

level = nextLevel;

}

doc.head = nullptr;

doc.total\_sentences = 0;

}

**InpS.cpp**

#include "Del.h"

#include "InpS.h"

#include "list.h"

#include <fstream>

#include <iostream>

#include <cstdlib>

void ReadDocument(Form\_V& doc, const char\* filename) {

InitDocument(doc);

std::ifstream in(filename);

if (!in.is\_open()) {

std::cerr << "Error: Could not open input file '" << filename << "'. Please ensure the file exists in the same directory as the executable.\n";

std::exit(1);

}

std::cout << "Successfully opened input file: " << filename << "\n";

EL\_V\* currentLevel = new EL\_V;

if (!currentLevel) {

std::cerr << "Error: Failed to allocate memory for document level.\n";

in.close();

std::exit(1);

}

InitLevel(currentLevel);

AddLevelToDoc(doc, currentLevel);

int sentenceId = 0;

char c;

bool inWord = false;

EL\_Stroka\* currentWordBlock = nullptr;

while (in.get(c)) {

if (c == '\n') {

EL\_V\* newLevel = new EL\_V;

if (!newLevel) {

std::cerr << "Error: Failed to allocate memory for new document level.\n";

in.close();

DeleteDocument(doc);

std::exit(1);

}

InitLevel(newLevel);

AddLevelToDoc(doc, newLevel);

currentLevel = newLevel;

continue;

}

if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {

if (!inWord) {

EL\_Stroka\* block = nullptr;

InitBlock(block, LETTERS);

if (!block) {

std::cerr << "Error: Failed to allocate memory for letter block.\n";

in.close();

DeleteDocument(doc);

std::exit(1);

}

block->sentence\_id = sentenceId;

AddBlockToLine(currentLevel->line, block);

currentWordBlock = block;

inWord = true;

}

int pos = currentWordBlock->content.letters.size;

if (pos < 5) {

currentWordBlock->content.letters.data[pos] = c;

currentWordBlock->content.letters.size++;

currentWordBlock->content.letters.data[pos + 1] = '\0';

}

else {

EL\_Stroka\* partBlock = nullptr;

InitBlock(partBlock, LETTERS);

if (!partBlock) {

std::cerr << "Error: Failed to allocate memory for letter part block.\n";

in.close();

DeleteDocument(doc);

std::exit(1);

}

partBlock->content.letters.is\_word\_part = true;

partBlock->sentence\_id = sentenceId;

partBlock->content.letters.data[0] = c;

partBlock->content.letters.size = 1;

partBlock->content.letters.data[1] = '\0';

currentWordBlock->next\_word\_block = partBlock;

currentWordBlock = partBlock;

}

}

else {

inWord = false;

currentWordBlock = nullptr;

if (c == ' ') {

EL\_Stroka\* block = nullptr;

InitBlock(block, SPACES);

if (!block) {

std::cerr << "Error: Failed to allocate memory for space block.\n";

in.close();

DeleteDocument(doc);

std::exit(1);

}

block->sentence\_id = sentenceId;

block->content.spaces.count = 1;

AddBlockToLine(currentLevel->line, block);

int spaceCount = 1;

while (in.peek() == ' ') {

in.get(c);

spaceCount++;

}

block->content.spaces.count = spaceCount;

}

else if (c == ',' || c == ';' || c == ':') {

EL\_Stroka\* block = nullptr;

InitBlock(block, COMMA);

if (!block) {

std::cerr << "Error: Failed to allocate memory for comma block.\n";

in.close();

DeleteDocument(doc);

std::exit(1);

}

block->sentence\_id = sentenceId;

block->content.comma.comma = c;

AddBlockToLine(currentLevel->line, block);

}

else if (c == '.' || c == '!' || c == '?') {

EL\_Stroka\* block = nullptr;

InitBlock(block, PUNCTUATION);

if (!block) {

std::cerr << "Error: Failed to allocate memory for punctuation block.\n";

in.close();

DeleteDocument(doc);

std::exit(1);

}

block->sentence\_id = sentenceId;

block->content.punctuation.symbol = c;

AddBlockToLine(currentLevel->line, block);

sentenceId++;

doc.total\_sentences = sentenceId;

}

}

}

in.close();

}

void PrintDocument(const Form\_V& doc, std::ostream& out) {

out << "\n===== Document =====\n";

int level\_num = 1;

for (EL\_V\* level = doc.head; level; level = level->next) {

out << "Level " << level\_num++ << ":\n";

EL\_Stroka\* block = level->line.head;

while (block) {

if (block->type == LETTERS && block->content.letters.is\_word\_part) {

block = block->next;

continue;

}

out << " ";

switch (block->type) {

case LETTERS: {

out << "[WORD: \"" << block->content.letters.data << "\"";

EL\_Stroka\* word\_part = block->next\_word\_block;

while (word\_part) {

out << " -> \"" << word\_part->content.letters.data << "\"";

word\_part = word\_part->next\_word\_block;

}

out << " (Sent: " << block->sentence\_id << ")]";

break;

}

case SPACES:

out << "[SPACE x" << block->content.spaces.count << "]";

break;

case COMMA:

out << "[COMMA: '" << block->content.comma.comma << "']";

break;

case PUNCTUATION:

out << "[PUNCT: '" << block->content.punctuation.symbol << "']";

break;

}

out << "\n";

block = block->next;

}

out << "----------------\n";

}

out << "Total sentences: " << doc.total\_sentences << "\n";

out << "====================\n\n";

}

void PrintDocumentConcise(const Form\_V& doc, std::ostream& out) {

if (!doc.head) {

out << "Document is empty.\n";

return;

}

EL\_V\* level = doc.head;

while (level) {

if (level->line.head) {

EL\_Stroka\* block = level->line.head;

while (block) {

if (block->type == LETTERS) {

out << block->content.letters.data;

EL\_Stroka\* part = block->next\_word\_block;

while (part) {

out << part->content.letters.data;

part = part->next\_word\_block;

}

}

else if (block->type == SPACES) {

for (int i = 0; i < block->content.spaces.count; i++) {

out << ' ';

}

}

else if (block->type == COMMA) {

out << block->content.comma.comma;

}

else if (block->type == PUNCTUATION) {

out << block->content.punctuation.symbol;

}

block = block->next;

}

out << "\n";

}

level = level->next;

}

}

**List.cpp**

#include "list.h"

void InitDocument(Form\_V& doc) {

doc.head = nullptr;

doc.total\_sentences = 0;

}

void InitLevel(EL\_V\* level) {

if (!level) {

return;

}

level->line.head = nullptr;

level->next = nullptr;

}

void AddLevelToDoc(Form\_V& doc, EL\_V\* level) {

if (!level) return;

if (!doc.head) {

doc.head = level;

}

else {

EL\_V\* current = doc.head;

while (current->next) {

current = current->next;

}

current->next = level;

}

}

void InitBlock(EL\_Stroka\*& block, BlockType type) {

block = new EL\_Stroka;

if (!block) return;

block->type = type;

block->sentence\_id = 0;

block->next = nullptr;

block->next\_word\_block = nullptr;

if (type == LETTERS) {

block->content.letters.size = 0;

block->content.letters.is\_word\_part = false;

for (int i = 0; i < 6; i++) {

block->content.letters.data[i] = '\0';

}

}

else if (type == SPACES) {

block->content.spaces.count = 1;

}

else if (type == COMMA) {

block->content.comma.comma = ',';

}

else if (type == PUNCTUATION) {

block->content.punctuation.symbol = '\0';

}

}

void AddBlockToLine(Line& line, EL\_Stroka\* block) {

if (!block) return;

if (!line.head) {

line.head = block;

}

else {

EL\_Stroka\* current = line.head;

while (current->next) {

current = current->next;

}

current->next = block;

}

}

**listFunc.cpp**

#include "listFunc.h"

#include "list.h"

void GetFullWord(EL\_Stroka\* block, char\* result) {

if (!block || !result) {

if (result) result[0] = '\0';

return;

}

int pos = 0;

EL\_Stroka\* current = block;

while (current && pos < 255) {

int i = 0;

while (i < current->content.letters.size && pos < 255) {

result[pos++] = current->content.letters.data[i++];

}

current = current->next\_word\_block;

}

result[pos] = '\0';

}

void CompareStrings(const char\* str1, const char\* str2, int& result) {

result = 0;

if (!str1 || !str2) {

result = (str1 ? 1 : (str2 ? -1 : 0));

return;

}

while (\*str1 && \*str2) {

if (\*str1 != \*str2) {

result = \*str1 - \*str2;

return;

}

str1++;

str2++;

}

if (\*str1 != \*str2) {

result = \*str1 ? 1 : -1;

}

}

void CopyString(char\* dest, const char\* src, int maxLen) {

if (!dest || !src) return;

int i = 0;

while (i < maxLen - 1 && src[i]) {

dest[i] = src[i];

i++;

}

dest[i] = '\0';

}

bool IsSubstring(const char\* str, const char\* substr) {

if (!str || !substr) return false;

if (!\*substr) return true;

while (\*str) {

const char\* s = str;

const char\* sub = substr;

while (\*s && \*sub && \*s == \*sub) {

s++;

sub++;

}

if (!\*sub) return true;

str++;

}

return false;

}

**Main.cpp**

#include "listFunc.h"

#include "InpS.h"

#include "Del.h"

#include "Operate.h"

#include <fstream>

#include <iostream>

void ShowMenu() {

std::cout << "\n======= Menu =======\n"

<< "1. Insert word in matching sentence\n"

<< "2. Remove specific punctuation in matching sentence\n"

<< "3. Remove all punctuation in matching sentence\n"

<< "4. Find new sentences by last word\n"

<< "5. Exit\n"

<< "====================\n"

<< "Choose option: ";

}

void DisplayMatchingSentences(const Form\_V& doc, const char\* word, int\* matchingSentences, int& matchCount, std::ostream& detailedOut, std::ostream& conciseOut) {

detailedOut << "\n=== Sentences ending with word \"" << word << "\" ===\n";

conciseOut << "\n=== Sentences ending with word \"" << word << "\" ===\n";

std::cout << "\n=== Sentences ending with word \"" << word << "\" ===\n";

std::cout << std::flush;

matchCount = 0;

char lastWord[256] = { 0 };

int currentSentence = -1;

EL\_Stroka\* lastWordBlock = nullptr;

bool found = false;

int cmpResult = 0;

for (EL\_V\* level = doc.head; level; level = level->next) {

if (!level || !level->line.head) continue;

EL\_Stroka\* block = level->line.head;

while (block) {

if (block->sentence\_id != currentSentence) {

if (currentSentence != -1 && lastWordBlock && lastWord[0] != '\0') {

CompareStrings(lastWord, word, cmpResult);

if (cmpResult == 0) {

if (matchCount < 100) {

matchingSentences[matchCount++] = currentSentence;

}

detailedOut << "Sentence " << currentSentence << ":\n";

conciseOut << "Sentence " << currentSentence << ": ";

std::cout << "Sentence " << currentSentence << ": ";

for (EL\_V\* printLevel = doc.head; printLevel; printLevel = printLevel->next) {

EL\_Stroka\* printBlock = printLevel->line.head;

while (printBlock) {

if (printBlock->sentence\_id == currentSentence) {

if (printBlock->type == LETTERS && !printBlock->content.letters.is\_word\_part) {

EL\_Stroka\* wordPart = printBlock;

while (wordPart) {

std::cout << wordPart->content.letters.data;

conciseOut << wordPart->content.letters.data;

wordPart = wordPart->next\_word\_block;

}

detailedOut << "[WORD: \"" << printBlock->content.letters.data;

wordPart = printBlock->next\_word\_block;

while (wordPart) {

detailedOut << " -> \"" << wordPart->content.letters.data << "\"";

wordPart = wordPart->next\_word\_block;

}

detailedOut << "\" (Sent: " << printBlock->sentence\_id << ")]\n";

}

else if (printBlock->type == SPACES) {

for (int i = 0; i < printBlock->content.spaces.count; i++) {

std::cout << ' ';

conciseOut << ' ';

}

detailedOut << "[SPACE x" << printBlock->content.spaces.count << "]\n";

}

else if (printBlock->type == COMMA) {

char symbol = printBlock->content.comma.comma;

std::cout << symbol;

conciseOut << symbol;

detailedOut << "[COMMA: '" << symbol << "']\n";

}

else if (printBlock->type == PUNCTUATION) {

char symbol = printBlock->content.punctuation.symbol;

std::cout << symbol;

conciseOut << symbol;

detailedOut << "[PUNCT: '" << symbol << "']\n";

}

}

printBlock = printBlock->next;

}

}

std::cout << "\n";

conciseOut << "\n";

detailedOut << "----------------\n";

found = true;

}

}

currentSentence = block->sentence\_id;

lastWord[0] = '\0';

lastWordBlock = nullptr;

}

if (block->type == LETTERS && !block->content.letters.is\_word\_part) {

lastWordBlock = block;

GetFullWord(block, lastWord);

}

block = block->next;

}

}

if (currentSentence != -1 && lastWordBlock && lastWord[0] != '\0') {

CompareStrings(lastWord, word, cmpResult);

if (cmpResult == 0) {

if (matchCount < 100) {

matchingSentences[matchCount++] = currentSentence;

}

detailedOut << "Sentence " << currentSentence << ":\n";

conciseOut << "Sentence " << currentSentence << ": ";

std::cout << "Sentence " << currentSentence << ": ";

for (EL\_V\* printLevel = doc.head; printLevel; printLevel = printLevel->next) {

EL\_Stroka\* printBlock = printLevel->line.head;

while (printBlock) {

if (printBlock->sentence\_id == currentSentence) {

if (printBlock->type == LETTERS && !printBlock->content.letters.is\_word\_part) {

EL\_Stroka\* wordPart = printBlock;

while (wordPart) {

std::cout << wordPart->content.letters.data;

conciseOut << wordPart->content.letters.data;

wordPart = wordPart->next\_word\_block;

}

detailedOut << "[WORD: \"" << printBlock->content.letters.data;

wordPart = printBlock->next\_word\_block;

while (wordPart) {

detailedOut << " -> \"" << wordPart->content.letters.data << "\"";

wordPart = wordPart->next\_word\_block;

}

detailedOut << "\" (Sent: " << printBlock->sentence\_id << ")]\n";

}

else if (printBlock->type == SPACES) {

for (int i = 0; i < printBlock->content.spaces.count; i++) {

std::cout << ' ';

conciseOut << ' ';

}

detailedOut << "[SPACE x" << printBlock->content.spaces.count << "]\n";

}

else if (printBlock->type == COMMA) {

char symbol = printBlock->content.comma.comma;

std::cout << symbol;

conciseOut << symbol;

detailedOut << "[COMMA: '" << symbol << "']\n";

}

else if (printBlock->type == PUNCTUATION) {

char symbol = printBlock->content.punctuation.symbol;

std::cout << symbol;

conciseOut << symbol;

detailedOut << "[PUNCT: '" << symbol << "']\n";

}

}

printBlock = printBlock->next;

}

}

std::cout << "\n";

conciseOut << "\n";

detailedOut << "----------------\n";

found = true;

}

}

if (!found) {

detailedOut << "No sentences with last word \"" << word << "\" found.\n";

conciseOut << "No sentences with last word \"" << word << "\" found.\n";

std::cout << "No sentences with last word \"" << word << "\" found.\n" << std::flush;

}

else {

std::cout << "Matching sentences: ";

for (int i = 0; i < matchCount; i++) {

std::cout << matchingSentences[i] << " ";

}

std::cout << "\n" << std::flush;

}

}

int main() {

Form\_V document;

std::ofstream detailedOut("output\_detailed.txt", std::ofstream::trunc);

std::ofstream conciseOut("output\_concise.txt", std::ofstream::trunc);

if (!detailedOut.is\_open() || !conciseOut.is\_open()) {

std::cerr << "Error: Unable to open output files!\n";

return 1;

}

ReadDocument(document, "input.txt");

if (!document.head) {

std::cerr << "Error: Failed to read input file or file is empty!\n";

detailedOut.close();

conciseOut.close();

return 1;

}

detailedOut << "=== Initial Document ===\n";

PrintDocument(document, detailedOut);

conciseOut << "=== Initial Document ===\n";

PrintDocumentConcise(document, conciseOut);

int matchingSentences[100];

int matchCount = 0;

char lastWord[100] = { 0 };

while (true) {

std::cout << "\nEnter last word to find sentences (or 'exit' to quit, max 99 characters): ";

std::cout << std::flush;

std::cin.getline(lastWord, 100);

if (std::cin.fail()) {

std::cerr << "Error: Invalid input. Please try again.\n";

std::cin.clear();

std::cin.ignore(10000, '\n');

continue;

}

int len = 0;

while (lastWord[len]) len++;

while (len > 0 && (lastWord[len - 1] == ' ' || lastWord[len - 1] == '\n')) {

lastWord[len - 1] = '\0';

len--;

}

int cmpResult = 0;

CompareStrings(lastWord, "exit", cmpResult);

if (cmpResult == 0) {

std::cout << "Exiting...\n" << std::flush;

conciseOut << "\n=== Final Document ===\n";

PrintDocumentConcise(document, conciseOut);

DeleteDocument(document);

detailedOut.close();

conciseOut.close();

return 0;

}

DisplayMatchingSentences(document, lastWord, matchingSentences, matchCount, detailedOut, conciseOut);

if (matchCount == 0) {

std::cout << "Please try another word.\n" << std::flush;

continue;

}

while (true) {

ShowMenu();

std::cout << std::flush;

int choice;

if (!(std::cin >> choice)) {

std::cerr << "Invalid input! Please enter a number.\n";

std::cin.clear();

std::cin.ignore(10000, '\n');

continue;

}

std::cin.ignore();

if (choice == 4) {

break;

}

if (choice == 5) {

std::cout << "Exiting...\n" << std::flush;

conciseOut << "\n=== Final Document ===\n";

PrintDocumentConcise(document, conciseOut);

DeleteDocument(document);

detailedOut.close();

conciseOut.close();

return 0;

}

if (choice < 1 || choice > 5) {

std::cout << "Invalid option!\n" << std::flush;

continue;

}

int sentenceNum;

std::cout << "Enter sentence number (from matching sentences: ";

for (int i = 0; i < matchCount; i++) {

std::cout << matchingSentences[i] << " ";

}

std::cout << "): " << std::flush;

if (!(std::cin >> sentenceNum)) {

std::cerr << "Invalid sentence number!\n";

std::cin.clear();

std::cin.ignore(10000, '\n');

continue;

}

std::cin.ignore();

bool validSentence = false;

for (int i = 0; i < matchCount; i++) {

if (matchingSentences[i] == sentenceNum) {

validSentence = true;

break;

}

}

if (!validSentence) {

std::cerr << "Error: Sentence " << sentenceNum << " is not in the matching list!\n";

continue;

}

switch (choice) {

case 1: {

char targetWord[100];

std::cout << "Enter target word (or 'all' for all words, max 99 characters): " << std::flush;

std::cin.getline(targetWord, 100);

if (std::cin.fail()) {

std::cerr << "Error: Invalid input. Please try again.\n";

std::cin.clear();

std::cin.ignore(10000, '\n');

continue;

}

int targetLen = 0;

while (targetWord[targetLen]) targetLen++;

while (targetLen > 0 && (targetWord[targetLen - 1] == ' ' || targetWord[targetLen - 1] == '\n')) {

targetWord[targetLen - 1] = '\0';

targetLen--;

}

char newWord[100];

std::cout << "Enter new word (max 99 characters): " << std::flush;

std::cin.getline(newWord, 100);

if (std::cin.fail()) {

std::cerr << "Error: Invalid input. Please try again.\n";

std::cin.clear();

std::cin.ignore(10000, '\n');

continue;

}

int newLen = 0;

while (newWord[newLen]) newLen++;

while (newLen > 0 && (newWord[newLen - 1] == ' ' || newWord[newLen - 1] == '\n')) {

newWord[newLen - 1] = '\0';

newLen--;

}

InsertWord(document, targetWord, newWord, sentenceNum);

detailedOut << "\n=== Insert Word Operation ===\n";

PrintOperationResult(document, targetWord, newWord, sentenceNum, detailedOut);

break;

}

case 2: {

char symbol;

std::cout << "Enter punctuation symbol to remove: " << std::flush;

std::cin >> symbol;

std::cin.ignore();

RemoveSpecificPunctuation(document, sentenceNum, symbol, detailedOut);

detailedOut << "Last word searched: " << lastWord << "\n";

break;

}

case 3: {

RemoveAllPunctuation(document, sentenceNum, detailedOut);

detailedOut << "Last word searched: " << lastWord << "\n";

break;

}

}

}

}

DeleteDocument(document);

detailedOut.close();

conciseOut.close();

return 0;

}

**Operate.cpp**

#include "Operate.h"

#include "list.h"

#include "listFunc.h"

#include "inpS.h"

#include <fstream>

#include <iostream>

void InsertWord(Form\_V& doc, const char\* targetWord, const char\* newWord, int sentenceNum) {

if (!targetWord || !newWord || !doc.head) return;

int cmpResult = 0;

bool insertBeforeAll = false;

CompareStrings(targetWord, "all", cmpResult);

if (cmpResult == 0) insertBeforeAll = true;

EL\_V\* level = doc.head;

while (level) {

if (!level->line.head) {

level = level->next;

continue;

}

EL\_Stroka\* block = level->line.head;

EL\_Stroka\* prev = nullptr;

while (block) {

EL\_Stroka\* nextBlock = block ? block->next : nullptr;

if (!block || block->sentence\_id != sentenceNum) {

prev = block;

block = nextBlock;

continue;

}

if (block->type == LETTERS && !block->content.letters.is\_word\_part) {

char currentWord[256] = { 0 };

GetFullWord(block, currentWord);

bool shouldInsert = false;

if (insertBeforeAll) {

shouldInsert = true;

}

else {

if (IsSubstring(currentWord, targetWord)) {

shouldInsert = true;

}

}

if (shouldInsert) {

EL\_Stroka\* newBlock = nullptr;

InitBlock(newBlock, LETTERS);

if (!newBlock) {

std::cerr << "Error: Failed to allocate memory for new word block.\n";

return;

}

newBlock->sentence\_id = sentenceNum;

int i = 0;

while (newWord[i] && i < 5) {

newBlock->content.letters.data[i] = newWord[i];

newBlock->content.letters.size++;

i++;

}

newBlock->content.letters.data[i] = '\0';

const char\* remaining = newWord + i;

EL\_Stroka\* currentPart = newBlock;

while (\*remaining) {

EL\_Stroka\* partBlock = nullptr;

InitBlock(partBlock, LETTERS);

if (!partBlock) {

std::cerr << "Error: Failed to allocate memory for word part block.\n";

EL\_Stroka\* temp = newBlock;

while (temp) {

EL\_Stroka\* next = temp->next\_word\_block;

delete temp;

temp = next;

}

return;

}

partBlock->content.letters.is\_word\_part = true;

partBlock->sentence\_id = sentenceNum;

i = 0;

while (\*remaining && i < 5) {

partBlock->content.letters.data[i] = \*remaining;

partBlock->content.letters.size++;

remaining++;

i++;

}

partBlock->content.letters.data[i] = '\0';

currentPart->next\_word\_block = partBlock;

currentPart = partBlock;

}

EL\_Stroka\* spaceAfter = nullptr;

InitBlock(spaceAfter, SPACES);

if (!spaceAfter) {

std::cerr << "Error: Failed to allocate memory for space after block.\n";

EL\_Stroka\* temp = newBlock;

while (temp) {

EL\_Stroka\* next = temp->next\_word\_block;

delete temp;

temp = next;

}

return;

}

spaceAfter->sentence\_id = sentenceNum;

spaceAfter->content.spaces.count = 1;

newBlock->next = spaceAfter;

spaceAfter->next = block;

if (prev) {

prev->next = newBlock;

}

else {

level->line.head = newBlock;

}

prev = spaceAfter;

}

else {

prev = block;

}

}

else {

prev = block;

}

block = nextBlock;

}

level = level->next;

}

}

void RemoveSpecificPunctuation(Form\_V& doc, int sentenceNum, char symbol, std::ostream& out) {

for (EL\_V\* level = doc.head; level; level = level->next) {

if (!level || !level->line.head) continue;

EL\_Stroka\* block = level->line.head;

EL\_Stroka\* prev = nullptr;

while (block) {

EL\_Stroka\* nextBlock = block->next;

if (block->sentence\_id != sentenceNum) {

prev = block;

block = nextBlock;

continue;

}

if ((block->type == PUNCTUATION && block->content.punctuation.symbol == symbol) ||

(block->type == COMMA && block->content.comma.comma == symbol)) {

if (prev) {

prev->next = block->next;

}

else {

level->line.head = block->next;

}

delete block;

block = nextBlock;

continue;

}

prev = block;

block = nextBlock;

}

}

out << "\n=== Remove Specific Punctuation Operation ===\n";

out << "Removed symbol: " << symbol << "\nSentence: " << sentenceNum << "\n";

PrintDocument(doc, out);

}

void RemoveAllPunctuation(Form\_V& doc, int sentenceNum, std::ostream& out) {

for (EL\_V\* level = doc.head; level; level = level->next) {

if (!level || !level->line.head) continue;

EL\_Stroka\* block = level->line.head;

EL\_Stroka\* prev = nullptr;

while (block) {

EL\_Stroka\* nextBlock = block->next;

if (block->sentence\_id != sentenceNum) {

prev = block;

block = nextBlock;

continue;

}

if (block->type == PUNCTUATION || block->type == COMMA) {

if (prev) {

prev->next = block->next;

}

else {

level->line.head = block->next;

}

delete block;

block = nextBlock;

continue;

}

prev = block;

block = nextBlock;

}

}

out << "\n=== Remove All Punctuation Operation ===\n";

out << "Sentence: " << sentenceNum << "\n";

PrintDocument(doc, out);

}

void PrintOperationResult(const Form\_V& doc, const char\* targetWord, const char\* newWord, int sentenceNum, std::ostream& out) {

out << "Inserted word \"" << newWord << "\" before \"" << targetWord << "\" in sentence " << sentenceNum << "\n";

PrintDocument(doc, out);

}

**Del.h**

#ifndef DEL\_H

#define DEL\_H

#include "list.h"

void DeleteDocument(Form\_V& doc);

#endif

**InpS.h**

#ifndef INPS\_H

#define INPS\_H

#include "list.h"

#include <iostream>

#include <fstream>

void ReadDocument(Form\_V& doc, const char\* filename);

void PrintDocument(const Form\_V& doc, std::ostream& out);

void PrintDocumentConcise(const Form\_V& doc, std::ostream& out);

#endif

**List.h**

#ifndef LIST\_H

#define LIST\_H

enum BlockType { LETTERS, SPACES, COMMA, PUNCTUATION };

struct Letters {

char data[6];

int size;

bool is\_word\_part;

};

struct Spaces {

int count;

};

struct Comma {

char comma;

};

struct Punctuation {

char symbol;

};

struct EL\_Stroka {

BlockType type;

union {

Letters letters;

Spaces spaces;

Comma comma;

Punctuation punctuation;

} content;

int sentence\_id;

EL\_Stroka\* next;

EL\_Stroka\* next\_word\_block;

};

struct Line {

EL\_Stroka\* head;

};

struct EL\_V {

Line line;

EL\_V\* next;

};

struct Form\_V {

EL\_V\* head;

int total\_sentences;

};

void InitDocument(Form\_V& doc);

void InitLevel(EL\_V\* level);

void AddLevelToDoc(Form\_V& doc, EL\_V\* level);

void InitBlock(EL\_Stroka\*& block, BlockType type);

void AddBlockToLine(Line& line, EL\_Stroka\* block);

#endif

**listFunc.cpp**

#ifndef LISTFUNC\_H

#define LISTFUNC\_H

#include "list.h"

void GetFullWord(EL\_Stroka\* block, char\* result);

void CompareStrings(const char\* str1, const char\* str2, int& result);

void CopyString(char\* dest, const char\* src, int maxLen);

bool IsSubstring(const char\* str, const char\* substr);

#endif

**Operate.H**

#ifndef OPERATE\_H

#define OPERATE\_H

#include "list.h"

#include <iostream>

void InsertWord(Form\_V& doc, const char\* targetWord, const char\* newWord, int sentenceNum);

void RemoveSpecificPunctuation(Form\_V& doc, int sentenceNum, char symbol, std::ostream& out);

void RemoveAllPunctuation(Form\_V& doc, int sentenceNum, std::ostream& out);

void PrintOperationResult(const Form\_V& doc, const char\* targetWord, const char\* newWord, int sentenceNum, std::ostream& out);

void DisplayMatchingSentences(const Form\_V& doc, const char\* word, int\* matchingSentences, int& matchCount, std::ostream& detailedOut, std::ostream& conciseOut);

#endif

# Результаты работы программы и контрольные примеры

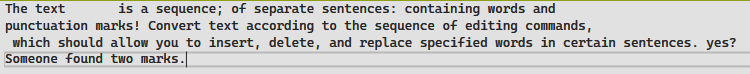
Были обработаны все возможные вариации исполнения программы.

Консоль:

Изображение выглядит как текст, снимок экрана, письмо, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Входной файл:



Выходной подробный файл: Выходной неподробный файл:

=== Initial Document ===

===== Document =====

Level 1:

[WORD: "The" (Sent: 0)]

[SPACE x1]

[WORD: "text" (Sent: 0)]

[SPACE x7]

[WORD: "is" (Sent: 0)]

[SPACE x1]

[WORD: "a" (Sent: 0)]

[SPACE x1]

[WORD: "seque" -> "nce" (Sent: 0)]

[COMMA: ';']

[SPACE x1]

[WORD: "of" (Sent: 0)]

[SPACE x1]

[WORD: "separ" -> "ate" (Sent: 0)]

[SPACE x1]

[WORD: "sente" -> "nces" (Sent: 0)]

[COMMA: ':']

[SPACE x1]

[WORD: "conta" -> "ining" (Sent: 0)]

[SPACE x1]

[WORD: "words" (Sent: 0)]

[SPACE x1]

[WORD: "and" (Sent: 0)]

[SPACE x1]

----------------

Изображение выглядит как текст, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.Level 2:

[WORD: "punct" -> "uatio" -> "n" (Sent: 0)]

[SPACE x1]

[WORD: "marks" (Sent: 0)]

[PUNCT: '!']

[SPACE x1]

[WORD: "Conve" -> "rt" (Sent: 1)]

[SPACE x1]

[WORD: "text" (Sent: 1)]

[SPACE x1]

[WORD: "accor" -> "ding" (Sent: 1)]

[SPACE x1]

[WORD: "to" (Sent: 1)]

[SPACE x1]

[WORD: "the" (Sent: 1)]

[SPACE x1]

[WORD: "seque" -> "nce" (Sent: 1)]

[SPACE x1]

[WORD: "of" (Sent: 1)]

[SPACE x1]

[WORD: "editi" -> "ng" (Sent: 1)]

[SPACE x1]

[WORD: "comma" -> "nds" (Sent: 1)]

[COMMA: ',']

----------------

Level 3:

[SPACE x1]

[WORD: "which" (Sent: 1)]

[SPACE x1]

[WORD: "shoul" -> "d" (Sent: 1)]

[SPACE x1]

[WORD: "allow" (Sent: 1)]

[SPACE x1]

[WORD: "you" (Sent: 1)]

[SPACE x1]

[WORD: "to" (Sent: 1)]

[SPACE x1]

[WORD: "inser" -> "t" (Sent: 1)]

[COMMA: ',']

[SPACE x1]

[WORD: "delet" -> "e" (Sent: 1)]

[COMMA: ',']

[SPACE x1]

[WORD: "and" (Sent: 1)]

[SPACE x1]

[WORD: "repla" -> "ce" (Sent: 1)]

[SPACE x1]

[WORD: "speci" -> "fied" (Sent: 1)]

[SPACE x1]

[WORD: "words" (Sent: 1)]

[SPACE x1]

[WORD: "in" (Sent: 1)]

[SPACE x1]

[WORD: "certa" -> "in" (Sent: 1)]

[SPACE x1]

[WORD: "sente" -> "nces" (Sent: 1)]

[PUNCT: '.']

[SPACE x1]

[WORD: "yes" (Sent: 2)]

[PUNCT: '?']

----------------

Level 4:

[WORD: "Someo" -> "ne" (Sent: 3)]

[SPACE x1]

[WORD: "found" (Sent: 3)]

[SPACE x1]

[WORD: "two" (Sent: 3)]

[SPACE x1]

[WORD: "marks" (Sent: 3)]

[PUNCT: '.']

----------------

Total sentences: 4

====================

=== Sentences ending with word "marks" ===

Sentence 0:

[WORD: "The" (Sent: 0)]

[SPACE x1]

[WORD: "text" (Sent: 0)]

[SPACE x7]

[WORD: "is" (Sent: 0)]

[SPACE x1]

[WORD: "a" (Sent: 0)]

[SPACE x1]

[WORD: "seque -> "nce"" (Sent: 0)]

[COMMA: ';']

[SPACE x1]

[WORD: "of" (Sent: 0)]

[SPACE x1]

[WORD: "separ -> "ate"" (Sent: 0)]

[SPACE x1]

[WORD: "sente -> "nces"" (Sent: 0)]

[COMMA: ':']

[SPACE x1]

[WORD: "conta -> "ining"" (Sent: 0)]

[SPACE x1]

[WORD: "words" (Sent: 0)]

[SPACE x1]

[WORD: "and" (Sent: 0)]

[SPACE x1]

[WORD: "punct -> "uatio" -> "n"" (Sent: 0)]

[SPACE x1]

[WORD: "marks" (Sent: 0)]

[PUNCT: '!']

----------------

Sentence 3:

[WORD: "Someo -> "ne"" (Sent: 3)]

[SPACE x1]

[WORD: "found" (Sent: 3)]

[SPACE x1]

[WORD: "two" (Sent: 3)]

[SPACE x1]

[WORD: "marks" (Sent: 3)]

[PUNCT: '.']

----------------

=== Insert Word Operation ===

Inserted word "MARKS" before "marks" in sentence 0

===== Document =====

Level 1:

[WORD: "The" (Sent: 0)]

[SPACE x1]

[WORD: "text" (Sent: 0)]

[SPACE x7]

[WORD: "is" (Sent: 0)]

[SPACE x1]

[WORD: "a" (Sent: 0)]

[SPACE x1]

[WORD: "seque" -> "nce" (Sent: 0)]

[COMMA: ';']

[SPACE x1]

[WORD: "of" (Sent: 0)]

[SPACE x1]

[WORD: "separ" -> "ate" (Sent: 0)]

[SPACE x1]

[WORD: "sente" -> "nces" (Sent: 0)]

[COMMA: ':']

[SPACE x1]

[WORD: "conta" -> "ining" (Sent: 0)]

[SPACE x1]

[WORD: "words" (Sent: 0)]

[SPACE x1]

[WORD: "and" (Sent: 0)]

[SPACE x1]

----------------

Level 2:

[WORD: "punct" -> "uatio" -> "n" (Sent: 0)]

[SPACE x1]

[WORD: "MARKS" (Sent: 0)]

[SPACE x1]

[WORD: "marks" (Sent: 0)]

[PUNCT: '!']

[SPACE x1]

[WORD: "Conve" -> "rt" (Sent: 1)]

[SPACE x1]

[WORD: "text" (Sent: 1)]

[SPACE x1]

[WORD: "accor" -> "ding" (Sent: 1)]

[SPACE x1]

[WORD: "to" (Sent: 1)]

[SPACE x1]

[WORD: "the" (Sent: 1)]

[SPACE x1]

[WORD: "seque" -> "nce" (Sent: 1)]

[SPACE x1]

[WORD: "of" (Sent: 1)]

[SPACE x1]

[WORD: "editi" -> "ng" (Sent: 1)]

[SPACE x1]

[WORD: "comma" -> "nds" (Sent: 1)]

[COMMA: ',']

----------------

Level 3:

[SPACE x1]

[WORD: "which" (Sent: 1)]

[SPACE x1]

[WORD: "shoul" -> "d" (Sent: 1)]

[SPACE x1]

[WORD: "allow" (Sent: 1)]

[SPACE x1]

[WORD: "you" (Sent: 1)]

[SPACE x1]

[WORD: "to" (Sent: 1)]

[SPACE x1]

[WORD: "inser" -> "t" (Sent: 1)]

[COMMA: ',']

[SPACE x1]

[WORD: "delet" -> "e" (Sent: 1)]

[COMMA: ',']

[SPACE x1]

[WORD: "and" (Sent: 1)]

[SPACE x1]

[WORD: "repla" -> "ce" (Sent: 1)]

[SPACE x1]

[WORD: "speci" -> "fied" (Sent: 1)]

[SPACE x1]

[WORD: "words" (Sent: 1)]

[SPACE x1]

[WORD: "in" (Sent: 1)]

[SPACE x1]

[WORD: "certa" -> "in" (Sent: 1)]

[SPACE x1]

[WORD: "sente" -> "nces" (Sent: 1)]

[PUNCT: '.']

[SPACE x1]

[WORD: "yes" (Sent: 2)]

[PUNCT: '?']

----------------

Level 4:

[WORD: "Someo" -> "ne" (Sent: 3)]

[SPACE x1]

[WORD: "found" (Sent: 3)]

[SPACE x1]

[WORD: "two" (Sent: 3)]

[SPACE x1]

[WORD: "marks" (Sent: 3)]

[PUNCT: '.']

----------------

Total sentences: 4

====================

=== Insert Word Operation ===

Inserted word "EEEEEEEEE" before "e" in sentence 0

===== Document =====

Level 1:

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "The" (Sent: 0)]

[SPACE x1]

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "text" (Sent: 0)]

[SPACE x7]

[WORD: "is" (Sent: 0)]

[SPACE x1]

[WORD: "a" (Sent: 0)]

[SPACE x1]

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "seque" -> "nce" (Sent: 0)]

[COMMA: ';']

[SPACE x1]

[WORD: "of" (Sent: 0)]

[SPACE x1]

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "separ" -> "ate" (Sent: 0)]

[SPACE x1]

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "sente" -> "nces" (Sent: 0)]

[COMMA: ':']

[SPACE x1]

[WORD: "conta" -> "ining" (Sent: 0)]

[SPACE x1]

[WORD: "words" (Sent: 0)]

[SPACE x1]

[WORD: "and" (Sent: 0)]

[SPACE x1]

----------------

Level 2:

[WORD: "punct" -> "uatio" -> "n" (Sent: 0)]

[SPACE x1]

[WORD: "MARKS" (Sent: 0)]

[SPACE x1]

[WORD: "marks" (Sent: 0)]

[PUNCT: '!']

[SPACE x1]

[WORD: "Conve" -> "rt" (Sent: 1)]

[SPACE x1]

[WORD: "text" (Sent: 1)]

[SPACE x1]

[WORD: "accor" -> "ding" (Sent: 1)]

[SPACE x1]

[WORD: "to" (Sent: 1)]

[SPACE x1]

[WORD: "the" (Sent: 1)]

[SPACE x1]

[WORD: "seque" -> "nce" (Sent: 1)]

[SPACE x1]

[WORD: "of" (Sent: 1)]

[SPACE x1]

[WORD: "editi" -> "ng" (Sent: 1)]

[SPACE x1]

[WORD: "comma" -> "nds" (Sent: 1)]

[COMMA: ',']

----------------

Level 3:

[SPACE x1]

[WORD: "which" (Sent: 1)]

[SPACE x1]

[WORD: "shoul" -> "d" (Sent: 1)]

[SPACE x1]

[WORD: "allow" (Sent: 1)]

[SPACE x1]

[WORD: "you" (Sent: 1)]

[SPACE x1]

[WORD: "to" (Sent: 1)]

[SPACE x1]

[WORD: "inser" -> "t" (Sent: 1)]

[COMMA: ',']

[SPACE x1]

[WORD: "delet" -> "e" (Sent: 1)]

[COMMA: ',']

[SPACE x1]

[WORD: "and" (Sent: 1)]

[SPACE x1]

[WORD: "repla" -> "ce" (Sent: 1)]

[SPACE x1]

[WORD: "speci" -> "fied" (Sent: 1)]

[SPACE x1]

[WORD: "words" (Sent: 1)]

[SPACE x1]

[WORD: "in" (Sent: 1)]

[SPACE x1]

[WORD: "certa" -> "in" (Sent: 1)]

[SPACE x1]

[WORD: "sente" -> "nces" (Sent: 1)]

[PUNCT: '.']

[SPACE x1]

[WORD: "yes" (Sent: 2)]

[PUNCT: '?']

----------------

Level 4:

[WORD: "Someo" -> "ne" (Sent: 3)]

[SPACE x1]

[WORD: "found" (Sent: 3)]

[SPACE x1]

[WORD: "two" (Sent: 3)]

[SPACE x1]

[WORD: "marks" (Sent: 3)]

[PUNCT: '.']

----------------

Total sentences: 4

====================

=== Insert Word Operation ===

Inserted word "ALL" before "all" in sentence 3

===== Document =====

Level 1:

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "The" (Sent: 0)]

[SPACE x1]

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "text" (Sent: 0)]

[SPACE x7]

[WORD: "is" (Sent: 0)]

[SPACE x1]

[WORD: "a" (Sent: 0)]

[SPACE x1]

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "seque" -> "nce" (Sent: 0)]

[COMMA: ';']

[SPACE x1]

[WORD: "of" (Sent: 0)]

[SPACE x1]

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "separ" -> "ate" (Sent: 0)]

[SPACE x1]

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "sente" -> "nces" (Sent: 0)]

[COMMA: ':']

[SPACE x1]

[WORD: "conta" -> "ining" (Sent: 0)]

[SPACE x1]

[WORD: "words" (Sent: 0)]

[SPACE x1]

[WORD: "and" (Sent: 0)]

[SPACE x1]

----------------

Level 2:

[WORD: "punct" -> "uatio" -> "n" (Sent: 0)]

[SPACE x1]

[WORD: "MARKS" (Sent: 0)]

[SPACE x1]

[WORD: "marks" (Sent: 0)]

[PUNCT: '!']

[SPACE x1]

[WORD: "Conve" -> "rt" (Sent: 1)]

[SPACE x1]

[WORD: "text" (Sent: 1)]

[SPACE x1]

[WORD: "accor" -> "ding" (Sent: 1)]

[SPACE x1]

[WORD: "to" (Sent: 1)]

[SPACE x1]

[WORD: "the" (Sent: 1)]

[SPACE x1]

[WORD: "seque" -> "nce" (Sent: 1)]

[SPACE x1]

[WORD: "of" (Sent: 1)]

[SPACE x1]

[WORD: "editi" -> "ng" (Sent: 1)]

[SPACE x1]

[WORD: "comma" -> "nds" (Sent: 1)]

[COMMA: ',']

----------------

Level 3:

[SPACE x1]

[WORD: "which" (Sent: 1)]

[SPACE x1]

[WORD: "shoul" -> "d" (Sent: 1)]

[SPACE x1]

[WORD: "allow" (Sent: 1)]

[SPACE x1]

[WORD: "you" (Sent: 1)]

[SPACE x1]

[WORD: "to" (Sent: 1)]

[SPACE x1]

[WORD: "inser" -> "t" (Sent: 1)]

[COMMA: ',']

[SPACE x1]

[WORD: "delet" -> "e" (Sent: 1)]

[COMMA: ',']

[SPACE x1]

[WORD: "and" (Sent: 1)]

[SPACE x1]

[WORD: "repla" -> "ce" (Sent: 1)]

[SPACE x1]

[WORD: "speci" -> "fied" (Sent: 1)]

[SPACE x1]

[WORD: "words" (Sent: 1)]

[SPACE x1]

[WORD: "in" (Sent: 1)]

[SPACE x1]

[WORD: "certa" -> "in" (Sent: 1)]

[SPACE x1]

[WORD: "sente" -> "nces" (Sent: 1)]

[PUNCT: '.']

[SPACE x1]

[WORD: "yes" (Sent: 2)]

[PUNCT: '?']

----------------

Level 4:

[WORD: "ALL" (Sent: 3)]

[SPACE x1]

[WORD: "Someo" -> "ne" (Sent: 3)]

[SPACE x1]

[WORD: "ALL" (Sent: 3)]

[SPACE x1]

[WORD: "found" (Sent: 3)]

[SPACE x1]

[WORD: "ALL" (Sent: 3)]

[SPACE x1]

[WORD: "two" (Sent: 3)]

[SPACE x1]

[WORD: "ALL" (Sent: 3)]

[SPACE x1]

[WORD: "marks" (Sent: 3)]

[PUNCT: '.']

----------------

Total sentences: 4

====================

=== Remove Specific Punctuation Operation ===

Removed symbol: ,

Sentence: 3

===== Document =====

Level 1:

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "The" (Sent: 0)]

[SPACE x1]

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "text" (Sent: 0)]

[SPACE x7]

[WORD: "is" (Sent: 0)]

[SPACE x1]

[WORD: "a" (Sent: 0)]

[SPACE x1]

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "seque" -> "nce" (Sent: 0)]

[COMMA: ';']

[SPACE x1]

[WORD: "of" (Sent: 0)]

[SPACE x1]

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "separ" -> "ate" (Sent: 0)]

[SPACE x1]

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "sente" -> "nces" (Sent: 0)]

[COMMA: ':']

[SPACE x1]

[WORD: "conta" -> "ining" (Sent: 0)]

[SPACE x1]

[WORD: "words" (Sent: 0)]

[SPACE x1]

[WORD: "and" (Sent: 0)]

[SPACE x1]

----------------

Level 2:

[WORD: "punct" -> "uatio" -> "n" (Sent: 0)]

[SPACE x1]

[WORD: "MARKS" (Sent: 0)]

[SPACE x1]

[WORD: "marks" (Sent: 0)]

[PUNCT: '!']

[SPACE x1]

[WORD: "Conve" -> "rt" (Sent: 1)]

[SPACE x1]

[WORD: "text" (Sent: 1)]

[SPACE x1]

[WORD: "accor" -> "ding" (Sent: 1)]

[SPACE x1]

[WORD: "to" (Sent: 1)]

[SPACE x1]

[WORD: "the" (Sent: 1)]

[SPACE x1]

[WORD: "seque" -> "nce" (Sent: 1)]

[SPACE x1]

[WORD: "of" (Sent: 1)]

[SPACE x1]

[WORD: "editi" -> "ng" (Sent: 1)]

[SPACE x1]

[WORD: "comma" -> "nds" (Sent: 1)]

[COMMA: ',']

----------------

Level 3:

[SPACE x1]

[WORD: "which" (Sent: 1)]

[SPACE x1]

[WORD: "shoul" -> "d" (Sent: 1)]

[SPACE x1]

[WORD: "allow" (Sent: 1)]

[SPACE x1]

[WORD: "you" (Sent: 1)]

[SPACE x1]

[WORD: "to" (Sent: 1)]

[SPACE x1]

[WORD: "inser" -> "t" (Sent: 1)]

[COMMA: ',']

[SPACE x1]

[WORD: "delet" -> "e" (Sent: 1)]

[COMMA: ',']

[SPACE x1]

[WORD: "and" (Sent: 1)]

[SPACE x1]

[WORD: "repla" -> "ce" (Sent: 1)]

[SPACE x1]

[WORD: "speci" -> "fied" (Sent: 1)]

[SPACE x1]

[WORD: "words" (Sent: 1)]

[SPACE x1]

[WORD: "in" (Sent: 1)]

[SPACE x1]

[WORD: "certa" -> "in" (Sent: 1)]

[SPACE x1]

[WORD: "sente" -> "nces" (Sent: 1)]

[PUNCT: '.']

[SPACE x1]

[WORD: "yes" (Sent: 2)]

[PUNCT: '?']

----------------

Level 4:

[WORD: "ALL" (Sent: 3)]

[SPACE x1]

[WORD: "Someo" -> "ne" (Sent: 3)]

[SPACE x1]

[WORD: "ALL" (Sent: 3)]

[SPACE x1]

[WORD: "found" (Sent: 3)]

[SPACE x1]

[WORD: "ALL" (Sent: 3)]

[SPACE x1]

[WORD: "two" (Sent: 3)]

[SPACE x1]

[WORD: "ALL" (Sent: 3)]

[SPACE x1]

[WORD: "marks" (Sent: 3)]

[PUNCT: '.']

----------------

Total sentences: 4

====================

Last word searched: marks

=== Remove All Punctuation Operation ===

Sentence: 0

===== Document =====

Level 1:

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "The" (Sent: 0)]

[SPACE x1]

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "text" (Sent: 0)]

[SPACE x7]

[WORD: "is" (Sent: 0)]

[SPACE x1]

[WORD: "a" (Sent: 0)]

[SPACE x1]

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "seque" -> "nce" (Sent: 0)]

[SPACE x1]

[WORD: "of" (Sent: 0)]

[SPACE x1]

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "separ" -> "ate" (Sent: 0)]

[SPACE x1]

[WORD: "EEEEE" -> "EEEE" (Sent: 0)]

[SPACE x1]

[WORD: "sente" -> "nces" (Sent: 0)]

[SPACE x1]

[WORD: "conta" -> "ining" (Sent: 0)]

[SPACE x1]

[WORD: "words" (Sent: 0)]

[SPACE x1]

[WORD: "and" (Sent: 0)]

[SPACE x1]

----------------

Level 2:

[WORD: "punct" -> "uatio" -> "n" (Sent: 0)]

[SPACE x1]

[WORD: "MARKS" (Sent: 0)]

[SPACE x1]

[WORD: "marks" (Sent: 0)]

[SPACE x1]

[WORD: "Conve" -> "rt" (Sent: 1)]

[SPACE x1]

[WORD: "text" (Sent: 1)]

[SPACE x1]

[WORD: "accor" -> "ding" (Sent: 1)]

[SPACE x1]

[WORD: "to" (Sent: 1)]

[SPACE x1]

[WORD: "the" (Sent: 1)]

[SPACE x1]

[WORD: "seque" -> "nce" (Sent: 1)]

[SPACE x1]

[WORD: "of" (Sent: 1)]

[SPACE x1]

[WORD: "editi" -> "ng" (Sent: 1)]

[SPACE x1]

[WORD: "comma" -> "nds" (Sent: 1)]

[COMMA: ',']

----------------

Level 3:

[SPACE x1]

[WORD: "which" (Sent: 1)]

[SPACE x1]

[WORD: "shoul" -> "d" (Sent: 1)]

[SPACE x1]

[WORD: "allow" (Sent: 1)]

[SPACE x1]

[WORD: "you" (Sent: 1)]

[SPACE x1]

[WORD: "to" (Sent: 1)]

[SPACE x1]

[WORD: "inser" -> "t" (Sent: 1)]

[COMMA: ',']

[SPACE x1]

[WORD: "delet" -> "e" (Sent: 1)]

[COMMA: ',']

[SPACE x1]

[WORD: "and" (Sent: 1)]

[SPACE x1]

[WORD: "repla" -> "ce" (Sent: 1)]

[SPACE x1]

[WORD: "speci" -> "fied" (Sent: 1)]

[SPACE x1]

[WORD: "words" (Sent: 1)]

[SPACE x1]

[WORD: "in" (Sent: 1)]

[SPACE x1]

[WORD: "certa" -> "in" (Sent: 1)]

[SPACE x1]

[WORD: "sente" -> "nces" (Sent: 1)]

[PUNCT: '.']

[SPACE x1]

[WORD: "yes" (Sent: 2)]

[PUNCT: '?']

----------------

Level 4:

[WORD: "ALL" (Sent: 3)]

[SPACE x1]

[WORD: "Someo" -> "ne" (Sent: 3)]

[SPACE x1]

[WORD: "ALL" (Sent: 3)]

[SPACE x1]

[WORD: "found" (Sent: 3)]

[SPACE x1]

[WORD: "ALL" (Sent: 3)]

[SPACE x1]

[WORD: "two" (Sent: 3)]

[SPACE x1]

[WORD: "ALL" (Sent: 3)]

[SPACE x1]

[WORD: "marks" (Sent: 3)]

[PUNCT: '.']

----------------

Total sentences: 4

====================

Last word searched: marks

=== Sentences ending with word "yes" ===

Sentence 2:

[SPACE x1]

[WORD: "yes" (Sent: 2)]

[PUNCT: '?']

----------------

# Выводы

В результате выполнения курсовой работы были повторно использованы навыки по работе с иерархическим списком и хранением информации по блокам в языке C++. Была смоделирована система редактирования текста.