# **Case Study on Plagiarism Detection System Using C Programming**

## **Abstract**

In today's digital era, the issue of plagiarism has become increasingly prevalent in academia, publishing, and content creation. Detecting plagiarism ensures originality and maintains ethical standards. This case study introduces a lightweight, console-based Plagiarism Detection System developed in the C programming language. The system employs word extraction, normalization, and Jaccard similarity computation to quantify the textual similarity between two documents. Designed for simplicity and educational purposes, this tool highlights how fundamental programming concepts can address real-world problems in an efficient and understandable way.

## **Introduction**

Plagiarism, the unethical act of copying someone else's work without proper acknowledgment, is a major concern in educational institutions and industries. Advanced detection systems exist using Natural Language Processing (NLP) and Machine Learning (ML), but many scenarios require a lightweight and fast tool to detect direct textual similarities.

The presented system uses a straightforward algorithm based on set theory to detect the similarity between two text files. The goal is to calculate the proportion of shared unique words using the Jaccard similarity coefficient. This percentage helps identify whether two documents have significant content overlap, making it a useful tool for students, teachers, and content reviewers.

## **Objective**

The key objectives of the Plagiarism Detection System are:

* To develop a simple, efficient C program that reads two text files.
* To extract, normalize, and store unique words from each file.
* To compute the Jaccard similarity percentage, which quantifies the extent of similarity between the two documents.
* To provide a clear, numerical representation of potential plagiarism.

This system focuses on direct word-to-word comparison without semantic analysis, suitable for detecting literal copying rather than paraphrased or restructured plagiarism.

## **Algorithm Table for Plagiarism Detection System**

| **Step No.** | **Process Name** | **Description** | **Function Used** |
| --- | --- | --- | --- |
| 1 | Start Program | Initialize variables, open two text files for reading. | main() |
| 2 | Extract Words from Files | Read each word from the text files, normalize, and filter out duplicates. | extract\_words() |
| 3 | Normalize Words | Convert each word to lowercase and remove any non-alphabetic characters to standardize comparison. | normalize() |
| 4 | Check for Word Uniqueness | Ensure only unique words are stored to eliminate repetition in further calculation. | is\_in\_list() |
| 5 | Calculate Intersection | Compare both word sets to count the number of shared unique words (intersection). | jaccard\_similarity() |
| 6 | Calculate Union Count | Combine both word sets subtracting the intersection for accurate union count. | jaccard\_similarity() |
| 7 | Calculate Similarity Score | Apply Jaccard Similarity Formula: (Intersection / Union) \* 100 to get percentage similarity. | jaccard\_similarity() |
| 8 | Display Result | Print the final similarity percentage on the console for user evaluation. | printf() in main() |
| 9 | End Program | Close all files and exit the program. | fclose() and return |

## **Algorithm Explanation**

The core logic of the system revolves around the following steps:

### **1. Word Normalization**

* **Purpose**:  
   To clean the words by removing non-alphabetic characters and converting them to lowercase.
* **Implementation**:  
   The normalize() function loops through each character of a word, filtering out unwanted characters and standardizing the word to lowercase. This ensures uniform comparison.

void normalize(char \*word) {

int i, j = 0;

char temp[WORD\_LEN];

for (i = 0; word[i]; i++) {

if (isalpha(word[i])) {

temp[j++] = tolower(word[i]);

}

}

temp[j] = '\0';

strcpy(word, temp);

}

### **2. Unique Word Extraction**

* **Purpose**:  
   To read two files, extract words, normalize them, and store only distinct words in an array.
* **Implementation**:  
   The extract\_words() function reads the file word by word using fscanf(), applies normalization, and checks for duplication using is\_in\_list() before adding new words to the list.

int extract\_words(const char \*filename, char words[][WORD\_LEN]) {

FILE \*file = fopen(filename, "r");

if (!file) {

printf("Error opening file: %s\n", filename);

return -1;

}

char word[WORD\_LEN];

int count = 0;

while (fscanf(file, "%49s", word) != EOF) {

normalize(word);

if (strlen(word) > 0 && !is\_in\_list(words, count, word)) {

strcpy(words[count++], word);

}

}

fclose(file);

return count;

}

### **3. Similarity Calculation using Jaccard Index**

* **Purpose**:  
   To compute the percentage of similarity between the two sets of unique words.
* **Implementation**:  
   The jaccard\_similarity() function calculates the number of common words (intersection) and divides it by the total unique words (union\_count), producing a similarity percentage.

float jaccard\_similarity(char words1[][WORD\_LEN], int count1, char words2[][WORD\_LEN], int count2) {

int intersection = 0;

for (int i = 0; i < count1; i++) {

if (is\_in\_list(words2, count2, words1[i])) {

intersection++;

}

}

int union\_count = count1 + count2 - intersection;

if (union\_count == 0) return 0.0;

return ((float)intersection / union\_count) \* 100.0;

}

## **Working Flow**

1. The program reads two input text files: file1.txt and file2.txt.
2. Extracts all words and normalizes them to lowercase, removing punctuation and special characters.
3. Stores only unique words from each file.
4. Computes the Jaccard similarity, which reflects the percentage overlap.
5. Prints the plagiarism similarity percentage to the user.

## **Example Output**

Plagiarism Similarity: 68.42%

This output indicates that approximately **68% of the words** are shared between the two documents, suggesting a high likelihood of copied content.

## **Conclusion**

This C-based Plagiarism Detection System offers a clean and computationally efficient way to detect textual similarities between two files. By leveraging the Jaccard similarity formula, it provides a simple, quantitative metric to evaluate potential plagiarism.

While this system is limited to exact word matching and does not detect paraphrasing or structural rewording, it serves as an excellent foundational tool for learning about text analysis and set-based similarity metrics.

Future enhancements could include:

* Sentence-level similarity analysis.
* Synonym handling.
* Semantic analysis using external NLP libraries.