**Project Report : Email Spam Filter**

**Abstract**

The purpose of this project is to develop a simple email classification system that can categorize emails as "Spam" or "Not Spam" based on the presence of certain predefined keywords. Using basic text processing techniques, the program reads a list of emails, extracts their content, and analyzes them against a set of spam-related keywords. If the email contains a certain number of these keywords, it is classified as spam; otherwise, it is classified as not spam. This system can be used as a fundamental component in building more complex spam filters for email systems.

**Introduction**

In the modern digital world, spam emails are a significant nuisance. Spam emails can contain unsolicited advertisements, phishing attempts, or other unwanted content. Filtering such emails before they reach the inbox is crucial to maintaining a clean, safe, and efficient communication system. This project focuses on developing a basic spam classification system using keyword-based analysis.

The email classification system utilizes a set of spam-related keywords to determine if an email is likely to be spam. It reads emails from a file, processes each one by checking for the occurrence of these keywords, and classifies the email as either spam or not spam based on a threshold.

**Problem Statement**

Spam emails are an ongoing issue for email users and providers. Identifying and filtering these emails manually can be time-consuming and ineffective. Therefore, an automated spam classification system is needed to improve the efficiency of email systems. The problem tackled by this project is to design an algorithm that can classify an email as spam or not spam based on certain keywords commonly found in spam messages.

The primary goal of the project is to:

* Read email data from a file.
* Check the email content for predefined spam-related keywords.
* Classify the email as spam if it contains a certain number of these keywords.

This system will demonstrate a basic approach to spam classification, which can be further enhanced with more advanced methods like machine learning or natural language processing (NLP) in the future.

**Data Structure Used**

The following data structures are used in the program:

1. **Arrays**:
   * spam\_keywords[]: A constant array that holds the list of predefined spam keywords. It is a simple array of strings (character pointers) that are compared against the words in each email.
   * email\_copy[]: A character array used to store a copy of the email content that is transformed to lowercase for easier keyword matching.
2. **Strings**:
   * The program uses C-style strings (null-terminated arrays of characters) for email processing. Functions like strtok are used to split the email content into words, and strcmp is used to compare those words against the spam keywords.
3. **Integer Counters**:
   * spam\_count: An integer used to count the occurrences of spam keywords in the email content. If this count exceeds a predefined threshold, the email is classified as spam.
4. **File Handling**:
   * The program uses file handling (fopen, fgets, fclose) to read the email data from a text file (emails.txt). Each email is processed individually by reading its content, which is then analyzed for spam keywords.

**Approach and Methodology**

1. **File Reading**:
   * The program opens the file emails.txt using the fopen function. This file is assumed to contain email data, where each line consists of an email label (either "spam" or "not spam") followed by the email content.
2. **Text Processing**:
   * The email content is copied into the email\_copy array, and the to\_lowercase function is called to convert the entire email to lowercase. This step ensures that the keyword matching is case-insensitive.
3. **Tokenization**:
   * The strtok function is used to tokenize the email content, breaking it down into individual words. These words are then compared against the predefined list of spam keywords.
4. **Keyword Matching**:
   * Each tokenized word is compared against the spam keywords array using the strcmp function. If a match is found, the spam count is incremented.
5. **Classification**:
   * If the number of matched spam keywords (stored in spam\_count) exceeds the predefined threshold (SPAM\_THRESHOLD), the email is classified as spam. Otherwise, it is classified as not spam.
6. **Output**:
   * For each email, the program outputs the original email content, the true label from the file, and the classification result (either spam or not spam).

**Conclusion**

This spam email classification program provides a basic mechanism for filtering spam emails based on keyword matching. While the current implementation uses a simple keyword-based approach, it demonstrates the potential for creating spam detection systems. The program can be expanded by incorporating more sophisticated techniques such as machine learning models, natural language processing (NLP), or Bayesian filtering to increase accuracy and handle more complex email patterns.

By automating the classification of spam emails, the system can save time and effort for users and reduce the burden on email providers in managing spam. Future work could involve improving the threshold mechanism, introducing more dynamic keyword lists, and integrating the system with email clients for real-time spam detection.

**Potential Improvements**

1. **Dynamic Keyword List**: Instead of hardcoding the keywords, the system could dynamically update the list from external sources or user feedback.
2. **Advanced Text Processing**: More advanced techniques such as stemming, lemmatization, or machine learning could be used to improve classification accuracy.
3. **Spam Analysis**: Analyzing the context in which keywords appear (e.g., using NLP techniques) could improve classification beyond simple keyword matching.
4. **Performance Optimization**: The current implementation uses strtok and strcmp, which may not be the most efficient for large datasets. Optimizing the algorithm for speed and memory efficiency could make the system more scalable.

### UID-24BCU1005, 24BCU10003, 24BCU10031