Offline - 3: Basic NLP Operations

Full Marks: 100 **Deadline: 14 July, 2025 11:59 P.M**

Problem Overview

In this assignment, you will write a **C program** to perform basic Natural Language Processing (NLP) operations on a set of text documents. The program will process documents by normalizing case, tokenizing text, removing stop words, applying simple stemming, and compute Term Frequency (TF), Inverse Document Frequency (IDF), and TF-IDF scores. The program will run in an **interactive loop**, allowing users to input documents and execute commands to analyze the text. You will use functions and basic string operations, leveraging standard library functions from `<string.h>` and `<ctype.h>`, without using pointers.

Program Requirements

1. Global Constants and Declarations

- Define constants:

```
#define MAX_DOCS 50
#define MAX_LEN 5000
#define MAX_TOKENS 500
#define MAX_TOKEN_LEN 50
```

- Declare a 2D array to store documents:

```
char documents[MAX_DOCS][MAX_LEN];
```

- Declare a 2D array to store tokens (words) after tokenization:

```
char tokens[MAX_TOKENS][MAX_TOKEN_LEN];
```

- Initialize all documents with empty strings:

```
for (int i = 0; i < MAX_DOCS; i++) documents[i][0] = '\0';
```

- Define a fixed array of stop words:

```
#define NUM_STOP_WORDS 8
char stop_words[NUM_STOP_WORDS][MAX_TOKEN_LEN] = {"the", "is", "a", "an",
    "and", "in", "of", "to"};
```

2. Program Flow

The program starts by displaying a help message listing commands. It then enters a loop prompting for commands (e.g., `Enter command:`). Use `fgets` for string inputs to handle spaces. The first command should typically be `set` to input documents.

3. User Commands

The following table lists valid commands and their behavior:

Command	Description
`set`	Prompt for the number of documents and their text.
`preprocess`	**Normalization**: Convert all documents to lowercase and display them. **Tokenization**: Tokenize all documents into words and display the tokens. **Stop-words Removal**: Remove stop words from tokens and display the filtered tokens. **Stemming**: Apply simple stemming (remove suffixes like "ing", "ed", "s") and display results.
`tf`	Compute and display Term Frequency for a specified word across documents.
`idf`	Compute and display Inverse Document Frequency for a specified word.
`tfidf`	Compute and display TF-IDF scores for a specified word across documents.
`help`	Print the list of all available commands.
`stat`	Display TF, IDF, and TF-IDF for all tokens across all documents in a matrix format.
`exit`	Exit the program.

4. Behavior Details

A. `set`

- Prompt:

```
Enter number of documents (1-50):
```

- Prompt for each document: [**Document ID starts from one**]

```
Enter document 1:
```

- Use `fgets` and remove trailing newlines. If a document exceeds `MAX_LEN - 1`, reject it and print:

```
Document too long.
```

- Store valid documents in the `documents` array.
- If the input is invalid, print:

```
Invalid number of documents. Must be from 1 to 50.
```

- After taking input, print:

```
Documents set successfully. Please, enter 'preprocess' command now. It
    will not take other commands.
```

- Print the message above if other commands are given except `preprocess`.

B. `preprocess`

This command includes **normalization**, **tokenization**, **stop-words removal**, and **stemming** orderly.

- If no documents are set, print:

```
No documents set. Use 'set' command first.
```

Normalization:

- Write a function `void normalize_case_all()` that converts all documents to lowercase using `tolower` from `<ctype.h>`.
- Modify the `documents` array in-place.
- Display:

```
Document 1: this is the first document. it contains some simple text.
Document 2: second document here! it is slightly different in structure.
...
```

Tokenization:

- Write a function `void tokenize_all()` that splits all documents into words (tokens) using whitespace and punctuation (i.e., spaces ( ), commas (,), periods (.), colons (:), semicolons (;), question marks (?), exclamation points (!)) as delimiters.
- Clear the `tokens` array, then tokenize all documents of the `documents` array combined. Store words in the `tokens` array (up to `MAX_TOKENS`) and update `token_count` (**a global variable, indicating the number of tokens in the `tokens` array**).
- Use `isalnum` from `<ctype.h>` to identify valid token characters, treating non-alphanumeric characters as delimiters.
- Display:

```
1. this
2. is
3. the
...
```

Stop-words Removal:

- Write a function `void remove_stop_words_all()` that removes stop words from the `tokens` array preserving the order of appearances of other words.
- Compare each token against `stop_words` using `strcmp` from `<string.h>`.
- Update the `tokens` array and `token_count`.
- Display:

```
1. this
2. first
3. document
...
```

Stemming

- Write a function `void stem_all_tokens()` that removes suffixes "ing", "ed", or "s" from tokens inside the `tokens` array in-place.
- Apply stemming to all tokens in the `tokens` array. Check the last 3 or 2 characters and remove if they match with given suffixes.
- Display:

```
1. thi
2. first
3. document
...
```

C. `tf`

- Write a function `double compute_tf(char word[], int doc_id)` that computes Term Frequency (TF): (number of occurrences of `word` in document `doc_id` with stemmed words) / (total words in document `doc_id` except stop-words).
- Prompt:

```
Enter word to compute TF: simple
```

- If no documents are set, print:

```
No documents set. Use 'set' command first.
```

- Display (4 decimal places):

```
Document 1: 0.1250
Document 2: 0.0000
Document 3: 0.0000
```

D. `idf`

- Write a function `double compute_idf(char word[])` that computes IDF:

$$\text{IDF}(\texttt{word}) = \log\left(\frac{\text{MAX\_DOCS}}{1 + \text{number of documents with stemmed words containing } \texttt{word}}\right)$$

- Use `log10` from `<math.h>` for base-10 logarithm.
- Prompt for a word as in `tf`.
- If no documents are set, print:

```
No documents set. Use 'set' command first.
```

- Display (4 decimal places):

```
IDF for 'third': 1.3979
```

E. `tfidf`

- Write a function `void compute_tfidf_all(char word[])` that computes TF-IDF (TF * IDF) for each document.
- Reuse `compute_tf` and `compute_idf`.
- Prompt for a word as in `tf`.
- If no documents are set, print:

```
No documents set. Use 'set' command first.
```

- Display (4 decimal places):

```
TF-IDF for 'document':
Document 1: 0.1371
Document 2: 0.1567
Document 3: 0.1371
```

F. `help`

- Print the list of commands with descriptions in a formatted manner, aligning command names to the left:

```
set - Set documents (input or predefined)
preprocess - Apply normalization, tokenization, stop-words removal, and
    stemming orderly
tf - Compute and display Term Frequency for a specified word across
    documents.
...
```

G. `stat`

- Write a function `void display_stat()` that computes and displays TF, IDF, and TF-IDF for all unique tokens (belonging to the `tokens` array) in **alphabetically sorted order** across all documents in a matrix format.
- If no documents are set, print:

```
No documents set. Use 'set' command first.
```

- Display the tokens in **alphabetically sorted order** (4 decimal places):

```
============== TF ================
                            doc1        doc2        doc3
contain                   0.1250      0.0000      0.0000
different                 0.0000      0.1429      0.1250
...

=============== IDF ===============
                             IDF
contain                   1.3979
different                 1.2218
...

============= TF-IDF ==============
                            doc1        doc2        doc3
contain                   0.1747      0.0000      0.0000
different                 0.0000      0.1745      0.1527
...
```

H. `exit`

- Terminate the loop and exit.

5. Unknown Commands

If an unrecognized command is entered, print:

```
Unknown command. Type 'help' for list of commands.
```

6. Function Requirements

- Implement at least the following (function parameters do not have to match exactly, yours may vary depending on how you declare the variables and arrays):

```c
void normalize_case_all();
void tokenize_all();
void remove_stop_words_all();
void stem_all_tokens();
double compute_tf(char word[], int doc_id);
double compute_idf(char word[]);
void compute_tfidf_all(char word[]);
void display_stat();
```

- Use array indexing for accessing elements, not pointers. This entire assignment can be solved without pointers.
- Use `<string.h>` (any functions can be used) and `<ctype.h>` (`tolower`, `isalnum`) as needed.

Sample Input/Output

Test with inputs like:

- "This is a test document." (for `set`)
- Word "document" (for `tf`, `idf`, `tfidf`)

Refer to provided `.txt` files for sample input/output.

Mark Distribution

Component	Marks
`set` command and Normalization	10
Tokenization and Stop-words removal	20
Stemming	15
`tf`, `idf`, and `tfidf` commands	25
`stat` command with proper formatting	10
Help message and unknown command handling	10
Proper function implementation and code readability	10
Total	**100**

Submission Guidelines

Rename the .c file by your student ID `2405ABC` and upload the 2405ABC.c file to Moodle. Test thoroughly for edge cases (e.g., empty documents, no tokens, invalid inputs).