Offline - 3: Basic NLP Operations

Full Marks: 100 **Deadline: 23 June, 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 computing 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];
```

- Declare a 2D array to store tokens (words) after removing stop-words from **tokens** array:

```
char tokens_except_stop_words[MAX_TOKENS][MAX_TOKEN_LEN];
```

- Declare a 2D array to store stemmed tokens (words) after stemming the tokens of **tokens_except_stop_words** array:

```
char stemmed_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.
`normalize`	Convert all documents to lowercase and display them.
`tokenize`	Tokenize all documents into words and display the tokens.
`remove_stop`	Remove stop words from tokens and display the filtered tokens.
`stem`	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:

```
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 `documents`.

- If the input is invalid, print:

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

B. `normalize`

- Write a function `void normalize_case_all()` that converts all documents to lowercase using `tolower` from `<ctype.h>`.
- Modify `documents` directly.
- If no documents are set, print:

```
No documents set. Use ‘set’ command first.
```

- Display:

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

C. `tokenize`

- 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 `tokens` array, then tokenize all documents combined. Store words in `tokens` (up to `MAX_TOKENS`) and update `token_count`.
- Use `isalnum` from `<ctype.h>` to identify valid token characters, treating non-alphanumeric characters as delimiters.
- If no documents are set, print:

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

- Display:

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

D. `remove_stop`

- Write a function `void remove_stop_words_all()` that removes stop words from the `tokens` array and store the rest of the tokens in the `tokens_except_stop_words` array.
- Compare each token against `stop_words` using `strcmp` from `<string.h>`.
- Update `tokens` and `token_count`.

- If no tokens exist, print:

```
No tokens available. Use 'tokenize' command first.
```

- Display:

```
1. first
2. document
3. contains
...
```

E. `stem`

- Write a function `void stem_all_tokens()` that removes suffixes “ing”, “ed”, or “s” from tokens inside the `tokens_except_stop_words` array and store the stemmed tokens in the `stemmed_tokens` array.
- Apply to all tokens in `tokens`. Check the last 3 or 2 characters and remove if they match.
- If no tokens exist, print:

```
No tokens available. Use 'tokenize' command first.
```

- Display:

```
1. first
2. document
3. contain
...
```

F. `tf`

- Write a function `double compute_tf(char word[], int doc_id)` that computes Term Frequency: (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:
```

- 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
...
```

G. `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 'word': 0.4771
```

H. `tfidf`

- Write a function `void compute_tfidf_all(char word[])` that computes TF-IDF for each document: TF * IDF.
- 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 'word':
Document 1: 0.0596
Document 2: 0.0000
...
```

I. `help`

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

```
set - Set documents (input or predefined)
normalize - Convert documents to lowercase
tokenize - Split documents into tokens
...
```

J. `stat`

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

```
No tokens available. Use 'tokenize' command first.
```

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

```
============== TF ================
            doc1    doc2    doc3
contain     0.0123  0.1234  0.0012
document    0.1895  0.4449  0.1212
...

=============== IDF ===============
            IDF
contain     0.4771
document    0.3010
...

============= TF-IDF ==============
            doc1    doc2    doc3
contain     0.0059  0.0589  0.0006
document    0.0570  0.1339  0.0365
...
```

K. `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):

```
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[]);
```

```c
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>` (`strcmp`, `strlen`, `strncpy`) 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` and `normalize` commands	10
`tokenize` and `remove_stop` commands	20
`stem` command	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

Submit your `main.c` file in a folder named `2405ABC`. Zip the folder and upload to Moodle. Test thoroughly for edge cases (e.g., empty documents, no tokens, invalid inputs).

Blindly copying *from other* ***students or any other source*** *(ChatGPT, internet, etc.) will result in a* ***-100%*** *for the assignment. Discuss with classmates, but do not copy code.*