

Compiler Design Lab

Aditya Agarwal

Roll no.: 14

Reg no.:220905106

LAB1 - BASIC FILE HANDLING OPERATIONS

Q1 - To count the number of lines and characters in a file.

CODE:

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE *file;
    char filename[100];
    char c;
    int lines = 0, characters = 0;

    // Prompt user to enter the file name
    printf("Enter the filename: ");
    scanf("%s", filename);

    // Open the file in read mode
    file = fopen(filename, "r");

    // Check if the file exists
    if (file == NULL) {
        printf("Cannot open file %s \n", filename);
        exit(0);
    }

    // Read the file character by character
    while ((c = fgetc(file)) != EOF) {
        characters++; // Increment character count

        // Check if the character is a newline
        if (c == '\n') {
            lines++;
        }
    }

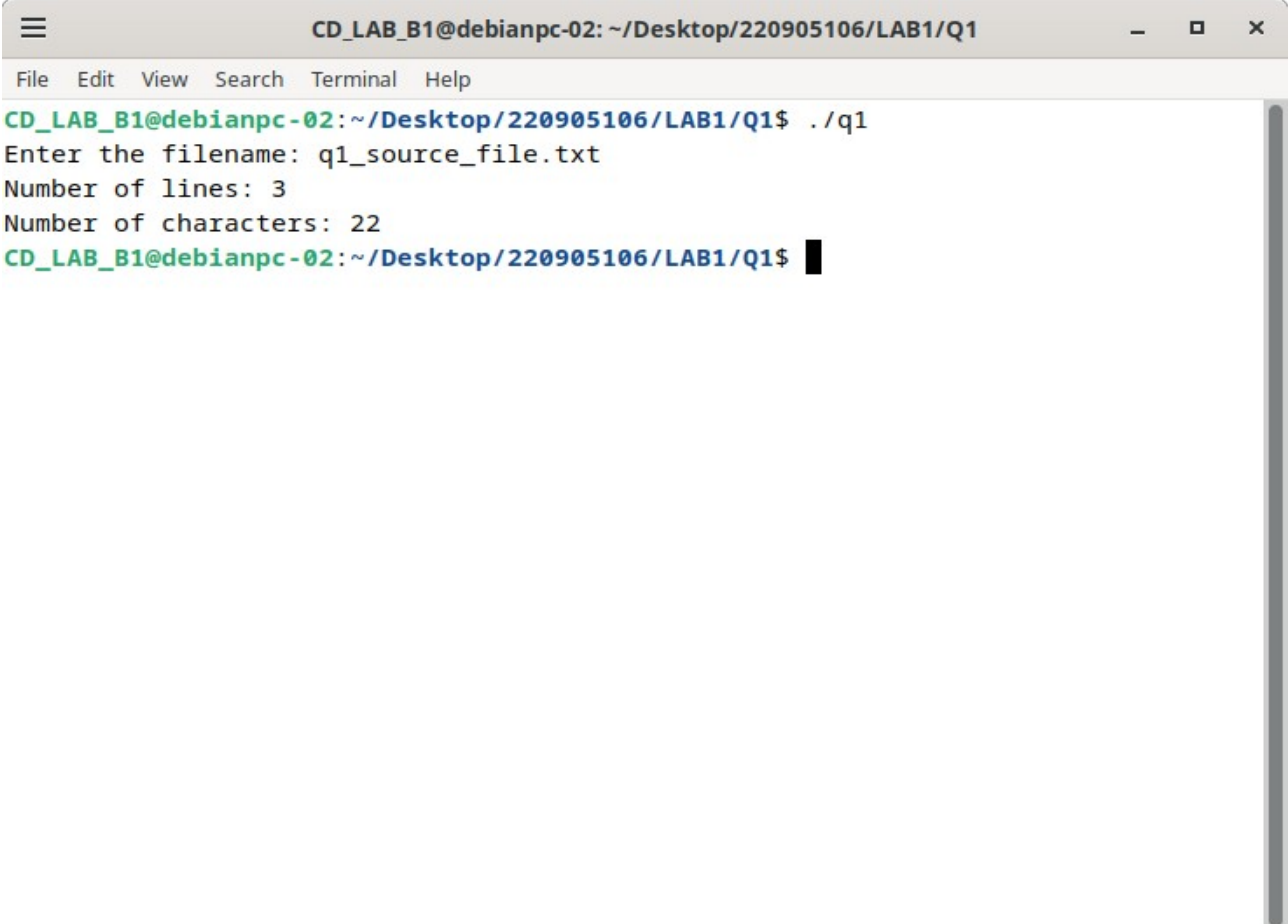
    // Close the file
    fclose(file);

    // Print the results
    printf("Number of lines: %d\n", lines);
    printf("Number of characters: %d\n", characters);
}
```

```
    return 0;  
}
```

SOURCE FILE: aditya
 agarwal
 cdlab

OUTPUT:



The screenshot shows a terminal window titled "CD_LAB_B1@debianpc-02: ~/Desktop/220905106/LAB1/Q1". The terminal has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The command prompt shows the user running `./q1`. The program prompts for a filename, and the user enters `q1_source_file.txt`. The program then displays "Number of lines: 3" and "Number of characters: 22". The prompt returns to the user.

```
CD_LAB_B1@debianpc-02: ~/Desktop/220905106/LAB1/Q1$ ./q1  
Enter the filename: q1_source_file.txt  
Number of lines: 3  
Number of characters: 22  
CD_LAB_B1@debianpc-02: ~/Desktop/220905106/LAB1/Q1$
```

Q2 - To reverse the file contents and store in another file. Also display the size of file using file handling function.

CODE:

```
#include <stdio.h>  
#include <stdlib.h>  
  
int main() {  
    FILE *sourceFile, *destinationFile;  
    char sourceFilename[100], destinationFilename[100];  
    long fileSize;
```

```

char *buffer;

// Prompt the user for the source file name
printf("Enter the source filename: ");
scanf("%s", sourceFilename);

// Open the source file in read mode
sourceFile = fopen(sourceFilename, "r");
if (sourceFile == NULL) {
    printf("Cannot open file %s\n", sourceFilename);
    exit(0);
}

// Move the file pointer to the end of the file to get the file size
fseek(sourceFile, 0, SEEK_END);
fileSize = ftell(sourceFile);
printf("Size of the file: %ld bytes\n", fileSize);

// Allocate memory to hold the file contents
buffer = (char *)malloc(fileSize * sizeof(char));
if (buffer == NULL) {
    printf("Memory allocation failed.\n");
    fclose(sourceFile);
    exit(0);
}

// Read the file content into the buffer
fseek(sourceFile, 0, SEEK_SET); // Move file pointer to the beginning
fread(buffer, sizeof(char), fileSize, sourceFile);

// Prompt the user for the destination file name
printf("Enter the destination filename: ");
scanf("%s", destinationFilename);

// Open the destination file in write mode
destinationFile = fopen(destinationFilename, "w");
if (destinationFile == NULL) {
    printf("Cannot open file %s\n", destinationFilename);
    free(buffer);
    fclose(sourceFile);
    exit(0);
}

// Write the contents in reverse order to the destination file
for (long i = fileSize - 1; i >= 0; i--) {
    fputc(buffer[i], destinationFile);
}

printf("File contents have been reversed and stored in %s\n", destinationFilename);

// Free the allocated memory and close the files
free(buffer);

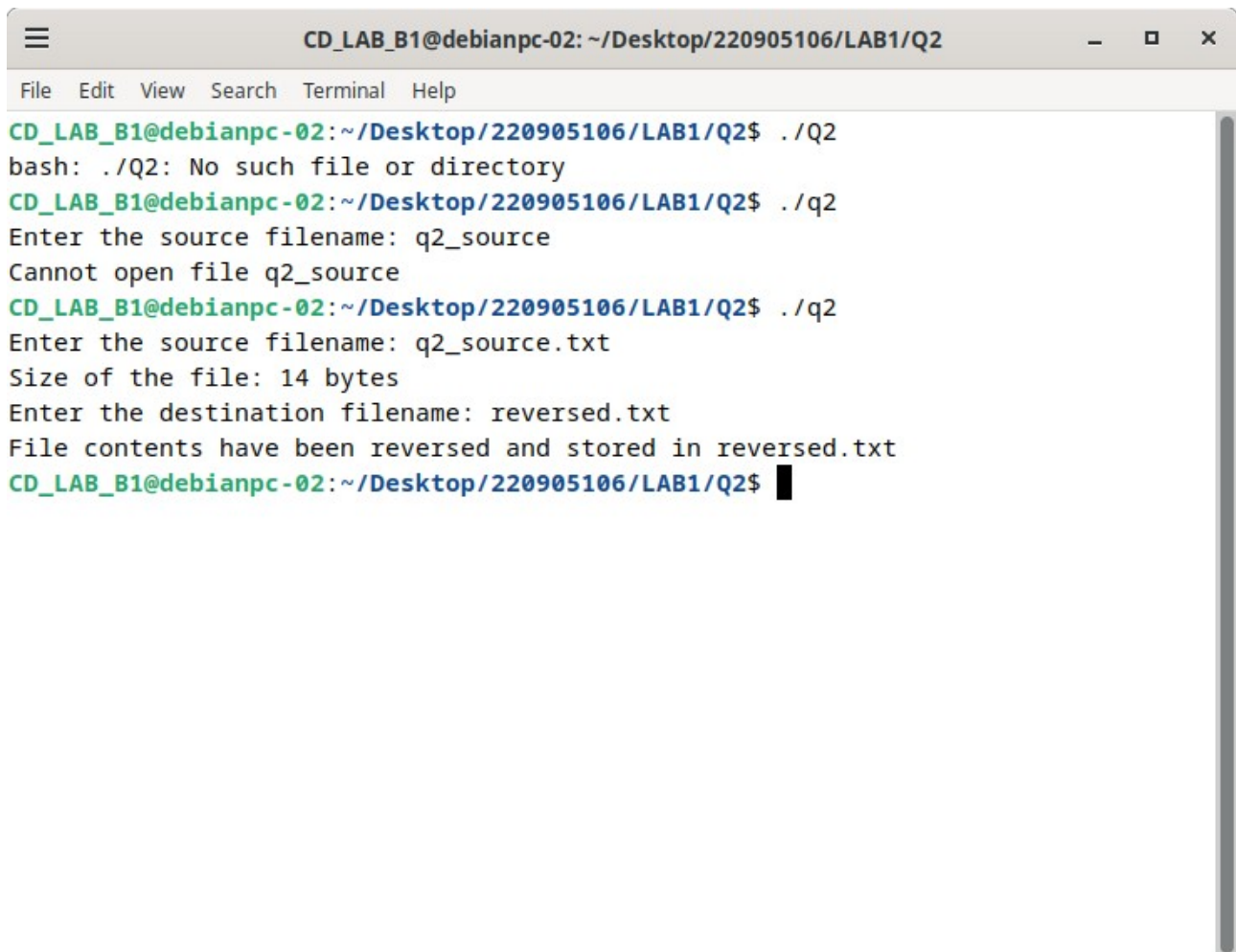
```

```
fclose(sourceFile);
fclose(destinationFile);

return 0;
}
```

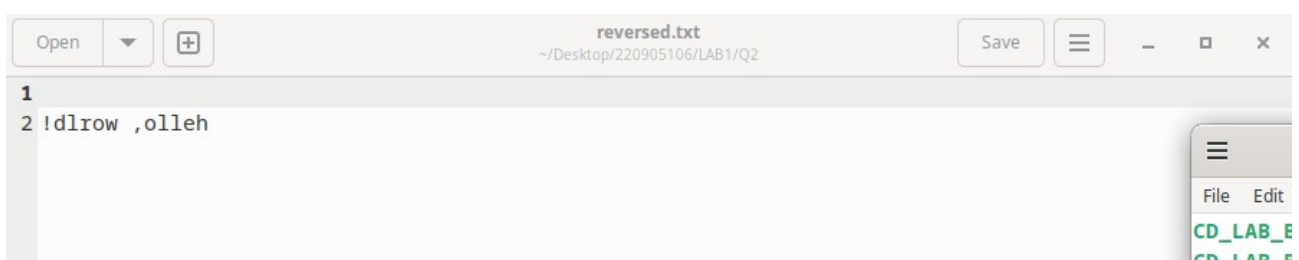
SOURCE FILE: Hello, world!

OUTPUT:



```
CD_LAB_B1@debianpc-02: ~/Desktop/220905106/LAB1/Q2
File Edit View Search Terminal Help
CD_LAB_B1@debianpc-02:~/Desktop/220905106/LAB1/Q2$ ./Q2
bash: ./Q2: No such file or directory
CD_LAB_B1@debianpc-02:~/Desktop/220905106/LAB1/Q2$ ./q2
Enter the source filename: q2_source
Cannot open file q2_source
CD_LAB_B1@debianpc-02:~/Desktop/220905106/LAB1/Q2$ ./q2
Enter the source filename: q2_source.txt
Size of the file: 14 bytes
Enter the destination filename: reversed.txt
File contents have been reversed and stored in reversed.txt
CD_LAB_B1@debianpc-02:~/Desktop/220905106/LAB1/Q2$
```

REVERSED FILE:



```
reversed.txt
~/Desktop/220905106/LAB1/Q2
Save
1
2 !dlrow ,olleh
```

Q3 - That merges lines alternatively from 2 files and stores it in a resultant file.

CODE:

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    FILE *file1, *file2, *resultFile;
    char file1Name[100], file2Name[100], resultFileName[100];
    char line[256]; // Buffer to hold a single line of text

    // Prompt the user for the filenames
    printf("Enter the name of the first file: ");
    scanf("%s", file1Name);

    printf("Enter the name of the second file: ");
    scanf("%s", file2Name);

    printf("Enter the name of the resultant file: ");
    scanf("%s", resultFileName);

    // Open the files
    file1 = fopen(file1Name, "r");
    if (file1 == NULL) {
        printf("Cannot open file %s\n", file1Name);
        exit(0);
    }

    file2 = fopen(file2Name, "r");
    if (file2 == NULL) {
        printf("Cannot open file %s\n", file2Name);
        fclose(file1);
        exit(0);
    }

    resultFile = fopen(resultFileName, "w");
    if (resultFile == NULL) {
        printf("Cannot create file %s\n", resultFileName);
        fclose(file1);
        fclose(file2);
        exit(0);
    }

    // Read and merge lines alternately
    int toggle = 0; // Used to switch between files
    while (1) {
        if (toggle == 0 && fgets(line, sizeof(line), file1)) {
            fputs(line, resultFile);
        } else if (toggle == 1 && fgets(line, sizeof(line), file2)) {
            fputs(line, resultFile);
        } else {
            break;
        }
        toggle = 1 - toggle;
    }

    fclose(file1);
    fclose(file2);
    fclose(resultFile);
}
```

```

        // If one file is exhausted, continue with the other
        if (feof(file1) && feof(file2)) break;
    }
    toggle = 1 - toggle; // Switch between file1 and file2
}

printf("Lines have been merged alternately into %s\n", resultFileName);

// Close all files
fclose(file1);
fclose(file2);
fclose(resultFile);

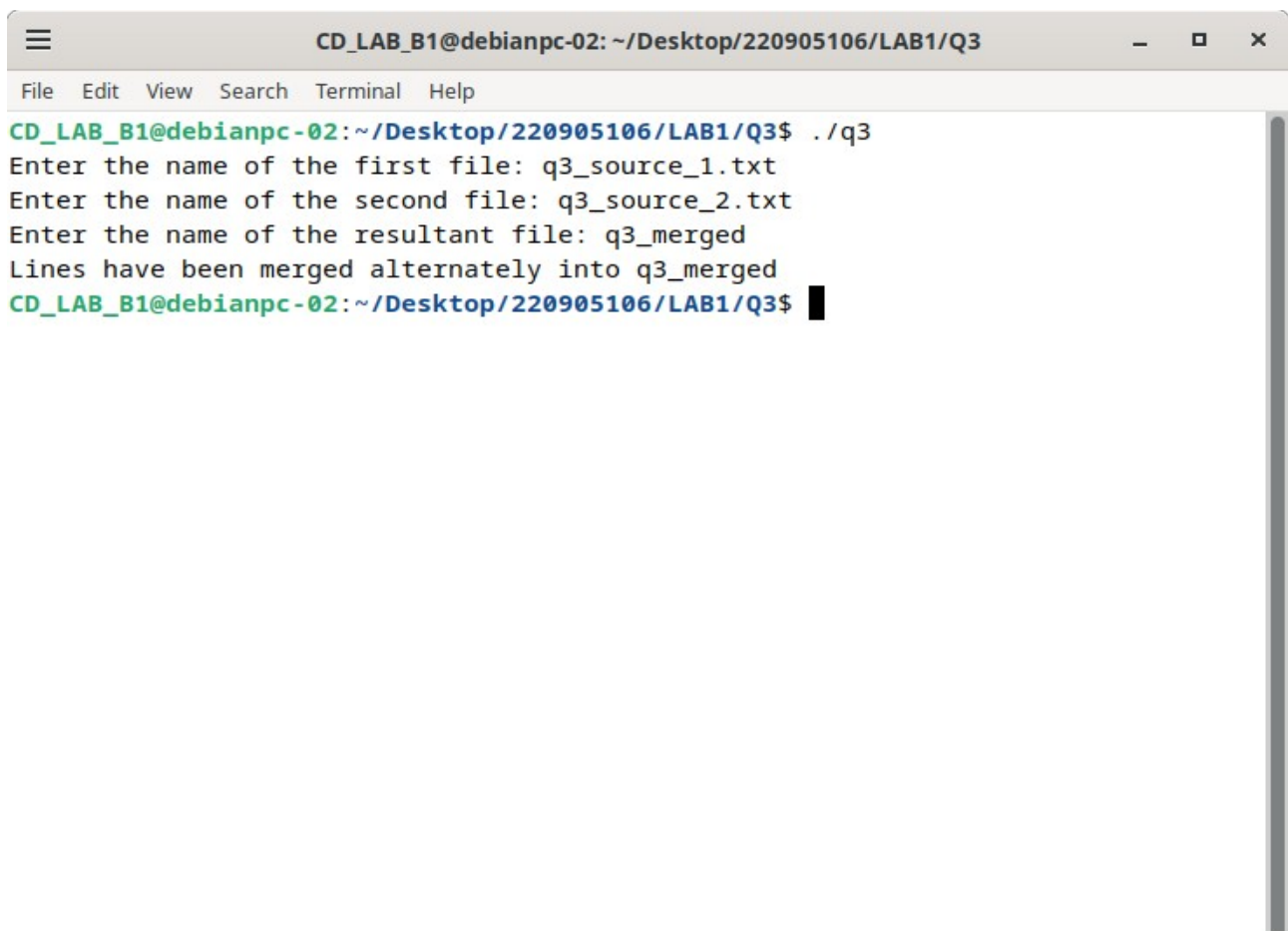
return 0;
}

```

SOURCE FILE 1: i am line 1 from file 1
 i am line 2 from file 1
 i am line 3 from file 1

SOURCE FILE 2: i am line 1 from file 2
 i am line 2 from file 2
 i am line 3 from file 2

OUTPUT:

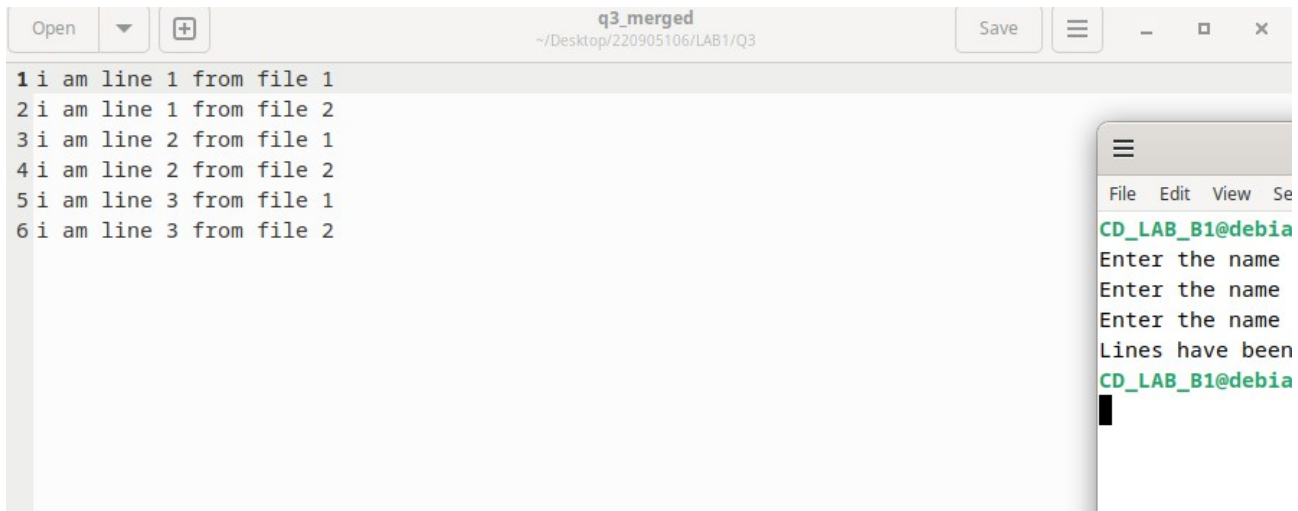


```

CD_LAB_B1@debianpc-02: ~/Desktop/220905106/LAB1/Q3
File Edit View Search Terminal Help
CD_LAB_B1@debianpc-02:~/Desktop/220905106/LAB1/Q3$ ./q3
Enter the name of the first file: q3_source_1.txt
Enter the name of the second file: q3_source_2.txt
Enter the name of the resultant file: q3_merged
Lines have been merged alternately into q3_merged
CD_LAB_B1@debianpc-02:~/Desktop/220905106/LAB1/Q3$

```

MERGED FILE:



The screenshot shows a code editor window titled 'q3_merged' with the path '~/.Desktop/220905106/LAB1/Q3'. The editor contains six lines of text: '1 i am line 1 from file 1', '2 i am line 1 from file 2', '3 i am line 2 from file 1', '4 i am line 2 from file 2', '5 i am line 3 from file 1', and '6 i am line 3 from file 2'. To the right, a terminal window is open, showing the prompt 'CD_LAB_B1@debian' and the text 'Enter the name', 'Enter the name', 'Enter the name', 'Lines have been', and 'CD_LAB_B1@debian'.

Q4 - That accepts an input statement, identifies the verbs present in them and performs the following functions

a. INSERT: Used to insert a verb into the hash table.

Syntax: insert (char *str)

b. SEARCH: Used to search for a key(verb) in the hash table. This function is called by INSERT

function. If the symbol table already contains an entry for the verb to be inserted, then it returns

the hash value of the respective verb. If a verb is not found, the function returns -1.

Syntax: int search (key)

CODE:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

// Hash Table Size
#define TABLE_SIZE 50

// Hash Table
char *hashTable[TABLE_SIZE] = {NULL};

// Auxiliary Verbs List
const char *auxiliary_verbs[] = {"is", "am", "are", "was", "were", "be", "being", "been", NULL};

// Function to Hash a String
```

```

int hash(const char *str) {
    int sum = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        sum += str[i];
    }
    return sum % TABLE_SIZE;
}

// Function to Insert a Verb into the Hash Table
void insert(char *verb) {
    int index = hash(verb);
    while (hashTable[index] != NULL) {
        index = (index + 1) % TABLE_SIZE;
    }
    hashTable[index] = strdup(verb); // Duplicate and store
    printf("Verb '%s' inserted at index %d.\n", verb, index);
}

// Function to Search for a Verb
int search(const char *verb) {
    int index = hash(verb);
    int start = index;
    while (hashTable[index] != NULL) {
        if (strcmp(hashTable[index], verb) == 0) {
            return index; // Verb found
        }
        index = (index + 1) % TABLE_SIZE;
        if (index == start) break; // Full loop
    }
    return -1; // Verb not found
}

// Function to Check if a Word is a Verb
int is_verb(const char *word) {
    // Check for auxiliary verbs
    for (int i = 0; auxiliary_verbs[i] != NULL; i++) {
        if (strcmp(word, auxiliary_verbs[i]) == 0) {
            return 1;
        }
    }
    // Check for common verb endings
    int len = strlen(word);
    if (len > 3 && strcmp(&word[len - 3], "ing") == 0) return 1; // -ing ending
    if (len > 2 && strcmp(&word[len - 2], "ed") == 0) return 1; // -ed ending
    if (len > 1 && word[len - 1] == 's') return 1; // -s ending
    return 0;
}

int main() {
    char sentence[256];
    printf("Enter a sentence: ");
    fgets(sentence, sizeof(sentence), stdin);
}

```



```

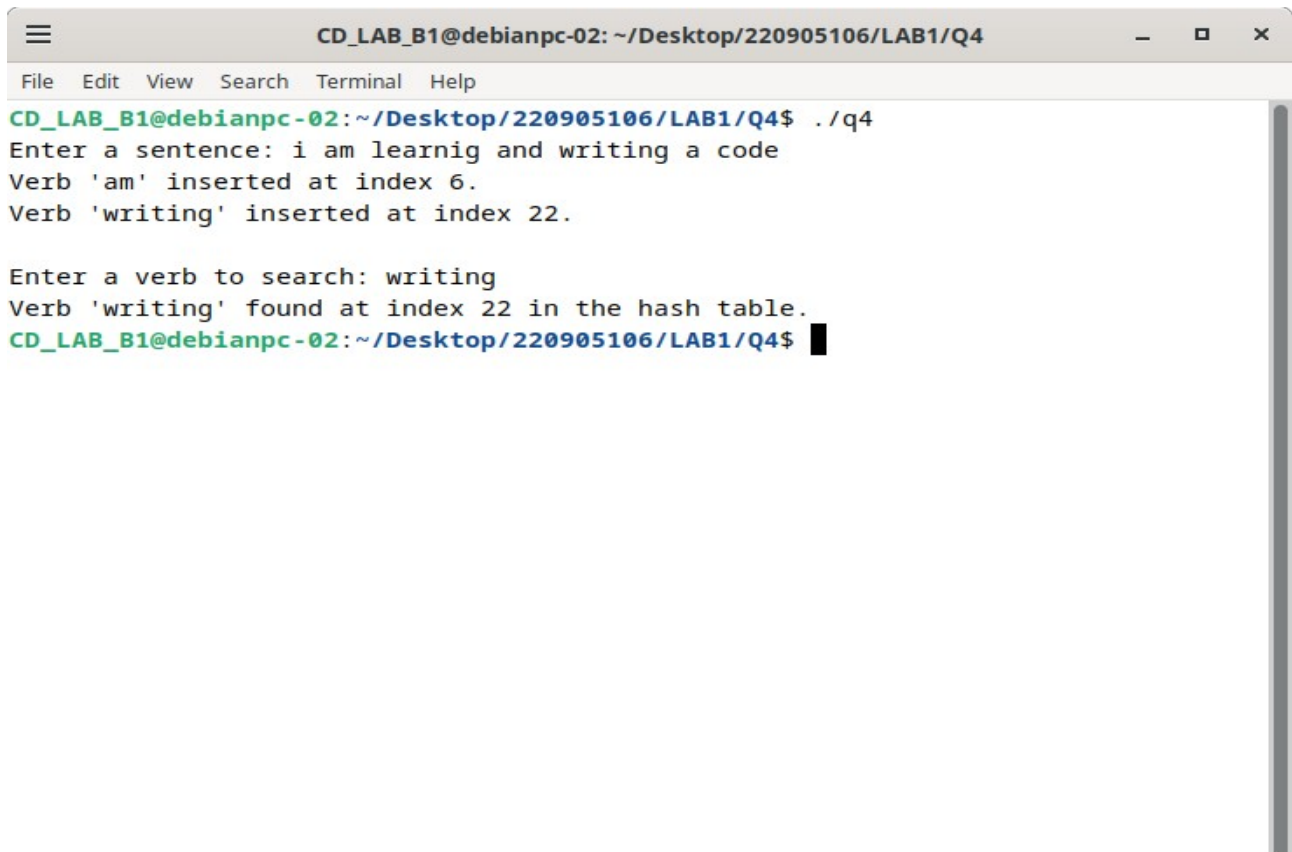
// Tokenize Sentence
char *word = strtok(sentence, ".,!?");
while (word != NULL) {
    // Check if Word is a Verb
    if (is_verb(word)) {
        if (search(word) == -1) { // Avoid duplicates
            insert(word);
        }
    }
    word = strtok(NULL, ".,!?");
}

// Search for a Verb
char verb[50];
printf("\nEnter a verb to search: ");
scanf("%s", verb);
int index = search(verb);
if (index != -1) {
    printf("Verb '%s' found at index %d in the hash table.\n", verb, index);
} else {
    printf("Verb '%s' not found in the hash table.\n", verb);
}

return 0;
}

```

OUTPUT:



```

CD_LAB_B1@debianpc-02: ~/Desktop/220905106/LAB1/Q4
File Edit View Search Terminal Help
CD_LAB_B1@debianpc-02:~/Desktop/220905106/LAB1/Q4$ ./q4
Enter a sentence: i am learnig and writing a code
Verb 'am' inserted at index 6.
Verb 'writing' inserted at index 22.

Enter a verb to search: writing
Verb 'writing' found at index 22 in the hash table.
CD_LAB_B1@debianpc-02:~/Desktop/220905106/LAB1/Q4$

```

