```
/*
Header section:
    Compilation instructions: gcc hw1.c -o hw1
    Execute instructions: ./hw1 [keywords] < [input file name]
        Note: keywords should be delimited by a space (" ").
*/


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


//Global variables
int num_keys;
struct keywords{
    char *word;
    int count;
};

/*
Method: prints each table element including keyword and final count.
Parameters: updated struct array (must call init_table and update_table first)
Return type: N/A
*/
void display_table(struct keywords arr[]){
    int i;
    for(i = 0; i < num_keys; i++){
        printf("Array[%d]:\n\tKeyword: %s\n\tCount: %d\n", i, arr[i].word, arr[i].count
);
    }
}


/*
Method: Reads & tokenizes each line, then iterates through the table to check for match
es w/ keywords.
        Updates the count variable if a match is found.
Parameters: filled struct array (must run init_table first)
return type: N/A
*/
void update_table(struct keywords arr[]){
    //variable declaration
    size_t maxlen = 0;
    ssize_t n;
    char *line = NULL;
    char *token, *prev, *temp;
    int i, k;

    while( (n = getline(&line, &maxlen, stdin) > 0)){                    //loop until th
e last line of the text
        token = strtok(line, " ");
        prev = NULL;
        while(token != NULL){                                            //loop until th
e token is null (no tokens remining in line)
        prev = token;
            for(i = 0; i < num_keys; i++){                               //check each ke
yword
                if( (strcmp(token, arr[i].word)) == 0){                  //if token == k
eyword
                    arr[i].count++;
                }
            }
```

```
            token = strtok(NULL, " ");
        }

        /*
        The first loop takes the last token in the line to be null, ending the loop and
 ignoring the final token.
        To counterract this, the previous token is stored and when the loop ends, the p
rev token is seperated
        from the newline character and then run checked for matches w/ the keywords aga
in.
        */
        temp = strtok(prev, "\n");                                          //seperate
final string from newline character '\n'
        if(temp != NULL){
            for(i = 0; i < num_keys; i++){                                  //check eac
h keyword
                if( (strcmp(temp, arr[i].word)) == 0){                      //if token
== keyword

                    arr[i].count++;
                }
            }
        }

    }
}


/*
Method: Accepts command-line input for each keyword and initializes a dynamic table(arr
ay of structs)
Parameters: empty struct array, argv (argument vector from main function)
Return type: N/A
*/
void init_table(struct keywords arr[], char** input_arr){
    //fill array w/ each keyword from command line input
    int i;
    for(i = 1; i < num_keys+1; i++){                                        //start at
1 rather than 0 to ignore the ./hw1 string

        arr[i-1].word = malloc(strlen(input_arr[i]));                       //allocate
enough space to the word section of each struct

        strcpy(arr[i-1].word, input_arr[i]);                                //copy each
 keyword from command line input to word section of each struct in the table

        arr[i-1].count = 0;                                                 //each coun
t is initialized at 0 as the input file hasn't been read yet.

    }
}

int main(int argc, char** argv){
    //initialize variables
    int i;
    num_keys = argc-1;
    struct keywords arr[num_keys];

    //call needed functions
    init_table(arr, argv);
    update_table(arr);
    display_table(arr);
```

```
    return 0;
}
```