Statistics

Introduction:

The “Statistics” is a C program, with a user menu that performs text analysis of data creates a word database and calculate statistics regarding its frequency in the given text. This program lets the users to load text from files or input text, through the keyboard handle files, analysis it by computing the statistics and saves the results in a file. It uses a linked list data structure to store words and their corresponding frequencies.

Requirements to build the program:

- Using file handling, process multiple files in one analysis, read file only once
- Program writes results into a file
- Take input from keyboard or file
- Appropriate data structure, linked list must be used
- Must have menu

Data structures:

The program uses 3 C standard libraries.(Fig.1)

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <string.h>
```

Figure 1 Used libraries

- stdio.h is used to provide input output function as well as for file handling
- stdlib.h is used to (dynamic) memory allocation
- string.h is used to work with and manipulate strings

To manage the data and represent the data in the linked list structure ‘Words’ was created .(Fig.2)

```
// Word list representing list in code
typedef struct Words {
    char *word;
    int wordcounter;
    struct Words *next;
} Words;
```

Figure 2 Structure "Words"

It has 3 members:

- word- char type variable to store the strings
- wordcounter – integer type, to store the number of word occurrence in the text.
- *next – pointer to data structure, to next node in the linked list.

Variables used in main part:

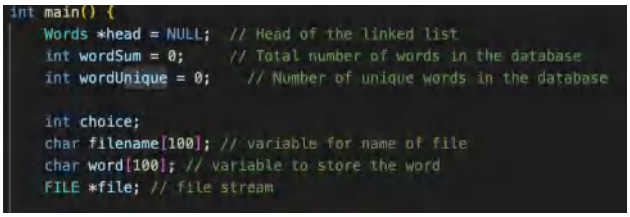


Figure 3 Variables with descriptions

To represent the menu "Switch" statement with 6 cases is used. Menu works with do-while loop and it is represented by printf and scanf functions. .(Fig.4)

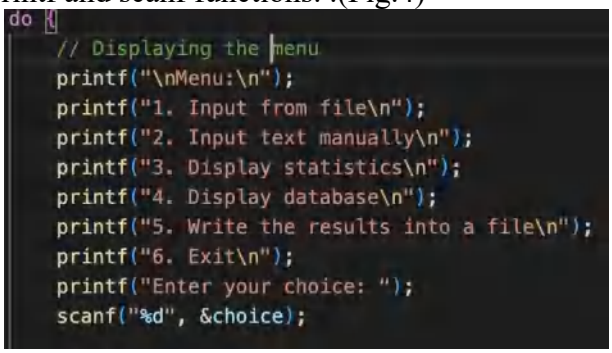


Figure 4 Code to show menu

Each case is a number – input from user given to variable "choice" and each case call for a function, according to its goal. While condition for this loop is user choice not equal to 6, that is exit option in menu. Default condition for switch is any other integer number beside in options, in this case it will display error message. (Fig.5)

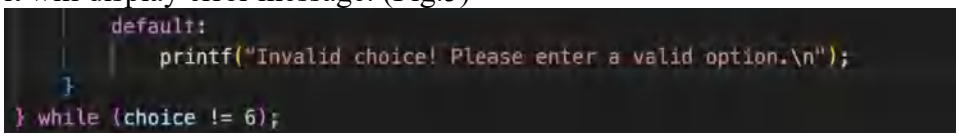


Figure 5 Part of Switch statement

Also, there is a checking the choice for integer number to avoid the infinite loop. (Fig.6) If input is not an integer, program display error message and ask for input again.

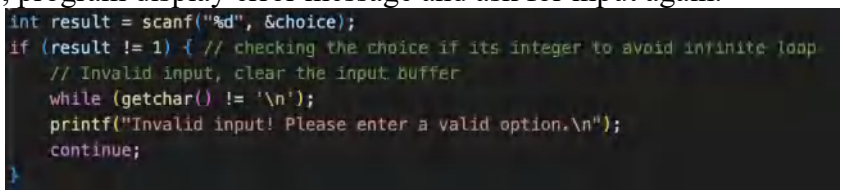


Figure 6 Choice check

Description of functions:

To create the program 5 functions in total were created.

```c
// Prototypes of fucntions before main code to call them
Words *pushword(Words *head, const char *word);
void display(Words *head);
void stats(Words *head, int *wordSum, int *wordUnique);
void FileWriter(Words *head, FILE *outputFile);
void delList(Words *head);
```

Figure 7 Prototypes of functions

*pushword

The first function is *pushword is used to insert the word to the linked list. It has 2 parameters: Words *head – pointer to the head of linked list and *word – pointer to the string, which needs to be inserted. Its return type is 'Words *' it returns a pointer to the new head of linked list. Because it updates the list and consequently changes its head, it must return address of new head.

```c
// Function to pushword a word into the linked list
Words *pushword(Words *head, const char *word) {
    Words *newNode = (Words *)malloc(sizeof(Words));
    if (newNode == NULL) {
        // Memory allocation failed
        printf("Memory allocation failed for new node\n");
        free(newNode); // free the allocated memory
        return NULL;
    }

    newNode->word = strdup(word);
    if (newNode->word == NULL) {
        // Memory allocation failed for the word
        printf("Memory allocation failed for word: %s\n", word);
        free(newNode); // free the allocated memory for word
        return NULL;
    }

    newNode->wordcounter = 1;
    newNode->next = NULL;

    // To check if the word already exists in the list
    Words *current = head;
    Words *prev = NULL;
    while (current != NULL) {
        if (strcasecmp(current->word, word) == 0) {
            // Word already exists in the linked list, so free the new node
            current->wordcounter++; // increment wordcounter
            free(newNode->word);
            free(newNode);
            return head;
        }
        prev = current;
        current = current->next;
    }

    // Insert the new node at the beginning of the list
    newNode->next = head;
    head = newNode;
    return head;
}
```

*Figure 8 *pushword function*

This function allocates memory to new node "newWord" structure using malloc and checks it for memory allocation error to avoid memory leak. Then copies the word to this new nodes data. After checking this word for presence in the list by comparing strings in using 'strcasecmp' library function, so program will not be case sensitive. If, it exists it increments its counter, frees

created node and returns head. If word in not in the list, it inserts the word at the beginning of list and updates the head of list. This function is used in both inserting text though keyboard and reading from file cases.

display

This function has one parameter, Words *head – a pointer to head of linked list. It is void type, so it does not return anything, but performs operation.

```
// Function to display the linked list
void display(Words *head) {
    Words *current = head;
    while (current != NULL) {
        printf("%s: %d\n", current->word, current->wordcounter);
        current = current->next;
    }
}
```

Figure 9 display function

It initializer new pointer “current” equal to pointer to head and traverses through linked list, while printing the data of each node in list: the word and number of its occurrence. This function is used to show the statistics of the text.

stats

This function is used to calculate the statistics of words in the inputted text. It has 3 parameters: pointer to head of the linked list, integer variable’s pointer that is used to store total number of words and pointer to other integer variable to store number of unique words. It is void type, so it does not return anything, but performs operation.

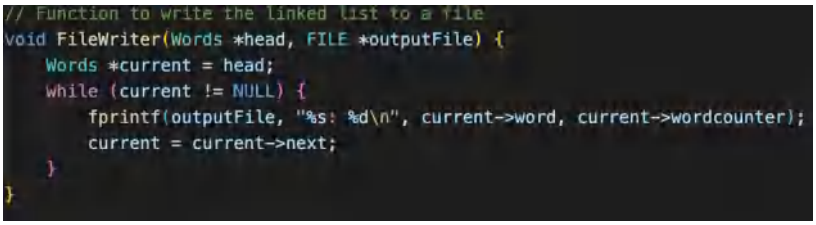
```
// Function to calculate statistics of the words in the linked list
void stats(Words *head, int *wordSum, int *wordUnique) {
    *wordSum = 0;
    *wordUnique = 0;

    Words *current = head;
    while (current != NULL) {
        (*wordUnique)++;
        *wordSum += current->wordcounter;
        current = current->next;
    }
}
```

Figure 10 stats function

It initializes new pointer “current” equal to pointer to head and traverses through linked list, while incrementing the number of unique words and calculating the sum of words, total number of words is sum of each word node’s wordcounter. Number of unique words is only incremented, because linked list contain only unique words.

FileWriter

This function is used to handle files and insert the data of linked list into file. It has 2 parameters, pointer to head of linked list and pointer to FILE structure that shows output file, where it writes the results. It is also void type, so it does not return anything, but performs operation.



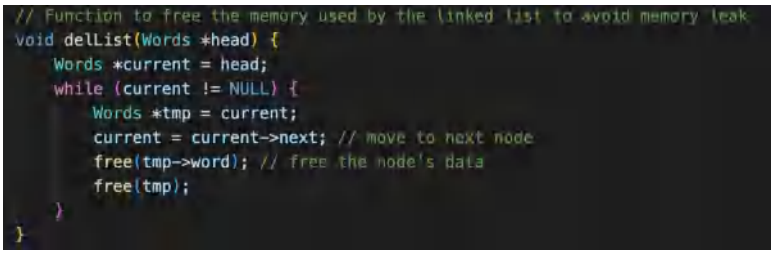
```c
// Function to write the linked list to a file
void FileWriter(Words *head, FILE *outputFile) {
    Words *current = head;
    while (current != NULL) {
        fprintf(outputFile, "%s: %d\n", current->word, current->wordcounter);
        current = current->next;
    }
}
```

Figure 11 FileWriter function

It also traverses through linked list, structure is very similar to display function, but instead of printing the data of linked list out, it writes it into a given file.

delList

This function is used to free the memory allocated to nodes of linked list to avoid a memory leak. It has only one parameter – pointer to head of linked list. It initializes two pointers: current and tmp, both point to head of linked list, they are used to move through list and free the data of each node.



```c
// Function to free the memory used by the linked list to avoid memory leak
void delList(Words *head) {
    Words *current = head;
    while (current != NULL) {
        Words *tmp = current;
        current = current->next; // move to next node
        free(tmp->word); // free the node's data
        free(tmp);
    }
}
```

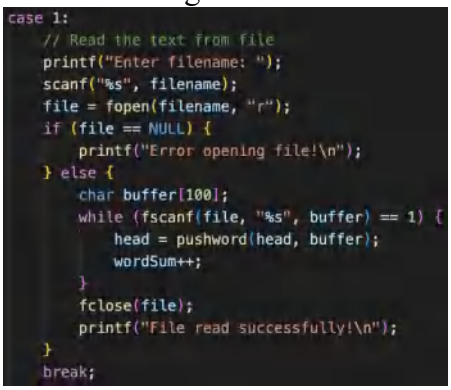
Figure 12 delList function

Implementation of functions in main part

Each swith case in main part will call for proper functions.

Case 1 – Input from file

If user choses 1 option in menu, the program will ask for a name of file and and then by function from standard library it will open file in read mode “r”. It initializes variable string “buffer” and allocates 100 char size to it. If the name is correct, program scans text in file and read word one by one and copies then into “buffer” while calling the *pushword function to insert the previous “buffer’s data into linked list and incrementing the total sum of words.



```c
case 1:
    // Read the text from file
    printf("Enter filename: ");
    scanf("%s", filename);
    file = fopen(filename, "r");
    if (file == NULL) {
        printf("Error opening file!\n");
    } else {
        char buffer[100];
        while (fscanf(file, "%s", buffer) == 1) {
            head = pushword(head, buffer);
            wordSum++;
        }
        fclose(file);
        printf("File read successfully!\n");
    }
    break;
```

Figure 13 Case 1

If it could not open the file error will appear. If everything is successful, success message will appear, and programs returns to main page.

Case 2 – Input text manually
If case 2 is chosen, program asks for a text and will read it with while loop until word “exit7” is inputted by user.

```
case 2:
    // Enter text manually
    printf("Enter text (type 'exit7' to stop):\n");
    while (1) {
        scanf("%s", word);
        if (strcmp(word, "exit7") == 0) {
            break;
        }
        head = pushword(head, word);
        wordSum++;
    }
    break;
```

Figure 14 Case 2

Case 3 – Display statistics
This case call function stats by giving parameters: pointer to head of linked list, pointers to variables: “wordSum” and “wordUnique”. After functions is called and the variables are calculated/updated. The data of variables are printed out.

```
case 3:
    // Display statstistics
    stats(head, &wordSum, &wordUnique);
    printf("Total words: %d\n", wordSum);
    printf("Unique words: %d\n", wordUnique);
    break;
```

Figure 15 Case 3

Case 4 – Display database
This case only calls for a function display giving the parameter – pointer to head of linked list.

```
case 4:
    // Display database
    display(head);
    break;
```

Figure 16 Case 4

Case 5 – Write the results into a file
In this case, program asks user for a name of file to write the data of linked list and opens it in “w” write mode. If there was error, error message will occur. Other way, it calls for a function FileWriter, by fclose closes file and shows success message.

```c
case 5:
    // Write the results into a file
    printf("Enter output filename: ");
    scanf("%s", filename);
    file = fopen(filename, "w");
    if (file == NULL) {
        printf("Error opening file for writing!\n");
    } else {
        FileWriter(head, file);
        fclose(file);
        printf("Data written to file successfully!\n");
    }
    break;
```

Figure 17 Case 5

Default case

Default case is any input which is not in the choice menu, in this case program shows message asking right value again.

```c
default:
    printf("Invalid choice! Please enter a valid option.\n");
```

Figure 18 Default case

Case 6 – Exit

This case finishes the do-while loop and and after the delist function is called to free the linked list to avoid memory leak and program stops.

```c
} while (choice != 6);

// Free the memory used by the linked list
delList(head);

return 0;
```

Figure 19 Exit and freeing memory

Testing of work

Inputting the text by file.

```
Menu:
1. Input from file
2. Input text manually
3. Display statistics
4. Display database
5. Write the results into a file
6. Exit
Enter your choice: 1
Enter filename: text.txt
File read successfully!
```

Figure 20 Menu

File text.txt was inputted and 3 option was chosen to display statistics of text. Another file can be inputted by doing the option 1 again.

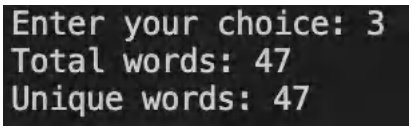


Figure 21 Test result

After 4th option was chosen to show the database

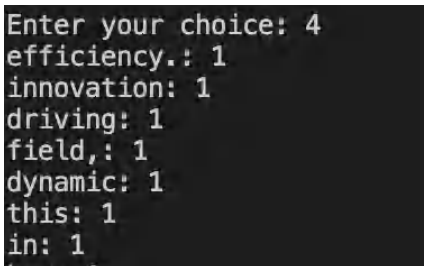


Figure 22 Example results

After 5th option was chosen and output.txt was entered as a name of output file

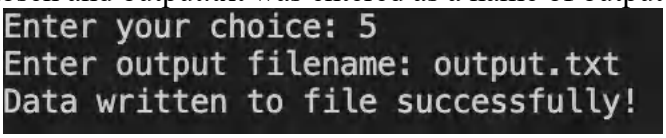


Figure 23 Writing results into a file

And new file is created in the root file of program

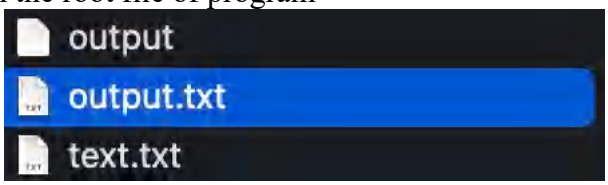


Figure 24 Results

Inputting the text manually is analogical.

Drawback of program and ways to improve

The program is not perfect, and it has limits. The problem is the calculations are correctly done with naked texts only, program is sensitive for punctuation marks, two same words with and without punctuation marks are counted differently. In future, the function to separate words from the can be added.

Summary:

In the end, this is menu driven C program, which analyses text either from text or keyboard input and calculates the total number of words, number of unique words and occurrence of each word in text by using a linked list. It also, uses file handling, can handle multiple files at once. Program is capable of allocating necessary memory and freeing them to avoid the memory leak while its work.