CE243-5-AU
Assignment1 Report

Word count (excluding references, code blocks etc.): 1137

Word count (Total): 1420

Introduction

The assignment given introduces a real-life problem. The goal is to format the text file in the such way that:

a. Only one sentence should occupy one line with one of a specified punctuation mark.
b. If a sentence contains one of C keywords, it should occupy one line by itself.
c. If a keyword has punctuation mark following it, the keyword should occupy one line itself as well.
d. Finally, the output should be printed in the console and written to file.

Header file description

I started off from Header file (.h) where I defined the integer array to store punctuation marks and array of pointers to store keywords.

```
int punctArr[4] = {
	63,
	...
	46
};

char *keyArr[32] = {
	"auto",
	...
	"while"
```

```
};
```

Also, I defined the length of both arrays as variables, to access later in the main program.

```
int punctArrLen = sizeof(punctArr)/sizeof(punctArr[0]);
```

```
int keyArrLen = sizeof(keyArr)/sizeof(keyArr[0]);
```

Then it was included into main.c by `#include "stuffDef.h"`

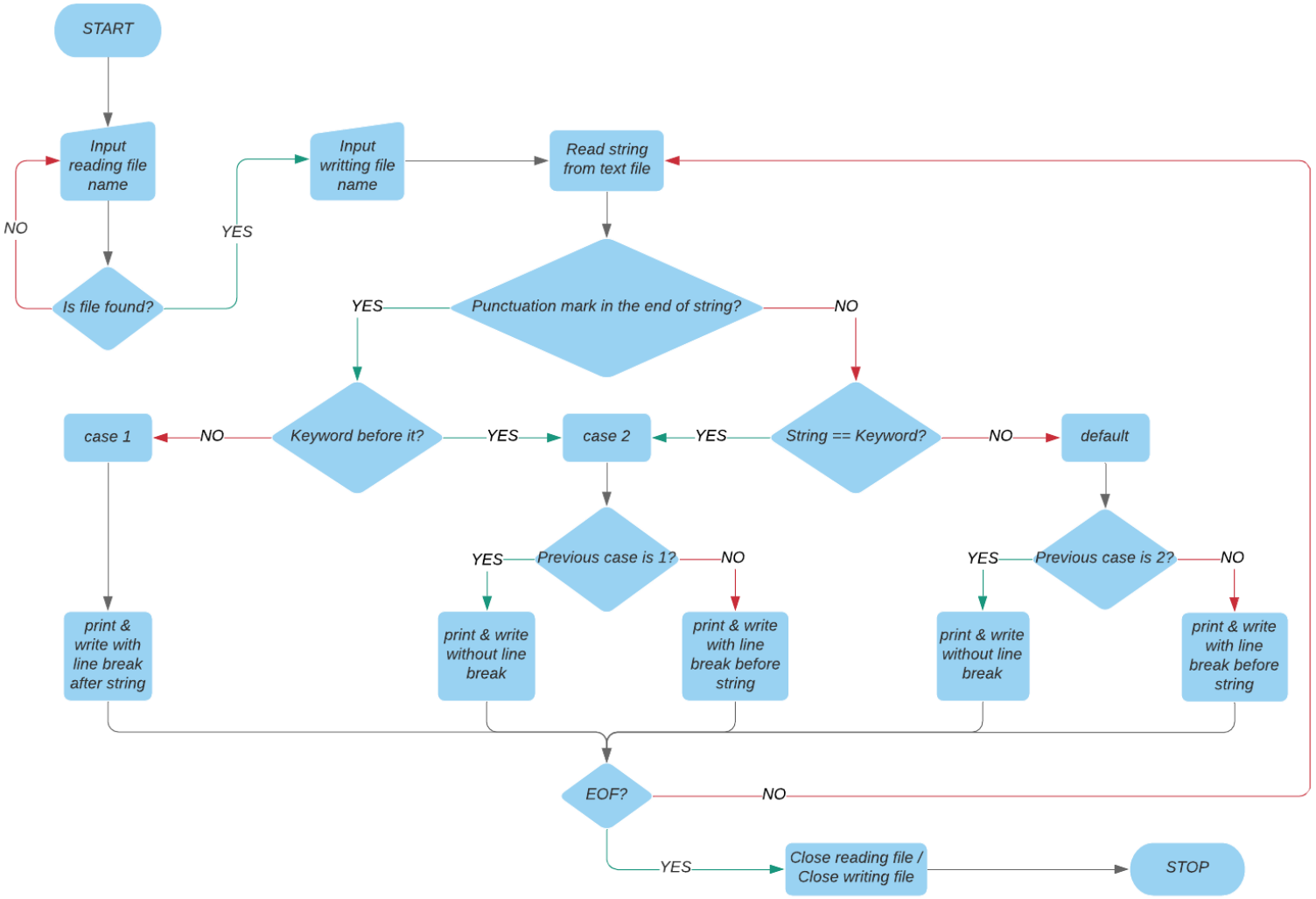
Main function

Structure

The main function can be broken down in few blocks and sub blocks:

- Initialization of variables
- Reading / Writing file opening
- Printing / Writing while loop
  - Local variables initialization
  - Determine the case to which string applies (1, 2 or default (0))
  - Switch statement for actual printing of a single string
- Printing number of “Total number of lines” line
- Closing files

For a better visual representation, I’ve got a flowchart.



In sake of simplicity some of the non-main steps like, counting the number of lines or tracking the variables were not included in the flowchart.

Beginning, file opening

It starts from initialization of global variables that I'll use throughout the program:

```
int
    was,
    count = 0;
```

`was` for remembering state of previous string

`count` for counting the number of lines

```
char
        string[25],
        fName_read[40],
        fName_write[40];
FILE *fpr = NULL;
FILE *fpw = NULL;
```

Here pointers are used to store file memory location. [1]

While loop is used to get the correct name of a file. It will print

```
"No such file exists in project directory\n"
```

if file wasn't found. No while loop for writing file. It will create a new one if name didn't match any file name from project directory. [2]

Printing/ writing while loop and logic

Almost everything else is fitted inside a while loop because the rest is meant to define what case the string applies to and print according to rules stated in the assignment instructions. Inside a loop, first thing to do is to define local variables:

```
char
        newStr[25] = {0},
        localStr[25];
int
        state = 0,
        len = strlen(string),
        newStrLen = strlen(newStr);
```

Char arrays are used to copy string for further modification and analyzation.

Integer `state` is introduced, it will store the result of analyzation of the string.

`state = 1;` means that string has a punctuation mark at the end

`state = 2;` means that string has a punctuation mark at the end and is a keyword or is a keyword on its own

*Improvement was made because I realize that the case when there is a keyword and case when there is a keyword with a mark, lead to the same solution and same printing statements. Therefore, I decided to group them up into one case.

Any other values of `state` will mean that it is `default` case in the switch statement.

The main logic of a whole program is in this part. Here it decides which state to use. First, it checks if there is a punctuation mark at the end. If there is no, program checks if it is a keyword by iterating through a keyword array in a for loop and comparing the string to each element in an array using `strcmp(string1, string2)`[3]. It is the same method that was used in checking for a punctuation mark. So, if there was a punctuation mark then `state = 1` and the next step would be to check how many marks there are, is it a keyword or not, and give it a state. For this there is a while loop with argument 1, so it won't stop unless you break from it. This loop is used to trim the string. To compare the string with keyword array, it is required to erase all punctuation marks from the string. To do this it will delete last character and check whether there is a need to erase next one. If there is no need `break;` is used to escape while loop and go inside a for loop described above to check for a keyword, in case of success; `state = 2`. If string didn't match any of "requirements" then it is a usual word and will have: `state = 0`.

Switch statement, decision making

After state was determined, switch statement [4] is used to manage what to print. Depending on the case it will print a string with or without a line carriage (`"\n"`).

```
switch(state) {
	case 1 :
		...
		was = 1;
		break;
	case 2 :
		if(was == 1) { ... } else { ...}
		was = 2;
		break;
	default :
		if(was == 2){ ...} else { .. }
		was = 3;
	}
```

For an easier understanding it is better to break the code into “case” pieces.

Case 1:

Punctuation mark(s) after a word (not a keyword)

`count++;`

In this case, a string is always printed with linefeed in the end. So, it looks like a whole sentence is taking a whole line. The `printf(...)` is followed by `fprintf(fpw, ...)` [5] to write in both console and file. Need to use pointer `fpw` to indicate to which file it supposed to be written to.

`was = 1;` this will indicate the most recent state

`break;` necessary to exit switch

Case 2:

Keyword and punctuation mark(s) or keyword on its own

To escape the problem where extra lines were printed and empty lines occurred. The if – else statement was implemented. So, if the `"\n"` was printed in case 1 and `was = 1`, then no need to print it again. Otherwise, print with linefeed before string. Same as in case 1, `printf` and `fprintf` are both used.

```
was = 2;
break;
```

Default:

Simply, just a word

Again, so no duplicate lines are produced, the if – else statement implemented. If there was no linefeed printed before (e.g. in previous `printf(...)`, `"\n"` went before string), then print line carriage before string `printf(" \n%s ", string);`, otherwise print without `"\n"`.

```
was = 3;
```

No need to use break, it’s last statement in the sequence.

In every case, whenever linefeed was used, `count++;` was used too, to count number of lines. `Count` has a global scope to be used outside of the while loop.

After major while loop reached EOF(end of file) the loop ends.

Final bits

Now print `printf("\nTotal number of lines is: %d\n", count);` number of lines printed. And close the files:

```
fclose(fpr);
```

```
fclose(fpw);
```

`return 0;` Returns 0 if program ran successfully.

Conclusion

To conclude I think, that the design created, fits the problem and provides a reliable result, because it meets all requirements and provides correct output. The code was debugged, improved and modified throughout. It is made in a simple manner and may be explained easily.

The exercise gave me an opportunity to practice my skills gained in labs and to learn from different sources about the language and its structure. I understand the idea of pointers and array, how to open a file and write into it, creating new header files and how to use most of the keywords. I've met C language for the first time this year, and I learned about structure "grammar" of a language, basic concept and logic.

If I would ever to make this again, I would be far from what I have now as I think, that my program is close to be optimal, even though it's hard to judge it in so small programs.

References

[1]: Pointers, Lecture 2: https://moodle.essex.ac.uk/pluginfile.php/796586/mod_resource/content/3/CE243-T2-Expressions-in-C.pdf and https://overiq.com/c-programming-101/array-of-pointers-to-strings-in-c/ , also https://www.cs.bu.edu/teaching/cpp/string/array-vs-ptr/

[2]: fopen(), Learned from here: https://www.tutorialspoint.com/cprogramming/c_file_io.htm and Lab 2: https://moodle.essex.ac.uk/pluginfile.php/681379/mod_resource/content/3/CE243-Lab2.pdf

[3]: strcmp(), Learned from here: https://www.tutorialspoint.com/c_standard_library/c_function_strcmp.htm and Lab 2: https://moodle.essex.ac.uk/pluginfile.php/681379/mod_resource/content/3/CE243-Lab2.pdf

[4]: Switch Lecture Unit 3: https://moodle.essex.ac.uk/pluginfile.php/799646/mod_resource/content/2/CE243-T3-Program-Control%20Statements-in-C.pdf and https://www.tutorialspoint.com/cprogramming/switch_statement_in_c.htm

[5]: fprintf() Lecture Unit 4: https://moodle.essex.ac.uk/pluginfile.php/801698/mod_resource/content/3/CE243-T4-Input-Output-in-C.pdf and Lab 4: https://moodle.essex.ac.uk/pluginfile.php/686077/mod_resource/content/3/CE243-Lab4.pdf

Code

```
#include <stdio.h>
#include <string.h>
#include "stuffDef.h"

//created by Dr. Heinz Doofenshmirtz id: 1804336
//on 04-11-19
int main(void) {
    //Declare the variables
    int
        was,
        count = 0; ///was defines the previous state, count counts the lines
    char
        string[25],
        fName_read[40],
        fName_write[40];

    FILE *fpr = NULL; //file pointer for reading
    FILE *fpw = NULL; //file pointer for writing

    //open file         most of this part is taken from Lab 2, completed by me
```

```
    while(fpr == NULL) {
        //asking for input
        printf("Enter the name of the file to read: ");
        scanf("%s", fName_read);
        fpr = fopen(fName_read, "r");

        if(fpr != NULL) {
            //break if file found
            break;
        }
        printf("No such file exists in project directory\n");
    }
    printf("\n");

    //open/create a file to write
    printf("Enter the name of the file to write: ");
    scanf("%s", fName_write);
    fpw = fopen(fName_write, "w");
    printf("\n");

    //printing from file in while loop
    //writing to file
    while (fscanf(fpr, "%s", string) != EOF) {

        //Local variables declaration
        char
            newStr[25] = {0},
            localStr[25];

        int
            state = 0,
            len = strlen(string),
```

```
    newStrLen = strlen(newStr);

///Determine the state

//Check for punctuation marks
for (int x = 0; x < punctArrLen; x++) {
    if(string[len - 1] == punctArr[x]) {
        state = 1;
    }
}

//Check if there is a punct. mark after a keyword
if(state == 1) {

    strcpy(localStr, string);

    /*
    quitting the loop by break keyword
    while loop needed for the case when
    there are few consequent punctuation marks
    after a keyword
    */
    while(1) {
        //local variables
        int
            thereIs = 0,
            localLen = strlen(localStr);

        //Erase last character (punctuation mark)
        for(int x = 0; x < localLen - 1; x++) {
            newStr[x] = localStr[x];
        }
```

```
            newStrLen = strlen(newStr);

            //compares last character to an array of punct. marks
            for (int x = 0; x < punctArrLen; x++) {
                if(newStr[newStrLen - 1] == punctArr[x]) {
                    thereIs = 1;
                    break; //break because only one match possible
                }
            }

            //break if there is only one punct. mark
            if(thereIs == 0){
                break;

            //if there is punct. mark left in a string, updates changes
            //and restores newStr
            } else if(thereIs == 1) {
                strcpy(localStr, newStr);
                memset(newStr,0,sizeof(newStr));
            }
        }

        //compares string to a keyword array
        for(int x = 0; x < keyArrLen; x++) {
            if(strcmp( newStr, keyArr[x]) == 0) {
                state = 2;
                break; //break because only one match possible
            }
        }

    } else {
```

```
        //Check if there is just a keyword
            for(int x = 0; x < keyArrLen; x++) {
                if(strcmp( string, keyArr[x]) == 0) {
                    state = 2;
                }
            }
        }

        switch(state) {
            case 1 :       ///state(case) 1: punct. mark
                count++;
                printf("%s \n", string);
                fprintf(fpw, "%s \n", string);
                was = 1;
                break;
            case 2 :      ///state(case) 2: keyword and punct. mark or keyword on its own
                if(was == 1) {
                    printf("%s ", string);
                    fprintf(fpw, "%s ", string);
                } else {
                    count++;
                    printf(" \n%s ", string);
                    fprintf(fpw, " \n%s ", string);
                }
                was = 2;
                break;
            default :
                if(was == 2){
                    count++;
                    printf(" \n%s ", string);
                    fprintf(fpw, " \n%s ", string);
                } else {
```

```
                    printf("%s ", string);
                    fprintf(fpw, "%s ", string);
                }
                was = 3;
            }
        }
        printf("\nTotal number of lines is: %d\n", count);
        fprintf(fpw, "\nTotal number of lines is: %d\n", count);

        //close files
        fclose(fpr);
        fclose(fpw);

        return 0;
}
```

Output(from file)

C allows the programmer to write directly to memory unlike most of other programming languages.

Key constructs in C such as pointers and arrays are designed to structure and manipulate memory in a machine-independent fashion.

Thirty two Keywords in C are also called as reserved words such as

else,

for,

volatile,

char,

int,

float,

goto,

auto,

break

and

```
void.

C gives control over the memory layout of data structures!

Dynamic memory allocation in C is under the control of the programmer unlike languages
like Java and Perl that shield the programmer from memory allocation and pointers.

This can be useful since dealing with memory allocation when building a high-level
program is a highly error-prone process.

C provides a uniform interface when dealing with low-level code such as the part of the
OS that controls a device.

Do these capabilities exist in most other languages?

Answer is no!!!


Total number of lines is: 20
```

Output(from console)

```
Enter the name of the file to read: assignment1.txt


Enter the name of the file to write: a


C allows the programmer to write directly to memory unlike most of other programming
languages.

Key constructs in C such as pointers and arrays are designed to structure and manipulate
memory in a machine-independent fashion.

Thirty two Keywords in C are also called as reserved words such as

else,

for,

volatile,

char,

int,

float,

goto,

auto,

break

and

void.

C gives control over the memory layout of data structures!
```

Dynamic memory allocation in C is under the control of the programmer unlike languages like Java and Perl that shield the programmer from memory allocation and pointers.

This can be useful since dealing with memory allocation when building a high-level program is a highly error-prone process.

C provides a uniform interface when dealing with low-level code such as the part of the OS that controls a device.

Do these capabilities exist in most other languages?

Answer is no!!!

Total number of lines is: 20

Process returned 0 (0x0) execution time : 5.927 s

Press any key to continue.