CS2263 Assignment 3

Kohdy Nicholson

Design of my program in steps:

1. Gather html input from stdin, store in the input array.
2. Perform one full loop over input, removing all html comments from the input array.
3. Perform another full loop over input, identifying all remaining html tags
    a. If an html tag is encountered, record the first occurrence in the index table and set the occurrence to 1.
        i. From this point, find other occurrences of specific tag the rest of the input array.
        ii. If another occurrence is found, increase occurrence of this tag by 1 and remove this occurrence from the string, as it has now been processed.
4. After the all html tags have been identified and counter in the index table and occurrence array, display the results.

About my functions:

- is_letter(): checks if the given character is a letter. This is because all html tags begin with a letter.
- is_num(): checks if the given character is a number. This is because some html tags have numbers in them such as h1-h6.
- is_match(): this check whether the target html tag is a match to the source html tag, then removes the target html tag from the string as it now has been processed. Compares two html tags, removes one of them.
- find_occurrences(): this is where a big majority of the magic happens. This function is triggered after we first find an html tag in the main() function of htags. What this does, is it looks for another occurrence of an html tag, from the occurrence of the first html tag which it was triggered by, using the is_letter() function. So, in main() we find stop once we find <p, then we trigger the find_occurrences() starting from <p onwards. If we hit the closing brace for our <p, we will skip it as </ is not a < followed by a letter, but rather is followed by a /. Therefore, if and only if we hit another opening <p, we will increase the occurrence. At the same time, our is_match() function will be cleaning up all of the matched occurences so we do not have to process them a second time.
- is_comment(): This checks if we have hit a comment. We use this before any html tag processing is done, so we can strip out the comments beforehand.
- remove_comment(): If is_comment() is equal to 1, we remove the contents inside the comment using the pointer which points to the beginning of the comment.

I used character pointers and characters for processing everything outside of the input string. My index table is an array of character pointers, which hold the address to the beginning of an html tag in the input string. My occurrence table works in parallel with the index table, and is an array of integers which contain the frequency at which each html tag occurs. The index table, occurrence table, and html input

array are all stored on the heap. All other character pointers are used in the stack frames in order to keep track of the position in the index table, occurrence table, and the input string. E.g.

```
<!DOCTYPE html>
<html lang = "en"> <!-- attribute that the langage is english -->
 ^
This memory address at this position in the input array is
stored in the index table, occurrence is set to 1.
    <head>
        <meta charset="utf-8">
        <title>Hello (Suessian) world!</title>
    </head>
    <body>
    <strong>Hello!</strong>
    <ol> <!-- what if these are <ol></ol> tags? -->
              ^^^^^^^^^^^^^^^^^^^^^^^^^
             The contents between <!-- --> are removed.
        <li> fish </li>
        ^^^
        This <li> will be recorded in the index table.
        <li> <blink>fish</blink> </li>
        ^^^
        The occurrence will be incremented for <li> and this
occurrence will be removed, so we do not process it again.
    </ol>
    <p style="color:red"> fish </p> <!-- specify colour -->
    <p style="color:blue"> fish </p>
    </body>
</html>
```

htags.c source:

```
/*
    htags.c

    Description:
    Program that counts the occurence of html tags from std input.

    Author:
    Kohdy Nicholson

    Date:
    2020-05-24
*/
```

```c
#include "parser.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char** argv){
    // Only allowable data structure for holding a string
    int n = 100000;
    char* contents = (char*)malloc(n);
    if(contents == (char*)NULL){
        fprintf(stderr,"Memory failure, terminating");
        return EXIT_FAILURE;
    }

    // Table of character pointers to the first occurence of html tags
    char** index_table = (char**)malloc(100);
    if(index_table == (char**)NULL){
        fprintf(stderr,"Memory failure, terminating");
        return EXIT_FAILURE;
    }

    // Parallel integer table for the amount of occurences of each html tag in th
e index table
    int* occurences = (int*)malloc(100);
    if(occurences == (int*)NULL){
        fprintf(stderr,"Memory failure, terminating");
        return EXIT_FAILURE;
    }

    // base pointers for the html contents, index table and occurences table.
    char* contents_base_ptr = contents;
    int* occurence_base_ptr = occurences;
    char** index_base_ptr = index_table;

    // Stack pointers for the occurence and index tables
    int* occurrence_stack_ptr = occurences;
    char** index_stack_ptr = index_table;

    // Stack pointer to be used by the program for stack sizes
    char* stack_size;

    // Size of the contents string
    int size;
```

```
    // variable for storing individual characters into the contents string
    char input;

    /*

    *** STEP ONE: FILL THE contents ARRAY WITH HTML FILE CONTENTS ***

    */
    while(scanf("%c", &input) != EOF){
        *contents_base_ptr = input;
        contents_base_ptr++;
    }

    /*

    *** STEP TWO: REMOVE CONTENT WITHIN COMMENT TAGS ***

    */
    size = strlen(contents);
    contents_base_ptr = contents;
    stack_size = contents_base_ptr + size;
    while(contents_base_ptr < stack_size){ // First we need to remove comments fr
om the html contents.
        if(*contents_base_ptr == '<'){
            if(is_comment(contents_base_ptr)){
                remove_comment(contents_base_ptr + 4); // We add 4 to the base po
inter address to skip the comment start '<!--'
            }
        }
        contents_base_ptr++;
    }

    /*

    *** STEP THREE: FIND FIRST OCCURENCE OF HTML TAGS ***

    */
    contents_base_ptr = contents;
    while(contents_base_ptr < stack_size){
        if(*contents_base_ptr == '<'){
            if(is_letter(*(contents_base_ptr+1)) == 1){

                /*
```

```
                *** STEP FOUR: FIND OTHER OCCURENCES OF HTML TAG. REMOVE TAGS ONC
E CONFIRMED.

                */
                *index_stack_ptr = contents_base_ptr + 1;
                *occurrence_stack_ptr = find_occurrences(contents, *index_stack_p
tr);
                occurrence_stack_ptr++;
                index_stack_ptr++;
            }
        }
        contents_base_ptr++;
    }

    /*

    *** STEP FIVE: PRINT DETAILS FOR HTML TAGS ***

    */
    char* char_ptr;
    while(index_base_ptr != index_stack_ptr){
        char_ptr = *index_base_ptr;
        while(is_letter(*char_ptr) == 1 || is_num(*char_ptr) == 1){
            printf("%c", *char_ptr);
            char_ptr++;
        }

        printf("\t\t%d\n", *occurence_base_ptr);

        index_base_ptr++;
        occurence_base_ptr++;
    }

    free(contents);
    free(index_table);
    free(occurences);

    return EXIT_SUCCESS;
}
```

parser.c source:

```c
/*
    parser.c

    Description:
    Functions for parsing html elements.

    Author:
    Kohdy Nicholson

    Date:
    2020-05-24
*/

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

/*
Function to check if the pointer contains a letter value
*/
int is_letter(char ch){
    int result = 0;
    if((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z')) result = 1;
    return result;
}

/*
Function to check if the pointer contains a number value
*/
int is_num(char ch){
    int result = 0;
    if(ch >= '0' && ch <= '9') result = 1;
    return result;
}

/*
Function which matches an html tag, then removes it.
*/
int is_match(char* source, char* target){
```

```
    char* tgt_base_ptr = target;
    int result = 0;

    // compare characters from the source to the target, based on letters and num
bers.
    while(is_letter(*source) == 1 || is_num(*source) == 1){
        if(*source == *target){
            result = 1;
        }else{
            result = 0;
            break; // break if a difference is found. not a match.
        }
        source++;
        target++;
    }

    // If everything in source is in target
    if(result == 1){
        if(is_letter(*target) == 1 || is_num(*target) == 1){
            result = 0; // Extra check incase source is something like 'col', and
 target is 'colgroup'
        }else{
            while(tgt_base_ptr != target){
                target--;
                *target = ' '; // Begin removing the duplicate tag
            }
        }
    }

    return result;
}

/*
Function for finding the occurences of an html tag.
*/
int find_occurrences(char* contents, char* current){
    int size = strlen(contents);
    int occured = 1;
    int i = (int)(current - contents);

    contents = current;
    while(i < size){
        if(*contents == '<'){
            if(is_letter(*(contents + 1)) == 1){
                if(is_match(current, (contents + 1)) == 1){
```

```
                    occured++;
                }
            }
        }
        contents++;
        i++;
    }

    return occured;
}

/*
Function for checking to see if we have encountered a comment.
*/
int is_comment(char* start){
    int result = 0;

    if(*start == '<'){
        start++;
        if(*start == '!'){
            start++;
            if(*start == '-'){
                start++;
                if(*start == '-'){
                    result = 1;
                }
            }
        }
    }

    return result;
}

/*
Function for removing comment contents. Loop will only terminate when we reach th
e comment ending '-->'.
*/
void remove_comment(char* start){
    char* ending_dash = start + 1;
    char* ending_arrow = ending_dash + 1;
    while(*start != '-' || *ending_dash != '-' || *ending_arrow != '>'){

        *start = ' '; // This will begin removing everything inside the comment.

        start++;
```

```
        ending_dash++;
        ending_arrow++;
    }
}

/*
Function to print out the name and occurences of the tag
*/
void print_tag_details(char* tag, int* occured){
    while(is_letter(*tag) == 1){
        printf("%c", *tag);
        tag++;
    }

    printf("\t\t%d\n", *occured);
}
```

Output on HelloWorld.html:

```
PS C:\Users\Kohdy\Documents\cs2263\W3Ass> Get-Content .\HelloWorld.html | .\
htags.exe
html            1
head            1
meta            1
title           1
body            1
p               1
PS C:\Users\Kohdy\Documents\cs2263\W3Ass>
```

Output on Sample.html:

```
PS C:\Users\Kohdy\Documents\cs2263\W3Ass> gcc .\htags.c .\parser.c -o htags
PS C:\Users\Kohdy\Documents\cs2263\W3Ass> Get-Content .\Sample.html | .\htag
s.exe
html            1
head            1
meta            1
title           1
body            1
strong          1
ol              1
li              2
blink           1
p               2
PS C:\Users\Kohdy\Documents\cs2263\W3Ass>
```

Output on David Bremner's CS3613 website home page, with the whole website also contained inside a comment (websiteception):

```
PS C:\Users\Kohdy\Documents\cs2263\W3Ass> gcc .\htags.c .\parser.c -o htags
PS C:\Users\Kohdy\Documents\cs2263\W3Ass> Get-Content .\cs3613.html | .\htag
s.exe
html            1
head            1
meta            1
title           1
link            4
body            1
div             13
span            6
a               39
ul              5
li              21
table           1
tr              6
td              12
strong          6
p               9
h2              2
br              4
i               4
h3              2
PS C:\Users\Kohdy\Documents\cs2263\W3Ass>
```

Testing remove_comment() function and program:

```
[knichol1@id415m13 W3Ass]$ make
available command:
        make help                   (this command)
        make htags                  (to build your C program)
        make remove_comments_test   (to build the test C program)
        make test                   (to run every test case)
        make compound_test          (to run htags compound tests)
        make remove_comment_test    (to run test cases against the function to remove
_comments)
[knichol1@id415m13 W3Ass]$ make test
gcc -Wall -c htags.c
gcc -Wall -c parser.c
gcc -Wall -o htags htags.o parser.o
./htags < Data/Sample.html > Sample_test.result
./TestPassed.sh Sample_test.result Data/Sample_test.expected

######   Passed   ###### Sample_test.result is equal to Data/Sample_test.expected

./htags < Data/HelloWorld.html > HelloWorld_test.result
./TestPassed.sh HelloWorld_test.result Data/HelloWorld_test.expected

######   Passed   ###### HelloWorld_test.result is equal to Data/HelloWorld_test.expe
cted

./htags < Data/hello.html > hello_test.result
./TestPassed.sh hello_test.result Data/hello_test.expected

######   Passed   ###### hello_test.result is equal to Data/hello_test.expected

gcc -Wall -c remove_comments_test.c
gcc -Wall -o remove_comments_test remove_comments_test.o parser.o
./remove_comments_test < Data/Sample.html > remove_comment_test1.result
./TestPassed.sh remove_comment_test1.result Data/remove_comment_test1.expected

######   Passed   ###### remove_comment_test1.result is equal to Data/remove_comment_
test1.expected

./remove_comments_test < Data/HelloWorld.html > remove_comment_test2.result
./TestPassed.sh remove_comment_test2.result Data/remove_comment_test2.expected

######   Passed   ###### remove_comment_test2.result is equal to Data/remove_comment_
test2.expected

[knichol1@id415m13 W3Ass]$
```

remove_comment_test1.result output:

```
[knichol1@id415m13 W3Ass]$ cat remove_comment_te
remove_comment_test1.result  remove_comment_test2.result
[knichol1@id415m13 W3Ass]$ cat remove_comment_test1.result
<!DOCTYPE html>
<html lang = "en"> <!--                                          -->
        <head>
                <meta charset="utf-8">
                <title>Hello (Suessian) world!</title>
        </head>
        <body>
        <strong>Hello!</strong>
        <ol> <!--                                        -->
                <li> fish </li>
                <li> <blink>fish</blink> </li>
        </ol>
        <p style="color:red"> fish </p> <!--                  -->
        <p style="color:blue"> fish </p>
        </body>
</html>[knichol1@id415m13 W3Ass]$
```