Question 1

Problem Definition/Specification

A program is needed which will take a file input and print out 5 tables which display various characteristics of the words in the file. For specific details, refer to textbook Exercise 8.31 and to the Assignment 2 handout for CS 350.

Problem Analysis

The program must read in a collection of text from an input text file and process the information to generate 5 tables for the output. The first of these tables must print out the number of occurrences of each letter of the alphabet within the input file. The second table must print out the number of occurrences of each length word found in the input text. For example, if there are 10 6-letter words in the input text, then 10 occurrences should be displayed next to the number 6. The third table must print out every word found within the text and display next to it the number of times that word can be found within the text. For this table, the words must be sorted in the order which they appeared in the input text file. The fourth table should include all the same information as the previous table; however, the entries of the table must be sorted by the words in alphabetical order. The fifth and final table must include the same entries as the previous two tables, however this time sorted by the number of occurrences of each word. Additionally, the program must allow the user to specify an output file optionally. If the output file is supplied, the tables will be printed to the output file, however if an output file is not supplied, the tables will be printed to the console. All the calculations necessary to generate the tables must be performed efficiently to allow the program to run in an acceptable amount of time when given large input files.

Program Inputs

- A file name for an input text file
- A file name for an output text file (optional)

Program Outputs

- A table listing letters versus occurrences
- A table listing word lengths versus occurrences
- A table listing words versus occurrences sorted in order of appearance
- A table listing words versus occurrences sorted by words
- A table listing words versus occurrences sorted by occurrences

Formula or Relations

There are 26 letters in the alphabet, and therefore an integer array of size 26 will be enough for tracking individual letter occurrences. Additionally, a Linked List is a list of objects that include information and point to other objects. They are made up of smaller structures typically called nodes, and these nodes can be simulated in C using structures and pointers.

Algorithm Design

Initial Algorithm

1. Read in the inputs from the user
2. Open a file at the specified input file name if it exists (if not, exit the program)
3. Create a new "head" node to the created Linked List within the program
4. Move through the input file and add words to the Linked List starting at the head of the List and working towards the end
5. If a node is found with the same word that is to be added, instead increase the number of occurrences at that node
6. While adding nodes to the Linked List, also increment array positions in arrays for tracking letter occurrences and word length occurrences
7. Close the file when all words have been processed
8. If an output file name is input, open a new file with the specified name and complete the following steps in that file; otherwise complete them at the console.
9. Print out the first table by outputting the values at each position in the letter occurrences array
10. Print out the second table by outputting the values at each position in the word length occurrences array
11. Print out the third table by printing out the values of the Linked List (words and occurrences), and add each to two additional new Linked Lists, which sort the values by word and occurrences respectively as they are added in
12. Use the new Linked List sorted by words to print out the fourth table
13. Use the new Linked List sorted by occurrences to print out the fifth table

Refinement

1. Determine the number of arguments input by the user
    a. If only 2 arguments are found, read in the input file name to a character array
    b. If 3 arguments are found, read in the output file name to a character array in addition to the input character array
2. Open the file at the given file name and exit the program if no such file exists
3. Create 2 integer arrays for tracking the number of occurrences of each letter (size 26) and each word length (size capped at 20)
4. Create the head node for the Linked List; see below for description of Linked List functionality
    a. Define the Linked List as a structure with a character array word, an integer for the number of occurrences of that word, and a pointer to the next List structure in the chain
    b. Create new nodes by using malloc() in a separate function
    c. Add nodes to an existing chain using another separate function by inputting the head reference to the function

    d. Within this function, loop through each element within the List structure by using the references to successive nodes within each node, and check each to determine whether the word at that node is the same as the one being added. If this is the case, do not add a new node and instead increment the occurrences value at that node
5. While the end of the file has not been reached, read in each word from the input text file and add it to the Linked List at the head reference
    a. Also loop through each individual word, and increment the values at each position in the alphabet occurrences array based on each character found
    b. Additionally, when the word length has been determined, increment the respective part of the word length occurrences array
    c. Each word will also be run through a separate function to convert it to lowercase (to ignore the case) and remove punctuation from the end of words
6. Close the input file
7. If an output file name was read in at the start of the program, complete the rest of the steps with reference to that output file, and otherwise complete them with reference to the console
    a. That is, there are two separate functions for printing out the tables, and the one selected depends on the existence of a given output file. One uses printf and the other uses fprintf with a FILE reference to the output file
8. Print out the first table by using the array which keeps track of the number of occurrences of each letter
9. Print out the second table by using the array which keeps track of the number of occurrences of each word length
10. Print out the third table by looping through the previously made Linked List and printing out each of the words and their corresponding numbers of occurrences
    a. Also create 2 new Linked Lists by adding them into two separate unique functions that add the nodes differently
    b. The first of the Lists has its members added using a function which sorts the nodes by the value of the words in each node as they are added to the List
    c. The second of the Lists has its members added using a function which sorts the nodes by the number of occurrences associated with each node as they are added to the List
11. Loop through the second created List (sorted by words) and print out the word and its corresponding occurrence at each node
12. Loop through the third created List (sorted by occurrences) and print out the number of occurrences and their corresponding words

Desk Checking/Test Plan

In order to properly test the program, a comprehensive input file should be used. To ensure that the numbers of occurrences can be thoroughly tested, it will be helpful to use an input with a decent amount of repeated words. As such, the input to test the program will be song lyrics, as they typically contain repetition. The lyrics to tests will come from the song “Smells Like Teen Spirit” by Nirvana. Correct output can be determined by examining the output of the tables. If the tables are cleanly printed and the number of occurrences appear correctly, it can be

determined that the program was correctly implemented. Additionally, it must be observed that the last 3 tables are sorted correctly and as discussed above. The program will also be tested twice, one with the output file specified and one without it.

Implementation

```c
/*
* Name: Kollin Labowski
* Date: October 26, 2020
* Class: CS 350
* Description: The following program takes text from a specified input file and prints out 5 tables displaying the
* contents of the file. The actual displays of each individual table are detailed below. The program makes use of a
* LinkedList data structure, which has been created using a structure. See below for more details on each feature.
*/

//Include statements used to provide various functionality to the program
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <ctype.h>

//Constants used mainly to standardize lengths for arrays
const int ALPHABET_SIZE = 26;
const int WORD_LENGTH = 20;
const int FILE_NAME_LENGTH = 40;

//Used to access both the input and output files
FILE *filePtr;
FILE *outPtr;

//Used to determine the number of occurrences of each letter and each word length respectively
int letterOccurences[26];
int lengthOccurences[20];

//Structure used to simulate a node in a LinkedList structure
struct List
{
    char word[20];
    int occurences;
    struct List* next;
};

//Used to refer to individual pieces of data within the List
typedef struct List* Node;

//Refers to the head of the main LinkedList
Node head;

//Function prototypes: see below for individual functionality
Node makeNode();
void addNode(Node startNode, char addedWord[WORD_LENGTH]);
void cleanWord(char inWord[WORD_LENGTH]);
void printTables();
void outputTables();
```

```
50
51  /*
52  * Main method. This is where the input and optional output files are read in and where the main
53  * LinkedList is created and filled with values.
54  */
55  void main(int argc, char *argv[])
56  {
57      //Initialize the input (and optionally output) file names
58      char inFileName[FILE_NAME_LENGTH];
59      for(int i = 0;  i < FILE_NAME_LENGTH; i++)
60      {
61          inFileName[i] = argv[1][i];
62      }
63      char outFileName[FILE_NAME_LENGTH];
64      if(argc == 3)
65      {
66          for(int i = 0;  i < FILE_NAME_LENGTH; i++)
67          {
68              outFileName[i] = argv[2][i];
69          }
70      }
71
72      //Open the input file if it exists
73      filePtr = fopen(inFileName, "r");
74      if(filePtr == NULL)
75      {
76          printf("Input file name is invalid");
77          return;
78      }
79
80      //Fill the tracking arrays with values to avoid errors
81      for(int i = 0; i < ALPHABET_SIZE; i++)
82          letterOccurences[i] = 0;
83
84      for(int i = 0; i < WORD_LENGTH; i++)
85          lengthOccurences[i] = 0;
86
87      //Begin to make the List
88      head = makeNode();
89      Node cur;
90      char curWord[WORD_LENGTH];
91      for(int i = 0; i < WORD_LENGTH; i++)
92          curWord[i] = ' ';
93
```

```
94      //Fill the List with values until the end of the file has been reached
95      while(!feof(filePtr))
96      {
97          fscanf(filePtr, "%s", curWord);
98          cleanWord(curWord);
99          for(int i = 0; curWord[i] != '\0'; i++)
100         {
101             if(!ispunct(curWord[i]) && curWord[i] != ' ')
102                 letterOccurences[curWord[i] - 97]++;
103             if(i + 1 < WORD_LENGTH && (curWord[i + 1] == ' ' || curWord[i + 1] == '\0'))
104                 lengthOccurences[i]++;
105         }
106         addNode(head, curWord);
107     }
108
109     fclose(filePtr);
110
111     cur = head;
112
113     //If the optional output file was given, print to the file; otherwise print to console
114     if(argc < 3)
115     {
116         printTables();
117     }
118     else
119     {
120         outPtr = fopen(outFileName, "w");
121         if(outPtr == NULL)
122         {
123             printf("Output file name is invalid");
124             return;
125         }
126         outputTables();
127     }
128 }
129
130 /*
131 * This method allocates memory to create a new node, and is called every time a new one is made
132 */
133 Node makeNode()
134 {
135     Node temp;
136     temp = (Node)malloc(sizeof(struct List));
137     temp->next = NULL;
138 }
139
```

```
/*
* This method adds a node to the main List, at the end of the List. If the value is already in the List,
* it instead increments the occurences field.
*/
void addNode(Node startNode, char addedWord[WORD_LENGTH])
{
    Node temp, current;
    temp = makeNode();
    for(int i = 0; i < WORD_LENGTH; i++)
    {
        temp->word[i] = addedWord[i];
    }
    if(startNode == NULL)
        startNode = temp;
    else
    {
        current = startNode;
        while(current->next != NULL)
        {
            current = current->next;

            if(strcmp(current->word, temp->word) == 0)
            {
                current->occurences = current->occurences + 1;
                temp = NULL;
                return;
            }
        }
        current->next = temp;
    }
    temp -> occurences = 1;
}

/*
* Converts an input word to a form usable by the program, so lowercase and omitting punctuation at the end
*/
void cleanWord(char inWord[WORD_LENGTH])
{
    for(int i = 0; i < WORD_LENGTH; i++)
    {
        inWord[i] = tolower(inWord[i]);
        if(inWord[i] == '\0' && i - 1 >= 0 && ispunct(inWord[i - 1]))
        {
            inWord[i] = ' ';
            inWord[i - 1] = '\0';
            break;
        }
    }
}

```

```
/*
* Adds a node to a new List which sorts in alphabetical order by word as nodes are added in
*/
void wordAddNode(Node startNode, Node addNode)
{
    Node temp = startNode;
    while(temp -> next != NULL && strcmp((temp -> next) -> word, addNode -> word) < 0)
    {
        temp = temp -> next;
    }
    //printf("\n%s\n", temp -> word);
    Node placeholder = temp -> next;
    temp -> next = addNode;
    addNode -> next = placeholder;
}

/*
* Similar to the previous method, however instead sorts the new List by the occurences variable as values are added
*/
void numAddNode(Node startNode, Node addNode)
{
    Node temp = startNode;
    while(temp -> next != NULL && (temp -> next) -> occurences < addNode -> occurences)
    {
        temp = temp -> next;
    }
    //printf("\n%s\n", temp -> word);
    Node placeholder = temp -> next;
    temp -> next = addNode;
    addNode -> next = placeholder;
}
```

```
/*
 * Prints out all 5 tables to the console
 */
void printTables()
{
    //TABLE 1A
    //Table A prints the number of occurences of each letter of the alphabet

    printf("\nTable 1a:\n");
    printf("----------------\n");

    for(int i = 0; i < ALPHABET_SIZE; i++)
    {
        printf("| %c | %8d |\n", i + 65, letterOccurences[i]);
    }

    printf("----------------\n");

    //TABLE 1B
    //Table B prints the number of occurences of each word length, capping the table at the highest length

    //Find highest nonzero amount
    int cap = WORD_LENGTH - 1;

    while(cap - 1 >= 0 && lengthOccurences[cap - 1] == 0)
    {
        cap--;
    }

    printf("\nTable 1b:\n");
    printf("-----------------\n");

    for(int i = 0; i < cap; i++)
    {
        printf("| %2d | %8d |\n", i + 1, lengthOccurences[i]);
    }

    printf("-----------------\n");
```

```
260
261     //TABLE 1C
262     //Table C prints the number of occurences of each word sorted by order of appearence
263
264     printf("\nTable 1c:\n");
265
266     Node cur;
267     cur = head;
268
269     Node newHead = makeNode();
270     Node newHead2 = makeNode();
271
272     printf("-----------------------------------\n");
273
274     while(cur->next != NULL)
275     {
276         cur = cur->next;
277         printf("| %20s | %8d |\n", cur->word, cur->occurences);
278
279         Node tempNode = makeNode();
280         Node tempNode2 = makeNode();
281
282         for(int i = 0; i < WORD_LENGTH; i++)
283             tempNode -> word[i] = cur -> word[i];
284         tempNode -> occurences = cur -> occurences;
285         wordAddNode(newHead, tempNode);
286
287         for(int i = 0; i < WORD_LENGTH; i++)
288             tempNode2 -> word[i] = cur -> word[i];
289         tempNode2 -> occurences = cur -> occurences;
290         numAddNode(newHead2, tempNode2);
291     }
292
293     printf("-----------------------------------\n");
294
```

```
294
295         //TABLE 1D
296         //Table D prints the number of occurences of each word sorted in alphabetical order
297
298         printf("\nTable 1d:\n");
299
300         printf("-----------------------------------\n");
301
302         cur = newHead;
303
304         while(cur->next != NULL)
305         {
306             cur = cur->next;
307             printf("| %20s | %8d |\n", cur->word, cur->occurences);
308         }
309
310         printf("-----------------------------------\n");
311
312         //TABLE 1E
313         //Table E prints the number of occurences of each word sorted by number of occurences
314
315         printf("\nTable 1e:\n");
316
317         printf("-----------------------------------\n");
318
319         cur = newHead2;
320
321         while(cur->next != NULL)
322         {
323             cur = cur->next;
324             printf("| %20s | %8d |\n", cur->word, cur->occurences);
325         }
326
327         printf("-----------------------------------\n");
328     }
329
```

```c
/*
* Output the tables to the given output file
*/
void outputTables()
{
    //TABLE 1A
    //Table A prints the number of occurences of each letter of the alphabet

    fprintf(outPtr, "\nTable 1a:\n");
    fprintf(outPtr, "----------------\n");

    for(int i = 0; i < ALPHABET_SIZE; i++)
    {
        fprintf(outPtr, "| %c | %8d |\n", i + 65, letterOccurences[i]);
    }

    fprintf(outPtr, "----------------\n");

    //TABLE 1B
    //Table B prints the number of occurences of each word length, capping the table at the highest length

    //Find highest nonzero amount
    int cap = WORD_LENGTH - 1;

    while(cap - 1 >= 0 && lengthOccurences[cap - 1] == 0)
    {
        cap--;
    }

    fprintf(outPtr, "\nTable 1b:\n");
    fprintf(outPtr, "-----------------\n");

    for(int i = 0; i < cap; i++)
    {
        fprintf(outPtr, "| %2d | %8d |\n", i + 1, lengthOccurences[i]);
    }

    fprintf(outPtr, "-----------------\n");

```

```
369     //TABLE 1C
370     //Table C prints the number of occurences of each word sorted by order of appearence
371
372     fprintf(outPtr, "\nTable 1c:\n");
373
374     Node cur;
375     cur = head;
376
377     Node newHead = makeNode();
378     Node newHead2 = makeNode();
379
380     fprintf(outPtr, "----------------------------------\n");
381
382     while(cur->next != NULL)
383     {
384         cur = cur->next;
385         fprintf(outPtr, "| %20s | %8d |\n", cur->word, cur->occurences);
386
387         Node tempNode = makeNode();
388         Node tempNode2 = makeNode();
389
390         for(int i = 0; i < WORD_LENGTH; i++)
391             tempNode -> word[i] = cur -> word[i];
392         tempNode -> occurences = cur -> occurences;
393         wordAddNode(newHead, tempNode);
394
395         for(int i = 0; i < WORD_LENGTH; i++)
396             tempNode2 -> word[i] = cur -> word[i];
397         tempNode2 -> occurences = cur -> occurences;
398         numAddNode(newHead2, tempNode2);
399     }
400
401     fprintf(outPtr, "----------------------------------\n");
402
```

```
403         //TABLE 1D
404         //Table D prints the number of occurences of each word sorted in alphabetical order
405 
406         fprintf(outPtr, "\nTable 1d:\n");
407 
408         fprintf(outPtr, "-----------------------------------\n");
409 
410         cur = newHead;
411 
412         while(cur->next != NULL)
413         {
414             cur = cur->next;
415             fprintf(outPtr, "| %20s | %8d |\n", cur->word, cur->occurences);
416         }
417 
418         fprintf(outPtr, "-----------------------------------\n");
419 
420         //TABLE 1E
421         //Table E prints the number of occurences of each word sorted by number of occurences
422 
423         fprintf(outPtr, "\nTable 1e:\n");
424 
425         fprintf(outPtr, "-----------------------------------\n");
426 
427         cur = newHead2;
428 
429         while(cur->next != NULL)
430         {
431             cur = cur->next;
432             fprintf(outPtr, "| %20s | %8d |\n", cur->word, cur->occurences);
433         }
434 
435         fprintf(outPtr, "-----------------------------------\n");
436 
437         printf("Successfully printed to file");
438     }
439     //End of Program
```

Testing

Test 1: (with output file)

```
C:\Users\kk_la\Desktop\350 Assignment2\Code>a inText.txt outText.txt
Successfully printed to file
```

Note: The following outputs are found in the output file outText.txt

Table 1a:

A	77
B	11
C	3
D	41
E	132
F	12
G	15
H	77
I	68
J	1
K	2
L	123
M	14
N	62
O	109
P	6
Q	3
R	39
S	54
T	65
U	35
V	3
W	43
X	0
Y	13
Z	0

Table 1b:

1	26
2	32
3	54
4	42
5	48
6	25
7	6
8	4
9	9
10	4
11	0
12	1

Table 1c:

load	1
up	1
on	1
guns	1
bring	1
your	1
friends	1
it's	5
fun	1
to	3
lose	1
and	8
pretend	1
she's	1
over-bored	1
self-assured	1
oh	3
no	1
i	10
know	1
a	16
dirty	1
word	1
hello	36
how	9
low	9
with	3
the	4
lights	3
out	3
less	3
dangerous	3
here	6
we	6
are	6
now	6
entertain	6

us	6
feel	4
stupid	3
contagious	3
mulatto	3
an	3
albino	3
mosquito	3
my	3
libido	3
yeah	3
hey	2
i'm	1
worse	1
at	1
what	1
do	1
best	1
for	1
this	1
gift	1
blessed	1
our	1
little	1
group	1
has	1
always	2
been	1
will	1
until	1
end	1
forget	1
just	1
why	1
taste	1
guess	1
it	2
makes	1
me	1

smile	1
found	1
hard	2
find	1
well	1
whatever	1
never	1
mind	1
denial	9

Table 1d:

a	16
albino	3
always	2
an	3
and	8
are	6
at	1
been	1
best	1
blessed	1
bring	1
contagious	3
dangerous	3
denial	9
dirty	1
do	1
end	1
entertain	6
feel	4
find	1
for	1
forget	1
found	1
friends	1
fun	1
gift	1
group	1
guess	1
guns	1
hard	2
has	1
hello	36
here	6
hey	2
how	9
i	10
i'm	1
it	2
it's	5

just	1
know	1
less	3
libido	3
lights	3
little	1
load	1
lose	1
low	9
makes	1
me	1
mind	1
mosquito	3
mulatto	3
my	3
never	1
no	1
now	6
oh	3
on	1
our	1
out	3
over-bored	1
pretend	1
self-assured	1
she's	1
smile	1
stupid	3
taste	1
the	4
this	1
to	3
until	1
up	1
us	6
we	6
well	1
what	1
whatever	1
why	1
will	1
with	3
word	1
worse	1
yeah	3
your	1

Table 1e:

mind	1
never	1
whatever	1
well	1
find	1
found	1
smile	1
me	1
makes	1
guess	1
taste	1
why	1
just	1
forget	1
end	1
until	1
will	1
been	1
has	1
group	1
little	1
our	1
blessed	1
gift	1
this	1
for	1
best	1
do	1
what	1
at	1
worse	1
i'm	1
word	1
dirty	1
know	1
no	1
self-assured	1

over-bored	1
she's	1
pretend	1
lose	1
fun	1
friends	1
your	1
bring	1
guns	1
on	1
up	1
load	1
hard	2
it	2
always	2
hey	2
yeah	3
libido	3
my	3
mosquito	3
albino	3
an	3
mulatto	3
contagious	3
stupid	3
dangerous	3
less	3
out	3
lights	3
with	3
oh	3
to	3
feel	4
the	4
it's	5
us	6
entertain	6
now	6
are	6

```
|                    we |          6 |
|                  here |          6 |
|                   and |          8 |
|                denial |          9 |
|                   low |          9 |
|                   how |          9 |
|                     i |         10 |
|                     a |         16 |
|                 hello |         36 |
-------------------------------------
```

Test 2: (No output file specified)

```
C:\Users\kk_la\Desktop\350 Assignment2\Code>a inText.txt

Table 1a:
---------------
| A |       77 |
| B |       11 |
| C |        3 |
| D |       41 |
| E |      132 |
| F |       12 |
| G |       15 |
| H |       77 |
| I |       68 |
| J |        1 |
| K |        2 |
| L |      123 |
| M |       14 |
| N |       62 |
| O |      109 |
| P |        6 |
| Q |        3 |
| R |       39 |
| S |       54 |
| T |       65 |
| U |       35 |
| V |        3 |
| W |       43 |
| X |        0 |
| Y |       13 |
| Z |        0 |
---------------
```

```
Table 1b:
----------------
|  1 |       26 |
|  2 |       32 |
|  3 |       54 |
|  4 |       42 |
|  5 |       48 |
|  6 |       25 |
|  7 |        6 |
|  8 |        4 |
|  9 |        9 |
| 10 |        4 |
| 11 |        0 |
| 12 |        1 |
----------------
```

Table 1c:

load	1
up	1
on	1
guns	1
bring	1
your	1
friends	1
it's	5
fun	1
to	3
lose	1
and	8
pretend	1
she's	1
over-bored	1
self-assured	1
oh	3
no	1
i	10
know	1
a	16
dirty	1
word	1
hello	36
how	9
low	9
with	3
the	4
lights	3
out	3
less	3
dangerous	3
here	6
we	6
are	6
now	6
entertain	6
us	6
feel	4
stupid	3
contagious	3
mulatto	3
an	3
albino	3
mosquito	3
my	3
libido	3
yeah	3
hey	2
i'm	1
worse	1
at	1
what	1
do	1
best	1
for	1
this	1
gift	1
blessed	1

our	1
little	1
group	1
has	1
always	2
been	1
will	1
until	1
end	1
forget	1
just	1
why	1
taste	1
guess	1
it	2
makes	1
me	1
smile	1
found	1
hard	2
find	1
well	1
whatever	1
never	1
mind	1
denial	9

Table 1d:

a	16
albino	3
always	2
an	3
and	8
are	6
at	1
been	1
best	1
blessed	1
bring	1
contagious	3
dangerous	3
denial	9
dirty	1
do	1
end	1
entertain	6
feel	4
find	1
for	1
forget	1
found	1
friends	1
fun	1
gift	1
group	1
guess	1
guns	1
hard	2
has	1
hello	36
here	6
hey	2
how	9
i	10
i'm	1
it	2
it's	5
just	1
know	1
less	3
libido	3
lights	3
little	1
load	1
lose	1
low	9
makes	1
me	1
mind	1
mosquito	3
mulatto	3
my	3
never	1
no	1
now	6
oh	3
on	1
our	1

out	3
over-bored	1
pretend	1
self-assured	1
she's	1
smile	1
stupid	3
taste	1
the	4
this	1
to	3
until	1
up	1
us	6
we	6
well	1
what	1
whatever	1
why	1
will	1
with	3
word	1
worse	1
yeah	3
your	1

Table 1e:

mind	1
never	1
whatever	1
well	1
find	1
found	1
smile	1
me	1
makes	1
guess	1
taste	1
why	1
just	1
forget	1
end	1
until	1
will	1
been	1
has	1
group	1
little	1
our	1
blessed	1
gift	1
this	1
for	1
best	1
do	1
what	1
at	1
worse	1
i'm	1
word	1
dirty	1
know	1
no	1
self-assured	1
over-bored	1
she's	1
pretend	1
lose	1
fun	1
friends	1
your	1
bring	1
guns	1
on	1
up	1
load	1
hard	2
it	2
always	2
hey	2
yeah	3
libido	3
my	3
mosquito	3
albino	3

an	3
mulatto	3
contagious	3
stupid	3
dangerous	3
less	3
out	3
lights	3
with	3
oh	3
to	3
feel	4
the	4
it's	5
us	6
entertain	6
now	6
are	6
we	6
here	6
and	8
denial	9
low	9
how	9
i	10
a	16
hello	36

Question 2

Problem Definition/Specification

A program is needed that will process an image using an edge detection algorithm. Refer to the instructions in part 2 of the Assignment 2 handout.

Problem Analysis

The program must take an input image in portable pixel format (ppm) and process it in different ways to produce 6 new images. The first 3 of these images are to be generated using a Sobel edge detection algorithm. The equation for performing the edge detection is given in the Assignment 2 handout. The first of the 3 images to be generated with edge detection will use a horizontal mask to detect horizontal edges in the image. The second of the 3 will use a vertical mask to detect vertical edges. The third image of the 3 will be a combination of the horizontal and vertical edge detection images. The masks are to be 3x3 matrices, and they are given in the project handout. Of the total of 6 images, the latter 3 images will be binary images generated from the first 3 images. This will be done by calculating the mean and standard deviation of each of the previous 3 image arrays to find a threshold value. The images should then have their components set to 0 if the value at that component is less than the threshold value. This will produce 3 images; a horizontal binary image, a vertical binary image, and a combined horizontal vertical binary image. These images should all be saved into new ppm files and output as such. Additionally, the program should print out the average and standard deviation of each of the first 3 images as well as the input image.

Program Inputs

- A file name for an input ppm file
- A file name for a horizontal edge output ppm file
- A file name for a vertical edge output ppm file
- A file name for a combined edge output ppm file
- A file name for a horizontal binary output ppm file
- A file name for a vertical binary output ppm file
- A file name for a combined binary output ppm file

Program Outputs

- A ppm file with the input image processed as a horizontal edge
- A ppm file with the input image processed as a vertical edge
- A ppm file with the input image processed as a combined edge
- A ppm file with the input image processed as a horizontal binary
- A ppm file with the input image processed as a vertical binary
- A ppm file with the input image processed as a combined binary
- The average and standard deviation of the input image and 3 edge images

Formula or Relations

The following equation is given in the Assignment 2 handout and is used to perform Sobel edge detection:

$$E(x,y) = \sum_{p=-a}^{a} \sum_{q=-a}^{a} w(p,q) I(x+p, y+q),$$

where $a=(n-1)/2$.

The following equation shows the relation between the horizontal edge image, the vertical edge image, and the combined horizontal vertical edge image:

$$E(x,y) = |E_h(x,y)| + |E_v(x,y)|$$

The following matrices represent the masks that are used to perform the Sobel edge detection:

$$w_h = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \quad w_v = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

The following equations show how to calculate the threshold used for generating the binarized images, as well as how it should be used to determine values in the binarized image:

$$T = \mu + \acute{K}\sigma,$$

$$B(x,y) = \begin{cases} 1 & \text{if } E(x,y) \geq T \\ 0 & \text{otherwise} \end{cases}$$

This equation can be used to find the average of the values in one of the image matrices:

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i ;$$

This equation can be used to find the standard deviation of the values in one of the image matrices:

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \mu)^2}$$

Algorithm Design

Initial Algorithm

1. Determine whether the input was entered in the correct format, and exit the program if it was not
2. Read in the input image from a file using a separate function and set the image to a pointer
3. Create a new matrix for a new image and loop through it, performing the horizontal edge detection on each member in the original image and setting it equal to its corresponding position in the new matrix
4. Save the resulting matrix to a new image file using a separate function
5. Create a new matrix for a new image and loo through it, this time instead performing the vertical edge detection on each member in the original image and setting it equal to its corresponding position in the new matrix
6. Save the resulting matrix to a new image file
7. Create a new matrix for a new image, and set each of its positions to the sum of the corresponding positions in the previous two matrices
8. Save this new matrix to an image file
9. Calculate the mean and standard deviation of the input image as well as each of the previous 3 matrices' values
10. Calculate the threshold values of the 3 edge images
11. Create a new matrix and set its values to the max brightness value where the value at the given location is greater than or equal to the threshold value for the horizontal edge image
12. Save this new image to a file
13. Create another new matrix and repeat step 11, but with the vertical edge image
14. Save this new image to a file
15. Create another new matrix and repeat step 11 once more, but with the combined edge image
16. Save this image to a file

Refinement

1. Determine whether the correct number of arguments were passed in by the user, and if not, quit the program
2. Use a separate function to read in the input image at the given file name and set it to a pointer for keeping track of
    a. In the separate function, create a new FILE reference using the given input file name
    b. Determine whether the file is the correct type of file
    c. Allocate memory for the image using the malloc() method
    d. Return the value of the image as a pointer to the image
3. Create a new 2D unsigned character array of the same size as the image pointer

4. Loop through this new array, and for each position in the array, create a new 3x3 array with the values surrounding it and pass it into a separate function for horizontal edge detection
    a. Within this function use a mask with the given horizontal matrix from the Formulas or Relations section defined as a 2D array
    b. Loop through both arrays and add up the products of each corresponding position in the two arrays
    c. Return the calculated sum
5. Write the new array to a file as an image, using a separate function
    a. Create a new FILE reference with the input name for the output file, this file name being the second file name input
    b. Determine whether the file was successfully created
    c. Close the file when it has been written to
6. Repeat steps 3-5 using the vertical mask for vertical edge detection instead, and using the third file name that was input as the file to output the image to
7. Create another array of the same size as the original input image
8. Set each position in this 2D array to the sum of the corresponding positions in the previous 2 image arrays (horizontal and vertical) using a nested for loop
9. Write this function to a new file with the next output file name from the input arguments, using the same function as the previous 2 written images
10. Determine the average of the values at the points in the input image as well as the 3 edge detection images using a nested loop that loops through the 2D arrays to find the sum of all values. Then outside the loop, divide each of the sums by the size of the total matrix
11. Determine the standard deviations of each of the matrices again using nested loops, this time following the standard deviation equation in the Formulas and Relations section
12. Determine the threshold values for each image by adding their respective averages and standard deviations
13. Create a new unsigned character 2D array of the same size as the previous ones, and at each point in the array, determine whether the corresponding point in the horizontal edge matrix is greater than or equal to the threshold value, and if so set the value at the point equal to 232. Otherwise set it to 0
14. Save this new image to another file using the next file name from the list of arguments
15. Repeat steps 13 and 14 using instead the vertical edge matrix and its corresponding threshold value
16. Repeat steps 13 and 14 again using instead the combined edge matrix and its corresponding threshold value
17. Print out the average and standard deviation of the input image as well as the 3 edge detection images

Desk Checking/Test Plan

This program will be checked using one of the images provided on the eCampus page for the class. This image will be used as the input in the arguments list. Determining whether the code works correctly can be accomplished by examining the 6 output images and determining whether they appear correctly. The horizontal edge detection image should detect most horizontal edges whereas the vertical edge detection image should detect most vertical edges. The binarized images should include only 2 shades and should show the outline of the edges in the image. The image to be used for testing is seen below.

Implementation

```c
/*
* Name: Kollin Labowski
* Date: October 27, 2020
* Course: CS 350
* Description: This program takes in an input image and outputs 6 images. The first 3 of these images use edge detection
* and the latter 3 use a threshold to binarize the images. See code below for more thorough explanation on the functionality
* of each section.
*/

//Include statements
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

//Constant for determining array sizes
const int MASK_SIZE = 3;

int SIZE, TYPE;

typedef unsigned char *image_ptr;

//Function prototypes
image_ptr readImage(char *fName, int *size, int *type);
int getNum(FILE *myPtr);
void writeImage(image_ptr imgPtr, char *fName, int size, int keyNum);
int maskComputeH(int arr[][3]);
int maskComputeV(int arr[][3]);

/*
* Main method: This method finds the file names to read/write to and performs necessary processing
* using some additional functions
*/
void main(int argc, char **argv)
{
    //Determine whether the correct amount of arguments are input
    if(argc != 8)
    {
        printf("Incorrect input format: use %s imageIn.ppm outEh.ppm outEv.ppm outE.ppm outBh.ppm outBv.ppm outB.ppm", argv[0]);
        return;
    }

    int size, type;

    //READ INPUT IMAGE

    image_ptr imagePtr = readImage(argv[1], &size, &type);

    unsigned char imageA[size][size];

```

```
50      for(int i = 0; i < size; i++)
51      {
52          for(int j = 0; j < size; j++)
53          {
54              int temp[MASK_SIZE][MASK_SIZE];
55              for(int k = 0; k < MASK_SIZE; k++)
56              {
57                  if(i - 1 < 0 || i + 1 >= size || j - 1 < 0 || j + 1 >= size)
58                      break;
59
60                  for(int l = 0; l < MASK_SIZE; l++)
61                  {
62                      temp[k][l] = imagePtr[(i - 1 + k) * size + (j - 1 + l)];
63                  }
64              }
65              int tempInt = maskComputeH(temp);
66              imageA[i][j] = tempInt;
67          }
68      }
69
70      //BEGIN FIRST OUTPUT
71      //Edge detection horizontally
72
73      image_ptr imagePtr2 = (image_ptr) imageA;
74
75      writeImage(imagePtr2, argv[2], size, type);
76
77      unsigned char imageB[size][size];
78
79      for(int i = 0; i < size; i++)
80      {
81          for(int j = 0; j < size; j++)
82          {
83              int temp[MASK_SIZE][MASK_SIZE];
84              for(int k = 0; k < MASK_SIZE; k++)
85              {
86                  if(i - 1 < 0 || i + 1 >= size || j - 1 < 0 || j + 1 >= size)
87                      break;
88
89                  for(int l = 0; l < MASK_SIZE; l++)
90                  {
91                      temp[k][l] = imagePtr[(i - 1 + k) * size + (j - 1 + l)];
92                  }
93              }
94              int tempInt = maskComputeV(temp);
95              imageB[i][j] = tempInt;
96          }
97      }
```

```
 98
 99      //BEGIN SECOND OUTPUT
100      //Edge detection vertically
101
102      image_ptr imagePtr3 = (image_ptr) imageB;
103
104      writeImage(imagePtr3, argv[3], size, type);
105
106      unsigned char imageC[size][size];
107
108      for(int i = 0; i < size; i++)
109      {
110          for(int j = 0; j < size; j++)
111          {
112              int tempInt = imageA[i][j] + imageB[i][j];
113              imageC[i][j] = tempInt;
114          }
115      }
116
117      //BEGIN THIRD OUTPUT
118      //Edge detection combined AND gather information for images 4-6
119
120      image_ptr imagePtr4 = (image_ptr) imageC;
121
122      writeImage(imagePtr4, argv[4], size, type);
123
124      //Used for binarized images
125      int avgI = 0;
126      int avgA = 0;
127      int avgB = 0;
128      int avgC = 0;
129
130      for(int i = 0; i < size; i++)
131      {
132          for(int j = 0; j < size; j++)
133          {
134              avgI += imagePtr[(i * size) + j];
135              avgA += imageA[i][j];
136              avgB += imageB[i][j];
137              avgC += imageC[i][j];
138          }
139      }
140
141      avgI /= (size * size);
142      avgA /= (size * size);
143      avgB /= (size * size);
144      avgC /= (size * size);
145
```

```
146     //Used for binarized images
147     int stdevI = 0;
148     int stdevA = 0;
149     int stdevB = 0;
150     int stdevC = 0;
151
152     for(int i = 0; i < size; i++)
153     {
154         for(int j = 0; j < size; j++)
155         {
156             stdevI += ((imagePtr[i * size + j] - avgI) * (imagePtr[i * size + j] - avgI));
157             stdevA += ((imageA[i][j] - avgA) * (imageA[i][j] - avgA));
158             stdevB += ((imageB[i][j] - avgB) * (imageB[i][j] - avgB));
159             stdevC += ((imageC[i][j] - avgC) * (imageC[i][j] - avgC));
160         }
161     }
162
163     stdevI /= ((size * size) - 1);
164     stdevI = (int)sqrt(stdevI);
165     stdevA /= ((size * size) - 1);
166     stdevA = (int)sqrt(stdevA);
167     stdevB /= ((size * size) - 1);
168     stdevB = (int)sqrt(stdevB);
169     stdevC /= ((size * size) - 1);
170     stdevC = (int)sqrt(stdevC);
171
172     int tA = avgA + stdevA;
173     int tB = avgB + stdevB;
174     int tC = avgC + stdevC;
175
176     //BEGIN OUTPUTS 4-6
177     //Binarized images that are horizontal, vertical, and combined
178
179     unsigned char imageD[size][size];
180     unsigned char imageE[size][size];
181     unsigned char imageF[size][size];
182
```

```
183     for(int i = 0; i < size; i++)
184     {
185         for(int j = 0; j < size; j++)
186         {
187             if(imageA[i][j] >= tA)
188                 imageD[i][j] = 232;
189             else
190                 imageD[i][j] = 0;
191             if(imageB[i][j] >= tB)
192                 imageE[i][j] = 232;
193             else
194                 imageE[i][j] = 0;
195             if(imageC[i][j] >= tC)
196                 imageF[i][j] = 232;
197             else
198                 imageF[i][j] = 0;
199         }
200     }
201 
202     image_ptr imagePtr5 = (image_ptr) imageD;
203     image_ptr imagePtr6 = (image_ptr) imageE;
204     image_ptr imagePtr7 = (image_ptr) imageF;
205 
206     writeImage(imagePtr5, argv[5], size, type);
207     writeImage(imagePtr6, argv[6], size, type);
208     writeImage(imagePtr7, argv[7], size, type);
209 
210     printf("\nInput Average: %d, Input Standard Deviation: %d\n", avgI, stdevI);
211     printf("Edge Response H Average: %d, Edge Response H Standard Deviation: %d\n", avgA, stdevA);
212     printf("Edge Response V Average: %d, Edge Response V Standard Deviation: %d\n", avgB, stdevB);
213     printf("Edge Response Combined Average: %d, Edge Response Combined Standard Deviation: %d\n", avgC, stdevC);
214 }
215 //End main method
```

```
216
217 /*
218 * This method reads in an image from a given file. It is used once by the main method to
219 * read the image at the input file name and prepare it for processing
220 */
221 image_ptr readImage(char *fName, int *size, int *type)
222 {
223     FILE *myPtr;
224
225     if((myPtr = fopen(fName, "rb")) == NULL)
226     {
227         printf("Unable to open file: %s\n", fName);
228         exit(1);
229     }
230
231     char charOne, charTwo;
232
233     charOne = getc(myPtr);
234     charTwo = getc(myPtr);
235
236     if(charOne != 'P')
237     {
238         printf("Input file is NOT a PPM file\n");
239         exit(1);
240     }
241
242     *size = getNum(myPtr);
243     *type = charTwo - '0';
244
245     SIZE = *size;
246     TYPE = *type;
247
248     int maxValue;
249     float scale;
250
```

```
251     switch(charTwo)
252     {
253         case '4':
254             scale = 0.125;
255             maxValue = 1;
256             break;
257         case '5':
258             scale = 1.0;
259             maxValue = getNum(myPtr);
260             break;
261         case '6':
262             scale = 3.0;
263             maxValue = getNum(myPtr);
264             break;
265         default :
266             printf("This is not a RAWBITS file\n");
267             exit(1);
268             break;
269     }
270 
271     int rowSize = (*size) * scale;
272     unsigned long totalSize = (unsigned long)(*size) * rowSize;
273 
274     image_ptr ptr = (image_ptr) malloc(totalSize);
275 
276     if(ptr == NULL)
277     {
278         printf("Unable to malloc for %lu bytes\n", totalSize);
279         exit(1);
280     }
281 
282     unsigned long totalBytes = 0;
283     unsigned long offset = 0;
284 
285     for(int i = 0; i < (*size); i++)
286     {
287         totalBytes += fread(ptr + offset, 1, rowSize, myPtr);
288         offset += rowSize;
289     }
290 
291     if(totalSize != totalBytes)
292     {
293         printf("Failed to read %ld bytes\nRead %ld bytes\n", totalSize, totalBytes);
294         exit(1);
295     }
296 
297     fclose(myPtr);
298     return ptr;
299 }
300 
```

```
/*
* A helper function used by the above readImage function
*/
int getNum(FILE *myPtr)
{
    char ch;

    do
    {
        ch = getc(myPtr);
    } while((ch == ' ') || (ch == '\t') || (ch == '\n') || (ch == '\r'));

    if((ch < '0') || (ch > '9'))
    {
        if(ch == '#')
        {
            while(ch == '#')
            {
                while(ch != '\n')
                    ch = getc(myPtr);
                ch = getc(myPtr);
            }
        }
        else
        {
            printf("ASCII fields contain garbage\n");
            exit(1);
        }
    }

    int ind = 0;

    do
    {
        ind = ind * 10 + (ch - '0');
        ch = getc(myPtr);
    } while((ch >= '0') && (ch <= '9'));

    return ind;
}

```

```
/*
* A method which writes an image to a file with an input file name. Called by the main method each time
* a new processed image has been created and needs to be added to a file.
*/
void writeImage(image_ptr imgPtr, char *fName, int size, int keyNum)
{
    float scale;
    switch(keyNum)
    {
        case 4:
            scale = 0.125;
            break;
        case 5:
            scale = 1.0;
            break;
        case 6:
            scale = 3.0;
            break;
        default :
            printf("Output dile is not a RAWBITS file\n");
            exit(1);
            break;
    }

    FILE *myPtr;

    if((myPtr = fopen(fName, "wb")) == NULL)
    {
        printf("Unable to open %s file to ouput\n", fName);
        exit(1);
    }

    fprintf(myPtr, "P%d\n%d %d\n", keyNum, size, size);
    if(keyNum != 4)
        fprintf(myPtr, "255\n");

    int rowSize = size * scale;
    long totalSize = (long) rowSize * size;
    long offset = 0;
    long totalBytes = 0;

    for(int i = 0; i < size; i++)
    {
        totalBytes += fwrite(imgPtr + offset, 1, rowSize, myPtr);
        offset += rowSize;
    }

```

```
389     if(totalBytes != totalSize)
390         printf("Tried to write %ld bytes but instead wrote %ld\n", totalSize, totalBytes);
391 
392     fclose(myPtr);
393 }
394 
395 /*
396 * Used in the main method to compute the value at each point in the image matrix. This uses Sobel
397 * Edge Detection, with a horizontal mask which is shown below as maskH
398 */
399 int maskComputeH(int arr[][3])
400 {
401     int maskH[3][3] = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}};
402     int sum = 0;
403 
404     for(int i = 0; i < 3; i++)
405     {
406         for(int j = 0; j < 3; j++)
407         {
408             sum += (maskH[i][j] * arr[i][j]);
409         }
410     }
411 
412     return abs(sum);
413 }
414 
415 /*
416 * Used in the main method to compute the value at each point in the image matrix. This uses Sobel
417 * Edge Detection, with a vertical mask which is shown below as maskV
418 */
419 int maskComputeV(int arr[][3])
420 {
421     int maskV[3][3] = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}};
422     int sum = 0;
423 
424     for(int i = 0; i < 3; i++)
425     {
426         for(int j = 0; j < 3; j++)
427         {
428             sum += (maskV[i][j] * arr[i][j]);
429         }
430     }
431 
432     return abs(sum);
433 }
434 //End of program
```

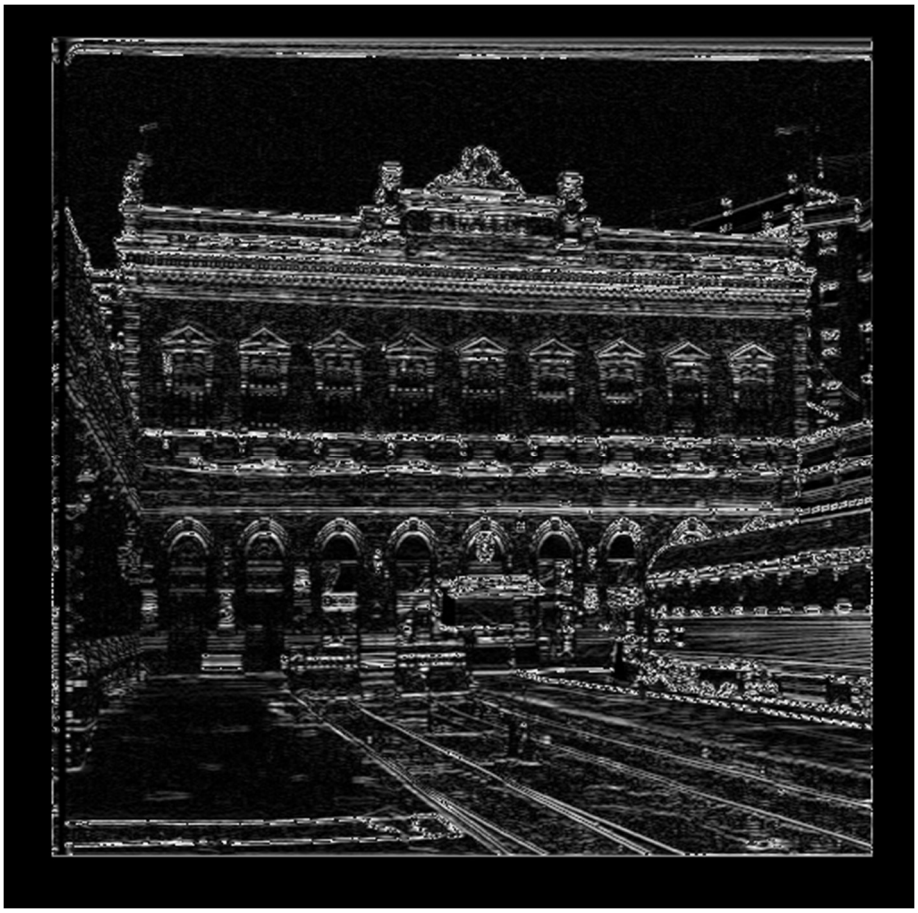
Testing

Test 1: (Incorrect inputs)

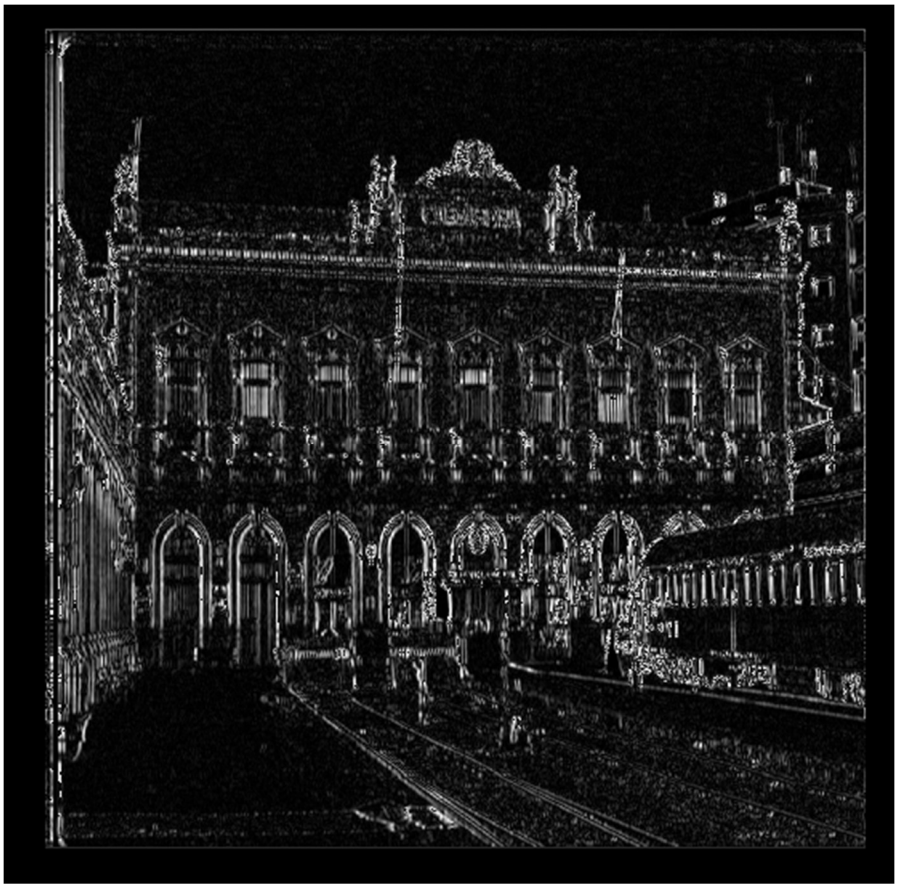
```
C:\Users\kk_la\Desktop\350 Assignment2\Code>a trainsta512x512.pgm
Incorrect input format: use a imageIn.ppm outEh.ppm outEv.ppm outE.ppm outBh.ppm outBv.ppm outB.ppm
```

Test 2: (Correct inputs)

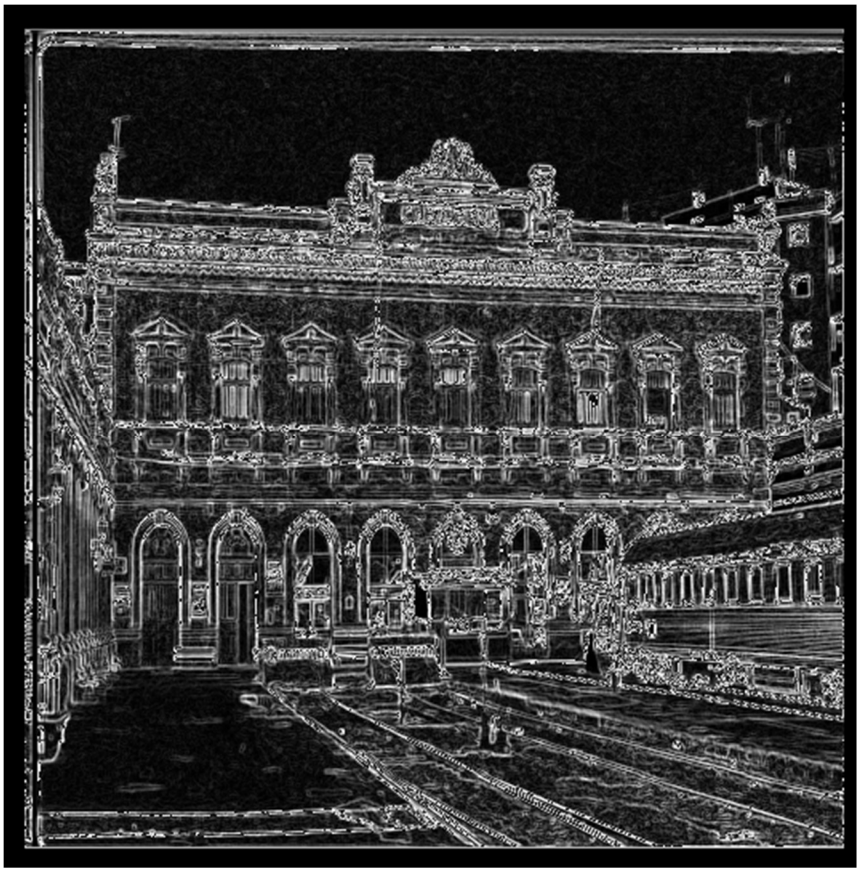
```
C:\Users\kk_la\Desktop\350 Assignment2\Code>a trainsta512x512.pgm picA.pgm picB.pgm picC.pgm picD.pgm picE.pgm picF.pgm

Input Average: 120, Input Standard Deviation: 67
Edge Response H Average: 41, Edge Response H Standard Deviation: 51
Edge Response V Average: 31, Edge Response V Standard Deviation: 44
Edge Response Combined Average: 65, Edge Response Combined Standard Deviation: 60
```



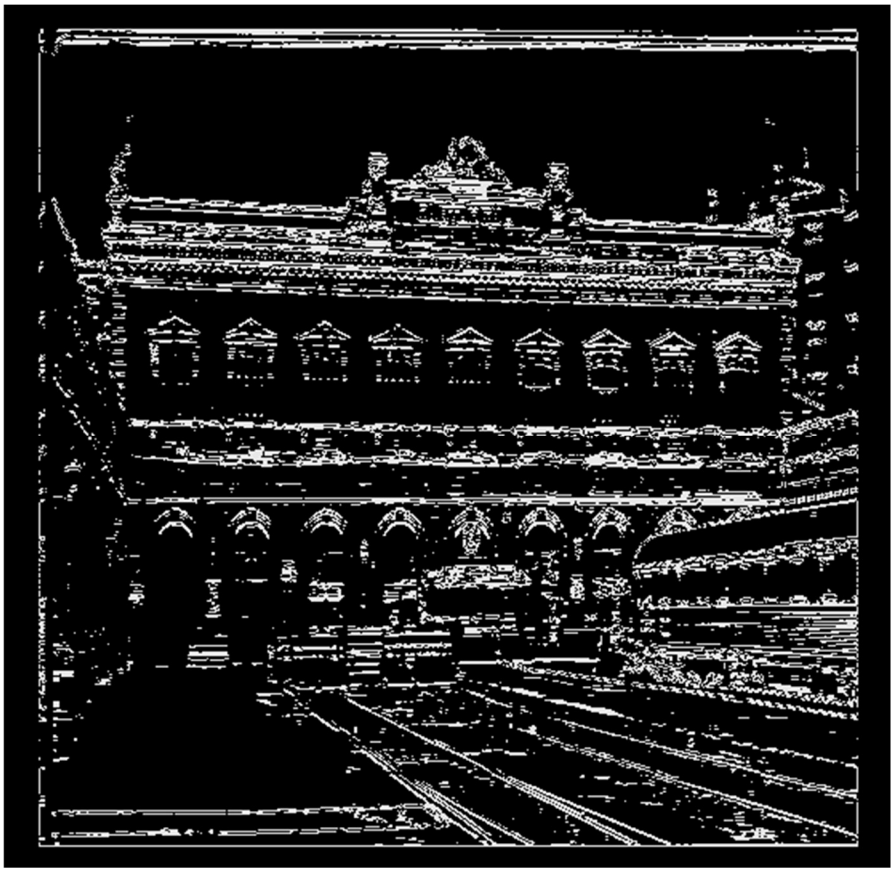
picA.pgm (Horizontal Edge Detection)



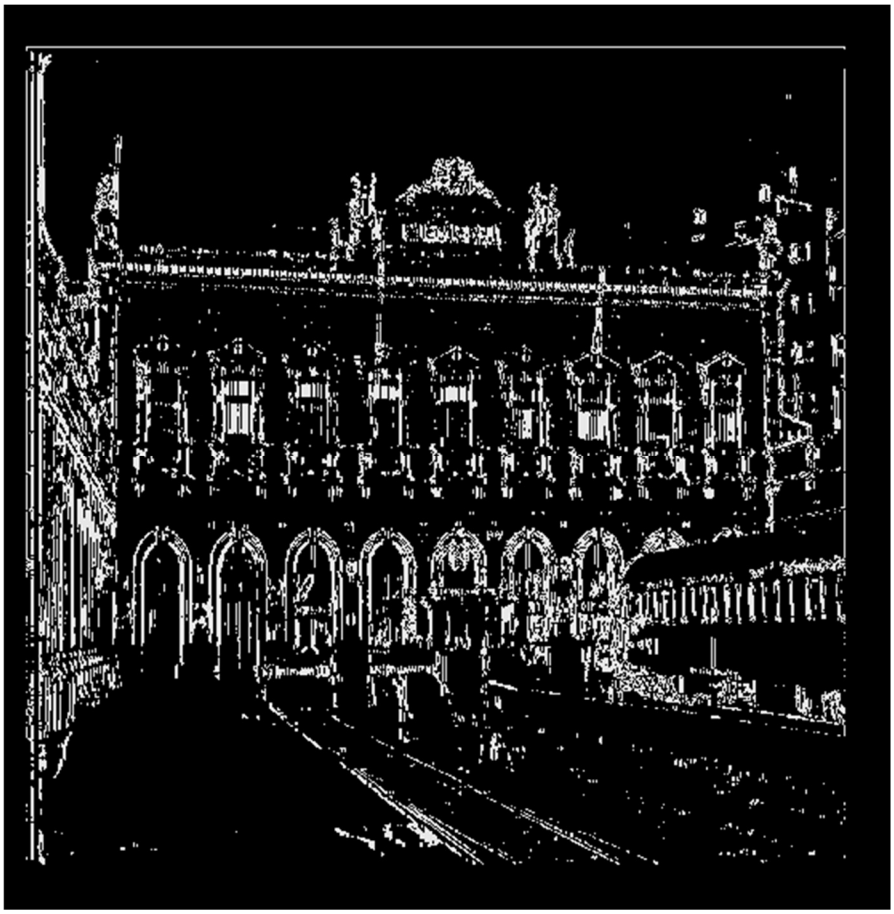
picB.pgm (Vertical Edge Detection)



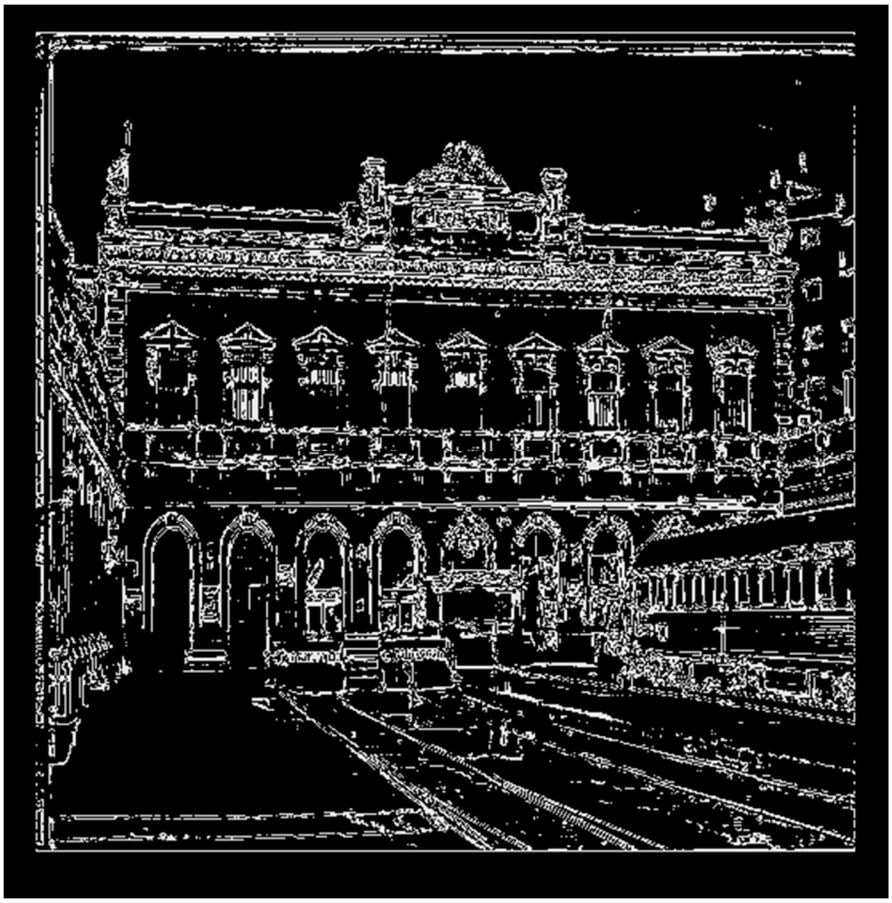
picC.pgm (Combined Edge Detection)



picD.pgm (Horizontal Binary)



picE.pgm (Vertical Binary)



picF.pgm (Combined Binary)

The images all appear to have been generated as expected according to the handout for Assignment 2, so therefore it can be concluded that the code was written correctly