Alg. 1:

**Overview**: This function writes the differences between two files to an output file.

**Input**: File f1, file f2, file output

**Output**: None

**Conditions**: The difference should only account for the words and order of the words from f2 to f1. It should not consider punctuation or spacing.

**Algorithm**:

1. A and B = strings of characters to hold words in f1 and f2, respectively
2. Declare c1 and c2
3. Declare i and j
4. While the end of neither f1 nor f2 have been reached:
   1. c1 = character from f1
   2. c2 = character from f2
   3. i and j = 0
   4. While c1 is not whitespace or punctuation or the end of f1 hasn’t been reached:
      1. A[i] = c1
      2. Increment i by 1
      3. c1 = next character in f1
   5. While c2 is not whitespace or punctuation or the end of f2 hasn’t been reached:
      1. B[j] = c2
      2. Increment j by 1
      3. c2 = next character in f2
   6. If A is not the same as B, then write B to output
5. (Next while loops are used only if f1 and f2 have a different # of words)
6. While the end of f1 hasn’t been reached:
   1. i = 0
   2. While c1 is not whitespace or punctuation or the end of f1 hasn’t been reached:
      1. A[i] = c1
      2. Increment i by 1
      3. Get the next character in the file and store it in c1
   3. If A is not empty, then write something to signify an absence of a word to output
   4. c1 = next character in f1
7. While the end of f2 hasn’t been reached:
   1. j = 0
   2. While c2 is not whitespace or punctuation or the end of f2 hasn’t been reached:
      1. B[j] = c2
      2. Increment j by 1
      3. c2 = next character in f2
   3. If B is not empty, then write B to output
   4. c2 = next character in f2

<http://practicalcryptography.com/ciphers/columnar-transposition-cipher/>

Alg. 2:

**Overview**: This function encrypts a file using the Columnar Transposition Cipher.

**Input**: A string of characters K, a file source

**Output**: None

**Conditions**: The characters in K and every word in source is in lowercase and only alphabetical characters

**Algorithm For Encryption**:

* + - 1. counter = 0
      2. Declare c
      3. While the end of source hasn’t been reached:
         1. c = character from source
         2. If c is not whitespace, then increment counter by 1
      4. Store the length of K in a variable called len
      5. rows = (counter + (len – 1)) / len
      6. A = list of strings of characters
      7. Declare i and j
      8. Iterating through rows (i) in A:
         1. j = 0
         2. While j is not equal to len and the end of source hasn’t been reached yet:

c = next character in source

If c is not whitespace, then:

Set A[i][j] equal to c

Increment j by 1

* + - 1. Decrement i by 1
      2. While j < len:
         1. A[i][j] = x
         2. Increment j by 1

(I stores the integers representing the indices of letters in each row (I stands for indices))

* + - 1. I = list of size len
      2. T = copy of K
      3. Sort T alphabetically
      4. Iterating through the characters (i) of K:
         1. Iterating through the characters (j) of T:

If K[i] is equal to T[j], then I[i] = j

(B will contain the encrypted words)

* + - 1. Create a list B of strings of characters with same size as A
      2. index = 0
      3. Iterate through rows (i) in A:
         1. Iterating through characters in strings (j) in A

index = I[j]

B[i][index] = A[i][j]

* + - 1. Create a new file encrypted

(Write column by column, not row by row)

* + - 1. Iterate over the characters in strings (j) in B
         1. Iterating over rows (i) of B:

Write B[i][j] to encrypted

<http://practicalcryptography.com/ciphers/columnar-transposition-cipher/>

Alg. 3:

**Overview:** This function decrypts a file encrypted by the Columnar Transposition Cipher.

**Input:** A string of characters K, a file source

**Output:** None

**Conditions:** Every word in source is in lowercase

**Algorithm For Decryption:**

counter = 0

Declare c

While the end of source hasn’t been reached:

c = character from source

If c is not whitespace, then increment counter by 1

len = length of K

rows = counter / len

A = list of strings of characters

Declare i and j

(Insert message into A column by column)

Iterate through A until len has been reached, incrementing j each time:

i = 0

While i is not equal to rows and the end of source hasn’t been reached:

c = character from source

If c is not whitespace, then:

A[i][j] = c

Increment i by 1

(I stores the integers representing the indices of letters in each row (I stands for indices))

I = list of size len

T = copy of K

Sort T alphabetically

Iterating through characters (i) in K:

Iterating through characters (j) of T:

If K[i] is equal to T[j], then I[i] = j

(B will contain the decrypted words)

Create a list B of characters of with same size as A

index = 0

Iterating through rows (i) in A:

Iterating through characters in strings (j) in A:

index = I[j]

B[i][index] = A[i][j]

Create a new file decrypted

Iterating over rows (i) of B:

Iterating over characters in strings (j) in B:

Write B[i][j] to decrypted

Alg. 4:

**Overview**: This function removes a file from the disk in the current directory and displays the appropriate error message if the removal fails.

**Input**: A string filename

**Output**: None

**Conditions**: Function must be verbose (?)

**Algorithm**:

1. Remove the file
2. If the file is removed successfully, then display a message describing it
3. Otherwise, display the correct error message if the removal failed

Alg. 5:

**Overview**: This function verifies .c file by ensuring that all pairs of brackets, quotes, parentheses, etc. are matched and nested correctly.

**Input**: A file source

**Output**: int – 1 if the file is formatted correctly, 0 if not

**Conditions**: Curly brackets are the only characters that need to have the same indenting for their opening and closing variants.

**Algorithm**:

(total stores the total number of {, (, [, “, and ‘)

1. total = 0
2. A = list of strings of characters
3. B = string of characters
4. Declare an integer c
5. i and j = 0
6. While the end of source hasn’t been reached:
   1. c = character from source
   2. j = 0
   3. While c is not a newline character and the end of source hasn’t been reached:
      1. If c is a tab character, then:
         1. B[j] = c
         2. Increment j by 1
      2. Otherwise, if c is {, then:
         1. A[i] = B
         2. A[i][j] = c
         3. Increment i by 1
         4. Increment total by 1
      3. Otherwise, if c is [ or ( or ‘ or “, then:
         1. A[i][0] = c
         2. Increment i by 1
         3. Increment total by 1
      4. Otherwise, if c is }, then:
         1. A[i] = B
         2. A[i][j] = c
         3. Increment i by 1
         4. Increment total by 1
      5. Otherwise, if c is ] or ), then:
         1. A[i][0] = c
         2. Increment i by 1
         3. Increment total by 1
      6. c = next character in source
7. sqOpenCount, sqCloseCount, parOpenCount, parCloseCount, apostCount, quoteCount = 0
8. Iterating through characters in strings (i) of A:
   1. If A[i][0] is [, then:
      1. Increment sqOpenCount by 1
   2. If A[i][0] is ], then:
      1. Increment sqCloseCount by 1
   3. Otherwise, if A[i][0] is (, then:
      1. Increment parOpenCount by 1
   4. Otherwise, if A[i][0] is ), then:
      1. Increment parCloseCount by 1
   5. Otherwise, if A[i][0] is ‘, then:
      1. Increment apostCount by 1
   6. Otherwise, if A[i][0] is “, then:
      1. Increment quoteCount by 1
9. If (sqOpenCount does not equal sqCloseCount) or (parOpenCount does not equal parCloseCount) or (apostCount mod 2 does not equal 0) or (quoteCount mod 2 does not equal 0)
   1. Return 0

(Now checking if curly brackets are correct and have consistent indents)

1. curlOpenCount, curlCloseCount = 0

(T stores number of tabs in a line)

1. T = list that is the same size as B
   1. Fill T with 0s
2. Iterating through rows (i) of A:
   1. Iterating through the characters in strings (j) of A
      1. If A[i][j] is a tab character, then increment T[i] by 1
      2. Otherwise, if A[i][j] is {, then:
         1. Increment curlOpenCount by 1
      3. Otherwise, if A[i][j] is }, then:
         1. Increment curlCloseCount by 1
3. If curlOpenCount does not equal curlCloseCount, then:
   1. Return 0
4. Sort T in ascending order

(sameCount counts the number of lines that have the same number of tabs)

1. sameCount = 1

(i starts at 1 because we only care about comparing the current tab count and the previous one)

1. Iterating through the elements (i starting at 1) of T:
   1. If T[i] is equal to T[i-1], Then increment sameCount by 1
   2. Otherwise, if T[i] is not equal to T[i-1], then:
      1. If sameCount mod 2 does not equal 0, then return 0
      2. Otherwise, sameCount = 1
2. Return 1

Alg. 6:

**Overview**: Write a function that translates a file from English to Pig Latin

**Input**: A file source

**Output**: None

**Condition**: All letters are lowercase, and there is no punctuation in the middle of words (ex. don’t).

**Algorithm**:

1. Declare strings of characters word, newWord, twoLetters, and threeLetters
2. Declare c
3. Declare i
4. Create a new file called pigged
5. While the end of source hasn’t been reached:
   1. c = character from source
   2. i = 0
   3. While c is not whitespace and c is not punctuation and the end of source hasn’t been reached:
      1. word[i] = c
      2. Increment i by 1
      3. c = next character in source
   4. If the length of word > 1, then:
      1. If the length of word > 2, then:
         1. threeLetters[0] = word[0]
         2. threeLetters[1] = word[1]
         3. threeLetters[2] = word[2]
      2. twoLetters[0] = word[0]
      3. twoLetters[1] = word[1]

(Rule 1)

* 1. If (twoLetters is the same as “xr” or twoLetters is the same as “yt”) or (word[0] is a vowel), then:
     1. Concatenate “ay” to end of word
     2. newWord = word

(Rule 2/4 –three consonants at start of word)

* 1. Otherwise, if all characters in threeLetters are not vowels or y, then:
     1. Iterating through characters (i starting at 3) in word:
        1. newWord[i-3] = word[i]
     2. Concatenate threeLetters to newWord
     3. Concatenate “ay” to newWord

(Rule 2/4 – two consonants at start of word)

* 1. Otherwise, if all characters in twoLetters are not vowels or y or twoLetters[1] is not q, then:
     1. Iterating through characters (i starting at 2) in word:
        1. newWord[i-2] = word[i]
     2. Concatenate twoLetters to newWord
     3. Concatenate “ay” to newWord

(Rule 2/3/4)

* 1. Otherwise, if word[0] is not a vowel, then:
     1. If (threeLetters[1] is ‘q’) and (threeLetters[2] is ‘u’), then:
        1. Iterating through characters (i starting at 3) in word:
           1. newWord[i-3] = word[i]
        2. Concatenate threeLetters to newWord
        3. Concatenate “ay” to newWord
     2. Otherwise,
        1. Iterating through characters (i starting at 1) in word:
           1. newWord[i-1] = word[i]
        2. Concatenate word[0] to newWord
        3. Concatenate “ay” to newWord
  2. If the length of newWord is not 0, then write it to pigged
  3. Write c to pigged