# [CCPROG2] Machine Project

Test Script

Submitted by:
Daniel III Lapitan Ramos
Gwen Kathleen Roco

Submitted to:
Sir Adrian Perez

# Contents

**Contents** **2**

**Unit Tests** **3**

tokenizer.c functions 3

hash_table.c functions 6

utils.c functions 7

cleaner.c functions 9

**Integration/System-Level Tests** **13**

# Unit Tests

## tokenizer.c functions

`void add_token (TokenList *tl, char *token)`

This function adds a token to the linked list.
Pre-condition: the linked list is null-terminated.

| # | Test Description | Sample Input | Expected | Actual | P/F |
|---|---|---|---|---|---|
| 1 | Adding to an empty list. | `tl→tokens = {NULL}`<br>`token = “Hello”` | `tl→tokens = {”Hello”, NULL}` | `tl→tokens = {”Hello”, NULL}` | P |
| 2 | Adding to a list with one element. | `tl→tokens = {”Hello”, NULL}`<br>`token = “world”` | `tl→tokens = {”Hello”, “world”, NULL}` | `tl→tokens = {”Hello”, “world”, NULL}` | P |
| 3 | Adding to a list with more than one element | `tl→tokens = {”Hello” “world”, “CCPROG”, “Structured”, “data”, NULL}`<br>`token = ”types”` | `tl→tokens = {”Hello” “world”, “CCPROG”, “Structured”, “data”, “types”, NULL}` | `tl→tokens = {”Hello” “world”, “CCPROG”, “Structured”, “data”, “types”, NULL}` | P |
| 4 | Adding non-alphanumeric characters | `tl→tokens = {NULL}`<br>`token = “!”` | `tl→tokens = {”!”, NULL}` | `tl→tokens = {”!”, NULL}` | P |
| 5 | Adding numeric characters | `tl→tokens = {”CCPROG”, NULL}`<br>`token = “2”` | `tl→tokens = {”CCPROG”, “2”, NULL}` | `tl→tokens = {”CCPROG”, “2”, NULL}` | P |
| 6 | Adding escape sequences | `tl→tokens = {”Hello”, “world”}`<br>`token = “\n”` | `tl→tokens = {”Hello”, “world”}`<br>`token = “\n”` | `tl→tokens = {”Hello”, “world”}`<br>`token = “\n”` | P |

`void remove_token (TokenList *tl, char *token)`

This function removes the specified token from the linked list.
Pre-condition: the linked list is null-terminated.

| # | Test Description | Sample Input | Expected | Actual | P/F |
|---|---|---|---|---|---|
| 1 | Removing the head node. | `tl→tokens = {"Hello", "world", "!", NULL} token = "Hello"` | `tl→tokens = {"world", "!", NULL}` | `tl→tokens = {"world", "!", NULL}` | P |
| 2 | Removing the tail node. | `tl→tokens = {"Hello", "world", "!", NULL} token = "!"` | `tl→tokens = {"Hello", "world", NULL}` | `tl→tokens = {"Hello", "world", NULL}` | P |
| 3 | Removing a node within the list | `tl→tokens = {"Hello", "world", "!", NULL} token = "world"` | `tl→tokens = {"Hello", "!", NULL}` | `tl→tokens = {"Hello", "!", NULL}` | P |
| 4 | Removing from an empty list | `tl→tokens = {NULL} token = "world"` | *invalid* (input remains intact as function early returns) | *invalid* (input remains intact as function early returns) | P |
| 5 | Removing a token not found in the list | `tl→tokens = {"Hello", "world", "!", NULL} token = "CCPROG"` | *invalid* (input remains intact as function early returns) | *invalid* (input remains intact as function early returns) | P |

`bool is_token_found(TokenList *tokenList, char *token)`

Check if the token is in the linked list
Pre-condition: the linked list is null-terminated

| # | Test Description | Sample Input | Expected | Actual | P/F |
|---|---|---|---|---|---|
| 1 | The token is not in the list | `tl→tokens = {"Hello", " ", "world", "!", "\n", NULL} token = "CCPROG"` | `return false` | `return false` | P |
| 2 | The token is the first element. | `tl→tokens = {"Hello", " ", "world", "!", "\n", NULL} token = "Hello"` | `return false` | `return false` | P |
| 3 | The token is the last element. | `tl→tokens = {"Hello", " ", "world", "!", "\n", NULL} token = "\n"` | `return false` | `return false` | P |

| | | | | | |
|---|---|---|---|---|---|
| 4 | The token is within the list. | `tl→tokens = {"Hello", " ", "world", "!", "\n", NULL}`<br>`token = "\n"` | *return false* | *return false* | P |

`void increment_token_frequency(TokenList *tokenList, char *token)`

This function increases the frequency of a token.
Pre-condition: the token must exist in the list

| # | Test Description | Sample Input | Expected | Actual | P/F |
|---|---|---|---|---|---|
| 1 | Incrementing from 0; incrementing first element | `*tokenList = { {.token = "hello", .frequency = 0} } token = "hello"` | `"hello".frequency = 1` | `"hello".frequency = 1` | P |
| 2 | Incrementing from 1 | `*tokenList = { {.token = "hello", .frequency = 1} } token = "hello"` | `"hello".frequency = 2` | `"hello".frequency = 2` | P |
| 3 | Incrementing from more than one | `*tokenList = { {.token = "hello", .frequency = 7} } token = "hello"` | `"hello".frequency = 8` | `"hello".frequency = 8` | P |
| 4 | Incrementing middle element | `*tokenList = { {.token = "hello", .frequency = 0}, {.token = "world", .frequency = 2}, {.token = "!", .frequency = 11} } token = "world"` | `"world".frequency = 3` | `"world".frequency = 3` | P |
| 5 | Incrementing last element | `*tokenList = { {.token = "hello", .frequency = 0}, {.token = "world", .frequency = 2}, {.token = "!", .frequency = 11} } token = "!"` | *`"!".frequency = 12`* | *`"!".frequency = 12`* | p |

## hash_table.c functions

`void add_element(HashTable *hashTable, char *item)`

This function adds elements based on the hash function.
Pre-condition: item is unique (not currently present in the hash table)

| # | Test Description | Sample Input | Expected | Actual | P/F |
|---|---|---|---|---|---|
| 1 | No collision. | `hashTable = {NULL, NULL, NULL, ...} item = “hello”` | `hashTable = {NULL, {“hello”, NULL}, NULL, ...}` | `hashTable = {NULL, {“hello”, NULL}, NULL, ...}` | P |
| 2 | First collision. | `hashTable = {NULL, {“hello”, NULL}, NULL, ...} item = “ “` | `hashTable = {NULL, {“hello”, “ ”, NULL}, NULL, ...}` | `hashTable = {NULL, {“hello”, “ ”, NULL}, NULL, ...}` | P |
| 3 | More than one collision. | `hashTable = {NULL, {“hello”, “ ”, “world”, “!”, “!”, NULL}, NULL, ...} item = “!”` | `hashTable = {NULL, {“hello”, “ ”, “world”, “!”, “!”, “!”, NULL}, NULL, ...}` | `hashTable = {NULL, {“hello”, “ ”, “world”, “!”, “!”, “!”, NULL}, NULL, ...}` | P |

The _get_hash_function() is stubbed with the value of 1.

`bool contains(Hash *hashTable, char *item)`

This function returns 1 if the specified item is in the hash table; 0 if otherwise.

| # | Test Description | Sample Input | Expected | Actual | P/F |
|---|---|---|---|---|---|
| 1 | No collision. | `hashTable = {..., {“hello”, NULL}, ...} item = “hello”` | `true` | `true` | P |
| 2 | First collision. | `hashTable = {..., {“xy”, “yX”, “z7”, NULL}, ...} item = “z7”` | `true` | `true` | P |
| 3 | More than one collision. | `hashTable = {..., {“hello”, NULL}, ...} item = “world”` | `true` | `true` | P |

## utils.c functions

`TokenList tokenizer(char *input)`

This function separates a string to a list of tokens; that is, either a sequence of alphabetic characters or a single non-alphabetic character.
Pre-condition: string is null-appended

| # | Test Description | Sample Input | Expected | Actual | P/F |
|---|---|---|---|---|---|
| 1 | A single alphabetic token | `input = “Hello”` | `return TokenList = {”Hello”}` | `return TokenList = {”Hello”}` | P |
| 2 | A single non-alphabetic token | `input = “,”` | `return TokenList = {”,”}` | `return TokenList = {”,”}` | P |
| 3 | Mixed alphabetic and numeric tokens | `input = “Hel1o”` | `return TokenList = {”Hel”, “1”, “o”}` | `return TokenList = {”Hel”, “1”, “o”}` | P |
| 4 | Mixed alphabetic and whitespace tokens | `input = “Hello world”` | `return TokenList = {”Hello”, “world”}` | `return TokenList = {”Hello”, “world”}` | P |
| 5 | Mixed alphabetic and special character tokens | `input = “Hello!”` | `return TokenList = {”Hello”, “!”}` | `return TokenList = {”Hello”, “!”}` | P |
| 6 | String with all types of tokens | `input = “Hel1o, world!\t”` | `return TokenList = {”Hel”, “1”, “o”, “,” “ “, “world”, “!”, “\t”}` | `return TokenList = {”Hel”, “1”, “o”, “,” “ “, “world”, “!”, “\t”}` | P |

`void swap(TokenNode *a, TokenNode *b)`

This function swaps two nodes in a linked list
Pre-condition: Non-NULL nodes have a valid string value.

| # | Test Description | Sample Input | Expected | Actual | P/F |
|---|---|---|---|---|---|
| 1 | No node is NULL. | a = {.token = “hello”, .frequency = 2}<br>b = {.token = “world”, .frequency = 10} | a = {.token = “world”, .frequency = 10}<br>b = {.token = “hello”, .frequency = 2} | a = {.token = “world”, .frequency = 10}<br>b = {.token = “hello”, .frequency = 2} | P |
| 2 | Node A is NULL. | a = NULL<br>b = {.token = “hello”, .frequency = 2} | a = NULL<br>b = {.token = “hello”, .frequency = 2} | a = NULL<br>b = {.token = “hello”, .frequency = 2} | P |
| 3 | Node B is NULL. | a = {.token = “hello”, .frequency = 2}<br>b = NULL | a = {.token = “hello”, .frequency = 2}<br>b = NULL | a = {.token = “hello”, .frequency = 2}<br>b = NULL | P |

`void sort_tokens(TokenList *tl)`

This function sorts the tokens in the token list based on word frequency in descending order.

| # | Test Description | Sample Input | Expected | Actual | P/F |
|---|---|---|---|---|---|
| 1 | The list is in ascending order. | tl→frequency = {1, 2, 3, 4, 5} | tl→frequency = {5, 4, 3, 2, 1} | tl→frequency = {5, 4, 3, 2, 1} | P |
| 2 | The list is in descending order. | tl→frequency = {5, 4, 3, 2, 1} | tl→frequency = {5, 4, 3, 2, 1} | tl→frequency = {5, 4, 3, 2, 1} | P |
| 3 | The list is in random order | tl→frequency = {3, 1, 4, 2, 5} | tl→frequency = {5, 4, 3, 2, 1} | tl→frequency = {5, 4, 3, 2, 1} | P |

## cleaner.c functions

`void clean_whitespace(Summary *summary)`

This function removes unnecessary whitespaces.
Pre-condition: All whitespaces are simple spaces, not newlines

| # | Test Description | Sample Input | Expected | Actual | P/F |
|---|---|---|---|---|---|
| 1 | All whitespace tokens do not have neighboring whitespace tokens | summary->tokenList = {“hello”, “ “, “world”} | summary->tokenList = {“hello”, “ “, “world”} | summary->tokenList = {“hello”, “ “, “world”} | P |
| 2 | A whitespace token has a neighboring whitespace token | summary->tokenList = {“hello”, “ “, “ “, “world”} | summary->tokenList = {“hello”, “ “, “world”} | summary->tokenList = {“hello”, “ “, “world”} | P |
| 3 | Trailing whitespace | summary->tokenList = {“hello”, “ “, “world”, “ “} | summary->tokenList = {“hello”, “ “, “world”} | summary->tokenList = {“hello”, “ “, “world”} | P |
| 4 | Leading whitespace | summary->tokenList = {“ “, “hello”, “ “, “world”} | summary->tokenList = {“hello”, “ “, “world”} | summary->tokenList = {“hello”, “ “, “world”} | P |

`void to_lowercase(Summary *summary)`

This function converts the string of each token to its lowercase counterpart. clean_whitespace is called.

| # | Test Description | Sample Input | Expected | Actual | P/F |
|---|---|---|---|---|---|
| 1 | All tokens are uppercase | summary->tokenList = {“HELLO”, “ “, “WORLD”} | summary->tokenList = {“hello”, “ “, “world”} | summary->tokenList = {“hello”, “ “, “world”} | P |
| 2 | All tokens are lowercase | summary->tokenList = {“hello”, “ “, “world”} | summary->tokenList = {“hello”, “ “, “world”} | summary->tokenList = {“hello”, “ “, “world”} | P |
| 3 | The tokens are mixed-case | summary->tokenList = {“Hello”, “ | summary->tokenList = {“hello”, | summary->tokenList = {“hello”, | P |

| | | | | | |
|---|---|---|---|---|---|
| | | “, “wOrlD”} | “ “, “world”} | “ “, “world”} | |
| 4 | The tokens have special characters | summary->tokenList = {“Hello”, “,”, “ “, “wOrlD”} | summary->tokenList = {“hello”, “,”, “ “, “world”} | summary->tokenList = {“hello”, “,”, “ “, “world”} | P |
| 5 | The tokens have numeric characters | summary->tokenList = {“Hello”, “ “, “w”, “0”, “r”, “lD”} | summary->tokenList = {“hello”, “ “, “w”, “0”, “r”, “ld”} | summary->tokenList = {“hello”, “ “, “w”, “0”, “r”, “ld”} | P |
| 6 | The tokens have all token types. | summary->tokenList = {“Hello”, “,”, “ “, “w”, “0”, “r”, “lD”} | summary->tokenList = {“hello”, “,”, “ “, “w”, “0”, “r”, “ld”} | summary->tokenList = {“hello”, “,”, “ “, “w”, “0”, “r”, “ld”} | P |

`void remove_special_char(Summary *summary)`

This function removes all tokens tagged as special characters. If the special token is preceding an alphanum token, it is replaced with a single space. clean_whitespace is called.

| # | Test Description | Sample Input | Expected | Actual | P/F |
|---|---|---|---|---|---|
| 1 | Token list has no special tokens | summary->tokenList = {“HELLO”, “ “, “WORLD”} | summary->tokenList = {“HELLO”, “ “, “WORLD”} | summary->tokenList = {“HELLO”, “ “, “WORLD”} | P |
| 2 | Token list has one special token | summary->tokenList = {“HELLO”, “,”, “ “, “WORLD”} | summary->tokenList = {“HELLO”, “ “, “WORLD”} | summary->tokenList = {“HELLO”, “ “, “WORLD”} | P |
| 3 | Token list has more than one special tokens | summary->tokenList = {“HELLO”, “,”, “ “, “WORLD”, "!"} | summary->tokenList = {“HELLO”, “ “, “WORLD”} | summary->tokenList = {“HELLO”, “ “, “WORLD”} | P |
| 4 | Special token is preceding an alpha token | summary->tokenList = {“HELLO”, “ “, “WO", "-", "RLD”} | summary->tokenList = {“HELLO”, “ “, “WO", " ", "RLD”} | summary->tokenList = {“HELLO”, “ “, “WO", " ", "RLD”} | P |
| 5 | Special token is preceding a numeric tokens | summary->tokenList = { “HELLO”, “ “, "-", "2”} | summary->tokenList = { “HELLO”, “ “, " ", "2”} | summary->tokenList = { “HELLO”, “ “, " ", "2”} | P |

`void remove_numbers(Summary *summary)`

This function removes all tokens tagged as numeric. clean_whitespace is called.

| # | Test Description | Sample Input | Expected | Actual | P/F |
|---|---|---|---|---|---|
| 1 | Token list has no numeric tokens | `summary->tokenList = {“HELLO”, “ “, “WORLD”}` | `summary->tokenList = {“HELLO”, “ “, “WORLD”}` | `summary->tokenList = {“HELLO”, “ “, “WORLD”}` | P |
| 2 | Token list has one numeric token | `summary->tokenList = {“HELLO”, “,”, “ “, “9”, “WORLD”}` | `summary->tokenList = {“HELLO”, “ “, “WORLD”}` | `summary->tokenList = {“HELLO”, “ “, “WORLD”}` | P |
| 3 | Token list has more than one numeric token | `summary->tokenList = {“HELLO”, “8”, “ “, “WORLD”, "9"}` | `summary->tokenList = {“HELLO”, “ “, “WORLD”}` | `summary->tokenList = {“HELLO”, “ “, “WORLD”}` | P |
| 4 | Numeric token is in between two alphabetical tokens | `summary->tokenList = {“HELLO”, “ “, “W”, “0”, RLD”}` | `summary->tokenList = {“HELLO”, “ “, “W”, “RLD”}` | `summary->tokenList = {“HELLO”, “ “, “W”, “RLD”}` | P |

`void remove_stopwords(Summary *summary)`

This function removes all matching stopwords. clean_whitespace is called.

| # | Test Description | Sample Input | Expected | Actual | P/F |
|---|---|---|---|---|---|
| 1 | Token list has no stopwords | `summary->tokenList = {“HELLO”, “ “, “WORLD”}` | `summary->tokenList = {“HELLO”, “ “, “WORLD”}` | `summary->tokenList = {“HELLO”, “ “, “WORLD”}` | P |
| 2 | Token list has one stopword | `summary->tokenList = {“HELLO”, “ ”, “the”, “ “, “WORLD”}` | `summary->tokenList = {“HELLO”, “ “, “WORLD”}` | `summary->tokenList = {“HELLO”, “ “, “WORLD”}` | P |
| 3 | Token list has more than one stopword | `summary->tokenList = {“HELLO”, “ “, “WORLD”, " ", “I”, “ “, “AM”}` | `summary->tokenList = {“HELLO”, “ “, “WORLD”}` | `summary->tokenList = {“HELLO”, “ “, “WORLD”}` | P |

`void clean_all(Summary *summary)`

This function removes all matching stopwords. clean_whitespace is called.

| # | Test Description | Sample Input | Expected | Actual | P/F |
|---|---|---|---|---|---|
| 1 | Token list does not need any changes | `summary->tokenList = {“hello”, “ “, “world”}` | `summary->tokenList = {“hello”, “ “, “world”}` | `summary->tokenList = {“hello”, “ “, “world”}` | P |
| 2 | Token list needs some changes | `summary->tokenList = {“HELLO”, “ ”, “the”, “ “, “9” “WORLD”}` | `summary->tokenList = {“hello”, “ “, “world”}` | `summary->tokenList = {“hello”, “ “, “world”}` | P |
| 3 | Token list needs all changes to be made | `summary->tokenList = {“1”, “HELLO”, “ “, “WORLD”, "!", “ “, “I”, “ “, “AM”, “ “}` | `summary->tokenList = {“hello”, “ “, “world”}` | `summary->tokenList = {“hello”, “ “, “world”}` | P |

# Integration/System-Level Tests

Some functions are too integrated to be tested independently (such as functions in engine.c). We can still test these functions by modifying the method of testing. The following tables show how we tested these functions.[1]

| Test Scenario: User navigates the main menu | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **#** | **Test Description** | **Steps** | **Sample Data** | **Pre-Conditions** | **Post-Conditions** | **Expected** | **Actual** | **P/F** |
| 1 | Valid choice input | 1. Launch application.<br>2. Enter “1” | Input: “1” | Config is initialized. | Program will proceed correctly according to choice input | Program proceeds to asking for a file name | Program proceeds to asking for a file name | P |
| 2 | Invalid choice below minimum input | 1. Launch application.<br>2. Enter “0” | Input: “0” | Config is initialized. | Program prompts user to enter a valid input | Error screen | Error screen | P |
| 3 | Invalid choice above maximum input | 1. Launch application.<br>2. Enter “9” | Input: “9” | Config is initialized. | Program prompts user to enter a valid input | Error screen | Error screen | P |

[1] We also tested for any memory leaks that may occur for each path via Valgrind.

| Test Scenario: User enters input file for processing | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **#** | **Test Description** | **Steps** | **Sample Data** | **Pre-Conditions** | **Post-Conditions** | **Expected** | **Actual** | **P/F** |
| 1 | Input file exists in raw_files folder.<br><br>File is big.<br><br>Mode is “clean”. | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “a.txt”. | Filename: “a.txt” | Config is initialized.<br><br>File “a.txt” exists in raw_files folder.<br><br>“a.txt” is a big file | Input file will be cleaned accordingly | Loading Screen | Loading Screen | P |
| 2 | Input file exists in raw_files folder.<br><br>File is small.<br><br>Mode is “clean”. | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “1.txt”. | Filename: “1.txt” | Config is initialized.<br><br>File “1.txt” exists in raw_files folder.<br><br>“1.txt” is a small file | Input file will be cleaned accordingly | Screen shows metadata and cleaner functions | Screen shows metadata and cleaner functions | P |
| 3 | Input file exists in cleaned_files folder.<br><br>File is big.<br><br>Mode is “analyze”. | 1. Launch application.<br>2. Choose “analyze document”.<br>3. Enter “a.txt”. | Filename: “a.txt” | Config is initialized.<br><br>File “a.txt” exists in cleaned_files folder.<br><br>“a.txt” is a big file. | Input file will be analyzed accordingly | Loading Screen | Loading Screen | P |
| 4 | Input file exists in cleaned_files folder.<br><br>File is small.<br><br>Mode is “analyze”. | 4. Launch application.<br>5. Choose “analyze document”.<br>6. Enter “1.txt”. | Filename: “1.txt” | Config is initialized.<br><br>File “1.txt” exists in cleaned_files folder.<br><br>“1.txt” is a small file. | Input file will be analyzed accordingly | Screen shows metadata and analyzing functions | Screen shows metadata and analyzing functions | P |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | Input file does not exist in raw_files folder<br><br>Mode is “clean”. | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “z.txt”. | Filename: “z.txt” | Config is initialized.<br><br>File “z.txt” does not exist in raw_files folder. | Program prompts user to enter a valid input file | Error screen | Error screen | P |
| 6 | Input file does not exist in cleaned_files folder<br><br>Mode is “analyze” | 1. Launch application.<br>2. Choose “analyze document”.<br>3. Enter “z.txt”. | Filename: “z.txt” | Config is initialized.<br><br>File “z.txt” does not exist in cleaned_files folder. | Program prompts user to enter a valid input file | Error screen | Error screen | P |

| Test Scenario: User is choosing a cleaning function | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **#** | **Test Description** | **Steps** | **Sample Data** | **Pre-Conditions** | **Post-Conditions** | **Expected** | **Actual** | **P/F** |
| 1 | Valid choice input of “To Lowercase” | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “a.txt”.<br>4. Choose “to lowercase” | Filename: “a.txt” | Config is initialized.<br><br>File “a.txt” is a valid input file.<br><br>Mode is “clean”. | Program will continue to clean unless chosen otherwise | Program asks user if they want to continue cleaning | Program asks user if they want to continue cleaning | P |
| 2 | Valid choice input of “Remove Special Character” | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “a.txt”.<br>4. Choose “remove special character” | Filename: “a.txt” | Config is initialized.<br><br>File “a.txt” is a valid input file.<br><br>Mode is “clean”. | Program will continue to clean unless chosen otherwise | Program asks user if they want to continue cleaning | Program asks user if they want to continue cleaning | P |
| 3 | Valid choice input of “Remove Numbers” | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “a.txt”.<br>4. Choose “remove numbers” | Filename: “a.txt” | Config is initialized.<br><br>File “a.txt” is a valid input file.<br><br>Mode is “clean”. | Program will continue to clean unless chosen otherwise | Program asks user if they want to continue cleaning | Program asks user if they want to continue cleaning | P |
| 4 | Valid choice input of “Clean Whitespaces” | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “a.txt”.<br>4. Choose “clean whitespaces” | Filename: “a.txt” | Config is initialized.<br><br>File “a.txt” is a valid input file.<br><br>Mode is “clean”. | Program will continue to clean unless chosen otherwise | Program asks user if they want to continue cleaning | Program asks user if they want to continue cleaning | P |
| 5 | Valid choice input of “Remove Stopwords” | 1. Launch application.<br>2. Choose “clean document”. | Filename: “a.txt” | Config is initialized.<br><br>File “a.txt” is a valid | Program will continue to clean unless chosen | Program asks user if they want to | Program asks user if they want to | P |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 3. Enter “a.txt”.<br>4. Choose “remove stopwords” | | input file.<br><br>Mode is “clean”. | otherwise | continue cleaning | continue cleaning | |
| 6 | Valid choice input of “All” | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “a.txt”.<br>4. Choose “all” | Filename: “a.txt” | Config is initialized.<br><br>File “a.txt” is a valid input file.<br><br>Mode is “clean”. | Program will save the cleaned file as the user’s chosen output filename in the cleaned_files folder | Program prompts users to enter an output filename | Program prompts users to enter an output filename | P |
| 7 | Invalid choice below minimum input | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “a.txt”.<br>4. Enter “0” | Filename: “a.txt” | Config is initialized.<br><br>File “a.txt” is a valid input file.<br><br>Mode is “clean”. | Program prompts user to enter a valid input | Error screen | Error screen | P |
| 8 | Invalid choice above maximum input | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “a.txt”.<br>4. Enter “9” | Filename: “a.txt” | Config is initialized.<br><br>File “a.txt” is a valid input file.<br><br>Mode is “clean”. | Program prompts user to enter a valid input | Error screen | Error screen | P |

| Test Scenario: User is in the “Continue Cleaning?” Screen | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **#** | **Test Description** | **Steps** | **Sample Data** | **Pre-Conditions** | **Post-Conditions** | **Expected** | **Actual** | **P/F** |
| 1 | Valid choice input of “Y” | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “1.txt”.<br>4. Choose “to lowercase”.<br>5. Enter “Y”. | Filename: “1.txt” | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “clean”. | Program will continue to clean unless chosen otherwise | Program asks user if they want to continue cleaning | Program asks user if they want to continue cleaning | P |
| 2 | Valid choice input of “N” | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “1.txt”.<br>4. Choose “to lowercase”.<br>5. Enter “N”. | Filename: “1.txt” | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “clean”. | Program will save the cleaned file as the input filename in the cleaned_files folder | Program prompts user to enter a filename for the output file | Program prompts user to enter a filename for the output file | P |
| 3 | Invalid choice input | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “1.txt”.<br>4. Choose “remove special character”<br>5. Enter “X”. | Filename: “1.txt” | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “clean”. | Program prompts user to enter a valid input | Error Screen | Error Screen | P |

| Test Scenario: User is analyzing a document | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **#** | **Test Description** | **Steps** | **Sample Data** | **Pre-Conditions** | **Post-Conditions** | **Expected** | **Actual** | **P/F** |
| 1 | Valid choice of “Word Count” | 1. Launch application.<br>2. Choose “analyze document”.<br>3. Enter “1.txt”<br>4. Choose “word count” | Filename: “1.txt” | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “analyze”. | Program will restart from main menu & output file will be saved in analysis folder as “1_wcount.txt” after entering | Screen displays first 10 words with the most word count | Screen displays first 10 words with the most word count | P |
| 2 | Valid choice of “N-gram Count” | 1. Launch application.<br>2. Choose “analyze document”.<br>3. Enter “1.txt”.<br>4. Choose “n-gram count” | Filename: “1.txt” | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “analyze”. | Program will analyze input file in terms of “n-gram count” | Program prompts user to enter the value of N | Program prompts user to enter the value of N | P |
| 3 | Valid choice of “Concordance” | 1. Launch application.<br>2. Choose “analyze document”.<br>3. Enter “1.txt”.<br>4. Choose “concordance” | Filename: “1.txt” | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “analyze”. | Program will analyze input file in terms of “concordance” | Program prompts user to enter a keyword | Program prompts user to enter a keyword | P |
| 4 | Invalid choice below minimum input | 1. Launch application.<br>2. Choose “analyze document”.<br>3. Enter “1.txt”.<br>4. Enter “0”. | Filename: “1.txt” | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “analyze”. | Program prompts user to enter a valid input | Error screen | Error screen | P |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | Invalid choice above maximum input | 1. Launch application.<br>2. Choose “analyze document”.<br>3. Enter “1.txt”.<br>4. Enter “6”. | Filename: “1.txt” | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “analyze”. | Program prompts user to enter a valid input | Error screen | Error screen | P |

| Test Scenario: User is analyzing a document in terms of “n-gram count” | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **#** | **Test Description** | **Steps** | **Sample Data** | **Pre-Conditions** | **Post-Conditions** | **Expected** | **Actual** | **P/F** |
| 1 | Valid input N where N > 0 and N < 5 | 1. Launch application.<br>2. Choose “analyze document”.<br>3. Enter “1.txt”.<br>4. Choose “n-gram count”<br>5. Enter “3”. | Filename: “1.txt”<br>N = 3 | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “analyze”. | Program will restart from main menu after entering | Screen shows top 10 analyzed “n-gram count” results | Screen shows top 10 analyzed “n-gram count” results | P |
| 2 | Invalid input where N is below the minimum input | 1. Launch application.<br>2. Choose “analyze document”.<br>3. Enter “1.txt”.<br>4. Choose “n-gram count”<br>5. Enter “0”. | Filename: “1.txt”<br>N = 0 | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “analyze”. | Program prompts user to enter a value from 1 to 4 | Error Screen | Error Screen | P |
| 3 | Invalid input where N is above the maximum input | 1. Launch application.<br>2. Choose “analyze document”.<br>3. Enter “1.txt”.<br>4. Choose “n-gram count”<br>5. Enter “5”. | Filename: “1.txt”<br>N = 5 | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “analyze”. | Program prompts user to enter a value from 1 to 4 | Error Screen | Error Screen | P |

| Test Scenario: User is analyzing a document in terms of “concordance” | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **#** | **Test Description** | **Steps** | **Sample Data** | **Pre-Conditions** | **Post-Conditions** | **Expected** | **Actual** | **P/F** |
| 1 | Valid input “N” where N > 0 and N < 5 | 1. Launch application.<br>2. Choose “analyze document”.<br>3. Enter “1.txt”.<br>4. Choose “concordance”.<br>5. Enter “2”. | Filename: “1.txt”<br>N = 2 | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “analyze”. | Program will analyze according to input N | Program prompts user to enter a keyword | Program prompts user to enter a keyword | P |
| 2 | Invalid input where “N” is below the minimum input | 1. Launch application.<br>2. Choose “analyze document”.<br>3. Enter “1.txt”.<br>4. Choose “concordance”.<br>5. Enter “0”. | Filename: “1.txt”<br>N = 0 | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “analyze”. | Program prompts user to enter a value from 1 to 4 | Error screen | Error screen | P |
| 3 | Invalid input where “N” is above the maximum input | 1. Launch application.<br>2. Choose “analyze document”.<br>3. Enter “1.txt”.<br>4. Choose “concordance”.<br>5. Enter “5”. | Filename: “1.txt”<br>N = 5 | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “analyze”. | Program prompts user to enter a value from 1 to 4 | Error screen | Error screen | P |
| 4 | Valid input “keyword” is found in input file | 1. Launch application.<br>2. Choose “analyze document”.<br>3. Enter “1.txt”.<br>4. Choose “concordance”.<br>5. Enter “2”<br>6. Enter “new” | Filename: “1.txt”<br>N = 2<br>Keyword = “new” | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “analyze”. | Program prompts user to enter a valid keyword | Screen shows first 10 analyzed “concordance” results | Screen shows first 10 analyzed “concordance” results | P |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | Valid input “keyword” is not found in input file | 1. Launch application.<br>2. Choose “analyze document”.<br>3. Enter “1.txt”.<br>4. Choose “concordance”.<br>5. Enter “2”.<br>6. Enter “rawr”. | Filename: “1.txt”<br>N = 2<br>Keyword = “rawr” | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “analyze”. | Program prompts user to enter a valid keyword | Error screen | Error screen | P |

| Test Scenario: User returns to previous screens | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **#** | **Test Description** | **Steps** | **Sample Data** | **Pre-Conditions** | **Post-Conditions** | **Expected** | **Actual** | **P/F** |
| 1 | Choosing “Back” in Main Menu | 1. Launch application.<br>2. Enter “6”. | | None | Program exits | Program exits | Program exits | P |
| 2 | Choosing “Back” in Cleaning Functions Screen without starting the cleaning | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “1.txt”.<br>4. Enter “8”. | Filename: “1.txt” | None | Program will restart from main menu | Screen returns to main menu | Screen returns to main menu | P |
| 3 | Choosing “Back” in Cleaning Functions Screen after continuing the cleaning | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “1.txt”.<br>4. Enter “1”.<br>5. Enter “Y”.<br>6. Enter “8”. | Filename: “1.txt” | None | Program will restart from main menu | Screen returns to main menu | Screen returns to main menu | P |
| 4 | Choosing “Back” in Analyzing Functions Screen | 1. Launch application.<br>2. Choose “analyze document”.<br>3. Enter “1.txt”. | Filename: “1.txt” | None | Program will restart from main menu | Screen returns to main menu | Screen returns to main menu | P |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 4. Enter “5”. | | | | | | |

| Test Scenario: Reading of Raw and Cleaned Project Gutenberg Documents | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **#** | **Test Description** | **Steps** | **Sample Data** | **Pre-Conditions** | **Post-Conditions** | **Expected** | **Actual** | **P/F** |
| 1 | Read a specific number of characters | 1. Launch application.<br>2. Choose “clean document.”<br>3. Enter input filename.<br>4. Choose “All”.<br>5. Enter output filename.<br>6. Press Enter key.<br>7. Choose Exit. | Filename: “a.txt”<br>Number of characters: 1000 | Config is initialized. Default number of content characters is configured. | Program exits. | File is created containing approximately less than 1000 content characters | File is created containing approximately less than 1000 content characters | P |
| 2 | Read all of the characters. | 1. Launch application.<br>2. Choose “clean document.”<br>3. Enter input filename.<br>4. Choose “All”.<br>5. Enter output filename.<br>6. Press Enter key.<br>7. Choose Exit. | Filename: “a.txt”<br>Number of characters: 0 | Config is initialized. Default number of content characters is configured. | Program exits. | File is created containing content characters approximately the size of the original document | File is created containing content characters approximately the size of the original document | P |
| 3 | Document has complete metadata.<br><br>Mode is “clean”. | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter filename. | Filename: “a.txt” | a.txt has complete metadata | Screen shows cleaning functions. | Screen shows metadata of the document | Screen shows metadata of the document | P |
| 4 | Document has complete metadata. | 1. Launch application.<br>2. Choose “analyze document”. | Filename: “a.txt” | a.txt has complete metadata | Screen shows analyzing functions. | Screen shows metadata of the document | Screen shows metadata of | P |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Mode is “analyze”. | 3. Enter filename. | | | | | the document | |
| 5 | Document has incomplete metadata.<br><br>Mode is “clean”. | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter filename. | Filename: “b.txt” | b.txt has incomplete metadata | Screen shows cleaning functions. | Screen shows metadata of the document | Screen shows metadata of the document | P |
| 6 | Document has incomplete metadata.<br><br>Mode is “analyze”. | 1. Launch application.<br>2. Choose “analyze document”.<br>3. Enter filename. | Filename: “b.txt” | b.txt has incomplete metadata | Screen shows analyzing functions. | Screen shows metadata of the document | Screen shows metadata of the document | P |
| 7 | File is too large | 1. Launch application.<br>2. | Number of content characters: 0 | Config is initialized. | Malloc for the string will fail and will return NULL. | Error screen suggesting user to limit characters | Error screen suggesting user to limit characters | P |

| Test Scenario: Cleaning a Project Gutenberg document | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **#** | **Test Description** | **Steps** | **Sample Data** | **Pre-Conditions** | **Post-Conditions** | **Expected** | **Actual** | **P/F** |
| 1 | Choosing the “to lowercase” option. | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “1.txt”.<br>4. Choose “to lowercase”<br>5. Do not continue cleaning.<br>6. Enter “1.txt”<br>7. Press Enter. | Filename: “1.txt” | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “clean”. | The cleaned file will be saved in “cleaned_files” folder with the name “1.txt” | All alphabetical characters are lowercase. Cleaned file displays the metadata and a series of words separated by a single space. | All alphabetical characters are lowercase. Cleaned file displays the metadata and a series of words separated by a single space. | P |
| 2 | Choosing the “remove special characters” option. | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “1.txt”.<br>4. Choose “remove special characters”<br>5. Do not continue cleaning.<br>6. Enter “1.txt”<br>7. Press Enter. | Filename: “1.txt” | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “clean”. | The cleaned file will be saved in “cleaned_files” folder with the name “1.txt” | All special characters are either removed or replaced with a single space. Cleaned file displays the metadata and a series of words separated by a single space. | All special characters are either removed or replaced with a single space. Cleaned file displays the metadata and a series of words separated by a single space. | P |
| 3 | Choosing the “remove numbers” option. | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “1.txt”.<br>4. Choose “remove numbers”<br>5. Do not continue | Filename: “1.txt” | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “clean”. | The cleaned file will be saved in “cleaned_files” folder with the name “1.txt” | All numbers are removed. Cleaned file displays the metadata and a series of words separated by a | All numbers are removed. Cleaned file displays the metadata and a series of words separated by a | P |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | cleaning.<br>6. Enter “1.txt”<br>7. Press Enter. | | | | single space. | single space. | |
| 4 | Choosing the “clean whitespaces” option. | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “1.txt”.<br>4. Choose “clean whitespaces”<br>5. Do not continue cleaning.<br>6. Enter “1.txt”<br>7. Press Enter. | Filename: “1.txt” | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “clean”. | The cleaned file will be saved in “cleaned_files” folder with the name “1.txt” | Cleaned file displays the metadata and a series of words separated by a single space. | Cleaned file displays the metadata and a series of words separated by a single space. | P |
| 5 | Choosing the “remove stopwords” option. | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “1.txt”.<br>4. Choose “remove stopwords”<br>5. Do not continue cleaning.<br>6. Enter “1.txt”<br>7. Press Enter. | Stopwords Filename: stopwords (in dat folder)<br><br>Input Filename: “1.txt” | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “clean”. | The cleaned file will be saved in “cleaned_files” folder with the name “1.txt” | All stopwords are removed. Cleaned file displays the metadata and a series of words separated by a single space. | All stopwords are removed. Cleaned file displays the metadata and a series of words separated by a single space. | P |
| 6 | Choosing the “all” option. | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “1.txt”.<br>4. Choose “all”<br>5. Enter “1.txt”<br>6. Press Enter. | Stopwords Filename: stopwords (in dat folder)<br><br>Input Filename: “1.txt” | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “clean”. | The cleaned file will be saved in “cleaned_files” folder with the name “1.txt” | Cleaned file shows metadata, only contains alphabetical words, does not contain any stopwords, and all words are separated by a single space. | Cleaned file shows metadata, only contains alphabetical words, does not contain any stopwords, and all words are separated by a single space. | P |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 7 | Choosing multiple options: “remove special characters” and “to lowercase” | 1. Launch application.<br>2. Choose “clean document”.<br>3. Enter “1.txt”.<br>4. Choose “remove special characters”.<br>5. Continue cleaning.<br>6. Choose “to lowercase”.<br>7. Do not continue cleaning.<br>8. Enter “1.txt”.<br>9. Press Enter. | Filename: “1.txt” | Config is initialized.<br><br>File “1.txt” is a valid input file.<br><br>Mode is “clean”. | The cleaned file will be saved in “cleaned_files” folder with the name “1.txt” | Cleaned file shows metadata. All special characters are either removed or replaced with a single space. All alphabetical characters are in lowercase. All words are separated by a single space. | Cleaned file shows metadata. All special characters are either removed or replaced with a single space. All alphabetical characters are in lowercase. All words are separated by a single space. | |

| Test Scenario: Analyzing Project Gutenberg document(s) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **#** | **Test Description** | **Steps** | **Sample Data** | **Pre-Conditions** | **Post-Conditions** | **Expected** | **Actual** | **P/F** |
| 1 | Choosing the “word count” option. | 1. Launch application.<br>2. Choose “analyze document (single-file)”.<br>3. Enter “a.txt”.<br>4. Choose “word count”<br>5. Press Enter. | Filename: “a.txt” | Config is initialized.<br><br>File “a.txt” is a valid input file. “a.txt” is a cleaned file that does not contain special characters and numbers, and all alphabetical characters are in lowercase.<br><br>Mode is “analyze”. | The complete analyzed file will be saved in “analysis” folder with the name “a_wcount.txt” | Analyzed file shows the word count of each word in “a.txt” | Analyzed file shows the word count of each word in “a.txt” | P |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 2 | Choosing the “n-gram count” option. | 1. Launch application.<br>2. Choose “analyze document (single-file)”.<br>3. Enter “a.txt”.<br>4. Choose “n-gram count”<br>5. Enter “3”.<br>6. Press Enter. | Filename: “a.txt”<br><br>N = 3 | Config is initialized.<br><br>All sample data are valid. “a.txt” is a cleaned file that does not contain special characters and numbers, and all alphabetical characters are in lowercase.<br><br>Mode is “analyze”. | The complete analyzed file will be saved in “analysis” folder with the name “a_ngram.txt” | Analyzed file shows the n-gram count (3) of each word in “a.txt” | Analyzed file shows the n-gram count (3) of each word in “a.txt” | P |
| 3 | Choosing the “concordance” option. | 1. Launch application.<br>2. Choose “analyze document (single-file)”.<br>3. Enter “a.txt”.<br>4. Choose “n-gram count”<br>5. Enter “3”.<br>6. Enter “still”.<br>7. Press Enter. | Filename: “a.txt”<br><br>N = 3<br><br>Keyword = “still” | Config is initialized.<br><br>All sample data are valid. All alphabetic characters in “a.txt” are in lowercase.<br><br>Mode is “analyze”. | The complete analyzed file will be saved in “analysis” folder with the name “a_concord.txt” | Analyzed file shows all concordance cases of the keyword (“still”) given N (3) | Analyzed file shows all concordance cases of the keyword (“still”) given N (3) | P |