ELEN006W Engineering Programming

Assignment 1

Farah Sinada

w1575467

Completion Date: 6/1/2017

Lecturer: Andrzej Tarczynski

Declaration:

I declare that this report and attached C++ programme were written entirely and solely by myself. I am aware of the University of Westminster regulations on plagiarism and penalties that are imposed on those who break the regulations.

2. Design of the software:

The aim of this programme is to read a text file and to determine the number of characters, text objects, words and sentences, of which there are interrogative and exclamatory. Integer variables for each counter were set to zero at the beginning of the program. The initial states of the Boolean expressions were also set. The overall flowchart of this programme is shown below:

START

TextObjCount = 0 WordCount = 0

TObjStart = true WordStart = true

TObjStarted = false WStarted = false

SenCount = 0 SenExCount = 0

SenQCount = 0 String: Filename

Ask user for filename and open file

Read character from file

Is MyFile good?

Is it not a Whitespace?

Terminate program

Is letter?

WordStart?

Is Whitespace character?

Is Sentence Ender?

Character Count ++

Process Text Object

Process Word

Process Text Object

Process Word

Process Word and Sentence

Process Word

Word Started?

Word Started?

Process Word

WordCount ++

YES

NO

YES

YES

YES

YES

YES

YES

YES

NO

NO

NO

NO

NO

NO

NO

Is ‘,’?

WordStart?

Process Word

WordCount ++

NO

YES

YES

NO

From this flowchart it can be seen that there are a number of different processes for each section of the program. There are also a number of Boolean expressions that are used as markers. They are: Text Object Start and Started and Word Start and Started. The word markers are also used for sentences.

The first process is the file termination process. This occurs if the file is deemed not ‘good’. There are a number of output processes that occur if this is deemed true. Firstly, the Boolean markers are checked to see whether they are true or false. Their state determines the final action to be taken with the counters.

The character counter is also checked. If it has a value of zero, then this means that the file either doesn’t exist, or it is empty. If this is the case, the sentence: “The file is empty” is printed.

If the character counter is not zero, then the other calculated statistics are printed.

**File Termination flowchart:**

Is MyFile good?

NO

Is Text Object Started?

Text Object Count ++

Is Word Started?

YES

YES

NO

Character Count == 0

Print file statistic: number of characters, text objects, words, total sentences, and exclamatory and interrogative sentences

Print: “This is an empty file”

STOP

Word Count ++

NO

NO

YES

Process file

YES

After the result is printed, this line of code is executed:

cout << "Press any key to continue "; cin >> Z;

This line is used to stop the .exe file of the program to close before displaying the results on screen.

The next process is the character counter; from the overall flowchart it can be seen that it is incremented by one through every iteration of the code.

Boolean functions were created to process the characters in the file. There are three functions; the first is for letter recognition, the second is for whitespace characters and the third for sentence punctuation. They were used to create the ‘if’ statements that govern the behaviour of the program. The first ‘isLetter’ function is defined as the upper case and lower case letters of the alphabet. The second ‘isWhite’ function is defined as the ‘new line’, ‘tab’ or ‘space’ characters. The third and final function ‘isSentEnder’ is defined as the end of sentence punctuation marks, which are: ‘.’, ‘?’ and ‘!’.

Since they are Boolean functions, they return a value of either true or false when the character encountered either fits their definition or does not. Below are the functions:

bool isLetter(char B)

{

bool result;

result = (B >= 'A' && B <= 'Z') || (B >= 'a' && B <= 'z');

return result;

}

bool isWhite(char C)

{

bool result;

result = (C == ' ' || C == '\n' || C == '\t');

return result;

}

bool isSentEnder(char D)

{

bool result;

result = (D == '.' || D == '!' || D == '?');

return result;

}

The next stage is processing a text object. This is covered in two areas. Since a text object is a group of characters separated by a whitespace character, there are two questions that are used to calculate how many there are; they are ‘is the character a whitespace character?’ and ‘is it not a whitespace character?’.

If it is a whitespace character, then the Text Object Start marker is set to true. When it is not a whitespace character it is set to false and Text Object Started is set to true. When the end of the text object is reached via a whitespace character, Text Object Started is set back to false, Text Object Start is set to zero and one is added to the text object counter. This is illustrated in the flowchart below.

**Text Object flowchart:**

Is it not a Whitespace character?

Is it a whitespace character?

Text Object Start = false

Text Object Started = true

Process letter

Text Object Count ++

Text Object Started?

Text Object Start = true

Text Object Started = false

Process Word

YES

YES

YES

YES

NO

NO

NO

NO

Text Object Start?

Read character from file

Other processes

Continue program

Word processing is similar. There are four separate places where words are processed and those are in the letter, whitespace character, comma and the sentence ender statements.

The sentence ender and comma statements are needed to be able to process words that are succeeded not by whitespace characters but rather by sentence ender punctuation marks, which are ‘.’, ‘?’ and ‘!’ and by a comma.

The comma statement is identical to the letter statement, only differing in that if a word has started, the word counter would be incremented.

**Word flowchart:**

Is letter?

WordStart?

Is Whitespace character?

Is Sentence Ender?

Word Start = false

Word Started = true

Process Text Object

Process Sentence

Word Start = false

Word Started = false

Word Started?

Word Started?

Word Start = true

Word Started = false

WordCount ++

YES

YES

YES

YES

YES

YES

NO

NO

NO

NO

NO

NO

Word Start = false

Word Started = false

Word Count ++

Sentence Count ++

Read character from file

Other processes

Is ‘,’?

WordStart?

WordCount ++

YES

Word Start = false

Word Started = false

YES

NO

If all the statements are not applicable, then the Word Start and Word Started markers are both set to false at the end of the loop. This is because if none of the conditions are met, this means that characters other than letters were detected and therefore cannot be counted as words.

Sentences are processed in the ‘is sentence ender?’ statement. Using the Word Started marker, if a word had already started and was succeeded by a sentence ender punctuation mark, then the sentence counter would be incremented by one. Within the same statement it is asked whether that sentence was ended by an exclamation mark or a question mark. The respective counter is incremented if those statements are found to be true.

If a word has not started the Word Start and Word Started variables are set in the same way as at the end of the loop. This would mean that if a sentence is incorrectly punctuated in that a punctuation mark is immediately succeeded by a letter, it would not count as a word. It would however still count as a sentence if there were other words that were preceded by a whitespace and ended with a sentence ender punctuation mark. Here is an example: ‘.This is a sentence.’ would still count as a sentence because a whitespace character precedes the word ‘is’.

**Sentence flowchart:**

Is Sentence Ender?

Word Start = false

Word Started = false

Word Count ++

Sentence Count ++

Continue program

Word Started?

Sentence Exc Count ++

Sentence Q Count ++

YES

YES

NO

NO

Is ‘A’ == ‘?’

Is ‘A’ == ‘!’

YES

YES

NO

NO

Read character from file

Other processes

Program code:

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

bool isLetter(char B);

bool isWhite(char C);

bool isSentEnder(char D);

int main() {

char Z; //to stop file exiting before outout is shown

char A; //characters

int CharCount = 0;

int TObjCount = 0; //text objects

bool TObjStart = true;

bool TObjStarted = false;

bool WordStart = true; // words, also using WordStart as sentence start marker

bool WStarted = false;

int WordCount = 0;

//total sentences

int SenCount = 0;

//total exclamatory and interrogative sentences

int SenExcCount = 0;

int SenQCount = 0;

string Filename; //filename

cout << "Please enter filename with extension: ";

cin >> Filename;

ifstream MyFile(Filename);

while (true)

{

MyFile.read(&A, 1);

if (!(MyFile.good()))

{

if (TObjStarted) TObjCount++;

if (WStarted) WordCount++;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

if (CharCount == 0) //for an empty file

{

cout << "\nThis is an empty file\n";

cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "Press any key to continue "; cin >> Z;

break;

}

cout << "\nThere are " << CharCount << " characters,\n";

cout << "\n " << TObjCount << " text objects,\n";

cout << "\n " << WordCount << " words,\n";

cout << "\n " << SenCount << " sentences,\n";

cout << "\n " << SenQCount << " interrogative sentences and\n";

cout << "\n " << SenExcCount << " exclamatory sentences in this file\n";

cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "Press any key to continue "; cin >> Z;

break;

}

CharCount++; //it will always be executed, if at end of loop can be skipped, also doesnt count eof file character

if (!isWhite(A))

{

if (TObjStart)

{

TObjStart = false;

TObjStarted = true;

}

}

if (isLetter(A))

{

if (WordStart)

{

WordStart = false;

WStarted = true;

}

continue;

}

if (isWhite(A))

{

if (TObjStarted) TObjCount++; //text object counter

TObjStart = true;

TObjStarted = false;

if (WStarted) WordCount++; //for words

WordStart = true;

WStarted = false;

continue;

}

if (A == ',')

{

if (WStarted) WordCount++; //for words

WordStart = false;

WStarted = false;

continue;

}

if (isSentEnder(A))

{

if (WStarted)

{

SenCount++;

WordCount++;

if (A == '?')

{

SenQCount++;

}

else if (A == '!')

{

SenExcCount++;

}

}

WordStart = false;

WStarted = false;

continue;

}

WordStart = false;

WStarted = false;

}

return 0;

}

bool isLetter(char B)

{

bool result;

result = (B >= 'A' && B <= 'Z') || (B >= 'a' && B <= 'z');

return result;

}

bool isWhite(char C)

{

bool result;

result = (C == ' ' || C == '\n' || C == '\t');

return result;

}

bool isSentEnder(char D)

{

bool result;

result = (D == '.' || D == '!' || D == '?');

return result;

}

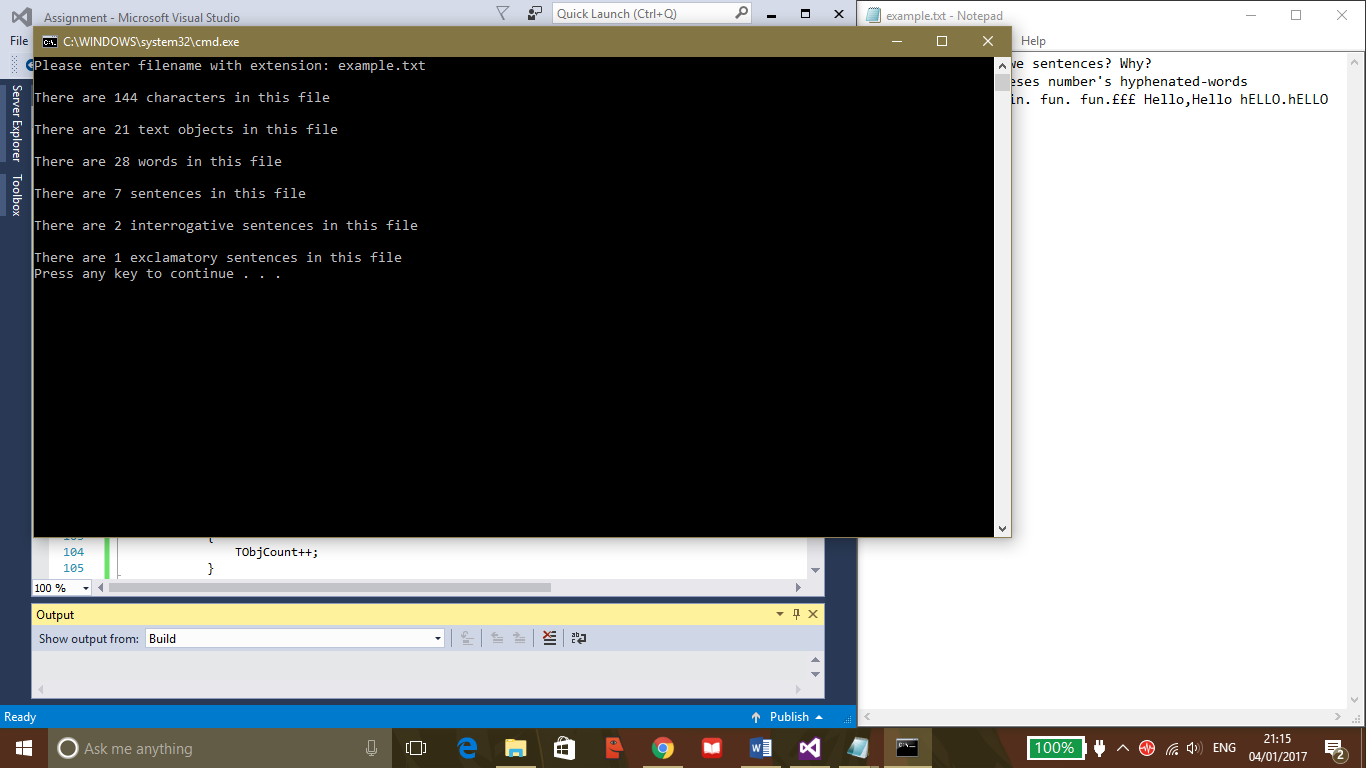
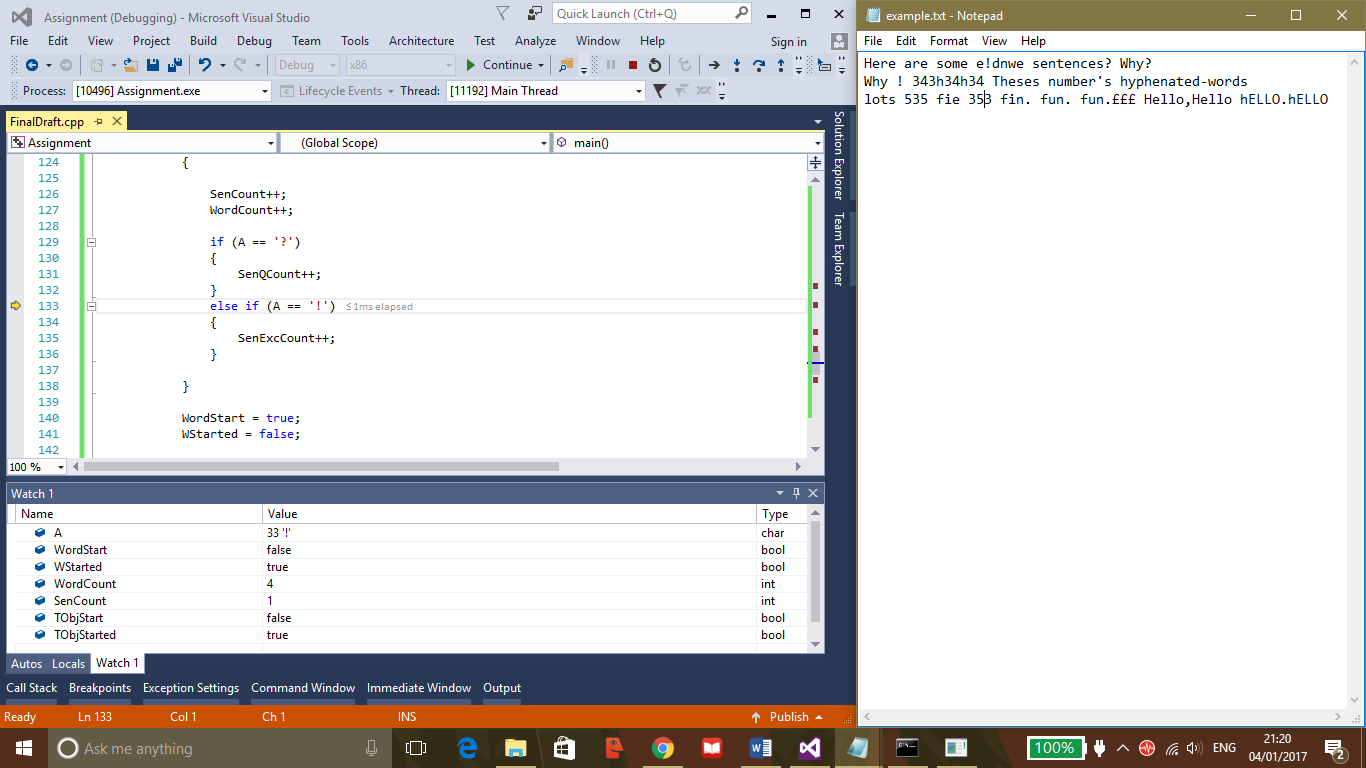
3. Report on programme debugging:

Each section of the program was written separately and added sequentially to the overall program. As each section was added, the program was tested using the debugger mode and the ‘watch’ function to check whether it still produced the same results, to understand what was occurring with the variables, and to also check whether the Boolean expressions were reacting appropriately.

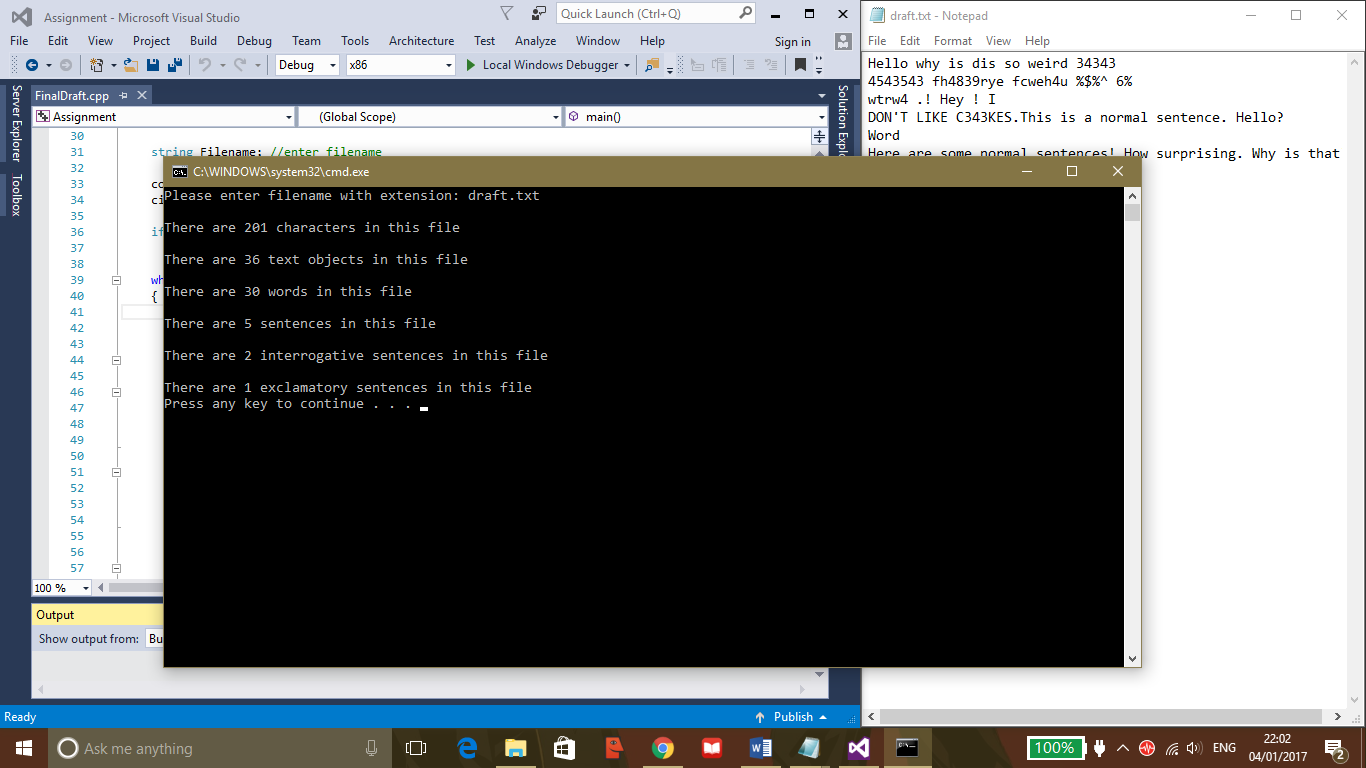
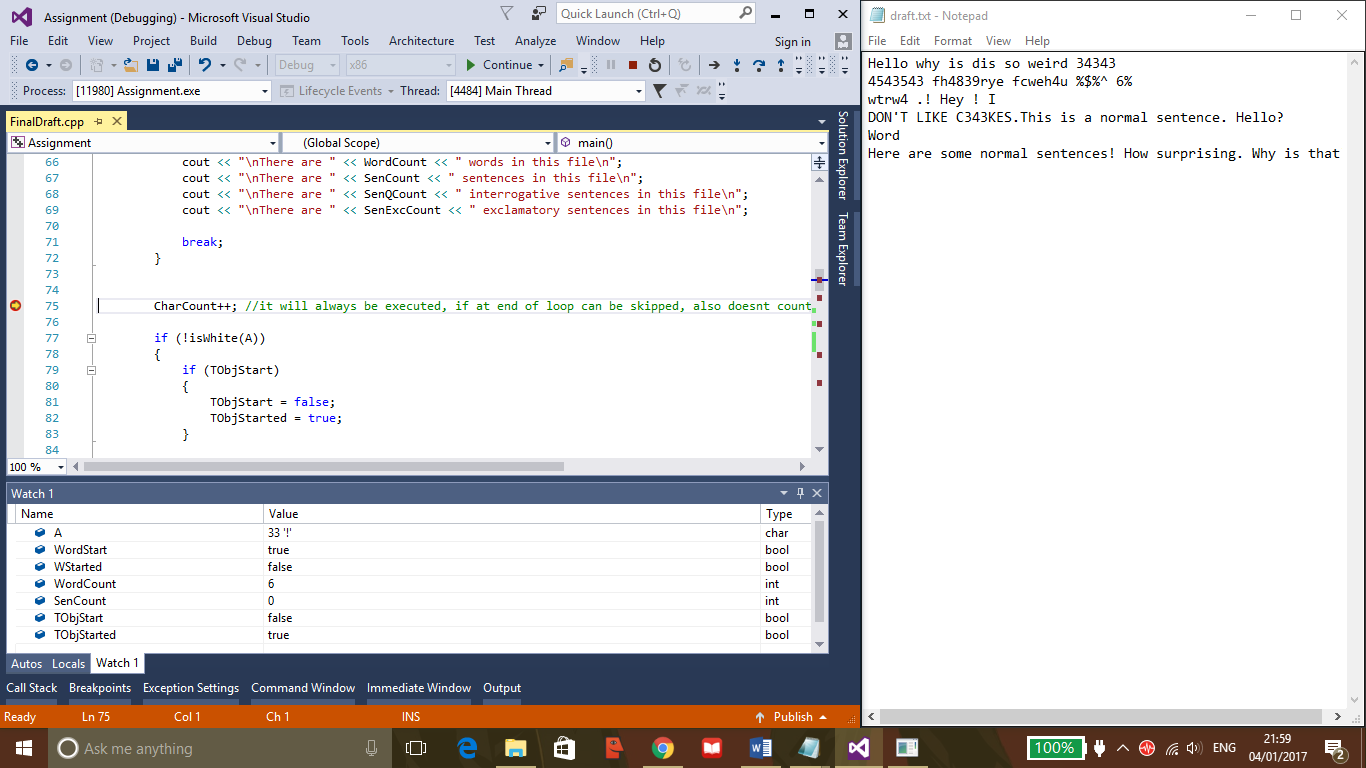
The first section that was written was the character counter. After testing with a text file, the word count program was added. Testing the program produced some unexpected results for the character count of the file, but the word count worked as expected.

Using the debugger mode and the ‘watch’ function, the character counter variable was monitored. Break points were added to the character counter and the beginning of the word processing statement. It then became clear that as it was placed at the end of the loop, it would only count the characters if all other conditions were false. Therefore, moving the character counter to the top of the loop underneath the exit conditions solved this problem as it was then processed before any other function. It was not placed before the exit condition as it would otherwise count the end of file character and add it to the overall sum.

Testing the sentence processing statements provided the following outputs for the following text files:

1.

This file showed more sentences than there should have been.

2.

This file showed more words than were actually there. It was then recognised that the fault was due to the method in which the Boolean word markers were set in the ‘isSentEnder’ ‘if’ statement. I previously had the expressions reset the same as in the whitespace ‘if’ statement. These statements were within the ‘isSentEnder’ statement:

WordStart = true;

WStarted = false;

This would mean that words that started straight after the end of a sentence without being preceded by a whitespace character would be counted as words, therefore allowing the sentence to be counted. This was fixed by setting word start and started back to false as shown below.

WordStart = false;

WStarted = false;

When first writing the text object section, an ‘else’ statement was added to the ‘is whitespace?’ statement. This was to avoid having two ‘if’ statements that could essentially perform the same function. However, after testing with other sections of the program a separate ‘if’ statement for ‘is not a whitespace character?’ was created. This was because the ‘else’ section of the original whitespace statement, if true, would affect the word counting process and thereby affect the functioning of the rest of the program.

The final addition to the program was changing the method by which the program chose the text file that was to be opened. Originally, the filename was declared within the ifstream variable, thereby meaning that if a different file was to be chosen it had to be changed within the code. As show below:

ifstream MyFile("draft.txt");

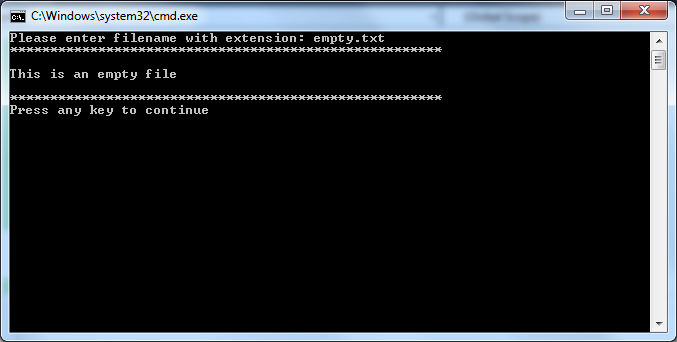
This was changed by adding a string variable called filename and using the user’s input to type in the name of the file. This meant that different files could be opened without changing the code.

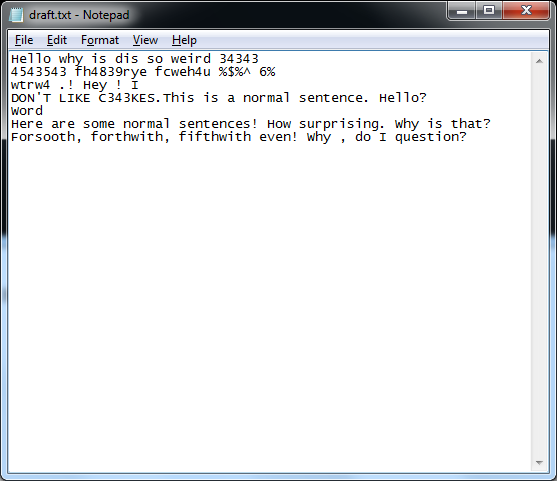
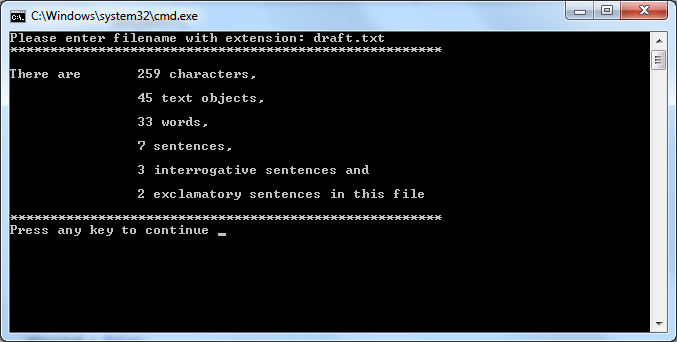
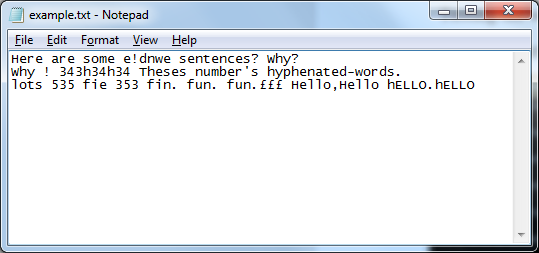
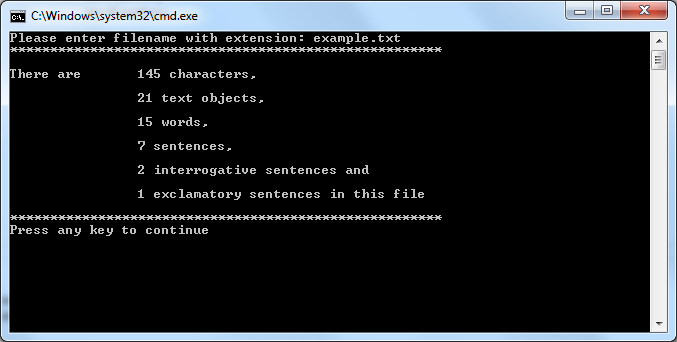
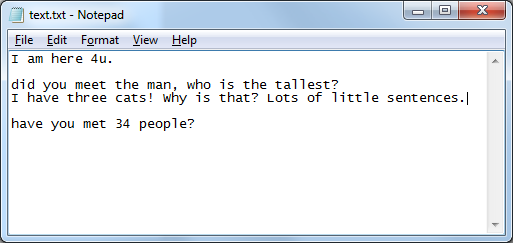
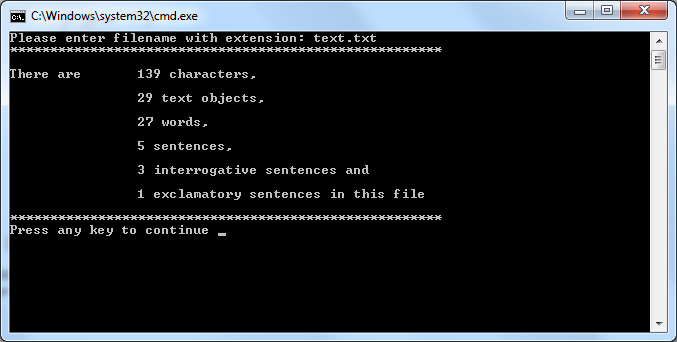
string Filename;

ifstream MyFile(Filename);

Here are the expected output screens and their corresponding text files:

1. The first is empty.txt, and it is an empty file. This is its output:



1. The next file is draft.txt. Here is its text and output:
2. The next file is example.txt. Here is its text and output:
3. Final file is text.txt. Here is its text and output:

4. User manual:

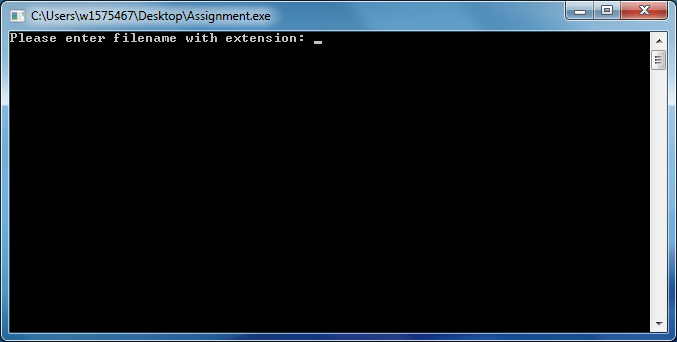
This program reads the contents of a text file, and outputs statistics relating to it. It tells you the total number of: characters, text objects, words and sentences of which there are interrogative and exclamatory in a text file.

Limitations:

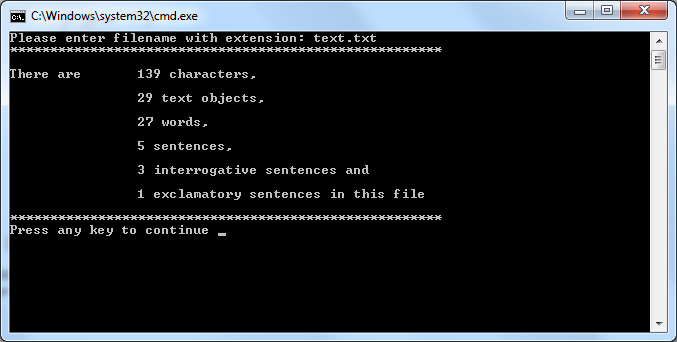
* You can only enter the filename of a file that is in the project folder of this project, it does not work if the file is located outside the folder.
* Nonsense and misspelled words are counted as ‘words’.
* Sentences will not be counted if not correctly punctuated, i.e. if an interrogative sentence is ended like so: ‘What is that ?’ As opposed to this: ‘What is that?’ then it would not be counted.
* Sentences that have ‘not words’ in the middle will count as a sentence, e.g.: ‘I ate 4 apples’.

Instructions for the use of the program:

1. Open the project folder named Assignment, open the debug folder and double click on the assignment.exe file.
2. Upon launch of the program you will be asked to enter the filename.



* + There are 4 files to choose from: draft.txt, example.txt, text.txt and empty.txt. You must choose one and type its name. It must be spelled exactly and must be entered with its extension, for example: draft.txt.

1. The program will then calculate statistics and output the result on the screen. Here is an example:
2. If you would like to calculate statistics for another file, close the window and start from step 1 again.