

1 Introduction

This project is on creating a spell checking program that gives the user suggestions on the correct spelling of incorrectly spelled words in a document. As it states in the syllabus.

Projects are to be done strictly on your own and as with all assignments the sharing of files and code is strictly prohibited and constitutes an act of Academic Misconduct. Furthermore the use of any electronic medium, such as code repositories, forums, blogs, message boards, email, etc. is strictly prohibited and constitutes an act of Academic Misconduct.

The only person you may discuss this with in any form is me. You may use the textbook, the textbook example code, and the class example code that is posted on the MyClasses site.

When you are ready to submit your work create a folder called `Project01` in that folder have separate folders for each project, one folder per project. Put all the code files needed for that project in its respective folder. Do not include the files that the IDE creates, I just want the code files. Zip the entire `Project01` folder up into a single zip file and submit it.

2 Spell Checker

For this project you will produce three spellchecking programs. The first will check a single word input by the user and the second will allow the user to select a text file to load and spell check the entire file. The third is an interactive program that will allow the user to select replacement words during the spell checking process and a new file will be created with the corrected words. These obviously build on each other so you will probably want to complete them in order.

2.1 Program #1

This program is a single word case insensitive checker. The program will start by asking the user for a dictionary file. There are two that are supplied with the project. The `words.txt` file has about half a million words in it and the other `Dict.txt` contains about 49,000 words. Both of these are from US English dictionaries that are used by many Linux applications. You may wonder why you would use a dictionary that is a tenth of the size as another. A smaller dictionary would have an advantage over larger ones since it would contain words that were not as obscure as some in the other dictionary. If a user stumbled onto one of these more obscure words the smaller dictionary would flag it as misspelled but the larger dictionary would not. So for the average user the smaller file might be more accurate.

After the program processes a word it should ask the user if they would like to check another word and continue until they select to quit.

When the program checks a word it should state if the word is in the dictionary or not. If not the program should give the user suggestions on what the correct word might

be. Spell checkers usually have algorithms to determine the “distance” between words, i.e. quantify the difference, and they suggest words that are “close” to the one that is not in the dictionary. Here we will make lists of suggestions from common errors in spelling.

- Suggest words if they are one character different than the one the user typed in. For example, `berr` gave a suggestion of `BEAR`.
- Suggest any word that is the transpose of two adjacent letters. For example, the input of `sawn` produced `SWAN`.
- Suggest any word that is simply adding a character to what is input. For example, `berr` gave a suggestion of `BERRY`.
- Suggest any word that is simply removing a character to what is input. For example, `sawn` gave a suggestion of `SAW`.
- Suggest any word that is a continuation to what is input. For example, `berr` gave a suggestion of `BERRYLIKE`.

Several of these may produce the same suggestions. For example, when `berr` was processed it gave a suggestion of `BERRY` during both adding one letter and the word continuation. Make sure that your program has only one occurrence of any suggestion. A run of the program is below.

2.1.1 Example

```
Input dictionary filename: Dict.txt
Input a single word to spell check: berr

BERR is not in the dictionary.

Suggestions
-----
HERR
KERR
TERR
BARR
BURR
BEAR
BEER
BERG
BERK
BERM
BERN
BERT
BERRA
BERRY
ERR
BRR
BERRYLIKE

Check another word? Y/N: y

Input a single word to spell check: weaj
```

```
WEAJ is not in the dictionary.

Suggestions
-----
WEAK
WEAL
WEAN
WEAR

Check another word? Y/N: y

Input a single word to spell check: were

WERE is in the dictionary.

Check another word? Y/N: y

Input a single word to spell check: sawn

SAWN is not in the dictionary.

Suggestions
-----
DAWN
FAWN
LAWN
PAWN
YAWN
SEWN
SOWN
SWAN
SHAWN
SPAWN
AWN
SAN
SAW

Check another word? Y/N: n
```

2.2 Program #2

For the second program you will have the user input the dictionary file and the text file to be spellchecked. You will also give the user the option of sending the output to the screen, a file, or both. If the selection is a file or both you will ask the user for a filename to store the output. The program will then go through the document to be checked and report all misspelled words (according to the given dictionary) and suggestions for all of them. Since these documents that are being checked will have punctuation you will need to remove these. For example, if there is a phrase “He went to the game.” i a document you would need to remove the “ before He and the .” after game. On the other hand if you hit the contraction I’ve you would not want to remove the ’ and if you hit a hyphenated word like high-speed you would not want to remove the hyphen. So when you are removing punctuation leave the hyphens and apostrophes in place. This will create some problems with single quotes but that will be far less common than a contraction. The spell checking here is also to be case insensitive.

2.2.1 Example

```
Input dictionary filename: Dict.txt
Input document filename: TestDoc001.txt
Send output to the screen, a file, or both? S/F/B: b
Input document filename: out.txt
```

```
REFERS is not in the dictionary.
Suggestions
-----
REVERS
REFER

METHODS is not in the dictionary.
Suggestions
-----
METHOD

CURRENTLY is not in the dictionary.
Suggestions
-----
None

PROFESSIONALS is not in the dictionary.
Suggestions
-----
PROFESSIONAL

ATTACKS is not in the dictionary.
Suggestions
-----
ATTUCKS
ATTACK

HUMANS is not in the dictionary.
Suggestions
-----
HUMANE
HUMAN

HIGH-SPEED is not in the dictionary.
Suggestions
-----
None

COMPUTERS is not in the dictionary.
Suggestions
-----
COMPUTER
```

If there are no misspelled words the program should display a message to indicate this.

```
Input dictionary filename: words.txt
Input document filename: TestDoc001.txt
Send output to the screen, a file, or both? S/F/B: s
No misspelled words.
```

Some things to keep in mind when writing this. You are clearly doing a lot of searching in large data sets to complete this. As you know from class the binary search is much faster than the linear search but it requires that the files being searched are sorted. Neither of the

two dictionary files are sorted by ASCII values, which is what you will need. Remember there is a fast sorting algorithm for arrays and vectors in the algorithms include file.

2.3 Program #3

This program is a full featured spell checker that will also save and reformat the corrected document. This one will be case sensitive. The program will ask for a dictionary file and document to spell check. When it hits a word that is not in the dictionary it will display the word, the context of the word in the document, 4 words on either side of the misspelled word and the word itself will be in <<< and >>> to have it stand out. The program will also let the user select an action for the word. They can select K to keep the word, C to change the word by typing in the new word, or select from the list of suggestions. The suggestion list is created as in the last two programs but remember that this one is case sensitive. Only the first 10 suggestions should be displayed and the user will select these by the number keys.

When the checking is finished the program should save the new document to the file of the same name as the input document but with `Checked_` in front. The new file should also be reformatted so that each line contains at most 80 characters and the maximum number of words are being used on each line. In other words, each line could not take the next word or it would exceed 80 characters and the most words were used on each line. For example, from the example below the first line,

```
Classical cryptography refers to encryption methods that are no longer in use
```

has fewer than 80 characters. Adding the next word `today` would make the line too long. Also, moving the last word `use` to the next line would make the first line too short since we are not using the maximum number of words we can and stay under 80 characters.

Some further specifications for this program are as follows.

1. If the original word is capitalized then the replacement word should be as well.
2. Once a word has a replacement that replacement should be used for any further occurrences of the word. For example, in the run below we selected to keep the word `methods` then from there on out every occurrence of `methods` was kept.
3. When checking for a word in the dictionary make sure you check lowercase versions of capitalized words. For example, in the run below the word `Classical` is not flagged as a misspelling since the word `classical` is in the dictionary.

2.3.1 Example

The original document

Classical cryptography refers to encryption methods that are no longer in use today because they are no longer secure. Modern cryptography refers to encryption methods that are currently in use. The division between classical and modern is a little disputed among the professionals but the primary force between the two is the invention of the computer. The classical methods were strong enough to withstand attacks by humans but are easily broken with high-speed computers, hence the reason they are not in use today.

The saved document after running the program

Classical cryptography refers to encryption methods that are no longer in use today because they are no longer secure. Modern cryptography refers to encryption methods that are currently in use. The division between classical and modern is a little disputed among the professionals but the primary force between the two is the invention of the computer. The classical methods were strong enough to withstand attacks by humans but are easily broken with high-speed computers, hence the reason they are not in use today.

The console input and output from the program run on this file.

```
Input dictionary filename: Dict.txt
Input document filename: TD2.txt
```

```
"refers" is not in the dictionary.
```

```
Context: Classical cryptography <<< refers >>> to encryption methods that
```

```
Select Action
-----
```

```
K: Keep word as is.
C: Input word to replace.
1: revers
2: refer
```

```
Selection: k
```

```
"methods" is not in the dictionary.
```

```
Context: cryptography refers to encryption <<< methods >>> that are no longer
```

```
Select Action
-----
```

```
K: Keep word as is.
C: Input word to replace.
1: method
```

```
Selection: k
```

```
"longger" is not in the dictionary.
```

```
Context: methods that are no <<< longger >>> in use today because
```

```
Select Action
-----
```

```
K: Keep word as is.
C: Input word to replace.
1: logger
2: longer
```

```
Selection: 2
```

"tody" is not in the dictionary.

Context: no longer in use <<< tody >>> because they are no

Select Action

K: Keep word as is.

C: Input word to replace.

1: Cody

2: Jody

3: body

4: tidy

5: tony

6: toady

7: toddy

8: today

9: toy

Selection: 8

"seccure" is not in the dictionary.

Context: they are no longer <<< seccure. >>> Modern cryptography reffers to

Select Action

K: Keep word as is.

C: Input word to replace.

1: secure

Selection: 1

"reffers" is not in the dictionary.

Context: longer secure. Modern cryptography <<< reffers >>> to encryption methods that

Select Action

K: Keep word as is.

C: Input word to replace.

Selection: c

Input replacement word: refers

"currently" is not in the dictionary.

Context: encryption methods that are <<< currently >>> in luse. The division

Select Action

K: Keep word as is.

C: Input word to replace.

Selection: k

"luse" is not in the dictionary.

Context: that are currently in <<< luse. >>> The division between classical

Select Action

K: Keep word as is.

C: Input word to replace.

1: Duse
2: Muse
3: fuse
4: muse
5: ruse
6: lase
7: lose
8: lube
9: luge
0: lure

Selection: c
Input replacement word: use

"is" is not in the dictionary.

Context: between classical and modern <<< is >>> a little disputed aong

Select Action

K: Keep word as is.
C: Input word to replace.
1: As
2: Ms
3: Os
4: id
5: if
6: ii
7: in
8: it
9: iv
0: ix

Selection: k

"aong" is not in the dictionary.

Context: is a little disputed <<< aong >>> the professionals but the

Select Action

K: Keep word as is.
C: Input word to replace.
1: Cong
2: Hong
3: Kong
4: Long
5: Wong
6: Yong
7: bong
8: dong
9: gong
0: long

Selection: c
Input replacement word: among

"professionals" is not in the dictionary.

Context: little disputed among the <<< professionals >>> but the primary force

Select Action

K: Keep word as is.

C: Input word to replace.
1: professional

Selection: k

"stronh" is not in the dictionary.

Context: The classical methods were <<< stronh >>> enough to withstand attacks

Select Action

K: Keep word as is.
C: Input word to replace.
1: strong

Selection: 1

"attacks" is not in the dictionary.

Context: strong enough to withstand <<< attacks >>> by humans but are

Select Action

K: Keep word as is.
C: Input word to replace.
1: attack

Selection: k

"humans" is not in the dictionary.

Context: to withstand attacks by <<< humans >>> but are easily broken

Select Action

K: Keep word as is.
C: Input word to replace.
1: humane
2: human

Selection: k

"high-speed" is not in the dictionary.

Context: are easily broken with <<< high-speed >>> computers, hence the reason

Select Action

K: Keep word as is.
C: Input word to replace.

Selection: k

"computers" is not in the dictionary.

Context: easily broken with high-speed <<< computers, >>> hence the reason they

Select Action

K: Keep word as is.
C: Input word to replace.
1: computer

Selection: k

```
"nt" is not in the dictionary.

Context:  the reason they are <<< nt >>> in use today.

Select Action
-----
K: Keep word as is.
C: Input word to replace.
1: At
2: Ct
3: It
4: Lt
5: Mt
6: Pt
7: St
8: Ut
9: Vt
0: at

Selection: c
Input replacement word: not

Done spell checking the file. Saving new file.
Done.
```

3 Grading

The program itself should, of course, be nicely formatted and commented and should follow all the other rules of good programming style. Make sure you are following all the coding and documentation standards of the class that are published on the MyClasses site for this class.

The grading of the project will take two forms, a sample run and an inspection of the code. If the program does not run you will receive a zero for that portion. So even if the program is not complete you will get a better grade for a partial program that runs verses a program that does not run. So I would suggest a completion in stages approach. The run portion of the grading will test the user interface for usability and conforming to the specifications I have outlined above. The code inspection portion of the grade will involve commenting, readability, correct indentation, variable names, structure and style, correctness, and conforming to specifications.

I am looking for clean, easy to read, code. A good use of functions without overuse. Commenting that gets the point across concisely. An easy to use interface with nice looking output.