ECS 60 Programming Assignment #4, 50 points (2 hours) Winter 2017

Due Wednesday, March 1st, 11:59pm.
Files NOT to handin to cs60 p4 are: SpellRunner.cpp, mynew.*, CPUTimer.h,
Minimum files to handin to cs60 p4 are: Makefile, speller.h, speller.cpp, authors.csv.

You are to write an efficient program that will read a dictionary of 100,000 words, and then check a document of 2,000,000 words. The program should insert into an array the position of each misspelled word. I have provided the driver program SpellRunner.cpp, and the necessary class stubs in speller.h, and speller.cpp. You may add any classes and/or code you wish to speller.cpp, and speller.h. You may also use, and alter any Weiss files. CreateDocs.out creates files used for testing. All files mentioned, as well as my executable, speller.out, can be found in ~ssdavis/60/p4

Further specifications:

1. Command Line parameter is the name of the Dict file. SpellRunner parses the filename to determine the name of the corresponding Doc and Wrongs files.
2. Files
   - 2.1. Since SpellRunner.cpp handles all file input, you need not concerned with how to read them.
   - 2.2. File names are appended with three numbers: number of words in dictionary size, number of words in the document words, and the seed used for the random number generator.
   - 2.3. Dict files contain the words of the dictionary, each on its own line. You may assume that dictionary words will never be more than 20 characters long. Here is listing of number occurrences of different lengths of words from a set of 3 Dict files with 100,000 words each. You may wish to do further analysis.

```
[ssdavis@lect1 p4]$ awk '{print length}' Dict*.txt | sort | uniq -c | sort -n -k 2
    304 2          48689 7          12733 12          382 17
   3540 3          46551 8           7551 13          138 18
  14642 4          40081 9           3609 14           14 19
  27511 5          30596 10          1894 15      [ssdavis@lect1 p4]$
  40663 6          20400 11           702 16
```

   - 2.4. Doc files contain the words of the document, each on its own line.
   - 2.5. Wrongs files contain the position of each misspelled word in the corresponding Doc file.
     - 2.5.1. SpellRunner checks the array you fill to see if it matches the contents of the Wrongs file. It will check both for misspelled words that your program missed, and correctly spelled words that your program reported as misspelled.
   - 2.6. To copy the Doc files to your current directory type: **`cp ~ssdavis/60/p4/*.txt .`**
3. Dynamic Memory Allocation.
   - 3.1. You may not use malloc(), free(), maxRAM, or currentRAM anywhere in your source code, including any Weiss files. No entity larger the 80 bytes may be created without using new. If you use any outside files, other than template files, you must add #include “mynew.h” at the top of the files.
   - 3.2. The underlying code of the overloaded new provided by mynew.cpp uses the fact that the real memory manager works in increments of 16 bytes, and charges 8 bytes for its overhead no matter what size is requested. Using mem.cpp, I found that requesting 1 to 24 bytes costs 32 bytes, requesting 25 to 40 bytes costs 48 bytes, and requesting 41 to 56 bytes costs 64 bytes. It is these costs that are summed to determine your currentRAM figure. maxRAM holds the largest value currentRAM ever held.
4. Measurements
   - 4.1. CPU time starts just before constructing your Speller object in main(), and ends after your check() function returns. Thus, your destructors will not be called during CPU time.
     - 4.1.1. You may not have any global variables since they would be constructed before CPU time starts.
   - 4.2. "Real RAM" is the maximum size of real dynamic RAM to which your program ever grew.
5. Grading
   - 5.1. The program will be tested using three 100000 word dictionary, and 2,000,000 word document. The measurements will be the total of the three runs.
   - 5.2. If there are ANY error messages from SpellRunner, then the program will receive zero.
   - 5.3. If your code has no error messages, and takes less than 30 CPU time, then you will receive at least 20 points.
   - 5.4. CPU Time score = 15 * Sean's CPU / Your CPU)
   - 5.5. maxRAM score = 15 * Sean's max RAM / Your maxRAM)
   - 5.6. final score = min(55, 20 + CPU Time score + maxRAM score).
   - 5.7. You will lose 5 points if your program has a memory leak as determined by valgrind.
6. Makefile. The Makefile provided contains the minimum needed to create speller.out.

6.1. You may not use an optimization flags in your Makefile.
6.2. All code must be C++ source code.
6.3. To ensure that you handin all of the files necessary for your program, you should create a temp directory, and copy only *.cpp, *.h, and Makefile into it. Then try to make the program. Many students have forgotten to handin dsexceptions.h.

7. Suggestions.

7.1. Start early, and get something working without errors.
7.2. Don't fuss about speed or size until you have something working. Too many students fail to have a correct program by the deadline because they spend too much time tweaking early on.
7.2.1. You will learn a lot just getting it running. Then you can use this knowledge to improve your code.
7.3. To debug, add if-statements that are only true during the state in which you are interested. For example, if my program did not catch a misspelled word at position 5093, then I would add to my Speller::check() method "if (i == 5093) cout << "Help\n";". I then can set a breakpoint at the "cout << "Help\n";" line, and step through the code to see what is going on.
7.4. Leave lots of time for testing and debugging. Test with all of the available files.
7.5. Though I didn't use this fact for the benchmark, you should be aware that since the alphabet of words is composed of only the 26 lowercase letters. This means that you would need only 5 bits to discriminate between all the possible characters in a document.

```
int main(int argc, char* argv[])
{
  char line[80], **dictionary, **document, *dictFilePtr, *docFilePtr;
  int *wrongs, *misspelled, misspelledCount = 0, seed, dictSize, docSize,
    wrongCount, tempMaxRAM, tempCurrentRAM;
  strcpy(line, argv[1]);
  strtok(line, "-");
  dictSize = atoi(strtok(NULL, "-"));
  docSize = atoi(strtok(NULL, "-"));
  seed = atoi(strtok(NULL, "."));
  dictionary = new char*[dictSize + 3];
  dictFilePtr = readDictionary(argv[1], dictionary, dictSize);
  document = new char*[docSize + 3];
  docFilePtr = readDocument(document, dictSize, docSize, seed);
  wrongs = new int[docSize];
  readWrongs(wrongs, dictSize, docSize, seed, wrongCount);
  misspelled = new int[docSize];
  CPUTimer ct;
  maxRAM = currentRAM = 0;
  Speller *speller = new Speller(dictionary, dictSize);
  tempMaxRAM = maxRAM;
  tempCurrentRAM = currentRAM;
  delete [] dictFilePtr;
  maxRAM = tempMaxRAM;
  currentRAM = tempCurrentRAM;
  speller->check(document, docSize, misspelled, &misspelledCount);
  cout << "CPU Time: " << ct.cur_CPUTime() << " Real RAM: " << maxRAM << endl;
  checkAnswers(wrongs, wrongCount, misspelled, misspelledCount);
  cleanup(dictionary, docFilePtr, document, wrongs, misspelled);
  delete speller;
  return 0;
}  // main()ZZ
[ssdavis@lect1 p4]$ CreateDocs.out                   [ssdavis@lect1 p4]$ cat Doc-3-4-1.txt
Dictionary size: 3                                   lsrwwvov
Document size: 4                                     vwuboxynjm
Seed: 1                                              luov
[ssdavis@lect1 p4]$ cat Dict-3-4-1.txt               vwucoxynjm
luov                                                 [ssdavis@lect1 p4]
vwucoxynjm                                           [ssdavis@lect1 p4]$ cat Wrongs-3-4-1.txt
lsrwwvov                                             1
[ssdavis@lect1 p4]                                   [ssdavis@lect1 p4]$
```

```
[ssdavis@lect1 p4]$ speller.out Dict-100000-2000000-7.txt
CPU Time: 0.170697 Real RAM: 1230872
[ssdavis@lect1 p4]$
```