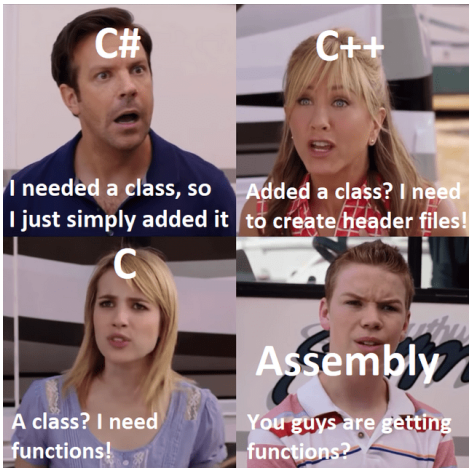
Take Home Programming Test

CSEE 3827: Fundamentals of Computer Systems

Martha Barker
Martha Kim

Deadline: Nov 19 11:59pm



1 Questions

During the test, you may pose questions about this prompt or any policies via private post on ED. As on an in-person test, staff will answer clarifying questions about the prompt, but will not provide assistance writing or debugging code.

2 Collaboration and Permissible Resources

- All work is to be individual, with no collaboration of any sort permitted.
- You may refer to all of this semester's course materials (i.e., slides, notes, problem sets and solutions, ED history).
- You may also refer to external resources (e.g., slides from other courses or semesters, the Harris and Harris textbook, etc.). Such materials should serve as references and their content should never be presented as your own.
- You may not solicit or consult solutions written by another human.
- AI-based coding assistants such as ChatGPT or Github Copilot are prohibited.
- Any evidence of academic integrity violations will be reported directly to the Office of Student Conduct.

3 Submission and Testing

- To submit, upload a single file named `bow.s` to gradescope.
- Prior to submission, gradescope will check that this file is named properly and that all the separators appear correct.
- The submission deadline is Tuesday November 19 at 11:59pm. Be sure to leave sufficient time to upload before that deadline, as late submissions will not be accepted.
- Each function is allocated a certain number of correctness points and convention points.
- After submission, each function will be tested for functionality, i.e., does it produce the correct results on the various test cases provided in the scaffolding. These points are not all or nothing and we will do our best to award partial credit as appropriate.
- In order to test calling conventions, functions need to work correctly, so functions that do not pass the functionality checks will not receive convention points.
- Functions that work correctly are eligible for the convention tests, and will be awarded convention points based on how they perform on those tests. As on the functionality tests, we will award partial credit when appropriate.

4 Bag of Words (BoW) Model

For this assignment, you will implement four helper functions for a MIPS implementation of the ”bag of words” model of text. The Bag of Words (BoW) model represents a large corpus of text as an unordered ”bag of words”. Each word in the text appears in the bag, along with a count of how many times the word appears in the text. For example, the BoW model for the sentence ”Ask not what your country can do for you — ask what you can do for your country” would be `ask:2, not:1, what:2, your:2, country:2, can:2, do:2, for:2, you:2` . This representation captures some features of the text, namely the set of words and their frequency, while disregarding others such as the word order and grammatical structure. While simplistic, it is useful in some basic natural language processing tasks such as document classification.

5 Functions to Implement

In this assignment, text is represented as a standard null-terminated ASCII string. Within the string, a word is defined as one or more letter characters (i.e., A-Z or a-z) bounded by any non-letter characters (i.e., anything other than A-Z or a-z, to include whitespace and punctuation). [Update 11/16 7a: Typo fixed to correctly list lowercase letters a-z.]

There are four function to implement. The first two functions do not depend on any other functions, while the last two will use the first two as helpers. You may work on them in any order you wish, but we recommend getting the helper functions working correctly before working on functions that use them.

Each of the four functions is delineated in the scaffolding by separator lines that begin

```
#### Do not move this separator.
```

Do not change any of these separator lines in your submission, and make sure that the full body of each function, and only that function, appears between the appropriate separators.

Each of the four functions is worth 25 points (20 for functionality and 5 for adherence to convention).

5.1 `isletter`

The `isletter` function takes an ASCII character in `$a0`. It should and return a 1 in `$v0` if the argument is an upper- or lower-case character (i.e., A-Z or a-z) and a 0 otherwise. Your implementation of `isletter` should not call any other functions.

5.2 `lettersmatch`

The `lettersmatch` function takes two ASCII letters in `$a0` and `$a1`. It should return a 1 in `$v0` if the letters match, regardless of case. It should return a 0 otherwise. You may assume that `lettersmatch` is called only on letters A-Z or a-z and do not need to handle any non-letter ASCII inputs. Your implementation of `lettersmatch` should not call any other functions.

5.3 `nextword`

The `nextword` function takes a pointer to a string in `$a0`. In `$v0` it should return a pointer to the first character of the next word in that string. Recall that words are defined as a sequence of one or more letters, and end with any non-letter character, not just spaces. If there is no next word in the string, it

should return 0. You may assume that the function is always called on a valid, non-null pointer. You should call `isletter` as needed.

5.4 `wordsmatch`

`wordsmatch` takes pointers to words in `$a0` and `$a1`. In `$v0` it should return 1 if the words match, and 0 otherwise. Two words match if each of their letters match, according to `lettersmatch`. You may assume that the argument pointers always point to letters, and there is thus no need to handle the cases where the pointers are null or point to a non-letter character. If either of the arguments points to the middle of the word, check remainder of that word as if it were the entire word. Your code should call `lettersmatch` and `isletter` as appropriate to implement this function.

6 Testing and Tips

- As in PS4 and PS5, the provided main function makes test calls to the functions to be implemented. If you are having trouble isolating an error, it helps to comment out or excise all but the one test invocation you are trying to debug.
- Do not implement any additional functions, beyond the four described above.
- Because all four of these functions will share a file, all of the labels will need to be unique. A simple technique is to prepend the function name on each label in the function. For example, if you want to have a `looptop` label for two functions `A` and `B`, label the returns `A_looptop` and `B_looptop` respectively.

  [Update 11/15 6pm: The first version of this doc had a bullet point talking about a function that is not part of this assignment. It was an error and has been removed.]