

433-141 Computing Fundamentals A – Semester 1, 2001
Project: Spell Checker
Due date: 5pm, 14 May 2001

Project Description

The aim of this project is to build a simple spell checker program.

You are provided with two Haskell scripts: `SpellChecker.hs` and `SPCLib.hs`. `SPCLib.hs` contains some predefined type, data and function definitions. **Do not make any changes to this file.** `SpellChecker.hs` contains some function specifications, including a first (very primitive) spell checking function, `doSpellCheck`. To make it more useful, you are asked to develop some additional functions that add various kinds of other features. For the purpose of this project **you only need to edit and submit the file `SpellChecker.hs`**. Submission information is provided at the end of this document.

Type definitions

The following type and data definitions are provided in `SPCLib.hs`. (**Do not make any changes to this file.**)

A `Word` is a list of characters not separated by spaces and only consisting of letters and numbers, and a `Dictionary` is a list of words:

```
type Word = String
type Dictionary = [Word]
```

Type `SpellCheck` describes a function which takes a `Dictionary` and a `String` and checks whether the words in the string are valid. The function uses `putStrLn` to directly report back to the screen any words that are invalid with respect to the dictionary:

```
type SpellCheck = Dictionary -> String -> IO ()
```

Internally, it manipulates a `CorrectionList` that, for each misspelt word, notes the line number that it appears on, and offers some alternative spellings. In the early stages, before we figure out how to suggest alternative spellings, the list of suggestions is just the `noAnswers` list. Function `presentNicely` takes a `CorrectionList` and formats it neatly for output to the screen, and function `prepareWords` breaks the input string up into a list of words and attaches a line number to each word. You do not need to change either of these two functions, but should note that `prepareWords` takes as its first argument a function that tells it how to construct words from a string. The behaviour of `prepareWords` can be modified by passing a different argument function, and is the subject of Task 3 below.

Before proceeding, you will need to read the code in both of these files (and the whole of this handout) to see how the various parts operate, and what is expected of you. The two Hugs scripts should be copied from <http://www.cs.mu.oz.au/141/proj/>.

Task 1: Testing against the dictionary (1 mark)

`SpellChecker.hs` includes three simple test functions, `test1`, `test2`, and `test3`, which try respectively to spell check the three strings `mary1`, `mary2`, and `mary3`. Start Hugs and look

at the three test strings, the sample dictionary `smallDict`, and the output of the three test functions.

In this task, you simply have to modify one line of `doSpellCheck` so that only words that are not in the dictionary are reported as being possible errors (instead of every word!) The output from `test1` for your revised `doSpellCheck` function should then be:

```
on line 1 word "Mary" might in fact be one of:
    sorry, no suggestions can be made
on line 2 word "Mary" might in fact be one of:
    sorry, no suggestions can be made
on line 3 word "it's" might in fact be one of:
    sorry, no suggestions can be made
```

Task 2: Casefolding before testing (2 marks)

Look again at the output you got in Task 1 – notice how "Mary" is reported as an error, even though the word "mary" is in the dictionary. Further modify the `doSpellCheck` function so that before each word from the `WordList` is tested against the dictionary, it is converted to all lower-case letters. Hint – you may find the prelude function `toLower` of some use. Now the output for `test1` should be:

```
on line 3 word "it's" might in fact be one of:
    sorry, no suggestions can be made
```

Looks good, until you try `test2`, and find that "contrary," and "grow?" are reported as errors (amongst others) – somehow the punctuation is getting muddled up with the words, and so the test against the dictionary is still failing.

Task 3: Parsing into words (4 marks)

This unfortunate behaviour is a consequence of the use of the prelude function `words` to split up each line. It cuts on whitespace characters rather than non-alphanumeric characters.

Write a new function `getWords :: String -> [Word]` that correctly cuts up a string into alphanumeric words. Modify `doSpellCheck` so that your new function `getWords` is used (passed in the call to `prepareWords`) rather than `words`. On `test2` you should now be getting:

```
on line 1 word "Mry" might in fact be one of:
    sorry, no suggestions can be made
on line 4 word "bells" might in fact be one of:
    sorry, no suggestions can be made
on line 4 word "cocle" might in fact be one of:
    sorry, no suggestions can be made
on line 4 word "shells" might in fact be one of:
    sorry, no suggestions can be made
on line 5 word "maids" might in fact be one of:
    sorry, no suggestions can be made
on line 5 word "alll" might in fact be one of:
    sorry, no suggestions can be made
```

Task 4: Plurals (4 marks)

The next problem to tackle is that of plurals, such as "maids", which we would like to match against the dictionary word "maid". Write a function `makePlural :: Word -> Word` that converts a word to its plural. This is actually a very complex task, and for our purposes we will make use of the list of rules below, to be applied in the order listed. (Yes, I know that this list is not complete! Think about "axis" and "fungus" and "house" and "mouse", and then agree with me that we are not going to write a comprehensive function for this task.)

```
—lf  →  —lves
—y   →  —ies
—x   →  —xes
—s   →  —ses
—    →  —s
```

Then incorporate your function into `doSpellCheck` so that you obtain this output for `test2`:

```
on line 1 word "Mry" might in fact be one of:
    sorry, no suggestions can be made
on line 4 word "cocle" might in fact be one of:
    sorry, no suggestions can be made
on line 5 word "alll" might in fact be one of:
    sorry, no suggestions can be made
```

To get full marks for this step your solution must be extensible: to add a new suffix rule, it should be simply a matter of adding a pair (such as ("y", "ies"), for example) in the correct spot in a list of suffix-replacement pairs.

Task 5: Offering suggestions, #1 (3 marks)

So far `doSpellCheck` does not make any suggestions as to how to correct the errors it identifies. Write a function `getLongPrefixes :: Dictionary -> Word -> [Word]` that, for a given dictionary, and a single word, returns a list of all of the words in the dictionary that have the same set of initial characters as the word. For example, `getLongPrefixes smallDict "mry"` should return `["maid", "marigold", "marry", "mary"]` since all of those words have one character in common with "mry" and there are no words in `smallDict` that have two letters in common; and `getLongPrefixes smallDict "cocle"` should return `["cockle"]`, since it is the only word in the dictionary that matches "cocle" for three characters.

Replace the use of `noAnswers` in `doSpellCheck` by a call to `getLongPrefixes` to make a fifth version of your spell checking function.

Task 6: Offering suggestions, #2 (1 marks)

This task is described at <http://www.cs.mu.oz.au/141/proj/>. It involves quite a lot of work for one lousy little mark, and you should only commence this task if you have completed all other aspects of the project and still have time on your hands.

Assessment

Mark allocations for each task are shown. The maximum possible mark overall is 15. Note that **if you do not use the correct type, data and function names**, as described above, automatic testing will fail and **you will be penalised**. Items taken into account in assessing submissions include: correctness of output; quality of the definitions and coding; and readability of your script, including documentation, consistent indenting, appropriate selection of names, and line-lengths limited to 80 characters. Bonus marks, if awarded, may compensate for deficiencies elsewhere but the final mark cannot exceed 15. Remember that the script should include a header comment that gives your name and enrolment number.

The project is to be done on an *individual* basis, and the submitted work is to be your own work. Soliciting help from newsgroups and the Internet in general will be treated as cheating, as will any other attempts to obtain help. You may, of course, discuss your project in general terms with tutors and the lecturing staff, but must refrain from exchanging notes, functions, files, or other program material with other students (and non-students). Students found cheating, or copying, or allowing copying to occur, will be subject to disciplinary action. Note also that sophisticated similarity checking software will be run over all pairs of student submissions. As a result of our use of this software over the last few years, a large number of students have been punished under the discipline regulations. Punishment is also extended to those who supply projects to other students. Students should review the “Principles of Responsible Student Behaviour” in *Student Manual Volume A*.

Submission

Your edited version of the file `SpellChecker.hs` file is to be submitted electronically from your 141 computer account by the due date and time, by issuing the command:

```
submit 141 A SpellChecker.hs
```

See the notes in *Student Manual Volume A* for more information about this command, and **be sure that you verify your submission** using the commands

```
verify 141 A > proj-verify.txt  
more proj-verify.txt
```

It is your responsibility to ensure (using `verify`) that your project has been successfully submitted. Mark penalties will be incurred for submissions received after the due date and time; no projects will be accepted more than one week late. To submit the project after the due date (5pm May 14) you should issue the command:

```
submit 141 A.late SpellChecker.hs
```

Note that you are expected to end up with multiple versions of function `doSpellCheck`. To give the markers maximal flexibility in assigning marks, you should make a fresh copy of your function before you start each stage, and comment out (using `{-` and `-}` pairs) the earlier version. That is, your submitted file should have multiple versions of `doSpellCheck` in it, only one of which is activated.

Final question

Do you think that this document was spellchecked?