The University of Queensland
School of Information Technology and Electrical Engineering

CSSE2310/CSSE7231 — Semester 1, 2021
Assignment 1 (v1.1 - 8 March 2021)

Marks: 75 (for CSSE2310), 85 (for CSSE7231)
Weighting: 10%
Due: 3:59pm 19 March, 2021

Specification changes since version 1.0 are shown in red and are summarised at the end of the document.

Introduction

The goal of this assignment is to give you practice at C programming. You will be building on this ability in the remainder of the course (and subsequent programming assignments will be more difficult than this one). You are to create a program (called `search`) which searches a dictionary of words to find words that match a certain pattern of letters and blanks. For example, you might use such a program to find all 5 letter words that begin with 'a' and end with 't'. The assignment will also test your ability to code to a particular programming style guide.

Student conduct

This is an individual assignment. You should feel free to discuss **general** aspects of C programming and the assignment specification with fellow students, including on the discussion forum. In general, questions like "How should the program behave if ⟨this happens⟩?" would be safe, if they are seeking clarification on the specification.

You must not actively help (or seek help from) other students or other people with the actual design and coding of your assignment solution. It is **cheating to look at another student's assignment code** and it is **cheating to allow your code to be seen or shared in printed or electronic form by others**. All submitted code will be subject to automated checks for plagiarism and collusion. If we detect plagiarism or collusion, formal misconduct proceedings will be initiated against you, and those you cheated with. That's right, if you share your code with a friend, even inadvertently, then **both of you are in trouble**. Do not post your code to a public place such as the course discussion forum or a public code repository, and do not allow others to access your computer - you must keep your code secure.

Uploading or otherwise providing the assignment specification to a third party including online tutorial and contract cheating websites is considered misconduct. The university is aware of these sites and they cooperate with us in misconduct investigations.

You may use code provided to you by the CSSE2310/CSSE7231 teaching staff **in this current semester** and you may use code examples that are found in man pages on moss. If you do so, you **must** add a comment in your code (adjacent to that code) that references the source of the code. If you use code from other sources then this is either misconduct (if you don't reference the code) or code without academic merit (if you do reference the code). Code without academic merit will be removed from your assignment prior to marking (which may cause compilation to fail) but this will not be considered misconduct.

The course coordinator reserves the right to conduct interviews with students about their submissions, for the purposes of establishing genuine authorship. If you write your own code, you have nothing to fear from this process.

In short - **Don't risk it!** If you're having trouble, seek help early from a member of the teaching staff. Don't be tempted to copy another student's code or to use an online cheating service. You should read and understand the statements on student misconduct in the course profile and on the school web-site: `https://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`

Specification

Command Line Arguments

Your program (`search`) is to accept command line arguments as follows:

```
./search [-exact|-prefix|-anywhere] [-sort] pattern [filename]
```

In other words, your program should accept between one and four command arguments - zero, one or two option arguments followed by a pattern, then optionally followed by a dictionary filename. The option argument that specifies the type of search is ONE of `-exact`, `-prefix` or `-anywhere`. If this argument is not given, the default search type is `-exact`. Search types are explained below. If the `-sort` argument is present, then the output will be sorted alphabetically (also explained below). The `-sort` option may come before the search type option if both are present. The only argument which must always be present is the pattern itself. The pattern consists only of question mark characters and/or letters (upper or lower case) and may be empty. A question mark indicates a match with any letter. Some example patterns are:

```
abc
a?d
A???c?e
?b?D?
??
```

The last argument (if present) is the path name of the dictionary file, i.e. the name of a file containing words – one per line. Each line (including the last) is assumed to be terminated by a newline character (\n) only. You may assume there are no blank lines and that no words are longer than 40 characters. The filename can be relative to the current directory or absolute (i.e. begins with a /). Some examples are:

`/usr/share/dict/words`
`../dictname`
`mydictionary` (refers to a file with this name in the current directory)

If the filename is not specified, the default should be used (`/usr/share/dict/words`).

Program Operation

The program is to operate as follows

- If the program receives an invalid command line, e.g. (but not limited to) an invalid number of command line arguments, or an option argument begins with '–' and it is not one of `-exact`, `-prefix`, `-anywhere` or `-sort`, or if more than one search type argument is present, then your program should print the message:
  `Usage: search [-exact|-prefix|-anywhere] [-sort] pattern [filename]`
  to standard error, and exit with a non-zero exit status.

- If the given dictionary filename does not exist or can not be opened for reading, your program should print the message:
  `search: file "`*`filename`*`" can not be opened`
  to standard error, and exit with a non-zero exit status. (The italicised *filename* is replaced by the actual filename i.e. the argument given on the command line. The double quotes must be present.)

- If the given pattern contains characters other than question marks (?) and letters (either case), then your program should print the message:
  `search: pattern should only contain question marks and letters`
  to standard error, and exit with a non-zero exit status.

- Your program should then print out all lines in the given file that match the given pattern (if any). Lines are printed to standard output. The details of the pattern matching are described below. When all matches have been printed, your program should exit. If at least one match was printed, your program should exit with an exit status of zero. If no matches were reported, your program should exit with a non-zero exit status.

- **Advanced Functionality:** If the `-sort` argument is present, then the output lines must be sorted alphabetically - i.e. in the same order as a standard dictionary, with the case of each letter being ignored for sorting purposes (but the case of the words in the dictionary must be preserved when output). If two matching words are identical for sorting purposes, e.g. *Miss* and *miss*, then they may be output in either order.
  Without the `-sort` argument, then the output lines must be in the same order as they were in the dictionary file. If you do not implement this advanced functionality, then your program should just ignore the `-sort` argument and output lines in the same order as they were in the dictionary file. (i.e. **do not throw an error** if `-sort` is specified but not implemented by your program).

Pattern Matching – Types of Search

As indicated above, there are three types of search (exact, prefix and anywhere). An **exact** match (the default type) means that the pattern should match a word exactly - matched words will be the same length as the pattern and each character in the pattern matches the corresponding character in the word (upper or lower case). A question mark in the pattern matches any letter in the word (but not numbers or punctuation characters). For example, consider a dictionary file containing the following lines:

```
3rd
Arctic
ardency
ardent
arduous
are
area
arena
aren't
```

Listing 1: Sample dictionary file

With **exact** matching, the pattern '`?r?`' will match '`are`' only (but not '`3rd`'). The pattern '`ar????`' will match '`Arctic`' and '`ardent`' but not '`aren't`'.

(Words that contain nonalphabetic characters such as numbers, hyphens or apostrophes are never reported as matches for any of the search types.)

With **prefix** matching, the pattern matches the beginning part of the word – i.e. any number of letters (including zero) may follow the matching characters. For a file containing the lines given above, the prefix pattern '`are`' will match '`are`', '`area`' and '`arena`'. The prefix pattern '`?????`' will match all words which are five or more letters long (and contain only letters).

With **anywhere** matching, the pattern matches any part of the word, i.e. any number of letters (including zero) may precede the matching characters and any number of letters (including zero) may follow the matching characters. For a file containing the lines given above, the anywhere pattern '`en`' will match '`ardency`', '`ardent`' and '`arena`'. The anywhere pattern '`re?`' will match '`area`' and '`arena`'. The anywhere pattern '`T`' will match '`Arctic`' and '`ardent`'.

An empty pattern will never be an **exact** match for any words, and will match every valid word for **prefix** and **anywhere** matching.

Other functionality

Your program must print nothing to standard output other than matching words. Your program must print nothing to standard error other than the exact messages given above. (Your program will never print a message to both standard output and standard error during the same run.)

Style

You must follow version 2.0.4 of the CSSE2310/CSSE7231 C programming style guide available on the course BlackBoard site.

Hints

1. You **may** wish to consider the use of the standard library functions `isalpha()`, `islower()`, `isupper()`, `toupper()` and/or `tolower()`. Note that these functions operate on integers (ASCII values) so you will need to cast characters to integers and vice versa as appropriate to avoid compiler warnings. For example, given that variable c is of type `char`, the following code will convert the character stored in c to upper case (without compiler warnings):
   `c = (char)toupper((int)c);`
2. Note that implementation of the advanced functionality (sorting) will require the use of dynamically allocated memory (i.e. with `malloc()` etc.) to store the list of words for sorting. Use of dynamically allocated memory is not required if you do not implement the advanced functionality (though you may use it if you wish).

3. Some functions which **may** be useful include `strcmp()`, `strlen()`, `exit()`, `fopen()`, `fgets()`, `fprintf()` and `qsort()`.

Forbidden Functions

You must not use any of the following C functions/statements. If you do so, you will get zero (0) marks for the assignment.

- `goto`~~`()`~~
- `longjmp()` and equivalent functions
- `system()`
- `execl()` or any other members of the exec family of functions
- POSIX regex functions

Submission

Your submission must include all source and any other required files (in particular you must submit a `Makefile`). Do not submit compiled files (eg `.o`, compiled programs) or dictionary files.

Your program must build on `moss.labs.eait.uq.edu.au` with:
`make`

Your program must be compiled with gcc with at least the following switches:
`-pedantic -Wall --std=gnu99`

You are not permitted to disable warnings or use pragmas to hide them.

If any errors result from the `make` command (i.e. the `search` executable can not be created) then you will receive 0 marks for functionality (see below). Any code without academic merit will be removed from your program before compilation is attempted (and if compilation fails, you will receive 0 marks for functionality).

Your program must not invoke other programs or use non-standard headers/libraries.

Your assignment submission must be committed to your subversion repository under
`https://source.eait.uq.edu.au/svn/csse2310-sXXXXXXX/trunk/ass1`
where sXXXXXXX is your moss/UQ login ID. Only files at this top level will be marked so **do not put source files in subdirectories**. You may create subdirectories for other purposes (e.g. your own test files) but these will not be considered in marking - they will not be checked out of your repository.

The initial structure of the repository will be created for you. Your latest submission will be marked and your submission time will be considered to be the commit time of your latest submission. **If you commit after the assignment deadline then we will mark that latest version and a late penalty will apply, even if you had made a submission (commit) before the deadline.**

You must ensure that all files needed to compile and use your assignment (including a Makefile) are committed and within the `trunk/ass1` directory in your repository (and not within a subdirectory) and not just sitting in your working directory. Do not commit compiled files or binaries. You are strongly encouraged to check out a clean copy for testing purposes.

The late submission policy in the CSSE2310/CSSE7231 course profile applies. Be familiar with it.

Marks

Marks will be awarded for functionality and style and documentation.

Functionality (60 marks)

Provided your code compiles (see above), you will earn functionality marks based on the number of features your program correctly implements, as outlined below. Partial marks will be awarded for partially meeting the functionality requirements. Not all features are of equal difficulty. **If your program does not allow a feature to be tested then you will receive 0 marks for that feature**, even if you claim to have implemented it.

For example, if your program can never open a file, we can not determine if your program can match patterns correctly. If your program takes longer than 10 seconds to run any test, then it will be terminated and you will earn no marks for the functionality associated with that test. The markers will make no alterations to your code (other than to remove code without academic merit).

Marks will be assigned in the following categories.

1. Program correctly handles invalid command lines (6 marks)
2. Program correctly handles dictionary files that are unable to be read (3 marks)
3. Program correctly handles invalid patterns (3 marks)
4. Program correctly performs exact pattern matching (without sorting) (12 marks)
5. Program correctly performs prefix pattern matching (without sorting) (12 marks)
6. Program correctly performs anywhere pattern matching (without sorting) (14 marks)
7. **Advanced Functionality:** Program correctly matches patterns (of all types) with sorting (10 marks)

Style (10 marks)

Style marks will be calculated as follows: Let

- W be the number of distinct compilation warnings recorded when your code is built (using the correct compiler arguments)
- A be the number of style violations detected by `style.sh` (automatic style violations)
- H be the number of **additional** style violations detected by human markers. Violations will not be counted twice

Your style mark S will be

$$S = 10 - (W + A + H)$$

If $W + A + H \geq 10$ then S will be zero (0) - no negative marks will be awarded. H will not be calculated (i.e. there will be no human style marking) if $W + A \geq 10$

The number of style guide violations refers to the number of violations of version 2.0.4 of the CSSE2310/CSSE7231 C Programming Style Guide.

A maximum of 5 violations will be penalised for each broad guideline area. The broad guideline areas are Naming, Comments, Braces, Whitespace, Indentation, Line Length and Overall. For naming violations, the penalty will be one violation per offending name (not per use of the name).

You should pay particular attention to commenting so that others can understand your code. The marker's decision with respect to commenting violations is final - it is the marker who has to understand your code. To satisfy layout related guidelines, you may wish to consider the `indent(1)` tool. Your style mark can never be more than your functionality mark - this prevents the submission of well styled programs which don't meet at least a minimum level of required functionality.

You are encouraged to use the `style.sh` tool installed on `moss` to style check your code before submission, however the marker has ultimate discretion – the tool is only a guide.

SVN commit history assessment (5 marks)

Markers will review your SVN commit history for your assignment from the time of handout up to your submission time.

This element will be graded according to the following principles:

- Appropriate use and frequency of commits (e.g. a single monolithic commit of your entire assignment will yield a score of zero for this section)
- Appropriate use of log messages to capture the changes represented by each commit

Again, don't overthink this. We understand that you are just getting to know Subversion, and you won't be penalised for a few "test commit" type messages. However, the markers must get a sense from your commit logs that you are practising and developing sound software engineering practices by documenting your changes as you go. In general, tiny changes deserve small comments - larger changes deserve more detailed commentary.

Documentation (10 marks) - for CSSE7231 students only

Please refer to the grading critera available on BlackBoard under "Assessment" for a detailed breakdown of how these submissions will be marked.

CSSE7231 students must submit a PDF document containing a written overview of the architecture and design of your program.
This document should describe, at a general level, the functional decomposition of the program, the key design decisions you made and why you made them.

- Submitted via Blackboard/TurnItIn prior to the due date/time
- Maximum 2 A4 pages in 12 point font
- Diagrams are permitted up to 25% of the page area. The diagram must be discussed in the text, it is not ok to just include a figure without explanatory discussion.

Don't overthink this! The purpose is to demonstrate that you can communicate important design decisions, and write in a meaningful way about your code. To be clear, this document is not a restatement of the program specification - it is a discussion of your design and your code.

If your documentation obviously does not match your code, you will get zero for this component, and will be asked to explain why.

Total mark

Let

- F be the functionality mark for your assignment.
- S be the style mark for your assignment.
- V be the SVN commit history mark.
- D be the documentation mark for your assignment (for CSSE7231 students).

Your total mark for the assignment will be:

$$M = F + \min\{F, S\} + \min\{F, V\} + \min\{F, D\}$$

out of 75 (for CSSE2310 students) or 85 (for CSSE7231 students).
In other words, you can't get more marks for style or SVN history or documentation than you do for functionality. Pretty code that doesn't work will not be rewarded!

Late Penalties

Late penalties will apply as outlined in the course profile.

Specification Updates

Any errors or omissions discovered in the assignment specification will be added here, and new versions released with adequate time for students to respond prior to due date. Potential specification errors or omissions can be discussed on the discussion forum or emailed to `csse2310@helpdesk.eait.uq.edu.au`.

Version 1.1 - 8 March 2021

1. Clarified that if the dictionary file does not exist then it is the complete filename argument (which may include a path) which is printed as part of the error message.
2. Updated the list of forbidden functions to make it clear that POSIX regex functions can not be used (as was stated already on the Assessment page in Blackboard).
3. Updated the description of invalid command lines to make it clear that the list given is an example of invalid command lines but not a complete list – and to make it clear that the filename argument might begin with '–'.

4. Specified that any C source files and the Makefile necessary to build your `search` program must be at the top level in your assignment one repository (i.e. in `trunk/ass1`) and not in a subdirectory. Any subdirectories in your assignment one submission will be ignored for marking purposes (but you may use them for your own purposes, e.g. test files).

5. Specified that the limit on any test run is 10 seconds. You will earn no marks for the functionality associated with that particular test if your program takes longer than 10 seconds to complete the test.

6. Clarified that the pattern command line argument may be empty and that an empty pattern will never be an exact match for any words, and will match every valid word for prefix and anywhere matching.