Unit home
Final Exam info
Lecture & Workshop recordings on LMS
2nd Project
Sample solution
help2002
Schedule
C textbooks
OS textbooks
Information resources
Extra reading
Past projects
Recent feedback
Working effectively

CITS2002 Systems Programming - Project 2 2022

see also: Project 2 clarifications Sample solution

Background

In recent years, computer disk sizes and densities have increased dramatically, with costs dropping to 4c/gigabyte (HDD) and 12c/gigabyte (SSD). As a consequence, we store an enormous number of files on our computers' disks, and finding required information is increasingly difficult. Historically, the names of files and directories alone reminded us of the location of information; today we require software tools to find information based on the *content* of files. In addition to file-system metadata storing information *about* our files and directories, we require an index of the *content* of our files.

As examples, the macOS operating system provides the Spotlight utility to index the contents of files, and there are a number of similar utilities developed for Linux.

> The **goal** of this project is to design and develop a command-line utility program, named ***trove***, to build and search a compressed index of the contents of files.
>
> Successful completion of the project will develop and enhance your understanding of advanced features of the C11 programming language, core Linux and macOS system-calls, POSIX function calls, and the use of the *make* utility.

(Some of) the contents of all required files are to be stored in a *trove-file*. A trove-file will contain the *absolute pathnames* of the files that have been indexed, and each of the 'words' found in those files.

The *words* indexed by *trove*, are minimum-length sequences of *alphanumeric* characters. Some example words are: *main*, *Australia*, and *GTHO351*. The character sequence *COVID-19* contains 2 words, but the second word might not be stored if indexed words must be at least 3 characters long.

Naturally, if many files are indexed, or if many words are found, the trove-file has the potential to become very large. Thus, the trove-file will be *compressed* when stored on disk.

Program invocation

From the command-line, the *trove* utility may be invoked in one of two ways, with command-line options indicating the actions to be performed.

1. **`prompt.1>`** ./trove [-f trovefile] *word*

   The first invocation requests *trove* to list the *absolute pathname* of files containing the indicated word. Filenames are simply listed one-per-line (their order is not important).

   If any file does not exist at the time of the search (it may have been deleted since the trove-file was built), then the file is *not* listed.

   If any files containing the requested word are found, then *trove* will exit with success; otherwise with failure.

2. **`prompt.2>`** ./trove [-f trovefile] [-b | -r | -u] [-l length] *filelist*

   The second invocation requests *trove* to build a new index, to remove information from an existing index, **or** to update an existing index.

   A *filelist* is a sequence of one-or-more filenames or directory names. The contents of each named file should be indexed, and each named directory (and its subdirectories) should be traversed to locate files to be indexed.

   Note that the files do not have to be text-files - they could equally be, for example, executable programs or object files.

   If the requested action can be performed successfully, then *trove* will exit with success; otherwise with failure.

The following command-line options are to be supported:

-b	build a new trove-file from the contents of a *filelist*. The new trove-file will replace an existing trove-file of the same name.
-f *trovefile*	provide the name of the trove-file to be built or searched. If the -f option is not provided, the

default name of the trove-file is **/tmp/trove**

-l *length* only words of at least the indicated length should be added to the trove-file. *length* must be a positive integer, for example: *-l 6*. If the -l option is not provided, the default length is **4**.

-r if any of the files from the *filelist* appear in the trove-file, remove all of their information from the trove-file.

-u update the trove-file with the contents of all files in the *filelist*. If the information from any of the files already exists in the trove-file, then that (old) information is first removed.

The *trove-file* format

While there is no required format for the trove-file, your design should consider some important factors:

- the filenames stored in the trove-file should be *absolute pathnames* - see *man realpath*
- to reduce the size of the trove-file (and, thus, the time to read it), each indexed word and each absolute filename should (ideally) be stored only *once*.
- the trove-file *may*, but *does not* have to be, a text-file.

Getting started

There is no required sequence of steps to undertake the project, and no sequence of steps will guarantee success. However, the following sequence is strongly recommended (and this is how the sample solution was built). It is assumed (considered essential for success!) that each step:

- is extensively tested before you proceed to the next step, and
- if any errors are found, the first error should be reported to *stderr*, and the program should immediately terminate.

The recommended initial steps:

0. **find a project partner**.
1. determine what activities are to be performed by the program. Determine what data needs to be stored during the execution of the program. Design one of more data types and data structures to store this long-term data, and create a new *trove.h* header file to define these constructs. These definitions will certainly evolve as your project develops.
2. break up the activities into fairly independent sets. Write a number of "empty" C files, each of which #includes your *trove.h* header file. Ensure that your *main()* function is implemented in a file named *trove.c*
3. write a *Makefile* to compile and link your C files, each of which will be dependent on the header file.
4. write the *main()* function, whose primary task is simply to receive and validate the command-line options and arguments. Write a *usage()* function to report the program's synopsis, should command-line processing detect any errors.
5. determine if *trove* is required to build/modify a trove-file, or to search an existing trove-file.
6. if building/modifying a trove-file, (recursively) traverse the *filelist* to find all of the files to be indexed. At this stage (as a debugging aid) just print each filename as it is found.
7. open each file found, and find all 'words' of at least the required length. At this stage (as a debugging aid) just print each word as it is found. Remember that the files do not have to be text-files.
8. store each word and each filename in which it is found in your data-structure(s).
9. ....
10. initially, store your trove-file on disk without any compression.
11. learn how to compress files using the */usr/bin/gzip* utility, and how to read the contents of a compressed file (without decompressing it) using the */usr/bin/zcat* utility. Both utilities are available under macOS and Linux. When confident that your program works as required, add code to compress the file - see *man gzip*
12. ....

It is anticipated that a successful project will very likely use (most of) the following Linux system-calls, and standard C11 & POSIX functions:

getopt(), malloc(), realloc(), strdup(), free(); opendir(), readdir(), closedir(), stat(), perror(); realpath(); access(), open(), read(), write(), close(); fork(), execl(), dup2(), wait().

Project requirements and fine print

1. Your project **must** be developed in multiple C11 source files and (at least one) header file.
2. Your submission will be compiled with the C11 compiler arguments **-std=c11 -Wall -Werror**, and marks will not be awarded if your submission does not compile.
3. Compilation and linking of your project **must** be supported by a *Makefile*, employing approriate variable definitions and automatic variables.
4. The default target of your *Makefile* **must** be named *trove*, and invocation of the *make* command must produce an executable program named *trove*.
5. The default name of the trove-file **must** be **/tmp/trove**, and the default minimum length of each indexed word must be **4**
6. Each trove-file **must** be stored on, and read from, disk using the */usr/bin/gzip* and */usr/bin/zcat* utilities.
7. When performing a search, the names of files containing the required word should be listed one-per-line (commencing in the first column). You may leave 'debug printing' in your submitted program, but all debug printing should be indented with one-or-more space or tab characters.

Assessment

The project is due at **5pm Friday 21st October (end of week 12)**, and is worth **25% of your final mark** for CITS2002.

It will be marked out of 50. The project may be completed **individually or in teams of two** (but not teams of three). The choice of project partners is up to you - you will not be automatically assigned a project partner.

You are **strongly** advised to work with another student who is around the same level of understanding and motivation as yourself. This will enable you to discuss your initial design together, and to assist each other to develop and debug your joint solution. Work together - do not attempt to split the project into two equal parts, and then plan to meet near the deadline to join your parts together.

25 of the possible 50 marks will come from the correctness of your solution. The remaining 25 marks will come from your programming style, including your use of meaningful comments; well chosen identifier names; appropriate choice of basic data-structures, data-types and functions; and appropriate choice of control-flow constructs.

During the marking, attention will obviously be given to the correctness of your solution. However, a correct and efficient solution should not be considered as the perfect, nor necessarily desirable, form of solution. Preference will be given to well presented, well documented solutions that use the appropriate features of the language to complete tasks in an easy to understand and easy to follow manner. That is, do not expect to receive full marks for your project simply because it works correctly. Remember, a computer program should not only convey a message to the computer, but also to other human programmers.

Submission requirements

1. The deadline for the project is **5pm Friday 21st October (end of week 12)**.
2. **You must submit your project electronically using *cssubmit*.**

   You should submit ALL C11 source-code (*.c) and header (*.h) files and a *Makefile* that specify the steps required to compile and link your application. You **do not need to submit** any data files/directories or testing scripts that you used while developing and testing your project. You can submit multiple files in one submission by first archiving them with *zip* or *tar*.

   The *cssubmit* program will display a receipt of your submission. You should print and retain this receipt in case of any dispute. Note also that *cssubmit* does not archive submissions and will simply overwrite any previous submission with your latest submission.
3. **At least one of your submission's C11 source files** must contain the C11 block comment:

```
//  CITS2002 Project 2 2022
//  Student1:   STUDENT-NUMBER1   FAMILY-NAME   FIRST-NAME   CONTRIBUTION
//  Student2:   STUDENT-NUMBER2   FAMILY-NAME   FIRST-NAME   CONTRIBUTION
```

   The CONTRIBUTION is a percentage indicating each team member's contribution to the project.
   If working alone, the CONTRIBUTION will be 100.
   If working in a team (of 2), the CONTRIBUTION might be 50/50, 60/40, 30/70, ...
4. If working as a team, only one team member should make the team's submission.
5. Your submission will be examined, compiled, and run on a contemporary **Linux** system (such as Ubuntu). If your submission does not compile on Linux, an attempt will be made to compile and run it on **macOS**

(Monterey). Your submission should work as expected on *one* of those platforms.

6. This project is subject to UWA's Policy on Assessment - particularly §5.3 *Principles of submission and penalty for late submission*, and Policy on Academic Conduct. In accordance with these policies, you may *discuss* with other students the general principles required to understand this project, but the work you submit must be the result of your own team's efforts. All projects will be compared using software that detects significant similarities between source code files. Students suspected of plagiarism will be interviewed and will be required to demonstrate their full understanding of their project submission.

Clarifications

Please post requests for clarification about any aspect of the project to *help2002* so that all students may remain equally informed. If necessary, the main project page will be updated to clarify any English in the project's description. Make reference to the online copy, and not a (dated) paper copy.

Good luck!

Chris McDonald.

The University of Western Australia

Computer Science and Software Engineering

CRICOS Code: 00126G

Presented by Chris.McDonald@uwa.edu.au