CS 242: Lab 3 Problems: (make, gdb and sccs)

Note: The lines starting with a '$' symbol represents a bash command line.

Make

1. Install build-essential, which is a software suite of several essential utility software/tools, like – make and gdb.
   ```
   $ sudo apt-get update
   $ sudo apt install build-essential
   ```

2. Download the Quest game files that are used in the corresponding tutorial:
   ```
   $ wget https://github.com/sap01/2019-cs-242-make-tut/raw/master/Quest.tar.gz
   ```

   Extract directory 'Quest':
   ```
   $ tar –xvzf Quest.tar.gz
   ```

3. Study the source files (.c) and the header files (.h) to understand their 'dependency structure': who depends on whom. Then create a simple makefile named 'Makefile' following the instructions from the beginning of Section 2 till Section 2.3 of the GNU Make manual: https://www.gnu.org/software/make/manual/make.html

4. Modify your makefile by defining a variable (user-defined variable): See Section 2.4 of the GNU Make manual: https://www.gnu.org/software/make/manual/make.html . Note: There are three types of variables in makefiles: user-defined variables, special variables and automatic variables.

5. Modify your makefile by adding some special variables (variables used by implicit rules): See Section 10.1 to 10.3 of the GNU Make manual: https://www.gnu.org/software/make/manual/make.html

6. Modify your makefile by adding some automatic variables: See Section 10.5.3 of the GNU Make manual: https://www.gnu.org/software/make/manual/make.html

7. Modify your makefile by replacing all the rules that have '.o' files as their targets with a single pattern rule: See Sections 10.5.1 to 10.5.2 of the GNU Make manual: https://www.gnu.org/software/make/manual/make.html

8. Would not it be convenient if pre-requisites are automatically generated? See Section 4.14 of the GNU Make manual: https://www.gnu.org/software/make/manual/make.html to learn how that can be achieved.

9. Suppose, you have a variable in your makefile whose value is defined in another makefile. Learn how your makefile can read the value of that variable from the other makefile: See Section 3.3 of the GNU Make manual: https://www.gnu.org/software/make/manual/make.html

10. In-built functions are very useful while writing a makefile. Learn about them in Section 8 of the GNU Make manual: https://www.gnu.org/software/make/manual/make.html

11. Modify your makefile by adding a rule for cleaning your directory: See Section 2.7 of the GNU Make manual: https://www.gnu.org/software/make/manual/make.html . Please carefully understand the utilities and pitfalls of phony targets: See Section 4.6 of the GNU Make manual: https://www.gnu.org/software/make/manual/make.html

12. Modify your makefile by adding a default rule: See Section 10.6 of the GNU Make manual: https://www.gnu.org/software/make/manual/make.html

13. Understand how to name your makefile: See Section 3.2 of the GNU Make manual: https://www.gnu.org/software/make/manual/make.html

14. By default, running 'make' displays the commands being executed. To hide/silence them, use the '-s' flag:
    $ make -s

15. The default shell used by a makefile to run its recipes is '/bin/sh' i.e. the Bourne shell. Change it to '/bin/bash' (the Bourne Again shell) using MAKESHELL, a special variable:
    MAKESHELL = /bin/bash

16. Create a sub-directory named 'include' inside directory 'Quest'. Move the header files inside that sub-directory. Then use the 'vpath' directive to search for header files inside sub-directory 'include'. Follow: https://www.gnu.org/software/make/manual/make.html#Selective-Search

    Also understand how special variable 'VPATH' (different from the 'vpath' directive) to set multiple directories to search for all types of pre-requisite files (not only header type of files). Follow: https://www.gnu.org/software/make/manual/make.html#General-Search

17. Check out a real makefile at: https://github.com/vim/vim/blob/master/Makefile . It is the makefile of the vim text editor. Don't worry; no question will come regarding this file in the exams. Check out the makefiles of your favourite utility software/tools for fun!

GDB (The GNU Debugger)

1. Please follow the step-by-step intro at:
   https://www.geeksforgeeks.org/gdb-step-by-step-introduction/

2. Also study the quick guide to debugging with GDB at:
   https://www.tutorialspoint.com/gnu_debugger/gdb_quick_guide.htm . Please skip Section 'GDB - Installation'.
   Kindly pay close attention to Section 'GDB - Debugging Examples'.
   Understand the difference between the 'next' and 'step' commands.

3. There is a Graphical User Interface (GUI) for GDB. It is called ddd (Data Display Debugger). Install it using:

   $ sudo apt install ddd

   ## Once installed, press the windows button -> search for ddd -> click on ddd. Open any source file (.c) inside it and play around with its interface. No in-depth questions will be asked from ddd in the exams; just remember its name and what it does.

SCCS (The Source Code Control System)

1. Install sccs by using the following command:
   $ sudo apt install cssc
   ## CSSC (Compatibly Stupid Source Control) is a software suite that contains SCCS.

2. Open 'An Introduction to the Source Code Control System' by Eric Allman at: http://sccs.sourceforge.net/man/sccs.me.html . Read Sections 1 and 2 to learn the basic functions and terminology of sccs.
   However, the term 'ID keywords' is not properly explained in the aforementioned reference. To mend the gap, read Section 'Incorporating Version-Dependent Information by Using ID Keywords' at:
   https://docs.oracle.com/cd/E19504-01/802-5880/6i9k05dhp/index.html .

3. Enter the following commands to create a new source file (prog.c) and put it into SCCS format:
   $ mkdir sccs_test
   $ cd sccs_test

   ## Create a file named prog.c and
   ## add a line to it.
   $ echo 'This is the first line.' > prog.c

   $ mkdir SCCS

   ## The following command puts prog.c into SCCS format.
   ## It creates an s-file for prog.c, named s.prog.c, and save it inside the SCCS sub-directory.
   $ sccs create prog.c

4. Enter the following command to get the latest version of prog.c for compilation i.e. in a read-only mode. See Section 4 at: http://sccs.sourceforge.net/man/sccs.me.html .
   $ sccs get prog.c

5. Enter the following commands for getting a copy of the latest version of prog.c for editing.
   $ sccs edit prog.c
   ## you may also use an equivalent command which is

```
## $ sccs get -e prog.c

## Gather info on what is going on.
## The following command tells you whether any file is being edited and if so then who is
editing it.
$ sccs info
prog.c: being edited: 1.2 1.3 cse 19/08/26 17:12:45
## User ‘cse’ it editing prog.c since 19/08/26 17:12:45.
## The version number of the file is 1.2.1.3.

## Gather info on whether user named cse is editing any file.
$ sccs info -ucse
prog.c: being edited: 1.2 1.3 cse 19/08/26 17:12:45

## Append (i.e. ‘>>’) a new line to prog.c
$ echo 'This is the second line.' >> prog.c


## Print the changes you have made
$ sccs diffs prog.c

## Put your changes into SCCS/s.prog.c i.e. create a delta, thus creating a new version
of prog.c.
$ sccs delta prog.c
## This command will ask for a comment. Add a comment that describes the change you
have made. For example, ‘Added a second line’.
## See Sections 5.1, 5.2 and 5.3 at: http://sccs.sourceforge.net/man/sccs.me.html .
```

6. Make some changes but discard them i.e. revert to the last version.

```
$ sccs edit prog.c

## Make some changes
$ echo ‘This is the third line.’ >> prog.c

## Discard the changes i.e. revert to the last version
$ sccs unedit prog.c
```

7. Get a specific version of a source file.

```
## Get version 1.1 of prog.c for compilation
$ sccs get -r1.1 prog.c
## or
## Get a copy of version 1.1 of prog.c for editing
$ sccs edit -r1.1 prog.c
```

8. Print the delta comments i.e. the change history of a file.
   ```
   $ sccs prt prog.c
   ```

9. In case, you have missed something while writing a comment in a specific version, edit the comment.

   ```
   ## Edit the delta comment of version 1.2 of prog.c.
   ## This command will prompt for entering the new comment.
   $ sccs cdc -r1.2 prog.c
   ```

10. Exclude a particular delta i.e. changes made in a particular version.

    ```
    ## Exclude delta 1.2
    $ sccs edit -x1.2 prog.c

    ## Check whether the changes of delta 1.2 are still present in the file or not. They should
    not be present.
    $ sccs diffs prog.c
    $ cat prog.c

    ## Save the changes in a new delta
    $ sccs delta prog.c

    ## Note:
    ## The following command excludes a range of deltas -- delta 1.3 to delta 1.5 .
    ## $ sccs edit -x1.3-1.5 prog.c

    ## The following command excludes delta 1.3 to the last delta of release 1.
    ## If prog.c has 10 versions in release 1: delta 1.1 to delta 1.10, then the following
    command excludes delta 1.3 to delta 1.10.
    ## $ sccs edit -x1.3-1 prog.c
    ```

11. Version numbers are usually in the form of 'x.y', where x = major release, y = minor changes. Suppose, prog.c is at version 1.3 (release 1 version 3). Get a copy of it for the next major release i.e. release 2.1 (release 2 version 1) using:
    ```
    $ sccs edit -r2 prog.c
    ## or
    ## You may change the release number of all source files in the current directory using:
    $ sccs edit -r2 SCCS
    ```

12. Use SCCS with Make: See Section 13 of 'An Introduction to the Source Code Control System' by Eric Allman at: http://sccs.sourceforge.net/man/sccs.me.html