B CS 2XA3/SE 2XA3 (2015/16, Term I) Proj 2 -- lab section L03

Back To Lab Menu | Back To Main Menu | Submissions | Log Out

sample solutions: makefile makefile1

In this project, there are two tasks, Task1 and Task2, and two deliverables, i.e. two files are to be created and submitted either via the Submission button at the top of this page, or using the `2xa3submit` command (see Lab 1). Both files are makefiles and are to be called `makefile` (for Task1) and `makefile1` (for Task2). The description of what these makefiles are supposed to do and contain is given below.

Task1: before you start working on `makefile`:

1. Download a text file t1 and save it on your workstation, then transfer it to ***moore*** and convert it to a unix text file (using dos2unix). *This file is necessary for the makefile to work.*
2. Download a text file t2 and save it on your workstation, then transfer it to ***moore*** and convert it to a unix text file. *This file is necessary for the makefile to work.*
3. Download a C++ program progAB.cpp and save it on your workstation, then transfer it to ***moore*** and convert it to a unix text file (using dos2unix). *This program is necessary for the makefile to work.*
4. Download a bash script sAB and save it on your workstation, then transfer it to ***moore*** and convert it to a unix text file (using dos2unix). Then make it executable (using chmod). *This script is necessary for the makefile to work.*

Task1: What should `makefile` do

1. It creates a bash script named `sBC` from `sAB` by replacing every occurrence of a string `AB` in the script `sAB` by a string character BC. Then it makes `sBC` executable using `chmod` command.
2. It creates a bash script named `sCD` from `sAB` by replacing every occurrence of a string `AB` in the script `sAB` by a string character CD. Then it makes `sCD` executable using `chmod` command.
3. It creates a C++ program (file) named `prog.cpp` by concatenation of the contents of the two files `t1`, and `t2` (in that order).
4. It creates a C++ program (file) named `progBC.cpp` from `progAB.cpp` by substituting every occurrence of a string `AB` in `progAB.cpp` by a string `BC` .
5. It creates a C++ program (file) named `progCD.cpp` from `progAB.cpp` by substituting every occurrence of a string `AB` in `progAB.cpp` by a string `CD` .
6. It creates the object files `progAB.o`, `progBC.o` and `progCD.o` by a typical compilation using `g++ -c` , for instance `g++ -c progAB.cpp` to create `progAB.o` .
7. It creates the executable `mAB` from `prog.cpp` and `progAB.cpp` by executing the script `sAB` .
8. It creates the executable `mBC` from `prog.cpp` and `progBC.cpp` by executing the script `sBC` .
9. It creates the executable `mCD` from `prog.cpp` and `progCD.cpp` by executing the script `sCD` .
10. It creates the executable `m1` by compilation of `prog.cpp` by the `g++` compiler with `-D_d1` flag, i.e. `g++ -o prog prog.cpp -D_d1`
11. It creates the executable `m2` by compilation of `prog.cpp` by the `g++` compiler with `-D_d2` flag, i.e. `g++ -o prog prog.cpp -D_d2`
12. It creates the executable `m3` by compilation of `prog.cpp` by the `g++` compiler without any flags, i.e. `g++ -o prog prog.cpp`
13. *When typing* `make all` *the executables* `mAB`, `mBC`, `mCD`, `m1`, `m2`, *and* `m3` *must be created.*

14. *When typing* `make clean` *all executable files* `mAB`, `mBC`, `mCD`, `m1`, `m2`, *and* `m3` *must be removed and the directory must only contain* `t1`, `t2`, `progAB.cpp` *and* `sAB` *as at the beginning.*
15. *When you execute* `mAB`, *you should see* `system AB is OPERATIONAL`
16. *When you execute* `mBC`, *you should see* `system BC is OPERATIONAL`
17. *When you execute* `mCD`, *you should see* `system CD is OPERATIONAL`
18. *When you execute* `m1`, *you should see* `Blue Jays give me blues.`
19. *When you execute* `m2`, *you should see* `I am blue because of Blue Jays.`
20. *When you execute* `m3`, *you should see* `NO specifications`
21. ***Some useful tips***:
    A substring (of any length, including a substring of length 1, i.e. single letter) can be "translated" to any other string by `sed` *command, for instance* `sed 's/HELLO/HELLO BYE/' fin > fout` *will read the input file* `fin` *and write into the output file* `fout` *while replacing the very first occurrence of* `HELLO` *in each line with* `HELLO BYE` . *To replace all occurrences, the global action must be indicated, i.e.* `sed 's/HELLO/HELLO BYE/g' fin > fout`

Task 2: before you start working on makefile1:

1. Create a directory `writer1` and in it a (unix) text file doc.txt with at least 1 line of length longer than 80 characters, the line having at least 6 sentences. *The content is not important as you will need this directory and this file only for testing*.
2. Create a directory `writer2` and in it a (unix) text file doc.txt with at least 1 line of length longer than 80 characters, the line having at least 6 sentences, but different from the file writer1/doc.txt . *The content is not important as you will need this directory and this file only for testing*.
3. Create a directory `writer3` and in it a (unix) text file doc.txt with at least 1 line of length longer than 80 characters, the line having at least 6 sentences, but different from the files writer1/doc.txt and writer2/doc.txt . *The content is not important as you will need this directory and this file only for testing*.
4. Create a directory `writer4` and in it a (unix) text file doc.txt with at least 1 line of length longer than 80 characters, the line having at least 6 sentences, but different from the files writer1/doc.txt, writer2/doc.txt, and writer3/doc.txt . *The content is not important as you will need this directory and this file only for testing*.
5. Download a C++ program format.cpp and transfer it to ***moore*** to the directory where you created writer1, writer2, writer3, and writer4. *This program is essential for making the makefile work.*

Task 2: what makefile1 should do:

1. There are 4 technical writers. Each of them writes his/her document in a unix text file doc.txt . The first writer has the directory writer1, the second writer has the directory writer2, ... , the fourth writer has the directory (*you guessed it* :-)) writer4.
2. But they write without proper formatting with lines that contain too many sentences and thus are too long, so before their documents may be put together, their writing must be "formatted" by a program format. The executable format is obtained from format.cpp by the usual compilation g++ -o format format.cpp . The program format expects the name (or pathname) of the file to be "formatted" as the first command line argument. For instance, to format the file doc.txt in writer1 you would use format writer1/doc.txt . The formatted text is displayed on the standard output, i.e. on the screen. Note that the input file doc.txt remains as it was, the formatted file is only displayed! *The program format makes lines that are at most 80 characters long and introduce an empty line after each sentence*.
3. The formatted documents are are all concatenated together (in the order 1 to 4) forming a file called document .
4. At the onset, the directory only contains writer1/doc.txt, writer2/doc.txt, ... , writer4/doc.txt and format.cpp and makefile1. Prepare the makefile makefile1 so that when make -f makefile1 document is executed, the documents of the 4 writers are formatted, concatenated together, producing document. The makefile can create temporary files, but they must be removed before the makefile is done, the same applies to the

executable format -- it may not remain in the directory. So, after running make -f makefile1 document, only the file document is added to the content of the directory.