Task 1. make file named `makefile1` The name of your file must be `makefile1` and below is a description of what it should do when executed. *Note that the names of the make files, the scripts, and of the files and directories, and the messages must be used exactly as described here, including the lower and upper cases.* Before you start working on the make file `makefile1`:

- Download a text file `partA` and save it on your workstation, then transfer it to ***moore*** to your working directory and convert it to a unix text file (using `dos2unix`). This file is necessary for the make file to work.
- Download a text file `partB` and save it on your workstation, then transfer it to ***moore*** to your working directory and convert it to a unix text file. This file is necessary for the make file to work.
- Download a C++ program `code1.cpp` and save it on your workstation, then transfer it to ***moore*** to your working directory and convert it to a unix text file. This program is necessary for the make file to work.
- Download a bash script `script0` and save it on your workstation, then transfer it to ***moore*** to your working directory and convert it to a unix text file. Then make it executable (using `chmod`). This script is necessary for the make file to work.
- The make file `makefile1` is to be located in the working directory.

What should `makefile1` do:

1. It creates a bash script named `script1` from `script0` by replacing every occurrence of the word `prog` by the word `code` and makes `script1` executable using `chmod` command.
2. It creates a bash script named `script2` from `script1` by replacing every occurrence of the character `1` by the character `2` and replacing every occurrence of the word `AB` by the word `BC` , and makes `script2` executable using `chmod` command.
   The best is to do it in three steps. In the first step use the command `tr` to replace every `1` by `2` , producing a temporary file.Then use the temporary file as an input to `sed` and use it to replace every occurrence of `AB` by `BC` , producing `script2` . Finally, in the third step, remove the temporary file.
3. It creates a bash script named `script3` from `script1` by replacing every occurrence of the character `1` by the character `3` and replacing every occurrence of the word `AB` by the word `CD` , and then makes `script3` executable.

4. It creates a C++ program `code.cpp` by concatenation of the contents of the two files `partA` and `partB` (in that order).
5. It creates a C++ program `code2.cpp` from `code1.cpp` by substituting every occurrence of the character `1` by the character `2` and by replacing every occurrence of the word `ALPHA` by the word `BETA` .
6. It creates a C++ program `code3.cpp` from `code1.cpp` by substituting every occurrence of the character `1` by the character `3` and by replacing every occurrence of the word `ALPHA` by the word `GAMMA` .
7. It creates object files `code1.o`, `code2.o`, and `code3.o` by compilation using the `g++ -c` , *for instance* `g++ -c code1.cpp` *to create* `code1.o` .
8. It creates an executable file `q1` from `code.cpp` and `code1.o` by executing the script `script1` . (*Note that it means that* `q1` *depends on the three files* `code.cpp` , `code1.o` , *and* `script1`).
9. It creates an executable file `q2` from `code.cpp` and `code2.o` by executing the script `script2` .
10. It creates an executable file `q3` from `code.cpp` and `code3.o` by executing the script `script3` .
11. It creates an executable file `ex1` from `code.cpp` by simple compilation with defining a `_W1` flag, i.e. `g++ -o ex1 code.cpp -D_W1`
12. It creates an executable file `ex2` from `code.cpp` by simple compilation with defining a `_W2` flag.
13. It creates an executable file `ex3` from `code.cpp` by simple compilation without defining any flag, i.e. `g++ -o ex1 code.cpp`
14. When typing `make -f makefile1 all` , the executables `q1`, `q2`, `q3`, `ex1`, `ex2`, and `ex3` must be created, but all the help files such as `code.cpp`, `code.o`, `code2.cpp`, `code2.o` etc. must be removed before the execution of the makefi le is finished.
15. When you execute `q1`, you should see a message `system ALPHA is GO`
16. When you execute `q2`, you should see a message `system BETA is GO`
17. When you execute `q3`, you should see a message `system GAMMA is GO`
18. When you execute `ex1`, you should see four messages `End of the term is approaching fast`
19. When you execute `ex2`, you should see four messages `End of the term is here`
20. When you execute `ex3`, you should see a message `It is past midterm`
21. When typing `make -f makefile1 clean` , all created files in the working directory must be removed and the directory must only contain `code1.cpp`, `partA`, `partB`, and `script0` as at the beginning (*Make sure not to remove* `makefile1` *nor* `makefile2` !)

A few useful hints:

- *If you do not have the current directory in the path, you have to refer to the files in your directory in the make file with the prefix* `./` *, for instance* `cat ./xxx` *instead of* `cat xxx` *. To quickly add the current directory to the path, execute the commands* `echo "PATH=$PATH:." >> ~/.bashrc` *and then* `source ~/.bashrc`
- *A single letter can be "translated" by* `tr` *command: for instance* `tr "1" "2" < fin > fout` *will read the input file named* `fin` *and output it to a file named* `fout` *and during this process it will translate each character* `1` *to* `2` *. The file* `fin` *will remain unchanged, the changes will be in the file* `fout` *.*
- *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 first occurrence of* `HELLO` *in each line with* `HELLO BYE` *. The file* `fin` *will remain unchanged, the changes will be in the file* `fout` *.*

Task 2. make file named `makefile2` The name of your make file must be `makefile2` and below is a description of what it should do when used. Before you start working on `makefile2:`

- Decide what will be your working directory.
- Download this ascii text data file `document1` to your workstation/laptop. Transfer it to ***moore*** to your working directory, and using `dos2unix` make sure it is a unix text file.
- Download this ascii text data file `document2` to your workstation/laptop. Transfer it to ***moore*** to your working directory, and using `dos2unix` make sure it is a unix text file.
- Download this ascii text data file `document3` to your workstation/laptop. Transfer it to ***moore*** to your working directory, and using `dos2unix` make sure it is a unix text file.
- Download this ascii text data file `document4` to your workstation/laptop. Transfer it to ***moore*** to your working directory, and using `dos2unix` make sure it is a unix text file.
- Download a C++ program `arrange.cpp` to your workstation/laptop. Transfer it to ***moore*** to your working directory, and using `dos2unix` make sure it is a unix text file.

- The make file `makefile2` is to be located in the working directory.

What should `makefile2` do:

1. When you type `make -f makefile2 build`
    1. directory `DOCREP1` is created in the current directory and the file `document1` is copied to `DOCREP1` with its name changed to `document`
    2. directory `DOCREP2` is created in the current directory and the file `document2` is copied to `DOCREP2` with its name changed to `document`
    3. directory `DOCREP3` is created in the current directory and the file `document3` is copied to `DOCREP3` with its name changed to `document`
    4. directory `DOCREP4` is created in the current directory and the file `document4` is copied to `DOCREP4` with its name changed to `document`
    5.
2. When you type `make -f makefile2 DOCUMENT`
    1. *Note that before you execute* `make -f makefile2 DOCUMENT`, *you should execute* `make -f makefile2 build`
    2. The contents of all the files `document` from the directories `DOCREP1` .. `DOCREP4` are formatted and concatenated together in that order to form the file `DOCUMENT` in the current directory.
    3. The reports are formatted using the software `arrange` . The executable `arrange` is obtained from the source file `arrange.cpp` by the standard C++ compilation: `g++ -o arrange arrange.cpp` . The program `arrange` expects the relative pathname of the file to be formatted as the first command line argument. For instance, to format the file `document` from `DOCREP1` , you would use `arrange DOCREP1/document` . The formatted text is released (displayed) on the standard output, i.e. on the screen. Note that the input file `document` remains as it was, the formatted file is only displayed !
    4. The makefile can create all kind of temporary files, but they all must be removed before the execution of the mak file is finished, the same applies to the executable `arrange`. So, after running `make -`

`f makefile2 DOCUMENT` , only the file `DOCUMENT` is added to the content of the working directory.

5. The text of `DOCUMENT` should look like this:

```
1-System is go. 2-System is go. 3-System is go.
4-System is go. 5-System is go. 6-System is go.
7-System is go. 8-System is go. 9-System is go.
10-System is go. 11-System is go. 12-System is go.
   13-System is go. 14-System is go. 15-System is go.
   16-System is go. 17-System is go. 18-System is go.
   19-System is go. 20-System is go.
   1-System is go. 2-System is go. 3-System is go.
   4-System is go. 5-System is go. 6-System is go.
   7-System is go. 8-System is go. 9-System is go.
   10-System is go. 11-System is go. 12-System is go.
   13-System is go. 14-System is go. 15-System is go.
   16-System is go.
   1-System is go. 2-System is go. 3-System is go.
   4-System is go. 5-System is go. 6-System is go.
   7-System is go. 8-System is go. 9-System is go.
   10-System is go. 11-System is go. 12-System is go.
   13-System is go.
   1-System is go. 2-System is go. 3-System is go.
   4-System is go. 5-System is go. 6-System is go.
7-System is go. 8-System is go. 9-System is go.
```

26. When you type `make -f makefile2 clean` , the directories `DOCREP1` .. `DOCREP4` are deleted with their contents, and so is a file `DOCUMENT` in the current directory. (*Make sure not to remove* `makefile1` *nor* `makefile2` !)