C: SE 2XA3 (2018/19, Term I) Major Lab 3 -- lab section L01

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

sample solutions: makefile1 and makefile2

If you are in the correct lab room and during the SCHEDULED time slot for your lab section for Major Lab 3 and working on one of the lab's workstations, go into Marks+Comments to see if your attendance was registered. If you signed on the course website too early (the most common problem), your attendance would not be recorded; in such case, please log out and sign on again. If your attendance is still not recorded, please, ask your TA to record your attendance and email Prof. Franek right away. If you are not using one of the lab's workstations, please, have the TA record your attendance.

The Major Labs are open book labs, you can bring and use any notes etc. You can access any website, Google, Wikipedia etc. The only thing that is not allowed is cooperation with other people, inside or outside the lab. You must submit your own work. The TA can only help with administrative and technical aspects and not with the solutions of the problems of the Major Lab.

In this lab, there are two tasks and two deliverables: text files `makefile1` and `makefile2`. Each can be submitted either via the course website, or using `2xa3submit`. For `2xa3submit` submission please use `2xa3submit AAA proj3 BBB` where `AAA` is your student number and `BBB` the name of the file you want to submit. You can submit every file as many times as you wish, the latest submission will be used for marking. The submission is opened from 8:30-11:20 or 14:30-17:20 depending on the lab section; after the closing time no submission is be possible. If submission is not possible for whatever reason (typically right after the submission closes), email the file or files as an attachment immediately (needed for the time verification) to Prof. Franek (franek@mcmaster.ca) with an explanation of the problem; you must use your official McMaster email account for that. Include the course code, your lab section, full name, and your student number. Note that if there is no problem with submission (typically the student using a wrong name for the file), you might be assessed a penalty for email submission, depending on the reason you used email.

Task 1. make file named `makefile1`

The name of your make file must be **makefile1** and below is a description of what it should do when used.

Before you start working on the make file `makefile1`:

- Decide what will be your working directory.
- Download a text file `frontpart` and save it on your workstation, then transfer it to ***moore*** to the 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 `endpart` and save it on your workstation, then transfer it to ***moore*** to the working directory and convert it to a unix text file. This file is necessary for the make file to work.
- Download a C++ program `pexam1.cpp` and save it on your workstation, then transfer it to ***moore*** and convert it to a unix text file. This program is necessary for the make file to work.
- Download a bash script `comp1` and save it on your workstation, then transfer it to ***moore*** to the 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.

1. The make file is to be located in the working directory.
2. It creates a bash script named `comp2` from `comp1` by replacing every occurrence of a string `X1` in the script `comp1` by a string `X2`. Then it makes `comp2` executable using `chmod` command.
3. It creates a C++ program (file) named `pexam.cpp` by concatenation of the contents of the two files `frontpart` and `endpart` (in this order).

4. It creates a C++ program (file) named `pexam2.cpp` from `pexam1.cpp` by substituting every occurrence of a string `Winter` in `pexam1.cpp` by a string `Summer` and by replacing every occurrence of the character `1` by the character `2` .
5. It creates a C++ program (file) named `pexam3.cpp` from `pexam1.cpp` by substituting every occurrence of the string `Winter` in `pexam1.cpp` by the string `Spring` producing a temporary file, and by replacing every occurrence of the character `1` in the temporary file by the character `3` producing `pexam3.cpp` .
6. It creates object files `pexam1.o`, `pexam2.o`, and `pexam3.o` by a typical compilation using `g++ -c` , for instance `g++ -c pexam1.cpp` to create `pexam1.o` .
7. It creates an executable `run1` from `pexam.cpp` by executing the script `comp1` .
8. It creates an executable `run2` from `pexam.cpp` by executing the script `comp2` .
9. It creates an executable `run3` by a compilation of `pexam.cpp` and linking of `pexam1.o`, `pexam2.o`, and `pexam3.o` together by the `g++` compiler with `-D_X3` flag, i.e. `g++ -o run3 pexam.cpp -D_X3 pexam1.o pexam2.o pexam3.o`
10. It creates an executable `run4` from `pexam.cpp` by a simple compilation of `pexam.cpp`, i.e. `g++ -o run4 pexam.cpp`
11. When typing `make -f makefile1 all` , the executables `run1`, `run2`, `run3`, and `run4` must be created, but no other files can be created Of course, you can use all kind of temporary files, but these must be destroyed before the make file is finished
12. When typing `make -f makefile1 clean` , all created files must be removed and the working directory must only contain `frontpart`, `endpart`, `pexam1.cpp` and `comp1` as at the beginning (*Do not delete* `makefile1` *nor* `makefile2` !)
13. When you execute `run1` , you should see a message
    `In Canada, there are officially four seasons`
14. When you execute `run2` , you should see a message
    `In Canada, there are four seasons: winter and summer`
15. When you execute `run3` , you should see messages
    ```
    Winter is nice
    Summer is nice
    Spring is nice
    ```
16. When you execute `run4`, you should see a message
    `In Canada, there are two seasons: winter and road repair`

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 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.
- In the working directory create a subdirectory `hackteam1` and in it a code file `myhack.cpp` downloaded from here . Make sure it is a unix text file by using `dos2unix` . You will need this directory and this file only for testing.
- In the working directory, create a subdirectory `hackteam2` and in it a code file `myhack.cpp` downloaded from here . Make sure it is a unix text file. You will need this directory and this file only for testing.
- In the working directory, create a subdirectory `hackteam3` and in it a code file `myhack.cpp` downloaded from here . Make sure it is a unix text file. You will need this directory and this file only for testing.
- Download a C++ program `tojoin.cpp` and transfer it to ***moore*** to the working directory and make sure it is a unix text file.

- Download a C++ program addsemi.cpp and transfer it to ***moore*** to the working directory and make sure it is a unix text file.

1. There are 3 teams in a hackathon. Each of them prepares its solution in a file `myhack.cpp` . The first team has the directory `hackteam1`, the second team has the directory `hackteam2`, ... , and the third team has the directory (you guessed it :-)) `hackteam3` .
2. But the programmers from all three teams cannot use the semicolon character `;` for technical reasons (the key with `;` is missing on their keyboards :-)), so they must end their C++ statements with a word `@semi` instead. So before their respective parts may be put together into one program, their code must be prepared by a program `addsemi` . The executable `addsemi` is obtained from `addsemi.cpp` by the usual compilation `g++ -o addsemi addsemi.cpp` . The program `addsemi` expects the relative pathname of the file to be prepared as the first command line argument. For instance, to prepare the file `myhack.cpp` in `hackteam1` , you would use `addsemi hackteam1/myhack.cpp` . The prepared text is displayed on the standard output, i.e. on the screen. Note that the input file `myhack.xpp` remains as it was, the prepared file is only displayed (Just FYI, `addsemi` replaces every occurrence of `@semi` with a semicolon `;`)
3. Besides the problems with the semicolon, all the coders make lines that contain too many statements. So before all the parts can be joined together, they must be modified to have lines no longer than 80 characters and have an empty line after each statement. This is what the program `tojoin` will do. The executable `tojoin` is obtained from `tojoin.cpp` by the usual compilation `g++ -o tojoin tojoin.cpp` . The program `tojoin` expects the relative pathname of the file to be modified for joining as the first command line argument. For instance, to modify for joining the file `xxx.cpp` you would use `tojoin xxx.cpp` . The modified text is displayed on the standard output, i.e. on the screen. Note that the input file `xxx.cpp` remains as it was, the modified file is only displayed Also note that the program `tojoin` expects the statements to be terminated with a semicolon.
4. The prepared and modified by `tojoin` texts are then all concatenated together (in the order 1 to 3) forming a file called `solution.cpp` .
5. At the onset, the working directory only contains `hackteam1/myhack.cpp`, `hackteam2/myhack.cpp`, `hackteam3/myhack.cpp`, `addsemi.cpp`, `tojoin.cpp`, and `makefile2` .
6. Prepare the make file `makefile2` so that when `make -f makefile2 solution.cpp` is executed, the codes of the 3 teams are prepared and modified for joining, concatenated together, producing `solution.cpp` . The make file can create temporary files, but they all must be removed before the make file is done, the same applies to the executables `addsemi` and `tojoin` -- they may not remain in the working directory. So, after running `make -f makefile2 solution.cpp` , only the file `solution.cpp` is added to the content of the working directory (*Do not delete* `makefile1` *nor* `makefile2` !)

The resulting file `solution.cpp` should look like this: a whole bunch of lines
`printf("hackteam1 solution");` separated by empty lines, followed by a whole bunch of lines
`printf("hackteam2 solution");` separated by empty lines, followed by a whole bunch of lines
`printf("hackteam3 solution");` separated by empty lines.