Lab #4
Advanced Make
Due in Lab, September 22, 2004

Name: _____

Lab Time: _____

Grade: _____/10

Localizing a Makefile

When installing the `gnuchess` program, we made use of a shell script called `configure`. This shell script automatically created the makefile that was needed to install the program. Sometimes distributions are not that nice, and you have to edit the makefile yourself.

1. In the directory `~csci3308/src` there is a tar file named `lab04.tar`. Copy this file to your `~/csci3308/tmp` directory.
2. Now change to your `~/csci3308/src` directory and unpack the tarfile with the command `tar xf ../tmp/lab04.tar`.
3. This should create a directory named `lab04`. Change into this directory.
4. Now read the `makefile`. It suggests that you change certain variables to fit your own directory structure. List here the variables you should change and the values you should give them. Use `$(HOME)` instead of `~`, and remember to surround every variable name with parentheses. Make those changes now. Note: you may need to create a build directory for lab04 (within your architecture-specific build directory) before running the makefile.

5. Type `make`. Was the program built? Was it installed?

6. If the program was not built, fix any error so that the program builds correctly. What did you have to type to get the program installed? (Note: there are two answers to this question both equally correct. List both answers if you can.)

7. Type `rehash` and `where answer`. Did the shell find the program? If not, what do you need to do to make the shell find the program?

8. Type `make clean`. What does this do?

9. Type `make install`. What happened and why?

10. Type `make all`. How was this different from `make install`?

11. How could you modify the makefile so that merely typing `make install` would work even if the program wasn't built yet? Hint: This will require that you change the rules that the makefile says you don't need to change. However, **don't actually make the change, just describe it**.

12. Copy your current makefile into a file called makefile-1. Be sure to turn in a copy of makefile-1 with your lab assignment. Type `make clean` again before going on to the next section.

Using Implicit Rules

Now you will modify the makefile to use an implicit rule. We will also modify the makefile to act more like the makefile that was used to install gnuchess in lab 01. In particular, we are going to transform it, such that it needs to be invoked in the build directory.

13. The first step in preparing for this new makefile is to remove all references to the BUILDDIR macro. We can do this safely, since we already saved a copy of our makefile in the file makefile-1. This step involves deleting the BUILDDIR macro definition and then removing all references to BUILDDIR in the makefile. Thus, for instance, the very first rule will now look like this:

    ```
    answer: answer.o
         g++ answer.o -o answer
    ```

    The next step is to remove the rule for compiling `answer.cc` into `answer.o`. This rule is unnecessary because make has an implicit rule for doing this. Look at the list of make implicit rules in the reference materials section of the class website. This list can also be found in the make man page. Find the rule `.cc.o` which is a suffix rule for making `.o` files from `.cc` files.

    This rule has the following structure:

    ```
    .cc.o:
         $(COMPILE.cc) $(OUTPUT_OPTION) $<
    ```

    So, you can see that the action for this rule is created via two macros, `COMPILE.cc` and `OUTPUT_OPTION` and an automatic variable that specifies that the first dependency (e.g. the .cc file) should be compiled by this command. (If you look at the top of the implicit rules webpage, you will see that the macro `COMPILE.cc` is further defined to consist of three other macros. The important one, for our purposes, is `CCC`.

14. Remove the rule for making `answer.o` from the makefile so that make will use the implicit rule. Now you need to configure the implicit rule's macros.

15. The implicit rule needs to know where to find the source file, `answer.cc`, so you need to set the `VPATH` variable. Do that now.

16. The implicit rule also needs to know where to place the intermediate file `answer.o`, so you need to set the `OUTPUT_OPTION` variable. Use an automatic variable in your definition of `OUTPUT_OPTION` so this rule will work for any `.o` file, not just `answer.o`. Add this macro definition to your makefile now.

17. By default, make uses the `cc` compiler, but you need to use `g++`, so you must set the `CCC` variable. Add this macro definition to your makefile now.

18. While we are at it, change the first rule of the makefile to make use of automatic variables.

19. Now change to lab04's architecture-specific build directory and type

    ```
    make -f $HOME/csci3308/src/lab04/makefile all
    ```

    What happened?

Be sure to hand in a copy of both makefiles with your lab.