Lab #2
Find and Grep
Due in Lab, September 8, 2004

Name: _____

Lab Time: _____

Grade: _____/10

Find

The find command is used to locate files in a file system. The find command starts at a single directory and descends recursively into all of its subdirectories to locate files. The find command is very useful, but it has an obscure syntax that can be hard to understand. The format of the find command is as follows:

```
find path operators
```

Type the following:

```
find ~/csci3308 -print
```

`-print` is an operator that prints the file name of a file. The directory `~/csci3308` says to start from that directory and apply `-print` to every file in every directory below it. This will print a lot of files, and could also be accomplished with ls -R.

Now type this:

```
find ~/csci3308 -name bin -print
```

This will print only files and directories named `bin` in your directory structure. This is the purpose of find: to locate files and directories that match specified characteristics.

You can also search for wildcard expressions. Type the following:

```
find ~/csci3308 -name \*.c -print
```

This should print every file that ends with `.c`. The backslash is used to prevent the shell from interpreting the asterick in the above command line. Another way of writing the same command is:

```
find ~/csci3308 -name '*.c' -print
```

1. Write a find command to print all files under your csci3308 directory whose names are exactly three characters.

Quoting in the Shell

The quotes above are necessary because you want `*.c` to be interpreted by the **find** program, and not the shell. Without the quotes the shell will process the `*` character and **find** will never see it. Find knows how to interpret shell wildcards, but you need to keep the shell from replacing them before find can see them.

There are three types of quotes in tcsh, single quotes, `'`, double quotes, `"`, and backslash, \. The purpose of quotes is to make the shell ignore characters that it would normally interpret or replace. Single and double quotes must come in matching pairs, and they affect everything between the quotes. Backslash affects only the single character immediately following the backslash. There are some minor differences between single quotes and double quotes. Finding the right quotes for a particular situation can sometimes be a process of trial and error.

Here is a brief summary of the differences between single quotes and double quotes when using them at the command line. Single quotes protect all metacharacters from interpretation, with the exception of the history character (i.e. !). Metacharacters are characters which have special meaning to the shell like * or \. Double quotes protect all metacharacters from interpretation, with the exception of the history character (!), the variable substitution character ($), and the back quote, which is used for command substitution (see below).

For a complete description of quoting-related issues, see the tcsh man page, or a book that covers Unix-related topics.

2. Type the following and record the results for each line next to it below:

```
echo $shell $term
echo '$shell $term'
echo "$shell $term"
echo \$shell $term
```

3. Why is the output of the first and third lines the same?

In addition to quoting, there is the backward single quote, `‘`, also called back tick or back quote. The back quote invokes command substitution, and its purpose is to cause more substitution, not less. Anything between two back quotes is considered to be a command. This command is executed and its output is placed on the command line where the original command used to be. Try the following:

```
echo date
echo ‘date‘
```

Back to Find

`-name` and `-print` are just two operators, and find has many more. Type the following:

```
find ~/csci3308 -perm -004 -print
```

This should find all files in your csci3308 directory that have read permission for the `others` category. You can use this command to check what files in your directory are publically accessible to other users.

4. Rewrite the command above to locate those files whose permissions are exactly "004". Use the `find` man page for help.

Look at the find operators in the find man page.

5. Write a find command that prints all files in your `~/csci3308` directory accessed less than 5 days ago. Try this command out to make sure it works.

6. Write a find command that prints all files in your `~` directory that are of type directory, and whose names contain a numeric character.

`Find` can create more complicated boolean expressions with its operators. The exclamation point stands for `not`. The following command finds files underneath the current directory whose names do *not* end in `.c`.

```
find .  !  -name '*.c' -print
```

Sometimes parentheses must be used for grouping especially when using `-a` (and), or `-o` (or). The parentheses must be quoted with either quotes or a backslash so that the shell does not interpret them.

```
find .  \( -type f -o -type d \)
           -a \( -name 'program*' -o -name bin \) -print
```

`Find` has two operators, `-exec` and `-ok` that can be used to run other programs on the files that are found. Try the two commands below:

```
find .  -type d -name bin -exec echo {} is cool \;
find .  -type d -name bin -ok echo {} is cool \;
```

7. What is the difference between `-exec` and `-ok`?

In every `-exec` or `-ok` command you can use the two characters {}. When the command is run these characters are replaced with the name of the file being found. If multiple files are found the command is run once for each file, and each time {} is replaced with a new name. Also, any `-exec` or `-ok` command must end with a semicolon (';') which must be quoted, in this case with backslash. If the semicolon is missing, the shell will report a syntax error.

Grep

Grep searches the contents of a file to find lines that match a pattern. Grep uses regular expression syntax for its pattern matching, which is different than shell wildcards. Once again, patterns have to be quoted to keep the shell from trying to evaluate them. In this lab we will be using `egrep`, expression grep, which is grep with a few extra features. In general, we will use the words `grep` and `egrep` interchangeably. In most cases, grep and egrep return the same results...we will try to highlight cases in which their output is different. For instance, the two commands below produce different output (You can look at the file /usr/dict/words if you want):

```
egrep '^b.*(na)+$' /usr/dict/words
 grep '^b.*(na)+$' /usr/dict/words
```

8. What is the output for each command and why is it different?

Returning to our discussion, grep is different from find in another way. Find only matches a filename if the entire filename matches a pattern. grep will display a line as a "match" if any part of the line matches an input pattern. Try this:

```
egrep fly /usr/dict/words
```

Grep finds not just the word fly, but every word that contains the word fly. If you want to match exactly your pattern and nothing more use the beginning and end of line characters, `^` and `$`.

```
egrep ^fly /usr/dict/words
egrep fly$ /usr/dict/words
egrep ^fly$ /usr/dict/words
```

Sometimes, you need to keep grep from interpreting special characters too. For example, you may want to find all lines with matched parenthesis in your `.cshrc` file. You start by trying this:

```
egrep '(.*)' ~/.cshrc
```

the shell sees `'(.*)'` and it knows not to evaluate special characters in this word. It strips off the quotes and passes `(.*)` to egrep. However, egrep uses parentheses for grouping. egrep assumes you want to find the pattern `.*`, and you put parentheses around it just to be careful. So egrep tries to match the pattern `.*` which matches anything and prints every line of your file. What you actually wanted was the literal character left parenthesis, `(`, then anything, `.*`, and then a literal right parenthesis, `)`. You need to put backslashes in front of the parentheses to make egrep treat them as literal characters.

```
egrep '\(.*\)' ~/.cshrc
```

So the single quotes are for the shell, and the backslashes are for egrep. You still need the single quotes or the shell would try to interpret the backslashes and asterisk. Sometimes getting the exact pattern you want can be tricky.

Note: this is an instance where grep and egrep return different results. grep does not assign special meaning to parentheses so the parentheses do not need to be quoted when using grep. egrep has a more powerful regular expression engine than grep in which parentheses are assigned a special meaning, which we will cover later in the semester.

9. Write an egrep command to search a file named `text` for lines that contain a backslash. To test this create a sample file named `text` in your tmp directory with some lines that contain a backslash, and some that don't.

Using Grep with Find

Grep and find can be combined to create a powerful search tool. Write a find command, and then add the `-exec` operator to run a grep command on the files that it finds. The following command finds all .c files in your directory structure that contain the word help:

```
find ~ -name '*.c' -exec egrep help {} \; -print
```