

## Instructions

Complete the problems below and turn in your answers by the due time above to receive extra credit. To receive credit, you must show both the relevant command(s) and output.

### Extra Credit Problems

1. Write a function called **mygetopts** that will process options like **getopts** does, but in addition, also creates and sets a variable for each option found named **opt\_option**, where *option* is the option letter found on the command line. For option-arguments (i.e. the *prefix* in `-P prefix`), also create a variable for the option-argument called **opt\_option\_arg**. The first argument to **mygetopts** is a string of the acceptable options in the same syntax and format as the first argument to **getopts**. Since **mygetopts** processes options itself (unlike **getopts**), you don't need to call **mygetopts** in a loop repeatedly – just call it once. Test your function by placing it in a program that calls **mygetopts**. Here is an example of your program being called:

```
$ test_myoptarg -abcd -e -i inputfile -f -P 'xxx yyy' -g file1 file2 file3
```

Make sure that program can process the command line above (including the remaining non-option arguments) using your **mygetopts** function. Demonstrate that each of the appropriate variables was created and set properly (in the case above, `opt_a`, `opt_b`, `opt_c`, `opt_d`, `opt_e`, `opt_i`, `opt_i_arg`, `opt_f`, `opt_P_arg`, and `opt_g`; the value of `opt_i_arg` is 'inputfile' and the value of `opt_P_arg` would be 'xxx yyy').

Here is a one implementation. Several people read this problem to mean that you were not allowed to use **getopts**, and ended up processing the options manually. It is important to read problem requirements carefully, and to validate your assumptions before you begin coding. This is a common mistake in programming. The cost of fixing mistakes, or spending time trying to code based upon invalid assumptions, is orders of magnitude more expensive than clarifying assumptions before design and coding begins.

```
mygetopts() {
    optstring="$1"
    shift
    while getopts $optstring option ; do
        case "$option" in
            ?)
                echo setting opt_$option
                eval opt_$option=1
                if [ -n "$OPTARG" ] ; then
                    eval opt_${option}_arg='${OPTARG}'
                    unset OPTARG
                fi
                ;;
        esac
    done
}
```

Here is a Bourne shell program that uses **mygetopts** to test it:

```
mygetopts 'abcdefghijklmnop:P:i:' "$@"

shift `expr $OPTIND - 1`
while [ $# -ne 0 ] ; do
    echo Arg: $1
    shift
done

set | grep '^opt'
```

2. Being able to read code written by others is even more important than writing code yourself. Describe what the program `~cappella/cis68b1/extra/mystery` does, and why it is coded the way it is, as thoroughly as you can, by commenting the code itself. There are many lines of code whose meaning is not clear – be sure to document as many of these as you can. Turn in the documented version of the code.

This original commented code is available at <http://www.shelldorado.com/scripts/cmds/rmc.txt>.