The Shell or Command Line Interpreter is the fundamental User interface to an Operating System. Your project is to write a simple shel, entitled **myshell** which has the following properties:

1) The shell must support the following internal commands:
    i) cd <directory> - change the current default directory to <directory>. If the <directory> argument is not present, report the current directory, by printing it out on the screen. If the directory specified by the user does not exist an appropriate error should be reported. This command should also change the PWD environment variable.
    ii) clr - clear the screen. Note that this does not take any arguments.
    iii) dir <directory> - list the contents of directory <directory>.
    iv) environ - list all the environment strings
    v) echo <comment> - display <comment> on the display followed by a new line. Note that multiple spaces/tabs may be reduced to a single space.
    vi) help - display the user manual using the more filter. this means that the manual should be displayed in fixed blocks of 20 or 25 lines, and when a space is entered by the user (that is the spacebar key is pressed) the next block of text is displayed.
    vii) pause - pause operation of the shell until the **Enter**
    viii) quit - quit the shell.
    ix) The shell environment should contain shell=/myshell where /myshell is the full path for the shell executable (not a hardwired path back to your directory, but the one from which it was executed).
2) All other command line input is interpreted as program invocation which should be done by the shell forking and execing the programs as its own child processes.
3) The shell must be able to take its command line input from a file. i.e. if the shell is invoked with a command line argument: myshell batchfile then batchfile is assumed to contain a set of command lines for the shell to process. When the end-of -file is reached, the shell should exit. Obviously, if the shell is invoked without a command line argument it solicits input from the user via a prompt on the display.
4) The shell must support i/o-redirection on output. i.e. the command line: **programname arg1 arg2 > outputfile**will execute the program programname with arguments arg1 and arg2, the output (possibly the screen) replaced by outputfile. Output redirection should also be possible for the internal commands: dir, environ, echo, & help. For example, if the command **help > output.txt** is given, then, with output redirection, the material which would usually be displayed on screen will now go to a file instead. If you wish, when displaying the help command, you may just output the entire helpfile to the output file without having to press space

multiple times. Alternatively, the user can simply press space a few times until the prompt returns. This is entirely up to you. if the redirection character is > then the outputfile is created if it does not exist and overwritten by the new contents if it does. If the redirection token is >> then outputfile is created if it does not exist and appended to if it does.

5) The shell must support background execution of programs. An ampersand (&) at the end of the command line indicates that the shell should return to the command line prompt immediately after launching that program.
6) The command line prompt must contain the pathname of the current directory.

Note: you can assume that all command line arguments (including the redirection symbols, >, >> and the background execution symbol, &) will be delimited from other command line arguments by white space - one or more spaces and/or tabs (see the command line in 4. above).