1. Get a copy of the sample program `s.sh.c` and compile and link it by typing:
   ```make s.sh < cr >```
   Make printouts of the source for this program. If you have a Windows NT system available you should try building it under Windows NT as well.

2. The program itself is a minimal shell, which has additional modifications in the case of Windows NT so that it can dialog through two pipes. These modifications are unnecessary in the case of Unix since both fork’d and exec’d processes inherit the open descriptors of the parent (except in the case of descriptors which have the `close-on-exec` flag set). This shell supports only the following features, but does show you what it is doing as it operates:
   
   i. It will parse a command line into tokens assuming that they are separated by whitespace (i.e. spaces, tabs, or a combination of these).
   
   ii. It will allow use of one wildcard (* or ?) on the command line, and will use `fnmatch()` to expand these arguments.
   
   iii. It has the following internal commands: `cd`, `dir`, `envp`, `help`, `pwd`, `tty`, and `exit`.
   
   iv. It will try to find external commands making use of the PATH environmental variable and it will get the st_mode of the (supposedly) executable file (e.g. `rwxr-xr-x`).
   
   v. It uses a `fork()..execve()` sequence of calls to execute commands under Unix and a `spawnve()` call to execute commands under Windows NT.
   
   vi. It allows modifying its own environment from the command line (e.g. `TERM=vt100` redefines the value of the environmental variable `TERM`.
   
   vii. It does allow backgrounding a job using the ampersand token `&` at the end of the command line. It also supports a `jobs` command.

   Note that the following features (common to most shells) are not supported:

   viii. It will not allow more than one wildcard character (* or ?) per command line.
   
   ix. It does not support redirection of input or output, using, for example, the > and < tokens.
   
   x. It does not support piping the output of one program to the input of another, using, for example, the | token.
   
   xi. Although it does maintain a command history and allow re-execution with the `!` [num] internal command it does not allow editing of current or previous command lines (c.f. the `set -o vi` option in `ksh` or `bash`).
   
   xii. It does not allow the use of `$` in a command to refer to the contents of an environmental variable (e.g. `echo $TERM`).