For this assignment, you get to extend your shell implementation by adding system calls to fork
new processes, make pipes and redirect input and output as appropriate. As we discussed in class, you can
assume that all the special syntax, >, <, | and & will be parsed into separate words by the command.c
component. This means you will need to use spaces to separate pipe and redirection syntax from other
words on the command line; a real shell doesn’t require this, but it’s probably OK for “rubbish.”

You should extend your shell to support the following:

- **Run external commands**
  If the user types a command that’s not a built-in, your shell should fork a child process and have the
  child exec the command. If the child can’t find the command, it should print a failure message to
  standard error and then exit with a failure exit status. Normally, your shell will wait for the new child
  process to terminate before issuing a new prompt.

  Your shell supports built-in commands and commands run via a separate executable. First, you will
  try to run a command via runBuiltin(). If the command isn’t built in, you will run it as a separate
  executable.

- **Backgrounding**
  If the user puts an & at the end of the command, your shell should run its command in background.
  This just means the shell process won’t wait for the child process to terminate before it prints out
  another prompt and tries to read the next input command.

  If a command ends with an &, your program should make a note that the command is to be back-
  grounded and then remove this last word from the command. This will make it easy for you to then
  process syntax for I/O redirection.

- **Stdin and stdout redirection**
  After removing any backgrounding notation, if the last two words in a command line are > and a
  filename, you will redirect output of the command to the given file. As we did in class, you will do this
  by forking a new process and then redirecting standard out.

  If the last two words in a command line are < and a filename, you will redirect input of the command
  from the given file.

- **Pipe Support**
  You will need to be able to support pipes from one command to another. Your program only needs to
  support up to one pipe on the command line. Remember, all writing ends of a pipe must be closed in
  order for the pipe to indicate EOF on the reading end. For the process that’s writing to the pipe, this
  will happen when it terminates. You just need to make sure that all other process that have a copy of
  the writing end of the pipe close their copy of this end.

  You don’t need to worry about combinations of pipes and stdin/stdout redirection.

- **Built-ins as separate processes**
  We would normally like to run built-in shell commands without creating a new process. This will let
  us run them much more efficiently, and it will let commands like “cd” change the directory of the
  shell itself. However, if we pipe, redirect or background a command, it’s going to need to be run as a
  separate process whether or not it’s a built-in. Modify your shell so that it creates a child process for
  built-ins only when they are backgrounded, run as part of a pipe or have their input/output redirected
to a file.
I suggest putting the logic for forking a child process into your main() function. Have main() fork a child before calling runBuiltins() in the cases where the built-in must be run as a separate process. Then, runBuiltins() will know that a separate child process has already been created, and I/O has been redirected if this was required.

For this assignment, we are going to use a SVN repository which will be stored in your home directory. After class on Monday, I’ll take a copy of the latest version of your project in the CVS repository, and I’ll create a SVN repository for the project. Your source code will be in a directory called “rubbish” under a svn repository called “srep” in your home directory. As before, start working on your shell by checking files out of the repository. When you’re done, commit all your changes back to the repository and I’ll grade the version that’s there in your repository.