Module 011 Project: A Scribbler 2 Shell
CSC 161, “Imperative Problem Solving and Data Structures”
Department of Computer Science · Grinnell College
March 15, 2016

The Pairs (revised after class meeting)

- Mattori Birnbaum and Jong Hoon Bae
- Zachary Susag and Anna Blindermann
- Eli Salm and Tanner Tufto
- Tyler Williams and Ella Nicolson
- Cory McCartan and Saung Thuya
- Ying Zhang and Josh Lavin
- Sophie Gaschott and Lex Martin
- Dennis Chan and Faizaan Ali
- Gemma Nash and Sanjay Sudhir
- Lilly Webster and Erhaan Ahmad
- Yuyin Sun, Colin Greenman, and Jae Hong Shin

The Assignment

The project is to design, implement, and test an interactive command interpreter through which a user can direct the activities of a Scribbler 2 robot. The general plan is that the interpreter will begin by establishing a connection to the robot and will then generate a prompt in the terminal window at which the program’s user can type a one-word command from a fixed repertoire of things the robot can do. Some of these actions may be simple, corresponding to single calls to functions in the MyroC library, but you can make them as complicated as you like.

The interpreter should read in, as a string, the command that the user types and execute the code that tells the robot, step by step, what to in order to carry out the action specified by the command. Having executed this code, the interpreter should then print out another prompt and wait for another command from the user.

The interpreter should also recognize a “quit” command, by which the user can leave the prompt-read-execute loop. The interpreter should then disconnect from the robot, print the number of user commands it has executed, and exit.

Also, the interpreter should recognize a “help” command, which displays (in the terminal window) a list of the commands, with brief descriptions of the corresponding actions.

Commands can take numerical parameters, as appropriate. (For instance, in a command that makes the robot beep, you might allow the user to specify the frequency and duration of the tone.)

The interpreter should recognize at least eleven commands altogether (including the “quit” and “help” commands), and at least five of these should accept one or more numerical parameters. You’ll find that the design and implementation of the interpreter are easier if you implement each of the commands except the “quit” command as a separate function. The main loop of the interpreter can manage all the input and output and call the appropriate action functions with the appropriate parameters.

Each command should be a full word, not just a letter. You should provide useful error reports if the user types in a word that is not one of the recognized commands, or if the parameters are missing, ill-formed, or out of range. You may assume that the user
types in the command and its parameters on a single line of not more than 1022 characters (not counting the newline character that terminates the line).

Submitting the Program

I’ll collect the programs in hard copy at the beginning of class on Friday, March 18.

We’re still working on a mechanism for submitting programs electronically without sharing them to the world. Until that’s ready, just include the full pathname of your program in the opening comment, and I’ll pull a copy directly from inside your home directory.