String formatter:
This program is a simple file that takes in user input and returns the same inputs later after being stored in local variables. On line 5, the print operator outputs the string immediately after to stdout. On line 6, the “my” keyword is used to statically define a local scope for the variable, “name”. The value of this new variable is assigned to the value provided by the user from stdout, and then the trailing newline character read in is stripped from the end of the string using the chomp function. This process is repeated 3 more times, so the user is prompted for a date, age, and a description of themselves and their answers are stored in the corresponding variables. Lines 22-24 simply print the values stored in the “name”, “date”, and “age” variables respectively. On lines 28 and 29 a maximum width and a current width are defined. Line 32 establishes a foreach loop that iterates through each word in the “about” variable and checks to see if the current width plus the length of the word plus 1 is greater than the width. If this is true, then the current word is printed on a newline and “current” is updated to be the length of the current word. If not, then the word is printed with a space before it and “current” is updated to the length of the current word plus 1. This check formats the printout of the “about” variable to create a new line every time maximum width is met to allow the full description to be easily read at once.

Log parser:
This program reads in a log file from the command line and returns the number of informational, warning, and error messages found in the file. On line 2, “use strict” is used to cause the program to stop if an error occurs during execution. Line 3 is used to display syntactical warnings to the programmer such as forgetting a curly brace or using formatting no longer supported by the language. Both of these tools can be useful for simple debugging. The “if” statement on line 8 checks to see if the number of command-line arguments submitted by the user is equal to 1 and sends an error message using “die” if the check is false. Line 12 creates a local variable to store the file name by reading the input from the command line. Line 14 opens the file, reads the contents of the file, and then stores the contents to a local variable or sends an error message if the process fails. The less than sign is used to denote reading from a file, but it could be replaced by a greater than sign to indicate that content should be written to the file instead. Lines 16-20 are used to define a hashmap that holds the counts for each of the 3 entry types in the log file. The while loop on line 22 continues to iterate through the file until there are no more lines to be read in from the file. Lines 23-25 are used to strip all trailing and leading white spaces from the line. In line 24 the “s/” in the regexp denotes searching for something in the string and replacing it with something else. The “\r$” indicates that any “\r” characters at the end of the line should be found, and the “//” indicates they should be replaced with an empty string, thereby stripping any trailing carriage return characters from the string. Line 25 does something similar by searching for any amount of whitespace characters at the beginning of the string, denoted by “^\s+”, uses the “or” operator to find an additional term, denoted by the “|” symbol, finds any amount of whitespace characters at the end of the string, denoted by “\s+$”, and then replaces all whitespace found with an empty string, denoted by “//g”. The line is then checked to see it contains the keyword to denote entry type, either “INFO”, “WARN”, or “ERROR” and the corresponding count in the hashmap is updated, if found. The program ends by printing the count of all three entry types to stdout with corresponding labels.