# Making Components

Total 100 points.

# Introduction

This assignment involves writing a system that reads in data files containing inventory data for parts and instructions on how the parts are used to make larger components. Then the system can answer queries about how many of a component can be made from current stock.

# Definitions

MachineIDs will be an M followed by 1 or more digits like M3 or M3516.

PartIDs will be a capital letter that isn’t M, possibly more capital letters (including M) followed by one or more digits like A1 or ABCM435.

Quantities will be positive integers, including 0.

Components and Parts are the same thing.

# Input Files

Your script starts by reading $ARGV[0] which is a plain text file. It has a list of components and what it takes to make them, 1 per line. A line will start with the PartID of the component to be made, the machine needed to make the part, then a list of PartID and non-zero Quantity pairs that are the amount of each subpart needed. Each field is separated by a space. Format:

*PartIDA MachineID PartID1 Quantity1 PartID2 Quantity2 …*

Example:

ABC123 M15 A1 3 B4 2

This means that part ABC123 is made of 3 A1 parts and 2 B4 parts on machine M15. A part can be made from 1 or more other parts together with exactly 1 machine. Example:

A1 M29 A2 1

Means A1 is made on machine M29 using 1 A2. To avoid infinite loops/recursion, and because it makes sense, you should assume that these build instructions will not result in a loop. So a situation like A1 is made from A2, A2 is made from A3, and A3 is made from A1 is not going to happen (you don’t have to test for it in your validity checks). Furthermore, to make things easier, a part will be used to make at most one other part. So A1 can’t be used to make both ABC123 and A5. Again, you may assume this and don’t need to check for it. Similarly, a machine will only be able to be used to make 1 type of part.

Next, your script reads $ARGV[1] which is a list of current stock. Each item will be a PartID followed by a Quantity.

Finally, your script reads $ARGV[2] which is a list of machines. Each line will be a MachineID followed by a Quantity.

Your script should verify that the each file and the system as a whole are valid before continuing. Each file is valid if it follows the formats described above. For the system to be valid every machine in $ARGV[0] must have a quantity in $ARGV[2] and every part in $ARGV[0] must have a quantity in $ARGV[1]. It is possible that some of those quantities are 0. Print a useful error message (what line in which file was invalid, which part or machine number isn’t in inventory, etc.) and exit cleanly if there is a problem.

# Input Source

After reading and verifying the input data files, your script should then check $ARGV[3]. If it exists then it will be the name of another text file. This text file will have 1 command on each line. Your code should loop through all the commands in the file, execute them and their results should print out to the screen as usual. If $ARGV[3] is not present, then your code should enter interactive mode. In this mode, your code should prompt the user with “Enter command: “ and then read STDIN. A command of Quit causes the system to exit (gracefully, not die). Execute any other command and then prompt the user again.

# Queries and Commands

These are the valid Queries:

Count MachineID prints out how many machines of type MachineID there are

Count PartID prints out how many parts of type PartID there are

HowMake PartID prints out information on how to make PartID like this:

To Make PartID

machine: MID

Q1 of PID1

Q2 of PID2

…

Where MID is the MachineID needed to make PartID, Q1 is how many of part PID1 is needed, Q2 is how many of PID2 is needed, etc. for however many parts are needed. The list of subparts needed should be sorted lexicographically.

CanMakeDirect PartID prints if there are enough machines and parts to make PartID with no other parts to be made (so the parts needed for PartID are available right now in the inventory)

CanMake PartID prints if it can make PartID, it will trace through the inbetween parts as well, but no parts are actually made

CountMake PartID as above, but prints how many can be made

ParallelMake PartID returns how many PartID can be made simultaneously (meaning duplicate machines and parts are available at all stages)

Boolean queries should print exactly “Yes” or “No” and numerical queries should just print their number. Then a newline is printed.

Commands:

Add PartID # adds # number of PartID to inventory

Add MachineID # adds # number of MachineID to machine list

Make PartID # attempts to make # number of PartID. This will update the counts of the various parts used and made. Further queries should now see this Part.

Quit: exits the program (for interactive mode)

Make PartID # prints out the number it was able to make (which may be 0).

Any other command or query should print “Invalid” and a newline.

# Grading and Extra Credit

For full credit, your system must be able to read the input files, validate them, and perform Count MachineID, Count PartID, CanMakeDirect PartID, Add PartID #, Add MachineID #, CanMake PartID, and Make PartID. CountMake partID and ParallelMake PartID are extra credit. Your code must be able to execute commands from a file and in interactive mode. This is actually quite simple—just put your ‘main’ code in a Sub. After reading data files and validating them, check for the command file. Then, either loop through the commands file calling your Sub or loop reading commands from STDIN and calling your Sub.

I suggest you consider recursion for CanMake.

Submit your final code in a file named components.pl. If you have additional .pm files submit them as well.