Enumerated Types and Unions

On the course homepage, you'll find a partial implementation of a source file, inventory.c, along with a sample input and a sample output file. You can also download these files using the following curl commands:

curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise22/inventory.c
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise22/input-1.txt
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise22/expected-1.txt

Inventory Representation

This exercise is intended to give us a little practice working with enumerated types and unions. The program will read a list of item descriptions from an input file given on the command line. Each item description includes a unique identification number (its SKU number) and a length, measured either in imperial units (inches) or metric units (meters). After reading in an item list, the program sorts all the items by length and them prints them out in the same format as they were given in the input.

The representation for the item list has already been created for you. It's an array of instances of a struct called Item. Each struct instance stores the SKU number for the item, along with its length. We use an enumerated type to remember whether each item is measured in metric or imperial units. Inside the struct, we also use a union to store the item's length. This union lets us store either:

- If the length is metric, we store it as a double value, representing a number of meters. This representation makes sense for a metric measurement, where different units are related by powers of 10.
- If the length is imperial, we store it as a mixed numeral, representing an integer number of inches and an additional fraction. This lets us record imperial measurements like you might normally, storing a length as something like 25 ¾ inches. You can assume that the denominator of this fraction is never zero (we won't test your code with invalid inputs like this).

The union inside the Item struct lets us use the same memory to represent the length either way, using it to store a double if that's what we need, or to store three integers instead, if that's what we need. Beware that the nesting of types gets a little deep for some of the fields. The item struct contains a union, and that union contains another, smaller struct as one of its fields. So, for example, if **item** is an item with a length measured in inches, then you'll need syntax like **item.length.denominator** to get to the denominator part of its length.

Inventory File Format

The input file looks like the following example. It starts with the number of item types. This is followed by a description of each item, one per line.

```
7
724 m 1.5
208 i 39 3 / 4
524 m 1.0
525 m 1.25
207 i 39 1 / 4
252 i 60 1 / 2
722 m 1.75
```

Each item is given by a unique Stock Keeping Unit (SKU) number. This is followed by the item's length. The length is given by the letter 'm' followed by a floating point number if it's in meters. If it's in inches, it's given as the letter 'i', followed by an integer number of inches followed by a fraction (two more integers with a '/' between them). You can assume the 'i' and 'm' letters are always lower-case and that the factional number of inches never has a zero in the denominator.

After your program sorts all the items, it will print out the list in the same format as above. For metric lengths, print them using the "%f" conversion specification, so you may get some additional fractional digits that

weren't there in the input. Once your program is working, you should be able to run it like:

```
$ ./inventory input-1.txt
7
207 i 39 1 / 4
524 m 1.000000
208 i 39 3 / 4
525 m 1.250000
724 m 1.500000
252 i 60 1 / 2
722 m 1.750000
```

Completing The Implementation

Much of the program is already written for you. You just need to add code in three areas to complete it:

- Near the start of main(), there's a loop to read all the item descriptions from the input file. You need to add code to read individual items. As you read each item, remember to set the enumerated type field (mtype) to record whether the length is metric or imperial. Then, fill in one of the members of the union to record the length.
- The program uses qsort() to sort the list of items. Inside the itemComp() function, you need to add code to compare the length of the two items you're given. Remember, your function should return negative if its first parameter has is shorter than the second parameter, zero if they're equal or positive if the first one is longer.
  Inside this function, you'll probably want to convert the lengths of the two items to metric in order to compare them. Don't change the representation of the items, just use some local variables temporarily hold the converted values. There's a preprocessor constant defined at the top of the program, if you want to use that to help with the conversion. Also, remember that the fractional part of the imperial length is represented as integers, so a division here will truncate (which you probably don't want; I made that mistake in writing my own solution).
- Near the end of main(), there's a loop to print out all the items. Fill in the code inside this loop to print each item, reporting the length in metric or imperial units like it was given in the input.

Submitting Your Work

When you're done, submit the source, inventory.c using the exercise_22 assignment on Moodle.