Honors Introduction to C (COP 3223H) Program 5
Pizza Shack Inventory and Finances

Objective
To give students practice writing a program with structs, functions and arrays, all interleaved.

The Problem: Pizza Shack Inventory and Finances
Pizza Shack periodically orders ingredients for their pizza. When pizza orders come in, they can only fill the order if they have enough of each ingredient for that particular order. If they don't, then they don't fill the order at all. In addition, for each order that is filled, Pizza Shack gets money, which they can then use to buy more ingredients. Obviously, the prices of the pizzas are more than the cost that the store pays for the raw ingredients; this is how Pizza Shack makes money!!!

In this program, you'll use structs to keep track of Pizza Shack's inventory, fill orders for pizzas, and keep track of the amount of money in the bank account. (We assume that the only transactions with the bank account are to take money from the account to buy ingredients and to put money into the account from sales.)

Since Pizza Shack processes many orders, you will read in all of the information about buying ingredients and processing orders from an input file.

Pricing Details
Each pizza has a base cost depending on its size as follows:

Small $7 Medium $10 Large $13 XLarge $16

Then, the first two toppings beyond the three required ingredients: dough, sauce and cheese, are free. Each additional topping is 50 cents. Thus, if a large pizza has 8 total ingredients (3 required ingredients plus 5 toppings), the price a customer pays for it is $13.00 + $1.50 = $14.50.

A pizza is sold only if enough of each ingredient is available.

Buying Ingredient Details
Each time an opportunity to buy ingredients comes up, they might be at a slightly different price. Also, to keep things simple each sale is fixed. Thus, you can't buy any number of pepperonis, you either have to buy the entire order or none of it. You will buy any order as long as you have the funds to do so. An order will be specified by the name of the ingredient, the number of units of it for the sale and the price per unit of the ingredient, for that purchase. Each ingredient name will be a unique lowercase identifier of 19 or fewer letters.

Individual Pizza Details

Each pizza that customers can buy has a name and a size. These combinations will be unique. In addition, each pizza will have a number of ingredients to make it, as well as list of those ingredients. Each ingredient will be a name and a quantity of number of units of that ingredient necessary to make that specific pizza. (For example, an XLarge pepperoni pizza might have 12 units of pepperoni while a Small pepperoni pizza might only have 5 units of pepperoni on it.) There will be no more than 100 pizza types total.

Implementation Requirements

You must use the following structs and constants:

```
#define MAXLEN 20
#define MAXPIZZATYPES 100
#define MAXINGREDIENTS 20

const float PRICES[4] = {7.00, 10.00, 13.00, 16.00};

typedef struct ingredient {
    char name[MAXLEN];
    int quantity;
} Ingredient;

typedef struct pizza {
    char name[MAXLEN];
    int numIngredients;
    Ingredient ingredientList[MAXINGREDIENTS];
    int size;
} Pizza;
```

Input File Specification (pizzasim.txt)

The first line of the file contains a single positive integer, *numPizzas (numPizzas ≤ 100)*, indicating the number of different pizza types. The descriptions of the pizzas will follow.

The first line of the description for each pizza will contain the name of the pizza (19 or fewer lowercase letters), followed by a space, followed by the size of the pizza (an integer 0, 1, 2 or 3, indicating small, medium, large and xlarge, respectively), followed by a space, followed by the number of ingredients, *numIngredients (3 ≤ numIngredients ≤ 20)* in the pizza. The following *numIngredients* lines will each contain information about the ingredients necessary to build this type of pizza. Each of these lines will have the name of the ingredient (19 or fewer lowercase letters), followed by a space, followed by the number of units of that ingredient needed for the pizza.

After the pizza descriptions, the next line will contain a single floating point number indicating the number of dollars currently in the bank account. (We assume that we start with no supplies.)

This will be followed by a line with a single positive integer, *numActions (numActions* ≤ *100000)*, indicating the number of actions to process.

Each process will be described in sequence, on a single line. Each process will either be buying an ingredient OR selling a particular number of a single type of pizza.

The format for a line describing a purchase of an ingredient is as follows:

The line will start with the key word "BUY" followed by a space, followed by the name of the ingredient (19 or fewer lowercase letters), followed by a space, followed by a positive integer representing the number of units of that ingredient, followed by a space, ending with a positive floating point number representing the total cost of the order. ***The input file will never require buying more than 20 different types of ingredients for the whole simulation.***

The format for a line describing the potential sale of a pizza is as follows:

The line will start with the key word "SELL" followed by a space, followed by the name of the pizza (19 or fewer lowercase letters), followed by a space, followed by a positive integer representing the number of that pizza for the order.

Output Specification (output to the screen)

For each of the actions in the input file, output a single line corresponding to the action.

For "BUY" actions, if enough money isn't available for the purchase, output the following line:

```
BUY could not be executed due to insufficient funds.
```

Otherwise, output a line with the following format:

```
Bought X units of Y. Now have Z units of Y in stock.
```

where X is the number of units of ingredient Y purchased, and Z is the number of units of ingredient Y in inventory after the purchase.

For "SELL" actions, if all the requisite ingredients are not available to create the pizza(s) for the order, output the following line:

```
SELL could not be executed due to insufficient ingredients.
```

Otherwise, output a line with the following format:

```
Sold X Y pizzas for $Z.
```

where X is the quantity of pizzas sold, Y is the name of the pizza and Z is the total money in dollars collected for the sale based on the pricing scheme previously described printed to two decimal places.

After outputting a line for each action, output a line indicating the amount of money left at the end of the simulation using the following format:

```
$X left after the end of the simulation.
```

where X is the number of dollars left printed to 2 decimal places.

Sample Input File

```
2
usual 1 3
dough 10
sauce 5
cheese 4
bigmeaty 3 6
dough 16
sauce 10
cheese 8
sausage 12
pepperoni 15
ham 14
1000.00
11
BUY pepperoni 1000 25.00
BUY sausage 200 20.00
BUY cheese 500 100.00
BUY sauce 500 50.00
BUY dough 1000 20.00
BUY sausage 10000 1000.00
BUY pepperoni 100 3.00
BUY ham 1000 20.00
SELL usual 20
SELL bigmeaty 5
SELL bigmeaty 12
```

Sample Output

```
Bought 1000 units of pepperoni. Now have 1000 units of pepperoni in stock.
Bought 200 units of sausage. Now have 200 units of sausage in stock.
Bought 500 units of cheese. Now have 500 units of cheese in stock.
Bought 500 units of sauce. Now have 500 units of sauce in stock.
Bought 1000 units of dough. Now have 1000 units of dough in stock.
BUY could not be executed due to insufficient funds.
Bought 100 units of pepperoni. Now have 1100 units of pepperoni in stock.
Bought 1000 units of ham. Now have 1000 units of ham in stock.
Sold 20 usual pizzas for $200.00.
Sold 5 bigmeaty pizzas for $82.50.
SELL could not be executed due to insufficient ingredients.
$1044.50 left after the end of the simulation.
```

Note: More substantial examples will be posted online at a later date.

NOTE: PLEASE TRY TO MATCH THE OUTPUT GIVE EXACTLY, AS GRADING WILL BE BASED ON PERFECTLY MATCHING MY OUTPUT.

Restrictions

Name the file you create and turn in *pizzasim.c*. Although you may use other compilers, your program must compile and run using gcc in Code::Blocks. Your program should include a header comment with the following information: your name, course number, section number, assignment title, and date. You should also include comments throughout your code, when appropriate. If you have any questions about this, please see a TA.

Deliverables

A single source file named *pizzasim.c* turned in through WebCourses.