CSCI 103 Introduction to Programming Spring 2015

Programming Exam

This is a 90-minute programming exam. It is semi-open book:

- You are allowed to read the following offline materials: past programs that you have written (including assignments, labs, and homework), the course textbook, and any of your notes.
- You are allowed to read the following online materials: the course website (including Websheets exercises), the course Piazza forum, and library reference pages from `cplusplus.com` and `cppreference.com`.
- You are not allowed to read any other materials.
- You are not allowed to communicate with anybody other than course staff.

Complete the programs on your laptop. You will upload them to the course website for submission (the link is inside this exam). Like a programming assignment: you can run some basic tests through the website, though they might not exhaustively check for all bugs; you can upload your work as many times as you like, but only the last submission will be graded.

We will primarily grade your code on correctness. Code that compiles will receive a bonus compared to code that does not. *Please write your name at the top of every file you submit.* Other than this, we will not be picky about style, but we recommend that you use good style because it will help you debug your code more quickly, and help us read your code.

Print your name, print your email address, and select your lecture section now. **You must hand this exam booklet back to the course staff at the end of the exam.**

Your Name: _____

Your USC e-mail: _____

Your Lecture Section: _____

29991	2PM MW	David Pritchard
29902	11AM TTh	David Pritchard
29906	2PM TTh	David Pritchard

Part 1: Recipe Scaling (17 points)

While studying at USC, you discover that you really like cooking. Because of this, you have many recipes on your computer. Here is one, named `bread.txt`:

```
3 cup_flour
1.25 tsp_salt
0.25 tsp_yeast
1.25 cup_water
```

Write a program `scale.cpp` that makes a recipe larger or smaller by multiplying all of the amounts by a given factor and printing the result. For example, `./scale 2 bread.txt` should print on the screen a recipe where all of the numbers have been multiplied by 2:

```
6 cup_flour
2.5 tsp_salt
0.5 tsp_yeast
2.5 cup_water
```

You must define the following function in `scale.cpp`, which should be useful:

```
// open this file & read its contents into parallel arrays (numbers, items)
// write the number of items at the address where n points
void read(const char filename[], int* n, double numbers[], string items[])
```

For example, if we pass in `"bread.txt"` as the filename, then the number 4 should be written at the address held by the pointer `n`, and the first 4 entries of the arrays should be filled in.

There will be fewer than 100 ingredients. Your `main` should statically allocate the parallel arrays and pass those arrays to the `read` function. Dynamic memory is not useful here.

See page 4 for sample tests and outputs. You can get these files and submit your code at

`http://bits.usc.edu/exam/recipes`

We recommend you upload your Part 1 code before starting Part 2.

Here are some additional clarifications and suggestions:

- Every line of input will contain a number, a space character, and then the item name.
  - Item names won't contain spaces.
  - Use `>>` to read the input. The functions `.get()` and `.getline()` are inconvenient.
  - You may assume that the named file exists and has the correct format.
  - You also may assume that the user gives the correct command-line arguments.
  - To determine when you've reached the end of the file, `.fail()` will be useful.
- The scale factor need not be an integer. E.g., `./scale 0.5 bread.txt` should work.
- There's no need to use output manipulators. Just use the default output format.
- Using `char*`s instead of `string` objects would make the `read` function very challenging. This approach will receive only partial credit and should only be done as a last resort.

Part 2A: Going Shopping (10 points)

In this part of this exam you will write a program `cost.cpp` that computes how much a recipe will cost at a store. Suppose `ralphs.txt` contains the following list of prices for items:

```
0.04 tsp_salt
0.50 cup_milk
0.10 cup_flour
0.30 egg
0.16 cup_water
0.40 tsp_yeast
0.12 tbsp_butter
```

Then if we run `./cost bread.txt ralphs.txt` it should print out `0.65` which represents

$$3 \times 0.10 + 1.25 \times 0.04 + 0.25 \times 0.40 + 1.25 \times 16 = 0.65,$$

the total price of the flour + salt + yeast + water in the recipe.

You should copy-and-paste your `read` function from part 1 into your part 2 code. (A library would be stylistically better, but makes compiling harder, so we won't do that.)

For Part 2A, you may assume that the store has all the items you're looking for. However, the order listed in the store's inventory may be different from the recipe. Do a simple linear brute-force search. You should not be sorting or binary searching.

You can assume that no file lists the same item twice. Assume that the inventory has less than 100 items. Like Part 1, item names will not contain spaces.

Part 2B: The Missing Ingredient (3 points)

Finally, modify your Part 2A code to allow for the fact that the store might be out of some item(s). Print out `impossible` instead of a number if the store is missing anything. See the next page for examples.

You may want to keep a copy of your working Part 2A solution just in case you break it while trying to complete Part 2B. Sample tests are given on the next page.

Remember to test and upload your code! To get testing files and submit your work, visit

`http://bits.usc.edu/exam/recipes`

Here are three sample recipes (you can download them online):

bread.txt	frenchtoast.txt	crepes.txt
3 cup_flour	1 cup_milk	2 egg
1.25 tsp_salt	3 egg	0.75 cup_milk
0.25 tsp_yeast	2 tbsp_honey	0.5 cup_water
1.25 cup_water	0.25 tsp_salt	1 cup_flour
	8 slices_bread	3 tbsp_butter
	4 tbsp_butter	

For part 1, we expect the following outputs:

./scale 2 bread.txt	./scale 0.5 frenchtoast.txt	./scale 100 crepes.txt
6 cup_flour	0.5 cup_milk	200 egg
2.5 tsp_salt	1.5 egg	75 cup_milk
0.5 tsp_yeast	1 tbsp_honey	50 cup_water
2.5 cup_water	0.125 tsp_salt	100 cup_flour
	4 slices_bread	300 tbsp_butter
	2 tbsp_butter	

Here are two sample inventories (you can download them online):

ralphs.txt	costco.txt
0.04 tsp_salt	0.05 tbsp_honey
0.50 cup_milk	0.10 slices_bread
0.10 cup_flour	0.08 tbsp_butter
0.30 egg	0.10 cup_water
0.16 cup_water	0.15 cup_flour
0.40 tsp_yeast	0.60 cup_milk
0.12 tbsp_butter	0.02 tsp_salt
	0.20 egg

For part 2A, we expect the following outputs:

`./cost bread.txt ralphs.txt`	should print `0.65`
`./cost crepes.txt ralphs.txt`	should print `1.515`
`./cost frenchtoast.txt costco.txt`	should print `2.425`
`./cost crepes.txt costco.txt`	should print `1.29`

For part 2B, we expect the following outputs:

`./cost bread.txt costco.txt`	should print `impossible`
`./cost frenchtoast.txt ralphs.txt`	should print `impossible`