Assignment 5: Recipes

Dates:

Assigned: 2013-07-18 Thursday
Due: **2013-08-01** Thursday at 11:59 pm

Questions – email rjmfff@mail.umkc.edu

Make sure that your source code is *committed* and *synced* to your student GitHub account. You do not need to turn in the code via Blackboard or anything else.

I will no longer answer questions on assignments on the day they are due, so start it BEFORE Thursday!

Specification:

Assignment 5 is a small program that contains an STL vector and map item. We will use these containers, as well as iterators to loop through our data.

Step-By-Step

This program will not contain any classes unless you choose to add them. We will have main() and several functions to handle different items. We will have a menu in main() that is displayed each time through the program loop.

Includes:

- iostream
- fstream
- map
- vector
- string

Function declarations:

- `void DisplayMenu();`
- `void AddRecipe( map< string, vector<string> >& recipes );`
- `void ExportRecipes( map< string, vector<string> >& recipes );`

Functionality

main()

In main(), start by declaring a map. Remember that a **map** is a key-value pair. Our keys will be a **string**, while our values will be a **vector of strings**.

The key is the name of the recipe, while the vector of strings is our list of ingredients.

```
map< string, vector<string> > recipes;
```

NOTE: If you do not have white-space between > and > above, you will get a compile error!

Create our standard sort of main menu – while the boolean flag "done" is false, keep looping (and, therefore, keep running the program).

Within the while loop, call DisplayMenu(). We will go over the contents of DisplayMenu() later.

After calling DisplayMenu(), get the user's choice (an integer), and then write an if statement for different options:

1. Add Recipe function is called
2. Export Recipes function is called
3. done is set to true

That's all we need for main().

void DisplayMenu()

Display Menu should show the following options:

1. Add New Recipe
2. Export Recipes
3. Quit

void AddRecipe(map< string, vector<string> >& recipes)

First, ask the user for a recipe name and store this in a variable (e.g., “name”).

Create a “done” boolean flag and set it to false.

Create a while loop that will continue looping until that done flag is true.

Within the while loop, prompt the user to enter an ingredient & amount, or “done” to stop adding ingredients.

Use the **getline** function to get a line of text, allowing for spaces.

```
getline( cin, ingredient, '\n' );
```

If the user enters an ingredient (rather than “done”), we are going to push this onto our vector...

```
recipes[ name ].push_back( ingredient );
cout << "Added \"" << ingredient << "\"." << endl << endl;
```

Notice that the recipe name that the user entered is being used as the key in our **recipes** map.

void ExportRecipe(map< string, vector<string> >& recipes)

Within ExportRecipe, first create an **ofstream** item and open a text file, “Recipes.txt”.

This function is going to iterate through all recipes and all ingredients within a recipe and print it to the file.

Do you remember the syntax for an iterator for-loop?

A for-loop is structured like:

for (declare new variable and initialize it; loop while this condition is true; increment by this)

The iterator is the same format, but we are declaring an ***iterator*** variable rather than an integer.

```
for ( map< string, vector<string> >::iterator recipe =
     recipes.begin();
          recipe != recipes.end();
          recipe++ )
{
     outfile << endl << endl << recipe->first << endl;

     // Iterate through all the ingredients and print
}
```

Declaring our iterator, and initializing it to “recipes.begin()”:

map< string, vector<string> >::iterator recipe = recipes.begin();

An Iterator type is specific to the data structure we're using. Since our **recipes** map is of type map< string, vector<string> >, our iterator is of type **map< string, vector<string> >::iterator**.

Our iterator's name is “recipe”. We are setting it equal to the beginning of our **recipes** map.

The ending condition is when our iterator **recipe** is equal to the map **recipes**' ending.

recipe != recipes.end();

While iterator recipe is not the end, we will continue looping.

Then, after each time through the loop, we will more the iterator *forward* to the next item in the map. We can use the ++ operator for this:

recipe++

Above is the code for iterating through each item in the **map.**

recipe->first will be the **Recipe's Name**

recipe->second will be the **Vector of ingredients**

We can output each recipe name now, but we aren't outputting each ingredient. We need a second iterator for-loop for this.

Can you write the second iterator on your own?

Hints:

- Our list of recipes is a **vector<string>**.
- Our iterator will start at the value:

```
recipe->second.begin();
```

- And, because this is a simple **string** item, we can output each ingredient by de-referencing it:

```
outfile << "\t" << *ingredient << endl;
```

If you need help, try to look at examples online, but also feel free to email me.

When you're done outputting all recipes and their ingredients, close the ofstream file.

The program should now be done.

Sample Screenshots

Adding Recipes:

```
MAIN MENU
1. Add new recipe
2. Export recipes
3. Quit
>> 1


ADD RECIPE
Enter recipe name: Enter new ingredient and amount. E.G. "1 cup flour"
or type "done" when done.
Chocolate Chip Cookie
Added "Chocolate Chip Cookie".

Enter new ingredient and amount. E.G. "1 cup flour"
or type "done" when done.
1 cup butter
Added "1 cup butter".

Enter new ingredient and amount. E.G. "1 cup flour"
or type "done" when done.
1 cup white sugar
Added "1 cup white sugar".

Enter new ingredient and amount. E.G. "1 cup flour"
or type "done" when done.
done


MAIN MENU
1. Add new recipe
2. Export recipes
3. Quit
>>
```

Sample output recipes:

```
Cake
     * 1 cup salt
     * 2 cups lemons

Cookies
     * 3 teaspoons cookie mix
     * 1 cup cayenne powder
```