# CS 474: Object-oriented Programming languages and environments

*C++ Project #2*

**Due time:** 5/3/2014 at 7:00 pm
Instructor: Ugo Buy
TA: Venkat Srinivasan

You are still working for *Talk Small & Hack Big*, the company that rolled the EROS product earlier on. This time the company wants to develop a Customer Order System (COS), which allows waiters to enter information about dishes ordered by patrons in a restaurant. Customers can only order food items from a limited selection listed below. However, each patron can order the same item multiple times, possibly prepared in different ways, and you must keep track of each item individually. Thus, the *food order list* of each customer can potentially have unlimited length. In addition, you must keep track of multiple order lists simultaneously. Your restaurant has a capacity of 10 patrons; your COS must be able to handle that many lists at all times. Finally, food items in each list should be kept in the order in which they will be served.

Here is a description of food items. Each food item will have a name (a C-style string), a price (a double), and a calorie count (an integer). Restaurant seats are numbered, and each food item list should indicate the seat number of the patron who ordered the list (i.e., an integer from 1 to 10). In addition, each of the food items adds the following items of information:

1. *Salmon* — This food item adds an enumerated type describing how it should be cooked (i.e., rare, medium, well-done), and another C-style string describing the side dish to included (e.g., mashed potatoes, baked potato, rice pilaf, etc.)

2. *Turkey sandwich* — This food item adds an enumerated type describing the kind of bread for the sandwich (i.e., white, whole wheat, or rye), and a list of condiments to be added to the sandwich (e.g., lettuce, tomato, mayonnaise, mustard, provolone cheese, etc.) The list of condiments may have unlimited length.

3. *Eggplant casserole* — This food item adds an integer describing the number of ounces in the serving and a C-style string describing the side dish, similar to the case of the salmon above.

Use a command line interface for entering the commands below. Your command line interface will prompt the user for a command, and then execute the command. Here is a list of commands. Make sure not to cause any memory leaks or dangling pointers in the implementation of these commands.

- *n — New_List.* This operation allows a user (i.e., the restaurant owner) to create a new food order list. COS prompts the user (i.e., a waiter) for the seat number of the patron requesting the list and an empty list is displayed on a the user's screen. An error is returned if the input seat number was already assigned to a patron, that is, the seat is not empty.

- *a — Add_List_Entry.* This operation allows a user to add a new food item to a food item list. The user is prompted for the patron number, and the type of food item (from the 3 above possibilities). Next, the user is prompted for the additional items pertaining to each food type. Upon completion of this interaction, the updated list is displayed on the user's screen. The new entry is assumed to be added at the end of the list.

- *c — Copy_List.* This operation allows a user to copy a food list from a patron to another patron. The user is prompted for the numbers of the two patrons. Next, the original food item list is copied to the destination patron. The two lists should not share any data structures at all. (This means that the first list is deep copied into the second list.) The two lists are displayed on the user's screen.

- *d — Delete_List.* When a patron leaves the restaurant, the corresponding food item list is deleted completely. Make sure not to cause any memory leaks. The seat of the patron is made available for new patrons.
- *l — List_patron.* The food order list of a given patron is displayed. The user is prompted for the patron number. An appropriate message is displayed if the input number does not have a food item list (e.g., because that seat is empty).
- *s — Swap_orders.* The command swaps the orders of two restaurant patrons. The user is prompted for the seat numbers of the two patrons and the corresponding lists are swapped. The new orders for the two patrons are displayed.
- *q — quit.* The COS is exited.

You must make use of an abstract superclass *FoodItem* with three concrete subclasses corresponding to the three foods above. These classes must implement a virtual deletion and virtual copying schemes. Make sure to code the big-three for each of the classes. You are at liberty to choose an appropriate data structure to hold the 10 food item lists.

*You must work alone on this project.* Submit this project using the digital dropbox feature of the Blackboard web page. No late submissions will be accepted.