Restaurant Simulation

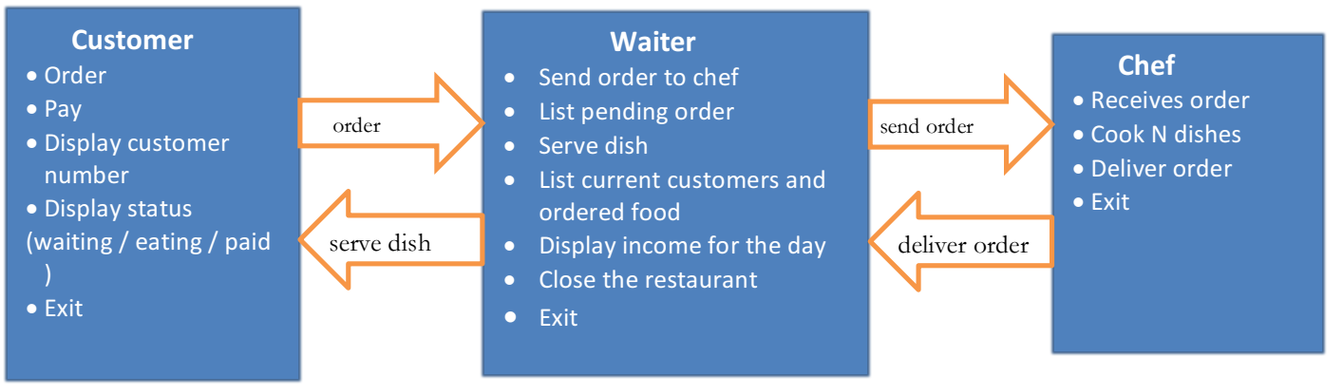
(Due: **Dec. 7, 11:59 PM**)

This project aims to simulate restaurant's daily activities. For this project, there are customers, a waiter and a chef. A customer orders food thru the waiter. The waiter takes the order and send it to the chef. The chef prepares the ordered dish and once cooked, delivers the dish to the waiter and serves it to the customer.

PROGRAM REQUIREMENTS:

Below is the Main Menu and possible actions and interactions of three entities in a restaurant, namely, Customer, Waiter, Chef. The program starts and exits with the Main Menu. The Main Menu allows the user to control the actions of the 3 entities.

Main Menu
- Customer
- Waiter
- Chef
- Exit



Here are the description of actions of each entity.

Customer.

- Order. Gives order to the waiter. A maximum of 3 dishes can be ordered. Order will not be taken when restaurant is full, i.e. a maximum of 5 customers at any given time, or when the restaurant is close.
- Pay. The customer pays the bill. The bill must be shown when this option is shown.
- Display Status. Status can be **waiting, eating** or **paid.** A customer is **waiting** when all or some of the ordered food are still pending (not cooked yet). **Eating** when all ordered food had been cooked and delivered by the waiter and **paid** when the customer has paid the ordered food. We assume that the customer leaves the restaurant right after he pays the bill.
- Exit. Exits customer action and returns to the Main Menu.

Waiter

- Send Order to Chef. The ordered food from the customer are added to the list of pending order and at the same time sent to the chef. Most recent order is added last in the list.
- List Pending Order. Displays ordered dish that are not prepared/cooked yet with the corresponding dish # and customer # who ordered the dish.
- Serve Dish. Serves the cooked ordered dish (delivered by the chef) to the customer and removes the dish from the list of pending order.
- List Current Customers and ordered food. Displays customer number and ordered food of current customers (who are still in the restaurant)
- Display Income for the Day. Displays the number of customers who paid and the total amount paid.
- Close the restaurant. Waiter opts to close the restaurant whether it has reached the maximum number of customers (up to 20) or not.
- Exit. Exits waiter action and returns to the Main Menu.

Chef

- Cook N dishes. The chef chooses the first N dishes from the list of pending order and cooks them. N must be inputted where N<=3.
- Deliver Order. The chef delivers the cooked dishes to the waiter.
- Exit. Exits chef action and returns to the Main Menu.

Other elements:

Dish. Each dish has a code and price.
Customer Details. Customer #, Ordered dishes, each of which has a dish code, dish code, dish price, dish status (pending or served) and total bill.

Limitations:

- The restaurant closes only when it has prepared a maximum of 20 order of dishes, has served a maximum of 10 customers, or the waiter chooses to close the restaurant.
- The restaurant can only accommodate a maximum of 5 customers at any given time and a maximum of 10 customers per day.

IMPLEMENTATION, DOCUMENTATION & TESTING REQUIREMENTS:

1. **Coding Standard.** Your program code must be implemented using the standard C language compiler ANSI C/DEV C++. Linux coding standard must be followed (https://developer.gnome.org/programming-guidelines/stable/c-coding-style.html.en)

2. **Functions.** It is best that you perform your coding "incrementally." This means:
   - Dividing the program specification into subproblems, and solving each problem separately according to your algorithm;
   - Code the solutions to the subproblems one at a time. Once you're done coding the solution for one subproblem, apply testing and debugging.

3. **Documentation.** Internal documentation must be employed.

   You are expected to have the following:
   - File comments or Introductory comments
   - Function comments

   File comments are found at the very beginning of your program before the preprocessor directives. Follow the format shown below. Note that items in between < > should be replaced with the proper information and items in between [ ] are considered to be optional parts.

```
/*   Programmed by:    <your name here>   <section>
     Description:      <Describe what this program does briefly>
     Last modified:    <date when last revision was made>
    [Acknowledgements: <list of sites or borrowed libraries and sources>]
*/
<Preprocessor directives>
<function declarations>
int
main()
{
}
```

   Function comments precede the function declaration. These are used to describe what the function does and the intentions of each parameter, if any. Follow the format below when writing function comments:

```
/*   <Description of function>
     [@param  (<type>)  <name>  <purpose>]
     [@return  (<type>) purpose]
*/
<type>
<function name> (<parameter list>)
```

Examples:

```
/*   This function updates the totals of carbon footprint
     @param  (float*)  total_income  is the
     @param  (float*)  total_customers is the

      :
     @return (void) no return value
*/
void
updateTotals  (float * total_income,
               float * total_ customers,
               … )
        { …
        }
```

4. **Test Cases**. Submit the list of test cases you have used. For each feature of your program, you have to fully test it before moving to the next feature. Sample questions that you should ask yourself are:
   a. What should be displayed on the screen if the user inputs an answer?
   b. What would happen if I input the borderline values (minimum and maximum values of a given range)?
   c. What would happen if I input incorrect inputs? (i.e. non -arrow keys)
   d. Is my program displaying the correct output?
   e. Is my program following the correct sequence of events (correct program flow)?
   f. Is my program terminating (ending/exiting) correctly? Does it exit when I press the command to quit?
   g. Does it exit when the program's goal has been met? Is there an infinite loop?
   h. and others…

5. **Readability and Understandability.** Your source code must:
   a. be readable and easy to understand
   b. have well-defined functions
   c. define meaningful variable names and functions
   d. include proper comments

SUBMISSION REQUIREMENTS:

1. You are to submit the following requirements online via our canvas online classroom.
   a. Brief discussion about the machine project (in your words). Maximum of 1 page.
   b. Source code of your program with inline documentation
   c. Test cases with actual screen capture and expected/correct output
   d. Executable file

2. **Deadline of Canvas online submission: Dec. 7, 11:59 PM.** Failure to submit on time will merit a 0.0 for the MP.

3. As a backup copy, email your source code to your **dlsu account**. Your email should be sent before the deadline.

4. You will demonstrate your project on a specified schedule during the last weeks of classes. Being unable to show up during the demo, or being unable to answer convincingly the questions during the demo will merit a grade of 0.0 for the MP.

5. Any requirement not fully implemented and instruction not followed will merit deductions.

6. See the CCPROG1 syllabus for the rubric for the evaluation of the project.

7. This is an individual project. Working in collaboration, asking other people's help, and/or copying other people's work are considered as cheating. Cheating is punishable by a grade of 0.0 for CCPROG1, aside from which, a cheating case will be filed with the Discipline Office.