Restaurant Food Delivery

(Due: **Dec. 5, 0800**)

This project aims to simulate a restaurant's food delivery services. For this project, there are customers, a restaurant manager and a chef. A customer orders food by calling the restaurant. The manager takes the call, receives the order and send it to the chef. The chef cooks the ordered dish, packs the order and sends it to the manager who will ask the driver to deliver the food to the customer.

PROGRAM REQUIREMENTS:

Below is the Main Menu and possible actions of three entities in a restaurant, namely, Customer, Manager, 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
- Manager
- Chef
- Exit

Customer
- Order
- Pay
- Display customer number
- Display status (booking/ waiting / received order)
- Exit

Manager
- Display Menu for the Day
- Send order to chef
- List pending order
- List current customers and ordered food
- Sends a driver to deliver food
- Display income for the day
- Display Dishes Served for the Day
- Close the restaurant
- Exit

Chef
- Receives order
- Cook N dishes
- Pack the order
- Exit

Here are the description of actions of each entity.

Customer.

- Order. Calls the restaurant and place the order. A maximum of 3 dishes can be ordered. Order will not be taken when the restaurant is accommodating many customers, i.e. a maximum of 5 customers at any given time, or when the restaurant is close.
- Pay. The customer pays the bill via e-wallet. The bill must be shown when this option is shown.
- Display Status. Status can be **booking, waiting** or **received order.** The status is **Booking** when the customer is placing the order to the restaurant. The status is **Waiting** when the order is paid and the food is still being prepared by the restaurant. The status is **Received Order** when all ordered food had been cooked and delivered by the driver.
- Exit. Exits customer action and returns to the Main Menu.

Manager

- Display the Menu for the Day. Displays the menu of dishes for the day and the price.
- 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 number code and customer number who ordered the dish.
- List Current Customers and ordered food. Displays customer number and ordered food of current customers. Previous customers should not be displayed.
- Send Driver. Sends the driver to deliver food to the customer.
- Display Income for the Day. Displays the number of customers who paid, the total amount paid, and the total income for the day.
- Display the Dishes Served for the Day. Displays the number of ordered dishes for the day with code and the price per dish.

- Close the restaurant. Manager opts to close the restaurant whether it has reached the maximum number of customers (up to 10) or not.
- Exit. Exits waiter action and returns to the Main Menu.

Chef

- Receives Order. Commands the Chef to accept the order from the Manager. This must be chosen first before *Cook N Dishes* task.
- Cook N dishes. The chef chooses the first N dishes from the list of pending order and cooks them. N must be inputted at the beginning of the program where N can be from 1 to 3.
- Pack Food. Packs the cooked ordered dish and sends to the manager. This should remove the dish from the list of pending order
- Exit. Exits chef action and returns to the Main Menu.

Other elements:

Dish. Each dish has a number code and price.
Customer Details. Customer number, Ordered dishes, each of which has a dish code, dish price, dish status (cooking or packed) 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 manager 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.

How to Approach the Machine Project

Step 1: Problem analysis and algorithm formulation

Read the MP Specifications again! Identify clearly what are the required information from the user, what kind of processes are needed, and what will be the output (s) of your program. Clarify with your professor any issues that you might have regarding the machine project.

When you have all the necessary information, identify the necessary functions that you will need to modularize the project. Identify the required data of these functions and what kind of data they will return to the caller. Write your algorithm for each of these modules/functions as well as the algorithm for your main program.

Step 2: Implementation

In this step, you are to translate your algorithm into proper C statements. While implementing, you are to perform the other phases of program planning and design (discussed in the other steps below) together with this step.

Follow the coding standard indicated in the course notes (Modules section in AnimoSpace).

You may choose to type your program in a text editor or an IDE (i.e. Dev-C IDE) at this point. Note that you are expected to use statements taught in class. You can explore other libraries and functions in C as long as you can clearly explain how these work. You may also use arrays, should these be applicable and you are able to properly justify and explain your implementation using these. For topics not covered, it is left to the student to read ahead, research, and explore by himself.

Note though that you are NOT ALLOWED to do the following:

- to declare and use global variables (i.e., variables declared outside any function),
- to use goto statements (i.e., to jump from code segments to code segments),
- to use the break or continue statement to exit a block. Break statement can only be used to break away from the switch block,
- to use the return statement or exit statement to prematurely terminate a loop or function or program,
- to use the exit statement to prematurely terminate a loop or to terminate the function or program, and
- to call the main() function to repeat the process instead of using loops.

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.

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

Documentation

While coding, you have to include internal documentation in your programs. You are expected to have the following:

- File comments or Introductory comments
- Function comments
- In-line comments

Introductory 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.

```
/*
     Description:      <Describe what this program does briefly>
     Programmed by:    <your name here>   <section>
     Last modified:    <date when last revision was made>
     Version:          <version number>
    [Acknowledgements: <list of sites or borrowed libraries and sources>]
*/
<Preprocessor directives>

<function implementation>

int main()
{
   return                                                              0;
   }
```

Function comments precede the function header. These are used to describe what the function does and the intentions of each parameter and what is being returned, if any. If applicable, include pre-conditions as well. Pre-conditions refer to the assumed state of the parameters. Follow the format below when writing function comments:

```
/*    <Description of function>
      Precondition: <precondition / assumption>
      @param <name> <purpose>
      @return <description of returned result>
*/
<return type>
<function name> (<parameter list>)
      :
```

Example:

```
/* This function computes for the area of a triangle
   Precondition: base and height are non-negative values
   @param base is the base measurement of the triangle in cm
   @param height is the height measurement of the triangle in cm
   @return the resulting area of the triangle
*/
float
getAreaTri (float base,
            float height)
{
    ...
}
```

In-Line Comments are other comments in major parts of the code. These are expected to explain the purpose or algorithm of groups of related code, esp. for long functions.

Step 3: Testing and Debugging

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:

1. What should be displayed on the screen if the user inputs an order?
2. What would happen if I input incorrect inputs? (e.g., values not within the range)
3. Is my program displaying the correct output?
4. Is my program following the correct sequence of events (correct program flow)?
5. Is my program terminating (ending/exiting) correctly? Does it exit when I press the command to quit? Does it exit when the program's goal has been met? Is there an infinite loop?
7. and others…

Important Points to Remember:

1. You are required to implement the project using the C language (C99 and NOT C++). Make sure you know how to compile and run in both the IDE (DEV-C++) and the command prompt (via
   `gcc –Wall <yourMP.c> -o <yourExe.exe>`
2. The implementation will require you to:
   - Create and Use Functions

   Note: Non-use of self-defined functions will merit a grade of **0** for the **machine project**. Too few self-defined functions may merit deductions. A general rule is to create a separate function for each option described above, unless some features are too similar that one function can serve the purpose for two [or more] of the options. Note that functions whose tasks are only to display are not included in the count for creating user-defined functions.
   - Appropriately use conditional statements, loops and other constructs discussed in class (Do not use brute force solution. **You are not allowed to use goto label statements, exit statements. You are required to pass parameters to functions and not allowed to declare global or static variables.**)
   - Consistently employ coding conventions
   - Include internal documentation (i.e., comments)
3. Deadline for the project is the **8:00AM of December 5, 2022 (Monday)** via submission through **AnimoSpace**. After this time, submission facility is locked and thus no MP will be accepted anymore and this will result to a **0.0** for your machine project.
4. The following are the deliverables:

Checklist:
❒ Upload in AnimoSpace by clicking **Submit Assignment** on Machine Project and adding the following files:
❒ source code*
❒ test script**
❒ email the softcopies of everything as attachments to **YOUR own email address** on or before the deadline

Legend:

*****Source Code also includes the internal documentation. The **first few lines of the source code** should have the following declaration (in comment) **BEFORE** the introductory comment:

```
/*******************************************************************************
This is to certify that this project is my own work, based on my personal efforts in studying and
    applying the concepts learned. I have constructed the functions and their respective
    algorithms and corresponding code by myself. The program was run, tested, and debugged
    by my own efforts. I further certify that I have not copied in part or whole or otherwise
    plagiarized the work of other students and/or persons.
                                                        <your full name>, DLSU
    ID# <number>
            *******************************************************************************
    */
```

Test Script should be in a table format, with header as shown below. There should be **at least 3 distinct test classes (as indicated in the description) **per function**. There is no need to create test scripts for functions that only perform displaying on screen.

Function Name		Test Description	Sample Input (either from the user or to the function)	Expected Result	Actual Result	
getAreaTri		**base and height measurements are less than 1.**	**base = 0.25 height = 0.75**	**…**	**…**	
		:::				

Test descriptions are supposed to be unique and should indicate classes/groups of test cases on what is being tested. Given the function getAreaTri(), the following are 3 distinct classes of tests:

i.) testing with base and height values smaller than 1
ii.) testing with whole number values for base and height
iii.) testing with floating point number values for base and height, larger than 1.

The following test descriptions are incorrectly formed:

Too specific: testing with base containing 0.25 and height containing 0.75
Too general: testing if function can generate correct area of triangle
Not necessary -- since already defined in pre-condition: testing with base or height containing negative values

5. **MP Demo:** You will demonstrate your project on a specified schedule during the last weeks of classes. Being unable to show up on time during the demo or being unable to answer convincingly the questions during the demo will merit a grade of **0.0** for the **MP**. The project is initially evaluated via black box testing (i.e., based on output of running program). Thus, if the program does not compile successfully using gcc -Wall and execute in the command prompt, a grade of 0 for the project will be incurred. However, a fully working project does not ensure a perfect grade, as the implementation (i.e., correctness and compliance in code) is still checked.
6. Any requirement not fully implemented and instruction not followed will merit deductions.
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 course, aside from which, a cheating case may be filed with the Discipline Office.
8. The above description of the program is the basic requirement. A maximum of 10 points will be given as bonus. Use of colors and other aesthetics may not necessarily incur bonus points. Sample additional features could be:
   (a) use of arrays or dynamic lists. This may require you to read ahead or research on how to apply these properly to your project.

Note that any additional feature not stated here may be added but **should not conflict with whatever instruction was given in the project specifications**. Bonus points are given upon the discretion of the teacher, based on the difficulty and applicability of the feature to the program. Note that **bonus points can only be credited if all the basic requirements are fully met** (i.e., complete and no bugs).

HONESTY POLICY AND INTELLECTUAL PROPERTY RIGHTS

Honesty policy applies. Please take note that you are NOT allowed to borrow and/or copy-and-paste – in full or in part any existing related program code from the internet or other sources (such as printed materials like books, or source codes by other people that are not online). **You should develop your own codes from scratch by yourself.**