SEFL-ASSESSMENT

FOR MATLAB PROJECT

STUDENT NAME: DO PHONG TRAN

STUDENT ID: 1873825

PROJECT NAME: INTERACTIVE CAFÉ MENU

I. Conceptual coverage (20/20)

All concepts in the courses from week 1 to week 6 are covered and correctly demonstrated.

1. Input/Output:

- **Input:**

```
% take customer name in string
name=input("Enter customer 's name:","s");

% take customer phone number in string
phone=input("Enter customer 's phone number:","s");
% ask customer for the right syntax: 10 digits number
while isnan(str2double(phone)) || strlength(phone)>10
    phone=input("Enter a real phone number:","s");
end
```

- **Output:**

```
% prepare a text file
receiptText = fopen("receipt.txt",'w');
% format text file
fprintf(receiptText, "*************************************************\n" + ...
    "                Morningstars Coffee\n\n\n" + ...
    "                      RECEIPT\n " + ...
    "              %s \n\nCustomer: %s\nContact number: %s\n\n", time, upper(name), phone);
fprintf(receiptText, orderList);
fprintf(receiptText,  "\n\nYour order costs: AUD %.2f\n " + ...
    "                    Thank you!\n" + ...
    "*************************************************", total);
fclose(receiptText);
```

2. Vectors, 2D Array

- **Vectors:**

```
% string vector contains dishes
menu=["Vietnamese Coffee", "Espresso", "Americano", "Cappuchino", "Latte", "Hot chocolate", "Tiramisu", "Ham & Cheese Croissant", "French Toast", "Blue berries cake"];
% vector contains each dish's price
price=[8.00 9.50 9.50 10.50 11.00 10.50 11.50 9.90 13.99 12.99];
% vector contain number represent dishes
allDish=[];
% vector contain quantity
quantity=[];
```

- **2D Array:**

```
% order string matrix
order=[];
for ii=1:length(dishName)
    % convert nums to string
    order=[order; dishName(ii), string(quantity(ii)), string(dishTotalPrice(ii)) ];
end
```

3. Loops

- **For loop:**

```
% order's total money
totalCash=0;
% vector contains each item total price
dishTotalPrice=[];
for i=1:length(quantity)
    % calculate order total money
    totalCash=totalCash+price(allDish(i))*quantity(i);
    % calculate and store each item total price
    dishTotalPrice=[dishTotalPrice price(allDish(i))*quantity(i)];
end
```

- **Nested loops:**

```
% match numbers represent dishes and dishes' names
dishName=[];
for m=1:length(allDish)
    for n=1:length(menu)
        if allDish(m)==n
            dishName=[dishName menu(allDish(m))];
        end
    end
end
```

- **While loop:**

```
% take input
finish=input("Do you want to finish your order? (Yes=y; No=n)","s");
% allow "y" for yes and "n" for no only
while finish~="y" && finish~="n"
    disp("Please enter the right systax!");
    finish=input("Do you want to finish your order? (Yes=y; No=n)","s");
end
```

4. Conditional execution

- **If-then-else statement:**

```
% stop order
if finish=="n"
    disp("Please continue ordering!")
else
    % order more
    if finish=="y"
        running=1;
    end
end
```

- **While loop and If-then-else statement:**

```
% choose dish
rightDish=0;
while rightDish==0
    % allow input from 1 to 10
    if dish<1 || dish>10 || dish~=round(dish) || isnan(dish)
        disp("Please enter the dish's number again!")
        dish=input("Choose your dishes:","s");
        dish=str2double(dish);
    else
        rightDish=1;
    end
end
```

5. String

```
orderList="";

% format the list as a table
for i=1:size(matrix, 1)

    % format dish name
    blank=23-strlength(matrix(i,1));
    for j=1:blank
        matrix(i,1)=matrix(i,1)+" ";
    end

    % add "x" after quantity
    matrix(i,2)=matrix(i,2)+"x\t";

    % add "AUD" to each dish total price
    matrix(i,3)=sprintf(" AUD %.2f",matrix(i,3));

    % each row contains dish name, quantity and price
    for j= 1:3
        orderList=orderList+matrix(i,j);
    end
    orderList=orderList+"\n";
end
```

6. Function

```
% this function is used for printing receipt as a text file
% take order details
function []= printReceipt(matrix, name, phone)

% all elements are string
arguments
    matrix(:,3) string
    name string
    phone string
end

% receipt time
time=datetime("now");

% order total cash
total=0;
% calculate order
for i=1:size(matrix,1)
    total=total+str2double(matrix(i,3));
end

% function for formatting list
orderList = formattingList(matrix);

% prepare a text file
receiptText = fopen("receipt.txt",'w');
% format text file
fprintf(receiptText, "**************************************************\n" + ...
    "                Morningstars Coffee\n\n\n" + ...
    "                      RECEIPT\n " + ...
    "               %s \n\nCustomer: %s\nContact number: %s\n\n", time, upper(name), phone);
fprintf(receiptText, orderList);
fprintf(receiptText,  "\n\nYour order costs: AUD %.2f\n " + ...
    "                     Thank you!\n" + ...
    "**************************************************", total);
fclose(receiptText);

% record receipt time
fprintf("Saved on %s.txt\n", time);
end
```

II. Value-add (20/20)

The main functionality of the program is to take the customer's order and print out the bill and the receipt. The program is expected to run like below:

```
------Interactive menu------
Enter customer 's name:Do Phong Tran
Enter customer 's phone number:0336855422
Welcome to Morningstars Cafe, Mr/Mrs Do Phong Tran !
Please take a look at our cafe's menu!
                ----------Menu----------
     ___Beverages___                    ___Price___
1.Vietnamese Coffee                      8.00
2.Espresso                               9.50
3.Americano                              9.50
4.Cappuchino                             10.50
5.Latte                                  11.00
6.Hot chocolate                          10.50
       ___Bakery___
7.Tiramisu                               11.50
8.Ham & Cheese Croissant                 9.90
9.French Toast                           13.99
10.Blue berries cake                     12.99
_____________________________________________________________________________
When ordering, enter the dishes'numbers'!
E.g: for Vietnamese traditional coffee, enter '1'
We don't take more than 9 per dish per time!
Choose your dishes:6
 Quantity:1
Do you want to finish your order? (Yes=y; No=n):n
Please continue ordering!
Choose your dishes:8
 Quantity:1
Do you want to finish your order? (Yes=y; No=n):y
Ordering successfully!

|                    ___Here is your order___              |
         Dish                Quantity              Price
       Hot chocolate             1                  10.50
Ham & Cheese Croissant           1                  9.90

           Total                                    20.40
Do you want to cancel your order? (Yes='y', No='n'):n
 You accepted your order!
Do you want a recipt? (Yes='y', No='n'):y
Printing receipt...
Saved on 17-Apr-2023 21:14:52.txt
```

The output file is a text file called " receipt.txt":

```
**************************************************
                    Morningstars Coffee


                          RECEIPT
                  17-Apr-2023 21:14:52

Customer: DO PHONG TRAN
Contact number: 0336855422

Hot chocolate              1x      AUD 10.50
Ham & Cheese Croissant     1x      AUD 9.90


Your order costs: AUD 20.40
                          Thank you!
**************************************************
```

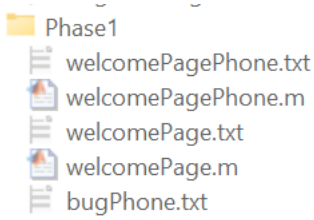
Besides its main functionality, the customer is asked to enter his/her name and phone number for a more personalized experience. The menu and the instructions for ordering are displayed clearly, and the bill is displayed after the customer finished ordering. The customer can cancel an order if they want to and choose if a receipt should be printed. The receipt is printed in a separate text file, and there is a function to format the receipt. I used 4 functions that are outside the course:

- *isnan* for while loop conditions: return a logical array containing 1 (true) where the elements of A are NaN, and 0 (false) where they are not.
- *argument* for declaring input arguments for a function.
- *datetime* for representing points in time.
- *fopen, fclose* for creating and writing on a text file.

III. Incremental development (20/20)

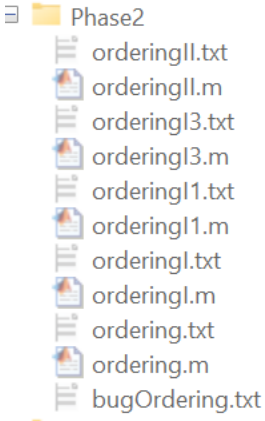
My development process contains 4 phases, each phase contains development files from small lines of code to emerging lines of code. Each development file has a test file with it.

- **Phase 1: Customer basic information and the café menu**



In this phase, the customer is asked to input name and phone number. After that, a menu will be displayed. The script “welcomePage.m” is a fixed code since I changed the previous code due to errors in the input phone number. The fixed code for phone number input is in “welcomePagePhone.m”

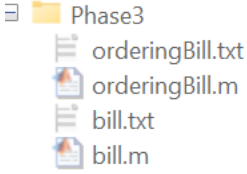
- **Phase 2: Ordering and storing customer order inside a string matrix**



In this phase, the customer will input the order: dish and quantity. The main script is “ordering.m” where the customer will input the order. This script is separated into 2 parts:

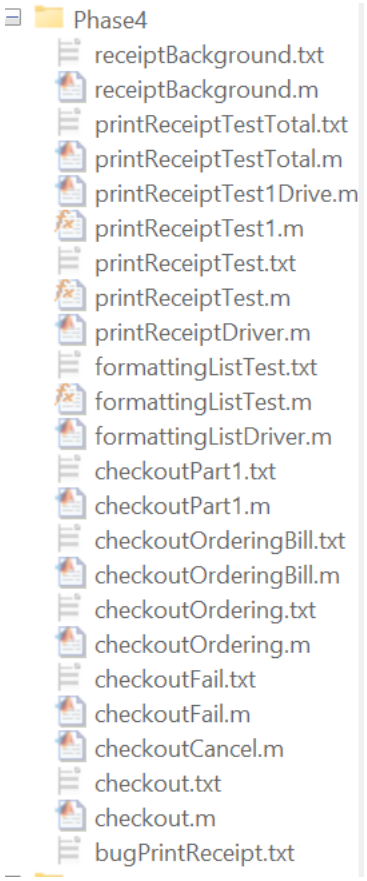
- **I. Ordering:** The customer inputs the order: dish and quantity. A separate script for it is “orderingI.m”. This script is also separated into three parts: input dish (“orderingI1.m”), input quantity, and input for “finish order or not” question (“orderingI3.m”). Since I used the same code from the input dish part for the input quantity part, I only wrote a separate script for the input dish part and a test file for it.
- **II. Store order in a matrix:** In this part, there are no interactions with the customer, only code for working with the customer ’s order to create a string matrix that contains the order ‘s detail (“orderingII.m”).

- **Phase 3: Bill**



In this phase, the program displays the order as a “bill” for rechecking orders (“bill.m”). There is also a script where I combined code for phase 2 and phase 3 together to check if the emerging code works well or not (“orderingBill.m”).

- **Phase 4: Checkout**



I created different scripts for each small line of code to check the functionality of the whole phase. In this phase, I created a main script called “checkout.m” to check if the customer wants to cancel the order or if the customer accepts the order. Then the customer will decide whether to print a receipt or not. All the inputs are Yes/No questions. And I included two separate functions for creating receipts and formatting receipts. The main script is separated into two parts:

- **Part 1: Print receipt or not?: “**checkoutPart1.m” is a fixed script for this part. The previous code for it is in “checkoutFail.m”. I found out that I didn’t include the situation where the customer wants to cancel their order. I created a script called “checkoutCancel.m” where I developed the code for canceling order an before I modified it in “checkoutPart1.m”.
- **Part 2: Print receipt using function:** In this part, I separated it into two files for two functions: “printReceiptTest.m” for creating and writing on a text file for the receipt and “formattingList.m” for formatting the receipt. At first, I only created “printReciptTest.m” for both creating and formatting the receipt on a text file. However, I found some bugs and I decided to create another function for formatting. I stored the former error code in “printReceiptTest1.m”. “printReceiptTestTotal.m” is a script where I stored the code for calculating the total price of the

order. Another script is “receiptBackground.m” where I created some code for the template of the receipt.

I also emerged code between phase 2 and phase 4 (“checkoutOrdering.m”) and between phase 2, phase 3, and phase 4 (“checkoutOrderingBill.m”). I wanted to check if these phases can work well together.

Each phase shows substantial evidence of development with different files for different parts of the phase and implements individual components of the solution. All files are well-commented and embedded in small increments of development.

IV. Testing (20/20)

Test files in four phases are all text-based and show evidence of careful program development. In each phase, test files are created to check the functionality of each individual program component. And there are test files to check if my emerging code works well. For this criterion, each script has its own test file. I also kept all the bugs when developing as proof.

- **Phase 1: Customer basic information and the café menu:**

  “welcomePage.txt”: Testing basic customer inputs: name, phone number, and the visualization of the menu (overall test file).

  “welcomPagePhone.txt”: Testing inputs for phone numbers specifically with valid inputs: a string of no more than 10 digits and invalid inputs: negative numbers, decimal, characters, void.
  I expected the code to keep asking for correct inputs.

  “bugPhone.txt”: Storing error code while developing inputs for phone numbers and explanations of how I fixed it.

- **Phase 2: Ordering and storing customer order inside a string matrix:**

  “ordering.txt”: Testing inputs for dish, quantity, and the matrix containing the order ‘s details. I expected the code to run with no cracks and handle both valid and invalid inputs (overall test file).

  - **I. Ordering:**

    “orderingI.txt”: Testing the overall loop for ordering, testing how the big code handles from inputs for dish, inputs for quantity to inputs for the “finish order or not” questions.

    “orderingI1.txt”: Testing valid inputs for dish and quantity and testing how the code handles wrong inputs like decimal, negative numbers, 0, and void. I expected the code to keep asking for valid inputs.

    “orderingI3.txt”: Testing valid inputs for “finish order or not” question: only “y” and “n” and testing how the code handles invalid inputs like numbers or different characters.
    I expected the code to keep asking for valid inputs.

"bugOrdering.txt": Storing error code while developing inputs for dish and quantity and explanations of how I fixed it.

- **II. Store order in a matrix:**
  "orderingII.txt": Testing my code for calculation and the string matrix which contains the order's details.

- **Phase 3: Bill**

  "bill.txt": Testing the visualization of the "bill", checking the format with simulated orders' details (overall test file).

  "billOrdering.txt": Testing the emerging code between phase 2 and phase 3. I expected the code takes inputs and handles invalid inputs correctly while outputting a "bill" with the correct order details.

- **Phase 4: Checkout**
  "checkout.txt": Testing the emerging code for producing a receipt. Testing inputs for canceling orders, printing receipts, and testing functions for printing and formatting receipts. I expected the code to produce a receipt on a separate text file (overall test file).

  - **Part 1: Print receipt or not?:**

    "checkoutPart1.txt": Testing valid inputs for canceling the order, "print receipt or not" questions: "y" and "n" and checking how my code handles invalid inputs like numbers or other characters. I expected the code to keep asking for valid inputs.

    "checkoutFail.txt": Storing the former code for part 1 (no code for canceling order) and explanations of why I updated the code for cancelling the order.

    "checkoutCancel.txt": Testing the code for canceling the order, testing valid inputs: "y" and "n" and checking how my code handles invalid inputs like numbers or other characters. I expected my code to keep asking for valid inputs.

  - **Part 2: Print receipt using function:**

    In this part, I wrote test files for 2 functions: one for printing receipts as test files and one for formatting the receipt. Since the function for formatting the receipt is called inside the function for printing receipts, I created a test file called "printReceiptTest.txt" and a driver called "printReceiptDriver.m" which contains test cases to check how both functions work together. I also created a separate test file for the function that formats the receipts called "formattingListTest.txt" and a driver that contains test cases called "formattingListDriver.m".

    There are test files for developing the function for printing receipt:

    "printReceiptTestTotal.txt": Testing code for calculation of the total price of the order.

"receiptBackground.txt": Checking the template of the receipt.

"printReceiptTest1.txt" and "printReceiptTest1Drive.m" and "bugPrintReceipt.txt": These three files contain my former function for both creating and editing receipts and explanations of why I had to develop a separate file for editing.

I also created two test files for emerging code between phases:

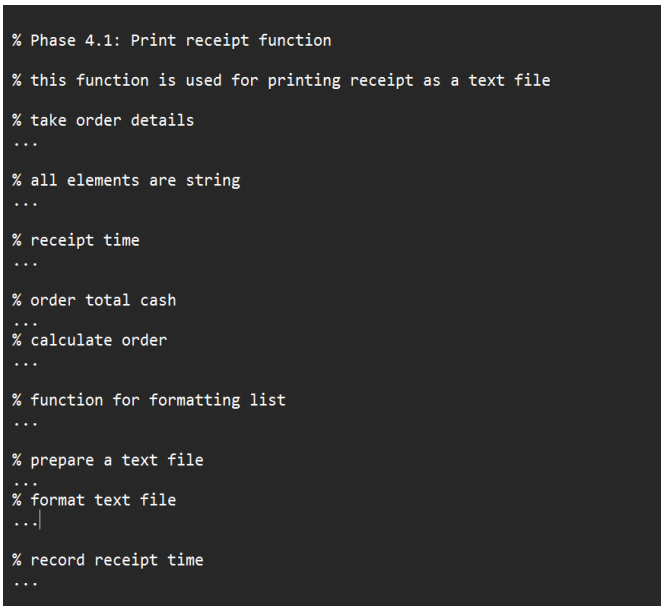
"checkoutOrdering.txt": Testing how the code after emerging phase 2 and phase 4 works together. I expected the output would be receipts with correct order 's details.

"checkoutOrderingBill.txt": Testing how the code after emerging phase 2, phase 3, and phase 4 works together. I expected the order details on the receipt and the bill to be the same.

V. Comments and Style (20/20)

In the final version, all test files and developments file have used comments throughout the whole code. All the variables and files are named correctly based on MATLAB style book 2. The basic structure of the comments looks like this:

```
% Phase 1: Customer basic information and the cafe menu

% clear window
...

% program start
...

% I. Customer basic information

% take customer name in string
...

% take customer phone number in string
...
% ask customer for the right syntax: 10 digits number
...

% print out customer name
...

% II. Menu

% show menu
...
```



Other stages of the program also have comment structures like the image above but simpler. The image above also shows the consistent use of indenting throughout the program.

The variable's names are consistent and named based on MATLAB style book such as name, phone, order, dish, dishNumber, totalCash, and dishTotalPrice. Two functions are named correctly: printReiceipt() and formattingList().