

Project Self-Assessment

Thi Tu Linh Nguyen - a1835497

The self-assessment shows evidence against each rubric criteria and estimates the score for each criterion.

1. Conceptual Coverage (20/20 points)

The program covers all concepts and demonstrates the correct use of each concept.

- Inputs

```
name = input("Customer's name: ", "s");
phone = input("Customer's phone number: ", "s");
```

- Outputs

```
% save receipt as txt file
receiptFilename = sprintf("%s.txt", currentTime);
receiptFile = fopen(receiptFilename,"w");
fprintf(receiptFile, headFormat, currentTime, upper(customerName), customerPhone);
fprintf(receiptFile, orderFormat);
fprintf(receiptFile, tailFormat, total);
fclose(receiptFile);
fprintf("Receipt saved as %s.txt", currentTime);
```

- Loops

- While loop:

```
% Handle order
orders=[];
while order ~= 0
    if order <= menuRows && order >= 1
        numDishes = input("Quantity: ");
        orders = [orders; numDishes, MENU(order,1), str2double(MENU(order,2)) * numDishes];
        order = input("Please enter dish number: ");
    else
        order = input("Please enter a valid key: ");
    end
end
```

- For loop:

```
for i = 1: size(menu,1)
    fprintf("(%d): %s - $%.2f\n", i, menu(i,1), menu(i,2));
end
```

- Vectors

```
% Store menu data in constants
MENUITEMS = ["Beef Pho", "Shrimp Cold Roll", "Vietnamese Roll", "Iced Coffee"];
MENUPRICES = [16.20, 7.00, 10.50, 6.50];
```

- Matrices

```
orders = [orders; numDishes, MENUITEMS(order), MENUPRICES(order) * numDishes];
```

- Conditional execution

```
% Submit the order
if order == 0
    if isempty(orders)
        disp("Ordered unsuccessfully.");
    else
        disp("Your Order (Quantity - Item): ");
        disp(orders(:,1:2));
        createreceipt(orders, name, phone);
    end
end
```

- Functions

```
% format order
function [orderFormat] = format(orders)
    orderFormat = "";
    for i=1:size(orders, 1)
        quantity = orders(i, 1);
        item = orders(i, 2);
        amt = orders(i, 3);

        % format quantity string
        quantity = quantity + " x\t";

        % format item string
        spaceNeeded = 20 - strlen(item);
        for j=1:spaceNeeded
            item = item + " ";
        end

        % format amt string
        amt = sprintf("%.2f", amt);

        orderFormat = orderFormat + quantity + item + amt + "\n";
    end
end
```

2. Value-add (20/20 points)

The main functionality of the program is to take customer's order and print a receipt for that order. The program is expected to run like below (italic texts are the user's inputs):

Customer's name:

Amy

Customer's phone number:

0478682996

=====MENU=====

(1): Beef Pho - \$16.20

(2): Shrimp Cold Roll - \$7.00

(3): Vietnamese Roll - \$10.50

(4): Iced Coffee - \$6.50

=====

INSTRUCTION

To order a dish, please enter the number associated with that dish.

To finish the order, press key 0.

Please enter dish number:

2

Quantity:

1

Please enter dish number:

3

Quantity:

4

Please enter dish number:

0

Your Order (Quantity - Item):

"1" "Shrimp Cold Roll"

"4" "Vietnamese Roll"

Rendering receipt...

Receipt saved as 23-Apr-2022 14:16:12.txt

In the output file “23-Apr-2022 14:16:12.txt”:

Vietnamese
Restaurant

RECEIPT

23-Apr-2022 14:16:12

Customer: AMY

Phone Number: 0478682996

1 x	Shrimp Cold Roll	\$7.00
4 x	Vietnamese Roll	\$42.00













Total: \$49.00

THANK YOU!

Some of the extensions beyond the main functionality are getting the customer's name and phone number, displaying the menu and instructions, displaying where the receipt file is saved, and formatting text in the receipt using the customized *format* function. Also, some of the new MATLAB functions used in the program are *fopen*, *fclose*, and *datetime*.

3. Incremental Development (20/20 points)

The development process consists of six stages, each of which is followed by one MATLAB file and one test file.

-  devStage01.m
-  devStage01Test
-  devStage02.m
-  devStage02Test
-  devStage03.m
-  devStage03Test
-  devStage04.m
-  devStage04
-  devStage05.m
-  devStage05Test
-  devStage06.m
-  devStage06Test

Each stage implements some components of the solution.

- Stage 01: Ask for some basic user inputs and display instructions for users to use the program.
- Stage 02: Ask for the main user inputs and use the while loop to keep asking for inputs until the user enters '0' (end of the program). Validate the previous inputs.
- Stage 03: Use the inputs the user provides, which are stored in a matrix, to handle the logical part of the program. Create a function to display the output of the program.
- Stage 04: Format the output of the program.
- Stage 05: Instead of displaying the output in the terminal, save the output into a text file. Create another function for a better visual of the text file.
- Stage 06: Save some of the previous components as functions for a better reading of the program.

4. Testing (20/20 points)

Test files in six stages are all text-based.

- Test 01: Since the program at this stage only displays text right away, the test will only show what we expect to see in the terminal after the program runs.
- Test 02: The test file shows test cases for phone number inputs and shows expected outputs when the input is not in the expected phone format. It also provides test cases for order inputs and covers cases such as the input is a string, negative number, number out of range, number in the range, and zero (end of the program).
- Test 03: The test file shows what we expect to see in the terminal after the user enters validate inputs.
- Test 04: Similar to test 03, but the output looks more user-friendly.
- Test 05: The test file shows what we expect to see in the output text file after the user enters validate inputs.
- Test 06: Similar to test 05, but the output looks more user-friendly.

5. Comments and Style (20/20 points)

The final version has used comments throughout the program. The basic structure of comments looks like this:

```
% Store menu data in a constant
...

% TODO01: Ask for customer's info: name, phone number, time, orders
...

% display menu and instruction
...

% ask for order
...

% Handle order
orders=[];
while order ~= 0
    ...
end

% Submit the order
if order == 0
    ...
end

% display menu
function [] = displayMenu(menu)
    ...
end

% display instruction
function [] = displayInstruction()
    ...
end

% TODO02: Create a function to display receipt
function [] = createreceipt(orders, customerName, customerPhone)
    ...
    % Record time of completing order
    ...

    % Calculate total of the receipt
    ...

    % format head, body, and tail of the receipt
    ...

    % save receipt as txt file
    ...
end

% format order
function [orderFormat] = format(orders)
    orderFormat = "";
    for i=1:size(orders, 1)
        ...

        % format quantity string
        ...

        % format item string
        ...

        % format amt string
        ...

        orderFormat = orderFormat + quantity + item + amt + "\n";
    end
end
```

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 names are consistent and sensible such as *name*, *phone*, *order*, *displayMenu*, *receiptFile*. The constants are formatted differently such as *MENUITEMS* and *MENUPRICES*.