Lab 1 - Description

(Stacks and Queues)

Lab Overview:

For this lab, we will be learning how to build and use linear data structures like stacks and queues. Stacks and queues are very common data structures in the CS world. In this lab, we will focus on implementing these data structures using the linked list data structure as a base.

Core Tasks:

1. Write the queue class.
2. Write the stack class
3. Test your classes with the provided main function.

Program Requirements:

Task 1: Write the queue class.

For task 1, you will need to write a *Queue* class. The class is detailed as follows:

- __init__(self, capacity) → **None**
    - **Complexity:** O(1)
    - **Valid Input:** An integer in (0, ∞]. Use a default, valid capacity on invalid input.
    - This is the constructor for the class. You will need to store the following here:
        - The *head* pointer
        - The *tail* pointer
        - A variable *maxSize*
        - A variable *currentSize*
        - Any other instance variables you want.
- enqueue(self, ticket) → **boolean**
    - **Complexity:** O(1)
    - **Valid Input:** An object of type: MealTicket. Return False on invalid input.
    - This is the enqueue method. It will add a meal ticket to the queue. It will then return True/False depending on if the enqueue was successful (**Hint**: Add a new ticket at the tail of the queue).
- dequeue(self) → **MealTicket**/ **boolean**
    - **Complexity:** O(1)
    - This is the dequeue method. It will remove the ticket at the front of the queue and return it or False if the queue is empty. (**Hint**: Dequeue a ticket at the head of the Queue).
- front(self) → **MealTicket**/ **boolean**

    - **Complexity:** O(1)
    - This method lets the user peak at the ticket at the front of the queue without deleting it. It will either return a meal ticket or false if the queue is empty.
- isEmpty(self) → **boolean**
    - **Complexity:** O(1)
    - This method will return True/False depending on if the queue is empty or not.
- isFull(self) → **boolean**
    - **Complexity:** O(1)
    - This method will return True/False depending on if the queue is full or not,

Note: See Figure 1 at the bottom of this section for a sample test run of the Queue class. Your program must perform ***exactly*** as shown in the figure. **Note 2**: All methods in the queue class have a hard complexity upper bound of O(1). Your implementation **must** be in O(1). **Note 3:** For this assignment you **must** make and use your own linked list as the base data structure. Using lists, dictionaries or any other data structure will result in a 0 for the assignment.

Task 2: Write the stack class

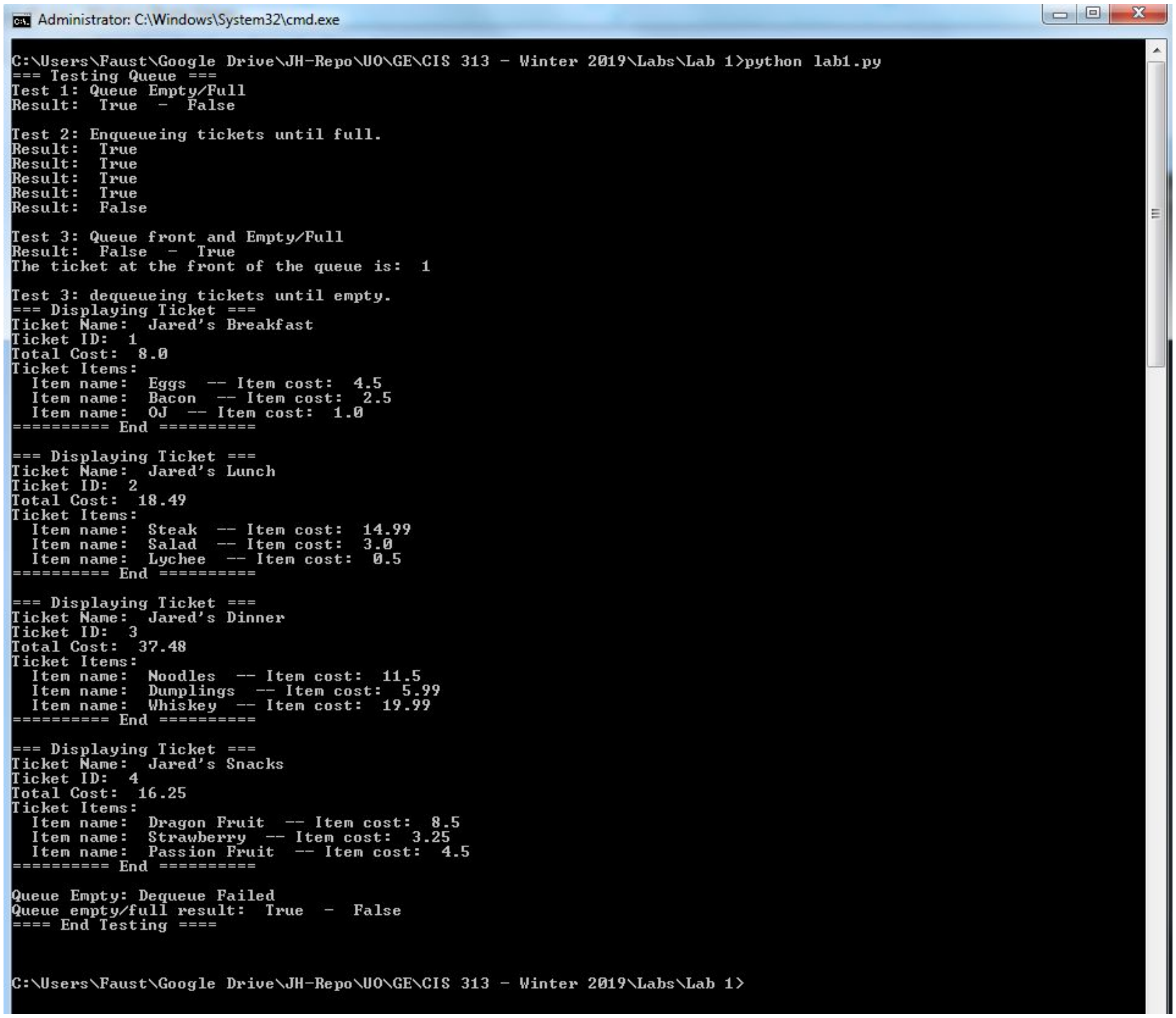
For task 2, you will need to write a *Stack* class. The class is detailed as follows:

- __init__(self, capacity) → **None**
    - **Complexity:** O(1)
    - **Valid Input:** An integer in (0, ∞]. Use a default, valid capacity on invalid input.
    - This is the constructor for the class. You will need to store the following here:
        - The *head* pointer
        - A variable *maxSize*
        - A variable *currentSize*
        - Any other instance variables you want.
- push(self, ticket) → **boolean**
    - **Complexity:** O(1)
    - **Valid Input:** An object of type: MealTicket. Return False on invalid input.
    - This is the push method. It will take a ticket and push it at the top of the stack. It will then return True/False depending on if the push was successful (**Hint**: Add a ticket at the head of the Stack).
- pop(self) → MealTicket/ **boolean**
    - **Complexity:** O(1)
    - This is the pop method. It will remove the ticket at the top of the stack and return it or False if the stack is empty (**Hint**: Pop a ticket from the head of the Stack).
- peek(self) → MealTicket/ **boolean**
    - **Complexity:** O(1)
    - This method lets the user peak at the ticket at the top of the stack without deleting it.
- isEmpty(self) → **boolean**
    - **Complexity:** O(1)

    - This method will return True/False depending on if the stack is empty or not.
- isFull(self) → **boolean**
    - **Complexity:** O(1)
    - This method will return True/False depending on if the stack is full or not,

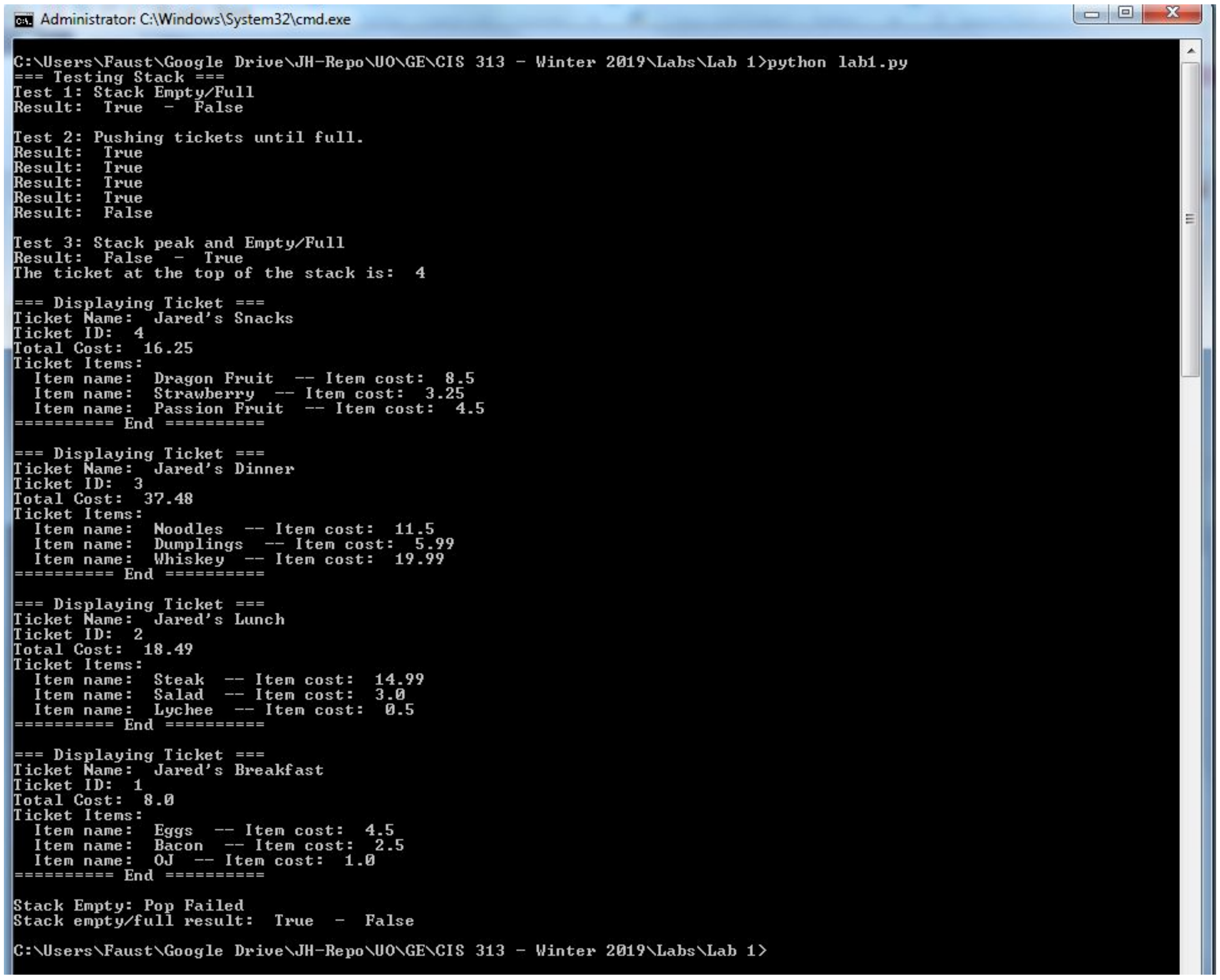
Note: See Figure 2 at the bottom of this section for a sample test run of the *Stack* class. Your program must perform ***exactly*** as shown in the figure. **Note 2**: All methods in the stack class have a hard complexity upper bound of O(1). Your implementation **must** be in O(1). **Note 3:** For this assignment you **must** make and use your own linked list as the base data structure. Using lists, dictionaries or any other data structure will result in a 0 for the assignment.

I/O Specifications:
For this assignment 2 files are provided: lab1-main.py and mealticket.py. You are free to use the main provided or to make your own. **Warning:** I will be testing your code with a more robust main that will check all of the corner cases and uses a wider variety of tickets. Keep this in mind while developing your data structures. The provided main tests some of the corner cases but not all of them. You will need to use the methodology discussed in lab to figure out all corner cases and account for them. Your programs **must** be robust (i.e. do not crash when given bad input or told to peak/front when the stack/queue is empty etc.)


Administrator: C:\Windows\System32\cmd.exe

```
C:\Users\Faust\Google Drive\JH-Repo\UO\GE\CIS 313 - Winter 2019\Labs\Lab 1>python lab1.py
=== Testing Queue ===
Test 1: Queue Empty/Full
Result:  True  -  False

Test 2: Enqueueing tickets until full.
Result:  True
Result:  True
Result:  True
Result:  True
Result:  False

Test 3: Queue front and Empty/Full
Result:  False  -  True
The ticket at the front of the queue is:  1

Test 3: dequeueing tickets until empty.
=== Displaying Ticket ===
Ticket Name:  Jared's Breakfast
Ticket ID:  1
Total Cost:  8.0
Ticket Items:
  Item name:  Eggs  -- Item cost:  4.5
  Item name:  Bacon  -- Item cost:  2.5
  Item name:  OJ  -- Item cost:  1.0
========== End ==========

=== Displaying Ticket ===
Ticket Name:  Jared's Lunch
Ticket ID:  2
Total Cost:  18.49
Ticket Items:
  Item name:  Steak  -- Item cost:  14.99
  Item name:  Salad  -- Item cost:  3.0
  Item name:  Lychee  -- Item cost:  0.5
========== End ==========

=== Displaying Ticket ===
Ticket Name:  Jared's Dinner
Ticket ID:  3
Total Cost:  37.48
Ticket Items:
  Item name:  Noodles  -- Item cost:  11.5
  Item name:  Dumplings  -- Item cost:  5.99
  Item name:  Whiskey  -- Item cost:  19.99
========== End ==========

=== Displaying Ticket ===
Ticket Name:  Jared's Snacks
Ticket ID:  4
Total Cost:  16.25
Ticket Items:
  Item name:  Dragon Fruit  -- Item cost:  8.5
  Item name:  Strawberry  -- Item cost:  3.25
  Item name:  Passion Fruit  -- Item cost:  4.5
========== End ==========

Queue Empty: Dequeue Failed
Queue empty/full result:  True  -  False
==== End Testing ====



C:\Users\Faust\Google Drive\JH-Repo\UO\GE\CIS 313 - Winter 2019\Labs\Lab 1>
```

Figure 1: Queue testing output



```
C:\Users\Faust\Google Drive\JH-Repo\UO\GE\CIS 313 - Winter 2019\Labs\Lab 1>python lab1.py
=== Testing Stack ===
Test 1: Stack Empty/Full
Result:  True  -  False

Test 2: Pushing tickets until full.
Result:  True
Result:  True
Result:  True
Result:  True
Result:  False

Test 3: Stack peak and Empty/Full
Result:  False  -  True
The ticket at the top of the stack is:  4

=== Displaying Ticket ===
Ticket Name:  Jared's Snacks
Ticket ID:  4
Total Cost:  16.25
Ticket Items:
  Item name:  Dragon Fruit  -- Item cost:  8.5
  Item name:  Strawberry  -- Item cost:  3.25
  Item name:  Passion Fruit  -- Item cost:  4.5
========== End ==========

=== Displaying Ticket ===
Ticket Name:  Jared's Dinner
Ticket ID:  3
Total Cost:  37.48
Ticket Items:
  Item name:  Noodles  -- Item cost:  11.5
  Item name:  Dumplings  -- Item cost:  5.99
  Item name:  Whiskey  -- Item cost:  19.99
========== End ==========

=== Displaying Ticket ===
Ticket Name:  Jared's Lunch
Ticket ID:  2
Total Cost:  18.49
Ticket Items:
  Item name:  Steak  -- Item cost:  14.99
  Item name:  Salad  -- Item cost:  3.0
  Item name:  Lychee  -- Item cost:  0.5
========== End ==========

=== Displaying Ticket ===
Ticket Name:  Jared's Breakfast
Ticket ID:  1
Total Cost:  8.0
Ticket Items:
  Item name:  Eggs  -- Item cost:  4.5
  Item name:  Bacon  -- Item cost:  2.5
  Item name:  OJ  -- Item cost:  1.0
========== End ==========

Stack Empty: Pop Failed
Stack empty/full result:  True  -  False

C:\Users\Faust\Google Drive\JH-Repo\UO\GE\CIS 313 - Winter 2019\Labs\Lab 1>
```

Figure 2: Stack testing output

Remarks:

All programs written in this class will be done using the newest version of Python available in the lab (Python 3.7/3.8). This is because Python 3.5+ is platform independent (i.e. you can code on a PC and run it on a Mac). The computers in KLA 26 all have python on them so make sure your code runs on the computers in the lab as it will be tested on an identical system. Feel free to install python on your own computer and bring it to the lab.

If you are new to python, here are some super helpful resources to get you up to speed:

1. Cheat sheets: https://www.pythoncheatsheet.org/ □ Most recommended! Many thumbs up!
2. Python documentation: https://docs.python.org/3/

All programming assignments are to be done **individually**. Your code will be looked at with professional software for cheating. ***Warning:*** This includes using online sources. (e.g. Do not go online and copy code from stack overflow. People have tried this before. You will fail.) Be extra careful with your code. Do not ever show your work to anyone other than the TA (me) or the professor. They will most likely copy your work and your will both fail.

Submission Requirements:

In order to receive any credit for the assignment the student ***must*** do the following:

1. Name your program **"<Duck-ID>_lab1.py"**. (i.e. my duck ID is jhall10 so my submission would be named **jhall10_lab1.py**. **Note:** your duck-ID is the same as your email id and the username to log on to CIS computers ***not*** your 951… number that is your UO PID.
2. Submit your python file onto Canvas.

That's it! Make sure that you test your code to make sure it works.

Grading:

Your work will be graded along three primary metrics: Correctness, Completeness, and Elegance.

Correctness: (60% of total grade)

- You wrote the queue as a FIFO and the stack as a LIFO data structure.
- You wrote the class methods as specified.
- Your class methods meet the complexity requirements.
- You utilize a linked list as the base data structure.
- Your classes are robust, fault tolerant and follow the specified behavior on invalid input.

Completeness (29% of total grade)

- Program contains 2 classes named: *Queue* and *Stack*
- Each class contains methods as defined above.
- The method signatures were implemented as specified.

Elegance: (11% of total grade)

- Your program is well organized.
- You make good use of whitespace, comments, and the file doc string.
- You write your code in a readable manner.
- You use descriptive variable/function/class names.

Late Policy:
The late policy for this class is a bit unique. Make sure to read the info below carefully:
Your homework is always due on a Tuesday at 11:59 pm. On Wednesday, I will take 10% off of the total points available. On Thursday, I will take 20% off. On Friday 30%, no homework will be accepted after Friday. I do not start grading the homework until the following Monday. Keep this in mind when submitting your assignment: It may be better for you to seek help and submit the homework a day late than to submit on time and fail.

If you encounter an unfortunate event or are working with a disability: Please email or speak to me. I am super flexible and am always on your side. I will give extensions as needed and am willing to work with you to make sure you get the most out of this course.