# Computational Problem Solving CSCI-603
# AiRIT Lab 6

6/16/2021

## 1 Introduction

The powers that be at RIT have decided to start a new airline dedicated to getting students to their Spring Break destination free of charge. The airline will of course be named "AiRIT."

From AiRIT's perspective, each student passenger comprises three important pieces of information:

1. **Full Name** - A first name and last name.
2. **Ticket Number** - A potentially variable length string comprising digits and possibly letters. The first character is always a digit and indicates the passenger's *boarding zone*, which will be a number between 1 and 4.
3. **Carry On** - Indicates whether or not the passenger has a carry-on bag that will need to be stowed in the overhead compartment on the plane.

AiRIT operates a single gate at the Greater Rochester International Airport. There is a separate line at the gate for each boarding zone. As passengers arrive at the gate, they take the next position at the back of the line that corresponds with their boarding zone. Local fire codes insist that the gate only allow a set maximum number of passengers to line up at any one time. Once the gate is full, any remaining passengers must wait outside of the gate until *all* of the passengers already in line have boarded a plane.

AiRIT's boarding policy is similar to that of other airlines. When the aircraft is ready for boarding, passengers will board based on their *boarding zone*, with the highest numbered boarding zone boarding *first*. That is to say that passengers assigned to boarding zone 4 will be the first to board the plane, followed by boarding zone 3, and so on, until all passengers have boarded or the plane is full. Passengers in each zone will board the plane in the order that they arrived at the gate and will load the plane from *back to front*, meaning that, as passengers board the plane, they will move to the unoccupied seat that is closest to the back of the plane. It is assumed that the passengers with carry-on luggage will stow their bags in an available space in the overhead compartments as they make their way to their seats. If the gate empties before the aircraft is full, the aircraft will take off with whatever passengers are on board *before* the next set of passengers lines up.

AiRIT's policy for deplaning is somewhat unique. Once the plane arrives at its destination, passengers *without* carry-on luggage deplane from *front to back*. This means that any passengers in boarding zone 1 should depart first, followed by passengers in boarding zone 2, and so on. Once all of the passengers without carry-on luggage have disembarked, the remaining passengers will begin deplaning again from *front to back*.

The AiRIT aircraft includes a maximum passenger capacity, but AiRIT runs the plane continuously, 24 hours a day, 7 days a week. As the plane fills up with passengers, it departs, drops its passengers off at their destination, and immediately returns to the gate ready for the next set of passengers. This continues until all student passengers have been safely sent along to their destinations.

### 1.1 Problem Solving Session

You will work in a group of three or four students as determined by the instructor. Each team will work together to complete the following activities.

1. Write code for a class `Passenger` to represent a student passenger.
   (a) Each passenger has a name (a string), a ticket number (a string), and whether or not the passenger has a carry-on bag. Make sure you use `__slots__` to limit the valid data member names.
   (b) Write a constructor that takes these arguments and initializes the data members.
   (c) Write a method that is suitable when printing out a `Passenger` object. It should return a string that contains the passenger's full name followed by its ticket number and whether or not has a carry-on.
2. Design a class `Gate` to represent the airline gate.
   (a) What data structure(s) will the gate need to maintain in order to allow the passengers to line up for their respective boarding zones?
   (b) There is a fire-code mandated maximum number of passengers allowed in the gate at a time. How will you enforce this?
3. Write the **pseudocode** for the function that, given a passenger and a gate, places the passenger into the appropriate line at the gate. This function should return `True` if there is still room at the gate, or `False` if the gate is full after the passenger gets into line.
4. What data structure(s) will the plane need to keep track of passengers as they load and unload the plane?
5. **(Optional)** Write the **pseudocode** for the function that, given a gate and an AiRIT aircraft, loads passengers onto the plane. For the purposes of this exercise, you can ignore whether or not the passenger has a carry-on.

**When your team has completed the required tasks, validate your solutions with your instructor or an SLI.**

## 2 Implementation

### 2.1 Provided Files

You have been provided with three files that contain passenger data as comma-separated values: `passengers_very_small.txt`, `passengers_small.txt`, and `passengers_large.txt` that contain data for 10, 100, and 1000 passengers respectively.

### 2.2 Main Program

The main program should be named `airit_simulation`. This program will simulate the AirRIT's boarding and disembarking policies. The file of passengers is provided to it on the command line. The format will be identical to the files provided. If it is not present, you should print the usage statement and exit:

```
Usage: python3 airit_simulation.py {filename}
```

If the file name is provided, but does not exist, you should print the following message and exit:

```
File not found: {filename}
```

If the file exists, its contents are guaranteed to all be valid.

Once it runs, the program will prompt for the following:

1. The fire-code mandated maximum number of passengers allowed at an airline gate. This number must be a positive integer.
2. The maximum passenger capacity of an AiRIT aircraft. This number must be a positive integer.

All user input must be error-checked (correct type and value), displaying a descriptive error message when needed. The user is given an unlimited number of chances to enter a valid value.

Once all the required information is provided correctly, execute a `simulation` method that repeats the following until every passenger has arrived at his or her destination:

1. Adds passengers to the boarding zone lines at the gate until the gate reaches maximum capacity.
2. Repeats the following until all of the passengers at the gate have boarded a plane:
   (a) Loads passengers on the plane until the plane is full. The passengers should be printed in the order that they are loaded onto the aircraft.
   (b) Unloads passengers from the plane until the plane is empty. The passengers should be printed in the order that they are unloaded from the aircraft. Remember that passengers without carry-on luggage disembark before those passengers that do have luggage.

### 2.3 Sample Run

The following example uses the provided `passengers_very_small.txt` file that contains data for 10 passengers.

```
> python3 airit_simulation passengers_very_small.txt
Gate capacity: 8
Aircraft capacity: 5
Beginning simulation...
Passengers are lining up at the gate...
 Austin Todd Leonardi, ticket: 20009, carry_on: False
 Zeke Francis Zwerka, ticket: 20005, carry_on: True
 Diana Christina Simmons, ticket: 30010, carry_on: False
 Gregg Constantino Miller, ticket: 30002, carry_on: True
 Austin Wolf, ticket: 40003, carry_on: True
 Tyler Christopher Wolf, ticket: 10004, carry_on: False
 Frank George Golden, ticket: 10008, carry_on: True
 Brett Michael Kirkwood, ticket: 40011, carry_on: False
The gate is full; remaining passengers must wait.
Passengers are boarding the aircraft...
 Austin Wolf, ticket: 40003, carry_on: True
 Brett Michael Kirkwood, ticket: 40011, carry_on: False
 Diana Christina Simmons, ticket: 30010, carry_on: False
 Gregg Constantino Miller, ticket: 30002, carry_on: True
 Austin Todd Leonardi, ticket: 20009, carry_on: False
The aircraft is full.
Ready for taking off ...
The aircraft has landed.
Passengers are disembarking...
 Austin Todd Leonardi, ticket: 20009, carry_on: False
 Diana Christina Simmons, ticket: 30010, carry_on: False
 Brett Michael Kirkwood, ticket: 40011, carry_on: False
 Gregg Constantino Miller, ticket: 30002, carry_on: True
 Austin Wolf, ticket: 40003, carry_on: True
Passengers are boarding the aircraft...
 Zeke Francis Zwerka, ticket: 20005, carry_on: True
 Tyler Christopher Wolf, ticket: 10004, carry_on: False
 Frank George Golden, ticket: 10008, carry_on: True
There are no more passengers at the gate.
Ready for taking off ...
The aircraft has landed.
Passengers are disembarking...
 Tyler Christopher Wolf, ticket: 10004, carry_on: False
 Frank George Golden, ticket: 10008, carry_on: True
 Zeke Francis Zwerka, ticket: 20005, carry_on: True
Passengers are lining up at the gate...
 Lucas Becker, ticket: 40007, carry_on: True
 Catherine Toper, ticket: 30006, carry_on: False
The last passenger is in line!
Passengers are boarding the aircraft...
 Lucas Becker, ticket: 40007, carry_on: True
 Catherine Toper, ticket: 30006, carry_on: False
There are no more passengers at the gate.
```

```
Ready for taking off ...
The aircraft has landed.
Passengers are disembarking...
 Catherine Toper, ticket: 30006, carry_on: False
 Lucas Becker, ticket: 40007, carry_on: True
Simulation complete; all passengers are at their destination!
```

### 2.4 Constraints

You may use the `Queue` and `Stack` data structures provided to you in lecture. You may not use any other Python data structures or advanced Python features not discussed in lecture.

### 2.5 Testing

Write a module named `airit_tests` to test the functions from your `airit_simulation` module. It must include sufficient test functions for each part of the main program. All test functions should be called from a `main` function in the `airit_tests` program.

For example, you may write a function to test your implementation of the function in charge of reading a passenger from the file. This function should do the following:

1. Open the `passengers_very_small.txt` file.
2. Loop, using read passenger to read one student passenger from the file at a time before printing the student passenger, terminating after the last one is read from the file.
3. Close the file.

## 3 Grading

Your grade will be determine as follows:

- 20%: results of the problem solving
- 10% Design: The solution is object oriented and breaks the problem down between various classes and methods.
- 55%: Functionality
  - 5%: Error handling (e.g. file not found, valid maximum passenger capacity)
  - 5%: File handling, reading passengers in from a file
  - 15%: Lining up passengers at the gate
  - 20%: Loading/unloading passengers to/from the aircraft
  - 10%: main simulation loop
- 10%: Your `airit_tests.py` program, which should include sufficient tests to verity that each and every function in your `airit_simulation` program functions as expected.

- 5%: Code Style and Documentation

Good code design is a significant graded component. Your program should not be just a "monolitic" main function. The main program should be very small and pass control into classes/methods. You should have things like a method to read the passenger file in, a method for lining up the passengers at the gate, a method to run the main simulation, etc.

## 4 Submission

Create a ZIP file named `lab6.zip` that contains all your source code. Submit the ZIP file to the MyCourses assignment before the due date (if you submit another format, such as 7-Zip, WinRAR, or Tar, you will not receive credit for this lab).