Go Programming Fundamentals

Assignment #3 Due Date:

Parking lot simulation

Programming Requirements

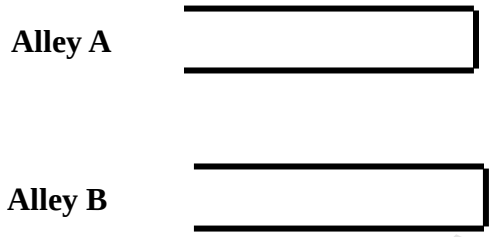
- Submit your assignment into a **single Go language file**.
- You must write **your name** at the top of your assignment source list.
- Write your **compiler version and operating system name** at the top of your assignment source list.
- Assignment that is turned in late will lose **one point per day** starting after the due date.
- Make sure that you do appropriate **error checking** in your program. (User-friendliness)
- Do not turn in **incomplete or crashing program**, you will receive **zero points**.
- Do not **Import third part packages** into your golang source file, your assignment will not be graded.
- Do not use golang **container** package for **Link List**, your assignment will not be graded.
- Make sure to read **grading policy** carefully that will tell you how your assignment is graded.

Assignment #2 Grading Policy

Category	**Points Possible**	**Points Received**
Use of a Link List.	20	
Error-Handling	15	
Treat Link List as Stack.	15	
Program Output according to assignment specifications	10	
Correctness and Efficiency	10	
Style and Code Readability	10	
Complete Documentation	10	
User-friendliness Points	10	
Total	100	

Assignment Description

This program simulates the operations of a parking lot. This particular parking lot consists of two alleys, each of which is wide enough for only one car and is closed at one end. You must use a Link List.



Alley A is the primary parking location. Alley B is used only as a place to move cars out of the way when retrieving a car parked in the middle or end of Alley A.

Program execution starts with both alleys empty. The program repeatedly prompts the user to specify one of four commands:

1. Park a car
2. Retrieve a car
3. Display the contents of Alley A
4. Terminate the program

To park a car, the program generates a new ticket number (1,2,...), issues a ticket stub to the customer, and parks the car at the front of Alley A.

To retrieve a car, the program prompts the user for the ticket stub number and begins to search Alley A. The program moves each car at the front of Alley A to Alley B until it finds the desired car. All of the cars that were temporarily placed in Alley B are then moved back to Alley A.

Input for the program

Each input command consists of a single lowercase or uppercase letter:

1. 'd' or 'D' (To display the alley contents)
2. 'p' or 'P' (To park a car)
3. 'r' or 'R' (To retrieve a car)
4. 'q' or 'Q' (to terminate the program)

Output from the program

Following **menu should be displayed** when you start your program:

D) isplay P) ark R) etrieve Q) uit:

In response to the Park command, the program generates a ticket number and displays this number, denoted below by <t>:

Ticket no. = <t>

In response to Retrieve command, the program displays the following **prompt** for the **ticket stub number**:

Ticket no. :

The output for the Display command is as follows, where each <t> represents the integer ticket number of a car in the alley:

Alley A: <t> <t> <t> ...

Error-Handling

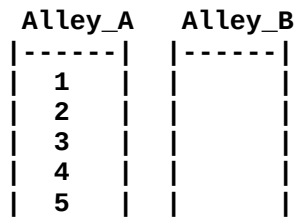
1. The program ignores invalid user commands.
2. If the user tries to retrieve a car with a nonexistent ticket stub number, the program displays the message: "CAR NOT PARKED IN MY LOT."

3. If Alley A is full and the user wants to park a car, the program displays the message: "PARKING LOT FULL".

Example 1

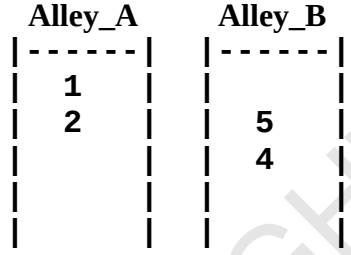
To park a car in to **Alley_A** user will select P, this will indicate that Car is parked in **Alley_A** and ticket number is automatically incremented which will be displayed on the screen.

For this assignment you allow **only 5 cars** to park in Alley A, on the 6th car display the message on the screen "**My Lot is Full**"

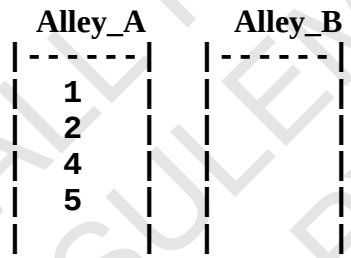


In the following diagram **Retrieve car #3**, user will select R to retrieve the car and your program should ask the user for ticket number. For this action you will remove the node from this link list.

D)isplay P)ark R)etrieve Q)uit: R
Enter the Ticket no. = 3



After the retrieve command is completed from the above action move cars from Alley_B back into Alley_A . Following diagram shows how Alley A will look like after retrieve command.



Summary of Commands Usage

The Retrive command first moves car(s) that are on top of the stack in **Alley_A** and put these cars in a reverse order into an **Alley B**. After the **Retrive** command completes put all of the cars from **Alley_B** back into **Alley_A**. When there are no cars in **Alley_A** and user issues a **Retrival** command display the message "**Lot is Empty, No cars to Retrieve**"

Example 2

Below is a sample execution of this program with all inputs:

D) isplay P) ark R) etrieve Q) uit: P
Ticket no. = 1

In above menu user selected P so you generate ticket number 1 and park the car in Alley A.

D) isplay P) ark R) etrieve Q) uit: P
Ticket no. = 2

D) isplay P) ark R) etrieve Q) uit: P
Ticket no. = 3

D) isplay P) ark R) etrieve Q) uit: d
Alley A: 3 2 1

D) isplay P) ark R) etrieve Q) uit: r
Ticket no. = 2

D) isplay P) ark R) etrieve Q) uit: r
Ticket no. = 5

CAR NOT PARKED IN MY LOT

Hints for Assignment #3

The car will move like a LIFO. For example there are 4 cars in Alley A.

1 ==> First car in Alley A (bottom of the stack)
2
3
4 ==> last car in Alley A (top of the stack)

Now you want to remove (retrieve) car#2 from Alley A, so first you will move cars to Alley B like this:

Remove car#4 from Alley A and put car# 4 into Alley B
Remove car#3 from Alley A and put car# 3 into Alley B
Remove car#2 from Alley A   (This car is retrived and does not go into Alley B)
Remove car#3 from Alley B and put car#3 into Alley A
Remove car#4 from Alley B and put car#4 into Alley A

Your Alley B should be empty

Your Alley A should have following cars:

1 ==> First car in Alley A (bottom of the stack)
3
4 ==> last car in Alley A (top of the stack)

I hope this explains how LIFO.