Cpp file

```
#include<iostream>
#include<string>
#include<fstream>
#include <cstdlib>
#include"Garage.h"
#include"Stack.h"
using namespace std;

int main()
{
	GarageD car;

	ifstream infile;
	ofstream outfile;

	infile.open("ParkingGarage.txt");
	if (infile.fail())
	{
		cout << "file can not open!" << endl;

	}
	string newcar, newcar2;
	while (infile >> newcar >> newcar2)
	{
		infile >> newcar;
		infile >> newcar2;

		if (newcar == "a" || newcar == "A")
		{
			car.arrive(newcar2);
		}
		else
			if (newcar == "d" || newcar == "D")
				car.depart(newcar2);
	}
	infile.close();
	return 0;
}
```

Garage.h

```
#include<iostream>
#include<string>
#include"stack.h"
#ifndef Garage_h
#define Garage_h
using namespace std;

struct car
{
	int moves;
	string plateNumber;
public:
	bool operator==(const car& x)
	{
		return(plateNumber == x.plateNumber);
	}
};

class GarageD
{

private:
	car a;
	car c;
	Stack<car> lane1;
	Stack<car> lane2;
	Stack<car> street;

public:
	GarageD();
	void arrive(string vehicle);
	void depart(string vehicle);

};
GarageD::GarageD()
{

}

void GarageD::arrive(string vehicle)
{
	if (!lane1.IsFull())
	{
		cout << vehicle << " has been parked in line 1.\n";
```

```
                a.plateNumber = vehicle;
                a.moves = 0;
                lane1.push(a);

        }
        else if (!lane2.IsFull())
        {
                cout << "line1 is full, the car is moving to line2" << endl;
                cout << vehicle << " has been parked in line 2.\n";
                a.plateNumber = vehicle;
                a.moves = 0;
                lane2.push(a);
        }
        else
                cout << "both lines are full, your car was not parkerd!" << endl;


}
void GarageD::depart(string vehicle)
{
        a.plateNumber = vehicle;
        c.plateNumber = vehicle;

        cout << endl;
        if (lane1.Search(a))
        {
                while (a.plateNumber != lane1.Top().plateNumber)
                {
                        c = lane1.Top();
                        c.moves++;
                        if (!lane2.IsFull()) {
                                lane2.push(c);
                                lane1.pop();
                        }
                        else {
                                street.push(c);
                                lane1.pop();
                        }
                }

                c = lane1.Top();
                c.moves++;
                cout << "Car is in line1, and it has been depart from line 1." << c.plateNumber <<
"Total number of moves" << c.moves << endl;
```

```
			lane1.pop();
		}
		else if (lane2.Search(a))
		{
			while (a.plateNumber != lane2.Top().plateNumber)
			{
				c = lane2.Top();
				c.moves++;
				if (!lane1.IsFull())
				{
					lane1.push(c);
					lane2.pop();
				}
				else {
					street.push(c);
					lane2.pop();
				}
			}
			c = lane2.Top();
			c.moves++;
			cout << "The car depart from line 2." << c.plateNumber << "Total number of
	moves" << c.moves << endl;
			lane2.pop();

		}

		while (!street.IsEmpty())
		{
			if (!lane1.IsFull())
			{
				c = street.Top();
				c.moves++;
				lane1.push(c);
				street.pop();
			}
			else
			{
				c = street.Top();
				c.moves++;
				lane2.push(c);
				street.pop();
			}
		}
	}
```

```
#endif
```

Stack.h

```
// file Stack.h
// array stack implementation
#ifndef Stackh
#define Stackh
#include <cstdlib>

template<class StackType>
class Stack {
	// LIFO objects

public:
	Stack(int MaxStackSize = 5);
	~Stack() { delete[] stack; }
	bool IsEmpty() const { return top == -1; }
	bool IsFull() const { return top == MaxTop; }
	StackType Top() const;
	void push(StackType x);
	void pop();
	bool Search(StackType x);

private:
	int top;    // current top of stack
	int MaxTop; // max value for top
	StackType* stack;   // element array
};

template<class StackType>
Stack<StackType>::Stack(int MaxStackSize)
{
	//Pre: none'
	//Post: Array of size MaxStackSize to implement stack
	// Stack constructor.
	MaxTop = MaxStackSize - 1;
	stack = new StackType[MaxStackSize];
	top = -1;
}
```

```
template<class StackType>
StackType Stack<StackType>::Top() const
{
        //Pre: stack is not empty
        // Post:  Returns top element.
        if (IsEmpty())

        return stack[top];
}

template<class StackType>
void Stack<StackType>::push(StackType x)
{
        //Pre: Stack is not full
        //Post: Push x to stack.
        //              Stack has one more element
        if (IsFull()) throw ("Push fails: full stack"); // Push fails
        stack[++top] = x;
}

template<class StackType>
void Stack<StackType>::pop()
{
        //Pre: Stack is not Empty
        //Post: Stack has one less element
        if (IsEmpty()) {
                throw ("Pop fails: Stack is empty");
                exit(1);
        }; // Pop fails
        top--;
}

template<class StackType>
bool Stack<StackType>::Search(StackType x)
{
        for (int i = 0; i <= top; i++)
        {
                if (*(stack + i) == x)
                        return true;
        }

        return false;
}
```

```
#endif#pragma once
```

Stack.cpp

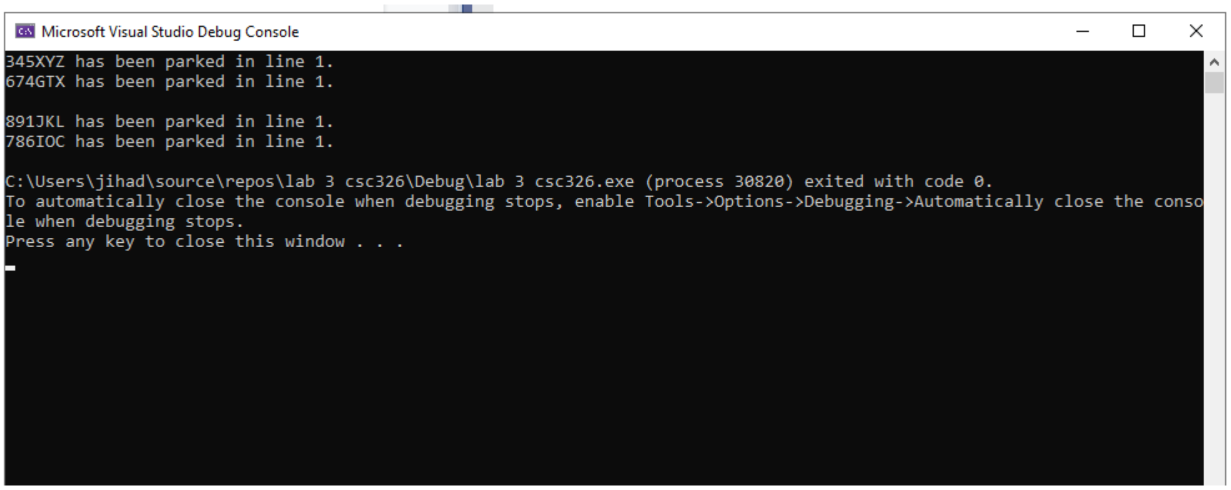
```
/*
#include <cstdlib>
#include <iostream>
#include <string>
#include <vector>
#include <ctime>

using namespace std;

class Car
{ // creates a car object, basically just stores arrival and departure times
public:


private:
    int arrivalTime; // arrival time of car
    int departureTime; // departure time of car


};

class Garage {
public:
    vector <Car> arrivals; //starts with all cars before they arrive
    int number() {  // returns the total number of car arrivals
        return arrivals.size();
    }


private:
    vector<Car> in_garage; // basically just the garage, after arrival before departure

};

int main()
{
    int garageSize = 0;
    Garage garage;
    int seed = time(NULL);  // seed value for pseudo-random number generator
    srand(seed);
    int currentTime = 0; //current time since midnight in seconds
    int stopTime = 86400;
    while (currentTime < stopTime) {
        /*could be shortened since rates are the same at two different times, but it
        might be better to leave it to be able to set rates differently later     */
     //   int nextTime;
     //   if (currentTime < 21600) { // time 0000-0600
     //       int nextRange = 500;
     //       nextTime = rand() % nextRange;
   //     }
   //     else if (currentTime < 25200) { // time 0600-0700

/*int nextRange = 180;
            nextTime = rand() % nextRange;
        }
        else if (currentTime < 28800) { // time 0700-0800
            int next_range = 60;

        }
        else if (currentTime < 39600) { // time 0800-1100
            int nextRange = 12;
            nextTime = rand() % nextRange;
        }
        else if (currentTime < 54000) { // time 1100-1400
            int nextRange = 60;
            nextTime = rand() % nextRange;
        }
        else if (currentTime < 68400) { // time 1400-1800
            int nextRange = 180;
            nextTime = rand() % nextRange;
        }
        else if (currentTime < 86400) { // time 1800-2400
            int nextRange = 500;
            nextTime = rand() % nextRange;
        }
    };
    */
```

ParkingGarage.txt

```
A 123ABC
```

A 345XYZ
D 123DEF
A 674GTX
A 896YUX
D 234FDS
A 567TYD
A 891JKL
D 435GHD
A 786IOC

Output


Microsoft Visual Studio Debug Console

```
345XYZ has been parked in line 1.
674GTX has been parked in line 1.

891JKL has been parked in line 1.
786IOC has been parked in line 1.

C:\Users\jihad\source\repos\lab 3 csc326\Debug\lab 3 csc326.exe (process 30820) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```