**Name: ABID ALI**

**ID: 2019380141**

**Email:** abiduu354@gmail.com

**◎ Problem:**

Enhanced Parking Lot Simulation

**◎ Problem Statement:**

In this assessment the STL stack adapter, the STL vector container is to be used to find STL algorithm in order to solve the problem we are facing in the given program. Therefore, we are supposed to complete the program which stimulates a multiple-aisle parking lot. In the given program we are supposed to complete the implementation for stimulation to the file named main.cpp. The purpose of this task is to keep track of each individual cars about that how many times individual cars are moved while handling the departure of other cars. In addition to that, alphabetized list of cars which visited the parking lot during the stimulation process is to be displayed. For this process we are being provided with the file data.txt with the arrival and departure data.

**◎ Goals:**

To complete this assessment, you will need to finish the implementation of the parking-lot simulation.

To begin, verify the files needed for this assessment.

1.Extract the archive to retrieve the files needed to complete this assessment.

Following is an ordered list of steps that serves as a guide to completing this assessment. Work and test incrementally. Save often

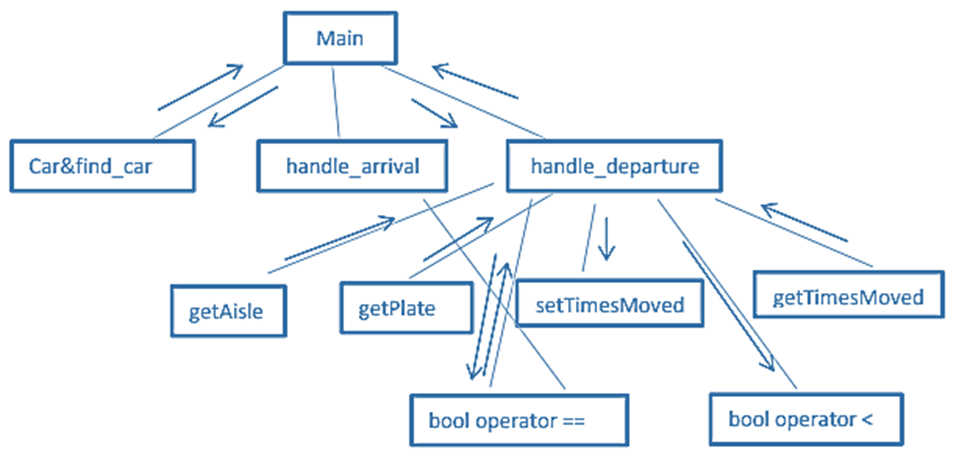
1.First, finish the implementation of function find\_car. This function returns a reference to the Car object stored in the vector cars whose license plate equals the parameter plate. Use the STL find function to pe rform this task. To use this function correctly, you must supply it with three arguments. The first two arguments specify a range to search. The third argument is the value that the function attempts to find. This argument must be of type Car.

2.Next, finish the implementation of function handle\_arrival. This function should iterate through the vector of stacks, looking for the first stack that does not contain three cars. If all five aisles (stacks) are full, output a message indicating such; otherwise place the license plate into the first non-full stack. This is essentially "parking" the car. For this arriving car, also add an entry of type Car to the vector cars. In this Car instance, make sure to record properly the index of the aisle where this car is parked.

3.Then, finish the implementation of function handle\_departure. This function should locate the departing vehicle from the cars vector using function find\_car. Then this function should remove the departing car's license plate from the appropriate aisle. Another stack must be used to move, temporarily, any cars that may be in front of the departing car. Record the number of times a car is moved when accommodating the departure of another car. For the departing car, display the number of times it was moved while it was parked in the lot.

4.Finally, write the code that displays an alphabetized list of all the cars that visited the parking lot. To do this you must first sort the vector cars using the STL sort function. Then, using iterators, traverse the sorted vector and display the license plates of the cars.

**◎ Structure Chart:**



**◎ Implementation:**

**Function:**

**Int main**

Vector function is being used in parking lot stimulation program instead of array because if we had used array, it would not allow us to increase the size of the array whereas vector is known as dynamic array meaning its size can be increased. Therefore, the vector function is being used as seen throughout the program.

*#include <iostream>*

*#include <fstream>*

*#include <cstdlib>*

*#include <string>*

*#include <stack>*

*#include <stdexcept>*

*#include <vector*

*#include <algorithm>*

*#include "Car.h"*

*using namespace std;*

*const unsigned int PARKING\_SPOTS\_PER\_AISLE = 3;*

*const unsigned int NUMBER\_OF\_AISLES = 5;*

*void handle\_arrival(vector<Car>&, vector<stack<string> >&, const string&);*

*void handle\_departure(vector<Car>&, vector<stack<string> >&, const string&);*

*Car& find\_car(vector<Car>&, string);*

In this program try function which first opens the file “data.text” and if there is error the

handle\_arrival

int main() {

{

        ifstream inf("data.txt");

        if (!inf) {

            cerr << "Fail to open ";

            return EXIT\_FAILURE;

        }

        vector<Car> cars;

        vector< stack<string> > parking\_lot(NUMBER\_OF\_AISLES);

        while (!inf.eof()) {

            string action, plate;

            inf >> plate >> action;

            if (action == "arrives") {

                handle\_arrival(cars, parking\_lot, plate);

            }

            else if (action == "departs") {

                handle\_departure(cars, parking\_lot, plate);

            }

            else {

                cerr << "Unknown Entry: " << action << endl;

            }

        }

        inf.close();

        cout << endl << "\nThe following cars visited the lot today:" << endl << endl;

        sort(cars.begin(), cars.end());

        vector<Car>::iterator it;

        for (it = cars.begin(); it != cars.end(); it++)

        {

            cout << it->getPlate() << endl;

        }

        getchar();

        getchar();

        return EXIT\_SUCCESS;

    }

    catch (exception& e) {

        cerr << e.what() << endl;

    }

    catch (...) {

        cerr << "Unknown!" << endl;

    return EXIT\_FAILURE;

    }

}

Parameters:

vector<Car>& cars, vector< stack<string> >& parking\_lot, const string& plate

**Names and types of any parameters:**

“find\_car”-object, “asile” – int, “parking\_lot”, “[departing.getAisle()”, “moving.setTimesMoved”, “moving.getTimesMoved”, temp.push, asile.top(), asile.pop()

Algorithm: Subprogram has two if statements one statement checks if the car lot is full or not the other statement is being proceeded after that where it checks the car arrival plates. Else the output will be the parking lot is full. The main purpose of this section is to handle the car arrivals.

**Names and types of any parameters:**

“asile” – int, “parking\_lot” – int, “parking\_lot.end” –int, departing.getAisle() , asile.top() , find\_car , moving.setTimesMoved, moving.getTimesMoved(),temp.push(asile.top(),asile.pop

void handle\_arrival(vector<Car>& cars, vector< stack<string> >& parking\_lot, const string& plate)

{

    vector<stack<string> >::iterator it;

    int asile = 0;

    bool full = true;

    for (it = parking\_lot.begin(); it != parking\_lot.end(); it++, asile++)

    {

        if (it->size() < PARKING\_SPOTS\_PER\_AISLE)

        {

            full = false;

            break;

        }

    }

    if (!full)

    {

        Car arrival(plate, asile);

        it->push(plate);

        cars.push\_back(arrival);

    }

    else

    {

        cout << "Unfortunately" << plate << ", the lot is full\n";

    }

}

Algorithm: This subprogram uses void function meaning no return type. The program has two while statements. The first while checks the plates when the asile. top() does not returns the reference to the top of components. Similarly, the second while function checks when temp file is not empty.

void handle\_departure(vector<Car>& cars, vector< stack<string> >& parking\_lot, const string& plate)

{

    Car departing = find\_car(cars, plate);

    stack<string> temp;

    stack<string> & asile = parking\_lot[departing.getAisle()];

    while (asile.top() != plate)

    {

        Car& moving = find\_car(cars, asile.top());

        moving.setTimesMoved(moving.getTimesMoved() + 1);

        temp.push(asile.top());

        asile.pop();

    }

    cout << departing.getPlate() << " was moved "<< departing.getTimesMoved() << " times while it was here\n";

    asile.pop();

    while (!temp.empty())

    {

        asile.push(temp.top());

        temp.pop();

    }

}

Car& find\_car(vector<Car>& cars, string plate)

{

    vector<Car>::iterator car;

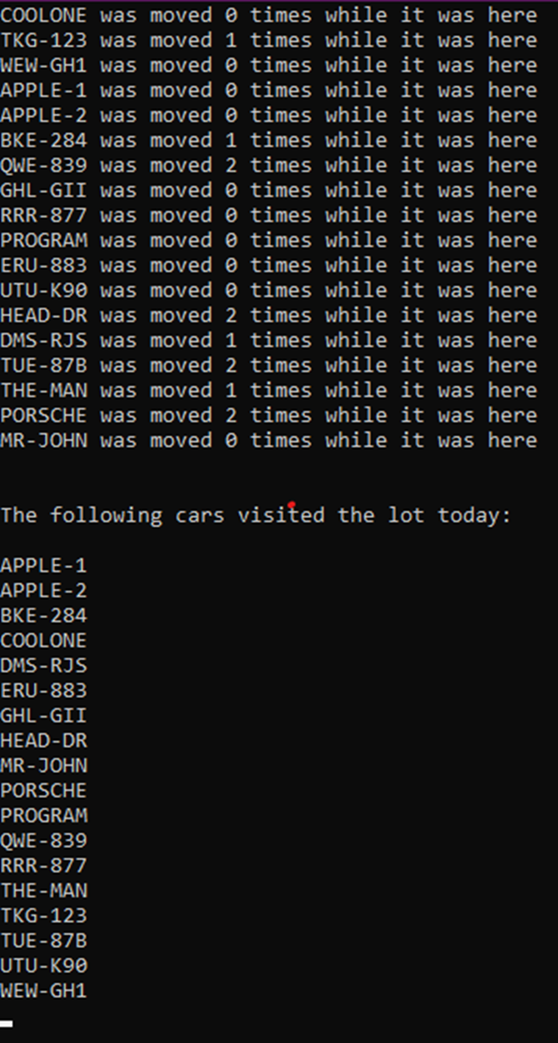
    Car key(plate);

    car = find(cars.begin(), cars.end(), key);

    return \*car;

}

**◎ Test Description and Results:**



The result shows how many times different car was being moved while they were in the parking lot. In addition to that all the license plate of the cars which visited in the parking lot were also displayed.

**◎ Epilogue:**

I had many bugs which had influence on program’s working and many troubles with compilation programs, because of syntax. After checking on the interment I figured out that my syntax was not correct. However, I managed to do understand the algorithm of a parking lot which is very interesting.

**◎ Attachments:**

main.cpp

**◎ Acknowledgement:**

I discuss with students. I had to search a lot of stuffs in the internet to understand very basics of it. It was a very hard task for me. I have spent many days on doing this

**◎ Remarks and Grade (by the instructor)**

Instructor Signature:

Grading Date: