Old Dominion University

Department of Computer Science

CS-250 Final Project

Fall 2016

Introduction:

American football is the most popular sport in USA and it is played by two teams of eleven players on a field. Fans like to go to stadiums to cheer for their favorite team. The Old Dominion Monarchs is the football team of Old Dominion University. Recently, the Old Dominion football team joined Conference USA (CUSA). As a result, the number of fans who want to watch the Old Dominion Monarchs games increased dramatically in the last few years. Organizing parking during the game time is sometimes very hard and time consuming. As a student in the Computer Science Department at ODU, you may propose a solution to help the university automate the parking control during the game day.

Description:

Help ODU organize the car parking during a football game. You will need to develop a machine to automate the process of choosing the parking lot for the visitors (football fans). There are 9 available parking lots (Lot-A, Lot-B, …… Lot-I) distributed in a 3x3 grid as in Figure-1. The maximum capacity of each lot varies as in the figure. The visitor(s) can access the parking lots only via one of three gates (Gate-A, Gate-B and Gate-C). Gate-A is the closest to the main entrance of the football stadium, and Gate-C is the farthest. Visitors are only allowed to enter the stadium via the main entrance. The cost of parking in each lot varies based on the distance between the lot and the football stadium as shown in Figure 1. The closer the parking lot to the stadium's main entrance, the more expensive the parking cost is. The starting cost of parking in lot-A is $25.00, and the cost of parking in each lot decreases by the amount of a $2.50 when you move one step away on the parking grid. Moves on the grid are only allowed in the horizontal or vertical directions. No diagonal moves are allowed. The price of parking also varies based on the total available parking spots. Once the total number of empty parking spots is less than **50**, the price of parking in all parking lots will go up additional **$5.75**. Once the total number of empty parking spots is less than **25**, the price of parking in all parking lots will go up **another additional** **$7.25**. For example, if the cost of Lot-A initially was **$25.00**, once the total number of empty parking spots is less than **50**, the cost of Lot-A becomes **$30.75**. And once the total number of empty parking spots is less than **25**, the cost of Lot-A becomes **$38**.

You will need to write a **C++** program to simulate the machine at each of the three gates. The visitor will choose any of the three gates to get into to the parking lots; your program should allow the visitor to choose the gate. The machine at each gate should display a list of all parking lots as in Table-1. Please note that Table-1 is a sample for the machine display at Gate-A at any random moment. The program should also display a message at the bottom of the machine display asking the visitor about his/her preferred criteria of choosing the parking lot. The following are the criteria for choosing the parking lot,

- If the visitor prefers to choose the parking lot based on the distance from the selected gate, then the program should automatically direct the visitor to the closest parking lot to that gate, which still has at least one available parking spot. If parking spots are available at two or more parking lots which are at equal distances from the gate, then the program should automatically choose one of the lots based on the chronological alphabetical order. The program should next display a message showing the name of the chosen parking lot and the required amount of money.
- If the visitor prefers to choose the parking lot based on the cost in $DD.CC, then the program should automatically direct the visitor to the parking lot which has the cheapest cost. If parking spots are available at two or more parking lots where the parking cost are the same, then the program should automatically choose one of the lot based on the chronological alphabetical order. Next, the program should display a message showing the name of the chosen parking lot and the required amount of money.

Next, the program should ask the visitor to enter money. The machine can only accept the 1, 5, 10 and 20 dollar bills and coins in the form of quarters only. No credit card payment is allowed; the machine accepts cash only. If the visitor enters the exact required amount of money, then the program should display the message "***thank you – Gate is now open***". If the visitor enters money exceeding the required amount, then the program should return change and display a message "***thank you – Gate is now open, your change is: $DD:CC***". You will need to replace the DD with the dollar amount and the CC with the cent amount. If the visitor enters money less than the required amount, then the program should keep asking the visitor to enter more money until the exact required amount is entered or exceeded. The program should next display either one of the two message "***thank you – Gate is now open***" or "***thank you – Gate is now open, your change is: $DD:CC***" based on the amount of money that was entered by the visitor.

Finally, the program should update the records of money in the cash register and subtract one from the available parking spots from the chosen lot.

The program must start with some money in the cash register of each machine. At the beginning, each machine has 500 quarters, 400 one-dollar-bill, 300 five-dollars-bill, 200- ten-dollars-bill and 50 twenty-dollars-bill. The machine usually returns change in forms of biggest possible bill(s).

The program must start while the parking lots are partially filled. The university allows some emergency-employee to park during the game time at any of the available parking spots. Each of the parking lots can be partially filled up to a 10% of its maximum capacity. You will need to use a random number generator to partially fill the parking lots at the beginning of your program. Please note the number of emergency-employee cars in any parking lot can be in the range from 0% to 10% of the maximum capacity of each lot.

Once all lots are filled, the program should display the message "**All parking Lots are full – Sorry!!**". Finally, the program should have an option to save the final information into a file for later analysis by the university. Information to be saved into the file should include,

1. Total number of visitor cars at the end of the day.
2. Total number of emergency-employee cars
3. Total money in the cash register at that moment
4. Total money visitors paid till that moment.

Please note that, the program must not, at any moment, direct the visitor to a parking lot that has zero available spots.

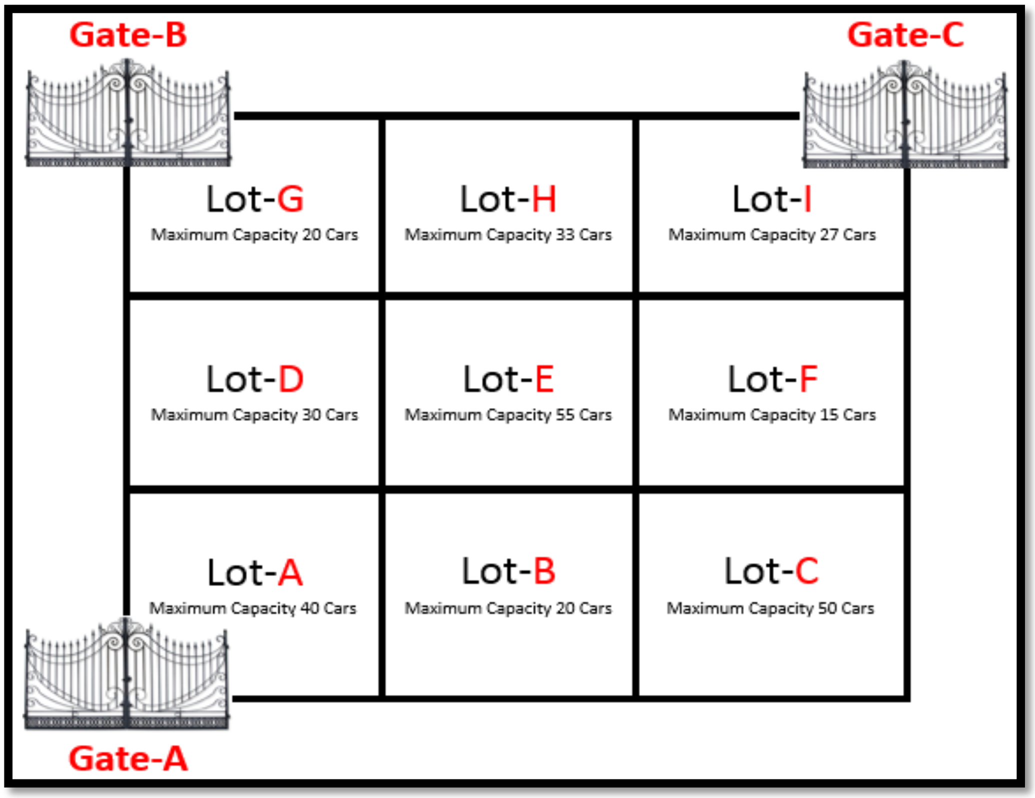

Figure-1

Gate A:

Parking Lot	Maximum Capacity	Available spots	Cost $DD.CC	Distance in meters from current Gate
A	40	27	25.00	10

B	20	7	22.50	20
C	50	38	20.00	30
D	30	11	22.50	20
E	55	41	20.00	30
F	15	7	17.50	40
G	20	13	20.00	30
H	33	21	17.50	40
I	27	11	15.00	50
Total	**290**	**176**		

Table-1

The two machines at Gate-B and Gate-C should also display similar tables with minor changes as follow. For example, if two visitors stopped at the exact moment at Gate-B or Gate-C which is also the exact moment a third visitor stopped at Gate-A. The major difference will be in the order of the parking lots as in Table-2 and Table-3. The machine will always start displaying the parking lots closer to the chosen gate.

Gate B:

Parking Lot	Maximum Capacity	Available spots	Cost $DD.CC	Distance in meters from current Gate
G	20	13	20.00	10
H	33	21	17.50	20
I	27	11	15.00	30
D	30	11	22.50	20
E	55	41	20.00	30
F	15	7	17.50	40
A	40	27	25.00	30
B	20	7	22.50	40
C	50	38	20.00	50
Total	**290**	**176**		

Table-2

Gate C:

Parking Lot	Maximum Capacity	Available spots	Cost $DD.CC	Distance in meters from current Gate
I	27	11	15.00	10
H	33	21	17.50	20
G	20	13	20.00	30
F	15	7	17.50	20
E	55	41	20.00	30
D	30	11	22.50	40
C	50	38	20.00	30
B	20	7	22.50	40
A	40	27	25.00	50
Total	**290**	**176**		

Table-3

Design & Structure:

I do not want to limit your creativity by providing the exact **ADT**s for you; however, I also need to make sure that students are following the appropriate techniques to design, refine and test their code. As a result, please follow the guidelines below when designing your code:

- Design your code into modules. The number of modules should be appropriate for the provided application.
- Each module should be assigned for a single ADT, and each module should be implemented into two files (.h “header” and .cpp “implementation”).
- When needed, use data sequences to control the number of modules.
- You will also need to provide one more module for the main application and name it parkingControlMain.cpp.
- Keep the main application module as simple as you can.
  - No function implementation should be provided inside the body of the main function.
  - The main() function should only call other functions and/or create objects of class(es) and declaring necessary variables.
- If you have to provide function implementation inside the main file (parkingControlMain.cpp) (not inside the main function), then you will need to provide a very good reason for doing so. For example, these functions should be highly related to the main application. Also, the number of these functions should not exceed **two functions** at most. The implementation of these functions should also be short and simple.
- **Never** #include .cpp file(s). If you do, this will result in a big marking penalty even if your program compiles and provides the expected output.
- Must use **Encapsulation** to hide your data.
- Avoid using stand alone functions. Your functions should always be member functions of a class or struct
- Use inline when needed
- Use function templates when needed
- Dynamically create your arrays on the Heap, and watch for memory leak and dangling pointers.

Deliverables:

You must submit all the following

- **Implementation**:
  - Submit all .h and .cpp files of your project.
  - Bundle all these files together into a single zipped file and name it **FinalProject_cslogin.zip**
  - The due date for submitting this file is Saturday, December 3rd at 11:00pm.
  - Also, submit a Readme.txt which specifies the reason for providing function implementation (if any) in main function. Also, keep the main function as simple as possible.

Input:

The program starts with the information read from a file which contains parking machine account details. The name of the input file is "**parking_account.txt**". The following is the sample input format,

```
5
One 400
Five 300
Ten 200
Twenty 50
Quarter 500
```

The first line of the input specifies the number of types of bills available to read followed by the name of the type name and the number of bills for that type.

Output:

The program should write:

1. Total number of visitor cars at the end of the day.
2. Total number of emergency-employee cars
3. Total money in the cash register at that moment
4. Total money visitors paid till that moment.

This information should be written into a file named "**parking_information.txt**". The following is a sample output format,

```
Total Visited Cars: ##
Total Emergency Cars: ##
Total Cash at the End of Day: $DD.CC
Total Amount Earned by the Parking Machine: $DD.CC
```

Grading:

Your final project will be marked based on the following criteria:

• Compilation	15%
• Expected Output	10%
• Main function implementation	5%
• Appropriate ADT's (Classes)	25%
• Use of structures (helpers)	10%
• Use of Encapsulation	15%
• Using Member Functions	10%
• Dynamically Creating your arrays	5%
• Comments and Indentation	5%

If your program doesn't compile, you will receive zero for the project and we will not consider whether you have met the other requirements or not.

Penalties:

The following are the penalties for the project,

• Including .cpp files in another .cpp file	-30%
• Using more than 2 stand-alone functions	-10%

These penalties will be deducted from your total points obtained, *i.e.* if a student obtains 95 out of 100 and that student included .cpp files in another .cpp file, then his/her grade will be 65 out of 100.