

OO Programming

**Advanced Software Development – University of Wales Trinity
Saint David**

Siobhan Stephens
1800709

Table of Contents

1. Introduction	2
1.1. Assignment Brief	2
1.2. System Requirements	2
2. System Design	3
2.1. Pseudocode	3
2.2. Consideration for System	4
2.3. Final Class Design	5
2.4. Class Descriptions	6
2.5. Final Solution	11
3. Testing	12
3.1. Unit Tests	12
3.2. Functional User Testing	16
4. User Guide	20
4.1. System Output Key	20
4.2. System Input	20
4.3. How To	20
5. Conclusion	23
6. References	24
7. Appendix	25
7.1. Appendix A – Testing Screenshots	25
7.2. Appendix B – Final Solution Source Code	45

1. Introduction

1.1. Assignment Brief

The assignment brief is as follows:

A local company has decided to improve security by placing barriers at its staff car park. It has been decided to issue staff identity cards which all employees are expected to carry with them whilst on site. These cards will also be used to gain entry into the staff car park.

A barrier and a card reader are placed at the entrance to the car park. The driver of an approaching car inserts his or her numbered card in the card reader which then checks that the card is known to the system. If recognised, the reader sends a signal to raise the barrier and the car can then enter the car park.

At the exit is also a barrier which is raised when a car leaves. When there are no places in the car park free, a sign at the entrance displays "FULL" and is only switched off when a car leaves.

You have been tasked with developing an OOP C++ car park application based upon the car park narrative above.

Requirements of the assignment:

- Apply OOP concepts including aggregation, inheritance, and polymorphism.
- Use the concept of inheritance to develop two specialized IDReaders. One is an ID barcode reader, the other is a numberplate reader.
- Develop a suit of unit tests and a suitable test harness.
- Include a brief user guide explaining how to run the solution.

The creator of the C++ language, Stroustrup, states that "Object-oriented programming is a technique for programming - a paradigm for writing "good" programs for a set of problems", and has three main key points: Encapsulation; Inheritance and Polymorphism [1].

1.2. System Requirements

After thoroughly reading the given scenario, it was concluded that the car park must include a minimum: an entry and exit barrier, a full sign, sensors for the barriers, an ID reader for the ID card and a reader to read car number plates – along with a dynamic data structure to store the number plates, specifically a singly linked list.

In order to consider the individual elements of the solution, including the classes, methods and their attributes, the solution must be described in more depth, and so pseudocode is to be used to outline the processes of the car park.

2. System Design

2.1.Pseudocode

Given the scenario outlined in section 1, the following initial pseudocode has been produced to describe the process that must occur in the car park solution.

1. Poll entry barrier sensor
2. Poll exit barrier sensor
3. If car present at entry barrier then
4. Poll id barcode reader
5. Poll number plate reader
6. If #free spaces ≥ 1 (there is room in the car park) then
7. If id barcode number is valid then
8. Raise entry barrier to allow car through
9. Wait for car to go through the barrier
10. While car is not clear of the entry barrier
11. Poll the entry barrier sensor until it is clear
12. Lower the entry barrier (car has now entered the car park)
13. Add the number plate to the list
14. Decrement the number of free spaces
15. Else print 'invalid id..'
16. End if
17. End if
18. If car present at exit barrier then
19. Poll number plate reader
20. If the number plate exists in the list
21. Raise exit barrier to allow car through
22. Wait for car to go through the barrier
23. While car is not clear of the exit barrier
24. Poll the exit barrier sensor until it is clear
25. Lower the exit barrier (car has now left the car park)
26. Remove the number plate from the list
27. Increment the number of free spaces
28. Else print 'please get help'
29. End if
30. If free spaces < 1
31. Switch on the full sign
32. Else
33. Switch off the full sign

From this pseudocode, the following objects are required as attributes of the Car Park:

- a) Entry Sensor
- b) Exit Sensor
- c) Entry Barrier
- d) Exit Barrier
- e) Entry Barrier Sensor
- f) Exit Barrier Sensor
- g) ID Barcode Reader
- h) Numberplate Reader
- i) Full Sign
- j) Validate ID
- k) Numberplate List (dynamic linked list is to be used)

It was then concluded that the following classes would be required, with the following methods as a minimum:

1. CarPark
 - a. setCapacity
 - b. setFreeSpaces
 - c. getFreeSpaces
 - d. incrementSpaces
 - e. decrementSpaces
 - f. log
 - g. update
2. Sensor
 - a. readSensorData
 - b. update
3. Barrier
 - a. raise
 - b. lower
 - c. update
4. IDReader (BarcodeReader & Numberplate Reader)
 - a. readData
 - b. update
5. FullSign
 - a. switchOn
 - b. switchOff
 - c. update
6. ValidateID
 - a. validateIDNo
7. LinkedList
 - a. addItem
 - b. searchItem
 - c. removeItem
 - d. printList

The above is not a final list of classes, but an initial list outlining the basic required classes and methods.

2.2.Consideration for System

Given the implementation of the system without physical sensors, barriers and ID readers, consideration must be made as to how these sensor/reader values are to be replicated.

Therefore, it has been decided that any sensor or reader data values are to be represented by user input. To facilitate this requirement, a user input helper class and suitable methods will be implemented, which can be reused in the various required classes.

Additionally, if user input is to be collected, it should be validated before use, and therefore, a validation class will be implemented to validate against the required input types.

2.3.Final Class Design

After further consideration of the system objectives, required processes and objects, the following comprehensive class diagram has been produced.

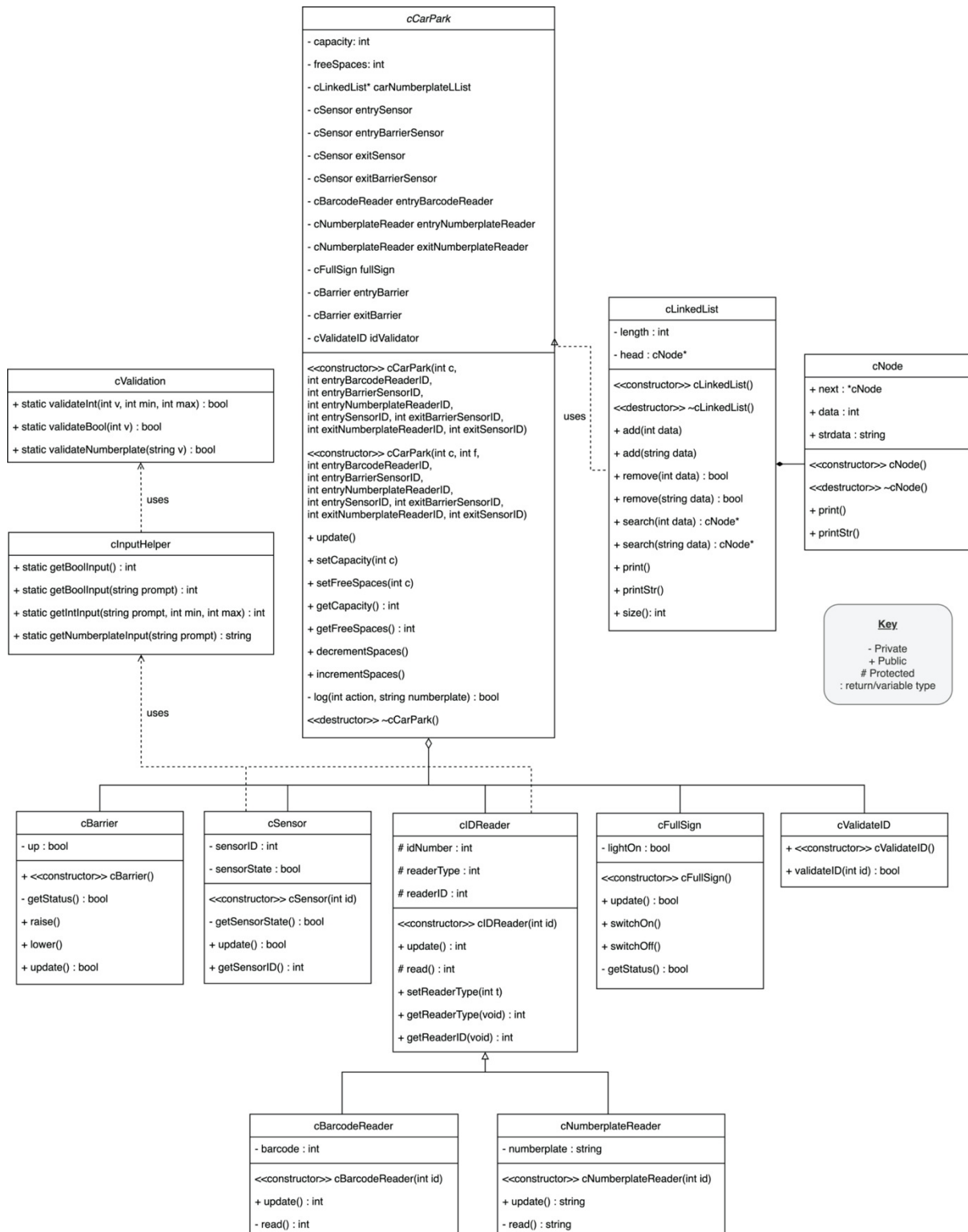


Figure 1. Class Diagram for Car Park system

2.4. Class Descriptions

2.4.1. *cCarPark*

The *cCarPark* class is the main class for the car park, with the processing logic outlined in the pseudocode in section 2.1 occurring within the *update()* method.

cCarPark will have the following attributes:

- a. **capacity**: an integer to store the total capacity (number of spaces) of the car park
- b. **freeSpaces**: an integer to store the current free spaces within the car park
- c. **carNumberplateLList**: a linked list (object of *cLinkedList* class) to store the number plates of the cars currently in the car park
- d. **entrySensor**: the sensor that is on the outside of the car park, that will detect a car waiting to enter the car park, it is an object of type *cSensor*.
- e. **entryBarrierSensor**: the sensor that is under the entry barrier, that will detect a car that is currently in the way of the barrier, so that the barrier doesn't lower itself onto a car. It is an object of type *cSensor*.
- f. **exitSensor**: the sensor that is on the inside of the car park, that will detect a car waiting to exit the car park, it is an object of type *cSensor*.
- g. **exitBarrierSensor**: the sensor that is under the exit barrier, that will detect a car that is currently in the way of the barrier, so that the barrier doesn't lower itself onto a car. It is an object of type *cSensor*.
- h. **entryBarcodeReader**: will be an object derived from a specialised type of *cIDReader*, *cBarcodeReader*, which uses inheritance. Will be used to gather the barcode number of the ID supplied upon entry to the car park, and used to validate the entry.
- i. **entryNumberplateReader**: : will be an object derived from a specialised type of *cIDReader*, *cNumberplateReader*, which uses inheritance. Will be used to gather the numberplate of the car waiting to enter the car park, and will be stored within the *carNumberplateLList* upon entering.
- j. **exitNumberplateReader**: : will be an object derived from a specialised type of *cIDReader*, *cNumberplateReader*, which uses inheritance. Will be used to gather the numberplate of the car waiting to leave the car park, and will be removed from the *carNumberplateLList* once it leaves.
- k. **fullSign**: will be an object of type *cFullSign*, representing the car park's full sign.
- l. **entryBarrier**: will be an object of type *cBarrier*, representing the park's entry barrier.
- m. **exitBarrier**: will be an object of type *cBarrier*, representing the park's exit barrier.
- n. **IdValidator**: will create an object of class *cValidateID*. In a later implementation a constructor would be used to provide specialist instantiation instructions and values eg database credentials when creating an object, however the given implementation has no attributes.

cCarPark will have the following methods:

- a. **cCarPark(int c, int entryBarcodeReaderID, int entryBarrierSensorID, int entryNumberplateReaderID, int entrySensorID, int exitBarrierSensorID, int exitNumberplateReaderID, int exitSensorID)**: constructor that takes an integer *c*, along with multiple attribute IDs as a parameter, used to set the total capacity of the car park and construct the relevant objects. The number of free spaces will be set to the capacity.
- b. **cCarPark(int c, int f, int entryBarcodeReaderID, int entryBarrierSensorID, int entryNumberplateReaderID, int entrySensorID, int exitBarrierSensorID, int exitNumberplateReaderID, int exitSensorID)**: overloading constructor that takes integer *c* for capacity, along with integer *f* (in addition to attribute ID parameters) to

set the number of free spaces in the car park, if some spaces are taken at the time of instantiation. Demonstrates static polymorphism.

- c. **update()**: the main processing logic of the car park system will reside here, following the pseudocode in section 2.1. It is here the links will be made to the other classes, with objects (sometimes multiple) being created for cBarrier, cSensor, cBarcodeReader, cNumberplateReader, and cFullSign. These will include the entryBarrier, exitBarrier, entryBarrierSensor, exitBarrierSensor and so on.
- d. **setCapacity(int c)**: setter method to set the capacity attribute using integer parameter c.
- e. **setFreeSpaces(int f)**: setter method to set the freeSpaces attribute using integer parameter f.
- f. **decrementSpaces()**: setter method to decrement the number of freeSpaces by 1. Contains logic to ensure number of freeSpaces does not go below 0.
- g. **incrementSpaces()**: setter method to increment the number of freeSpaces by 1. Contains logic to ensure number of freeSpaces does not go above capacity.
- h. **log(int action, string numberplate)**: function to write to an event log. This will be a cout statement for the current solution however this could be used in the future to write to a file or database. Returns a boolean true if successful 'write' or false if unsuccessful.
- i. **~cCarPark()**: destructor to delete the linked list.

The final destructor method is required due to the creation of the linked list, in order to deallocate it from memory, as confirmed by IBM, who state that "Destructors are usually used to deallocate memory and do other cleanup for a class object and its class members when the object is destroyed. A destructor is called for a class object when that object passes out of scope or is explicitly deleted" [2].

The cCarPark class will include the headers for cBarrier, cSensor, cIDReader, cFullSign, cLinkedList and cValidateID, which will allow the constructor method to create objects as required. The use of these functions within cCarPark is aggregation.

2.4.2. cBarrier

cBarrier class represents the physical barriers within the car park, of which there will be two, an entry barrier and an exit barrier.

cBarrier will have the following single attribute:

- a. **up**: an integer representing the state of the barrier – up (or down)

cCarPark will have the following methods:

- a. **cBarrier()**: constructor that defaults up to false.
- b. **update()**: function that calls getStatus() to return the state of the barrier
- c. **raise()**: method that sets up = true to raise the barrier.
- d. **lower()**: method that sets up = false to lower the barrier.
- e. **getStatus()**: private method that gets the state of the barrier, returning up as a boolean

2.4.3. cSensor

cSensor class represents the physical sensors within the car park, of which there will be four, namely entrySensor, exitSensor, entryBarrierSensor and exitBarrierSensor. The purpose of entrySensor and exitSensor are to detect cars waiting to enter or leave the car park, with the purpose of entryBarrierSensor and exitBarrierSensor to detect whether cars are underneath the barriers – so that barriers do not lower onto cars, causing damage.

cSensor class will have the following attributes:

- a. **sensorID**: an integer representing the uniqueID of the sensor (assume provided by physical device).
- b. **sensorState**: a boolean representing the current state of the sensor.

cSensor class will have the following methods:

- a. **getSensorState()**: a private function to gather the current state of the sensor, returning a boolean sensorState. As stated in section 2.2, a consideration has been made for this system to include user input to replicate sensor data, therefore, this function will call the cInputHelper class getBoolInt(string prompt) with a suitable prompt string to obtain input from the user.
- b. **getSensorID()**: a function to return the ID of the sensor (set with constructor).
- c. **update()**: a public function to return the current state of the sensor. It calls getSensorState() to gather the input and returns the result.

2.4.4. *cIDReader, cBarcodeReader and cNumberplateReader*

The cIDReader class will use inheritance, and will have two derived classes, cBarcodeReader and cNumberplateReader. As described by Prof Mehtra et al, "in C++ inheritance is a form of software reusability in which a new class acquires all the properties (the data members) and behaviors (the member function) of existing class in hierarchy. The existing class is known as base class and newly created class is called derived class" [3]. The purpose of the class is to represent the physical ID readers within the car park, of which there will be two: a number plate reader and a barcode reader (for ID card).

cIDReader class will have the following attributes:

- a. **idNumber**: an integer representing the generic idNumber.
- b. **readerType**: an integer representing the type of reader.
- c. **readerID**: an integer representing the ID of the reader, such as a unique device ID.

cBarcodeReader class will additionally have the following attribute:

- a. **barcode**: an integer representing the barcode number

cNumberplateReader class will additionally have the following attribute:

- a. **numberplate**: a string representing the car numberplate

cIDReader class will have the following methods:

- a. **read()**: a private function that 'reads' an ID, and returns an integer. This function implements the getIntInput function from the cInputHelper class to obtain user input.
- b. **update()**: calls the read function and returns the user input.
- c. **cIDReader(int id)**: constructor for the cIDReader, sets readerID attribute value, and sets default values for idNumber and readerType.
- d. **setReaderType(int t)**: method that sets the type of reader (if it is to differ) – not used specifically within the given solution but added for completeness.
- e. **getReaderType()**: function that returns readerType as an integer.
- f. **getReaderID()**: function that returns readerID as an integer.

cBarcodeReader class will additionally have the following methods:

- g. **cBarcodeReader(int id)**: constructor for the cBarcodeReader, sets readerID = id parameter, and sets default values for barcode and readerType.
- a. **read()**: a private function that 'reads' a barcode and returns an integer value. This function implements the getIntInput function from the cInputHelper class to obtain user input.
- a. **update()**: calls the read function and returns the user input as an integer.

cNumberplateReader class will additionally have the following methods:

- a. **cNumberplateReader(int id)**: constructor for the cNumberplateReader, sets readerID = id parameter, and sets default values for numberplate and readerType.
- b. **read()**: a private function that 'reads' a numberplate and returns a string value representing the number plate. This function implements the getNumberplateInput function from the cInputHelper class to obtain user input.
- b. **update()**: calls the read function and returns the user input as a string.

2.4.5. cFullSign

The purpose of the cFullSign class is to represent the car park's full sign, which is switched on when the car park is full, and switched off when the car park is empty.

The cFullSign class will have the following single attribute:

- a. **lightOn**: a boolean value representing the state of the full sign light

The cFullSign class will have the following methods:

- a. **cFullSign()**: the constructor which will set the lightOn to false as default.
- b. **getStatus()**: a private function to return the current status of the full sign light as a boolean.
- c. **update()**: a function which calls the getStatus() method to return a boolean representing the current state of the full sign.
- d. **switchOn()**: a setter method which switches on the full sign by setting lightOn = true.
- e. **switchOff()**: a setter method which switches off the full sign by setting lightOn = false.

2.4.6. cValidateID

The purpose of the cValidateID class is to act as the layer of validation for the ID card. Normally ID verification would include a lookup to a database table, using SQL etc, however for the given mock solution, a simple validation method will be used.

The rule for a valid ID is simple: an ID must be between 1000 and 9999 and must be divisible by 2 MOD 0 (an even number).

The cValidateID class could currently be implemented using a static method, as the constructor does not define any items and it does not have any attributes, however, in the future the class may include a database layer, with various items being set within the constructor, and so it has been left.

The class will contain a single method along with the default constructor:

- a. **validateID(int id)**: This method will take an integer id as a parameter, and apply the logic described above. If the ID meets the criteria, a boolean true will be returned, otherwise, it will return false.

2.4.7. cLinkedList and cNode

It was decided that a singly linked list is to be used as a dynamic data structure for storing the car numberplates. A singly linked list is a type of data structure that is made up of nodes that are created using self-referential structures. Each of these nodes contain two parts: the data and the reference to the next node. Only the head (pointer) to the first list node is required to access the whole linked list. The final node in the list has a NULL head [4].

Following good object-oriented practice, creating a reusable class allows for future reuse, and would be the best solution.

The cLinkedList class contains class cNode - cNode would have no use outside of the linkedList class, and therefore they are combined. The cNode class represents each node of the linked list. It holds the pointer the next node, and also the data to be stored in that node.

The cNode class will have the following attributes:

- a. ***next**: a pointer to the next node
- b. **data**: the data in the node (if an integer)
- c. **strdata**: the data in the node (if a string) *note, tried to use TYPE identifier but experienced issues, therefore they are both defined and used separately.

It also will have the following methods:

- a. **cNode()**: default constructor method
- b. **~cNode()**: default destructor method
- c. **print()**: a method to print the data contained within the node (integer)
- d. **printStr()**: a method to print the data contained within the node (string)

cLinkedListClass will have the following attributes:

- a. **length**: an integer representing the length of the linked list
- b. ***head**: a pointer to the head (start) of the linked list.

cLinkedListClass will have the following methods:

- a. **cLinkedList()**: constructor method which sets the length to 0 and the head to null.
- b. **~cLinkedList()**: default destructor method
- c. **add(int data)**: a method to add an item to the linked list. Creates a new cNode, sets the data within that node, sets the new node to point to the node after current node, sets the current node to point to new node, and increments the size of the list.
- d. **add(string data)**: as above – overloading to allow for string data (used for numberplates). Demonstrates static polymorphism.
- e. **remove(int data)**: a function to remove an item from the linked list. Traverses through the linked list looking for the data in each node, if found, link the current pointer's pointer to the next node, delete the current pointer, and decrement the length of the list. If successful, returns a Boolean true, otherwise returns false.
- f. **remove(string data)**: as above - overloading to allow for string data (used for numberplates).
- g. **search(int data)**: Search function is used to traverse through the linked list to find a given value (data parameter). If no match is found, NULL will be returned, otherwise, a pointer to the node containing the data will be returned.
- h. **search(string data)**: as above – overloading to allow for string data (used for numberplates)
- i. **print()**: the print method will be used to print the current linked list contents.
- j. **printStr()**: as above, to print string data, such as for printing the list of current number plates.
- k. **size()**: the size function will return an integer value representing the size (length) of the linked list.

2.4.8. *cInputHelper*

The `cInputHelper` class has been included as a result of the consideration discussed in section 2.2. The purpose of the class is to provide static functions which prompt a user for a given input, validate that input and then return the inputted value. As a result, the class will have no attributes.

The `cInputHelper` class will contain the following static methods:

- a. **getBoolInput():** get bool input will prompt the user to input a boolean integer (0 or 1). It will then check for cin errors, such as if a char is entered. If this occurs, it will clear the cin buffer and ask for input again, continuing until valid input has been obtained. Additionally, if an integer value is inputted, it is then validated using the `cValidation` class's `validateBool`, which does a basic check (if 0 or 1) and returns true or false if valid/not. If not valid, the user will be prompted to enter a valid boolean, continuing in a loop until valid input is obtained.
- b. **getBoolInput(string prompt):** as above, using overloading to take a string parameter prompt, which will be used as the user prompt message.
- c. **getIntInput(string prompt, int min, int max):** getIntInput will follow the same structure as above, but will also require a min and max parameter. These parameters will be used as part of the validation which will check the integer inputted is within these boundaries. The validation once again will use the `cValidation` class, this time with the `validateInt` function. The valid int input value is returned.
- d. **getNumberplateInput(string prompt):** getNumberplateInput will follow a similar structure, requiring string input using `getline()`. Validation will be done using `cValidation`'s `validateNumberplate` function, which will check the input against a basic regular expression. The valid numberplate value is returned as a string.

2.4.9. *cValidation*

The `cValidation` method will be used by the `cInputHelper` class as described above, and will provide static methods that can be used to validate the inputs required by the system. These may be trivial validation methods, however in principle this class could be extended greatly to cover a vast array of inputs and requirements and can be reused within different solutions.

The static methods that will be included in the `cValidation` class are:

- a. **validateInt(int value, int min, int max):** this function could be used to validate any integer inputs in the system, and compares the value input to the provided min and max values. Will return true if valid, and false if invalid.
- b. **validateBool(int value):** validateBool function will implement a trivial check to see if an integer is a 1 or a 0 and return true if it is, and if it is not, will return false.
- c. **validateNumberplate(string value):** validateNumberplate will take a string parameter, and compare it to a regular expression of a number plate – this may be simplified for the current solution. If the string matches the regex, it will return true, and otherwise return false.

2.5. Final Solution

The final solution was made by directly following the class diagram and pseudocode outlined throughout section 2. Full source code can be found in Appendix B.

3. Testing

In order to test the car park system, and its classes and methods, both unit testing and functional user testing is to be used.

3.1. Unit Tests

Unit tests were created using the C++ Catch2 framework and are outlined in the file UnitTests.cpp. The single header catch.hpp file (noted in Appendix B) is required to run the unit tests. Catch is a simple to use unit testing framework for C++ [5].

Test Number	Test Type	Test Data	Reason	Expected Outcome	Actual Outcome	Pass/Fail
U-A1	Test Value	cCarPark * testCarParkCapacity5 = new cCarPark(5);	Test Car Park Capacity is Set to 5 Correctly	5	5	Pass
U-A2	Test Value	cCarPark * testCarParkCapacity5FreeSpaces2 = new cCarPark(16,2);	Test Car Park Capacity is Set to 16	16	16	Pass
U-A3	Test Value	cCarPark * testCarParkCapacity5FreeSpaces2 = new cCarPark(5,2);	Test Car Park Free Spaces is Set to 2	2	2	Pass
U-A4	Test Value	7, 2, 5	Test Car Park Set Free Spaces Setter Method sets the correct value	7, 2, 5	7, 2, 5	Pass
U-A5	Test Value	170, 1705454545	Test Car Park Set Free Spaces Setter Method (2) - (above 10 capacity) - should set to max capacity instead	10, 10	10, 10	Pass
U-A6	Test Value	cCarPark * testCarPark = new cCarPark(10,6);	Check that the GETTER method is returning expected value (6)	6	6	Pass
U-A7	Test Value	31, 0	Test Car Park Decrement Free Spaces Method correctly decrements spaces by 1	30, 0	30, 0	Pass
U-A8	Test Value	9, 3	Test Car Park Increment Free Spaces Method correctly increments spaces by 1	10, 4	10, 4	Pass

U-A9	Test Value	Max Capacity = 10, Test Value = 10	Test Car Park Increment Free Spaces Method with Max Capacity	10	10	Pass
U-A10	Test Value	Test Value = 0	Test Car Park Decrement Free Spaces Method with Max Capacity	0	0	Pass
U-B1	Test Return	call update() to get state	Test Barrier Constructor Sets up to false (also tests update() method returns correct state)	FALSE	FALSE	Pass
U-B2	Test Return	call raise() method	Test Barrier raise() sets up to true	TRUE	TRUE	Pass
U-B3	Test Return	call lower() method	Test Barrier lower() sets up to false	FALSE	FALSE	Pass
U-C1	Test Return	call update() to get state	Test FullSign Constructor Sets lightOn to false	FALSE	FALSE	Pass
U-C2	Test Return	call switchOn() method	Test FullSign SwitchOn sets lightOn to true	TRUE	TRUE	Pass
U-C3	Test Return	call switchOff() method	Test FullSign SwitchOff sets lightOn to false	FALSE	FALSE	Pass
U-D1	Test Value	cIDReader testIDReader(16);	Test IDReader Constructor Sets IDReader readerID to 16	16	16	Pass
U-D2	Test Value	testIDReader.setReaderType(12);	Test IDReader setReaderType method correctly sets reader type	12	12	Pass
U-D3	Test Value	cBarcodeReader testBarcodeReader(22);	Test BarcodeReader Constructor Sets BarcodeReader readerID to 22	22	22	Pass
U-D4	Test Value	testBarcodeReader.getReaderType()	Test BarcodeReader Constructor Sets ReaderType correctly	1	1	Pass
U-D5	Test Value	cNumberplateReader testNumberplateReader(7);	Test NumberplateReader Constructor Sets NumberplateReader readerID correctly	7	7	Pass

U-D6	Test Value	testNumberplateReader.getReaderType()	"Test NumberplateReader Constructor Sets ReaderType correctly	2	2	Pass
U-E1	Test Value	testLList->size()	Test LinkedList Constructor sets default values	0	0	Pass
U-E2	Test Value	9, 2, 17645585, 0, -123456	Test LinkedList (int) add function adds item to the list	!=NULL	!=NULL	Pass
U-E3	Test Value	abc, CE13NHM, A1	Test LinkedList (str) add function adds item to the list	!=NULL	!=NULL	Pass
U-E4	Test Value	44, 89, 5656734, -5545676	Test LinkedList (int) remove function removes item from the list	NULL	NULL	Pass
U-E5	Test Value	R45NMH, CE19 KLG, A1	Test LinkedList (str) remove function removes item from the list	NULL	NULL	Pass
U-E6	Test Return	1, 2, 3, 4, 5, 6	Test LinkedList size returns the correct list size	6	6	Pass
U-E7	Test Return	itemthatdoesnotexist	Test remove item that does not exist from linked list	FALSE	FALSE	Pass
U-F1	Test Value	cSensor testSensor(3);	Test Sensor Constructor sets the ID of sensor	3	3	Pass
U-F2	Test Return	cSensor testSensor12(12);	Test getSensorID method returns correct sensorID	12	12	Pass
U-G1	Test Value	1000, 2000, 2222	Test ID Validator correctly marks Valid IDs as valid	TRUE	TRUE	Pass
U-G2	Test Value	1, 10000, 999, -2222, 3333	Test ID Validator correctly marks invalid IDs as invalid (valid would be 1000-9999 and % 2)	FALSE	FALSE	Pass
U-H1	Test Value	[value=1, min=0, max=100] [value=99, min=0, max=100] [value=0, min=0, max=100] [value=100, min=0, max=100] [value=51, min=0, max=100]	Test Validator validateInt identifies valid input (valid, mid, valid boundary)	TRUE	TRUE	Pass

U-H2	Test Value	[value=-1, min=0, max=100] [value=101, min=0, max=100] [value=1005345454, min=0, max=100] [value=-105345454, min=0, max=100]	Test Validator validateInt identifies invalid input (invalid mid, invalid boundary, invalid extreme)	FALSE	FALSE	Pass
U-H3	Test Value	0, 1	Test Validator validateBool correctly identifies valid input (valid, mid, valid boundary)	TRUE	TRUE	Pass
U-H4	Test Value	[-1, 2, 35, 34545455]	Test Validator validateBool correctly identifies invalid input (invalid boundary, invalid, invalid extreme)	FALSE	FALSE	Pass
U-H5	Test Value	CE16 NMH, N6ABC, N6 ABC, A1, R44 PXC	Test Validator validateNumberplate correctly identifies valid input	TRUE	TRUE	Pass
U-H6	Test Value	1234abdf645457, 1233435454554535, TESTSTRING	Test Validator validateNumberplate correctly identifies invalid input	FALSE	FALSE	Pass
U-I1	Test Value	CE15MNQ, R44ABC	Test Add Function using LinkedList (search for number plate string to ensure it has been stored)	!=NULL	!=NULL	Pass
U-I2	Test Value	R44ABC, CE15MNQ, A5	Test Remove Function using LinkedList	==nullptr	==nullptr	Pass

Figure 2. Unit Testing Table

3.2.Functional User Testing

The purpose of the functional user testing is to test the functionality of the solution, and ensure that it works as expected, with the correct values and outputs presented to the user.

The following test plan outlines functional tests and the reason for each, in order to test the solution in addition to the unit tests.

Test Number	Test Parameters	Reason	Expected Outcome	Actual Outcome	Pass/Fail
F-1 [LA-0] (Test F1, Loop A, Iteration 0)	capacity = 10, freeSpaces = 10, entryCarPresent = 1, exitCarPresent = 0, idno = 2222, entryRegNo = "ABC123" carUnderEntryBarrier = 1, carUnderEntryBarrier = 0	Test Valid Entry Loop (valid ID, valid RegNo), ensure spaces are decremented and the numberplate is added to the list	freeSpaces = 9, number plate "ABC123" added to list	freeSpaces = 9, number plate "ABC123" added to list	Pass
F-2 [LA-1] (Test F2, Loop A, Iteration 1)	capacity = 10, freeSpaces = 9, entryCarPresent = 0, exitCarPresent = 1, exitRegNo = "ABC123" carUnderExitBarrier = 1, carUnderExitBarrier = 0	Valid Exit Loop (valid reg no - added from F-1), ensure spaces are incremented and the number plate is removed from the list	freeSpaces = 10, number plate "ABC123" removed from list	freeSpaces = 10, number plate "ABC123" removed from list	Pass
F-3 [LA-2] (Test F3, Loop A, Iteration 2)	None	Test the loop works correct when there are no triggers (no entry sensor or exit sensor events).	Continuation of loop with suitable print statements (print no of spaces)	Continuation of loop with suitable print statements (print no of spaces)	Pass

F-4 [LB-0]	capacity = 10, freeSpaces = 10, entryCarPresent = 1, exitCarPresent = 0, idno = ABDC, idno=A, idno=3333, entryRegNo = teststring123, entryRegNo = "ABC123" carUnderEntryBarrier = 0	Entry Loop, invalid ID (invalid data, and then invalid ID eg 3333) and invalid reg number. Ensure user is forced to re-input values until they are valid.	User must re-enter numberplate and idno until they are valid/sensible.	User had to re-enter numberplate and idno until they were valid/sensible.	Pass
F-5 [LB-1]	capacity = 10, freeSpaces = 9, entryCarPresent = 0, exitCarPresent = 1, exitRegNo = "123" carUnderExitBarrier = 1, carUnderExitBarrier = 0	Exit loop, input a number plate that is not on the list. Ensure the notification to ask for help is displayed with no functional errors.	Print statement to user to get help.	Print statement to user to get help.	Pass
F-6 [LC-0 & LC-1]	Loop Iteration 0: capacity = 1, freeSpaces = 1, entryCarPresent = 1, exitCarPresent = 0, idno = 2222, entryRegNo = "ABC123" carUnderEntryBarrier = 0, Loop Iteration 1: entryCarPresent = 1, exitCarPresent = 0, idno = 4444, entryRegNo = BNM123" carUnderEntryBarrier = 0	Valid Entry Loop, no free spaces. Ensure the fullSign is switched on during loop iteration i=0 and that the second car is not admitted to the car park.	FullSign switched on, car not admitted	FullSign switched on, car not admitted	Pass
F-7 [LC-2]	Loop Iteration 2: capacity = 1, freeSpaces = 0, entryCarPresent = 0, exitCarPresent = 1, exitRegNo = "ABC123" carUnderExitBarrier = 0	Valid Exit Loop, car leaves the full car park. Ensure the spaces are incremented, the fullSign is switched off during the loop iteration i=2 and that the number plate is removed from the list.	FullSign switched off, car removed from list	FullSign switched off, car removed from list	Pass

F-8 [LD0-4]	capacity = 5, freeSpaces = 5 Loop Iteration 0: entryCarPresent = 1, exitCarPresent = 0, idno = 1110, entryRegNo = "CA14YYU" carUnderEntryBarrier = 0 Loop Iteration 1: entryCarPresent = 1, exitCarPresent = 0, idno = 1000, entryRegNo = "R55NAM" carUnderEntryBarrier = 0 Loop Iteration 2: entryCarPresent = 1, exitCarPresent = 0, idno = 2000, entryRegNo = "WX10MAD" carUnderEntryBarrier = 0 Loop Iteration 3: entryCarPresent = 1, exitCarPresent = 0, idno = 3000, entryRegNo = "LK16JHU" carUnderEntryBarrier = 0 Loop Iteration 4: entryCarPresent = 1, exitCarPresent = 0, idno = 4000, entryRegNo = "K87KJK" carUnderEntryBarrier = 0	Valid Entry Loop, ensure spaces are decremented correctly and the number plates are added to the list	0 free spaces at end of loop iteration 4, full sign shown, all number plates displayed in list	0 free spaces at end of loop iteration 4, full sign shown, all number plates displayed in list	Pass
F-9 LE~	Capacity = 2, free spaces = 2 Loop Iteration 0: entryCarPresent = 1, exitCarPresent = 0, idno = 1000, entryRegNo = "K87KJK" carUnderEntryBarrier = 0 Loop Iteration 1: entryCarPresent = 1, exitCarPresent = 0, idno = 2000, entryRegNo = "R44NQA" carUnderEntryBarrier = 0 Loop Iteration 2: entryCarPresent = 0, exitCarPresent = 1, exitRegNo = "K87KJK" carUnderExitBarrier = 0	Add two cars (until car park full), and then remove two cars (2 spaces available). Ensure the outputs are presented to the user as expected.	Add first car – 1 space available. Add second car – 0 spaces available. Full Sign displayed. Remove first car – 1 space available. Remove second car – 2 spaces available.	Added first car – 1 space available. Added second car – 0 spaces available. Full Sign was displayed. Remove first car – 1 space available. Remove second car – 2	Pass

	Loop Iteration 3: entryCarPresent = 0, exitCarPresent = 1, exitRegNo = "R44NQA" carUnderExitBarrier = 0			spaces available.	
--	---	--	--	-------------------	--

Figure 3. Functional User Testing Table

4. User Guide

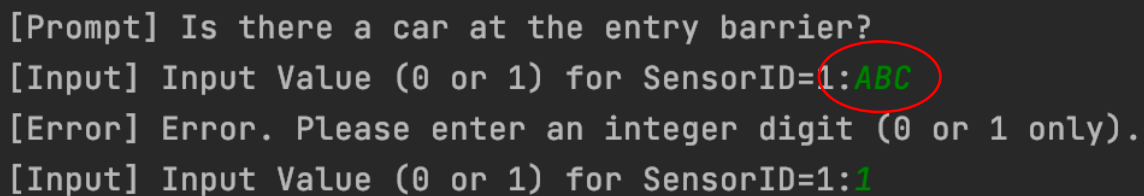
4.1. System Output Key

- a. [Input] - The system expects user input, see the specific prompt for information
- b. [Prompt] – A prompt message, often followed by [Input]
- c. [Error] – An error has occurred, see accompanying message for information
- d. [Info] – An information message displayed to the user
- e. [Event] – A message representing an event

4.2. System Input

When prompted by the system using a [Prompt] and [Input] key, please enter suitable input as required by the prompt, and then press enter to submit. An example is shown below, with green text representing the user input.

If invalid input is entered, the system will again request valid input from the user.

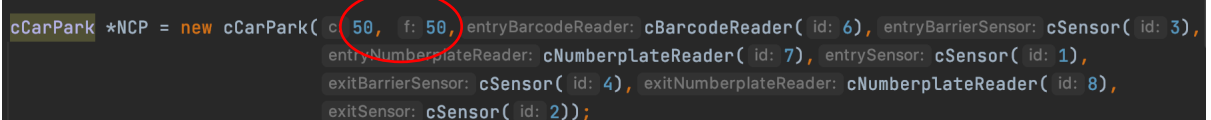


```
[Prompt] Is there a car at the entry barrier?  
[Input] Input Value (0 or 1) for SensorID=1:ABC  
[Error] Error. Please enter an integer digit (0 or 1 only).  
[Input] Input Value (0 or 1) for SensorID=1:1
```

Figure 4. User guide Input

4.3. How To

1. Set the required Carpark parameters (c = capacity, f = free spaces). Set the ID number for each of the attributes (barriers, sensors, readers).



```
cCarPark *NCP = new cCarPark( c: 50, f: 50, entryBarcodeReader: cBarcodeReader( id: 6), entryBarrierSensor: cSensor( id: 3),  
entryNumberplateReader: cNumberplateReader( id: 7), entrySensor: cSensor( id: 1),  
exitBarrierSensor: cSensor( id: 4), exitNumberplateReader: cNumberplateReader( id: 8),  
exitSensor: cSensor( id: 2));
```

Figure 5. User Guide Set Up

2. Run the program

3. The user will first be prompted to enter a value for the entry barrier. This is to replicate the state of the entry barrier. Enter a 1 or a 0 (1 is yes, 0 is no). The same process will occur for the exit barrier.

```
[Info] Car Park Created! Capacity: 50, Free Spaces: 50.
-----
[Info] Loop Count = 0: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:
```

Figure 6. User Guide Step 3

4. If there is a car at the entry barrier (replicated state 1), the user will be prompted to enter a value for the car number plate, and the ID card's barcode. To replicate a valid ID, enter a value between 1000 and 9999, that is divisible by 2 MOD 0. Such as 1000, 2000, 2222, 4000 etc. Any other value will be invalid.

```
[Event] * A Car is Present at the Entry to the Car Park *

[Input] Input Valid Numberplate:CA14YYY
[Input] Input ID Barcode Number (value between 1 and 9999:2222

[Event] * Entry Barrier is Raised *
```

Figure 7. User Guide Step 4

5. The user will then be prompted to replicate the entryBarrierSensor state. The system will loop until the car clears the entryBarrier (input 0).

```
[Event] * Entry Barrier is Raised *

[Prompt] Is there a car still under the entry barrier?
[Input] Input Value (0 or 1) for SensorID=3:1

[Info] ~ Car Still Under Entry Barrier ~
[Info] Poll entryBarrier sensor until the car has gone through the barrier

[Prompt] Is there a car still under the entry barrier?
[Input] Input Value (0 or 1) for SensorID=3:0

[Info] ~ Car is now through the Entry Barrier ~
[Event] * Entry Barrier is Lowered *
[Event Log] Car Registration: CA14YYY entered Car Park on Thu Feb  4 16:05:35 2021
```

Figure 8. User Guide Step 5

6. Once the car is clear of the entry barrier, the entry barrier is lowered, the number plate is added to the list, and a log is written stating the action (see [Info] and [Event]).

7. If there is a car at the exit barrier (replicated state 1), the user will be prompted to enter a value for the car number plate.

If the user enters a number plate that is not on the list, an error message will be shown.

8. Similarly to the entry barrier, the user will then be prompted to replicate the exitBarrierSensor state.

The system will loop until the car clears the exitBarrier (input 0). See below for a visual example of this scenario.

```
[Event] * Car Present at the Exit of the Car Park *

[Input] Input Valid Numberplate:CA14YYY
[Event] * Exit Barrier is Raised *

[Prompt] Is there a car still under the exit barrier?
[Input] Input Value (0 or 1) for SensorID=4:1
[Info] ~ Car Still Under Exit Barrier ~

[Prompt] Is there a car still under the exit barrier?
[Input] Input Value (0 or 1) for SensorID=4:0
[Info] ~ Car is Through the Exit Barrier ~

[Event] * Exit Barrier Lowered *

[Event Log] Car Registration: CA14YYY left Car Park on Thu Feb  4 16:16:03 2021
```

Figure 9. User Guide Step 8

9. Once the car is clear of the exit barrier, the exit barrier is lowered, the number plate is removed from the list, and a log is written stating the action (see [Info] and [Event]).

10. Full Sign information (current state), number of free spaces and the list of number plates of cars currently in the car park will also be displayed.

11. The program will loop, as per the loop iterations defined in main.cpp.

End of user guide

5. Conclusion

The Car Park solution provided demonstrates the key principles of Object-Oriented Programming: Encapsulation; Inheritance and Polymorphism [1].

The class design and implemented classes, attributes and methods, follow encapsulation, by combining the properties and methods related to each same object, without repetition. Inheritance has been used within cIDReader to created derived classes cBarcodeReader and cNumberplateReader. Static polymorphism was also included through the use of method overloading, which can be found throughout various constructor methods.

The solution is a working solution which meets the requirements and assignment brief, however, there are areas in which would benefit from future improvement. One of these areas is the cCarPark update method. A pitfall of the current solution is the current delay when waiting for user input for both the entryBarrierSensor and exitBarrierSensor – the solution waits until the car is clear of the barrier, which, if waiting for a car to enter the car park, could delay a car leaving the car park. It may be best to separate out the process so that they can both run in parallel.

Overall, the car park system has been a success. The testing conducted, both unit testing and functional user testing, covered the majority of use cases, classes and methods, and proves that the solution is working and producing output as required.

6. References

- [1] B. Stroustrup, "What is Object-Oriented Programming?," *IEEE Software*, vol. 5, no. 3, pp. 10–20, 1988, doi: 10.1109/52.2020.
- [2] "Destructors (C++ only)."
https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.4.0/com.ibm.zos.v2r4.cbclx01/cplr380.htm (accessed Jan. 12, 2021).
- [3] V. Mehtra Assistant Professor and P. Gupta, "Inheritance in C++ Programming Language," *International Journal of Scientific Research and Engineering Development*, vol. 2, Accessed: Jan. 12, 2021. [Online]. Available: www.ijrsred.com.
- [4] Y. Langsam, M. J. Augenstein, and A. M. Tenenbaum, "Data structures using C and C++," 2000, Accessed: Jan. 12, 2021. [Online]. Available: <http://localhost:8080/xmlui/handle/123456789/742>.
- [5] "GitHub - catchorg/Catch2: A modern, C++-native, header-only, test framework for unit-tests, TDD and BDD - using C++11, C++14, C++17 and later (or C++03 on the Catch1.x branch)." <https://github.com/catchorg/Catch2> (accessed Jan. 15, 2021).

7. Appendix

7.1. Appendix A – Testing Screenshots

7.1.1. Unit Testing

7.1.1.1. Full Unit Tests – 39 tests

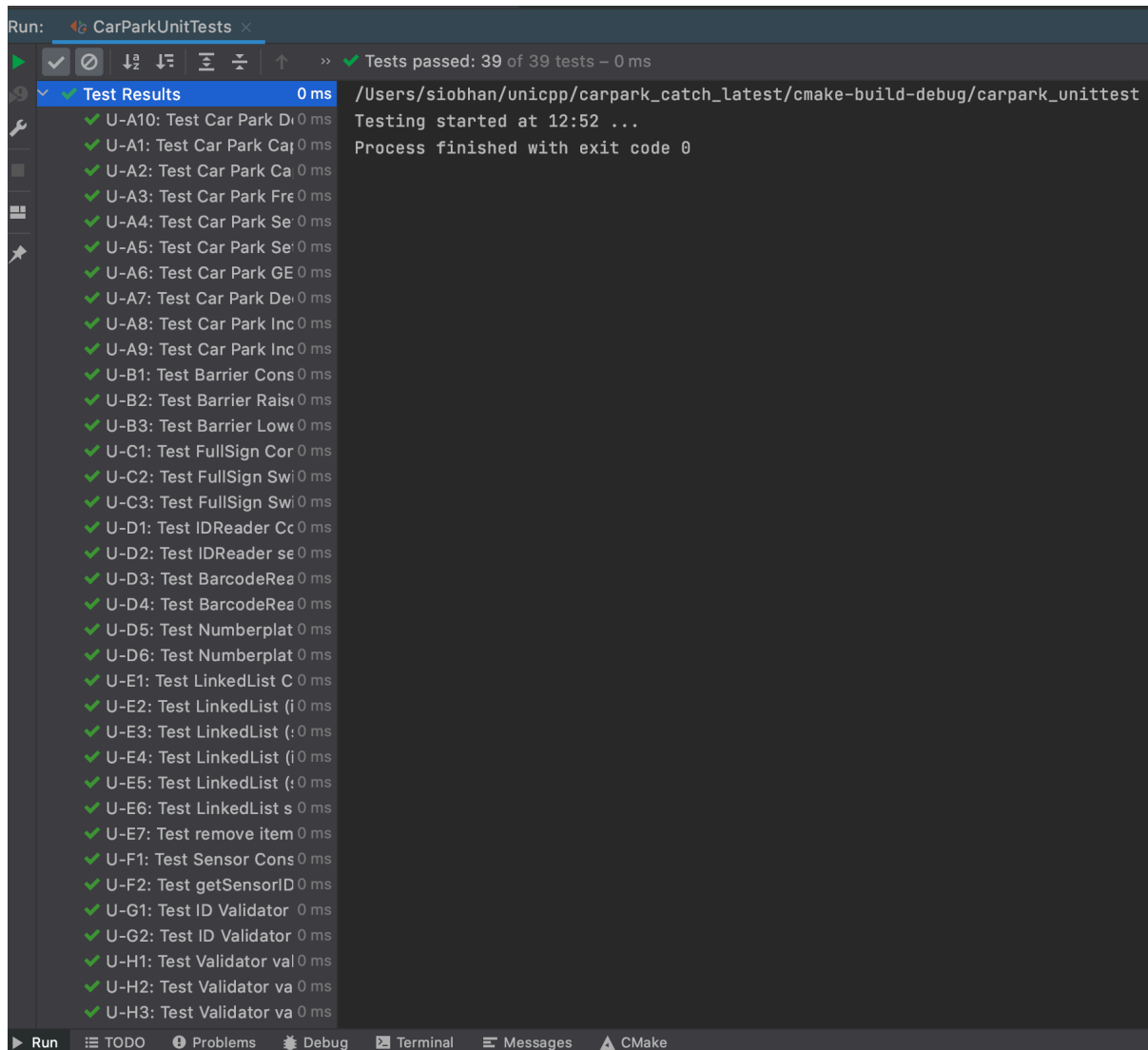


Figure 10. Screenshot of IDE run, showing full list of Unit Tests, 39 tests, 39 pass – Unit Testing Pass

7.1.2. Functional User Testing

7.1.2.1. Test F-1 Setup (Loop A)

```
int main()
{
    auto start = std::chrono::system_clock::now();//get the current time
    //Create new instance of CarPark NCP and initialise with the following values for test F-1
    //capacity = 10 (freeSpaces = 10)
    //entrySensor id = 1
    //exitSensor id = 2
    //entryBarrierSensor id = 3
    //exitBarrierSensor id = 4
    //entryBarcodeReader id = 6
    //entryNumberplateReader id = 7
    //exitNumberplateReader id = 8

    cCarPark *NCP = new cCarPark( c: 10, entryBarcodeReader: cBarcodeReader( id: 6), entryBarrierSensor: cSensor( id: 3),
    entryNumberplateReader: cNumberplateReader( id: 7), entrySensor: cSensor( id: 1),
    exitBarrierSensor: cSensor( id: 4), exitNumberplateReader: cNumberplateReader( id: 8),
    exitSensor: cSensor( id: 2));

    cout << endl << "Car Park Created! Capacity: " << NCP->getCapacity() << ", Free Spaces: " << NCP->getFreeSpaces() << endl;
    //Run the polling loop (loop 3 times for test F-1)
    for(int i=0;i<3;i++)
    {
        cout << "-----" << endl;
        cout << "[Info] Loop Count = " << i << ": Polling carpark components and reacting to state..." << endl << endl;
        NCP->update();
        cout << endl;
    }
    cout << "----- END -----" << endl;
    << "\n";
    delete NCP;
    auto end = std::chrono::system_clock::now();//get the current time
    std::chrono::duration<double> elapsed_seconds = end-start;//calculate the elapsed time
    std::cout << "[Info] Car Park System - Elapsed Time: " << elapsed_seconds.count() << "s\n";//print the elapsed time
    return 0;
};
```

Figure 11. F-1 Setup - main.cpp code

7.1.2.2. Test F-1 [LA-0] (Loop A Iteration 0)

```
Car Park Created! Capacity: 10, Free Spaces: 10.

-----
[Info] Loop Count = 0: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:1
[Prompt] Is there a car at the exit barrier?
[Input] Input Value (0 or 1) for SensorID=2:0

[Event] * A Car is Present at the Entry to the Car Park *

[Input] Input Valid Numberplate:ABC123
[Input] Input ID Barcode Number (value between 1 and 9999:2222

[Event] * Entry Barrier is Raised *

[Prompt] Is there a car still under the entry barrier?
[Input] Input Value (0 or 1) for SensorID=3:1

[Info] ~ Car Still Under Entry Barrier ~
[Info] Poll entryBarrier sensor until the car has gone through the barrier

[Prompt] Is there a car still under the entry barrier?
[Input] Input Value (0 or 1) for SensorID=3:0

[Info] ~ Car is now through the Entry Barrier ~
[Event] * Entry Barrier is Lowered *
[Event Log] Car Registration: ABC123 entered Car Park on Thu Feb  4 13:43:23 2021

[Info] 9 Spaces Available in the Car Park!
[Event] ***** Full Sign Switched Off *****
[Info] Current list of number plates (cars currently in the car park):
===== numberplate list start =====
1: ABC123
===== numberplate list end =====
```

Figure 12. Test F-1 LA-0 Output (valid car entering car park) - Test Pass

7.1.2.3. F-2 [LA-1] (Loop A Iteration 1)

```
===== numberplate list start =====
1: ABC123
===== numberplate list end =====

-----
[Info] Loop Count = 1: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:0
[Prompt] Is there a car at the exit barrier?
[Input] Input Value (0 or 1) for SensorID=2:1

[Event] * Car Present at the Exit of the Car Park *

[Input] Input Valid Numberplate:ABC123
[Event] * Exit Barrier is Raised *

[Prompt] Is there a car still under the exit barrier?
[Input] Input Value (0 or 1) for SensorID=4:1
[Info] ~ Car Still Under Exit Barrier ~

[Prompt] Is there a car still under the exit barrier?
[Input] Input Value (0 or 1) for SensorID=4:0
[Info] ~ Car is Through the Exit Barrier ~

[Event] * Exit Barrier Lowered *

[Event Log] Car Registration: ABC123 left Car Park on Thu Feb  4 13:46:31 2021

[Info] 10 Spaces Available in the Car Park!
[Event] ***** Full Sign Switched Off *****

-----
[Info] Loop Count = 2: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
```

Figure 13. F-2 LA-1 Output (valid car leaving the car park) - Test Pass.

7.1.2.4. F-3 [LA-2] (Loop A Iteration 2)

```
[Event] * Car Present at the Exit of the Car Park *

[Input] Input Valid Numberplate:ABC123
[Event] * Exit Barrier is Raised *

[Prompt] Is there a car still under the exit barrier?
[Input] Input Value (0 or 1) for SensorID=4:1
[Info] ~ Car Still Under Exit Barrier ~

[Prompt] Is there a car still under the exit barrier?
[Input] Input Value (0 or 1) for SensorID=4:0
[Info] ~ Car is Through the Exit Barrier ~

[Event] * Exit Barrier Lowered *

[Event Log] Car Registration: ABC123 left Car Park on Thu Feb  4 13:46:31 2021

[Info] 10 Spaces Available in the Car Park!
[Event] ***** Full Sign Switched Off *****

-----
[Info] Loop Count = 2: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:0
[Prompt] Is there a car at the exit barrier?
[Input] Input Value (0 or 1) for SensorID=2:0

[Info] 10 Spaces Available in the Car Park!
[Event] ***** Full Sign Switched Off *****

-----  END  -----
[Info] Car Park System - Elapsed Time: 314.96s

Process finished with exit code 0
```

Figure 14. F-3 LA-2. Continuation of loop with no event. No change – Test Pass.

7.1.2.5. Test F-4 LB-0 (Loop B Iteration 0)

```
Car Park Created! Capacity: 10, Free Spaces: 10.

-----
[Info] Loop Count = 0: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:1
[Prompt] Is there a car at the exit barrier?
[Input] Input Value (0 or 1) for SensorID=2:0

[Event] * A Car is Present at the Entry to the Car Park *

[Input] Input Valid Numberplate:teststring123
[Error] Error. Please enter a valid Numberplate.
[Input] Input Valid Numberplate:ABC123
[Input] Input ID Barcode Number (value between 1 and 9999:A
[Error] Error. Please enter an integer value between 1 and 9999:
[Input] Input ID Barcode Number (value between 1 and 9999:3333
[Error] Invalid ID Barcode Number. Please try again.
[Input] Input ID Barcode Number (value between 1 and 9999:4444

[Event] * Entry Barrier is Raised *

[Prompt] Is there a car still under the entry barrier?
[Input] Input Value (0 or 1) for SensorID=3:0

[Info] ~ Car is now through the Entry Barrier ~
[Event] * Entry Barrier is Lowered *
[Event Log] Car Registration: ABC123 entered Car Park on Thu Feb  4 14:00:49 2021

[Info] 9 Spaces Available in the Car Park!
[Event] ***** Full Sign Switched Off *****
[Info] Current list of number plates (cars currently in the car park):
===== numberplate list start =====
1: ABC123
===== numberplate list end =====
```

Figure 15. L4 LB-0 Test Input Values for Entry Numberplate and ID No (Type and Not Valid ID)- Test Pass

7.1.2.6. Test F-5 LC-1 (Loop C Iteration 0)

```
[Info] 9 Spaces Available in the Car Park!
[Event] ***** Full Sign Switched Off *****
[Info] Current list of number plates (cars currently in the car park):
===== numberplate list start =====
1: ABC123
===== numberplate list end =====

-----
[Info] Loop Count = 1: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:0
[Prompt] Is there a car at the exit barrier?
[Input] Input Value (0 or 1) for SensorID=2:1

[Event] * Car Present at the Exit of the Car Park *

[Input] Input Valid Numberplate:CA14YYU
[Error] Number plate not found. 'Press Help Button' to contact car park agent.

[Info] 9 Spaces Available in the Car Park!
[Event] ***** Full Sign Switched Off *****
[Info] Current list of number plates (cars currently in the car park):
===== numberplate list start =====
1: ABC123
===== numberplate list end =====

-----
[Info] Loop Count = 2: Polling carpark components and reacting to state...
```

Figure 16. Test F-5 LB-1 Try to exit with a number plate that does not exist. Print help statement shown – Test Pass.

7.1.2.7. Test F-6 Setup (free spaces = 1)

```
int main()
{
    auto start = std::chrono::system_clock::now();//get the current time
    //Create new instance of CarPark NCP and initialise with the following values for test F-1
    //capacity = 10 (freeSpaces = 1)
    //entrySensor id = 1
    //exitSensor id = 2
    //entryBarrierSensor id = 3
    //exitBarrierSensor id = 4
    //entryBarcodeReader id = 6
    //entryNumberplateReader id = 7
    //exitNumberplateReader id = 8

    cCarPark *NCP = new cCarPark( c: 10, f: 1, entryBarcodeReader: cBarcodeReader( id: 6), entryBarrierSensor: cSensor( id: 3),
                                   entryNumberplateReader: cNumberplateReader( id: 7), entrySensor: cSensor( id: 1),
                                   exitBarrierSensor: cSensor( id: 4), exitNumberplateReader: cNumberplateReader( id: 8),
                                   exitSensor: cSensor( id: 2));

    cout << endl << "Car Park Created! Capacity: " << NCP->getCapacity() << ", Free Spaces: " << NCP->getFreeSpaces() << endl;
    //Run the polling loop (loop 3 times for test F-1)
    for(int i=0;i<3;i++)
    {
        cout << "-----" << endl;
        cout << "[Info] Loop Count = " << i << ": Polling carpark components and reacting to state..." << endl << endl;
        NCP->update();
        cout << endl;
    }
    cout << "----- END -----" << endl;
    delete NCP;
    auto end = std::chrono::system_clock::now();//get the current time
    std::chrono::duration<double> elapsed_seconds = end-start;//calculate the elapsed time
    std::cout << "[Info] Car Park System - Elapsed Time: " << elapsed_seconds.count() << "s\n";//print the elapsed time
    return 0;
};
```

Figure 17. Setup for test F-6, note freeSpaces is set in constructor to 1.

7.1.2.8. Test F-6 LC-0

```
Car Park Created! Capacity: 10, Free Spaces: 1.
-----
[Info] Loop Count = 0: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:1
[Prompt] Is there a car at the exit barrier?
[Input] Input Value (0 or 1) for SensorID=2:0

[Event] * A Car is Present at the Entry to the Car Park *

[Input] Input Valid Numberplate:ABC123
[Input] Input ID Barcode Number (value between 1 and 9999:2222

[Event] * Entry Barrier is Raised *

[Prompt] Is there a car still under the entry barrier?
[Input] Input Value (0 or 1) for SensorID=3:0

[Info] ~ Car is now through the Entry Barrier ~
[Event] * Entry Barrier is Lowered *
[Event Log] Car Registration: ABC123 entered Car Park on Thu Feb  4 14:18:26 2021

[Info] Car Park is Full!
[Event] ***** Full Sign Switched On *****
[Info] Current list of number plates (cars currently in the car park):
===== numberplate list start =====
1: ABC123
===== numberplate list end =====
```

Figure 18. Loop Iteration 0 for Test F-6

7.1.2.9. Test F-6 LC-1

```
[Info] Car Park is Full!
[Event] ***** Full Sign Switched On *****
[Info] Current list of number plates (cars currently in the car park):
===== numberplate list start =====
1: ABC123
===== numberplate list end =====

-----
[Info] Loop Count = 1: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:1
[Prompt] Is there a car at the exit barrier?
[Input] Input Value (0 or 1) for SensorID=2:0

[Event] * A Car is Present at the Entry to the Car Park *

[Input] Input Valid Numberplate:BNM123
[Input] Input ID Barcode Number (value between 1 and 9999:4444

[Info] Car Park is Full!
[Event] ***** Full Sign Switched On *****
[Info] Current list of number plates (cars currently in the car park):
===== numberplate list start =====
1: ABC123
===== numberplate list end =====
```

Figure 19. F-6 LC-1. Try to enter car park that is full. Full sign is switched on and car is not allowed in – Test Pass.

7.1.2.10. Test F-7 LC-2:

```
[Event] ***** Full Sign Switched On *****
[Info] Current list of number plates (cars currently in the car park):
===== numberplate list start =====
1: ABC123
===== numberplate list end =====

-----
[Info] Loop Count = 2: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:0
[Prompt] Is there a car at the exit barrier?
[Input] Input Value (0 or 1) for SensorID=2:1

[Event] * Car Present at the Exit of the Car Park *

[Input] Input Valid Numberplate:ABC123
[Event] * Exit Barrier is Raised *

[Prompt] Is there a car still under the exit barrier?
[Input] Input Value (0 or 1) for SensorID=4:0
[Info] ~ Car is Through the Exit Barrier ~

[Event] * Exit Barrier Lowered *

[Event Log] Car Registration: ABC123 left Car Park on Thu Feb  4 14:22:36 2021

[Info] 1 Spaces Available in the Car Park!
[Event] ***** Full Sign Switched Off *****

----- END -----
[Info] Car Park System - Elapsed Time: 206.742s

Process finished with exit code 0
```

Figure 20. F-7 LC-2. Valid Car Exit Car Park - Test Pass

7.1.2.11. Test F-8 LD - Valid Entry Loop, ensure spaces are decremented correctly and the number plates are added to the list

```
Car Park Created! Capacity: 5, Free Spaces: 5.
-----
[Info] Loop Count = 0: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:1
[Prompt] Is there a car at the exit barrier?
[Input] Input Value (0 or 1) for SensorID=2:0

[Event] * A Car is Present at the Entry to the Car Park *

[Input] Input Valid Numberplate:CA14YYU
[Input] Input ID Barcode Number (value between 1 and 9999:1100

[Event] * Entry Barrier is Raised *

[Prompt] Is there a car still under the entry barrier?
[Input] Input Value (0 or 1) for SensorID=3:0

[Info] ~ Car is now through the Entry Barrier ~
[Event] * Entry Barrier is Lowered *
[Event Log] Car Registration: CA14YYU entered Car Park on Thu Feb  4 14:52:44 2021

[Info] 4 Spaces Available in the Car Park!
[Event] ***** Full Sign Switched Off *****
[Info] Current list of number plates (cars currently in the car park):
===== numberplate list start =====
1: CA14YYU
===== numberplate list end =====
```

Figure 21. F-8 LD-0 – Car 1 is successfully added to car park, spaces decremented – Test Pass

```

-----
[Info] Loop Count = 1: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:1
[Prompt] Is there a car at the exit barrier?
[Input] Input Value (0 or 1) for SensorID=2:0

[Event] * A Car is Present at the Entry to the Car Park *

[Input] Input Valid Numberplate:R55NAM
[Input] Input ID Barcode Number (value between 1 and 9999:1000

[Event] * Entry Barrier is Raised *

[Prompt] Is there a car still under the entry barrier?
[Input] Input Value (0 or 1) for SensorID=3:0

[Info] ~ Car is now through the Entry Barrier ~
[Event]* Entry Barrier is Lowered *
[Event Log] Car Registration: R55NAM entered Car Park on Thu Feb  4 14:53:00 2021

[Info] 3 Spaces Available in the Car Park!
[Event] ***** Full Sign Switched Off *****
[Info] Current list of number plates (cars currently in the car park):
===== numberplate list start =====
1: R55NAM
2: CA14YYU
===== numberplate list end =====

```

Figure 22. F-8 LD-1 - Car 2 is successfully added to car park, spaces decremented – Test Pass

```

-----
[Info] Loop Count = 2: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:1
[Prompt] Is there a car at the exit barrier?
[Input] Input Value (0 or 1) for SensorID=2:0

[Event] * A Car is Present at the Entry to the Car Park *

[Input] Input Valid Numberplate:WX10MAD
[Input] Input ID Barcode Number (value between 1 and 9999:2000

[Event] * Entry Barrier is Raised *

[Prompt] Is there a car still under the entry barrier?
[Input] Input Value (0 or 1) for SensorID=3:0

[Info] ~ Car is now through the Entry Barrier ~
[Event]* Entry Barrier is Lowered *
[Event Log] Car Registration: WX10MAD entered Car Park on Thu Feb  4 14:53:16 2021

[Info] 2 Spaces Available in the Car Park!
[Event] ***** Full Sign Switched Off *****
[Info] Current list of number plates (cars currently in the car park):
===== numberplate list start =====
1: WX10MAD
2: R55NAM
3: CA14YYU
===== numberplate list end =====

```

Figure 23. F-8 LD-2 - Car 3 is successfully added to car park, spaces decremented – Test Pass

```

-----
[Info] Loop Count = 3: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:1
[Prompt] Is there a car at the exit barrier?
[Input] Input Value (0 or 1) for SensorID=2:0

[Event] * A Car is Present at the Entry to the Car Park *

[Input] Input Valid Numberplate:LK16JHU
[Input] Input ID Barcode Number (value between 1 and 9999:3000

[Event] * Entry Barrier is Raised *

[Prompt] Is there a car still under the entry barrier?
[Input] Input Value (0 or 1) for SensorID=3:0

[Info] ~ Car is now through the Entry Barrier ~
[Event] * Entry Barrier is Lowered *
[Event Log] Car Registration: LK16JHU entered Car Park on Thu Feb  4 14:53:27 2021

[Info] 1 Spaces Available in the Car Park!
[Event] ***** Full Sign Switched Off *****
[Info] Current list of number plates (cars currently in the car park):
===== numberplate list start =====
1: LK16JHU
2: WX10MAD
3: R55NAM
4: CA14YYU
===== numberplate list end =====

```

Figure 24. F-8 LD3 - Car 4 is successfully added to car park, spaces decremented – Test Pass

```

-----
[Info] Loop Count = 4: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:1
[Prompt] Is there a car at the exit barrier?
[Input] Input Value (0 or 1) for SensorID=2:0

[Event] * A Car is Present at the Entry to the Car Park *

[Input] Input Valid Numberplate:K87KJK
[Input] Input ID Barcode Number (value between 1 and 9999:4000

[Event] * Entry Barrier is Raised *

[Prompt] Is there a car still under the entry barrier?
[Input] Input Value (0 or 1) for SensorID=3:0

[Info] ~ Car is now through the Entry Barrier ~
[Event] * Entry Barrier is Lowered *
[Event Log] Car Registration: K87KJK entered Car Park on Thu Feb  4 14:53:51 2021

[Info] Car Park is Full!
[Event] ***** Full Sign Switched On *****
[Info] Current list of number plates (cars currently in the car park):
===== numberplate list start =====
1: K87KJK
2: LK16JHU
3: WX10MAD
4: R55NAM
5: CA14YYU
===== numberplate list end =====

----- END -----
[Info] Car Park System - Elapsed Time: 85.1564s

```

Figure 25. F-8 LD4 – Car 5 is successfully added to car park, spaces decremented, full sign shown – Test Pass

7.1.2.12. F-9 LE – Test Increment Spaces to Full, and the Decrement Spaces to Empty

```
Car Park Created! Capacity: 2, Free Spaces: 2.
-----
[Info] Loop Count = 0: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:1
[Prompt] Is there a car at the exit barrier?
[Input] Input Value (0 or 1) for SensorID=2:0

[Event] * A Car is Present at the Entry to the Car Park *

[Input] Input Valid Numberplate:K87KJK
[Input] Input ID Barcode Number (value between 1 and 9999:1000

[Event] * Entry Barrier is Raised *

[Prompt] Is there a car still under the entry barrier?
[Input] Input Value (0 or 1) for SensorID=3:0

[Info] ~ Car is now through the Entry Barrier ~
[Event] * Entry Barrier is Lowered *
[Event Log] Car Registration: K87KJK entered Car Park on Thu Feb  4 15:06:09 2021

[Info] 1 Spaces Available in the Car Park!
[Event] ***** Full Sign Switched Off *****
[Info] Current list of number plates (cars currently in the car park):
===== numberplate list start =====
1: K87KJK
===== numberplate list end =====
```

Figure 26. F-9 LD-0. Car enter car park, spaces decremented correctly and number plate added to list – Pass

```

-----
[Info] Loop Count = 1: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:1
[Prompt] Is there a car at the exit barrier?
[Input] Input Value (0 or 1) for SensorID=2:0

[Event] * A Car is Present at the Entry to the Car Park *

[Input] Input Valid Numberplate:R44NQA
[Input] Input ID Barcode Number (value between 1 and 9999:2000

[Event] * Entry Barrier is Raised *

[Prompt] Is there a car still under the entry barrier?
[Input] Input Value (0 or 1) for SensorID=3:0

[Info] ~ Car is now through the Entry Barrier ~
[Event] * Entry Barrier is Lowered *
[Event Log] Car Registration: R44NQA entered Car Park on Thu Feb  4 15:06:29 2021

[Info] Car Park is Full!
[Event] ***** Full Sign Switched On *****
[Info] Current list of number plates (cars currently in the car park):
===== numberplate list start =====
1: R44NQA
2: K87KJK
===== numberplate list end =====

```

Figure 27. F-9 LD-1. Second car enters car park, spaces decremented correctly and number plate added to list. Full sign is also shown – Pass

```

-----
[Info] Loop Count = 2: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:0
[Prompt] Is there a car at the exit barrier?
[Input] Input Value (0 or 1) for SensorID=2:1

[Event] * Car Present at the Exit of the Car Park *

[Input] Input Valid Numberplate:K87KJK
[Event] * Exit Barrier is Raised *

[Prompt] Is there a car still under the exit barrier?
[Input] Input Value (0 or 1) for SensorID=4:0
[Info] ~ Car is Through the Exit Barrier ~

[Event] * Exit Barrier Lowered *

[Event Log] Car Registration: K87KJK left Car Park on Thu Feb  4 15:06:51 2021

[Info] 1 Spaces Available in the Car Park!
[Event] ***** Full Sign Switched Off *****
[Info] Current list of number plates (cars currently in the car park):
===== numberplate list start =====
1: R44NQA
===== numberplate list end =====

```

Figure 28. F-9 LD-2. First car leaves car park, spaces incremented correctly and number plate removed from the list. Full sign is also switched off – Pass

```

-----
[Info] Loop Count = 3: Polling carpark components and reacting to state...

[Prompt] Is there a car at the entry barrier?
[Input] Input Value (0 or 1) for SensorID=1:0
[Prompt] Is there a car at the exit barrier?
[Input] Input Value (0 or 1) for SensorID=2:1

[Event] * Car Present at the Exit of the Car Park *

[Input] Input Valid Numberplate:R44NQA
[Event] * Exit Barrier is Raised *

[Prompt] Is there a car still under the exit barrier?
[Input] Input Value (0 or 1) for SensorID=4:0
[Info] ~ Car is Through the Exit Barrier ~

[Event] * Exit Barrier Lowered *

[Event Log] Car Registration: R44NQA left Car Park on Thu Feb  4 15:07:09 2021

[Info] 2 Spaces Available in the Car Park!
[Event] ***** Full Sign Switched Off *****

```

Figure 29. F-9 LD-3. Second car leaves car park, spaces incremented correctly and number plate removed from the list. No list printed therefore it has cleared correctly, car park empty. – Pass

7.2. Appendix B – Final Solution Source Code

7.2.1. Barrier.cpp

```
#include "Barrier.h"
/**
 * cBarrier constructor
 * sets up to false as default
 */
cBarrier::cBarrier()
{
    up = false; //lower the barrier as default
}
/**
 * cBarrier raise
 * Raise method is used to set up to true (raise the barrier)
 */
void cBarrier::raise()
{
    //set up to true (to raise barrier)
    up = true;
}
/**
 * cBarrier lower
 * Lower method is used to set up to false (lower the barrier)
 */
void cBarrier::lower()
{
    //set up to false (to lower barrier)
    up = false;
}
/**
 * cBarrier getStatus
 * Getter function used to return the current status of the barrier (up)
 * @return up : bool
 */
bool cBarrier::getStatus()
{
    //return the current status of the barrier (up = true/raised or
    false/lowered)
    return up;
}
/**
 * cBarrier update
 * Update method is used to get the current status of the barrier,
 * by using the getStatus function
 * @return getStatus : bool
 */
bool cBarrier::update()
{
    return getStatus();
}
```

7.2.2. Barrier.h

```
#pragma once
#ifndef BARRIER_H
#define BARRIER_H
class cBarrier
{
public:
    cBarrier(void);
    bool update(void);
    void raise(void);
    void lower(void);

private:
    bool up;
    bool getStatus(void);
};
#endif
```

7.2.3. CarPark.cpp

```
#include <iostream>
#include <algorithm>
#include "CarPark.h"
#include <chrono>
#include <ctime>

using namespace std;

/**
 * cCarPark Constructor
 * Sets the default values, sets car park capacity and free spaces,
 * and calls the constructors for the attribute classes
 * @param c : int capacity
 * @param entryBarcodeReader : int id
 * @param entryBarrierSensor : int id
 * @param entryNumberplateReader : int id
 * @param entrySensor : int id
 * @param exitBarrierSensor : int id
 * @param exitNumberplateReader : int id
 * @param exitSensor : int id
 */
cCarPark::cCarPark(int c, cBarcodeReader entryBarcodeReader, cSensor
entryBarrierSensor,
                    cNumberplateReader entryNumberplateReader, cSensor
entrySensor, cSensor exitBarrierSensor,
                    cNumberplateReader exitNumberplateReader, cSensor
exitSensor)
    : entryBarcodeReader(entryBarcodeReader),
  entryBarrierSensor(entryBarrierSensor),
    entryNumberplateReader(entryNumberplateReader),
  entrySensor(entrySensor),
    exitBarrierSensor(exitBarrierSensor),
  exitNumberplateReader(exitNumberplateReader), exitSensor(exitSensor) {
    setCapacity(c); //set the capacity and number of free spaces (assume
car park is empty on startup)
    setFreeSpaces(c);
}

/**
 * cCarPark Constructor (overloading of above)
 * So that the user can also set the free spaces, eg if there are already
cars in the car park
 * however, this does not store their numberplates and they will need help
to leave the car park (manual override not implemented)
 * @param c - int capacity of car park
 * @param f - int freeSpaces in car park
 */
cCarPark::cCarPark(int c, int f, cBarcodeReader entryBarcodeReader, cSensor
entryBarrierSensor,
                    cNumberplateReader entryNumberplateReader, cSensor
entrySensor, cSensor exitBarrierSensor,
                    cNumberplateReader exitNumberplateReader, cSensor
exitSensor)
    : entryBarcodeReader(entryBarcodeReader),
  entryBarrierSensor(entryBarrierSensor),
    entryNumberplateReader(entryNumberplateReader),
  entrySensor(entrySensor),
    exitBarrierSensor(exitBarrierSensor),
  exitNumberplateReader(exitNumberplateReader), exitSensor(exitSensor) {
```

```

        setCapacity(c);
        setFreeSpaces(f);
    }

    /**
     * cCarPark setCapacity
     * Setter function to set car park capacity (total number of spaces)
     * @param c - int capacity
     */
    void cCarPark::setCapacity(int c) {
        //Set the capacity (total number of spaces) for the car park
        capacity = c;
    }

    /**
     * cCarPark setFreeSpaces
     * Setter function to set the number of free spaces in the car park
     * @param f - number of free spaces
     */
    void cCarPark::setFreeSpaces(int f) {
        //Set the number of free spaces in the car park
        if (f > capacity) {
            freeSpaces = capacity;
        } else {
            freeSpaces = f;
        }
    }

    /**
     * cCarPark getCapacity
     * Getter function to return the capacity of the car park
     * @return capacity : int
     */
    int cCarPark::getCapacity() {
        //return the full capacity of the car park (as set in constructor)
        return capacity;
    }

    /**
     * cCarPark incrementSpaces
     * Helper function to increment the number of free spaces by 1
     */
    void cCarPark::incrementSpaces() {
        //Increment the number of free spaces in the car park by one
        freeSpaces = freeSpaces + 1;
        if (freeSpaces > capacity) { //ensure free spaces cannot go above total
            freeSpaces = capacity;
        }
    }

    /**
     * cCarPark decrementSpaces
     * Helper function to decrement the number of free spaces by 1
     */
    void cCarPark::decrementSpaces() {
        //Decrement the number of free spaces in the car park by one
        freeSpaces = freeSpaces - 1;
        if (freeSpaces < 0) { //ensure free spaces cannot go below 0
            freeSpaces = 0;
        }
    }

```

```

}

/**
 * cCarPark getFreeSpaces
 * Getter function to return the current number of free spaces in the car
park
 * @return freeSpaces : int
 */
int cCarPark::getFreeSpaces() {
    //Get the current number of free spaces
    return freeSpaces;
}

/**
 * cCarPark log
 * Used to 'write' logs to a log file / database.
 * For the given mock solution, the function will only print the log
values,
 * however in the future this could be implemented with either a file
writer,
 * or a DB layer to store in a database.
 * @param action : string to store whether the car is going in or out
 * @param numberplate
 * @return boolean true if successful, false if unsuccessful (not
necessarily required for this
 * implementation, however when a writer to file or db layer is added, it
will be
 * important to understand if the logging was successful.
 */
bool cCarPark::log(int action, std::string numberplate) {
    std::string actionStr = "";
    switch (action) {
        case 1:
            actionStr = "entered";
            break;
        case 0:
            actionStr = "left";
            break;
        default:
            return false;
    }
    auto now = std::chrono::system_clock::now();
    std::time_t timestamp = std::chrono::system_clock::to_time_t(now);
    std::cout << "[Event Log] Car Registration: " << numberplate << " " <<
actionStr << " Car Park on "
                << std::ctime(&timestamp);
    return true;
}

// *****
/**
 * cCarParkUpdate
 * This is the controller function of the carpark system.
 * The comments within the function outline the logic of the system, which
is split into 3 main segments
 * 1. Poll the sensors
 * 2. Entry logic
 * 3. Exit logic
 * 4. Full Sign logic
 */
void cCarPark::update() {

```

```

//Declare and initialise required variables
//Note some variables are only required for this mock version of the
system (to replicate sensor data)

std::string entryRegNo = "";
std::string exitRegNo = "";
int idno = 0;

bool entryCarPresent = false;
bool exitCarPresent = false;
bool carUnderEntryBarrier = false;
bool carUnderExitBarrier = false;

//*****

/**
 * 1.
 * Poll the sensors at the entry and exit of the car park,
 * set the return values to variables entryCarPresent and
exitCarPresent
 */

    cout << "[Prompt] Is there a car at the entry barrier?" <<
endl; //prompt user to replicate sensor
    //the mock version of the system will gather user input to replicate
sensor when calling update function
    entryCarPresent = entrySensor.update();
    cout << "[Prompt] Is there a car at the exit barrier?" << endl; //prompt
user for input to replicate sensor
    //the mock version of the system will gather user input to replicate
sensor when calling update function
    exitCarPresent = exitSensor.update();

//*****

/**
 * 2.
 * If a car is present at the entry to the car park,
 * wait for them to input their ID card (triggered here manually by
user input to replicate)
 * Additionally, read the vehicle's number plate.
 * Next, check if there are free spaces in the car park.
 * If there are free spaces, run a check on the id card's barcode and
validate it.
 *     If the ID card is valid, raise the entry barrier and allow the
car to enter the car park
 *     wait for the car to pass the entryBarrier by polling the
entryBarrierSensor until it is clear
 *     once it is clear, lower the entry barrier and decrement the
number of free spaces
 *     If the ID card is not valid, notify the user to press the help
button in the car park
 * No free spaces logic will be considered below
 */

    if (entryCarPresent) {
        cout << endl << "[Event] * A Car is Present at the Entry to the Car
Park *" << endl << endl;
        //If car present at entrance then
        entryRegNo = entryNumberplateReader.update(); //read numberplate of
car

```

```

        //id reader needs to ask user for id input
        idno = entryBarcodeReader.update();//read barcode of id

        //If ID valid && car park not full then
        if (freeSpaces >= 1) {
            int retryCount = 0;
            bool idValid = false;
            //allow the user to try 3 times to enter valid input (already
once above), after that, display an error.
            while (retryCount < 2 && !idValid) {
                if (idValidator.validateID(idno)) {
                    idValid = true;
                    //Raise barrier and let car pass
                    entryBarrier.raise();
                    cout << endl << "[Event] * Entry Barrier is Raised *"
<< endl << endl;

                    //Replicate entryBarrierSensor state by asking user for
input.
                    cout << "[Prompt] Is there a car still under the entry
barrier?" << endl;
                    //Poll the entryBarrierSensor (will gather user input
to mock sensor state)
                    carUnderEntryBarrier = entryBarrierSensor.update();

                    //test the sensor at the entry barrier and lower the
barrier once the car has passed the barrier
                    //false = there isn't a car under the barrier / true =
the car is still under the barrier
                    //for the sake of the given mock program, the system
will continue to ask the user for input
                    //until the car is clear of the entry barrier.
                    while (carUnderEntryBarrier) {
                        cout << endl << "[Info] ~ Car Still Under Entry
Barrier ~" << endl;
                        cout << "[Info] Poll entryBarrier sensor until the
car has gone through the barrier" << endl << endl;
                        //Replicate entryBarrierSensor state by asking user
for input.
                        cout << "[Prompt] Is there a car still under the
entry barrier?" << endl;
                        //Poll the entryBarrierSensor (will gather user
input to mock sensor state)
                        carUnderEntryBarrier = entryBarrierSensor.update();
                    }
                    cout << endl << "[Info] ~ Car is now through the Entry
Barrier ~" << endl;
                    //Once the car is cleared of the barrier
                    entryBarrier.lower(); //lower the barrier
                    cout << "[Event] * Entry Barrier is Lowered *" << endl;
                    //Update records and decrement spaces
                    carNumberplateLList->add(entryRegNo);//add number plate
to the linked list
                    //Write the event to the log file
                    log(1, entryRegNo); //note return bool for error
checking isn't used in this solution
                    decrementSpaces();//decrement the number of free spaces
                } else {
                    cout << "[Error] Invalid ID Barcode Number. Please try
again." << endl;

```

```

        idno = entryBarcodeReader.update(); //read barcode of id
        retryCount++;
    }
}
if (!idValid) {
    cout << "[Error] Incorrectly Entered Invalid ID 3 times.
Please contact help." << endl;
}
}

// *****
/**
 * Ideally the two processes (entry and exit) should run in parallel
due to the
 * wait for user input/sensor data when waiting for a car to leave the
barriers.
 * However, it was felt by the author that given the mock solution that
this would
 * be a suitable compromise.
 */

// *****

/**
 * 3.
 * If a car is present at the exit to the car park,
 * read the vehicle's number plate.
 * If the number plate exists in the list of number plates (it should)
 *     Raise the exit barrier to allow the car to leave
 *     Poll the exitBarrierSensor to check that the car is clear of
the exit barrier
 *     until the car has left the car park.
 *     Once the car has left
 *     lower the exit barrier
 *     increment the number of free spaces
 *     remove the number plate from the list
 * If the number plate is not found, notify the user to contact help
 */

//If car present at exit then (sensor state gathered from user input
above)
if (exitCarPresent) {
    cout << endl << "[Event] * Car Present at the Exit of the Car Park
*" << endl << endl;
    //Read the number plate of the car exiting the car park
    exitRegNo = exitNumberplateReader.update();
    //check that the number plate exists in the list of number plates
    if (carNumberplateList->search(exitRegNo) != NULL) {
        exitBarrier.raise(); //raise barrier and let car pass to exit
        cout << "[Event] * Exit Barrier is Raised *" << endl << endl;

        //Replicate exitBarrierSensor state by asking user for input.
        cout << "[Prompt] Is there a car still under the exit barrier?"
<< endl;

        //Poll the exitBarrierSensor (will gather user input to mock
sensor state)
        carUnderExitBarrier = exitBarrierSensor.update();
        //test the sensor at the exit barrier and lower the barrier
once the car has passed the barrier
        //false = there isn't a car under the barrier / true = the car
is still under the barrier

```

```

        //for the sake of the given mock program, the system will
continue to ask the user for input
        //until the car is clear of the entry barrier.

        while (carUnderExitBarrier) {
            cout << "[Info] ~ Car Still Under Exit Barrier ~" << endl
<< endl;
            //Replicate exitBarrierSensor state by asking user for
input.
            cout << "[Prompt] Is there a car still under the exit
barrier?" << endl;
            //Poll the exitBarrierSensor (will gather user input to
mock sensor state)
            carUnderExitBarrier = exitBarrierSensor.update();
        }
        cout << "[Info] ~ Car is Through the Exit Barrier ~" << endl <<
endl;

        // Once the car is through, lower the barrier,
        // update records and increment number of free spaces
        exitBarrier.lower(); //lower the barrier

        cout << "[Event] * Exit Barrier Lowered *" << endl << endl;
        carNumberplateList->remove(exitRegNo); //remove number plate
from list of cars in car park
        incrementSpaces(); //increment number of free spaces
        //Write the event to the log file
        log(0, exitRegNo); //note return bool for error checking isn't
used in this solution
    } else {
        //If the number plate is not found in the list,
        //notify the user to contact help (manually override barrier
etc)
        cout << "[Error] Number plate not found. 'Press Help Button' to
contact car park agent." << endl;
    }
}

// *****
/**
 * 4.
 * Maintain full sign
 * If there are no free spaces in the car park, switch on the full sign
 * If there are free spaces in the car park, switch off the full sign
 */

cout << endl;
//If there are no free spaces
if (freeSpaces < 1) {
    cout << "[Info] Car Park is Full!" << endl; //print statement for
testing
    cout << "[Event] ***** Full Sign Switched On *****" <<
endl;
    //Switch on the full sign
    fullSign.switchOn();
} else {
    cout << "[Info] " << freeSpaces << " Spaces Available in the Car
Park!" << endl; //print statement for testing
    cout << "[Event] ***** Full Sign Switched Off *****" <<
endl;
    //Switch off the full sign
    fullSign.switchOff();
}

```

```

    }

    // *****
    if (carNumberplateLList->size() >= 1) {
        cout << "[Info] Current list of number plates (cars currently in
the car park):" << endl; //print statement for testing
        cout << "===== numberplate list start =====" << endl;
        carNumberplateLList->printStr(); //print the list of number plates
of cars currently in the car park
        cout << "===== numberplate list end =====" << endl;
    }
    // *****
}
/**
 * cCarPark destructor
 * delete instance of LinkedList so that the memory can be deallocated
 */
cCarPark::~cCarPark() {
    //Destructor for the car park. Delete the linked list.
    delete carNumberplateLList;
}

```

7.2.4. CarPark.h

```
#pragma once
#ifndef CARPARK_H
#define CARPARK_H
#include "FullSign.h"
#include "Barrier.h"
#include "Sensor.h"
#include "IDReader.h"
#include "ValidateID.h"
#include "LinkedList.h"
#include <string>

class cCarPark
{
public:
    cCarPark(int c, cBarcodeReader entryBarcodeReader, cSensor
entryBarrierSensor,
            cNumberplateReader entryNumberplateReader, cSensor
entrySensor, cSensor exitBarrierSensor,
            cNumberplateReader exitNumberplateReader, cSensor exitSensor);
    cCarPark(int c, int f, cBarcodeReader entryBarcodeReader, cSensor
entryBarrierSensor,
            cNumberplateReader entryNumberplateReader, cSensor
entrySensor, cSensor exitBarrierSensor,
            cNumberplateReader exitNumberplateReader, cSensor exitSensor);
    void update();
    void setCapacity(int c);
    void setFreeSpaces(int f);
    void decrementSpaces();
    void incrementSpaces();
    int getCapacity();
    int getFreeSpaces();
    ~cCarPark();

private:
    int capacity;
    int freeSpaces;
    bool log(int action, std::string numberplate);
    cLinkedList* carNumberplateLList = new cLinkedList;
    cSensor entrySensor;
    cSensor entryBarrierSensor;
    cSensor exitSensor;
    cSensor exitBarrierSensor;
    cBarcodeReader entryBarcodeReader;
    cNumberplateReader entryNumberplateReader;
    cNumberplateReader exitNumberplateReader;
    cFullSign fullSign;
    cBarrier entryBarrier;
    cBarrier exitBarrier;
    cValidateID idValidator;
};

#endif
```

7.2.5. FullSign.cpp

```
#include "FullSign.h"
/**
 * cFullSign constructor
 * Switches off the full sign as default
 */
cFullSign::cFullSign()
{
    //Constructor for Full Sign
    lightOn = false;
}
/**
 * cFullSign switchOn
 * switchOn method is used to set the lightOn to true, representing the
state of the full sign.
 */
void cFullSign::switchOn()
{
    lightOn = true;
}
/**
 * cFullSign switchOff
 * switchOff method is used to set the lightOn to false, representing the
state of the full sign.
 */
void cFullSign::switchOff()
{
    lightOn = false;
}
/**
 * cFullSign getStatus
 * The getStatus function is used to return the current state of the full
sign
 * The light can either be switched on (true) or switched off (false)
 * @return lightOn : bool
 */
bool cFullSign::getStatus()
{
    return lightOn;
}
/**
 * cFullSign update
 * The update method is used to return the current state of the full sign,
by using the getStatus function
 * @return getStatus() : bool
 */
bool cFullSign::update()
{
    return getStatus();
}
```

7.2.6. FullSign.h

```
#pragma once
#ifndef FULLSIGN_H
#define FULLSIGN_H
class cFullSign
{
public:
    cFullSign();
    bool update();
    void switchOn();
    void switchOff();
private:
    bool getStatus();
    bool lightOn;
};
#endif
```

7.2.7. IDReader.cpp

```
#include "IDReader.h"
#include "InputHelper.h"
using namespace std;

/**
 * Constructor for cIDReader
 * set default attribute values
 * @param id - used to set readerID
 */
cIDReader::cIDReader(int id)
{
    readerID = id;
    idNumber = 0;
    readerType = 0;
}

/**
 * cIDReader getReaderID
 * Setter method to set readerType
 * @param int t - type
 */
void cIDReader::setReaderType(int t){
    // set reader type. Not necessary for this solution however this could
    // be linked to
    // another system or be used to store an integer value representing the
    // type of reader/brand
    readerType = t;
}

/**
 * cIDReader getReaderType
 * Getter method to return readerType set during instantiation of object
 * @return readerType
 */
int cIDReader::getReaderType()
{
    return readerType;
}

/**
 * cIDReader getReaderID
 * Getter method to return readerID set during instantiation of object
 * @return readerID
 */
int cIDReader::getReaderID()
{
    return readerID;
}

/**
 * cIDReader read()
 * Method used to 'read' the current IDReader state.
 * If this were a working ID Reader, this would interact with a real
    sensor, likely
    using specialist libraries and interfaces.
    However, given the nature of the assignment, it was decided that it
    would simply ask the
    user for input representing the state of the sensor, and return the
    value of idNumber
 * @return idNumber - int
 */
int cIDReader::read()
{

```



```

    * @return result of call to read function
    */
int cBarcodeReader::update()
{
    return read();//call read method
}
//*****
/**
 * Constructor for cNumberplateReader
 * Sets default values including readerID, barcode and readerType
 * @param id - used to store ID of the number plate reader
 */
cNumberplateReader::cNumberplateReader(int id) : cIDReader(id) {
    numberplate = "";
    readerType = 2;
}
/**
 * cNumberplateReader read()
 * Method used to 'read' the car number plate.
 * If this were a working number plate reader, this would interact with a
real sensor, likely
 * using specialist libraries and interfaces.
 * However, given the nature of the assignment, it was decided that it
would simply ask the
 * user for input representing the state of the numberplate reader, and
return the value of numberPlate
 * @return int numberplate
 */
std::string cNumberplateReader::read()
{
    //ask user for input and read user input as if it was a real ID reader
    string inputString = "[Input] Input Valid Numberplate:";
    //getNumberplateInput takes the prompt string as a parameter
    numberplate = cInputHelper::getNumberplateInput(inputString);
    return numberplate;
}
/**
 * cNumberplateReader update()
 * update function is used to gather the current state of the reader, using
the read function.
 * @return result of call to read function
 */
std::string cNumberplateReader::update()
{
    return read();//call read function
}
//*****

```

7.2.8. IDReader.h

```
#pragma once
#ifndef IDREADER_H
#define IDREADER_H
#include <string>

class cIDReader
{
protected:
    int idNumber;
    int readerType;
    int readerID;
private:
    int read();
public:
    cIDReader(int id);
    int update();
    void setReaderType(int t);
    int getReaderType();
    int getReaderID();
};

class cBarcodeReader : public cIDReader
{
public:
    explicit cBarcodeReader(int id);
    int update();
private:
    int barcode;
    int read();
};

class cNumberplateReader : public cIDReader
{
public:
    explicit cNumberplateReader(int id);
    std::string update();
private:
    std::string numberplate;
    std::string read();
};
#endif
```

7.2.9. InputHelper.cpp

```
#include "InputHelper.h"
#include "Validation.h"
#include <iostream>
#include <string>

using namespace std;

/**
 * cInputHelper getBoolInput
 * Used to gather valid boolean (int) input from a user
 * Checks that input is a valid int (if statements check cin buffer)
 * If the input is not an int, returns an error, clears buffer and asks for
input again until valid
 * If a valid integer is input, it is then validated against the
cValidation class method validateBool,
 * which checks if an integer is 1 or 0. If this validation fails, the user
is prompted once again for valid input.
 * @return input : int - 0 or 1
 */
int cInputHelper::getBoolInput(){
    int input = 0; //declare and initialise input variable
    while(true) {
        cout << "[Input] Please enter an integer digit 0 (for false) or 1
(for true): ";
        if (cin >> input) {
            // read succeeded, now validate it is a valid bool int
            if (cValidation::validateBool(input)) {
                break;
            } else {
                cout << "[Error] Error. Please enter an integer digit (0 or
1 only)." << endl;
                continue;
            }
        } else if (cin.bad()) {
            // IO error
            cout << "[Error] Error, please try again." << endl;
            cin.clear(); //clear the cin buffer so that the loop can
continue
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        } else if (cin.eof()) {
            // EOF reached (perhaps combined with a format problem)
            cout << "[Error] Error, please try again." << endl;
            cin.clear(); //clear the cin buffer so that the loop can
continue
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        } else {
            // format problem
            cout << "[Error] Please enter an integer digit (0 or 1 only)."
<< endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        }
    }
    return input;
}
```

```

/**
 * cInputHelper getBoolInput (Overload function for above)
 * As above, used to gather valid boolean (int) input from a user
 * @param prompt - Takes string prompt as a parameter which is used as the
display message
 * @return input : int - 0 or 1
 */
int cInputHelper::getBoolInput(std::string prompt){
    int input = 0; //declare and initialise input variable
    while(true) {
        cout << prompt;
        if (cin >> input) {
            // read succeeded, now validate it is a valid bool int
            if (cValidation::validateBool(input)) {
                break;
            }else{
                cout << "[Error] Error. Please enter an integer digit (0 or
1 only)." << endl;
                continue;
            }
        } else if (cin.bad()) {
            // IO error
            cout << "[Error] Error, please try again." << endl;
            cin.clear(); //clear the cin buffer so that the loop can
continue
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        } else if (cin.eof()) {
            // EOF reached (perhaps combined with a format problem)
            cout << "[Error] Error, please try again." << endl;
            cin.clear(); //clear the cin buffer so that the loop can
continue
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        } else {
            // format problem
            cout << "[Error] Error. Please enter an integer digit (0 or 1
only)." << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        }
    }
    return input;
}
/**
 * cInputHelper getIntInput
 * Used to gather valid integer input from a user
 * Checks that input is a valid int (if statements check cin buffer)
 * If the input is not an int, returns an error, clears buffer and asks for
input again until valid
 * If a valid integer is input, it is then validated against the
cValidation class method validateInt,
 * with the parameters min and max, which checks if the integer is within
that range.
 * If this validation fails, the user is prompted once again for valid
input.
 * @param prompt - Takes string prompt as a parameter which is used as the
display message
 * @param min - minimum size of integer (used for comparison)
 * @param max - maximum size of integer (used for comparison)

```

```

* @return input : int - the valid integer user input
*/
int cInputHelper::getIntInput(std::string prompt, int min, int max) {
    int input = 0; // declare and initialise input variable
    while(true) {
        cout << prompt;
        if (cin >> input) {
            // read succeeded, now validate it is a valid integer value,
            within the min and max range using
            // Validation class validateInt
            if (cValidation::validateInt(input, min, max)) {
                break;
            } else {
                cout << "[Error] Error. Please enter an integer value
between " + std::to_string(min) + " and " + std::to_string(max) + ":" <<
endl;
                continue;
            }
        } else if (cin.bad()) {
            // IO error
            cout << "[Error] Error, please try again." << endl;
            cin.clear(); // clear the cin buffer so that the loop can
continue
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        } else if (cin.eof()) {
            // EOF reached (perhaps combined with a format problem)
            cout << "[Error] Error, please try again." << endl;
            cin.clear(); // clear the cin buffer so that the loop can
continue
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        } else {
            // format problem
            cout << "[Error] Error. Please enter an integer value between "
+ std::to_string(min) + " and " + std::to_string(max) + ":" << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        }
    }
    return input; // return the input value
}

/**
* cInputHelper getNumberplateInput
* Used to gather valid string numberplate input from a user
* The string input is validated using Validation class validateNumberplate
method.
* If the input is not deemed to be a valid numberplate, the user is
prompted once again for valid input.
* @param prompt - Takes string prompt as a parameter which is used as the
display message
* @return input : string - the valid numberplate string user input
*/
std::string cInputHelper::getNumberplateInput(std::string prompt) {
    std::cin.ignore(); // clear the buffer in case of leftover values,
especially important for getline()
    std::string input = ""; // declare and initialise input variable
    while(true) {
        cout << prompt;
        if (getline(cin, input)) {

```

```

        // read succeeded, now validate it is a valid numberplate value
using Validation class validateNumberplate
        if (cValidation::validateNumberplate(input)) {
            break;
        }else{
            cout << "[Error] Error. Please enter a valid Numberplate."
<< endl;
            continue;
        }
    } else if (cin.bad()) {
        // IO error
        cout << "[Error] Error, please try again." << endl;
        cin.clear();//clear the cin buffer so that the loop can
continue
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        continue;
    } else if (cin.eof()) {
        // EOF reached (perhaps combined with a format problem)
        cout << "[Error] Error, please try again." << endl;
        cin.clear();//clear the cin buffer so that the loop can
continue
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        continue;
    } else {
        // format problem
        cout << "[Error] Error. Please enter a valid Numberplate." <<
endl;cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        continue;
    }
}
return input;//return the input value
}

```

7.2.10. *InputHelper.h*

```
#include "InputHelper.h"
#include "Validation.h"
#include <iostream>
#include <string>

using namespace std;

/**
 * cInputHelper getBoolInput
 * Used to gather valid boolean (int) input from a user
 * Checks that input is a valid int (if statements check cin buffer)
 * If the input is not an int, returns an error, clears buffer and asks for
input again until valid
 * If a valid integer is input, it is then validated against the
cValidation class method validateBool,
 * which checks if an integer is 1 or 0. If this validation fails, the user
is prompted once again for valid input.
 * @return input : int - 0 or 1
 */
int cInputHelper::getBoolInput(){
    int input = 0; //declare and initialise input variable
    while(true) {
        cout << "[Input] Please enter an integer digit 0 (for false) or 1
(for true): ";
        if (cin >> input) {
            // read succeeded, now validate it is a valid bool int
            if (cValidation::validateBool(input)) {
                break;
            } else {
                cout << "[Error] Error. Please enter an integer digit (0 or
1 only)." << endl;
                continue;
            }
        } else if (cin.bad()) {
            // IO error
            cout << "[Error] Error, please try again." << endl;
            cin.clear(); //clear the cin buffer so that the loop can
continue
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        } else if (cin.eof()) {
            // EOF reached (perhaps combined with a format problem)
            cout << "[Error] Error, please try again." << endl;
            cin.clear(); //clear the cin buffer so that the loop can
continue
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        } else {
            // format problem
            cout << "[Error] Please enter an integer digit (0 or 1 only)."
<< endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        }
    }
    return input;
}
```

```

/**
 * cInputHelper getBoolInput (Overload function for above)
 * As above, used to gather valid boolean (int) input from a user
 * @param prompt - Takes string prompt as a parameter which is used as the
display message
 * @return input : int - 0 or 1
 */
int cInputHelper::getBoolInput(std::string prompt){
    int input = 0; //declare and initialise input variable
    while(true) {
        cout << prompt;
        if (cin >> input) {
            // read succeeded, now validate it is a valid bool int
            if (cValidation::validateBool(input)) {
                break;
            } else {
                cout << "[Error] Error. Please enter an integer digit (0 or
1 only)." << endl;
                continue;
            }
        } else if (cin.bad()) {
            // IO error
            cout << "[Error] Error, please try again." << endl;
            cin.clear(); //clear the cin buffer so that the loop can
continue
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        } else if (cin.eof()) {
            // EOF reached (perhaps combined with a format problem)
            cout << "[Error] Error, please try again." << endl;
            cin.clear(); //clear the cin buffer so that the loop can
continue
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        } else {
            // format problem
            cout << "[Error] Error. Please enter an integer digit (0 or 1
only)." << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        }
    }
    return input;
}
/**
 * cInputHelper getIntInput
 * Used to gather valid integer input from a user
 * Checks that input is a valid int (if statements check cin buffer)
 * If the input is not an int, returns an error, clears buffer and asks for
input again until valid
 * If a valid integer is input, it is then validated against the
cValidation class method validateInt,
 * with the parameters min and max, which checks if the integer is within
that range.
 * If this validation fails, the user is prompted once again for valid
input.
 * @param prompt - Takes string prompt as a parameter which is used as the
display message
 * @param min - minimum size of integer (used for comparison)
 * @param max - maximum size of integer (used for comparison)

```

```

* @return input : int - the valid integer user input
*/
int cInputHelper::getIntInput(std::string prompt, int min, int max) {
    int input = 0; // declare and initialise input variable
    while(true) {
        cout << prompt;
        if (cin >> input) {
            // read succeeded, now validate it is a valid integer value,
            within the min and max range using
            // Validation class validateInt
            if (cValidation::validateInt(input, min, max)) {
                break;
            } else {
                cout << "[Error] Error. Please enter an integer value
between " + std::to_string(min) + " and " + std::to_string(max) + ":" <<
endl;
                continue;
            }
        } else if (cin.bad()) {
            // IO error
            cout << "[Error] Error, please try again." << endl;
            cin.clear(); // clear the cin buffer so that the loop can
continue
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        } else if (cin.eof()) {
            // EOF reached (perhaps combined with a format problem)
            cout << "[Error] Error, please try again." << endl;
            cin.clear(); // clear the cin buffer so that the loop can
continue
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        } else {
            // format problem
            cout << "[Error] Error. Please enter an integer value between "
+ std::to_string(min) + " and " + std::to_string(max) + ":" << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        }
    }
    return input; // return the input value
}

/**
* cInputHelper getNumberplateInput
* Used to gather valid string numberplate input from a user
* The string input is validated using Validation class validateNumberplate
method.
* If the input is not deemed to be a valid numberplate, the user is
prompted once again for valid input.
* @param prompt - Takes string prompt as a parameter which is used as the
display message
* @return input : string - the valid numberplate string user input
*/
std::string cInputHelper::getNumberplateInput(std::string prompt) {
    std::cin.ignore(); // clear the buffer in case of leftover values,
especially important for getline()
    std::string input = ""; // declare and initialise input variable
    while(true) {
        cout << prompt;
        if (getline(cin, input)) {

```

```

        // read succeeded, now validate it is a valid numberplate value
using Validation class validateNumberplate
        if (cValidation::validateNumberplate(input)) {
            break;
        }else{
            cout << "[Error] Error. Please enter a valid Numberplate."
<< endl;
            continue;
        }
    } else if (cin.bad()) {
        // IO error
        cout << "[Error] Error, please try again." << endl;
        cin.clear();//clear the cin buffer so that the loop can
continue
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        continue;
    } else if (cin.eof()) {
        // EOF reached (perhaps combined with a format problem)
        cout << "[Error] Error, please try again." << endl;
        cin.clear();//clear the cin buffer so that the loop can
continue
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        continue;
    } else {
        // format problem
        cout << "[Error] Error. Please enter a valid Numberplate." <<
endl;cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        continue;
    }
}
return input;//return the input value
}

```

7.2.11. *LinkedList.cpp*

```
#include <iostream>
#include "LinkedList.h"

using namespace std;

/**
 * Default constructor for cNode
 */
cNode::cNode() = default;
/**
 * Destructor for cNode
 */
cNode::~~cNode()
{
    //Destructor for nodes
    // cout << "Destructor: Node has been deleted" << endl; //testing
    statement
}
/**
 * cNode print (int version)
 * Method used to print the contents of the node
 */
void cNode::print()
{
    //Print the contents of the node
    cNode *node = this;
    if (node != nullptr)
    {
        cout << this->data << endl;
    }
}
/**
 * cNode print (string version)
 * Method used to print the contents of the node
 */
void cNode::printStr()
{
    //Print the contents of the node
    cNode *node = this;
    if (node != nullptr)
    {
        cout << this->strdata << endl;
    }
}

//*****

/**
 * Constructor for cLinkedList
 * Sets length and initialises head
 */
cLinkedList::cLinkedList()
{
    this->length = 0;
    this->head = nullptr;
}
/**
 * Destructor for cLinkedList
 */
```

```

cLinkedList::~cLinkedList()
{
    // cout << "Destructor- Linked List deleted" << endl; //testing
    statement
}
/**
 * cLinkedList add (int)
 * Add method is used to add data to the linked list.
 * It creates a new code, and adds a pointer to the new node.
 * @param data : int - the data to be added
 */
void cLinkedList::add(int data)
{
    //Add an item to the linked list
    cNode *node = new cNode(); //Create a new node
    node->data = data; //Set the node data to the parameter data
    node->next = this->head; //Get new node to point to the node after
current node
    this->head = node; //Get current node to point to new node
    this->length++; //Increment the size of the list
}
/**
 * cLinkedList add (overloading for string)
 * Add method is used to add data to the linked list.
 * It creates a new code, and adds a pointer to the new node.
 * @param data : string - the data to be added
 */
void cLinkedList::add(std::string data)
{
    //Add an item to the linked list
    cNode *node = new cNode(); //Create a new node
    node->strdata = data; //Set the node data to the parameter data
    node->next = this->head; //Get new node to point to the node after
current node
    this->head = node; //Get current node to point to new node
    this->length++; //Increment the size of the list
}
/**
 * cLinkedList remove (int)
 * Remove is used to remove a node from the linked list
 * @param data : int - data to be removed
 * @return boolean true (if successful) or false (if unsuccessful)
 */
bool cLinkedList::remove(int data)
{
    /**
     * To remove an item in a linked list we must keep track of the
previous node,
     * and make the next pointer point to the current node's next node.
     * Consideration must be made for when the node to be removed IS the
first node.
     */
    if (this->length == 0)
    {
        //get list length, if empty, return error/false
        return false;
    }
    else if (this->head->data == data) //if the item is at the head node
    {
        cNode *current = head;
        this->head = this->head->next; //link the current pointer's pointer

```

```

to the next node
    delete current; //delete pointer
    this->length--; //decrement the length of the list
    return true; //return success
}
else
{ //else loop through the list to look for the data
    cNode *previous = this->head;
    cNode *current = head->next;
    while (current != nullptr)
    { //while loop through linked list
        if (current->data == data)
        { //if the data is at the current node, leave loop
            break;
        }
        else
        { //point to the next node to be searched in the loop
            previous = current;
            current = current->next;
        }
    }
    if (current == nullptr)
    { //if at the end of the list and no value found, return error
        // cout << "Can't remove value: no match found.\n"; //testing
statement
        return false; //return error/false
    }
    else
    { //else, 'remove' the item by changing the pointer to the next
node
        //node the item is not deleted, it simply tells the operating
system that
        //we are no longer using the location in memory so that it can
reclaim it for other uses.
        previous->next = current->next;
        delete current;
        this->length--;
        // cout << "Item removed from list" << endl; //testing statement
        return true; //return success
    }
}
}

/**
 * cLinkedList remove (overloading for string)
 * Remove is used to remove a node from the linked list
 * @param data : string - data to be removed
 * @return boolean true (if successful) or false (if unsuccessful)
 */
bool cLinkedList::remove(std::string data)
{
    /**
     * To remove an item in a linked list we must keep track of the
previous node,
     * and make the next pointer point to the current node's next node.
     * Consideration must be made for when the node to be removed IS the
first node.
     */
    if (this->length == 0)
    {
        //get list length, if empty, return error/false
        return false;
    }
}

```

```

    }
    else if (!this->head->strdata.compare(data)) //if the data item is at
the head node
    {
        cNode *current = head;
        this->head = this->head->next; //link the current pointer's pointer
to the next node
        delete current; //delete pointer
        this->length--; //decrement the length of the list
        return true; //return success
    }
    else
    { //else loop through the list to look for the data
        cNode *previous = this->head;
        cNode *current = head->next;
        while (current != nullptr)
        { //while loop through linked list
            if (!current->strdata.compare(data))
            { //if the data is at the current node, leave loop
                break;
            }
            else
            { //point to the next node to be searched in the loop
                previous = current;
                current = current->next;
            }
        }
        if (current == nullptr)
        { //if at the end of the list and no value found, return error
            // cout << "Can't remove value: no match found.\n";//testing
statement
            return false; //return error/false
        }
        else
        {
            //else, 'remove' the item by changing the pointer to the next
node
            //node the item is not deleted, it simply tells the operating
system that
            //we are no longer using the location in memory so that it can
reclaim it for other uses.
            previous->next = current->next;
            delete current;
            this->length--;
            return true; //return success
        }
    }
}
/**
 * cLinkedList search (int)
 * Search function is used to traverse through the linked list to find a
given value (data parameter)
 * If no match is found, nullptr is returned
 * If a match is found, the pointer to node containing the data is returned
 * @param data : int - the data to be searched for
 * @return pointer to the node containing the data
 */
cNode *cLinkedList::search(int data)
{
    //Search for an item within the linked list, returns the pointer to the
node as the result

```

```

    cNode *head = this->head; //set the node of the head of the linked list
    while (head)
    { //compare the node with the item to be searched for
        if (head->data == data)
        {
            return head; //if match, return the pointer to the node
        }
        else
        {
            head = head->next; //else increment the pointer to the next
node to be searched within the loop
        }
    }
    return nullptr;
}
/**
 * cLinkedList search (Overloading for string)
 * Search function is used to traverse through the linked list to find a
given value (data parameter)
 * If no match is found, nullptr is returned
 * If a match is found, the pointer to node containing the data is returned
 * @param data : string - the data to be searched for
 * @return pointer to the node containing the data
 */
cNode *cLinkedList::search(std::string data)
{
    //Search for an item within the linked list, returns the pointer to the
node as the result
    cNode *head = this->head; //set the node of the head of the linked list
    while (head)
    { //compare the node with the item to be searched for
        if (!head->strdata.compare(data))
        {
            return head; //if match, return the pointer
        }
        else
        {
            head = head->next; //else increment the pointer to the next
node to be searched within the loop
        }
    }
    return nullptr;
}
/**
 * cLinkedList print (int)
 * Print method is used to traverse through the linked list and print the
value in each node
 */
void cLinkedList::print()
{
    //if the list is not empty, loop through each node and print the data
using cout
    if(this->length !=0){
        cNode *head = this->head;
        int i = 1;
        while (head)
        {
            cout << i << ": " << head->data << endl;
            head = head->next;
            i++;
        }
    }
}

```

```

    }
}
/**
 * cLinkedList print (int)
 * Print method is used to traverse through the linked list and print the
value in each node
 */
void cLinkedList::printStr()
{
    //if the list is not empty, loop through each node and print the data
using cout
    if(this->length !=0){
        cNode *head = this->head;
        int i = 1;
        while (head)
        {
            cout << i << ": " << head->strdata << endl;
            head = head->next;
            i++;
        }
    }
}
/**
 * cLinkedList size
 * Size is a getter function used to return the size/length of the linked
list
 * @return length : int - list length
 */
int cLinkedList::size()
{
    //Return the size/length of the linked list
    return this->length;
}

```

7.2.12. *LinkedList.h*

```
#pragma once
#ifndef LINKEDLIST_H
#define LINKEDLIST_H
#include <string>
#include <vector>
/**
 * The LinkedList class contains a reference of Node class type.
 * This kind of implementation is mainly because of preservation of the Node
 * within a LinkedList for increased encapsulation.
 * Inner classes (he cNode class) are used especially when that class has no
 * other
 * use other than using inside another single class.
 * In this implementation Node is declared as an inner class because the
 * node can exist only in a linked list.
 * Without linked list a single node will be of no use.
 * And this Node class can only be instantiated right after instantiation of
 * linked list.
 * Thus, this supports the concept that no node can exist without a linked
 * list.
 * So typically this LinkedList and Node are bound together so that Node
 * couldn't exist without creating a LinkedList.
 * References:
 * [1] M. Goodrich, R. Tamassia and D. Mount, Data Structures and Algorithms
 * in C++, 2nd ed. John Wiley & Sons, 2011, p. 124.
 */
class cNode
{
public:
    cNode *next;
    int data;
    std::string strdata;
    cNode();
    ~cNode();
    void print();
    void printStr();
};
class cLinkedList
{
private:
    int length;
    cNode *head;
public:
    cLinkedList();
    ~cLinkedList();
    void add(int data);
    void add(std::string data);
    bool remove(int data);
    bool remove(std::string data);
    cNode *search(int data);
    cNode *search(std::string data);
    void print();
    void printStr();
    int size();
};
#endif
```

7.2.13. *main.cpp*

```
//main.cpp
#include "CarPark.h"
#include "LinkedList.h"
#include <iostream>
#include <chrono>
#include <ctime>
using namespace std;

/**
 * Main method of program.
 * Used to create a new instance of class cCarPark and
 * poll the update function of the car park.
 * @return
 */
int main()
{
    auto start = std::chrono::system_clock::now();//get the current time
    //Create new instance of CarPark NCP and initialise with the following
    values for test F-1
    //capacity = 10 (freeSpaces = 10)
    //entrySensor id = 1
    //exitSensor id = 2
    //entryBarrierSensor id = 3
    //exitBarrierSensor id = 4
    //entryBarcodeReader id = 6
    //entryNumberplateReader id = 7
    //exitNumberplateReader = 8

    cCarPark *NCP = new cCarPark(10, 10, cBarcodeReader(6), cSensor(3),
                                cNumberplateReader(7), cSensor(1),
                                cSensor(4), cNumberplateReader(8),
                                cSensor(2));

    cout << endl << "[Info] Car Park Created! Capacity: " << NCP-
>getCapacity() << ", Free Spaces: " << NCP->getFreeSpaces() << "." << endl;
    //Run the polling loop (loop 10 times to test solution)
    for(int i=0;i<10;i++)
    {
        cout <<
"
" << endl;
        cout << "[Info] Loop Count = " << i << ": Polling carpark
components and reacting to state..." << endl << endl;
        NCP->update();
        cout << endl;
    }
    cout << "----- END -----"
    << "\n";
    delete NCP;
    auto end = std::chrono::system_clock::now();//get the current time
    std::chrono::duration<double> elapsed_seconds = end-start;//calculate
the elapsed time
    std::cout << "[Info] Car Park System - Elapsed Time: " <<
elapsed_seconds.count() << "s\n";//print the elapsed time
    return 0;
};
```

7.2.14. *Sensor.cpp*

```
#include "Sensor.h"
#include "InputHelper.h"
using namespace std;

/**
 * cSensor Constructor
 * @param id - used to define the the sensorID for identification purposes
 * or later/future use for 'real' access to a sensor.
 */
cSensor::cSensor(int id)
{
    //set sensorID to the parameter id
    sensorID = id;
    //set sensorState to false as default.
    sensorState = false;
}

/**
 * cSensor getSensorID
 * Method to return the set sensorID.
 * SensorID cannot be reset using a set method, instead a new sensor object
must be created.
 * @return sensorID : int
 */
int cSensor::getSensorID()
{
    return sensorID; //return the sensorID
}

/**
 * cSensor getSensorState
 * Method used to 'read' the current sensor state.
 * If this were a working sensor, this would interact with a real sensor,
likely
 * using specialist libraries and interfaces.
 * However, given the nature of the assignment, it was decided that it
would simply ask the
 * user for input representing the state of the sensor, and return the
value of sensorState
 * @return sensorState : boolean - (true or false)
 */
bool cSensor::getSensorState()
{
    //use the input helper to get input from user to replicate sensor state
    string inputString = "[Input] Input Value (0 or 1) for SensorID=" +
std::to_string(sensorID) + ":";
    sensorState = cInputHelper::getBoolInput(inputString);
    //Simply return sensorState - As stated above, this would normally
contain logic to query a physical sensor.
    return sensorState;
}

bool cSensor::update()
{
    return getSensorState(); //call the getSensorState method and return its
value
}
```

7.2.15. *Sensor.h*

```
#pragma once
#ifndef SENSOR_H
#define SENSOR_H
class cSensor
{
public:
    cSensor(int id);
    bool update();
    int getSensorID();
private:
    int sensorID;
    bool sensorState;
    bool getSensorState();
};
#endif
```

7.2.16. UnitTests.cpp

```
#define CATCH_CONFIG_MAIN
#include "catch.hpp"
#include "CarPark.h"
#include "Validation.h"

/** Roy Osherove in his book 'The Art Of Unit Testing' states that
 * "Properties (getters/setters in Java) are good examples of code that
 * usually doesn't contain any logic, and doesn't require testing.
 * But watch out: once you add any check inside the property, you'll want
 * to make sure that logic is being tested." [1]
 * [1] R. Osherove, The Art of Unit Testing, Second Edition.
 */

/**
 * Please see attached design document for full testing schedule referring
 * to test numbers in comments.
 * The below tests are unit tests. Functional user testing is also to be
 * conducted.
 * Not every single method has been tested, due to constraints - for
 * example private methods,
 * however all non-private and static methods have been tested.
 */

//***** U-A: CarPark Class and Methods
*****

//U-A1: TEST CAR PARK CONSTRUCTORS
TEST_CASE( "U-A1: Test Car Park Capacity is Set to 5 Correctly",
"[5][CarParkInitialiseCapacity]" ) {
    //Create new car park with capacity of 5
    cCarPark *testCarParkCapacity5 = new cCarPark(5, cBarcodeReader(6),
                                                    cSensor(3),
                                                    cNumberplateReader(7),
                                                    cSensor(1),
                                                    cSensor(4),
                                                    cNumberplateReader(8),
                                                    cSensor(2));

    REQUIRE( testCarParkCapacity5->getCapacity() == 5 );
}

//U-A2: Test Constructor Initialises Capacity
TEST_CASE( "U-A2: Test Car Park Capacity is Set to 16 (2)",
"[16][CarParkInitialiseCapacity(2)]" ) {
    //Create a new car park with a capacity of 16 and 2 free spaces
    cCarPark *testCarParkCapacity5FreeSpaces2 = new cCarPark(16,2,

cBarcodeReader(6),
                                                    cSensor(3),

cNumberplateReader(7),
                                                    cSensor(1),
                                                    cSensor(4),

cNumberplateReader(8),
                                                    cSensor(2));

    REQUIRE( testCarParkCapacity5FreeSpaces2->getCapacity() == 16 );
}

//U-A3: CAPACITY AND FREE SPACES
TEST_CASE( "U-A3: Test Car Park Free Spaces is Set to 2",
"[2][CarParkInitialiseFreeSpaces]" ) {
```

```

        //create a new car park with a capacity of 5 and 2 free spaces
        cCarPark *testCarParkCapacity5FreeSpaces2 = new cCarPark(5,2,
cBarcodeReader(6),
                                                                    cSensor(3),

cNumberplateReader(7),
                                                                    cSensor(1),
                                                                    cSensor(4),

cNumberplateReader(8),
                                                                    cSensor(2));

        REQUIRE( testCarParkCapacity5FreeSpaces2->getFreeSpaces() == 2 );
    }
//U-A4: SET FREE SPACES METHOD (SETTER)
TEST_CASE( "U-A4: Test Car Park Set Free Spaces Setter Method (1)",
"[7][CarParkSetFreeSpaces]" ) {
    // Create a new car park with a capacity of 10, and explicitly state
    free spaces to 10
    cCarPark *testCarPark = new cCarPark(10,10, cBarcodeReader(6),
                                                                    cSensor(3),
                                                                    cNumberplateReader(7),
                                                                    cSensor(1),
                                                                    cSensor(4),
                                                                    cNumberplateReader(8),
                                                                    cSensor(2));

    testCarPark->setFreeSpaces(7);
    REQUIRE( testCarPark->getFreeSpaces() == 7 );
    testCarPark->setFreeSpaces(2);
    REQUIRE( testCarPark->getFreeSpaces() == 2 );
    testCarPark->setFreeSpaces(5);
    REQUIRE( testCarPark->getFreeSpaces() == 5 );
}
//U-A5: SET FREE SPACES (above 10 capacity) - should set to max capacity
instead
TEST_CASE( "U-A5: Test Car Park Set Free Spaces Setter Method (2)",
"[10][CarParkSetFreeSpacesMax]" ) {
    cCarPark *testCarPark = new cCarPark(10,10,cBarcodeReader(6),
                                                                    cSensor(3),
                                                                    cNumberplateReader(7),
                                                                    cSensor(1),
                                                                    cSensor(4),
                                                                    cNumberplateReader(8),
                                                                    cSensor(2));

    testCarPark->setFreeSpaces(170);
    REQUIRE( testCarPark->getFreeSpaces() == 10 );
    testCarPark->setFreeSpaces(1705454545);
    REQUIRE( testCarPark->getFreeSpaces() == 10 );
}
//U-A6: Check that the GETTER method is returning expected value (6)
TEST_CASE( "U-A6: Test Car Park GET Free Spaces Getter Method",
"[6][CarParkGetFreeSpaces]" ) {
    cCarPark *testCarPark = new cCarPark(10,6,cBarcodeReader(6),
                                                                    cSensor(3),
                                                                    cNumberplateReader(7),
                                                                    cSensor(1),
                                                                    cSensor(4),
                                                                    cNumberplateReader(8),
                                                                    cSensor(2));

    REQUIRE( testCarPark->getFreeSpaces() == 6 );
}
//U-A7: decrement spaces method test

```

```

TEST_CASE( "U-A7: Test Car Park Decrement Free Spaces Method",
"[9][CarParkDecrementFreeSpaces]" ) {
    cCarPark *testCarPark = new cCarPark(100,44,cBarcodeReader(6),
                                           cSensor(3),
                                           cNumberplateReader(7),
                                           cSensor(1),
                                           cSensor(4),
                                           cNumberplateReader(8),
                                           cSensor(2));

    testCarPark->setFreeSpaces(31);
    testCarPark->decrementSpaces();
    REQUIRE( testCarPark->getFreeSpaces() == 30 );
    testCarPark->setFreeSpaces(0);
    testCarPark->decrementSpaces();
    REQUIRE( testCarPark->getFreeSpaces() == 0 );
}

//U-A8: increment spaces method test
TEST_CASE( "U-A8: Test Car Park Increment Free Spaces Method",
"[10][CarParkIncrementFreeSpaces]" ) {
    cCarPark *testCarPark = new cCarPark(10,10,cBarcodeReader(6),
                                           cSensor(3),
                                           cNumberplateReader(7),
                                           cSensor(1),
                                           cSensor(4),
                                           cNumberplateReader(8),
                                           cSensor(2));

    testCarPark->setFreeSpaces(9);
    testCarPark->incrementSpaces();
    REQUIRE( testCarPark->getFreeSpaces() == 10 );
    testCarPark->setFreeSpaces(3);
    testCarPark->incrementSpaces();
    REQUIRE( testCarPark->getFreeSpaces() == 4 );
}

//U-A9: increment spaces method test on max capacity
TEST_CASE( "U-A9: Test Car Park Increment Free Spaces Method with Max
Capacity", "[10][CarParkIncrementFreeSpacesMax]" ) {
    cCarPark *testCarPark = new cCarPark(10,10,cBarcodeReader(6),
                                           cSensor(3),
                                           cNumberplateReader(7),
                                           cSensor(1),
                                           cSensor(4),
                                           cNumberplateReader(8),
                                           cSensor(2));

    testCarPark->setFreeSpaces(10);
    testCarPark->incrementSpaces();//without logic would raise to 11
    REQUIRE( testCarPark->getFreeSpaces() == 10 );
}

//U-A10: decrement spaces method test on 0 free spaces
TEST_CASE( "U-A10: Test Car Park Decrement Free Spaces Method with Max
Capacity", "[10][CarParkDecrementFreeSpacesMax]" ) {
    cCarPark *testCarPark = new cCarPark(10,10,cBarcodeReader(6),
                                           cSensor(3),
                                           cNumberplateReader(7),
                                           cSensor(1),
                                           cSensor(4),
                                           cNumberplateReader(8),
                                           cSensor(2));

    testCarPark->setFreeSpaces(0);
    testCarPark->decrementSpaces();//without logic would lower to -1
    REQUIRE( testCarPark->getFreeSpaces() == 0 );
}

```

```

//***** U-A: END - CarPark Class and Methods
*****

//***** U-B: Barrier Class and Methods
*****

//U-B1: TEST cBarrier CONSTRUCTOR
TEST_CASE( "U-B1: Test Barrier Constructor Sets up to false",
"[false][BarrierConstructorUpFalse]" ) {
    cBarrier testBarrier; //create new instance of cBarrier class called
testBarrier
    //get the status of the barrier using update method, ensure it has been
set to false by the constructor
    REQUIRE( testBarrier.update() == false );
}
//U-B2: TEST raise barrier method
TEST_CASE( "U-B2: Test Barrier Raise sets up to true",
"[true][BarrierRaise]" ) {
    cBarrier testBarrier; //create new instance of cBarrier class called
testBarrier
    testBarrier.raise(); //call raise method to set up to true
    //get the status of the barrier using update method, ensure it is true
    REQUIRE( testBarrier.update() == true );
}
//U-B3: TEST lower barrier method
TEST_CASE( "U-B3: Test Barrier Lower sets up to false",
"[true][BarrierLower]" ) {
    cBarrier testBarrier; //create new instance of cBarrier class called
testBarrier
    testBarrier.lower(); //call raise method to set up to false
    //get the status of the barrier using update method, ensure it is false
    REQUIRE( testBarrier.update() == false );
}

//***** U-B: END - Barrier Class and Methods
*****

//***** U-C: FullSign Class and Methods
*****

//U-C1: TEST cFullSign CONSTRUCTOR
TEST_CASE( "U-C1: Test FullSign Constructor Sets lightOn to false",
"[false][FullSignConstructorLightOnFalse]" ) {
    cFullSign testFullSign; //create new instance of cBarrier class called
testBarrier
    //get the status of the fullSign light using update method, ensure it
has been set to false by the constructor
    REQUIRE( testFullSign.update() == false );
}
//U-C2: TEST switchOn fullSign method
TEST_CASE( "U-C2: Test FullSign SwitchOn sets lightOn to true",
"[true][FullSignSwitchOn]" ) {
    cFullSign testFullSign; //create new instance of cBarrier class called
testBarrier
    testFullSign.switchOn(); //call switchOn method to set lightOn to true
    //get the status of the fullSign using update method, ensure lightOn is
true
    REQUIRE( testFullSign.update() == true );
}
//U-C3: TEST switchOff fullSign method
TEST_CASE( "U-C3: Test FullSign SwitchOff sets lightOn to false",

```

```

"[false][FullSignSwitchOff]" ) {
    cFullSign testFullSign; //create new instance of cBarrier class called
testBarrier
    testFullSign.switchOff();//call switchOff method to set lightOn to
false
    //get the status of the fullSign using update method, ensure lightOn is
false
    REQUIRE( testFullSign.update() == false );
}

//***** U-C: END - FullSign Class and Methods
*****

//***** U-D: IDReader Class and Methods
*****

//U-D1: TEST cIDReader CONSTRUCTOR
TEST_CASE( "U-D1: Test IDReader Constructor Sets IDReader readerID to 16",
"[16][IDReaderConstructorReaderID16]" ) {
    cIDReader testIDReader(16); //create new instance of cIDReader class
called testIDReader with id of 1
    //testIDReader constructor called above with int 16 as parameter
(should set readerID to 16)
    //call getReaderID method to ensure that this is the case
    REQUIRE( testIDReader.getReaderID() == 16 );
}
//U-D2: TEST cIDReader SetReaderType
TEST_CASE( "U-D2: Test IDReader setReaderType method correctly sets reader
type", "[12][IDReaderSetReaderType]" ) {
    cIDReader testIDReader(16); //create new instance of cIDReader class
called testIDReader with id of 1
    //call setReaderType method to ensure this can be set manually
    testIDReader.setReaderType(12);
    REQUIRE( testIDReader.getReaderType() == 12 );
}

//~~~~~ cBarcodeReader Class and Methods (Inheritance from cIDReader)
~~~~~
//U-D3: TEST cBarcodeReader CONSTRUCTOR
TEST_CASE( "U-D3: Test BarcodeReader Constructor Sets BarcodeReader
readerID to 22", "[22][BarcodeReaderConstructorReaderID22]" ) {
    cBarcodeReader testBarcodeReader(22); //create new instance of
cBarcodeReader class called testBarcodeReader with id of 22
    //get the status of the fullSign light using update method, ensure it
has been set to false by the constructor
    REQUIRE( testBarcodeReader.getReaderID() == 22 );
}
//U-D4: TEST cBarcodeReader CONSTRUCTOR sets ReaderType to 1
TEST_CASE( "U-D4: Test BarcodeReader Constructor Sets ReaderType
correctly", "[1][BarcodeReaderConstructorReaderType1]" ) {
    cBarcodeReader testBarcodeReader(22); //create new instance of
cBarcodeReader class called testBarcodeReader with id of 22
    //constructor for NumberPlateReader should set readerType to 1.
    REQUIRE( testBarcodeReader.getReaderType() == 1 );
}
//~~~~~ END cBarcodeReader Class and Methods (Inheritance from cIDReader)
~~~~~

//~~~~~ cNumberplateReader Class and Methods (Inheritance from cIDReader)
~~~~~
//U-D5: TEST cNumberplateReader CONSTRUCTOR

```

```

TEST_CASE( "U-D5: Test NumberplateReader Constructor Sets NumberplateReader
readerID correctly", "[7][NumberplateReaderConstructorReaderID7]" ) {
    cNumberplateReader testNumberplateReader(7); //create new instance of
cBarcodeReader class called testBarcodeReader with id of 22
    //get the status of the fullSign light using update method, ensure it
has been set to false by the constructor
    REQUIRE( testNumberplateReader.getReaderID() == 7 );
}
//U-D6: TEST cNumberplateReader CONSTRUCTOR sets ReaderType to 2
TEST_CASE( "U-D6: Test NumberplateReader Constructor Sets ReaderType
correctly", "[2][NumberplateReaderConstructorReaderType2]" ) {
    cNumberplateReader testNumberplateReader(7); //create new instance of
cBarcodeReader class called testBarcodeReader with id of 22
    //constructor for NumberPlateReader should set readerType to 2.
    REQUIRE( testNumberplateReader.getReaderType() == 2 );
}
//~~~~~ END cBarcodeReader Class and Methods (Inheritance from cIDReader)
~~~~~

//***** U-D: END - IDReader Class and Methods
*****

//***** U-E: LinkedList Class and Methods
*****

//U-E1: TEST LinkedList CONSTRUCTOR sets length to 0
TEST_CASE( "U-E1: Test LinkedList Constructor sets default values",
"[multiple][LinkedListConstructor]" ) {
    cLinkedList *testLLList = new cLinkedList();
    REQUIRE( testLLList->size() == 0 );
}
//U-E2: TEST LinkedList (int) Adds item to the list
TEST_CASE( "U-E2: Test LinkedList (int) add function adds item to the
list", "[NOTNULLPTR][LinkedListAddItemInt]" ) {
    cLinkedList *testLLList = new cLinkedList();
    testLLList->add(9);
    //search method either returns pointer if found, or returns null if not
found
    REQUIRE( testLLList->search(9) != nullptr );
    //test multiple times with various int data items
    testLLList->add(2);
    REQUIRE( testLLList->search(2) != nullptr );
    testLLList->add(17645585);
    REQUIRE( testLLList->search(17645585) != nullptr );
    testLLList->add(0);
    REQUIRE( testLLList->search(0) != nullptr );
    testLLList->add(-123456);
    REQUIRE( testLLList->search(-123456) != nullptr );
}
//U-E3: TEST LinkedList (str) Adds item to the list
TEST_CASE( "U-E3: Test LinkedList (str) add function adds item to the
list", "[NOTNULL][LinkedListAddItemStr]" ) {
    cLinkedList *testStrLLList = new cLinkedList();
    std::string item1 = "abc";
    std::string item2 = "CE13NHM";
    std::string item3 = "A1";
    testStrLLList->add(item1);
    //search method either returns pointer if found, or returns nullptr if
not found
    REQUIRE( testStrLLList->search(item1) != nullptr );
    //test multiple times with various int data items

```

```

    testStrLList->add(item2);
    REQUIRE( testStrLList->search(item2) != nullptr );
    testStrLList->add(item3);
    REQUIRE( testStrLList->search(item3) != nullptr );
    delete testStrLList;
}
//U-E4: TEST LinkedList (int) removes items from the list
TEST_CASE( "U-E4: Test LinkedList (int) remove function removes item from
the list", "[NULLPTR][LinkedListIntRemoveItem]" ) {
    cLinkedList *testLList = new cLinkedList();
    testLList->add(44);
    testLList->remove(44);
    //search method either returns pointer if found, or returns null if not
    found
    REQUIRE( testLList->search(44) == nullptr );
    //test multiple times with various int data items
    testLList->add(89);
    testLList->remove(89);
    REQUIRE( testLList->search(89) == nullptr );
    testLList->add(5656734);
    testLList->remove(5656734);
    REQUIRE( testLList->search(5656734) == nullptr );
    testLList->add(-5545676);
    testLList->remove(-5545676);
    REQUIRE( testLList->search(-5545676) == nullptr );
    delete testLList;
}
//U-E5: TEST LinkedList (str) removes items from the list
TEST_CASE( "U-E5: Test LinkedList (str) remove function removes item from
the list", "[NULLPTR][LinkedListStrRemoveItem]" ) {
    cLinkedList *testStrLList2 = new cLinkedList();
    testStrLList2->add("R45NMH");
    testStrLList2->remove("R45NMH");
    //search method either returns pointer if found, or returns null if not
    found
    REQUIRE( testStrLList2->search("R45NMH") == nullptr );
    //test multiple times with various int data items
    testStrLList2->add("CE19 KLG");
    testStrLList2->remove("CE19 KLG");
    REQUIRE( testStrLList2->search("CE19 KLG") == nullptr );
    testStrLList2->add("A1");
    testStrLList2->remove("A1");
    REQUIRE( testStrLList2->search("A1") == nullptr );
    delete testStrLList2;
}
//U-E6: Test that the size method returns the correct list size (6)
TEST_CASE( "U-E6: Test LinkedList size returns the correct list size",
"[6][LinkedListSize]" ) {
    cLinkedList *testLList3 = new cLinkedList();
    //add 6 items of data to an empty list
    testLList3->add(1);
    testLList3->add(2);
    testLList3->add(3);
    testLList3->add(4);
    testLList3->add(5);
    testLList3->add(6);
    REQUIRE( testLList3->size() == 6 );
    delete testLList3;
}
//U-E7: Test remove item that does not exist from linked list
TEST_CASE( "U-E7: Test remove item that does not exist from linked list",

```

```

"[FALSE][RemoveItemNotExitLinkedList]" ) {
    cLinkedList *testLList = new cLinkedList();
    REQUIRE( testLList->remove(7) == false );
    delete testLList;
}

//***** U-E: END - LinkedList Class and Methods
*****

//***** U-F: Sensor Class and Methods
*****

//U-F1: Test Sensor Constructor sets the ID of the sensor correctly
TEST_CASE( "U-F1: Test Sensor Constructor sets the ID of sensor",
"[3][SensorConstructor]" ) {
    cSensor testSensor(3);
    REQUIRE( testSensor.getSensorID() == 3 );
}
//U-F2: Test getSensorID method
TEST_CASE( "U-F2: Test getSensorID method returns correct sensorID",
"[12][SensorGetSensorID]" ) {
    cSensor testSensor12(12);
    REQUIRE( testSensor12.getSensorID() == 12 );
}
//Other sensor functions are private and will be included in the functional
user testing.

//***** U-F: END - Sensor Class and Methods
*****

//***** U-G: ValidateID Class and Methods
*****

//U-G1: Test ID Validator correctly marks IDs as valid (static method)
TEST_CASE( "U-G1: Test ID Validator correctly marks Valid IDs as valid",
"[true][IDValidatorValid]" ) {
    cValidateID idValidator;
    REQUIRE( idValidator.validateID(1000) == true );
    REQUIRE( idValidator.validateID(2000) == true );
    REQUIRE( idValidator.validateID(2222) == true );
}
//U-G2: Test ID Validator correctly marks IDs as invalid (static method)
TEST_CASE( "U-G2: Test ID Validator correctly marks invalid IDs as
invalid", "[false][IDValidatorInvalid]" ) {
    cValidateID idValidator;
    REQUIRE( idValidator.validateID(1) == false );
    REQUIRE( idValidator.validateID(10000) == false );
    REQUIRE( idValidator.validateID(999) == false );
    REQUIRE( idValidator.validateID(-2222) == false );
    REQUIRE( idValidator.validateID(3333) == false );
}

//***** U-G: END - ValidateID Class and Methods
*****

//***** U-H: Validation Class and Methods
*****

//U-H1: Test Validator validateInt correctly identifies valid input (static
method)
TEST_CASE( "U-H1: Test Validator validateInt identifies valid input",

```

```

"[true][ValidationValidateIntValid]" ) {
    //Test input 1 is a valid int between min=0 and max=100
    REQUIRE( cValidation::validateInt(1,0,100) == true );//value=1, min=0,
max=100
    //Test input 1 is a valid int between min=0 and max=100
    REQUIRE( cValidation::validateInt(99,0,100) == true );//value=99,
min=0, max=100
    //Test input 0 is a valid int between min=0 and max=100 (valid
boundary)
    REQUIRE( cValidation::validateInt(0,0,100) == true );//value=0, min=0,
max=100
    //Test input 0 is a valid int between min=0 and max=100 (valid
boundary)
    REQUIRE( cValidation::validateInt(100,0,100) == true );//value=100,
min=0, max=100
    //Test input 0 is a valid int between min=0 and max=100 (valid mid)
    REQUIRE( cValidation::validateInt(51,0,100) == true );//value=51,
min=0, max=100
}
//U-H2: Test Validator validateInt correctly identifies invalid input
(static method)
TEST_CASE( "U-H2: Test Validator validateInt identifies invalid input",
"[false][ValidationValidateIntInvalid]" ) {
    //Test input -1 with params between min=0 and max=100 (invalid
boundary)
    REQUIRE( cValidation::validateInt(-1, 0, 100) == false );//value=-1,
min=0, max=100
    //Test input 101 with params between min=0 and max=100 (invalid
boundary)
    REQUIRE( cValidation::validateInt(101, 0, 100) == false );//value=101,
min=0, max=100
    //Test input 101 with params between min=0 and max=100 (invalid
extreme)
    REQUIRE( cValidation::validateInt(1005345454, 0, 100) ==
false );//value=1005345454, min=0, max=100
    //Test input 101 with params between min=0 and max=100 (invalid
extreme)
    REQUIRE( cValidation::validateInt(-105345454, 0, 100) ==
false );//value=-105345454, min=0, max=100
}
//U-H3: Test Validator validateBool correctly identifies valid input
(static method)
TEST_CASE( "U-H3: Test Validator validateBool valid input",
"[true][IDValidatorValidateBoolValid]" ) {
    REQUIRE( cValidation::validateBool(1) == true );
    REQUIRE( cValidation::validateBool(0) == true );
}
//U-H4: Test Validator validateBool correctly identifies invalid input
(static method)
TEST_CASE( "U-H4: Test Validator validateBool invalid input",
"[false][IDValidatorValidateBoolInvalid]" ) {
    REQUIRE( cValidation::validateBool(-1) == false );
    REQUIRE( cValidation::validateBool(2) == false );
    REQUIRE( cValidation::validateBool(35) == false );
    REQUIRE( cValidation::validateBool(34545455) == false );
}
//U-H5: Test Validator validateNumberplate correctly identifies valid input
TEST_CASE( "U-H5: Test Validator validateNumberplate valid input",
"[true][IDValidatorValidateNumberplateValid]" ) {
    REQUIRE( cValidation::validateNumberplate("CE16 NMH") == true );
    REQUIRE( cValidation::validateNumberplate("N6ABC") == true );
}

```

```

    REQUIRE(cValidation::validateNumberplate("N6 ABC") == true);
    REQUIRE(cValidation::validateNumberplate("A1") == true);
    REQUIRE(cValidation::validateNumberplate("R44 PXC") == true);
}
//U-H6: Test Validator validateNumberplate correctly identifies invalid
input
TEST_CASE( "U-H6: Test Validator validateNumberplate invalid input",
"[false][IDValidatorValidateNumberplateInvalid]" ) {
    REQUIRE(cValidation::validateNumberplate("1234abdf645457") == false);
    REQUIRE(cValidation::validateNumberplate("1233435454554535") == false);
    REQUIRE(cValidation::validateNumberplate("TESTSTRING") == false);
}

//***** U-H: END - Validation Class and Methods
*****

```

7.2.17. ValidateID.cpp

```
#include "ValidateID.h"
/**
 * Note that only the default constructor is provided.
 * In future, this class could be used to build different types of
validators,
 * or a parameter such as validator type/id could be used to differentiate.
 * Alternatively, inheritance could be used to build different validator
types, based on
 * requirements.
 */
cValidateID::cValidateID()
= default;
/**
 * cValidateID validateID
 * ValidateID function is used to validate an ID (id parameter).
 * This would normally be set up with a database in the background and
another layer
 * to query the database.
 * It was decided that for the given solution only a simple function logic
would be used to
 * validate an ID, due to it not being considered on the assignment brief.
 * Therefore, this function could contain any simple logic, however it was
decided to create a
 * check that ID is a multiple of 2, and between 1000 and 9999.
 * @param id (the id to be checked)
 * @return boolean, true is valid id, false is invalid id
 */
bool cValidateID::validateID(int id)
{
    if ((id % 2 == 0) && (id <= 9999) && (id >= 1000))
    { //for the given solution any id that is a multiple of 2, and between
1000 and 9999 is valid.
        return true;
    }
    else
    {
        return false;
    }
}
```

7.2.18. *ValidateID.h*

```
#pragma once
#ifndef VALIDATEID_H
#define VALIDATEID_H

class cValidateID
{
public:
    cValidateID ();
    bool validateID(int id);
};
#endif
```

7.2.19. Validation.cpp

```
#include <string>
#include <regex>
#include "Validation.h"

using namespace std;

/**
 * validateInt method
 * A very simple validation method to validate an integer value.
 * In theory, this should include more complex type checking, however
 * it was kept simple for the scope of the application.
 * @param v - value to be tested
 * @param min - minimum value to be compared to
 * @param max - maximum value to be compared to
 * @return boolean true (valid) or false (invalid)
 */
bool cValidation::validateInt(int v, int min, int max) {
    if(v < min || v > max){
        return false;
    }
    return true;
}

/**
 * validateBool method
 * A very simple validation method to validate a boolean value.
 * In theory, this should include more complex type checking, however
 * it was kept simple for the scope of the application.
 * @param v value to be tested
 * @return boolean true (valid) or false (invalid)
 */
bool cValidation::validateBool(int v){
    if(v == 1 || v == 0){
        return true;
    }
    return false;
}

/**
 * validateNumberplate method
 * A very simple validation method to validate a numberplate input value.
 * In theory, this should include type checking, with complex regex however
 * it was kept simple for the scope of the application.
 * @param v value to be tested
 * @return boolean true (valid) or false (invalid)
 */
bool cValidation::validateNumberplate(std::string v) {
    //Very simple numberplate regex allowing a-z, A-Z and 0-9, length 1 to
    8
    regex str_expr ("^([a-zA-Z0-9\\s]{1,8})$");
    //compare the string and the regex to see if there is a match
    if (regex_match(v,str_expr)){
        //if valid string, return true
        return true;
    }
    //else return false, does not match regex
    return false;
}
```

7.2.20. *Validation.h*

```
#ifndef CARPARK_SYSTEM_VALIDATION_H
#define CARPARK_SYSTEM_VALIDATION_H

class cValidation
{
public:
    static bool validateInt(int v, int min, int max);
    static bool validateBool(int v);
    static bool validateNumberplate(std::string v);
};
#endif //CARPARK_SYSTEM_VALIDATION_H
```

7.2.21. CMakeLists.txt

```
cmake_minimum_required(VERSION 3.17)
project(carpark_system)

set(CMAKE_CXX_STANDARD 14)

add_executable(carpark_latest
    main.cpp
    Barrier.cpp
    Barrier.h
    CarPark.cpp
    CarPark.h
    FullSign.cpp
    FullSign.h
    IDReader.cpp
    IDReader.h
    LinkedList.cpp
    LinkedList.h
    Sensor.cpp
    Sensor.h
    ValidateID.cpp
    ValidateID.h
    Validation.cpp
    Validation.h
    InputHelper.cpp
    InputHelper.h)

add_executable(carpark_unittest
    UnitTests.cpp
    Barrier.cpp
    Barrier.h
    CarPark.cpp
    CarPark.h
    FullSign.cpp
    FullSign.h
    IDReader.cpp
    IDReader.h
    LinkedList.cpp
    LinkedList.h
    Sensor.cpp
    Sensor.h
    ValidateID.cpp
    ValidateID.h
    Validation.cpp
    Validation.h
    InputHelper.cpp
    InputHelper.h
    catch.hpp)
```

7.2.22. catch.hpp

Single header catch.hpp file used within the solution for unit testing can be download from the Catch github repository:
<https://github.com/catchorg/Catch2/releases/download/v2.13.4/catch.hpp>