

SCHOOL OF COMPUTER SCIENCES

ACADEMIC SESSION 2016 / 2017, SEMESTER 2

CPT 113 – PROGRAMMING METHODOLOGY & DATA STRUCTURES

ASSIGNMENT 2

DUE DATE : 10th APRIL 2017

NAME : CHONG KAR HIE

MATRIC NUMBER : 132893

GROUP : D

LECTURER’S NAME : MOHD AZAM OSMAN

Table of content

|  |  |
| --- | --- |
| Content | Page number |
| Introduction | 1 |
| Problem analysis   * Inputs * Outputs * Processes/formulas * Test data | 2  2  3  4-7 |
| System design   * Interface Design * Module/Function/Methods diagram * Data Structure Design * UML diagrams | 8-9  9  9-11  12 |
| Description of selected functions | 13 |
| C++ code (source code) | 14-29 |
| Print screen of sample input/output | 30-35 |

Introduction

The purpose of this assignment is to illustrate that the way that information of Honda car sales are being collected by using linked list. Nowadays, people are getting used to the life that cars are already become as an essential thing in their life. There are so many brands of cars that customers can choose their own desired brand and model to buy, and utilize the car. Car companies are having a lot of demands and they have produced cars in a large amount, hence they need to have a system to store the car sales information for the ease of searching car sales information, managing car sales information, and also calculate the total sales of the cars sold based on the on the road price of each car.

The company in this assignment, which is Honda, requires us to create a program which uses linked list to store car sales information. The program can arrange, search, delete car sales information, and it can calculate the total sales of the cars sold based on the on the road price of each car. So, it can bring convenience to the Honda company in order to manage a large amount of car sales information.

Problem analysis

**Inputs:-**

User’s input:-

1. New car’s engine number
2. New car’s model
3. New car’s color
4. New car’s displacement (cc)
5. New car’s selling price
6. New car’s retail price
7. New car’s on the road price with insurance
8. Car engine number to delete
9. Car engine number to display

**Outputs:-**

1. Car engine number
2. Car’s model
3. Car’s color
4. Car’s displacement (cc)
5. Car’s selling price
6. Car’s retail price
7. Car’s on the road price
8. Total number of the cars sold

**Processes:-**

First, the program will read all the existing car sales information from a text file named “CarDetails.txt” in order to store all of the existing car sales information into a linked list.

The program has 4 selections in its main menu which is:-

* Add car sales information
* Delete car sales information
* Display car sales information
* Exit program

For the first selection, user can input car sales information for a new car in order to save it inside the linked list and manipulate the data.

For the second selection, user can delete either a particular car sales information or all the car sales information. If the user choose to delete a particular car sales information, user needs to enter the car engine number of the respective car sales information in order to delete the car sales information. If car engine number is not found in the linked list, a message will prompt out to the user which car engine number is not found, hence no car sales information will be deleted. If the user choose to delete all car sales information, the whole linked list will be deleted there will be no car sales information after the deletion.

For the third selection, the program can display a particular car sales information, car sales information from highest on the road price to lowest on the road price, and all car sales information. If user choose to display a particular car sales information, user needs to enter the car engine number of the respective car sales information in order to show the details of the car sales information such as car engine number, model, color, displacement, selling price, retail price without insurance, and on the road price with insurance. If the user choose to display car sales information from highest on the road price to lowest on the road price, the program will show all the car sales information from highest to lowest on the road price. If the user choose to display all car sales information, the program will show all the car sales information. Both of these options will show the the total number of the cars sold which is the number of entries of the cars.

For the fourth option, all of the car sales information in the linked list will be saved into the same text file named “CarDetails.txt” and the program will be terminated.

**Test data:-**

173269

Honda Odyssey 2.4L EXV

Crystal Black Pearl

2356

247107

248288

255500

533332

Honda Accord 2.4 VTi-L

White Orchid Pearl

2356

166681

167820

172800

123456

Honda Accord 2.0 VTi-L

Lunar Silver Metallic

1997

148570

149352

153800

737373

Honda Accord 2.0 VTi

Modern Steel Metallic

1997

139818

140600

144800

255136

Honda Civic 1.5 TC-P

Dark Ruby Red Pearl

1498

131540

131883

135800

265493

Honda HR-V 1.8L V

Taffeta White

1799

117923

118606

122200

123333

Honda Civic 1.8 S

Crystal Black Pearl

1799

109744

110427

113800

523164

Honda HR-V 1.8L S

Ruse Black Metallic

1799

98447.1

99129.7

102200

421362

Honda City 1.5L V

Modern Steel Metallic

1497

88897.9

89240.9

92000

294632

Honda Jazz 1.5L V

Tinted Silver Metallic

1497

86380.6

86723.6

89400

483847

Honda City 1.5L E

Taffeta White

1497

81690.8

82033.8

84600

192365

Honda Jazz 1.5L S

Carnival Red

1497

71194

71537

73800

System design

**Interface design:-**

Pseudocode:

BEGIN

1. Initialize proceed = ‘Y’.
2. Declare unorderedLinkedList<carInfo> car1, orderedLinkedList<carInfo> car2, carInfo carinfo.
3. Execute car1.readDataUnordered(), car2.readDataOrdered.
4. Execute welcome().
5. Execute menu().
6. Execute validation(0, 3), then assign the value returned into choice.
7. switch(choice)
   1. if (choice == 0),
      1. Execute car2.writeData().
      2. Terminate the program.
   2. if (choice == 1),
      1. Get car engine number.
      2. Get model of the car.
      3. Get color of the car.
      4. Get displacement of the car.
      5. Get selling price of the car.
      6. Get retail price of the car.
      7. Get on the road price of the car.
      8. Execute car2.addordered(carinfo).
      9. Prompt user to select type of insertion for unordered linked list.
         1. if (selection == 1),
            1. Execute car1.insertfirst(carinfo).
         2. if (selection == 2),
            1. Execute car1.insertlast(carinfo).
   3. if (choice == 2),
      1. Execute menu2().
      2. Execute validation(0, 2), then assign the value returned into choice2.
      3. switch(choice2)
         1. if (choice2 == 0),
            1. Execute menu().
         2. if (choice2 == 1),
            1. Prompt user to input car engine number to delete.
            2. Get car engine number.
            3. Execute car1.deleteparticular(searchengineno).
            4. Execute car2.deleteparticular(searchengineno).
         3. if (choice2 == 2),
            1. Execute car1.deleteall().
            2. Execute car2.deleteall().
   4. if (choice == 3),
      1. Execute menu3().
      2. Execute validation(0, 3), then assign the value returned into choice3.
      3. switch(choice3)
         1. if (choice3 == 0),
            1. Execute menu().
         2. if (choice3 == 1),
            1. Prompt user to input car engine number to display.
            2. Get car engine number.
            3. Execute car2.displayparticular(searchengineno).
         3. if (choice3 == 2),
            1. Execute car2.displayotrbased().
         4. if (choice3 == 3),
            1. Execute car2.displayall().
8. Prompt user to input proceed.
   1. While proceed == ‘Y’ or proceed == ‘y’, loop for the whole program will be executed.

END

**Module/Function/Methods diagram:-**

1. menu() – to show the main menu.
2. menu2() – to show the menu of deletion.
3. menu3() – to show the menu of displaying car sales information.
4. welcome() – to display the welcome message to users.

**Data Structure Design:-**

Class linkedListType

Description: to store the information needed to create nodes for linked list, and the functions to delete a particular car sales information or all of the car sales information.

Variables:-

1. int count – to store the number of entries which is number of car sales information
2. nodeType<type> \*first – pointer to point to the first node of the linked list
3. nodeType<type> \*last – pointer to point to the last node of the linked list

Functions:-

1. void deleteparticular(string) – to delete a particular car sales information
2. void deleteall() – to delete all of the car sales information in the linked list

Class orderedLinkedList

Description: to read all of the car sales information from text file into ordered linked list, display car sales information from ordered linked list, add new car sales information to ordered linked list, and write all of the car sales information from ordered linked list into text file.

Variables – none.

Functions:-

1. void addordered(type) – to add new car sales information into ordered linked list
2. void displayall() – to display all of the car sales information to users
3. void displayotrbased() – to display all of the car sales information from highest on the road price to lowest on the road price to users
4. void displayparticular(string) – to display a particular car sales information to user
5. void readDataOrdered() – to read car sales information from text file into temporary variables which are set in the function
6. void addDataOrdered(string, string, string, int, double, double, double) – to store car sales information from temporary variables mentioned above into ordered linked list
7. void writeData() – to store all of the car sales information from ordered linked list into the text file before the program exits.

Class unorderedLinkedList

Description: to read all of the car sales information from text file into unordered linked list, and add new car sales information to unordered linked list based on the criteria which is chosen by the user.

Variables – none.

Functions:-

1. void insertfirst(type) – to add new car sales information in front of the unordered linked list as the first node of the unordered linked list
2. void insertlast(type) – to add new car sales information at the end of the unordered linked list as the last node of the unordered linked list
3. void readDataUnordered() – to read car sales information from text file into temporary variables which are set in the function
4. void addDataUnordered(string, string, string, int, double, double, double) – to store car sales information from temporary variables mentioned above into unordered linked list

struct nodeType

Description: to store a structure which contains variables for car sales information and the pointer for the struct nodeType.

Variables:-

1. type carInfo – a structure to store car sales information based on the variables
2. nodeType<type> \*link – a pointer for the struct nodeType in order to point to the next node

struct carInfo

Description: to store car sales information into the variables

Variables:-

1. string engineno – to store the car engine no
2. string model – to store the model of the car
3. string color – to store the color of the car
4. int displacement – to store the displacement (cc) of the car
5. double sellingprice – to store the selling price of the car (inclusive of 6% GST)
6. double retailprice – to store the retail price without insurance of the car (inclusive of 6% GST)
7. double ontheroad – to store the on the road price with insurance of the car (inclusive of 6% GST)

**UML diagram:-**

|  |
| --- |
| linkedListType |
| #count: int  <<ptr>>#first: nodeType<type>  <<ptr>>#last: nodeType<type> |
| +deleteparticular(string): void  +deleteall(): void |

|  |
| --- |
| orderedLinkedList |
| +addordered(type): void  +displayall(): void  +displayotrbased(): void  +displayparticular(string): void  +calculateotr(): double  +readDataOrdered(): void  +addDataOrdered(string, string, string, int, double, double, double): void  +writeData(): void |

|  |
| --- |
| unorderedLinkedList |
| +insertfirst(type): void  +insertlast(type): void  +readDataUnordered(): void  +addDataUnordered(string, string, string, int, double, double, double): void |

|  |
| --- |
| <<struct>> nodeType |
| +carInfo: type  <<ptr>>+link: nodeType<type> |

|  |
| --- |
| <<struct>> carInfo |
| +engineno: string  +model: string  +color: string  +displacement: int  +sellingprice: double  +retailprice: double  +ontheroad: double |

Description of selected functions

int validation(int min, int max)

* this is a integer return type function which has two parameters, which are minimum and maximum value of the question.
* this function will take input and check whether the input is in between of the range between minimum and maximum value before passing back to the variables assigned.

string validation2(int)

* this is a string return type function which has only one parameter, which is the mode of the validation
* the function will accept the input from the user first
* if the mode is 1, validation for the car engine number will be executed.
* If the mode is 2, validation for the color of the car will be executed.
* this function will then return the input to the variables assigned.

double validation3()

* this is a double return type function which does not have any parameter.
* this function will take input from the users first.
* then, the validation for selling price, retail price without insurance and on the road price with insurance of the car will be executed.
* this function will then return the input to the variables assigned.

C++ code (source code)

**main.cpp**

1. #include <iostream>
2. #include <iomanip>
3. #include <cstdlib>
4. #include <fstream>
5. #include <windows.h>
6. #include "orderedLinkedList.h"
7. #include "unorderedLinkedList.h"
8. #include <cassert>
9. #include <fstream>
11. **using** **namespace** std;
13. **void** welcome();
14. **void** menu();
15. **void** menu2();
16. **void** menu3();
17. **int** validation(**int**, **int**);
18. string validation2(**int**);
19. **double** validation3();
20. **int** main()
21. {
22. **int** choice, choice2, choice3, selection; //Variables declaration
23. string searchengineno;
24. string addengineno;
25. **char** proceed = 'Y';
26. unorderedLinkedList<carInfo> car1; //Declaration of objects
27. orderedLinkedList<carInfo> car2;
28. carInfo carinfo; //Declaration of struct carInfo
29. cout << fixed << showpoint << setprecision(2); //Set into 2 decimal point
31. car1.readDataUnordered(); //Function to read text file into unordered linked list
32. car2.readDataOrdered(); //Function to read text file into ordered linked list
33. welcome();
34. **while**(proceed == 'Y' || proceed == 'y') //While loop for looping the whole program
35. {
36. system("cls");
37. menu(); //Main menu
38. choice = validation(0, 3);
39. **switch**(choice)
40. {
41. **case** 0: car2.writeData(); //Function to store ordered linked list into text file
42. **return** 0;
43. **case** 1: system("cls");
44. cout << endl << endl << endl;
45. cout << "\t\t\tPlease enter the car engine number of the car: ";
46. addengineno = validation2(1); //Validation function for car engine number
47. carinfo.engineno = car2.checkengineno(addengineno); //Function to check the duplication of car engine number in linked list
49. cout << "\t\t\tPlease enter the model of the car: ";
50. getline(cin, carinfo.model);
52. cout << "\t\t\tPlease enter the color of the car: ";
53. carinfo.color = validation2(2); //Validation function for color of the cara
55. cout << "\t\t\tPlease enter the displacement (cc) of the car: ";
56. carinfo.displacement = validation(1000, 9999); //Validation function for displacement (cc) of the car
58. cout << "\t\t\tPlease enter the selling price of the car (inclusive of 6% GST): RM";
59. carinfo.sellingprice = validation3(); //Validation function for selling price of the car
61. cout << "\t\t\tPlease enter the retail price without insurance (inclusive of 6% GST): RM";
62. carinfo.retailprice = validation3(); //Validation function for retail price of the car
64. cout << "\t\t\tPlease enter the on the road price with insurance (inclusive of 6% GST): RM";
65. carinfo.ontheroad = validation3(); //Validation function for on the road price of the car
67. cout << "\t\t\tFor unordered linked list, do you want to..." << endl; //Ask user to input first or input last in unordered linked list
68. cout << "\t\t\t1. - Insert first" << endl;
69. cout << "\t\t\t2. - Insert last" << endl << endl;
70. cout << "\t\t\tYour selection: ";
71. selection = validation(1, 2);
72. car2.addordered(carinfo); //Insert information as a node in ordered linked list
73. **if**(selection == 1)
74. car1.insertfirst(carinfo); //Insert first in unordered linked list
75. **else** **if**(selection == 2)
76. car1.insertlast(carinfo); //Insert last in unordered linked list
77. **continue**;
78. **case** 2: menu2(); //Menu for deletion of information
79. choice2 = validation(0, 2);
80. **switch**(choice2)
81. {
82. **case** 0: menu(); //Main menu
83. **continue**;
84. **case** 1: system("cls");
85. cout << endl << endl << endl;
86. cout << "\t\t\tPlease enter the car engine number that you wished to delete: ";
87. searchengineno = validation2(1);
88. car1.deleteparticular(searchengineno); //Function to delete one node in unordered linked list
89. car2.deleteparticular(searchengineno); //Function to delete one node in ordered linked list
90. system("pause");
91. **continue**;
92. **case** 2: car1.deleteall(); //Function to delete all nodes in unordered linked list
93. car2.deleteall(); //Function to delete all nodes in ordered linked list
94. **continue**;
95. }
96. **case** 3: menu3();
97. choice3 = validation(0, 3);
98. **switch**(choice3)
99. {
100. **case** 0: menu();
101. **continue**;
102. **case** 1: system("cls");
103. cout << "Please enter the car engine number that you want to find: ";
104. searchengineno = validation2(1);
105. car2.displayparticular(searchengineno); //Function to display one node in ordered linked list
106. **continue**;
107. **case** 2: car2.displayotrbased(); //Function to display from highest to lowest on the road price
108. **continue**;
109. **case** 3: car2.displayall(); //Function to display all nodes in ordered linked list
110. **continue**;
111. }
112. }
113. cout << "\t\t\tDo you want to continue? Y/y for yes, any key to exit: ";
114. cin >> proceed;
115. }
116. car2.writeData(); //Function to store ordered linked list into text file
117. **return** 0;
118. }
120. **void** welcome() //First welcome screen of the program
121. {
122. cout << endl << endl << endl << "\t\t\t";
123. **char** welcome1[] = {'W', 'e', 'l', 'c', 'o', 'm', 'e', ' ', 't', 'o', '.', '.', '.'};
124. **for**(**int** i = 0; i < 13; i++)
125. {
126. cout << welcome1[i];
127. Sleep(50);
128. }
129. **for**(**int** j = 0; j < 3; j++)
130. {
131. system("cls");
132. cout << endl << endl << endl << "\t\t\t";
133. cout << "Welcome to..." << endl << endl;
134. cout << "\t\t\tHonda Cars Sales Displayer";
135. Sleep(300);
136. }
137. }
139. **void** menu() //Main menu
140. {
141. system("cls");
142. cout << endl << endl;
143. cout << "\t\t\tHonda Cars Sales Displayer" << endl;
144. cout << "\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;
145. cout << "\t\t\t\tMain Menu" << endl;
146. cout << "\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;
147. cout << "\t\t\t1. - Add car sales information" << endl;
148. cout << "\t\t\t2. - Delete car sales information" << endl;
149. cout << "\t\t\t3. - Display car sales information" << endl << endl;
150. cout << "\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;
151. cout << "\t\t\t0. - Exit program" << endl;
152. cout << "\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;
153. cout << "\t\t\tPlease enter your choice: ";
154. }
156. **void** menu2() //Menu of deletion of nodes
157. {
158. system("cls");
159. cout << endl << endl << endl;
160. cout << "\t\t\tPlease choose one of the following: " << endl;
161. cout << "\t\t\tDo you want to..." << endl << endl;
162. cout << "\t\t\t1. - Delete particular car sales information" << endl;
163. cout << "\t\t\t2. - Delete all car sales information" << endl << endl;
164. cout << "\t\t\tOr" << endl << endl;
165. cout << "\t\t\t0. - Go back"<< endl << endl;
166. cout << "\t\t\tPlease select: ";
167. }
169. **void** menu3() //Menu for displaying nodes of ordered linked list
170. {
171. system("cls");
172. cout << endl << endl << endl;
173. cout << "\t\t\tPlease choose one of the following: " << endl;
174. cout << "\t\t\tDisplay..." << endl << endl;
175. cout << "\t\t\t1. - The particular car sales information" << endl;
176. cout << "\t\t\t2. - The details car sales information based on the car on road price" << endl;
177. cout << "\t\t\t3. - All car sales information" << endl << endl;
178. cout << "\t\t\tOr" << endl << endl;
179. cout << "\t\t\t0. - Go back" << endl << endl;
180. cout << "\t\t\tPlease select: ";
181. }
183. **int** validation(**int** min, **int** max) //Validation function for int type data type
184. {
185. **int** input;
186. cin >> input;
187. **while**(!cin || cin.get() != '\n' || input < min || input > max)
188. {
189. cin.clear();
190. cin.ignore(100, '\n');
191. cout << "\t\t\tInvalid input. Please try again: ";
192. cin >> input;
193. }
194. **return** input;
195. }

198. string validation2(**int** mode) //Validation function for string type data type
199. {
200. string input;
201. getline(cin, input);
202. **switch**(mode)
203. {
204. **case** 1: **while**(input.length() != 6 || input.find\_first\_not\_of("1234567890") != string::npos)
205. {
206. cin.clear();
207. cin.ignore(100, '\n');
208. cout << "\t\t\tInvalid input. Please try again: ";
209. getline(cin, input);
210. }
211. **break**;
212. **case** 2: **while**(input.find\_first\_not\_of("abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ") != string::npos)
213. {
214. cin.clear();
215. cin.ignore(100, '\n');
216. cout << "\t\t\tInvalid input. Please try again: ";
217. getline(cin, input);
218. }
219. **break**;
220. }
221. **return** input;
222. }
224. **double** validation3() //Validation function for double type data type
225. {
226. **double** input;
227. cin >> input;
228. **while**(!cin || input <= 0 || cin.get() != '\n')
229. {
230. cin.clear();
231. cin.ignore(100, '\n');
232. cout << "\t\t\tInvalid input. Please try again: RM";
233. cin >> input;
234. }
235. **return** input;
236. }

**linkedListType.h**

1. #ifndef LINKEDLISTTYPE\_H
2. #define LINKEDLISTTYPE\_H
3. #include <iostream>
4. #include <string>
5. #include <fstream>
7. **using** **namespace** std;
9. **template**<**class** type> //Template is being used
10. **struct** nodeType //Declaration of nodeType as a struct
11. {
12. type carInfo;
13. nodeType<type> \*link; //pointer part of the node
14. };
16. **struct** carInfo //Declaration of carInfo as a struct
17. {
18. string engineno;
19. string model;
20. string color;
21. **int** displacement;
22. **double** sellingprice;
23. **double** retailprice;
24. **double** ontheroad;
25. };
27. **template**<**class** type>
28. **class** linkedListType
29. {
30. **public**:
31. linkedListType(); //Default constructor
32. ~linkedListType(); //Destructor
33. **void** deleteparticular(string); //Function to delete one node in linked list
34. **void** deleteall(); //Function to delete all nodes in linked list
35. **protected**:
36. **int** count;
37. nodeType<type> \*first; //Declaration of pointer first
38. nodeType<type> \*last; //Declaration of pointer last
39. };


43. **template** <**class** type>
44. linkedListType<type> :: linkedListType() //Default constructor to initialize variables
45. {
46. first = NULL;
47. last = NULL;
48. count = 0;
49. }
51. **template**<**class** type>
52. linkedListType<type> :: ~linkedListType() //Destructor
53. {
54. first = NULL;
55. last = NULL;
56. count = 0;
57. }
59. **template** <**class** type>
60. **void** linkedListType<type>::deleteparticular(string searchengineno)
61. {
62. nodeType<type> \*current; //Declaration of pointer current
63. nodeType<type> \*trail; //Declaration of pointer trail
64. **bool** found;
65. **if**(first == NULL) //If there is no linked list
66. cout << "\t\t\tCannot perform the operation! There is an empty list." << endl;
67. **else**
68. {
69. **if**(first->carInfo.engineno == searchengineno) //If first node is the same car engine number in the linked list with the input
70. {
71. current = first;
72. first = first->link;
74. **if**(first == NULL) // if the original list has only one node
75. {
76. last = NULL
77. ;           }
79. **delete** current;
80. count--;
81. }
82. **else**
83. {
84. found = **false**;
85. trail = first;
86. current = first->link;
87. **while**(current !=NULL && !found) //Condition to exit the loop: when current == NULL or when found == true
88. {
89. **if**(current->carInfo.engineno != searchengineno) //If car engine number pointed by the current pointer is not the same as the input
90. {
91. trail = current;
92. current = current->link;
93. }
94. **else**
95. found = **true**;
96. }
97. **if**(found) //If found == true
98. {
99. trail->link = current->link;
101. **if**(last == current)
102. last = trail;
104. **delete** current;
105. count--;
106. }
107. **else** //If car engine number is not exist
108. cout << "\t\t\tCar engine number doesn't exist! Please try again." << endl;
109. }
110. }
111. }
113. **template** <**class** type>
114. **void** linkedListType<type>::deleteall() //Function to destroy the linked list
115. {
116. nodeType<type> \*temp; //Declaration of pointer temp
117. **while**(first != NULL)
118. {
119. temp = first;
120. first = first->link;
121. **delete** temp;
122. }
124. first = NULL;
125. last = NULL;
126. count = 0;
127. }
129. #endif

**orderedLinkedList.h**

1. #include <iostream>
2. #ifndef ORDEREDLINKEDLIST\_H
3. #define ORDEREDLINKEDLIST\_H
4. #include <iomanip>
5. #include "linkedListType.h"
6. #include <fstream>
8. **using** **namespace** std;
10. **template** <**class** type>
11. **class** orderedLinkedList : **public** linkedListType<type>
12. {
13. **public**:
14. orderedLinkedList(); //Default constructor
15. ~orderedLinkedList(); //Destructor
16. **void** addordered(type); //Function to add nodes in ordered linked list
17. **void** displayall(); //Function to display all nodes in ordered linked list
18. **void** displayotrbased(); //Function display nodes in ordered linked list from highest to lowest on the road price
19. **void** displayparticular(string); //Function to display particular node in ordered linked list
20. string checkengineno(string); //Function to check duplication of car engine number in input
21. **void** readDataOrdered(); //Function to store car sales information from text file to temporary variables
22. **void** addDataOrdered(string, string, string, **int**, **double**, **double**, **double**); //Function to assign temporary variables to variables in struct carInfo
23. **void** writeData(); //Function to write car sales information from ordered linked list to text file
24. };

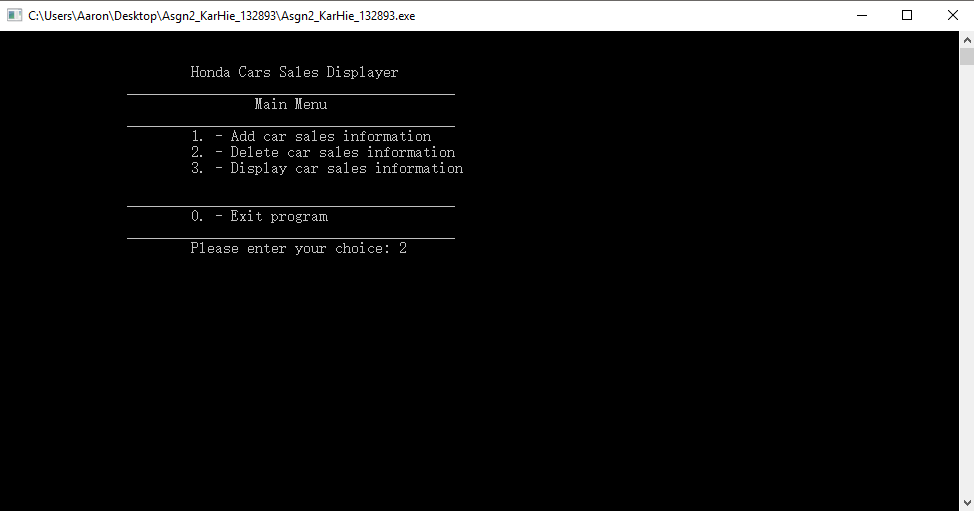

28. **template** <**class** type>
29. orderedLinkedList<type>::orderedLinkedList()
30. {
32. }
34. **template** <**class** type>
35. orderedLinkedList<type>::~orderedLinkedList()
36. {
38. }
40. **template** <**class** type>
41. **void** orderedLinkedList<type>::readDataOrdered()
42. {
43. string engine, mod, col;
44. **int** disp;
45. **double** sellingp, retailp, otrprice;
46. ifstream inData; //File stream declaration
47. inData.open("CarDetails.txt"); //Open the text file
48. **while**(inData >> engine) //When there is data to input after a "enter"
49. {
50. inData.ignore();
51. getline(inData, mod);
52. getline(inData, col);
53. inData >> disp;
54. inData.ignore();
55. inData >> sellingp;
56. inData.ignore();
57. inData >> retailp;
58. inData.ignore();
59. inData >> otrprice;
60. inData.ignore();
62. addDataOrdered(engine, mod, col, disp, sellingp, retailp, otrprice); //Function call
63. }
64. inData.close(); //Close the text file
65. }
67. **template**<**class** type>
68. **void** orderedLinkedList<type>::addDataOrdered(string engine, string mod, string col, **int** disp, **double** sellingp, **double** retailp, **double** otrprice) //Function to assign temporary variables to variables in struct carInfo
69. {
70. nodeType<type> \*current;
71. nodeType<type> \*trail;
72. nodeType<type> \*newNode;
73. **bool** found;
74. newNode = **new** nodeType<type>;
75. newNode->carInfo.engineno = engine; //Assigning variables
76. newNode->carInfo.model = mod;
77. newNode->carInfo.color = col;
78. newNode->carInfo.displacement = disp;
79. newNode->carInfo.sellingprice = sellingp;
80. newNode->carInfo.retailprice = retailp;
81. newNode->carInfo.ontheroad = otrprice;
82. newNode->link = NULL;
84. **if**(**this**->first == NULL) //When there is no linked list
85. {
86. **this**->first = newNode;
87. **this**->last = newNode;
88. **this**->count++;
89. }
90. **else**
91. {
92. current = **this**->first;
93. found = **false**;
94. **while**(current != NULL && !found)
95. {
96. **if**(current->carInfo.ontheroad <= otrprice) //When on the price price inputted is larger than the one pointed by current
97. found = **true**;
98. **else**
99. {
100. trail = current;
101. current = current->link;
102. }
103. }
104. **if**(current == **this**->first) //Assign the inputted node in front of the first node
105. {
106. newNode->link = **this**->first;
107. **this**->first = newNode;
108. **this**->count++;
109. }
110. **else** //Assign the inputted node in front of the current
111. {
112. trail->link = newNode;
113. newNode->link = current;
115. **if**(current == NULL)
116. **this**->last = newNode;
118. **this**->count++;
119. }
120. }
121. }
123. **template** <**class** type>
124. **void** orderedLinkedList<type>::writeData()
125. {
126. nodeType<type> \*current;
127. current = **this**->first;
128. ofstream outData;
129. outData.open("CarDetails.txt");
130. **while**(current != NULL) //Write each node of the ordered linked list to text file
131. {
132. outData << current->carInfo.engineno << endl << current->carInfo.model << endl << current->carInfo.color << endl << current->carInfo.displacement << endl << current->carInfo.sellingprice << endl << current->carInfo.retailprice << endl << current->carInfo.ontheroad << endl << endl;
133. current = current->link;
134. }
135. outData.close();
136. }
138. **template** <**class** type>
139. string orderedLinkedList<type>::checkengineno(string engine)
140. {
141. nodeType<type> \*current;
142. current = **this**->first;
143. **while**(current != NULL)
144. {
145. **if**(current->carInfo.engineno == engine) //To check the duplucation of the car engine number
146. {
147. cin.clear();
148. cin.ignore(100, '\n');
149. cout << "Engine number already exist. Please try again with another engine number: ";
150. getline(cin, engine);
151. **while**(engine.length() != 6 || engine.find\_first\_not\_of("1234567890") != string::npos)
152. {
153. cin.clear();
154. cin.ignore(100, '\n');
155. cout << "Invalid input. Please try again: ";
156. getline(cin, engine);
157. }
158. current = **this**->first; //To reset the current pointer to the first node
159. }
160. **else**
161. current = current->link;
162. }
163. **return** engine;
164. }
166. **template** <**class** type>
167. **void** orderedLinkedList<type>::addordered(type carinfo)
168. {
169. nodeType<type> \*current;
170. nodeType<type> \*trail;
171. nodeType<type> \*newNode;
173. **bool** found;
175. newNode = **new** nodeType<type>;
176. newNode->carInfo.engineno = carinfo.engineno; //Assigning variables
177. newNode->carInfo.model = carinfo.model;
178. newNode->carInfo.color = carinfo.color;
179. newNode->carInfo.displacement = carinfo.displacement;
180. newNode->carInfo.sellingprice = carinfo.sellingprice;
181. newNode->carInfo.retailprice = carinfo.retailprice;
182. newNode->carInfo.ontheroad = carinfo.ontheroad;
183. newNode->link = NULL;
185. **if**(**this**->first == NULL) //When there is no linked list
186. {
187. **this**->first = newNode;
188. **this**->last = newNode;
189. **this**->count++;
190. }
191. **else**
192. {
193. current = **this**->first;
194. found = **false**;
195. **while**(current != NULL && !found)
196. {
197. **if**(current->carInfo.ontheroad <= carinfo.ontheroad) //When on the road price inputted is larger than the one pointed by current
198. found = **true**;
199. **else**
200. {
201. trail = current;
202. current = current->link;
203. }
204. }
205. **if**(current == **this**->first) //Assign the inputted node in front of the first node
206. {
207. newNode->link = **this**->first;
208. **this**->first = newNode;
209. **this**->count++;
210. }
211. **else** //Assign the inputted node in front of the current
212. {
213. trail->link = newNode;
214. newNode->link = current;
216. **if**(current == NULL)
217. **this**->last = newNode;
219. **this**->count++;
220. }
221. }
222. }
224. **template** <**class** type>
225. **void** orderedLinkedList<type>::displayparticular(string searchengineno)
226. {
227. nodeType<type> \*current;
228. **bool** found = **false**;
229. current = **this**->first;
230. **while**(current !=NULL && !found)
231. {
232. **if**(current->carInfo.engineno == searchengineno) //If the inputted car engine no is the same as the one pointed by current
233. found = **true**;
234. **else**
235. current = current->link; //Current will point to the next node
236. }
237. **if**(found) //If found == true
238. {
239. cout << "Here is the car sales information: " << endl << endl;
240. cout << "Car engine number: " << current->carInfo.engineno << endl;
241. cout << "Model: " << current->carInfo.model << endl;
242. cout << "Color: " << current->carInfo.color << endl;
243. cout << "Displacement (cc): " << current->carInfo.displacement << endl;
244. cout << "Car price:- " << endl;
245. cout << "Selling price (inclusive of 6% GST): RM" << current->carInfo.sellingprice << endl;
246. cout << "Retail price without insurance (inclusive of 6% GST): RM" << current->carInfo.retailprice << endl;
247. cout << "On the road price with insurance (inclusive of 6% GST): RM" << current->carInfo.ontheroad << endl;
248. }
249. **else** //If found == false
250. {
251. cout << "Engine number doesn't exist! Please try again." << endl;
252. }
253. system("pause");
254. }
256. **template** <**class** type>
257. **void** orderedLinkedList<type>::displayotrbased()
258. {
259. nodeType<type> \*current;
260. system("cls");
261. current = **this**->first;
262. cout << "Here is the car sales information based on the on the road price of the cars:" << endl << endl;
263. cout << "Engine no." << "\t\t\t" << "Model" << setw(25) << "Color" << "\t" << "Disp." << "\t" << "Selling price" << "\t" << "Retail price" << setw(15) << "OTR price" << endl;
264. **while**(current != NULL) //When current is pointing to any node
265. {
266. cout << setw(9) << current->carInfo.engineno << setw(28) << current->carInfo.model << setw(25) << current->carInfo.color << setw(6) << current->carInfo.displacement << setw(17) << current->carInfo.sellingprice << setw(15) << current->carInfo.retailprice << setw(15) << current->carInfo.ontheroad << endl;
267. current = current->link; //Current points to the next node
268. }
269. cout << endl;
270. cout << "Total number of cars sold is: " << **this**->count << endl << endl;
271. system("pause");
272. }
274. **template** <**class** type>
275. **void** orderedLinkedList<type>::displayall()
276. {
277. nodeType<type> \*current;
278. system("cls");
279. current = **this**->first;
280. cout << "Below is all of the car sales information:" << endl << endl;
281. cout << "Engine no." << "\t\t\t" << "Model" << setw(25) << "Color" << "\t" << "Disp." << "\t" << "Selling price" << "\t" << "Retail price" << setw(15) << "OTR price" << endl;
282. **while**(current != NULL) //When current is pointing to any node
283. {
284. cout << setw(9) << current->carInfo.engineno << setw(28) << current->carInfo.model << setw(25) << current->carInfo.color << setw(6) << current->carInfo.displacement << setw(17) << current->carInfo.sellingprice << setw(15) << current->carInfo.retailprice << setw(15) << current->carInfo.ontheroad << endl;
285. current = current->link; //Current points to the next node
286. }
287. cout << endl;
288. cout << "Total number of cars sold is: " << **this**->count << endl << endl;
289. system("pause");
290. }
292. #endif

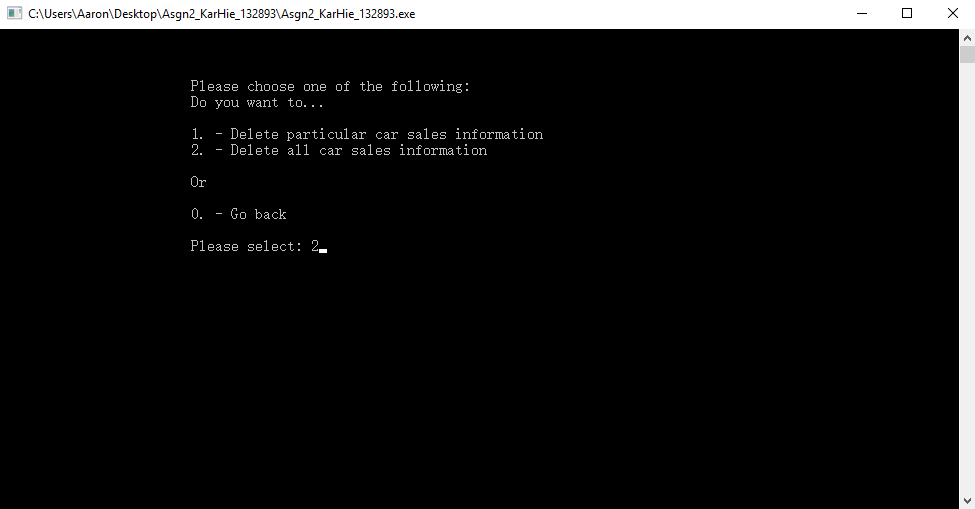
**unorderedLinkedList.h**

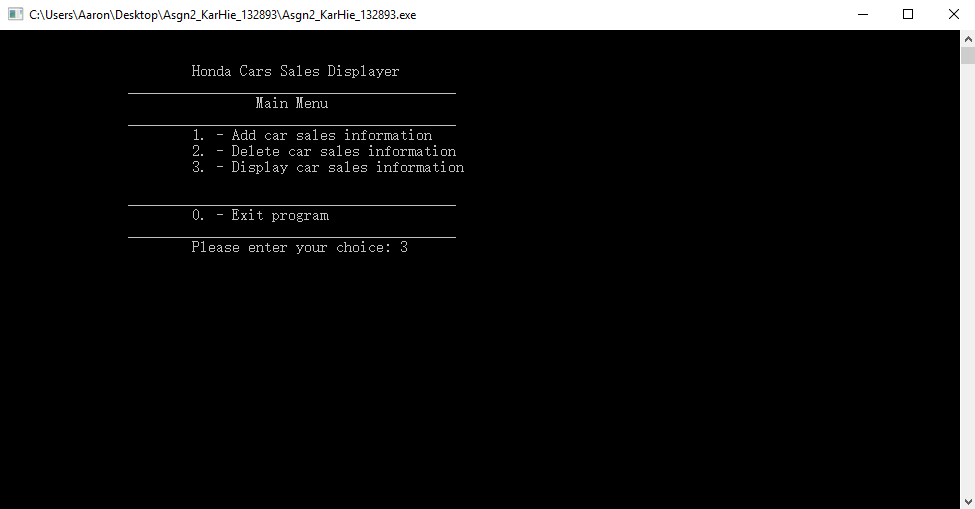
1. #ifndef UNORDEREDLINKEDLIST\_H
2. #define UNORDEREDLINKEDLIST\_H
3. #include <iostream>
4. #include "linkedListType.h"
5. #include <cassert>
6. #include <fstream>
8. **using** **namespace** std;
10. **template** <**class** type>
11. **class** unorderedLinkedList : **public** linkedListType<type>
12. {
13. **public**:
14. unorderedLinkedList(); //Default constructor
15. ~unorderedLinkedList(); //Destructor
16. **void** insertfirst(type); //Function to insert as the first node of the unordered linked list
17. **void** insertlast(type); //Function to insert as the last node of the unordered linked list
18. **void** readDataUnordered(); //Function to store car sales information from text file to temporary variables
19. **void** addDataUnordered(string, string, string, **int**, **double**, **double**, **double**); //Function to assign temporary variables to variables in struct carInfo
20. };
22. **template**<**class** type>
23. unorderedLinkedList<type>::unorderedLinkedList()
24. {
26. }
28. **template**<**class** type>
29. unorderedLinkedList<type>::~unorderedLinkedList()
30. {
32. }
34. **template**<**class** type>
35. **void** unorderedLinkedList<type>::readDataUnordered()
36. {
37. string engine, mod, col;
38. **int** disp;
39. **double** sellingp, retailp, otrprice;
40. ifstream inData; //File stream declaration
41. inData.open("CarDetails.txt"); //Open the text file
42. **while**(inData >> engine) //When there is data to input after a "enter"
43. {
44. inData.ignore();
45. getline(inData, mod);
46. getline(inData, col);
47. inData >> disp;
48. inData.ignore();
49. inData >> sellingp;
50. inData.ignore();
51. inData >> retailp;
52. inData.ignore();
53. inData >> otrprice;
54. inData.ignore();
56. addDataUnordered(engine, mod, col, disp, sellingp, retailp, otrprice); //Function call
57. }
58. inData.close(); //Close the text file
59. }
61. **template**<**class** type>
62. **void** unorderedLinkedList<type>::addDataUnordered(string engine, string mod, string col, **int** disp, **double** sellingp, **double** retailp, **double** otrprice) //Function to assign temporary variables to variables in struct carInfo
63. {
64. nodeType<type> \*newNode;
65. newNode = **new** nodeType<type>;
66. newNode->carInfo.engineno = engine;
67. newNode->carInfo.model = mod;
68. newNode->carInfo.color = col;
69. newNode->carInfo.displacement = disp;
70. newNode->carInfo.sellingprice = sellingp;
71. newNode->carInfo.retailprice = retailp;
72. newNode->carInfo.ontheroad = otrprice;
74. **if**(**this**->first == NULL) //When there is no linked list
75. {
76. **this**->first = newNode;
77. **this**->last = newNode;
78. }
79. **else**
80. {
81. **this**->last->link = newNode; //insert last
82. **this**->last = newNode;
83. newNode->link = NULL;
84. }
85. }
87. **template** <**class** type>
88. **void** unorderedLinkedList<type>::insertfirst(type carinfo)
89. {
90. nodeType<type> \*newNode;
91. newNode = **new** nodeType<type>;
92. assert(newNode != NULL);
93. newNode->carInfo.engineno = carinfo.engineno; //Assigning variables
94. newNode->carInfo.model = carinfo.model;
95. newNode->carInfo.color = carinfo.color;
96. newNode->carInfo.displacement = carinfo.displacement;
97. newNode->carInfo.sellingprice = carinfo.sellingprice;
98. newNode->carInfo.retailprice = carinfo.retailprice;
99. newNode->carInfo.ontheroad = carinfo.ontheroad;
100. newNode->link = **this**->first;
101. **this**->first = newNode;
102. **if**(**this**->last == NULL) //If last points to nothing
103. {
104. **this**->last = newNode; //Last pointer points to newNode
105. newNode->link = NULL;
106. }
107. }
109. **template** <**class** type>
110. **void** unorderedLinkedList<type>::insertlast(type carinfo)
111. {
112. nodeType<type> \*newNode;
113. newNode = **new** nodeType<type>;
114. assert(newNode != NULL);
115. newNode->carInfo.engineno = carinfo.engineno;
116. newNode->carInfo.model = carinfo.model;
117. newNode->carInfo.color = carinfo.color;
118. newNode->carInfo.displacement = carinfo.displacement;
119. newNode->carInfo.sellingprice = carinfo.sellingprice;
120. newNode->carInfo.retailprice = carinfo.retailprice;
121. newNode->carInfo.ontheroad = carinfo.ontheroad;
123. **if**(**this**->first == NULL) //If there is no linked list
124. {
125. **this**->first = newNode;
126. **this**->last = newNode;
127. newNode->link = NULL;
128. }
129. **else**
130. {
131. **this**->last->link = newNode;
132. **this**->last = newNode;
133. newNode->link = NULL;
134. }
135. }
137. #endif

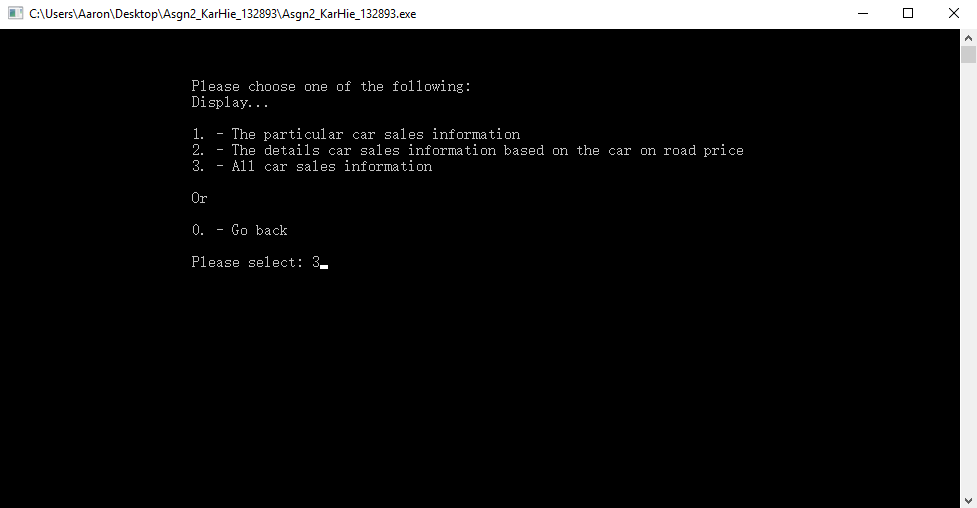
Print screen of sample input/output

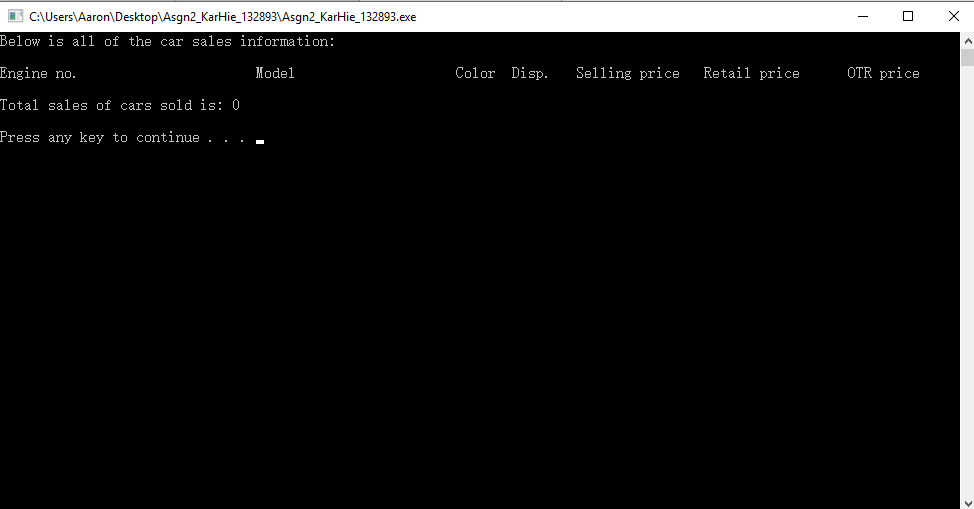
**Scenario 1: Delete all and show**



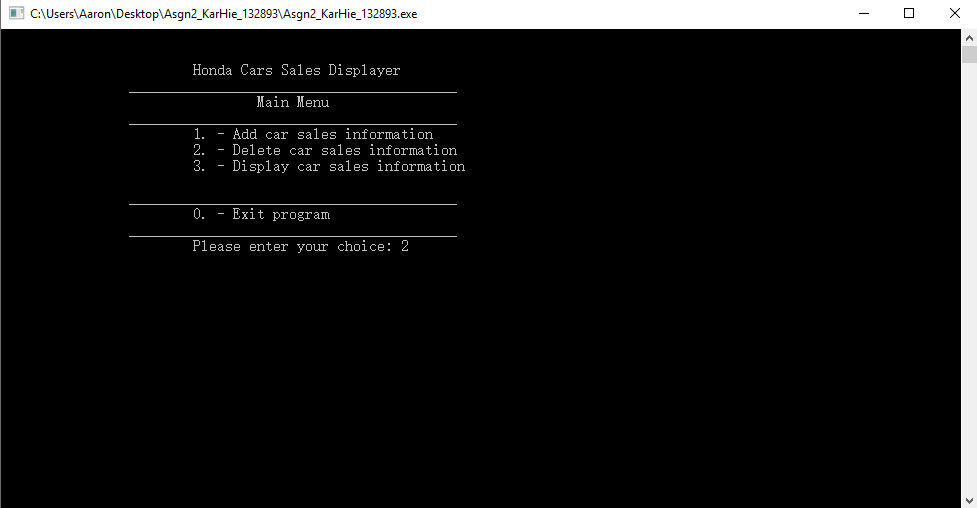


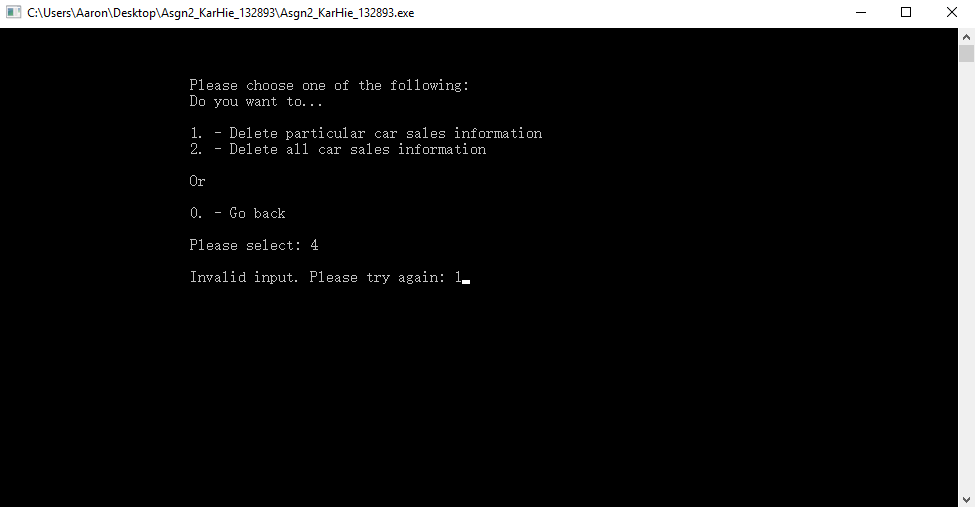


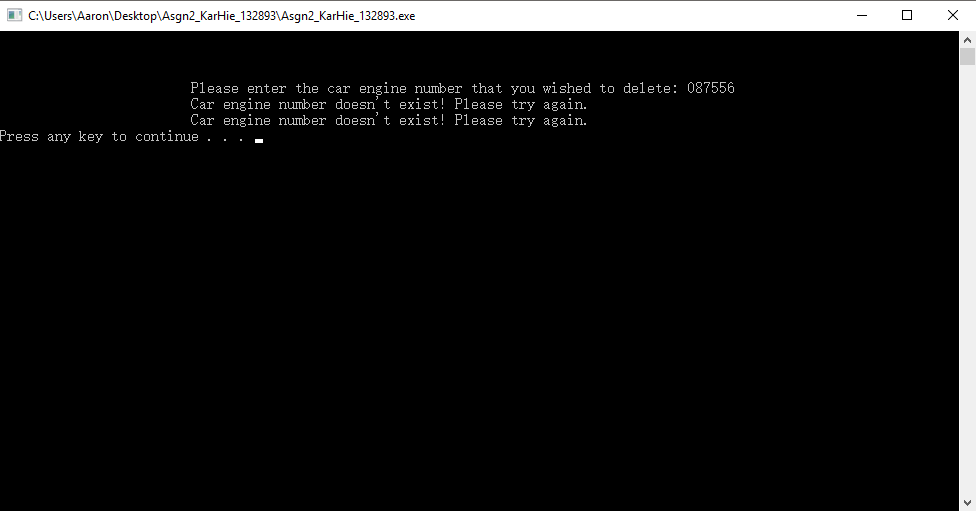




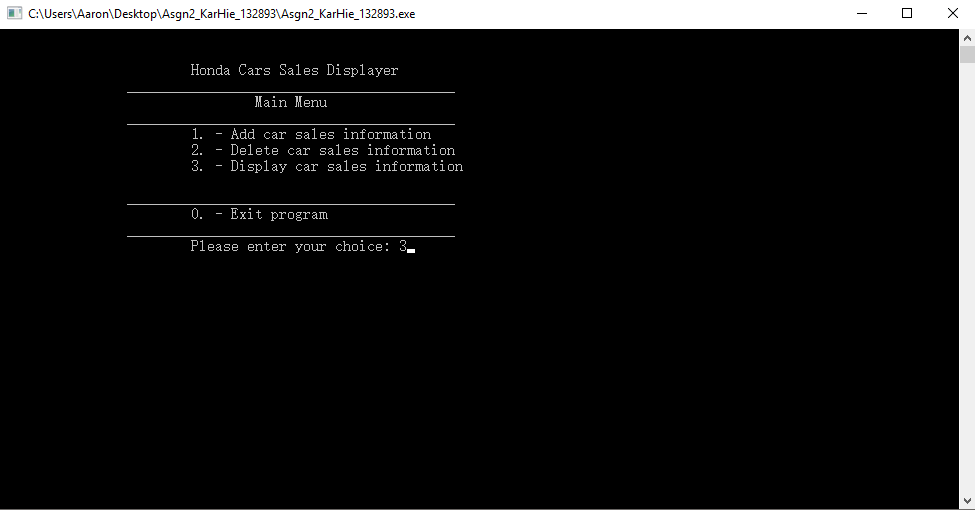
**Scenario 2: Delete particular with wrong input**

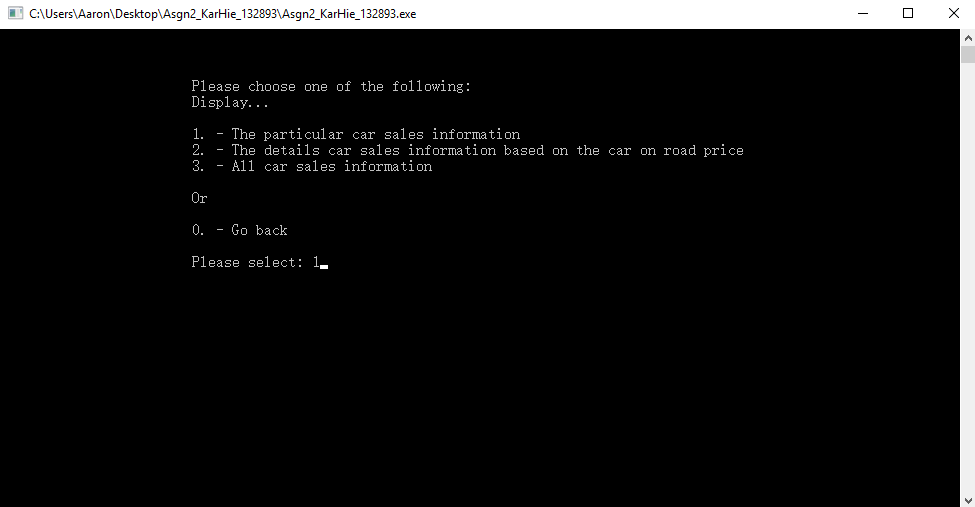


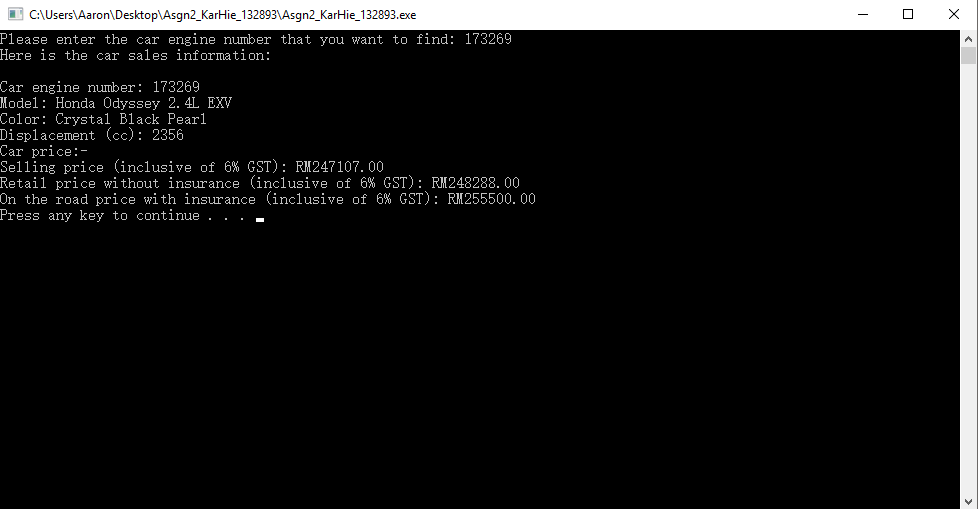




**Scenario 3: Display particular**







**Scenario 4: Wrong input and display**

