

**Database and Data Structures**

**In-course Group Assignment**

**Documentation**

Name : Ng Ho Kit

Lim Eng Shun

Student I.D. : TP035012

TP034846

Intake Code : UCDF1405ICT(SE)

Module Code : AAPP001-4-2-DBDS

Lecturer Name : Hasbullah Bin Osman

Due Date : 20th November 2015

Table of Contents

[1.0 Introduction 2](#_Toc435781763)

[2.0 Data Structure Concepts Implemented 3](#_Toc435781764)

[2.1 Linked List (Part A) 3](#_Toc435781765)

[2.1.1 Structure 3](#_Toc435781766)

[2.1.2 Add New Customer (Insert Node) 4](#_Toc435781767)

[2.1.3 Remove Customer (Delete Node) 5](#_Toc435781768)

[2.1.4 Print Out Entire List 6](#_Toc435781769)

[2.2 Queue (Part B) 7](#_Toc435781770)

[2.2.1 Structures 7](#_Toc435781771)

[2.2.2 Bus Arrive (Enqueue) 8](#_Toc435781772)

[2.2.3 Bus Leave (Dequeue) 10](#_Toc435781773)

[2.2.4 Print Out Entire Queue 11](#_Toc435781774)

[3.0 Sample Output 12](#_Toc435781775)

[3.1 Part A: Linked List 12](#_Toc435781776)

[3.1.1 Menu 12](#_Toc435781777)

[3.1.2 Selected option: 1 & 2 (Add customer) 12](#_Toc435781778)

[3.1.3 Selected option: 3 & 4 (Print out customer list) 13](#_Toc435781779)

[3.1.4 Selected option: 5 (remove customer) 14](#_Toc435781780)

[3.2 Part B: Queue 16](#_Toc435781781)

[3.2.1 Menu 16](#_Toc435781782)

[3.2.2 Bus Arrive 16](#_Toc435781783)

[3.2.3 Bus Leave 18](#_Toc435781784)

[3.2.4 Print Queue and Total Busses 19](#_Toc435781785)

[3.2.5 Search Bus 20](#_Toc435781786)

[4.0 Conclusion 21](#_Toc435781787)

[5.0 Member Contribution Report 22](#_Toc435781788)

[6.0 References 23](#_Toc435781789)

# 1.0 Introduction

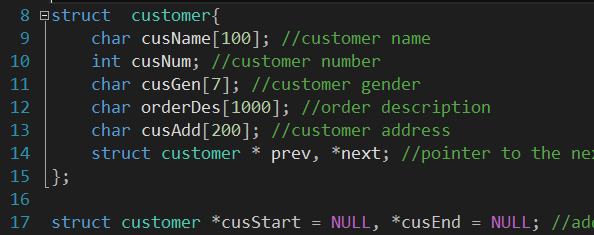
Linked list is a data structure used to collect a sequence of objects, which allows efficient addition and removal of the elements in the middle of the sequence. A linked list uses a sequence of nodes which stores a value and a reference to the next node in the sequence. The value in the node can be accessed via the first node of the list (Juan, 2006).

In this assignment, as the question requirement asked for, we implemented doubly linked list for Part A and also singly linked list for queue in Part B.

# 2.0 Data Structure Concepts Implemented

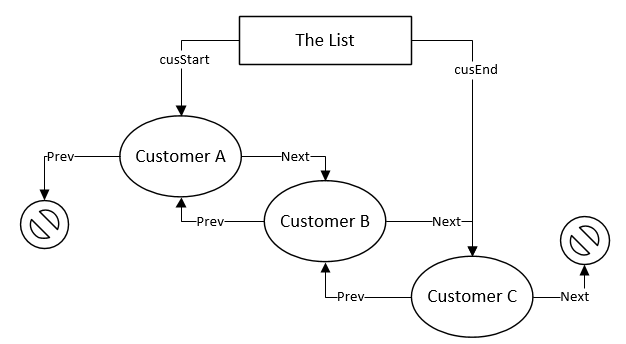
## 2.1 Linked List (Part A)

### 2.1.1 Structure



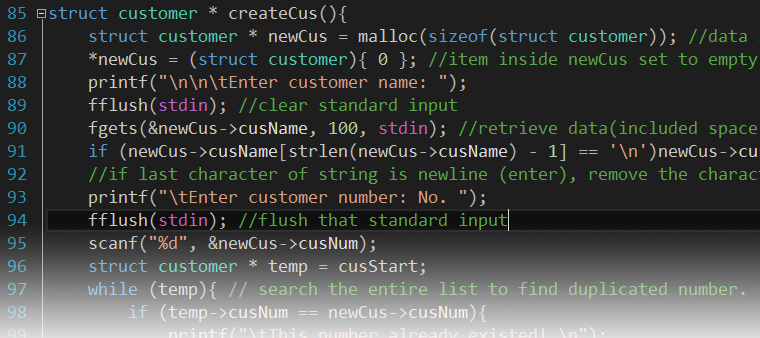
*Figure 1*

For a structure of a doubly linked list, the node contains two pointers which point to the previous node and the next node. When one of them are *NULL*, then means the node will be the end of the list. In the implementation, *struct customer* has shown the characteristic of a node in doubly linked list, it contains two pointers to *struct customer* which is next and previous, other members are for storing the data of customer. There is also two pointers for pointing the first node and last node of the list which is *cusStart* and *cusEnd*. The structure of the implemented linked list will be as shown as the diagram below.



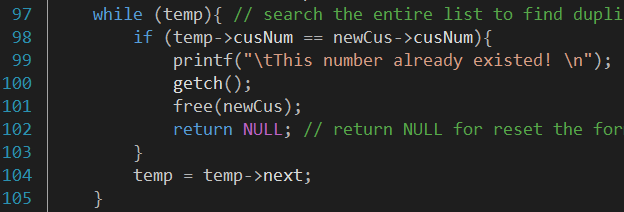
*Diagram 1*

### 2.1.2 Add New Customer (Insert Node)



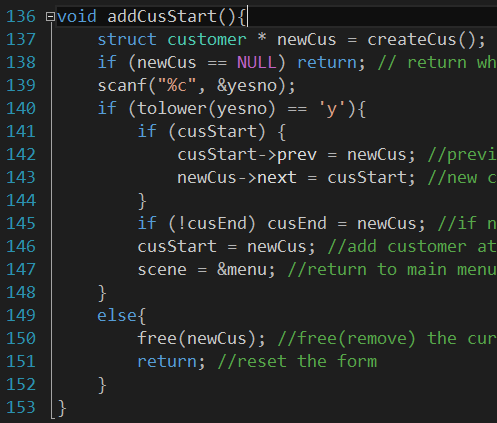
*Figure 2*

When adding a new customer (insert new node), a new *struct customer* have to be allocated in memory by using memory allocation (*malloc*) function. When a new block of memory was allocated, it was not empty because of previous use. So, a clean-up process which is set the certain block of memory into empty is necessary to be carried up. The block of memory will be suitable to be filled with data after it was empty. Validation processes will be carried up when filling up the customer number and gender.

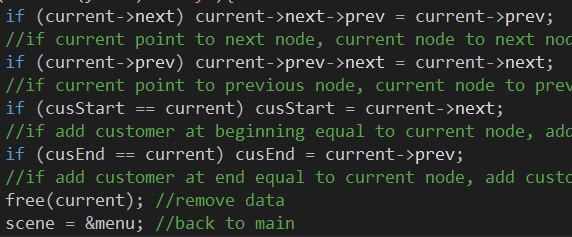


*Figure 3*

During the validation for the customer number, a while loop will be carried up to search through the entire list to make sure the customer number will not be duplicated because it is the identity for every customer. When the data was invalid, then the allocated *struct customer* will be free and the adding process will be restart to let user fill up data with a valid data.

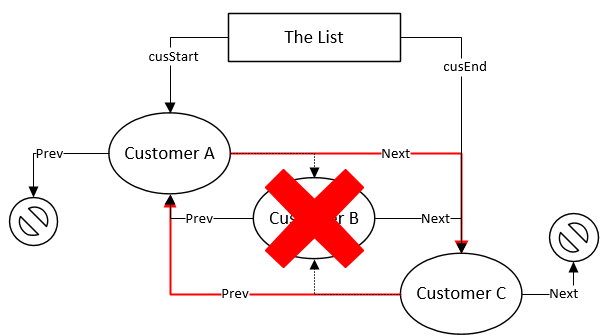
After all the data was filled, the details which just filled will be prompted out with a confirmation. When user is confirm to add the customer, the new customer node will be insert into the list, the location will depends on user. When adding the new customer to the beginning, if the list was empty, both *cusStart* and *cusEnd* will point to the new customer node. If there is a customer presented in the list, the previous node of *cusStart* will point to the new customer and the next node of new customer will point to the *cusStart*, so the new customer will be the new first customer in the list. When adding the new customer to the end, a similar process will be carried out but in the coding, the next will swap with previous and the *cusStart* will swap with *cusEnd*.

### 2.1.3 Remove Customer (Delete Node)

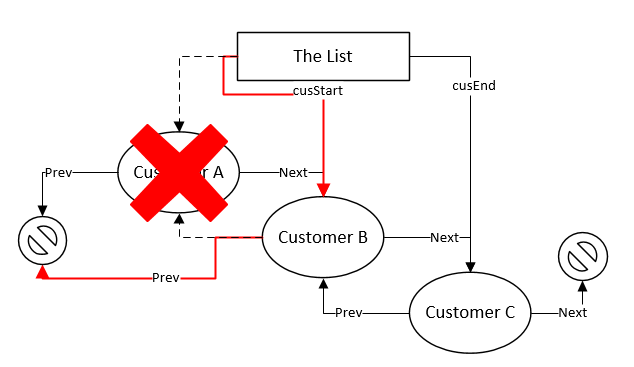


*Figure 5*

When deleting the node, the previous node of the next node of the node to be removed will be point to the previous node of the node to be removed. The next node of the previous node of the node to be removed will also point to the next node of the node to be removed. When the node which to be removed is the first node, then the first node will point to the next node of the node, same concept is applied when the node to be removed is the last node. After all the relationships with the node are removed from the list, the node are safe to be free.

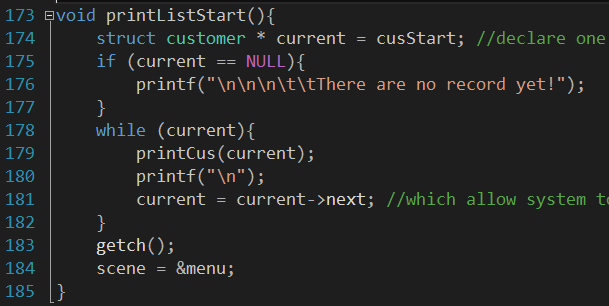


*Diagram 2, remove the node which at the center.*



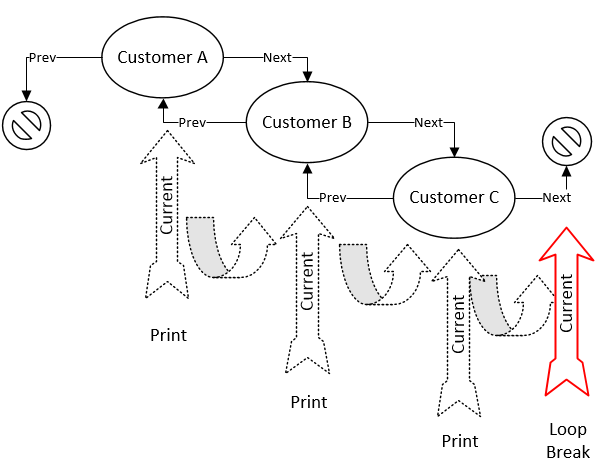
*Diagram 3, remove the first or the last node.*

### 2.1.4 Print Out Entire List



*Figure 6*

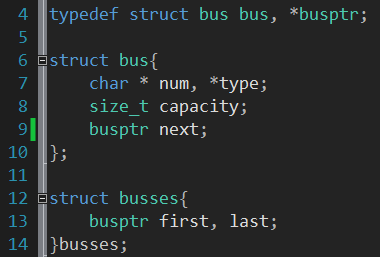
During the print entire list process, a while loop was carried out, if current node is existed, then the customer details will be printed and the current node will point to the next node (point previous node when print from end). The loop will break when the current node is not exist.



*Diagram 4, print entire list process*

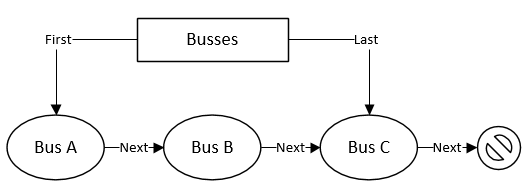
## 2.2 Queue (Part B)

### 2.2.1 Structures



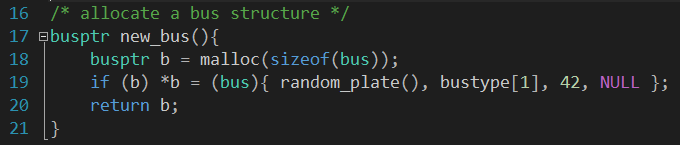
*Figure 7*

Queue can be implement in multiple ways such as singly array and singly linked list. In this program, the queue was implemented as a singly linked list. The benefit of use singly linked list to implement queue is easier to manage the data. Unlike doubly linked list, singly linked list can only go through one direction which means each node only contains the pointer which point to the next node. In the program implementation, *struct bus* has shown the characteristic of a node of singly linked list, it contains one pointer to *struct bus* which point to the next node, and other members are for storing the data of bus. There is also two pointers for pointing the first node and last node of the queue which is *busses.first* and *busses.last*. The structure of the implemented queue will be as shown as the diagram below.

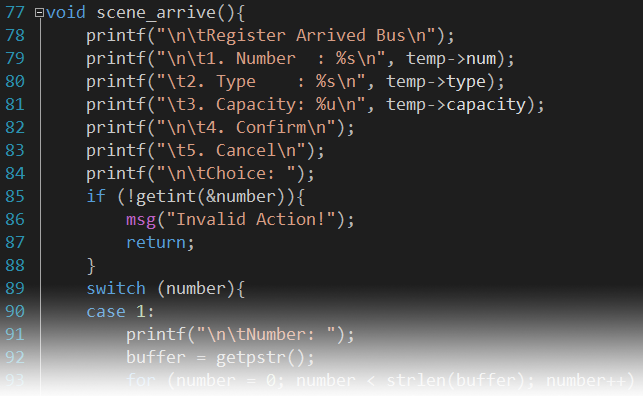


*Diagram 5*

### 2.2.2 Bus Arrive (Enqueue)

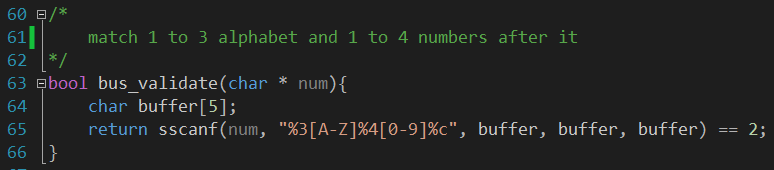


*Figure 8, allocate and initialize new bus*

**

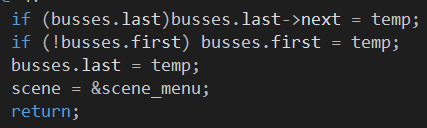
*Figure 9, scene to edit bus information*

Before adding a new bus into the queue, a new bus was allocated by using memory allocation (*malloc*) and initialized with random car plate (for testing purpose). After the new bus allocated, user will enter the scene which to edit the bus information. In the edit scene, validation process carried out when user edit the bus registration number (car plate).



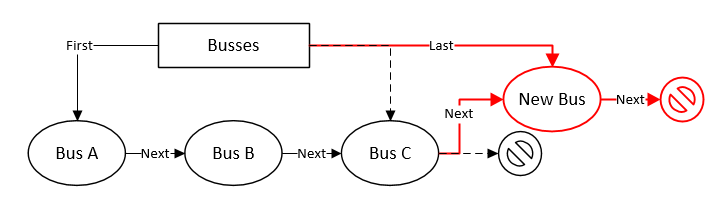
*Figure 10*

The validation of bus registration number is using string scan format function (*sscanf*) to match the pattern of car plate. The *sscanf* function will **return the number of data which successfully matched and stored** (cplusplus.com, n.d.). Base on this situation, an unused buffer with the enough size have to be passed to the function three times to make sure the matching function work properly. When the string contain a valid car plate without any excess character, two data will be matched and stored. By using this function, the bus registration number will able to be validated.

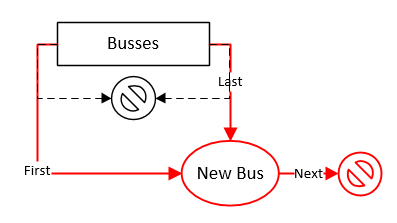


*Figure 11*

When user confirm to add the bus into the queue, an enqueue process will be carried out. If the last bus is not *NULL* (queue is not empty), then the next bus of last bus will point to the new bus. If the first bus is *NULL* (queue is empty), then point the first bus to the new bus. After that, the last bus will point to the new bus, so the new bus was added to the back of the queue.

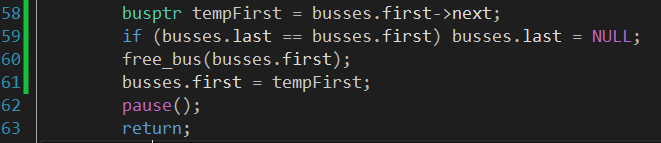


*Diagram 6, add bus when queue is not empty*



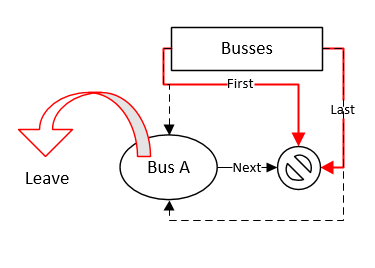
*Diagram 7, add bus when queue is empty*

### 2.2.3 Bus Leave (Dequeue)



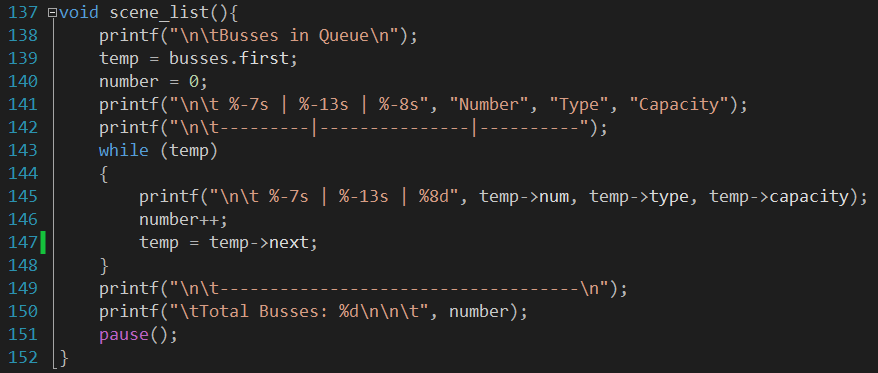
*Figure 12*

During a dequeuer process, the first bus will point to the next bus. The last bus will point to *NULL* when there is only one bus inside the queue.



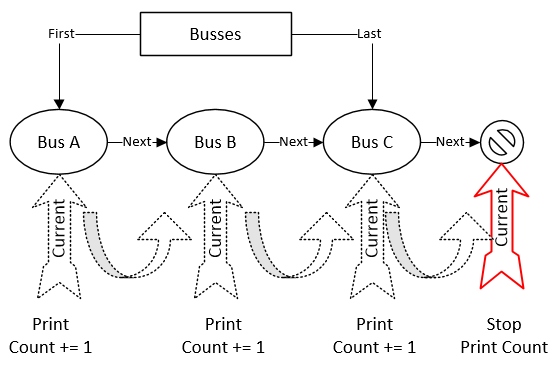
*Diagram 8, the only bus leaved*

### 2.2.4 Print Out Entire Queue



*Figure 13*

During the print entire list process, a while loop was carried out. The current bus will point to the first bus, if current bus is existed, then the bus will be printed and the current bus will point to the next bus. The loop will break when the current bus is not exist. The total number of busses will be counted during the loop.

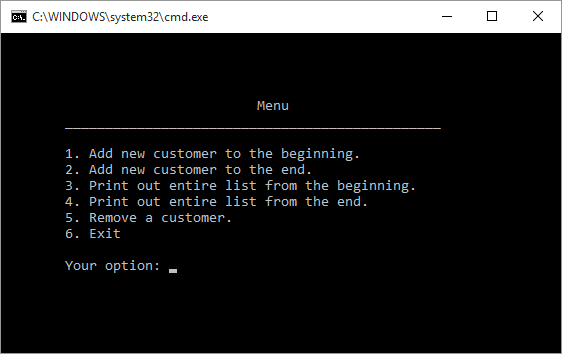


*Diagram 9*

# 3.0 Sample Output

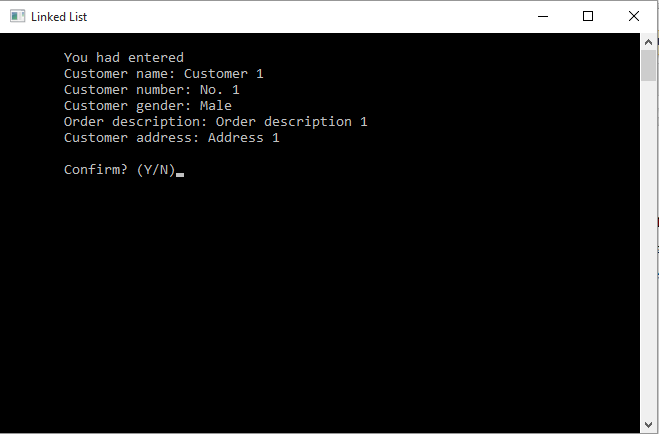
## 3.1 Part A: Linked List

### 3.1.1 Menu



*Figure 14*

### 3.1.2 Selected option: 1 & 2 (Add customer)



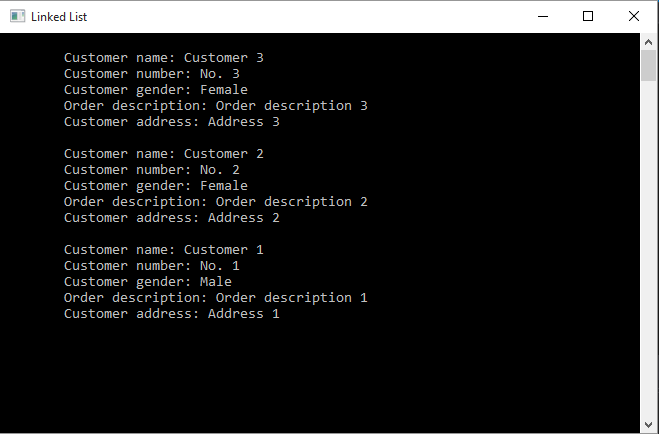
*Figure 15*

When user choose option 1 or 2, User are required to enter details as shown in figure above. Once the user finish fill up all the required details. System will pop up confirmation page, where the user have to confirm that he/she had entered correct information.

Once the user press “Y”, the system will redirect the user back to the menu page. If the user press “N”, the form will be reset and the user had to enter the required information again.

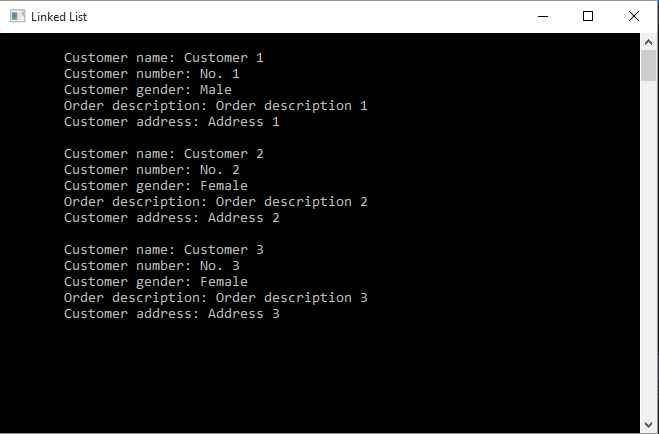
### 3.1.3 Selected option: 3 & 4 (Print out customer list)

The main different between option 3 and option 4 is that, option 3 will print out latest record first.



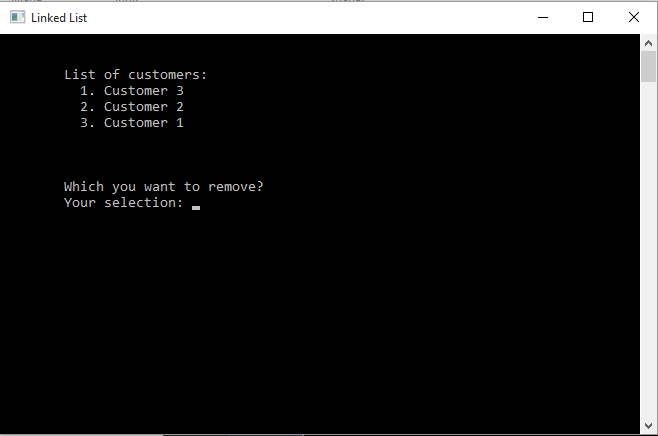
*Figure 16*

While for option 4, the record will print out the first record.



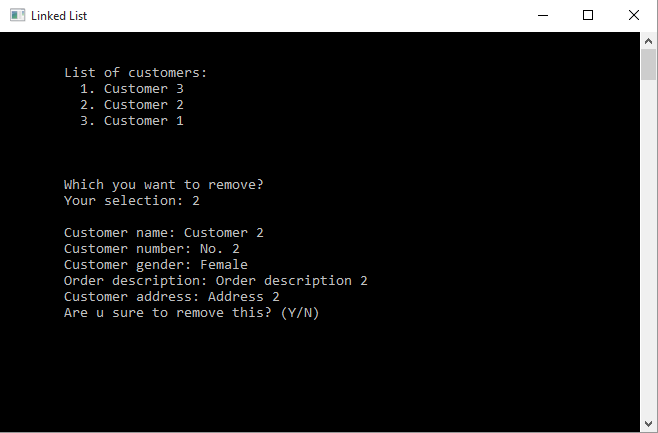
*Figure 17*

### 3.1.4 Selected option: 5 (remove customer)

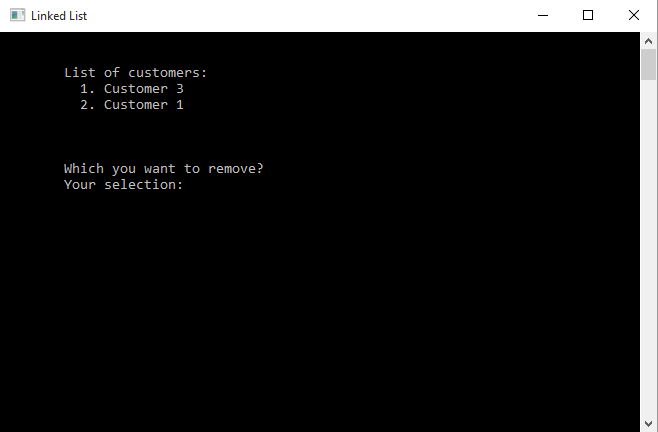


*Figure 18*

When user choose option number 5, user can choose what customer to be deleted. For example, user want to delete Customer 2. The system will appear confirmation whether the user want to remove the customer. Once the user confirmed, Customer 2 will be removed from the record.



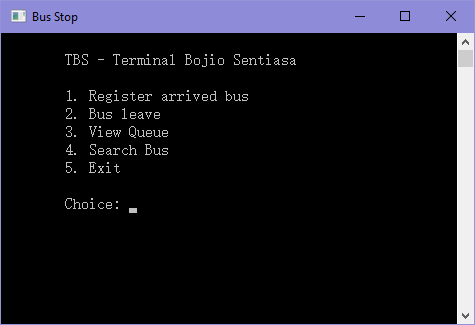
*Figure 19*



*Figure 20*

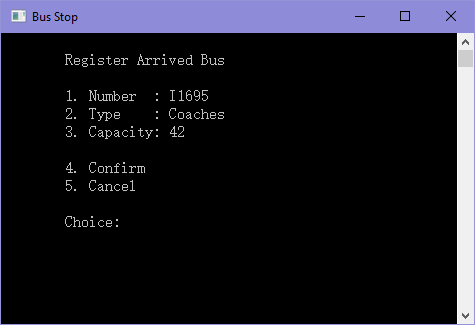
## 3.2 Part B: Queue

### 3.2.1 Menu

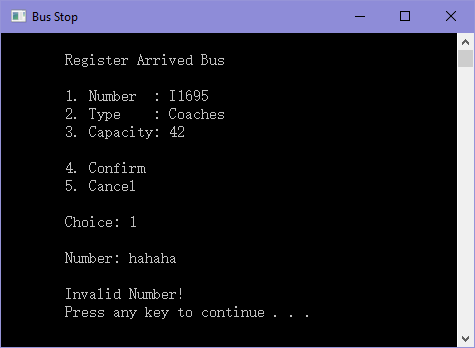


*Figure 21*

### 3.2.2 Bus Arrive

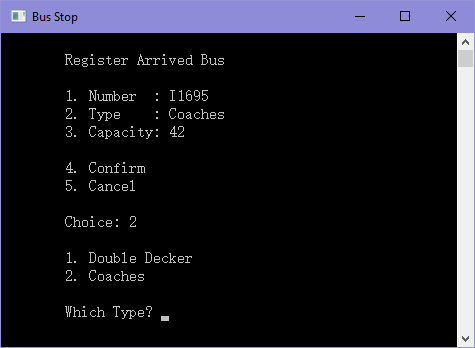


*Figure 22*



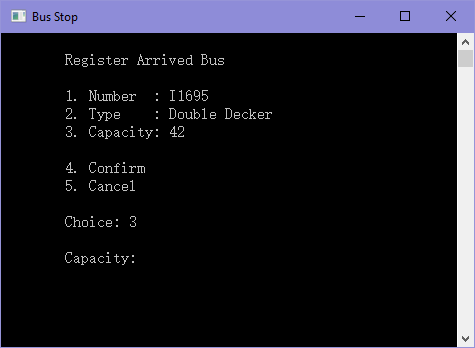
*Figure 23*

When user press option 1, user are required to insert bus carplate, the format of carplate can support up to 3 alphabets and 4 digits. For example, A1234, AB12 and ABC1234).



*Figure 24*

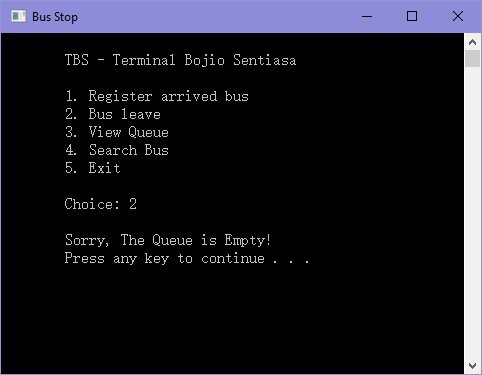
There are only 2 type of bus which is available in the terminal, which are double decker and coaches.



*Figure 25*

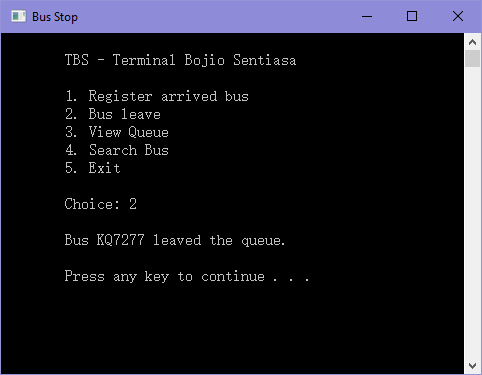
User are required to enter the capacity based on the bus type, some bus might have more capacity and some might have less.

### 3.2.3 Bus Leave



*Figure 26*

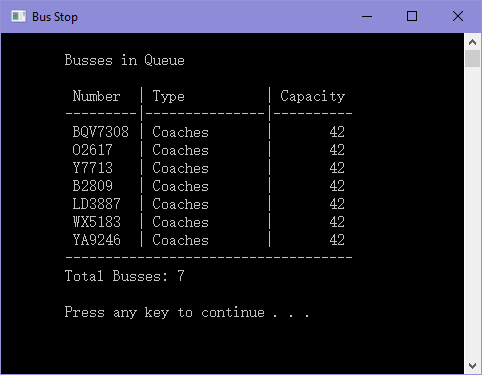
When the list is empty, the program will pop out a message, "Sorry, The queue is Empty!"



*Figure 27*

If there is any busses in the queue, the first bus that arrived at the terminal will leave the queue first.

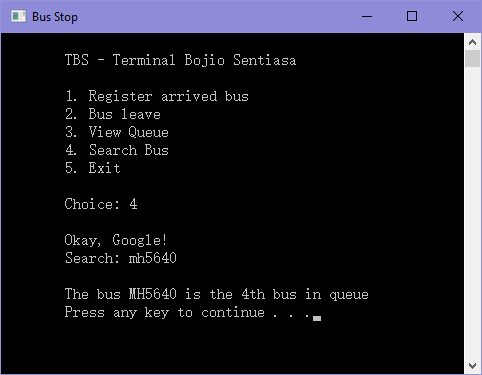
### 3.2.4 Print Queue and Total Busses



*Figure 28*

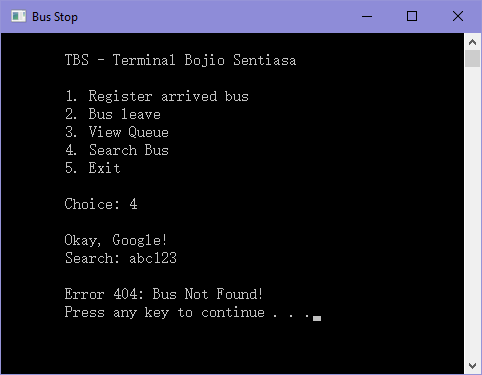
When the user select on option 3, the system will list out all the busses inside the queue and calculate total amount of bus that currently available at the terminal.

### 3.2.5 Search Bus



*Figure 29*

When the user want to search busses, user are required to enter the bus car-plate. For example, figure above shows that the bus MH5640 are at the 4th bus in the queue. If the user entered invalid car-plate or car-plate that are not existing inside the queue, System will pop out message, "Error 404: Bus Not Found!” Refer to *Figure 30.*



*Figure 30*

# 4.0 Conclusion

Compare to other data structures, linked list will be more flexible. Technically, linked list is able to contain unlimited items. But in the practical, the size of linked list are limited by the capacity of memory of the computer. An item can be simply added into or remove from any location of a linked list. To search an item in the linked list, the only way is loop through the whole list and compare every items. Compare to a fix size array, linked list doesn’t waste pointless memory, the memory block will be only allocated when required and be free after it was no longer required.

Queue has limited features due to its characteristics. Based on the main usage of queue, it cannot be sorted, insert or remove item from any location. Even tough queue has very limited features, but it will still be implemented in some program such as the ticket system at bank counter.

As conclusion, data structures improved the data managing of a program. Data structures make data easier to be accessed, the coding will be easier to be understood while data structures was implemented.

# 5.0 Member Contribution Report

|  |  |  |
| --- | --- | --- |
|  | **Ng Ho Kit** | **Lim Eng Shun** |
| **Linked List (Part A)** | 75% | 25% |
| **Queue (Part B)** | 25% | 75% |
| **Introduction** | 50% | 50% |
| **Explanation of Linked List** | 25% | 75% |
| **Explanation of Queue** | 75% | 25% |
| **Sample Output** | 50% | 50% |
| **Overall** | 50% | 50% |
| **Sign of Agreement** |  |  |

# 6.0 References

cplusplus.com, n.d. *sscanf.* [Online]   
Available at: http://www.cplusplus.com/reference/cstdio/sscanf/  
[Accessed 10 10 2015].

Juan, A. A., 2006. *PROGRAMMING with JAVA - Data Structures I.* [Online]   
Available at: http://www.uoc.edu/in3/emath/docs/java/ch20.pdf  
[Accessed 10 10 2015].