# CS 200

# Introduction to Programming

**Assignment 4**

Linked List – Stack , Queue and an Amusement Park

The course policy about plagiarism is as follows:

1. Students must not share the actual program code with other students.
2. Students must be prepared to explain any program code they submit.
3. Students cannot copy code from the Internet.
4. Students must indicate any assistance they received.
5. All submissions are subject to automated plagiarism detection.

Students are strongly advised that any act of plagiarism will be reported to the Disciplinary Committee.

**Note:**

The marks of this assignment will be scaled up accordingly because it has both parts in 80 marks. The linked list is part 1 this will be scaled to be out of 100 marks, the part 2 – stack and queue and part a and b of part 3 (reversing stack and queue) will also be scaled to be out of 100 marks. The part c of part 3 is optional. Solving it will earn you a bonus of 5 marks in some assignment or 10 marks in some lab or it can be used to cover two missed pre labs.

**Due:** 11:55 pm on Friday , December 9th , 2021

**Lead TA : Muhammad Hassnain (23100250@lums.edu.pk)**

## Smart pointer:

Smart pointers are class objects that behave like built-in pointers but also manage objects that you create with new ones so that you don't have to worry about when and whether to delete them - the smart pointers automatically delete the managed object for you at the appropriate time. To read an article explaining how to use smart pointers, click here.

In this assignment, you are not allowed to dynamically allocate memory.
You need to make use of smart pointers to implement this assignment.

## Late submission policy:

You are allowed to submit your assignment for up to three days, late, with the following penalties:

- 10% for work submitted up to 24 hours late
- 20% for work submitted up to 2 days late.
- 30% for work submitted up to 3 days late

No submission will be accepted after three days.

**Templates**:

Function templates are special functions that can operate with generic types. This allows us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type.

In C++, this can be achieved using template parameters. A template parameter is a special kind of parameter that can be used to pass a type as an argument: just like regular function parameters can be used to pass values to a function, template parameters allow to pass also types to a function. These function templates can use these parameters as if they were any other regular type.

The format for declaring function templates with type parameters is:

```
template <class identifier> function_declaration;
template <typename identifier> function_declaration;
```

For example, to create a template function that returns the greater one of two objects we could use:

```
template<class myType>
myType GetMax(myType a, myType b) {
if (a > b) return a;
else return b;
}
```

To learn more about templates, click here.

**PART 1:**
**DOUBLY LINKED LIST**

In this part, you will be applying what you learned in class to implement different functionalities of a linked list efficiently. The basic layout of a linked list is given to you in the LinkedList.h file.

The template ListItem in LinkedList.h represents a node in a linked list. The class LinkedList implements the linked list which contains pointers to head and tail and other function declarations. You are also given a file, test1.cpp for checking your solution. However, you are only allowed to make changes in LinkedList.cpp file.

**NOTE:**

When implementing functions, pay special attention to corner cases such as deleting from an empty list.

**Member functions:**

Write implementation for the following methods as described here.

- **LinkedList():**
  Simple default constructor.

- **LinkedList(const LinkedList<T>& otherList):**
  Simple copy constructor, given pointer to otherList this constructor copies all elements from otherList to new list.

- **void InsertAtHead(T item):**
  Inserts item at the start of the linked list.

- **void InsertAtTail(T item):**
  Inserts item at the end of the linked list.

- **void InsertAfter(T toInsert, T afterWhat):**

Traverse the list to find afterWhat and insert the toInsert after it.

- **ListItem<T> *getHead():**
  Returns the pointer to the head of the list.

- **ListItem<T> *getTail():**
  Returns the pointer to the tail of the list.

- **ListItem<T> *searchFor (T item):**
  Returns pointer to the item if it is in the list, returns null otherwise.

- **void deleteElement(T item):**

Find the element item and delete it from the list.

- **void deleteHead():**
  Delete the head of the list.

- **void deleteTail():**
  Delete the tail of the list.

- **int length():**
  Returns number of nodes in the list.

**Part 2 : STACKS AND QUEUES**

In this part, you will use your implementation of a linked list to write code for different methods of stacks and queues. As Part 1 is a pre-requisite for this part so you must have completed Part 1 before you attempt this part.

**STACK:**

Stack class contains LinkedList type object. You can only access member functions of the LinkedList class for your implementation of stack (and queue).

**Member functions:**

Write implementation for the following methods as described here.

- **Stack():**
  Simple base constructor.

- **Stack(const Stack<T>& otherStack):**
  Copies all elements from otherStack to the new stack.

- **void push(T item):**
  Pushes item at top of the stack.

- **T top():**
  Returns top of the stack without deleting it.

- **T pop():**
  Return and delete top of the stack.

- **int length():**
  Returns count of the number of elements in the stack.

- **bool isEmpty():**
  Return true if there is no element in the stack, false otherwise.

**QUEUE:**

**Member functions:**
Write implementation for the following methods as described here.

- **Queue() :**
  Simple base constructor.

- **Queue(const Queue<T>& otherQueue):**
  Copy all elements of otherQueue into the new queue.

- **void enqueue(T item):**
  Add item to the end of queue.

- **T front():**
  Returns element at the front of queue without deleting it.

- **T dequeue():**
  Returns and deletes element at front of queue.

- **int length():**
  Returns count of number of elements in the queue.

- **bool isEmpty():**
  Return true if there is no element in the queue, false otherwise.

## Part 3 : Amusement Park (a and b are necessary, c is bonus)

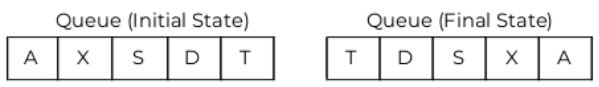
You have finally completed parts 1 and 2. You decide to sleep. When you wake up you are in a strange world. You are confused about where you are. You ask people around you about the strange-looking world. Someone tells you this is the planet Programatica. Where everything goes in terms of your code. You are confused about what to do. You see a lot of people clustered around someone everyone is saying, “The wise sage is here, ask him about your problems”. You decide to go there as well. Interestingly that, wise sage is no other than Farrukh. You ask Farrukh what is this place. What do I do? How do I get out from here? Farrukh asks you to calm down and be positive. You are very perplexed and confused so Farrukh says:

**“If you look for the light you will often find it but if you look for the dark that is all you will ever see”**

But you are still confused that, that how you will get back from here. You are worried about the final exams, you have to get back home. You ask Farrukh how will you get back to home. He points you to the amusement park, and you see people are fighting in the amusement park for some dispute. And Farrukh says:

**“Sometimes the best way to help yourself is to help someone else.”**

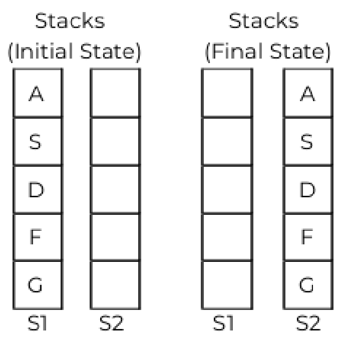
(a) You go into the park, and you ask people why are they fighting. You come to know that they are in a queue for prize distribution. But accidentally, the queue is reversed. The place is so tidy and there is also not enough time to form the queue again. u have also realized that programming is the super power on this planet. Luckily, you have mastery of queues, and you see a little empty space enough for one more queue. You can use that lane , but at the end, people should be standing in the same lane in reverse order.



Write a code to solve the above problem with the followings conditions fulfilled:

- You cannot hard code this part i.e your code should work for any number of people.
- Implement the two lanes as queues.
- You are not allowed to use any other data structure except for the given two queues.
- You are strictly not allowed to even use any variables.
- You are only allowed to use the member functions of the Queue you defined in part 2.

(b) You solved this problem, and the store owner suddenly called you. He is impressed by your skill. He tells you that they accidentally stacked the gifts in the reverse order. They are cumbersome, and only one gift can be lifted at a time, and they are also low on space. But luckily, there is enough space next to a stack of gifts (another table) to make a new stack. You are required to change the stacks such as.



Write a code to solve the above problem with the following conditions fulfilled:

- You cannot hard code this part i.e your code should work for any number of gifts.
- Implement the two tables as two stacks and a helper from the management as a variable.
- You are not allowed to use any other data structures except for the given two stacks and you cannot use more than one variable.
- You are only allowed to use member functions of Stack you defined in part 2.

(c) The management started to feel that they needed to uplift the spirit of the crowd somehow after causing a lot of mismanagement. They decided to invite all the customers to a burger joint for FREE BURGERS!

How the burger joint operates:

The burger joint asks each of its customers to construct the burger of his/her choice - choose one topping from the following list:
1. tbun - top bun
2. ketchup
3. mayo
4. lettuce
5. onions
6. pickles
7. patty
8. mustard
9. chipotle
10. bbun - bottom bun

The list above also describes the order in which the toppings will be assembled by the staff of the burger joint ( topping priority ).

Your job is to assemble the burgers and hand them out to the customers.

How the assembly line operates:

You will be standing on the assembly line, here is what you have to do:

1. You will receive the customer ID of the customer whose order is to be assembled next and the toppings in an unsorted order.
2. You have to sort the toppings according to their priority (refer to the topping list above).
3. Hand out the order to the customer.

Here's how you have to implement the above three steps:

**Step 1:**

Load all of the toppings in the topping_priority from the text file named "topping_priority.txt" (the toppings are in sorted order in this file). Now load the customers and their respected unsorted toppings from the text file named "assembly.txt" in the queue named customers.

**Step 2:**

Sort the order of each customer in the customers queue. Here you have to make use of the **assemble(Stack<string>&unsorted_toppings)** function. But you have a couple of restrictions :

You have two stacks:

1. **unsorted_toppings** (input stack)
2. **temp_stack** (local)

You have one variable:

1. **temp** (local)

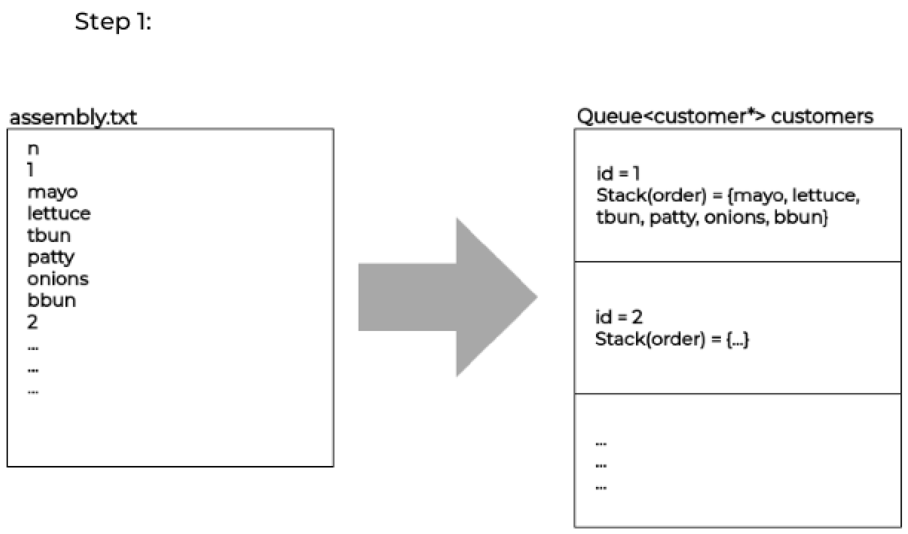
DO NOT CREATE ANYMORE VARIABLE OR DATA STRUCTURES IN THIS FUNCTION. YOU CANNOT USE ANY PREDEFINED LIBRARY OR HELPER FUNCTION

Use the variable and data structures mentioned above to sort the unsorted_toppings stack. You need to sort it according to toppings_priority .

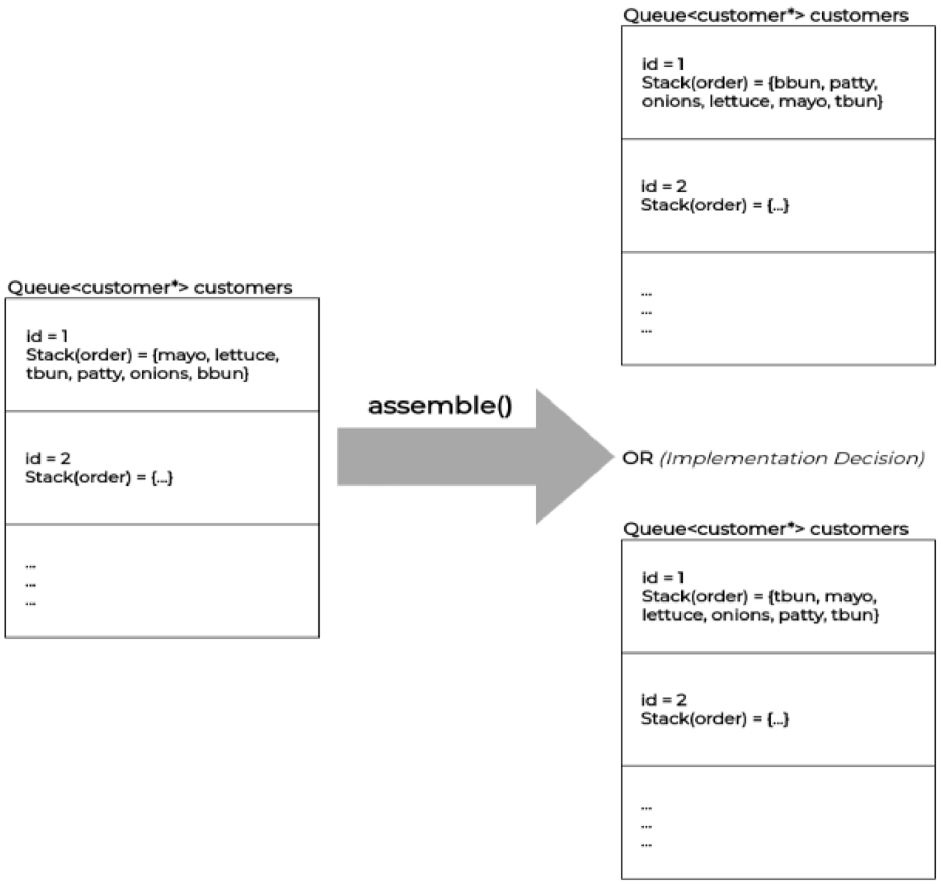
**Step 3:**

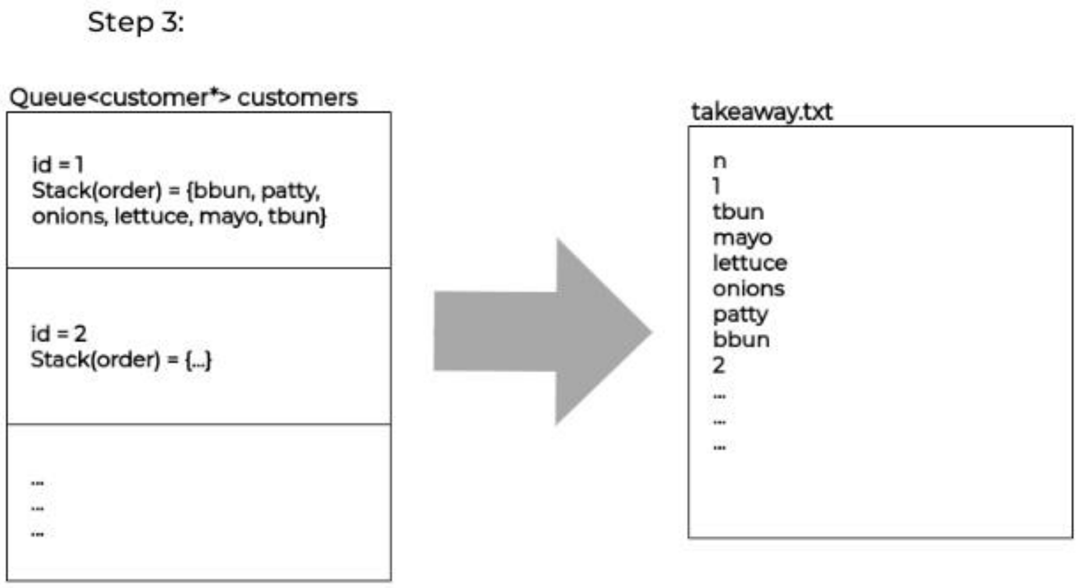
Use the **generateOutput()** function to output the orders of customers in a text file. Name the text file "takeaway.txt".

Here is an example run-through for the above three steps:



Step 2:


Queue<customer*> customers
id = 1
Stack(order) = {mayo, lettuce, tbun, patty, onions, bbun}
id = 2
Stack(order) = {...}
...
...
...
assemble()
Queue<customer*> customers
id = 1
Stack(order) = {bbun, patty, onions, lettuce, mayo, tbun}
id = 2
Stack(order) = {...}
...
...
...
OR *(Implementation Decision)*
Queue<customer*> customers
id = 1
Stack(order) = {tbun, mayo, lettuce, onions, patty, tbun}
id = 2
Stack(order) = {...}
...
...
...



**Running and Testing Code:**
To run the code, you have to use g++ compiler. You have 4 test files at your disposal:

1. test1.cpp: Test for LinkedList.cpp.
2. test2.cpp: Test for stack.cpp and queue.cpp.
3. test3.cpp: Test for part (a) Reversing Queue and part (b) Shifting Stack.
4. test4.cpp: Test for part (c) Sorting Stack.

Write your implementations in the cpp files and then run the test 1 and test 2 using the following commands:

Compile: g++ test1.cpp -std=c++11
Run: ./a.out

To run test 3, you have to give the number of winners/ gifts as an argument. For example, the following commands will check your implementation for 100 winners/ gifts:

Compile: g++ test3.cpp -std=c++11
Run: ./a.out 100

To run test 4, you have 3 assembly files at your disposal:

1. assembly10.txt: contains 10 scrambled burger toppings.
2. assembly50.txt: contains 50 scrambled burger toppings.

Suppose you want to run your test for 10 toppings, then copy the contents of “assembly10.txt” into “assembly.txt” and then run the test using the following commands:

Compile: g++ test4.cpp -std=c++11
Run: ./a.out 10

Similarly to run 50 toppings, copy contents of “assembly50.txt” into “assembly.txt” and run the test using the following command:

Run: ./a.out 50

You code needs to run for both files to earn the bonus marks.

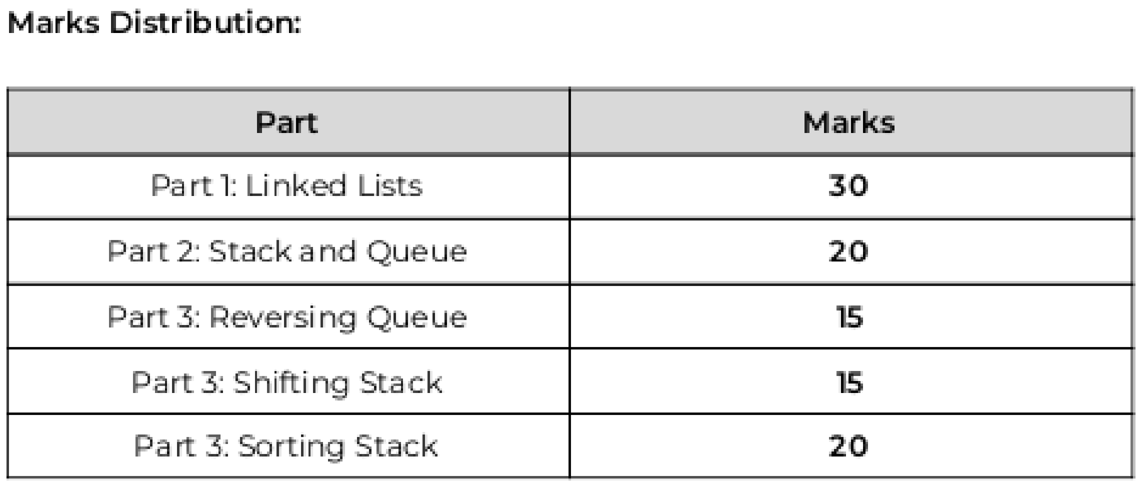
**Submission Guide:**

Zip the complete folder and use the following naming convention

**PA1_<rollnumber>.zip**

For example, if your roll number is 23100250 then your zip file name should:

**PA1_23100250.zip**



**Marks Distribution:**

| Part | Marks |
|---|---|
| Part 1: Linked Lists | **30** |
| Part 2: Stack and Queue | **20** |
| Part 3: Reversing Queue | **15** |
| Part 3: Shifting Stack | **15** |
| Part 3: Sorting Stack | **20** |

You have solved the problem of and helped so many people. As soon as you get out, you see Sir Waqar standing there in front of you. He says, he is proud of you for all the progress you all have made in the course. This was not easy, and you all did an amazing job. He is proud of all of you and he knows you all will do great things in your life. You ask him how to get out from here and go back to your home. Sir tells you that all you have to do is to believe in yourself.

جب اس انگارۂ خاکی میں ہوتا ہے یقیں پیدا
تو کر لیتا ہے یہ بال و پرِ رُوح الامیں پیدا

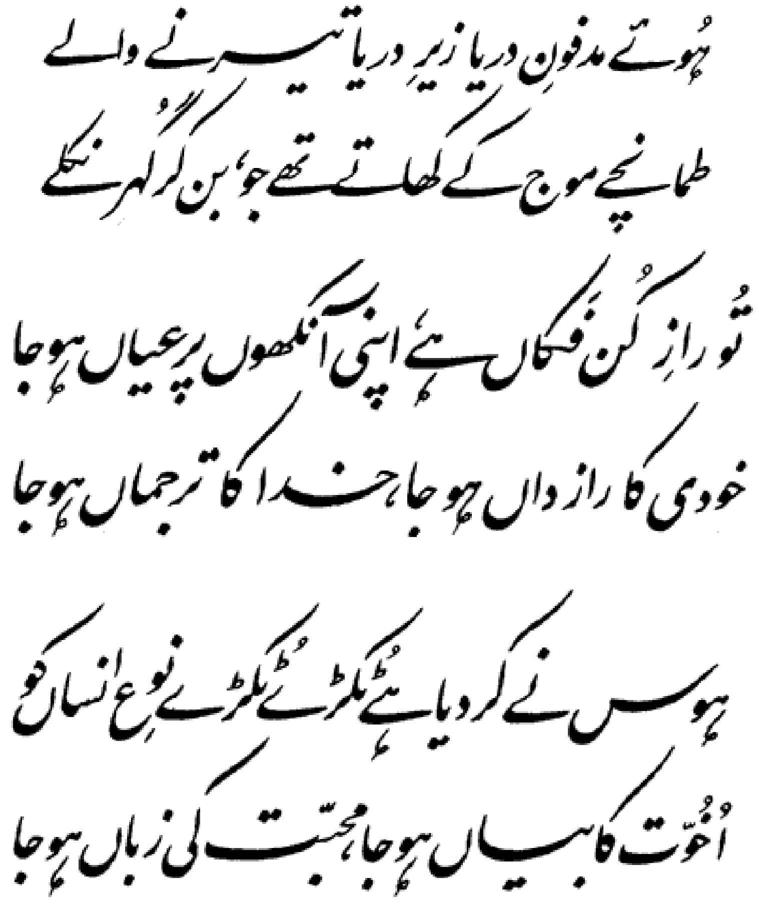
ترے علم و محبت کی نہیں ہے انتہا کوئی
نہیں ہے تجھ سے بڑھ کر سازِ فطرت میں نَوا کوئی

خدائے لم یزل کا دستِ قُدرت تُو، زباں تُو ہے
یقیں پیدا کر اے غافل کہ مغلوبِ گُماں تُو ہے

یقیں محکم، عمل پیہم، محبت فاتحِ عالم
جہادِ زندگانی میں ہیں یہ مردوں کی شمشیریں

**Note from Hassnain**

Good Job everyone. You all did a wonderful job in the course. You all have come so far compared to where you were at the start of the semester. This course was indeed difficult and you all did a wonderful job. Do not compare your progress with others, we all never have the same starting point. Your competition is with you and I am proud of you for what you all achieved in this short period of time. Insha'Allah you all will do great things in life. Ziada tension nahi laini, sab hou jata, the important thing is to never give up. I know sometimes everything looks really overwhelming and daunting, and you start to doubt yourself, in such cases you all should know that, I believe in you, and you all are capable of achieving great things. Best of luck everyone ^-^.  (I will be doing some more Iqbal promotion)



ہوئے مدفونِ دریا زیرِ دریا تیرنے والے
طمانچے موج کے کھاتے تھے جو، بن کر گہر نکلے

تو رازِ کُن فکاں ہے، اپنی آنکھوں پر عیاں ہو جا
خودی کا راز داں ہو جا، خدا کا ترجماں ہو جا

ہوس نے کر دیا ہے ٹکڑے ٹکڑے نوعِ انساں کو
اُخوّت کا بیاں ہو جا، محبّت کی زباں ہو جا

تو رہ نوردِ شوق ہے، منزل نہ کر قبول
لیلیٰ بھی ہم نشیں ہو تو محمل نہ کر قبول

(I so want to go on ...... But I think I have made my point with these. I hope someone understands this.)