COL100
Introduction to Computer Science
Assignment 6

1 Instructions

Dear Students, Assignment 6 has been released on Gradescope. Below, you'll find important details regarding the assignment, submission guidelines, and support channels for any queries.

2 Important Change

You have to write and run the code in your local machine. Assignment on **replit** won't be created for this assignment. You can take help of the TAs during the lab sessions in case you are facing issues in setting up the C environment.

2.1 Assignment Details:

Weightage: 8% of the overall grade
Submission Platform: Gradescope (https://www.gradescope.com/courses/561465)
Deadline: 30th October 2023, 11:55pm

2.2 Support and Queries:

1. **All queries about this assignment should be posted only in the relevant Piazza board.** No other queries shall be entertained. Please read through the previously posted queries before posting.

2. **Before using any function from any existing library or using any algorithm not covered in the class, get a confirmation from the instructors by posting on Piazza.**

Additional Resources:

For a refresher on the concepts related to the assignment, you can refer to the course lecture notes, slides, and relevant sections from the textbook. Assignment Grading: Your assignments will be graded on correctness, efficiency of your code, proper usage of comments, and adherence to the submission guidelines. We wish you the best of luck with the assignment.

3 Doubly Linked List

A doubly linked list is a type of linked list where each node contains two pointers, one pointing to the next node (as in a singly linked list), and another pointing to the previous node. This bidirectional connectivity makes it possible to traverse the list in both forward and backward directions.
In a doubly linked list, the first node is called the "head", and the last node is called the "tail". The head's previous pointer is always NULL, and the tail's next pointer is also NULL.
Here's a simple C program to demonstrate the creation and insertion of a doubly linked list. Define the structure of node of a doubly linked list.

```
Struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
}
```

Listing 1: Doubly LL Node structure

4 Student Record Management

At the prestigious Indian Institute of Technology, Delhi (IITD), efficient management of student records was a top priority. The administration faced the challenge of maintaining an up-to-date database of students, complete with academic details, extracurricular activities, and personal information. To address this challenge, they developed an advanced Student Records Management System.
In this problem, you have to write programs to manage student records using C structures and doubly linked lists. The student records are stored in a **Doubly Linked List** with each node representing a student record. Each student node has the following **attributes**:

1. Student_ID (integer)
2. GPA (float)

You need to implement the following tasks by creating function for each task:

4.1 Creating the student record :

At the start of each academic year, new students are admitted. To create a digital record for each student, the system allowed administrative staff to input their roll numbers and academic records. As records were added, the system dynamically formed a doubly linked list, where each node represented a student.
The node structure is as follows:

```
Struct Node {
    int id;
    float gpa;
    struct Node* next;
    struct Node* prev;
}
```

Listing 2: Student Node structure

Append each node to the end of the doubly linked list. Complete the `void main()` function in **main.c** file to take n records as input and create a doubly linked list with the above node structure.

4.1.1 Input format :

```
6
1 8.7
2 6.3
3 7.1
4 8.8
1 8.7
5 8.9
```

Listing 3: Student record input

First line of input contains size of linked list n and next n lines contains 2 space separated values- id and gpa.

4.2 Delete the oldest student record :

As the academic year progressed, students graduated or left the institution for various reasons. The system needed a mechanism to remove graduating students. It was designed to delete the head node (representing the graduating student) and return the new head. Complete the following function:

```
struct Node* deleteHead(struct Node* head)
```

Here head points to the first element of the list. Return the first element of the linked list after removing the head.

4.2.1 Input :

```
deleteHead(struct Node* head)
```

4.2.2 Output List:

```
2 6.3
3 7.1
4 8.8
1 8.7
5 8.9
```

4.3 Removing duplicate records :

Removing duplicate records of a particular student from a student records database is crucial to maintaining data accuracy and integrity. While cleaning up the records, the institute encountered some students with multiple entries. Ideally, the institute wants to keep only the oldest record of the student and delete any other occurrence.
Complete the following function:

```
struct Node* removeDuplicates(struct Node* head, int id)
```

Here head points to the first element of the list and id is the unique id of a student. Return the first element of the linked list after keeping only the oldest record of the student.

4.3.1 Input :

```
removeDuplicates(struct Node* head, 1)
```

4.3.2 Output List:

```
1 8.7
2 6.3
3 7.1
4 8.8
5 8.9
```

4.4 Reversing the records :

At the end of each academic year, the administration needed to review student records, often starting with the most recently admitted students. By reversing the linked list, they could easily access the details of the latest cohort. The system offered a "Reverse" function, which, when used, returned the new head of the list.
Complete the following function:

```
struct Node* reverse(struct Node* head)
```

Here head points to the first element of the list. Return the first element of the linked list after reversing the whole list.

4.4.1 Input :

```
reverse(struct Node* head)
```

4.4.2 Output List:

```
5 8.9
1 8.7
4 8.8
3 7.1
2 6.3
1 8.7
```

4.5 Rotating the records :

As new batches of students joined the institution, administrators wanted to view records based on the batch year. To simulate this, the system enabled them to rotate the linked list by a specified number of places to the right, effectively grouping records by academic year.
Complete the following function:

```
struct Node* rotateByKplaces(struct Node* head, int k)
```

Here head points to the first element of the list and k is the number of places the list has to be rotated to the right. Return the first element of the linked list after rotating the list.

4.5.1 Input :

```
rotateByKplaces(struct Node* head, 2)
```

4.5.2 Output List:

```
1 8.7
5 8.9
1 8.7
2 6.3
3 7.1
4 8.8
```

4.6 Creating a sorted list of the records :

Every year, when it was time to generate student merit scholarships, the administration needed the records to be sorted by GPA. The admin wants to create a **new doubly linked list** sorted by GPA.
Complete the following function:

```
struct Node* createSortedList(struct Node* head)
```

Here head points to the first element of the list. Return the first element of a new doubly linked list in which the nodes are sorted according to GPA in descending order.

4.6.1 Input :

```
createSortedList(struct Node* head)
```

4.6.2 Output List:

```
5 8.9
4 8.8
1 8.7
1 8.7
3 7.1
2 6.3
```