Assignment no. 1

Title: Linked List

Part 1 and Part 2 are mandatory.

Part 3 is bonus. You can expect part 3 type questions in the end semester exam.

- Part 1
    1. Create a new C file and name it "linked_list.c".
        a. Define a structure called "Node" with the following members:
            - An integer variable called "data" to store the data of the node.
            - A pointer to the next node called "next" to link the nodes together.
        b. Declare a global pointer variable called "head" to keep track of the head of the linked list. Initialize it to NULL.
        c. Implement the following functions:
            - void insertNode(int value): This function should insert a new node with the given value at the end of the linked list.
            - void deleteNode(int value): This function should delete the node with the given value from the linked list.
            - void displayList(): This function should display the elements of the linked list.
            - void reverseList(): This function should reverse the order of the nodes in the linked list.
            - int searchNode(int value): This function should search for a node with the given value in the linked list and return its position (index) if found, or -1 if not found.
        d. In the main() function, create a menu-driven program to interact with the linked list. The menu should provide the following options:
            - Insert a node at the end of the list
            - Delete a node by value
            - Display the list
            - Reverse the list
            - Search for a node
            - Exit the program
        e. Ensure that your code follows the standards coding practices, which typically include proper indentation, meaningful variable names, comments, and code modularity.
        f. Test your program with various inputs and validate the correctness of each operation.
        g. Document your code and include appropriate comments to explain the purpose of each function and significant sections of code.
        h. Optimize your code for efficiency and handle edge cases such as an empty list or invalid input.
        i. *Submit your code along with a comprehensive report documenting your implementation, including any challenges faced, how you addressed them, and an analysis of the time and space complexity of the different operations.*
        j. **[Bonus]**: Move the global pointer named head (point 1b) inside the main function. Now change the function prototypes (point 1c) so that it can be passed as a parameter and works the same. **Note: In practice, usage of global variables is a bad idea.**

- Part 2
    1. Create a new C file and name it "music_playlist.c".
        a. Define a structure called "Song" with the following members:
            - A character array called "title" to store the song title.
            - A character array called "artist" to store the artist name.
            - A pointer to the next song called "next" to link the songs together.
        b. Declare a pointer (*preferably* not global) variable called "head" to keep track of the head of the linked list. Initialize it to NULL.
        c. Implement the following functions for music playlist management:
            - void addSong(char title[], char artist[]): This function should add a new song to the linked list.
            - void deleteSong(char title[]): This function should delete the song with the given title from the linked list.
            - void displayPlaylist(): This function should display the details of all the songs in the linked list.
            - void playPlaylist(): This function should simulate playing the songs in the playlist sequentially.
            - int searchSong(char title[]): This function should search for a song with the given title in the linked list and return its position (index) if found, or -1 if not found.
        d. In the main() function, create a menu-driven program to interact with the music playlist. The menu should provide the following options:
            - Add a song to the playlist
            - Delete a song from the playlist
            - Display the playlist
            - Play the playlist
            - Search for a song
            - Exit the program
        e. Implement additional functions as needed to handle the menu options and perform the required operations on the linked list.
        f. Test your program with various inputs to validate the correctness of each operation.
        g. Document your code and include appropriate comments to explain the purpose of each function and significant sections of code.
        h. Optimize your code for efficiency and handle edge cases such as an empty list or invalid input.
        i. Submit your code along with a comprehensive report documenting your implementation, including any challenges faced, how you addressed them, and an analysis of the time and space complexity of the different operations.
        j. Note: You can add additional features to the music playlist application, such as shuffling the playlist, sorting the songs based on artist or duration, or adding a feature to skip songs, to enhance the assignment based on your requirements.
    2. Create a new C file and name it "task_manager.c".
        a. Define a structure called "Task" with the following members:
            - An integer variable called "taskId" to store the task ID.
            - A character array called "description" to store the task description.
            - An integer variable called "priority" to store the priority of the task.
            - A pointer to the next task called "next" to link the tasks together.
        b. Declare a pointer (*preferably* not global) variable called "head" to keep track of the head of the linked list. Initialize it to NULL.
        c. Implement the following functions for task management:

- void addTask(int taskId, char description[], int priority): This function should add a new task to the linked list.
- void deleteTask(int taskId): This function should delete the task with the given task ID from the linked list.
- void displayTasks(): This function should display the details of all the tasks in the linked list.
- void prioritizeTasks(): This function should prioritize the tasks based on their priority.
- int searchTask(int taskId): This function should search for a task with the given task ID in the linked list and return its position (index) if found, or -1 if not found.

d. In the main() function, create a menu-driven program to interact with the task manager. The menu should provide the following options:
   - Add a task
   - Delete a task
   - Display all tasks
   - Prioritize tasks
   - Search for a task
   - Exit the program

e. Implement additional functions as needed to handle the menu options and perform the required operations on the linked list.

f. Test your program with various inputs to validate the correctness of each operation.

g. Document your code and include appropriate comments to explain the purpose of each function and significant sections of code.

h. Optimize your code for efficiency and handle edge cases such as an empty list or invalid input.

i. Submit your code along with a comprehensive report documenting your implementation, including any challenges faced, how you addressed them, and an analysis of the time and space complexity of the different operations.

j. Note: You can add additional features to the task manager application, such as sorting the tasks based on priority or due dates, to enhance the assignment based on your requirements.

3. Create a new C file and name it "student_record.c".
   a. Define a structure called "Student" with the following members:
      - A character array called "name" to store the student's name.
      - An integer variable called "rollNumber" to store the student's roll number.
      - A floating-point variable called "marks" to store the student's marks.
      - A pointer to the next student record called "next" to link the records together.

   b. Declare a global pointer variable called "head" to keep track of the head of the linked list. Initialize it to NULL.

   c. Implement the following functions for student record management:
      - void addStudent(char name[], int rollNumber, float marks): This function should add a new student record to the linked list.
      - void deleteStudent(int rollNumber): This function should delete the student record with the given roll number from the linked list.
      - void displayStudents(): This function should display the details of all the students in the linked list.
      - int searchStudent(int rollNumber): This function should search for a student record with the given roll number in the linked list and return its position (index) if found, or -1 if not found.

   d. In the main() function, create a menu-driven program to interact with the student record management system. The menu should provide the following options:

- Add a student record
- Delete a student record
- Display all student records
- Search for a student record
- Exit the program

e. Implement additional functions as needed to handle the menu options and perform the required operations on the linked list.
f. Test your program with various inputs to validate the correctness of each operation.
g. Document your code and include appropriate comments to explain the purpose of each function and significant sections of code.
h. Optimize your code for efficiency and handle edge cases such as an empty list or invalid input.
i. Submit your code along with a comprehensive report documenting your implementation, including any challenges faced, how you addressed them, and an analysis of the time and space complexity of the different operations.
j. Note: You can add additional features to the student record management system, such as sorting the records based on roll numbers or calculating average marks, to enhance the assignment based on your requirements.

- Part 3
    1. Implement a linked list data structure. Include methods to insert a node at the beginning, at the end, and at a specific position in the list. Test your implementation by creating a linked list and performing various operations on it.
    2. Write a function to reverse a linked list in-place. Provide both iterative and recursive implementations for this task.
    3. Implement a function to find the middle node of a linked list. If the list has an even number of nodes, return the second middle node.
    4. Given two sorted linked lists, write a function to merge them into a single sorted linked list. Preserve the original order of the nodes in each list.
    5. Write a function to detect if a linked list contains a cycle. If it does, determine the node at which the cycle starts.
    6. Implement a function to remove duplicates from a sorted linked list. The resulting list should only contain unique elements.
    7. Implement a function to check if a linked list is a palindrome. Consider both a simple implementation and one that uses a stack for efficient comparison.
    8. Write a function to remove the nth node from the end of a linked list. Assume n is always valid ($1 \leq n \leq$ list length).
    9. Implement a function to split a linked list into two separate lists, where one list contains all the even-indexed elements and the other contains all the odd-indexed elements.