San Diego State University - CS 210
Programming Assignment 1: Space Mission Tracker using a Doubly Linked List
Total Points: 100

Mission Briefing

The year is 2087. Humanity has established deep-space exploration missions to uncover the secrets of the galaxy. You are a mission control engineer assigned to track Voyager-210, a spacecraft exploring the far reaches of space. Your task is to maintain a detailed log of Voyager-210's journey, ensuring that all celestial waypoints are recorded, modified, and removed efficiently. Using a **doubly linked list**, you will manage the spacecraft's route, allowing mission control to traverse its travel history in both forward and backward directions. Your implementation will help scientists monitor and analyze Voyager-210's findings in real-time.

Objective

In this assignment, students will implement a **doubly linked list** in C++ to simulate a space mission tracker. The spacecraft will navigate through a sequence of celestial waypoints, with each waypoint represented as a node in the doubly linked list. Students must implement **insertion, deletion, traversal, and retrieval operations** to manage the mission data.

This assignment reinforces concepts of **dynamic memory management, linked list manipulation, and efficient data structure traversal.**

Requirements

Students must implement the following functionalities in the `SpaceRoute` class:

- **Insertion Operations:**
    - Add a waypoint at the beginning of the list.

```cpp
void addWaypointAtBeginning(T& data);
```

    - Add a waypoint at the end of the list.

```
C/C++
void addWaypointAtEnd(T& data);
```

- Insert a waypoint at a specified index.

```
C/C++
void addWaypointAtIndex(int index, T& data);
```

- **Deletion Operations:**
  - Remove the first waypoint from the list.

```
C/C++
void removeWaypointAtBeginning();
```

  - Remove the last waypoint from the list.

```
C/C++
void removeWaypointAtEnd();
```

  - Remove a waypoint at a specified index.

```
C/C++
void removeWaypointAtIndex(int index);
```

- **Traversal Operations:**
  - Print all waypoints from the first to the last.

```
C/C++
void traverseForward();
```

  - Print all waypoints in reverse order from last to first.

```
C/C++
void traverseBackward();
```

- **Modification and Retrieval Operations:**
  - Retrieve a waypoint at a specified index.

```
C/C++
Node<T>* getWaypoint(int index);
```

  - Modify the details of a waypoint at a specified index.

```
C/C++
void setWaypoint(int index, T& data);
```

- **Printing:**
  - Implement a print function that allows displaying the entire route.

```
C/C++
void print();
```

All operations must ensure proper memory management using dynamic allocation and deallocation.

Testing with main function

```
C/C++

int main() {

    string mars = "Mars";

    string jupiter = "Jupiter";

    string saturn = "Saturn";
    string earth = "Earth";

    string venus = "Venus";


    SpaceRoute<string> voyagerRoute;


    voyagerRoute.addWaypointAtEnd(mars);

    voyagerRoute.addWaypointAtEnd(jupiter);

    voyagerRoute.addWaypointAtEnd(saturn);

    voyagerRoute.addWaypointAtBeginning(earth);

    voyagerRoute.addWaypointAtIndex(2, venus);


    cout << "Voyager Route (Forward):\n";

    voyagerRoute.traverseForward();


    cout << "\nVoyager Route (Backward):\n";

    voyagerRoute.traverseBackward();


    cout << "\nPrinting Route: \n";

    voyagerRoute.print();


    voyagerRoute.removeWaypointAtIndex(2);
    cout << "\nAfter Removing Venus: \n";

    voyagerRoute.print();



    return 0;

}
```

FAQ Section

1. **Does indexing start at 0?**
   Yes, the first waypoint is at **index 0**.
2. **Can I modify the function names or method signatures?**
   No, students **must not** modify the existing method signatures. Additional helper functions can be added if necessary.
3. **What happens if my code does not run?**
   Non-functional code will receive a **0**.
4. **Is a README file required?**
   Yes, failure to submit a README will result in a **10-point deduction**.
5. **Are late submissions accepted?**
   Late assignments incur a **25% deduction per day** with **no extensions**.
6. **Do I need to handle edge cases for all methods?**
   Yes, all methods must properly handle edge cases such as an empty list, accessing invalid indexes, and adding or removing nodes from different positions in the list. Failing to handle edge cases may result in deductions.

Grading Breakdown (100 Points)

- **Insertion Methods (30 points total)**
  - Beginning (10 points)
  - End (10 points)
  - Index (10 points)
- **Deletion Methods (30 points total)**
  - Beginning (10 points)
  - End (10 points)
  - Index (10 points)

- **Traversal Methods (20 points total)**
  - Forward (10 points)
  - Backward (10 points)
- **Getter and Setter Methods (10 points total)**
  - Get (5 points)
  - Set (5 points)
- **README Section (10 points total)**
  - Explanation of each method (5 points)
  - Justification of Big-O complexity (5 points)

Final Notes

This assignment is designed to strengthen understanding of **doubly linked lists** and their applications in **data management and traversal**. Ensure your implementation is functional and follows best coding practices.

Good luck and happy coding.