Circular Linked List Implementation Example

Instructor: Bostan Khan

Circular Linked List for Restaurant Table Management

- Scenario: Managing a restaurant with multiple tables of varying capacities.
- Objective: Efficiently assign tables to customers and track table occupancy.
- Implementation Overview:
  - Utilizes a circular linked list data structure.
  - Nodes represent tables, forming a circular queue.

Circular Linked List for Restaurant Table Management

- Features functionalities for table management:
  - Add tables
  - Seat customers
  - Mark tables as unoccupied
  - Remove tables
  - Display current queue status

Circular Linked List for Restaurant Table Management

- Implementation Highlights:
  - Circular nature optimizes table assignment.
  - Dynamic addition/removal of tables reflects changing restaurant dynamics.
  - Accurate tracking of table occupancy ensures efficient customer seating.

Circular Linked List for Restaurant Table Management

- Key Takeaways:
  - Demonstrates practical application of fundamental data structures.
  - Enables efficient solutions for real-world scenarios.
  - Essential for developing effective restaurant management systems.
- Let's Explore the Code!

The **Table** Class: Attibutes

```
class Table {
private:
    int tableNumber;
    int capacity;
    bool occupied;

public:
    Table(int _tableNumber, int _capacity) : tableNumber(_tableNumber),
capacity(_capacity), occupied(false) {}

    int getTableNumber() const { return tableNumber; }
    int getCapacity() const { return capacity; }
    bool isOccupied() const { return occupied; }
```

The **Table** Class: functions

```
// Function to seat customers at the table
    bool seatCustomers(int numCustomers) {
        if (!occupied && numCustomers <= capacity) {
            occupied = true;
            return true;
        }
        return false;
    }

    // Function to mark the table as unoccupied
    void makeUnoccupied() {
        occupied = false;
    }
};
```

The **Node** Class

```
// Node class for the circular linked list
class Node {
public:
    Table table;
    Node* next;

    Node(Table _table) : table(_table), next(nullptr) {}
};
```

The **CircularList** Class: Attributes and constructor

```
class CircularList {
private:
    Node* front;

public:
    CircularList() : front(nullptr) {}
```

The **CircularList** Class: addTable()

```
    void addTable(Table table) {
        Node* newNode = new Node(table);
        if (front == nullptr) {
            front = newNode;
            front->next = front;
        } else {
            newNode->next = front->next;
            front->next = newNode;
            front = newNode;
        }
    }
```

The **CircularList** Class: seatCustomers()

```
// Function to seat customers at the next available table
    void seatCustomers(int numCustomers) {
        if (front != nullptr) {
            Node* current = front->next;
            Node* bestFit = nullptr;
            int minAvailableSeats = INT_MAX;

            // Finding the table with the closest number of available seats
            do {
                if (!current->table.isOccupied() && current->table.getCapacity() >= numCustomers &&
                 current->table.getCapacity() < minAvailableSeats) {
                    minAvailableSeats = current->table.getCapacity();
                    bestFit = current;
                }
                current = current->next;
            } while (current != front->next);

            if (bestFit != nullptr) {
                cout << "Seating " << numCustomers << " customers at Table " <<
                                                        bestFit->table.getTableNumber() << endl;
                bestFit->table.seatCustomers(numCustomers);
            } else {
                cout << "No tables available with sufficient capacity." << endl;
            }
        } else {
            cout << "No tables available." << endl;
```

The **CircularList** Class: seatCustomers()

```
// Function to seat customers at the next available table
    void seatCustomers(int numCustomers) {
        if (front != nullptr) {
            Node* current = front->next;
            Node* bestFit = nullptr;
            int minAvailableSeats = INT_MAX;

            // Finding the table with the closest number of available seats
            do {
                if (!current->table.isOccupied() && current->table.getCapacity() >= numCustomers &&
                 current->table.getCapacity() < minAvailableSeats) {
                    minAvailableSeats = current->table.getCapacity();
                    bestFit = current;
                }
                current = current->next;
            } while (current != front->next);

            if (bestFit != nullptr) {
                cout << "Seating " << numCustomers << " customers at Table " <<
                                                        bestFit->table.getTableNumber() << endl;
                bestFit->table.seatCustomers(numCustomers);
            } else {
                cout << "No tables available with sufficient capacity." << endl;
            }
        } else {
            cout << "No tables available." << endl;
```

- current table/node pointer for traversing the list
- bestFit table/node pointer for the optimum table selection
- minAvailableSeats helper variable for keeping track of minimum capacity available in the unoccupied tables.

The **CircularList** Class: seatCustomers()

```
// Function to seat customers at the next available table
    void seatCustomers(int numCustomers) {
        if (front != nullptr) {
            Node* current = front->next;
            Node* bestFit = nullptr;
            int minAvailableSeats = INT_MAX;

            // Finding the table with the closest number of available seats
            do {
                if (!current->table.isOccupied() && current->table.getCapacity() >= numCustomers &&
                current->table.getCapacity() < minAvailableSeats) {
                    minAvailableSeats = current->table.getCapacity();
                    bestFit = current;
                }
                current = current->next;
            } while (current != front->next);

            if (bestFit != nullptr) {
                cout << "Seating " << numCustomers << " customers at Table " <<
                                                        bestFit->table.getTableNumber() << endl;
                bestFit->table.seatCustomers(numCustomers);
            } else {
                cout << "No tables available with sufficient capacity." << endl;
            }
        } else {
            cout << "No tables available." << endl;
```

Find the table with the minimum capacity that satisfies the customer requirement.

The **CircularList** Class: makeTableUnoccupied()

```
    // Function to mark a table as unoccupied when customers leave
    void makeTableUnoccupied(int tableNumber) {
        if (front != nullptr) {
            Node* current = front->next;
            do {
                if (current->table.getTableNumber() == tableNumber) {
                    current->table.makeUnoccupied();
                    cout << "Table " << tableNumber << " is now unoccupied." <<
endl;
                    return;
                }
                current = current->next;
            } while (current != front->next);
            cout << "Table " << tableNumber << " not found in the queue." <<
endl;
        } else {
            cout << "No tables available." << endl;
        }
    }
```

The **CircularList** Class: removeTable()

```
    // Function to remove a table from the queue
    void removeTable(int tableNumber) {
        if (front != nullptr) {
            Node* current = front->next;
            Node* prev = front;
            do {
                if (current->table.getTableNumber() == tableNumber) {
                    prev->next = current->next;
                    delete current;
                    return;
                }
                prev = current;
                current = current->next;
            } while (current != front->next);
            cout << "Table " << tableNumber << " not found in the queue." << endl;
        } else {
            cout << "No tables available." << endl;
        }
    }
```

The **CircularList** Class: displayQueue()

```
    // Function to display the current queue
    void displayQueue() {
        if (front != nullptr) {
            Node* current = front->next;
            do {
                cout << "Table " << current->table.getTableNumber() << " - Capacity:
" << current->table.getCapacity();
                if (current->table.isOccupied()) {
                    cout << " (Occupied)";
                } else {
                    cout << " (Unoccupied)";
                }
                cout << endl;
                current = current->next;
            } while (current != front->next);
        } else {
            cout << "No tables available." << endl;
        }
    }
};
```

The **main()** function (1/2)

```
int main() {
    CircularList queue;

    // Adding some tables to the queue
    queue.addTable(Table(1, 4));
    queue.addTable(Table(2, 6));
    queue.addTable(Table(3, 2));

    // Displaying the current queue
    cout << "Current Table Queue:" << endl;
    queue.displayQueue();

    // Seating customers at the next available table
    cout << endl << "Seating customers:" << endl;
    queue.seatCustomers(5);
    queue.seatCustomers(3);

    // Displaying the updated queue
    cout << endl << "Updated Table Queue:" << endl;
    queue.displayQueue();
```

The **main()** function (2/2)

```
    // Making a table unoccupied
    cout << endl << "Making Table 2 unoccupied:" << endl;
    queue.makeTableUnoccupied(2);

    // Displaying the updated queue
    cout << endl << "Updated Table Queue:" << endl;
    queue.displayQueue();

    // Removing a table from the queue
    cout << endl << "Removing Table 2:" << endl;
    queue.removeTable(2);

    // Displaying the updated queue
    cout << endl << "Updated Table Queue:" << endl;
    queue.displayQueue();

    return 0;
}
```