Evaluation Schema for Each Question

1. **Create a Singly Circular Linked List (3 marks)**
   - Correct initialization of the linked list structure (1 mark)
   - Correct handling of the circular nature (1 mark)
   - Insertion of nodes into the list (1 mark)
2. **Count the Number of Nodes in the Circular Linked List (3 marks)**
   - Correct traversal through the circular linked list (1 mark)
   - Correct node counting logic (1 mark)
   - Correct termination of the traversal (1 mark)
3. **Delete the First Node in the Circular Linked List (4 marks)**
   - Correct identification of the first node (1 mark)
   - Handling circular property of the list during deletion (1 mark)
   - Correct pointer update for the head and tail nodes (1 mark)
   - Proper deletion of the node from memory (1 mark)
4. **Convert the Infix Expression to Prefix and Evaluate It (8 marks)**
   - Correct conversion from infix to prefix (4 marks)
     - Each operator precedence and associativity handled correctly (2 marks)
     - Parentheses handled correctly (2 marks)
   - Correct evaluation of the prefix expression using the given values (4 marks)
     - Substituting correct values (1 mark)
     - Correct intermediate computations (2 marks)
     - Final correct result (1 mark)
5. **Pseudo Code for Majority Element in an Array (5 marks)**
   - Proper loop to count frequency of each element (1 mark)
   - Proper handling of majority condition n/2n/2n/2 (2 marks)
   - Correct majority check and result output (2 marks)
6. **Check if a String is a Palindrome Using Stack Operations (6 marks)**
   - Correct stack implementation (2 marks)
   - Proper string processing and pushing characters onto the stack (2 marks)
   - Correct comparison of characters (2 marks)
7. **Why Convert Infix to Postfix/Prefix (4 marks)**
   - Explanation of ambiguity and operator precedence in infix (2 marks)
   - Explanation of easier evaluation of postfix/prefix using stacks (2 marks)
8. **Limitations of Linear Queue and Solution (6 marks)**
   - Correct explanation of linear queue limitations (2 marks)
     - Queue overflow despite available space (1 mark)
     - Fixed size issue (1 mark)
   - Proposal of circular queue (1 mark)
   - Correct pseudo code for enqueue and dequeue operations (3 marks)
     - Proper pointer updates (2 marks)
     - Handling full and empty conditions (1 mark)

Solutions

1. **Pseudo Code to Create a Singly Circular Linked List**

```
vbnet
Copy code
Function createCircularList():
    Initialize head = null
    For each element in data:
        Create newNode
        If head is null:
            Set head = newNode
            Set newNode.next = head
        Else:
            Traverse to the last node
            Set lastNode.next = newNode
            Set newNode.next = head
```

2. **Pseudo Code to Count Nodes in Circular Linked List**

```
vbnet
Copy code
Function countNodes(head):
    If head is null:
        Return 0
    Initialize count = 1
    Set current = head.next
    While current != head:
        Increment count
        Move current to current.next
    Return count
```

3. **Pseudo Code to Delete the First Node in Circular Linked List**

```
sql
Copy code
Function deleteFirstNode(head):
    If head is null:
        Return null
    If head.next == head:  // Only one node
        Delete head
        Return null
    Set current = head
    Traverse to the last node
    Set lastNode.next = head.next
    Set temp = head
    Set head = head.next
    Delete temp
    Return head
```

4. **Convert Infix to Prefix and Evaluation**
   - Infix: `A^(B-C)*D/(E-(F+G*H))`
   - Prefix Conversion: `^A-*BC/D-E+FGH`
   - Evaluate using A=6, B=7, C=5, D=8, E=14, F=4, G=2, H=3

   Prefix Evaluation Steps:

   4. Evaluate `7 - 5 = 2`
   5. `6^2 = 36`
   6. `2 * 8 = 16`
   7. `2 * 3 = 6`
   8. `4 + 6 = 10`
   9. `14 - 10 = 4`

10. `16 / 4 = 4`
11. `36 * 4 = 144`

5. **Pseudo Code for Majority Element**

```
sql
Copy code
Function findMajorityElement(arr):
    Initialize candidate = -1, count = 0
    For each element in arr:
        If count == 0:
            candidate = element
            count = 1
        Else if element == candidate:
            Increment count
        Else:
            Decrement count
    Verify if candidate appears more than n/2 times
    If true, return candidate
    Else, return "No Majority Element"
```

6. **Pseudo Code for Checking Palindrome Using Stack**

```
vbnet
Copy code
Function isPalindrome(string):
    Initialize stack
    For each char in string:
        Push char onto stack
    For each char in string:
        If char != pop(stack):
            Return False
    Return True
```

7. **Why Convert Infix to Postfix/Prefix**
   - Infix expressions have ambiguous operator precedence.
   - Parentheses and precedence make manual evaluation error-prone.
   - Postfix/prefix expressions avoid ambiguity and can be evaluated easily using a stack, making them preferable for programming.
8. **Limitations of Linear Queue**
   - Queue overflow even when there is available space.
   - Circular queues overcome this limitation by wrapping around.

Circular Queue Pseudo Code

```
sql
Copy code
Function enqueue(queue, element):
    If queue is full:
        Return "Queue is full"
    Else:
        Add element at rear
        Adjust rear pointer (circular wrap-around)

Function dequeue(queue):
    If queue is empty:
        Return "Queue is empty"
    Else:
        Remove element at front
        Adjust front pointer (circular wrap-around)
```

Here are the C language implementations:

1. Create a Singly Circular Linked List

```
c
Copy code
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

struct Node* createCircularList(int arr[], int n) {
    struct Node* head = NULL;
    struct Node* temp = NULL;
    struct Node* last = NULL;

    for (int i = 0; i < n; i++) {
        temp = (struct Node*)malloc(sizeof(struct Node));
        temp->data = arr[i];
        temp->next = head;

        if (head == NULL) {
            head = temp;
            last = head;
        } else {
            last->next = temp;
            last = temp;
        }
    }
    last->next = head; // Make it circular
    return head;
}

void display(struct Node* head) {
    struct Node* temp = head;
    if (head != NULL) {
        do {
            printf("%d ", temp->data);
            temp = temp->next;
        } while (temp != head);
    }
}
```

2. Count the Number of Nodes in the Circular Linked List

```
c
Copy code
int countNodes(struct Node* head) {
    if (head == NULL) return 0;

    int count = 1;
    struct Node* temp = head->next;
    while (temp != head) {
        count++;
        temp = temp->next;
    }

    return count;
}
```

3. Delete the First Node in the Circular Linked List

```
c
Copy code
struct Node* deleteFirstNode(struct Node* head) {
    if (head == NULL) return NULL;

    struct Node* temp = head;
    struct Node* last = head;

    // Find the last node to maintain circular nature
    while (last->next != head) {
        last = last->next;
    }

    if (head->next == head) { // Only one node
        free(head);
        return NULL;
    } else {
        last->next = head->next;
        head = head->next;
        free(temp);
        return head;
    }
}
```

4. Convert Infix Expression to Prefix and Evaluation in C

For simplicity, you can use manual conversion of infix to prefix (`^A-*BC/D-E+FGH`). Below is the evaluation part:

```
c
Copy code
#include <stdio.h>
#include <math.h>

int evaluatePrefix(char* exp) {
    int stack[100];
    int top = -1;
    for (int i = strlen(exp) - 1; i >= 0; i--) {
        if (isdigit(exp[i])) {
            stack[++top] = exp[i] - '0';
        } else {
            int operand1 = stack[top--];
            int operand2 = stack[top--];

            switch (exp[i]) {
                case '+': stack[++top] = operand1 + operand2; break;
                case '-': stack[++top] = operand1 - operand2; break;
                case '*': stack[++top] = operand1 * operand2; break;
                case '/': stack[++top] = operand1 / operand2; break;
                case '^': stack[++top] = pow(operand1, operand2); break;
            }
        }
    }
    return stack[top];
}

int main() {
    char prefix[] = "^A-*BC/D-E+FGH"; // Converted manually
    int result = evaluatePrefix(prefix);
    printf("Result of prefix evaluation: %d\n", result);
    return 0;
}
```

5. Find the Majority Element in an Array

```
c
Copy code
#include <stdio.h>

int findMajorityElement(int arr[], int size) {
    int count = 0, candidate = -1;

    // Step 1: Find candidate
    for (int i = 0; i < size; i++) {
        if (count == 0) {
            candidate = arr[i];
            count = 1;
        } else {
            if (arr[i] == candidate) {
                count++;
            } else {
                count--;
            }
        }
    }

    // Step 2: Verify candidate
    count = 0;
    for (int i = 0; i < size; i++) {
        if (arr[i] == candidate) {
            count++;
        }
    }

    if (count > size / 2) {
        return candidate;
    } else {
        return -1; // No majority element
    }
}

int main() {
    int arr[] = {3, 4, 2, 4, 2, 4, 4};
    int size = sizeof(arr) / sizeof(arr[0]);
    int majority = findMajorityElement(arr, size);

    if (majority != -1) {
        printf("Majority Element is %d\n", majority);
    } else {
        printf("No Majority Element\n");
    }
    return 0;
}
```

6. Check if a String is a Palindrome Using Stack

```
c
Copy code
#include <stdio.h>
#include <string.h>

int isPalindrome(char str[]) {
    int n = strlen(str);
    char stack[n];
    int top = -1;

    // Push all characters onto stack
    for (int i = 0; i < n; i++) {
        stack[++top] = str[i];
    }

    // Compare characters while popping
    for (int i = 0; i < n; i++) {
        if (str[i] != stack[top--]) {
            return 0; // Not a palindrome
        }
    }

    return 1; // Palindrome
}

int main() {
    char str[] = "madam";

    if (isPalindrome(str)) {
        printf("The string is a palindrome\n");
    } else {
        printf("The string is not a palindrome\n");
    }

    return 0;
}
```

7. Limitations of Linear Queue and Circular Queue Implementation

Linear Queue Limitation:

- Queue overflow occurs even if space is available due to fixed starting position.

Circular Queue Implementation in C:

```
c
Copy code
#include <stdio.h>
#define MAX 5
```

```c
struct CircularQueue {
    int items[MAX];
    int front, rear;
};

int isFull(struct CircularQueue* q) {
    return (q->front == (q->rear + 1) % MAX);
}

int isEmpty(struct CircularQueue* q) {
    return (q->front == -1);
}

void enqueue(struct CircularQueue* q, int value) {
    if (isFull(q)) {
        printf("Queue is full\n");
        return;
    }
    if (q->front == -1) {
        q->front = 0;
    }
    q->rear = (q->rear + 1) % MAX;
    q->items[q->rear] = value;
}

int dequeue(struct CircularQueue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    int value = q->items[q->front];
    if (q->front == q->rear) {
        q->front = q->rear = -1; // Queue is empty
    } else {
        q->front = (q->front + 1) % MAX;
    }
    return value;
}

int main() {
    struct CircularQueue q;
    q.front = q.rear = -1;

    enqueue(&q, 10);
    enqueue(&q, 20);
    enqueue(&q, 30);
    printf("Dequeued: %d\n", dequeue(&q));
    enqueue(&q, 40);
    enqueue(&q, 50);
    enqueue(&q, 60); // Queue is full

    return 0;
}
```