Data Structure and Applications (IS233AI)
CIE 2 scheme and Solutions

QN	Scheme and Solutions	M
1.1	`calloc()` is ideal because it allocates memory and initializes all bytes to zero. 1m code: int *arr = (int *)calloc(100, sizeof(int)); 1m	2
1.2	`head = newnode;` incorrectly updates only the local pointer, not the actual head pointer. 1m Code Correction 1m	2

```
void insertBegin(struct node **head, int value){
    struct node *newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = value;
    newnode->next = *head;
    *head = newnode;
}
```

QN	Scheme and Solutions	M
1.3	1 3 5 5 3 1	2
1.4	The list nodes are not stored in contiguous memory, and each node can only be accessed through its pointer from the previous node. To delete a specific node, we must first reach it by starting from the head and moving node-by-node. We also need the address of the previous node to update its next pointer to skip the node being deleted.	2
1.5	Each carries half marks a. Strictly BT b. Complete BT c. Skewed BT d. Perfect Bt	2
1a	Any 2 valid points from the following each carries 2 marks	4

Malloc( )		Calloc( )	
1	Syntax of malloc( ) is ptr=(data_type*)malloc(size);	1	Syntax of calloc( ) is ptr=(data_type*)calloc(n, size);
2	Allocate a block of memory of size **bytes**	2	Allocates multiple blocks of memory each block with **same size**
3	Allocated space will not be initiated.	3	Each byte of allocated space in initialized to **Zero**
4	Since no initialization is takes place time efficiency is high compared to calloc	4	Since initialization is takes place time efficiency is low compared to malloc
5	If continuous memory location is not available in memory, allocation done at different and random location.	5	It allocates memory only the required space is available in memory otherwise it returns NULL.

6	Initialization of memory can be done by sing the following statement
	ptr=malloc(sizeof(int) * n);
	memset(ptr, 0, sizeof(int) * n);
		6	Implicitly the function do the memory initialization
			ptr=(int*)calloc(n, sizeof(int));

QN	Scheme and Solutions	M
1b	Correct handling of edge cases – **2 marks**, Loop logic – **2 marks,** Correct return – **2 marks**	4

```
struct node* secondLast(struct node *head)
{
    if (head == NULL || head->next == NULL)
        return NULL;

    struct node *temp = head;
    while (temp->next->next != NULL)
    {
        temp = temp->next;
    }
    return temp;
}
```

<table>
<tr><td>2a</td><td>Stepwise correct logic – 4 marks
Algorithm to insert at beginning of Circular SLL
<pre>
1. Create new node.
2. Set new node’s data.
3. If list empty:
       Make new node point to itself.
       Set head = newnode.
4. Else:
       Traverse to last node (node whose next = head).
       Set last->next = newnode.
       newnode->next = head.
       Update head = newnode.
</pre></td><td>4</td></tr>
<tr><td>2b</td><td>Base cases – 2 marks, Recursive comparison – 2 marks, Correct final linking – 2 marks
<pre>
struct node *merge(struct node *s1, struct node *s2)
{
    if (!s1) return s2;
    if (!s2) return s1;

    struct node *result;

    if (s1->data <= s2->data) {
        result = s1;
        result->next = merge(s1->next, s2);
    } else {
        result = s2;
        result->next = merge(s1, s2->next);
    }
    return result;
}
</pre></td><td>6</td></tr>
<tr><td>3a</td><td>Each advantage carries 1 mark

Bidirectional Traversal – Nodes can be traversed in both forward and backward directions.
Easier Deletion – A node can be deleted without needing to traverse the list to find its previous node.
Efficient Insertion/Deletion Before a Node – Inserting or deleting a node before a given node is simpler because the previous pointer is directly available.
Better Navigation – Moving back to the previous element is easier, which is useful in applications like browser history, playlists, and undo operations.
</td><td>4</td></tr>
<tr><td>3b</td><td><pre>
Node* addLists(Node *h1, Node *h2) {
    Node *t1 = h1, *t2 = h2;
    Node *result = NULL;
    int carry = 0;

    while (t1 && t1->next) t1 = t1->next;
    while (t2 && t2->next) t2 = t2->next;

    // Add from least significant to most significant
    while (t1 || t2 || carry) {
        int v1 = t1 ? t1->digits : 0;
        int v2 = t2 ? t2->digits : 0;

        int sum = v1 + v2 + carry;
        carry = sum / 100000; // because each node stores 5 digits
        sum = sum % 100000;

        insertFront(&result, sum);

        if (t1) t1 = t1->prev;
        if (t2) t2 = t2->prev;
    }
    return result;
}
</pre></td><td>6</td></tr>
</table>

4 (10)

```c
#include <stdio.h>
struct node {
    char song[50];
    struct node *next, *prev;
};
struct node *head = NULL;
void addSong(char name[]) {
    struct node *newnode = (struct node*)malloc(sizeof(struct node));
    strcpy(newnode->song, name);

    if (head == NULL) {
        head = newnode;
        head->next = head->prev = head;
    } else {
        struct node *last = head->prev;
        last->next = newnode;
        newnode->prev = last;
        newnode->next = head;
        head->prev = newnode;
    }
}
void removeSong(char name[]) {
    if (head == NULL) return;

    struct node *temp = head;

    do {
        if (strcmp(temp->song, name) == 0) {
            if (temp->next == temp) {
                head = NULL;
            } else {
                temp->prev->next = temp->next;
                temp->next->prev = temp->prev;
                if (temp == head)
                    head = temp->next;
            }
            free(temp);
            return;
        }
        temp = temp->next;
    } while (temp != head);
}
void playAll() {
    if (head == NULL) return;
    struct node *temp = head;
    do {
        printf("%s\n", temp->song);
        temp = temp->next;
    } while (temp != head);
}
```

5a (5)

Conditions for Strictly Binary Tree 2 marks difference 3 marks

- Every internal node has **exactly 2 children**.
- No node has only one child.

Strict Binary Tree	**Complete Binary Tree**
Each node has 0 or 2 children	All levels filled except last
No single-child node	Last level filled left to right

5b (5)

Internal Nodes: A, B, C, D, E, F (2marks)
Successors: D and E (2 marks)
Degree of B is: 2 (1 mark)