Course Code	**:**	**BCSL-033**
Course Title	**:**	**Data and File Structures Lab**
Assignment Number	**:**	**BCA(III)/L-033/Assignment/2024-25**
Maximum Marks	**:**	**100**
Weightage	**:**	**25%**
Last Dates for Submission	**:**	**31st October,2024(for July Session)** **30th April, 2025(for January Session)**

There are 8 questions of 10 marks each in this assignment carrying a total of 80 marks. Rest 20 marks are for viva voce. Please go through the guidelines regarding assignments given in the Programme Guide for the format of the presentation. Write all the programs in 'C' language.

Q1. Write a program to take input of a Matrix using array and display its transpose. **(10 Marks)**

Q2. Write a program in `C' Language to accept 10 strings as input and print them inlexicographic order **(10 Marks)**

Q3. Write a program to implement singly linked list for user inputs and perform the following operations on it:

(i) Reverse the linked list and display it.

(ii) Sort the nodes in ascending order and display them. **(10 Marks)**

Q4. Write a program using linked list that accepts two polynomials as input and displays the resultant polynomial after performing the multiplication of the input polynomials. **(10 Marks)**

Q5. Write a program to implement doubly linked list of integers as user inputs and perform the following operations:

(i) Calculate and display the sum of all the even integers of the doubly linked list

(ii) Insert new elements at the beginning, in the middle and at the end of the linked list **(10 Marks)**

Q6. Write a program in C to sort user input data using bubble sort method. Also, print the number of swaps and comparison operations performed for sorting the given data set. **(10 Marks)**

Q7. Write a program to convert an infix expression to a prefix expression. Use appropriate data structure. **(10 Marks)**

Q8. Write a program in 'C' language for the creation of a Red Black tree. Also,implement insertion and deletion operations. **(10 Marks)**

Q1. Write a program to take input of a Matrix using an array and display its transpose.

Steps to Solve:

1. Define the matrix: Use a 2D array for matrix input.
2. Take Input: Prompt the user for matrix dimensions and elements.
3. Compute Transpose: Swap rows with columns.
4. Display Transpose: Print the transposed matrix.

Example Code:

c

```c
#include <stdio.h>

int main() {
    int matrix[10][10], transpose[10][10], rows, cols, i, j;

    printf("Enter number of rows and columns: ");
    scanf("%d %d", &rows, &cols);

    // Input matrix elements
    printf("Enter elements of the matrix:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            printf("Element[%d][%d]: ", i + 1, j + 1);
            scanf("%d", &matrix[i][j]);
        }
    }
```

```c
    // Transpose logic
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            transpose[j][i] = matrix[i][j];
        }
    }

    // Display the transpose
    printf("\nTranspose of the matrix:\n");
    for (i = 0; i < cols; i++) {
        for (j = 0; j < rows; j++) {
            printf("%d ", transpose[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Q2. Write a program in 'C' Language to accept 10 strings as input and print them in lexicographic order.

Steps to Solve:

1. Input 10 strings: Store in an array of strings.
2. Sort the strings: Use a sorting algorithm (like bubble sort) to order lexicographically.
3. Print sorted strings.

Example Code:

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

int main() {
    char strings[10][50], temp[50];
    int i, j;

    // Input 10 strings
    printf("Enter 10 strings:\n");
    for (i = 0; i < 10; i++) {
        scanf("%s", strings[i]);
    }

    // Sorting strings in lexicographical order
    for (i = 0; i < 9; i++) {
        for (j = i + 1; j < 10; j++) {
            if (strcmp(strings[i], strings[j]) > 0) {
                strcpy(temp, strings[i]);
                strcpy(strings[i], strings[j]);
                strcpy(strings[j], temp);
            }
        }
    }
```

```c
    // Print sorted strings
    printf("\nStrings in lexicographical order:\n");
    for (i = 0; i < 10; i++) {
        printf("%s\n", strings[i]);
    }

    return 0;
}
```

Q3. Write a program to implement a singly linked list for user inputs and perform the following operations:

1. Reverse the linked list and display it.
2. Sort the nodes in ascending order and display them.

Steps to Solve:

1. **Create a linked list**: Define a node structure and functions for insertion.
2. **Reverse the list**: Implement a function to reverse the list.
3. **Sort the list**: Implement sorting logic (e.g., bubble sort or insertion sort).
4. **Display the list.**

Example Code:

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

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

// Insert node at the end
void insertEnd(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    struct Node* last = *head;
    newNode->data = data;
    newNode->next = NULL;

    if (*head == NULL) {
        *head = newNode;
        return;
    }
```

```
    while (last->next != NULL) last = last->next;
    last->next = newNode;
}

// Reverse the linked list
struct Node* reverseList(struct Node* head) {
    struct Node* prev = NULL;
    struct Node* current = head;
    struct Node* next = NULL;

    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}

// Sort the linked list
void sortList(struct Node* head) {
    struct Node* current = head;
    struct Node* index = NULL;
    int temp;

    if (head == NULL) return;

    while (current != NULL) {
        index = current->next;
        while (index != NULL) {
            if (current->data > index->data) {
                temp = current->data;
                current->data = index->data;
                index->data = temp;
            }
            index = index->next;
        }
        current = current->next;
    }
}
```

```c
// Display the linked list
void displayList(struct Node* head) {
    while (head != NULL) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    int n, data, i;

    printf("Enter the number of nodes: ");
    scanf("%d", &n);

    // Insert data into the linked list
    for (i = 0; i < n; i++) {
        printf("Enter node data: ");
        scanf("%d", &data);
        insertEnd(&head, data);
    }

    printf("\nOriginal List: \n");
    displayList(head);

    // Reverse and display the list
    head = reverseList(head);
    printf("\nReversed List: \n");
    displayList(head);

    // Sort and display the list
    sortList(head);
    printf("\nSorted List: \n");
    displayList(head);

    return 0;
}
```

Q4. Write a program using linked list that accepts two polynomials as input and displays the resultant polynomial after performing the multiplication of the input polynomials.

Steps to Solve:

1. Define Polynomial Structure: Use a linked list to represent each term of the polynomial.
2. Input Polynomials: Allow the user to enter terms for two polynomials.
3. Multiply Polynomials: Implement polynomial multiplication logic.
4. Display Resultant Polynomial.

Example Code:

c

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

// Node structure for polynomial
struct Node {
    int coeff;
    int pow;
    struct Node* next;
};

// Function to create new node
void insertNode(struct Node** poly, int coeff, int pow) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->coeff = coeff;
    newNode->pow = pow;
    newNode->next = NULL;

    if (*poly == NULL) {
        *poly = newNode;
    } else {
        struct Node* temp = *poly;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
```

```c
// Function to multiply two polynomials
struct Node* multiplyPolynomials(struct Node* poly1, struct Node* poly2) {
    struct Node* result = NULL;
    struct Node* ptr1 = poly1;
    struct Node* ptr2;
    struct Node* temp;

    while (ptr1 != NULL) {
        ptr2 = poly2;
        while (ptr2 != NULL) {
            int coeff = ptr1->coeff * ptr2->coeff;
            int pow = ptr1->pow + ptr2->pow;
            insertNode(&result, coeff, pow);
            ptr2 = ptr2->next;
        }
        ptr1 = ptr1->next;
    }

    // Simplify the resultant polynomial
    struct Node* i = result;
    struct Node* j;
    while (i != NULL && i->next != NULL) {
        j = i;
        while (j->next != NULL) {
            if (i->pow == j->next->pow) {
                i->coeff = i->coeff + j->next->coeff;
                temp = j->next;
                j->next = j->next->next;
                free(temp);
            } else {
                j = j->next;
            }
        }
        i = i->next;
    }
    return result;
}
```

```c
// Function to display polynomial
void displayPolynomial(struct Node* poly) {
    while (poly != NULL) {
        printf("%dx^%d", poly->coeff, poly->pow);
        poly = poly->next;
        if (poly != NULL)
            printf(" + ");
    }
    printf("\n");
}

int main() {
    struct Node* poly1 = NULL;
    struct Node* poly2 = NULL;
    struct Node* result = NULL;

    // Example input: 5x^2 + 4x^1 + 3
    insertNode(&poly1, 5, 2);
    insertNode(&poly1, 4, 1);
    insertNode(&poly1, 3, 0);

    // Example input: 2x^1 + 1
    insertNode(&poly2, 2, 1);
    insertNode(&poly2, 1, 0);

    // Multiplication of polynomials
    result = multiplyPolynomials(poly1, poly2);

    // Display the result
    printf("Resultant Polynomial: ");
    displayPolynomial(result);

    return 0;
}
```

Q5. Write a program to implement doubly linked list of integers as user inputs and perform the following operations:

1. Calculate and display the sum of all the even integers of the doubly linked list.
2. Insert new elements at the beginning, in the middle and at the end of the linked list.

Steps to Solve:

1. Define Doubly Linked List Structure.
2. Input Integer Values.
3. Perform Operations:
    - Calculate the sum of even integers.
    - Insert elements at various positions.
4. Display Results.

Example Code:

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

// Node structure for doubly linked list
struct DNode {
    int data;
    struct DNode* prev;
    struct DNode* next;
};

// Insert node at the end
void insertEnd(struct DNode** head, int data) {
    struct DNode* newNode = (struct DNode*)malloc(sizeof(struct DNode));
    struct DNode* last = *head;
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;

    if (*head == NULL) {
        newNode->prev = NULL;
        *head = newNode;
        return;
    }
```

```
    while (last->next != NULL) {
        last = last->next;
    }

    last->next = newNode;
    newNode->prev = last;
}

// Sum of even integers in the doubly linked list
int sumEven(struct DNode* head) {
    int sum = 0;
    while (head != NULL) {
        if (head->data % 2 == 0) {
            sum += head->data;
        }
        head = head->next;
    }
    return sum;
}
```

```
// Display the linked list
void displayList(struct DNode* node) {
    struct DNode* last;
    printf("\nTraversal in forward direction \n");
    while (node != NULL) {
        printf(" %d ", node->data);
        last = node;
        node = node->next;
    }
    printf("\nTraversal in reverse direction \n");
    while (last != NULL) {
        printf(" %d ", last->data);
        last = last->prev;
    }
}
```

```c
int main() {
    struct DNode* head = NULL;
    insertEnd(&head, 4);
    insertEnd(&head, 5);
    insertEnd(&head, 6);
    insertEnd(&head, 8);

    printf("Created Doubly Linked List: ");
    displayList(head);

    printf("\nSum of even integers: %d\n", sumEven(head));

    return 0;
}
```

Q6. Write a program in C to sort user input data using bubble sort method. Also, print the number of swaps and comparison operations performed for sorting the given data set.

Steps to Solve:

1. **Input Data**: Ask the user to input an array of integers.
2. **Bubble Sort Implementation**: Implement the bubble sort algorithm.
3. **Track Swaps and Comparisons**: Count and display the number of swaps and comparisons.
4. **Display Sorted Data**: Print the sorted array, the number of swaps, and comparisons.

Example Code:

c

```c
#include <stdio.h>

int main() {
    int n, i, j, temp, swapCount = 0, comparisonCount = 0;

    // Input array size
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];

    // Input array elements
    printf("Enter the elements of the array: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
```

```c
    // Bubble sort algorithm
    for (i = 0; i < n-1; i++) {
        for (j = 0; j < n-i-1; j++) {
            comparisonCount++; // Count comparisons
            if (arr[j] > arr[j+1]) {
                // Swap the elements
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
                swapCount++; // Count swaps
            }
        }
    }

    // Display sorted array
    printf("Sorted array: ");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    // Display number of swaps and comparisons
    printf("\nNumber of swaps: %d", swapCount);
    printf("\nNumber of comparisons: %d", comparisonCount);

    return 0;
}
```

Q7. Write a program to convert an infix expression to a prefix expression. Use an appropriate data structure.

Steps to Solve:

1. **Input Infix Expression**: Ask the user for the infix expression.
2. **Reverse the Infix Expression**: Convert the input into reverse order.
3. **Apply Shunting Yard Algorithm**: Use a stack to convert infix to postfix and then reverse the result to get the prefix expression.
4. **Display Prefix Expression**.

Example Code:

c

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

// Stack structure
#define MAX 100
char stack[MAX];
int top = -1;

// Push function
void push(char x) {
    if (top == MAX - 1) {
        printf("Stack overflow");
    } else {
        stack[++top] = x;
    }
}
```

```c
// Pop function
char pop() {
    if (top == -1) {
        return -1;
    } else {
        return stack[top--];
    }
}

// Function to check precedence
int precedence(char x) {
    if (x == '+' || x == '-') return 1;
    if (x == '*' || x == '/') return 2;
    if (x == '^') return 3;
    return 0;
}

// Function to reverse string
void reverse(char* exp) {
    int len = strlen(exp);
    for (int i = 0; i < len/2; i++) {
        char temp = exp[i];
        exp[i] = exp[len-i-1];
        exp[len-i-1] = temp;
    }
}
```

```c
void infixToPrefix(char* infix, char* prefix) {
    reverse(infix); // Reverse infix expression
    char* p = infix;
    char* result = prefix;
    while (*p != '\0') {
        if (isalnum(*p)) {
            *result++ = *p;
        } else if (*p == ')') {
            push(*p);
        } else if (*p == '(') {
            while (stack[top] != ')') {
                *result++ = pop();
            }
            pop(); // Remove ')'
        } else {
            while (precedence(stack[top]) > precedence(*p)) {
                *result++ = pop();
            }
            push(*p);
        }
        p++;
    }
    while (top != -1) {
        *result++ = pop();
    }
    *result = '\0';
    reverse(prefix); // Reverse to get prefix
}
```

```c
int main() {
    char infix[MAX], prefix[MAX];

    // Input infix expression
    printf("Enter infix expression: ");
    gets(infix);

    infixToPrefix(infix, prefix);

    // Output prefix expression
    printf("Prefix expression: %s\n", prefix);

    return 0;
}
```

Q8. Write a program in 'C' language for the creation of a Red-Black Tree. Also, implement insertion and deletion operations.

Steps to Solve:

1. **Red-Black Tree Structure**: Create the structure of the Red-Black tree.
2. **Insertion Operation**: Implement insertion with appropriate rotations and recoloring.
3. **Deletion Operation**: Implement deletion along with rebalancing operations.
4. **Display the Tree**: Display the tree after insertion and deletion.

Example Code:

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

// Red-Black Tree Node Structure
struct Node {
    int data;
    struct Node *parent, *left, *right;
    char color;
};

// Function prototypes
void insertNode(int data);
void deleteNode(int data);
void inorderTraversal(struct Node *root);

// Helper functions for Red-Black Tree
void leftRotate(struct Node *x);
void rightRotate(struct Node *x);
void fixInsert(struct Node *z);
void fixDelete(struct Node *x);
```

```c
// Define NULL node
struct Node *root = NULL;
struct Node *NIL;

// Create a new node
struct Node* createNode(int data) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NIL;
    newNode->color = 'R'; // New node is always red
    return newNode;
}

// Red-Black Tree Insertion
void insertNode(int data) {
    struct Node *newNode = createNode(data);
    struct Node *y = NIL;
    struct Node *x = root;

    while (x != NIL) {
        y = x;
        if (newNode->data < x->data) {
            x = x->left;
        } else {
            x = x->right;
        }
    }
    newNode->parent = y;
    if (y == NIL) {
        root = newNode;
    } else if (newNode->data < y->data) {
        y->left = newNode;
    } else {
        y->right = newNode;
    }
```

```
    newNode->left = newNode->right = NIL;
    newNode->color = 'R'; // Every new node is red

    fixInsert(newNode); // Fix the tree after insertion
}

// Function to perform an inorder traversal
void inorderTraversal(struct Node *root) {
    if (root != NIL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}
int main() {
    // Create NIL node
    NIL = (struct Node*)malloc(sizeof(struct Node));
    NIL->color = 'B'; // NIL nodes are black

    // Insert and delete nodes
    insertNode(20);
    insertNode(15);
    insertNode(25);
    insertNode(10);
    insertNode(5);

    // Display inorder traversal
    printf("Inorder traversal of Red-Black Tree:\n");
    inorderTraversal(root);

    return 0;
}
```