

# DATA STRUCTURE LAB FILE



*School of Computer  
Applications*

*Department of*

*Computer Applications*

## ***Submitted By***

<b><i>Student Name</i></b>	PAWAN KUMAR
<b><i>Roll No</i></b>	24/SCA/BCA(AI&ML)/048
<b><i>Programme</i></b>	BCA (AI&ML)
<b><i>Semester</i></b>	2 <sup>nd</sup> Semester
<b><i>Section/Group</i></b>	II C
<b><i>Department</i></b>	Computer Applications
<b><i>Batch</i></b>	2024-2027

***Submitted To***

***Submitted By***

***Faculty  
Name***

***Mrs. Sakshi***

1. Write a program in c to implement insertion in 1d Arrays?

**INPUT:**

```
#include <stdio.h>
#define MAX_SIZE 100
void insertElement(int arr[], int *n, int element, int position)
{
    if (position < 0 || position > *n) {
        printf("Invalid position! Please enter a position between
0 and %d.\n", *n);
        return;
    }
    for (int i = *n; i > position; i--) {
        arr[i] = arr[i - 1];
    }
    arr[position] = element;
    (*n)++;
}

void displayArray(int arr[], int n) {
    printf("Array elements: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

```

int main() {
    int arr[MAX_SIZE];
    int n;
    int element, position;
    printf("Enter the number of elements in the array (max %d):", MAX_SIZE);
    scanf("%d", &n);

    if (n > MAX_SIZE) {
        printf("Number of elements exceeds maximum size!\n");
        return 1;
    }
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    displayArray(arr, n);
    printf("Enter the element to insert: ");
    scanf("%d", &element);
    printf("Enter the position to insert the element (0 to %d):", n);
    scanf("%d", &position);
    insertElement(arr, &n, element, position);
    displayArray(arr, n);
    return 0;
}

```

**OUTPUT:**

main.c

Share

Run

Output

Clear

```
32
33     printf("Enter the number of elements in the array (max %d): ",
34           MAX_SIZE);
35     scanf("%d", &n);
36
37     if (n > MAX_SIZE) {
38         printf("Number of elements exceeds maximum size!\n");
39         return 1;
40     }
41
42     printf("Enter %d elements:\n", n);
43     for (int i = 0; i < n; i++) {
44         scanf("%d", &arr[i]);
45     }
46
47     displayArray(arr, n);
48
49     printf("Enter the element to insert: ");
50     scanf("%d", &element);
51     printf("Enter the position to insert the element (0 to %d): ", n);
52     scanf("%d", &position);
53
54     insertElement(arr, &n, element, position);
55     displayArray(arr, n);
56
57     return 0;
58 }
```

```
Enter the number of elements in the array (max 100): 5
Enter 5 elements:
3
4
6
2
4
Array elements: 3 4 6 2 4
Enter the element to insert: 3
Enter the position to insert the element (0 to 5): 5
Array elements: 3 4 6 2 4 3

=== Code Execution Successful ===
```

2. Write a program in C to implement deletion in 1D Arrays?

**INPUT:**

```
#include <stdio.h>
#define MAX_SIZE 100
void deleteElement(int arr[], int *n, int position) {
    if (position < 0 || position >= *n) {
        printf("Invalid position! Please enter a position between
0 and %d.\n", *n - 1);
        return;
    }
    for (int i = position; i < *n - 1; i++) {
        arr[i] = arr[i + 1];
    }
    (*n)--;
}

void displayArray(int arr[], int n) {
    printf("Array elements: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[MAX_SIZE];
    int n;
    int position;
    printf("Enter the number of elements in the array (max %d):
", MAX_SIZE);
    scanf("%d", &n);
    if (n > MAX_SIZE) {
        printf("Number of elements exceeds maximum size!\n");
        return 1;
    }
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    displayArray(arr, n);
    printf("Enter the position of the element to delete (0 to
%d): ", n - 1);
```

```

scanf("%d", &position);
deleteElement(arr, &n, position);
displayArray(arr, n);

return 0;
}

```

**OUTPUT:**

main.c

Share

Run

Output

Clear

```

1  #include <stdio.h>
2
3  #define MAX_SIZE 100
4  void deleteElement(int arr[], int *n, int position) {
5      if (position < 0 || position >= *n) {
6          printf("Invalid position! Please enter a position between 0 and\n", *n - 1);
7          return;
8      }
9      for (int i = position; i < *n - 1; i++) {
10         arr[i] = arr[i + 1];
11     }
12     (*n)--;
13 }
14 void displayArray(int arr[], int n) {
15     printf("Array elements: ");
16     for (int i = 0; i < n; i++) {
17         printf("%d ", arr[i]);
18     }
19     printf("\n");
20 }
21
22 int main() {
23     int arr[MAX_SIZE];
24     int n;
25     int position;
26     printf("Enter the number of elements in the array (max %d): ",
        MAX_SIZE);

```

```

Enter the number of elements in the array (max 100): 3
Enter 3 elements:
1
3
6
Array elements: 1 3 6
Enter the position of the element to delete (0 to 2): 2
Array elements: 1 3

=== Code Execution Successful ===

```

3. Write a program in C to concatenate two arrays?

**INPUT:**

```
#include <stdio.h>

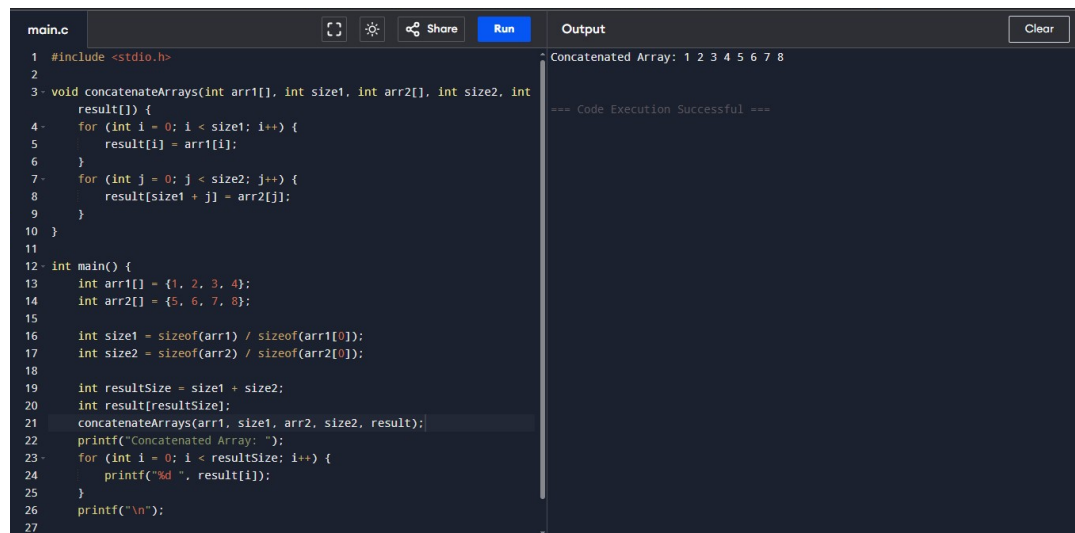
void concatenateArrays(int arr1[], int size1, int arr2[], int
size2, int result[]) {
    for (int i = 0; i < size1; i++) {
        result[i] = arr1[i];
    }
    for (int j = 0; j < size2; j++) {
        result[size1 + j] = arr2[j];
    }
}

int main() {
    int arr1[] = {1, 2, 3, 4};
    int arr2[] = {5, 6, 7, 8};

    int size1 = sizeof(arr1) / sizeof(arr1[0]);
    int size2 = sizeof(arr2) / sizeof(arr2[0]);
    int resultSize = size1 + size2;
    int result[resultSize];
    concatenateArrays(arr1, size1, arr2, size2, result);
    printf("Concatenated Array: ");
    for (int i = 0; i < resultSize; i++) {
        printf("%d ", result[i]);
    }
    printf("\n");

    return 0;
}
```

## OUTPUT:



```
main.c  [Icons]  Run  Output  Clear

1 #include <stdio.h>
2
3 void concatenateArrays(int arr1[], int size1, int arr2[], int size2, int
   result[]) {
4     for (int i = 0; i < size1; i++) {
5         result[i] = arr1[i];
6     }
7     for (int j = 0; j < size2; j++) {
8         result[size1 + j] = arr2[j];
9     }
10 }
11
12 int main() {
13     int arr1[] = {1, 2, 3, 4};
14     int arr2[] = {5, 6, 7, 8};
15
16     int size1 = sizeof(arr1) / sizeof(arr1[0]);
17     int size2 = sizeof(arr2) / sizeof(arr2[0]);
18
19     int resultSize = size1 + size2;
20     int result[resultSize];
21     concatenateArrays(arr1, size1, arr2, size2, result);
22     printf("Concatenated Array: ");
23     for (int i = 0; i < resultSize; i++) {
24         printf("%d ", result[i]);
25     }
26     printf("\n");
27 }
```

Concatenated Array: 1 2 3 4 5 6 7 8

=== Code Execution Successful ===

4. Write a program C to implement the following operations on 2D Array (addition, subtraction, multiplication, transportation)?

### INPUT:

```
#include <stdio.h>
#define MAX 10
void inputMatrix(int matrix[MAX][MAX], int rows, int cols) {
    printf("Enter elements of the matrix (%d x %d):\n", rows,
cols);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("Element [%d][%d]: ", i, j);
            scanf("%d", &matrix[i][j]);
        }
    }
}

void printMatrix(int matrix[MAX][MAX], int rows, int cols) {
```



```

        printf("Matrix (%d x %d):\n", rows, cols);
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                printf("%d ", matrix[i][j]);
            }
            printf("\n");
        }
    }

void addMatrices(int a[MAX][MAX], int b[MAX][MAX], int
result[MAX][MAX], int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            result[i][j] = a[i][j] + b[i][j];
        }
    }
}

void subtractMatrices(int a[MAX][MAX], int b[MAX][MAX], int
result[MAX][MAX], int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            result[i][j] = a[i][j] - b[i][j];
        }
    }
}

void multiplyMatrices(int a[MAX][MAX], int b[MAX][MAX], int
result[MAX][MAX], int rowsA, int colsA, int colsB) {
    for (int i = 0; i < rowsA; i++) {
        for (int j = 0; j < colsB; j++) {
            result[i][j] = 0;
            for (int k = 0; k < colsA; k++) {
                result[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}

void transposeMatrix(int matrix[MAX][MAX], int result[MAX][MAX],
int rows, int cols) {

```

```

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[j][i] = matrix[i][j];
            }
        }
    }

int main() {
    int a[MAX][MAX], b[MAX][MAX], result[MAX][MAX];
    int rowsA, colsA, rowsB, colsB;
    printf("Enter rows and columns for first matrix: ");
    scanf("%d %d", &rowsA, &colsA);
    inputMatrix(a, rowsA, colsA);
    printf("Enter rows and columns for second matrix: ");
    scanf("%d %d", &rowsB, &colsB);
    inputMatrix(b, rowsB, colsB);
    if (rowsA == rowsB && colsA == colsB) {
        addMatrices(a, b, result, rowsA, colsA);
        printf("Addition of matrices:\n");
        printMatrix(result, rowsA, colsA);

        subtractMatrices(a, b, result, rowsA, colsA);
        printf("Subtraction of matrices:\n");
        printMatrix(result, rowsA, colsA);
    } else {
        printf("Addition and subtraction are not possible for the
given matrices.\n");
    }
    if (colsA == rowsB) {
        multiplyMatrices(a, b, result, rowsA, colsA, colsB);
        printf("Multiplication of matrices:\n");
        printMatrix(result, rowsA, colsB);
    } else {
        printf("Multiplication is not possible for the given
matrices.\n");
    }
    transposeMatrix(a, result, rowsA, colsA);
    printf("Transpose of the first matrix:\n");
    printMatrix(result, colsA, rowsA);

    transposeMatrix(b, result, rowsB, colsB);

```

```

        printf("Transpose of the second matrix:\n");
        printMatrix(result, colsB, rowsB);

    return 0;
}

```

**OUTPUT:**

The screenshot shows a C program running in an IDE. The code defines a transpose function and a main function that takes user input for two matrices. The output shows the first matrix (3x4) and the second matrix (2x2) being entered. It then displays the transpose of the first matrix (4x3) and the transpose of the second matrix (2x2). The program also includes error messages for addition and subtraction when matrix dimensions are incompatible.

```

main.c
47- }
48- }
49- }
50- }
51-
52- void transposeMatrix(int matrix[MAX][MAX], int result[MAX][MAX], int
    rows, int cols) {
53-     for (int i = 0; i < rows; i++) {
54-         for (int j = 0; j < cols; j++) {
55-             result[j][i] = matrix[i][j];
56-         }
57-     }
58- }
59-
60- int main() {
61-     int a[MAX][MAX], b[MAX][MAX], result[MAX][MAX];
62-     int rowsA, colsA, rowsB, colsB;
63-     printf("Enter rows and columns for first matrix: ");
64-     scanf("%d %d", &rowsA, &colsA);
65-     inputMatrix(a, rowsA, colsA);
66-     printf("Enter rows and columns for second matrix: ");
67-     scanf("%d %d", &rowsB, &colsB);
68-     inputMatrix(b, rowsB, colsB);
69-     if (rowsA == rowsB && colsA == colsB) {
70-         addMatrices(a, b, result, rowsA, colsA);
71-         printf("Addition of matrices:\n");
72-         printMatrix(result, rowsA, colsA);
73-     }
74- }

```

Output:

```

Enter rows and columns for first matrix: 3
4
Enter elements of the matrix (3 x 4):
Element [0][0]: 1
Element [0][1]: 2
Element [0][2]:
2
Element [0][3]: 3
Element [1][0]: 2
Element [1][1]: 2
Element [1][2]: 3
Element [1][3]: 3
Element [2][0]: 4
Element [2][1]: 3
Element [2][2]: 3
Element [2][3]:
3
Enter rows and columns for second matrix: 2
2
Enter elements of the matrix (2 x 2):
Element [0][0]: 4
Element [0][1]: 3
Element [1][0]: 4
Element [1][1]: 3
Addition and subtraction are not possible for the given matrices.
Multiplication is not possible for the given matrices.
Transpose of the first matrix:
Matrix (4 x 3):

```

5. Write a program in C to implement operations on Stack using array?

**INPUT:**

```

#include <stdio.h>
#include <stdlib.h>
#define MAX 100
struct Stack {
    int top;
    int items[MAX];
};

void initStack(struct Stack* s) {
    s->top = -1;
}

int isFull(struct Stack* s) {
    return s->top == MAX - 1;
}

int isEmpty(struct Stack* s) {
    return s->top == -1;
}

```

```

void push(struct Stack* s, int item) {
    if (isFull(s)) {
        printf("Stack Overflow! Cannot push %d\n", item);
    } else {
        s->items[++(s->top)] = item;
        printf("%d pushed to stack\n", item);
    }
}

int pop(struct Stack* s) {
    if (isEmpty(s)) {
        printf("Stack Underflow! Cannot pop from empty stack\n");
        return -1;
    } else {
        return s->items[(s->top)--];
    }
}

void display(struct Stack* s) {
    if (isEmpty(s)) {
        printf("Stack is empty\n");
    } else {
        printf("Stack elements are:\n");
        for (int i = s->top; i >= 0; i--) {
            printf("%d\n", s->items[i]);
        }
    }
}

int main() {
    struct Stack s;
    initStack(&s);
    int choice, value;
    do {
        printf("\nStack Operations:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to push: ");

```

```

        scanf("%d", &value);
        push(&s, value);
        break;
    case 2:
        value = pop(&s);
        if (value != -1) {
            printf("Popped value: %d\n", value);
        }
        break;
    case 3:
        display(&s);
        break;
    case 4:
        printf("Exiting...\n");
        break;
    default:
        printf("Invalid choice! Please try again.\n");
    }
} while (choice != 4);
return 0;
}

```

**OUTPUT:**



```

        q->front = 0;
    }
    q->rear = (q->rear + 1) % MAX;
    q->items[q->rear] = item;
    printf("%d enqueued to queue\n", item);
}
}
int dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue Underflow! Cannot dequeue from empty
queue\n");
        return -1;
    } else {
        int item = q->items[q->front];
        if (q->front == q->rear) {
            q->front = -1;
            q->rear = -1;
        } else {
            q->front = (q->front + 1) % MAX; // Circular
increment
        }
        return item;
    }
}
void display(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
    } else {
        printf("Queue elements are:\n");
        int i = q->front;
        while (1) {
            printf("%d\n", q->items[i]);
            if (i == q->rear) break;
            i = (i + 1) % MAX;
        }
    }
}
int main() {
    struct Queue q;
    initQueue(&q);

```

```

int choice, value;

do {
    printf("\nQueue Operations:\n");
    printf("1. Enqueue\n");
    printf("2. Dequeue\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter value to enqueue: ");
            scanf("%d", &value);
            enqueue(&q, value);
            break;
        case 2:
            value = dequeue(&q);
            if (value != -1) {
                printf("Dequeued value: %d\n", value); }
            break;
        case 3:
            display(&q);
            break;
        case 4:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice! Please try again.\n");
    }
} while (choice != 4);
return 0;
}

```

**OUTPUT:**



```

main.c  [Icons]  Share  Run  Output  Clear
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define MAX 100 // Maximum size of the queue
5
6 // Queue structure
7 struct Queue {
8     int items[MAX];
9     int front;
10    int rear;
11 };
12
13 // Function to initialize the queue
14 void initQueue(struct Queue* q) {
15     q->front = -1; // Queue is initially empty
16     q->rear = -1;
17 }
18
19 // Function to check if the queue is full
20 int isFull(struct Queue* q) {
21     return (q->rear + 1) % MAX == q->front;
22 }
23
24 // Function to check if the queue is empty
25 int isEmpty(struct Queue* q) {
26     return q->front == -1;
27 }

```

```

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 4
4 enqueued to queue

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued value: 4

Queue Operations:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting...

```

7. Write a program in C to implement operations on circular queue using array?

**INPUT:**

```

#include <stdio.h>
#include <stdlib.h>
#define MAX 100
struct CircularQueue {
    int items[MAX];
    int front;
    int rear;
};

void initQueue(struct CircularQueue* q) {
    q->front = -1;
    q->rear = -1;
}

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

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

void enqueue(struct CircularQueue* q, int item) {
    if (isFull(q)) {
        printf("Queue Overflow! Cannot enqueue %d\n", item);
    } else {
        if (isEmpty(q)) {
            q->front = 0;
        }
        q->rear = (q->rear + 1) % MAX;
    }
}

```

```

        q->items[q->rear] = item;
        printf("%d enqueued to queue\n", item);
    }
}

int dequeue(struct CircularQueue* q) {
    if (isEmpty(q)) {
        printf("Queue Underflow! Cannot dequeue from empty
queue\n");
        return -1; // Return -1 to indicate queue is empty
    } else {
        int item = q->items[q->front];
        if (q->front == q->rear) {
            q->front = -1;
            q->rear = -1;
        } else {
            q->front = (q->front + 1) % MAX;
        }
        return item;
    }
}

void display(struct CircularQueue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
    } else {
        printf("Queue elements are:\n");
        int i = q->front;
        while (1) {
            printf("%d\n", q->items[i]);
            if (i == q->rear) break;
            i = (i + 1) % MAX;
        }
    }
}

int main() {
    struct CircularQueue q;
    initQueue(&q);
    int choice, value;
    do {
        printf("\nCircular Queue Operations:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
    }

```

```

printf("3. Display\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter value to enqueue: ");
        scanf("%d", &value);
        enqueue(&q, value);
        break;
    case 2:
        value = dequeue(&q);
        if (value != -1) {
            printf("Dequeued value: %d\n", value);
        }
        break;
    case 3:
        display(&q);
        break;
    case 4:
        printf("Exiting...\n");
        break;
    default:
        printf("Invalid choice! Please try again.\n");
}
} while (choice != 4);
return 0;
}

```

**OUTPUT:**

```
scanf("%d", &choice);

switch (choice) {
    case 1:
        printf("Enter value to enqueue: ");
        scanf("%d", &value);
        enqueue(&q, value);
        break;
    case 2:
        value = dequeue(&q);
        if (value != -1) {
            printf("Dequeued value: %d\n", value);
        }
        break;
    case 3:
        display(&q);
        break;
    case 4:
        printf("Exiting...\n");
        break;
    default:
        printf("Invalid choice! Please try again.\n");
}
} while (choice != 4);

return 0;
```

Output

Circular Queue Operations:  
1. Enqueue  
2. Dequeue  
3. Display  
4. Exit  
Enter your choice: 2  
Queue Underflow! Cannot dequeue from empty queue

Circular Queue Operations:  
1. Enqueue  
2. Dequeue  
3. Display  
4. Exit  
Enter your choice: 3  
Queue is empty

Circular Queue Operations:  
1. Enqueue  
2. Dequeue  
3. Display  
4. Exit  
Enter your choice: 1  
Enter value to enqueue: 2  
2 enqueued to queue

8. Write a program in c to implement insertion in a linked list (beg, mid, end)?

**INPUT:**

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int data) {
```

```

    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
    printf("Inserted %d at the beginning\n", data);
}
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        printf("Inserted %d at the end\n", data);
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    printf("Inserted %d at the end\n", data);
}
void insertAtPosition(struct Node** head, int data, int
position) {
    if (position < 1) {
        printf("Position should be >= 1\n");
        return;
    }
    if (position == 1) {
        insertAtBeginning(head, data);
        return;
    }
    struct Node* newNode = createNode(data);
    struct Node* temp = *head;
    for (int i = 1; i < position - 1 && temp != NULL; i++) {
        temp = temp->next;
    }
}

```

```

        if (temp == NULL) {
            printf("Position exceeds the length of the list.
Inserting at the end instead.\n");
            insertAtEnd(head, data);
        } else {
            newNode->next = temp->next;
            temp->next = newNode;
            printf("Inserted %d at position %d\n", data, position);
        }
    }
}

void displayList(struct Node* head) {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }
    struct Node* temp = head;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    int choice, data, position;
    do {
        printf("\nLinked List Operations:\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Display List\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert at beginning: ");
                scanf("%d", &data);
                insertAtBeginning(&head, data);
                break;

```

```

        case 2:
            printf("Enter value to insert at end: ");
            scanf("%d", &data);
            insertAtEnd(&head, data);
            break;
        case 3:
            printf("Enter value to insert and position: ");
            scanf("%d %d", &data, &position);
            insertAtPosition(&head, data, position);
            break;
        case 4:
            displayList(head);
            break;
        case 5:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice! Please try again.\n");
    }
} while (choice != 5);
return 0;
}

```

## OUTPUT:

The screenshot shows a code editor with a dark theme. The left pane contains the C code for linked list operations, which is identical to the code block above. The right pane, titled 'Output', shows the program's execution. It displays a menu of operations, the user's choice of 2 (insert at end), the input value 1, and the confirmation 'Inserted 1 at the end'. This sequence is repeated with choice 3, input 3, and a warning message 'Position exceeds the length of the list. Inserting at the end instead.' before confirming 'Inserted 3 at the end'. The output ends with the menu shown again.

```

printf("Enter value to insert at beginning: ");
scanf("%d", &data);
insertAtBeginning(&head, data);
break;
case 2:
    printf("Enter value to insert at end: ");
    scanf("%d", &data);
    insertAtEnd(&head, data);
    break;
case 3:
    printf("Enter value to insert and position: ");
    scanf("%d %d", &data, &position);
    insertAtPosition(&head, data, position);
    break;
case 4:
    displayList(head);
    break;
case 5:
    printf("Exiting...\n");
    break;
default:
    printf("Invalid choice! Please try again.\n");
}
} while (choice != 5);
return 0;

```

Linked List Operations:  
1. Insert at Beginning  
2. Insert at End  
3. Insert at Position  
4. Display List  
5. Exit  
Enter your choice: 2  
Enter value to insert at end: 1  
Inserted 1 at the end

Linked List Operations:  
1. Insert at Beginning  
2. Insert at End  
3. Insert at Position  
4. Display List  
5. Exit  
Enter your choice: 3  
Enter value to insert and position: 3  
3  
Position exceeds the length of the list. Inserting at the end instead.  
Inserted 3 at the end

Linked List Operations:  
1. Insert at Beginning  
2. Insert at End

9. Write a program in C to implement deletion from a linked list?

**INPUT:**

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
void insertEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}
void displayList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
void deleteFromBeginning(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty. Nothing to delete.\n");
        return;
    }
}
```



```

    }
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
}

void deleteFromEnd(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty. Nothing to delete.\n");
        return;
    }
    struct Node* temp = *head;
    struct Node* prev = NULL;
    if (temp->next == NULL) {
        free(temp);
        *head = NULL;
        return;
    }
    while (temp->next != NULL) {
        prev = temp;
        temp = temp->next;
    }
    free(temp);
    prev->next = NULL;
}

void deleteFromMiddle(struct Node** head, int position) {
    if (*head == NULL) {
        printf("List is empty. Nothing to delete.\n");
        return;
    }

    struct Node* temp = *head;
    if (position == 0) {
        deleteFromBeginning(head);
        return;
    }
    for (int i = 0; temp != NULL && i < position - 1; i++) {
        temp = temp->next;
    }
    if (temp == NULL || temp->next == NULL) {

```

```

        printf("Position %d does not exist in the list.\n",
position);
        return;
    }
    struct Node* next = temp->next->next;
    free(temp->next);
    temp->next = next;
}
int main() {
    struct Node* head = NULL;

    insertEnd(&head, 10);
    insertEnd(&head, 20);
    insertEnd(&head, 30);
    insertEnd(&head, 40);
    insertEnd(&head, 50);
    printf("Linked List before deletion:\n");
    displayList(head);
    printf("Deleting from the beginning:\n");
    deleteFromBeginning(&head);
    displayList(head);
    printf("Deleting from the end:\n");
    deleteFromEnd(&head);
    displayList(head);
    printf("Deleting from the middle (position 1):\n");
    deleteFromMiddle(&head, 1);
    displayList(head);
    while (head != NULL) {
        struct Node* temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
}

```

**OUTPUT:**

main.c	Run	Output
<pre>1 #include &lt;stdio.h&gt; 2 #include &lt;stdlib.h&gt; 3 4 // Define the structure for a node in the linked list 5 struct Node { 6     int data; 7     struct Node* next; 8 }; 9 10 // Function to create a new node 11 struct Node* createNode(int data) { 12     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node 13 )); 14     newNode-&gt;data = data; 15     newNode-&gt;next = NULL; 16     return newNode; 17 } 18 // Function to insert a node at the end of the linked list 19 void insertEnd(struct Node** head, int data) { 20     struct Node* newNode = createNode(data); 21     if (*head == NULL) { 22         *head = newNode; 23         return; 24     } 25     struct Node* temp = *head;</pre>	<div>Run</div>	<pre>Linked List before deletion: 10 -&gt; 20 -&gt; 30 -&gt; 40 -&gt; 50 -&gt; NULL Deleting from the beginning: 20 -&gt; 30 -&gt; 40 -&gt; 50 -&gt; NULL Deleting from the end: 20 -&gt; 30 -&gt; 40 -&gt; NULL Deleting from the middle (position 1): 20 -&gt; 40 -&gt; NULL  === Code Execution Successful ===</pre>