

Data structures lab program

1. Write a program in C to implement insertion in 1-D Arrays

```
main.c
1 #include <stdio.h>
2
3 int main() {
4     int arr[100], n, i, pos, value;
5
6     printf("Enter the number of elements in the array: ");
7     scanf("%d", &n);
8
9     printf("Enter %d elements:\n", n);
10    for(i = 0; i < n; i++) {
11        scanf("%d", &arr[i]);
12    }
13
14    printf("Enter the element to insert: ");
15    scanf("%d", &value);
16    printf("Enter the position to insert (0 to %d): ", n);
17    scanf("%d", &pos);
18    if(pos < 0 || pos > n) {
19        printf("Invalid position!\n");
20        return 1;
21    }
22
23    for(i = n; i > pos; i--) {
24        arr[i] = arr[i - 1];
25    }
26
27    arr[pos] = value;
28    n++;
29
30    printf("Array after insertion:\n");
31    for(i = 0; i < n; i++) {
32        printf("%d ", arr[i]);
33    }
34
35    return 0;
36 }
```

Output

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

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

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

```
#include <stdio.h>
int main() {
    int arr[100], n, i, pos;

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

    printf("Enter %d elements:\n", n);
    for(i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    printf("Enter the position to delete (0 to %d): ", n - 1);
    scanf("%d", &pos);

    if(pos < 0 || pos >= n) {
        printf("Invalid position!\n");
        return 1;
    }

    for(i = pos; i < n - 1; i++) {
        arr[i] = arr[i + 1];
    }

    n--;

    printf("Array after deletion:\n");
    for(i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

Output

```
Enter the number of elements in the array: 5
Enter 5 elements:
10 30 40 50 60
Enter the position to delete (0 to 4): 3
Array after deletion:
10 30 40 60

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

3. Write a program in C to implement linear and binary searching in 1-D Arrays

```
main.c 1 #include <stdio.h>
2 int linearSearch(int arr[], int n, int key) {
3     for(int i = 0; i < n; i++) {
4         if(arr[i] == key)
5             return i;
6     }
7     return -1;
8 }
9 int binarySearch(int arr[], int n, int key) {
10     int low = 0, high = n - 1, mid;
11     while(low <= high) {
12         mid = (low + high) / 2;
13         if(arr[mid] == key)
14             return mid;
15         else if(arr[mid] < key)
16             low = mid + 1;
17         else
18             high = mid - 1;
19     }
20     return -1;
21 }
22
23 int main() {
24     int arr[100], n, key, choice, i, result;
25
26     printf("Enter number of elements: ");
27     scanf("%d", &n);
28     printf("Enter %d elements:\n", n);
29     for(i = 0; i < n; i++) {
30         scanf("%d", &arr[i]);
31     }
32
33     printf("Enter the element to search: ");
34     scanf("%d", &key);
35     printf("Choose search method:\n");
36     printf("1. Linear Search\n");
37     printf("2. Binary Search (array must be sorted)\n");
38 }
```

Output

```
Enter number of elements: 4
Enter 4 elements:
30 50 40 60
Enter the element to search: 30
Choose search method:
1. Linear Search
2. Binary Search (array must be sorted)
2
Element found at index 0

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

4. Write a program in C to implement sorting in 1-D Arrays

```
main.c 1 #include <stdio.h>
2
3 int main() {
4     int arr[100], n, i, j, temp;
5
6     printf("Enter the number of elements: ");
7     scanf("%d", &n);
8
9     printf("Enter %d elements:\n", n);
10    for(i = 0; i < n; i++) {
11        scanf("%d", &arr[i]);
12    }
13
14    for(i = 0; i < n - 1; i++) {
15        for(j = 0; j < n - i - 1; j++) {
16            if(arr[j] > arr[j + 1]) {
17                temp = arr[j];
18                arr[j] = arr[j + 1];
19                arr[j + 1] = temp;
20            }
21        }
22    }
23
24    printf("Sorted array in ascending order:\n");
25    for(i = 0; i < n; i++) {
26        printf("%d ", arr[i]);
27    }
28
29    return 0;
30 }
31 }
```

Output

```
Enter the number of elements: 5
Enter 5 elements:
50 60 40 30 20
Sorted array in ascending order:
20 30 40 50 60

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

5. Write a program in C to concatenate two arrays

```
1 #include <stdio.h>
2
3 int main() {
4     int arr1[100], arr2[100], arr3[200];
5     int n1, n2, i, j;
6
7     printf("Enter the number of elements in the first array: ");
8     scanf("%d", &n1);
9     printf("Enter %d elements for the first array:\n", n1);
10    for(i = 0; i < n1; i++) {
11        scanf("%d", &arr1[i]);
12    }
13
14    printf("Enter the number of elements in the second array: ");
15    scanf("%d", &n2);
16    printf("Enter %d elements for the second array:\n", n2);
17    for(i = 0; i < n2; i++) {
18        scanf("%d", &arr2[i]);
19    }
20
21    for(i = 0; i < n1; i++) {
22        arr3[i] = arr1[i];
23    }
24
25    for(j = 0; j < n2; j++) {
26        arr3[i + j] = arr2[j];
27    }
28
29    printf("Concatenated array:\n");
30    for(i = 0; i < n1 + n2; i++) {
31        printf("%d ", arr3[i]);
32    }
33
34    return 0;
35 }
36
```

Enter the number of elements in the first array: 3
Enter 3 elements for the first array:
2 4 5
Enter the number of elements in the second array: 4
Enter 4 elements for the second array:
5 7 8 9
Concatenated array:
2 4 5 5 7 8 9
=== Code Execution Successful ===

6. Write a program in C to implement the following Operations on 2-D Array (addition; subtraction; multiplication; transpose)

```
1 #include <stdio.h>
2 int main() {
3     int a[10][10], b[10][10], result[10][10], transpose[10][10];
4     int i, j, k, r, c, choice;
5     printf("Enter number of rows and columns (max 10): ");
6     scanf("%d%d", &r, &c);
7
8     printf("Enter elements of Matrix A:\n");
9     for(i = 0; i < r; i++)
10         for(j = 0; j < c; j++)
11             scanf("%d", &a[i][j]);
12
13     printf("Enter elements of Matrix B:\n");
14     for(i = 0; i < r; i++)
15         for(j = 0; j < c; j++)
16             scanf("%d", &b[i][j]);
17
18     printf("\nChoose Operation:\n");
19     printf("1. Addition\n2. Subtraction\n3. Multiplication\n4. Transpose of A\n");
20     scanf("%d", &choice);
21
22     switch(choice) {
23         case 1:
24             for(i = 0; i < r; i++)
25                 for(j = 0; j < c; j++)
26                     result[i][j] = a[i][j] + b[i][j];
27
28             printf("Addition of Matrices:\n");
29             for(i = 0; i < r; i++) {
30                 for(j = 0; j < c; j++)
31                     printf("%d ", result[i][j]);
32                 printf("\n");
33             }
34             break;
35     }
36 }
37
```

Enter number of rows and columns (max 10): 2 2
Enter elements of Matrix A:
1 3
3 4
Enter elements of Matrix B:
4 5
6 7
Choose Operation:
1. Addition
2. Subtraction
3. Multiplication
4. Transpose of A
3
Multiplication of Matrices:
22 26
36 43
=== Code Execution Successful ===

```

37     case 2:
38         for(i = 0; i < r; i++)
39             for(j = 0; j < c; j++)
40                 result[i][j] = a[i][j] - b[i][j];
41
42         printf("Subtraction of Matrices:\n");
43         for(i = 0; i < r; i++) {
44             for(j = 0; j < c; j++)
45                 printf("%d ", result[i][j]);
46             printf("\n");
47         }
48         break;
49
50     case 3:
51
52         for(i = 0; i < r; i++) {
53             for(j = 0; j < c; j++) {
54                 result[i][j] = 0;
55                 for(k = 0; k < c; k++)
56                     result[i][j] += a[i][k] * b[k][j];
57             }
58         }
59
60         printf("Multiplication of Matrices:\n");
61         for(i = 0; i < r; i++) {
62             for(j = 0; j < c; j++)
63                 printf("%d ", result[i][j]);
64             printf("\n");
65         }
66         break;
67
68     case 4:
69
70         for(i = 0; i < r; i++)
71             for(j = 0; j < c; j++)
72                 transpose[j][i] = a[i][j];

```

Enter number of rows and columns (max 10): 2 2
Enter elements of Matrix A:
1 3
3 4
Enter elements of Matrix B:
4 5
6 7
Choose Operation:
1. Addition
2. Subtraction
3. Multiplication
4. Transpose of A
3
Multiplication of Matrices:
22 26
36 43
=== Code Execution Successful ===

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

```

main.c
1  #include <stdio.h>
2  #define SIZE 100
3
4  int stack[SIZE], top = -1;
5  void push(int value) {
6      if(top == SIZE - 1)
7          printf("Stack Overflow\n");
8      else {
9          top++;
10         stack[top] = value;
11         printf("Pushed %d\n", value);
12     }
13 }
14 void pop() {
15     if(top == -1)
16         printf("Stack Underflow\n");
17     else {
18         printf("Popped %d\n", stack[top]);
19         top--;
20     }
21 }
22 void peek() {
23     if(top == -1)
24         printf("Stack is Empty\n");
25     else
26         printf("Top element is %d\n", stack[top]);
27 }
28
29 void display() {
30     if(top == -1)
31         printf("Stack is Empty\n");
32     else {
33         printf("Stack elements:\n");
34         for(int i = top; i >= 0; i--)
35             printf("%d\n", stack[i]);
36     }

```

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 1
Enter value to push: 2
Pushed 2
Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 2
Popped 2
Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 3
Stack is Empty
Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 4

```
main.c
36     printf("%d\n", stack[i]);
37 }
38 }
39
40 int main() {
41     int choice, value;
42
43     while(1) {
44         printf("\nStack Operations Menu:\n");
45         printf("1. Push\n2. Pop\n3. Peek\n4. Display\n5. Exit\n");
46         printf("Enter choice: ");
47         scanf("%d", &choice);
48
49         switch(choice) {
50             case 1:
51                 printf("Enter value to push: ");
52                 scanf("%d", &value);
53                 push(value);
54                 break;
55             case 2:
56                 pop();
57                 break;
58             case 3:
59                 peek();
60                 break;
61             case 4:
62                 display();
63                 break;
64             case 5:
65                 return 0;
66             default:
67                 printf("Invalid choice\n");
68         }
69     }
70
71     return 0;
72 }
```

Output

```
Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 2
Popped 2

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 3
Stack is Empty

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 4
Stack is Empty

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 5

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

8. Write a program in C to implement operations on Stack using

linked list

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct Node {
5     int data;
6     struct Node* next;
7 };
8
9 struct Node* top = NULL;
10 void push(int value) {
11     struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
12     if(newNode == NULL) {
13         printf("Stack Overflow\n");
14         return;
15     }
16     newNode->data = value;
17     newNode->next = top;
18     top = newNode;
19     printf("Pushed %d\n", value);
20 }
21
22 void pop() {
23     if(top == NULL) {
24         printf("Stack Underflow\n");
25         return;
26     }
27     struct Node* temp = top;
28     printf("Popped %d\n", temp->data);
29     top = top->next;
30     free(temp);
31 }
32
33 void peek() {
34     if(top == NULL)
35         printf("Stack is Empty\n");
36     else
```

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 10
Invalid choice

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 2
Stack Underflow

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 3
Stack is Empty

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 4
Stack is Empty

9. Write a program in C to implement applications of Stack

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define SIZE 100
6
7 char stack[SIZE];
8 int top = -1;
9
10 void push(char ch) {
11     if(top < SIZE - 1)
12         stack[++top] = ch;
13 }
14
15 char pop() {
16     if(top != -1)
17         return stack[top--];
18     return '\0';
19 }
20
21 int isMatchingPair(char open, char close) {
22     return (open == '(' && close == ')') ||
23            (open == '[' && close == ']') ||
24            (open == '{' && close == '}');
25 }
26
27 int isBalanced(char* expr) {
28     for(int i = 0; i < strlen(expr); i++) {
29         char ch = expr[i];
30         if(ch == '(' || ch == '[' || ch == '{') {
31             push(ch);
32         } else if(ch == ')' || ch == ']' || ch == '}') {
33             char topChar = pop();
34             if(!isMatchingPair(topChar, ch))
35                 return 0;
36         }
37     }
38     return top == -1;
39 }
40
41 int main() {
42     char expr[SIZE];
43
44     printf("Enter an expression: ");
45     scanf("%s", expr);
46
47     if(isBalanced(expr))
48         printf("The expression is balanced.\n");
49     else
50         printf("The expression is NOT balanced.\n");
51
52     return 0;
53 }
```

Output

Enter an expression: (a+b)*(c-d)
The expression is balanced.

=== Code Execution Successful ===

```
33     char topChar = pop();
34     if(!isMatchingPair(topChar, ch))
35         return 0;
36 }
37 }
38 return top == -1;
39 }
40
41 int main() {
42     char expr[SIZE];
43
44     printf("Enter an expression: ");
45     scanf("%s", expr);
46
47     if(isBalanced(expr))
48         printf("The expression is balanced.\n");
49     else
50         printf("The expression is NOT balanced.\n");
51
52     return 0;
53 }
```

10. Write a program in C to implement operations on queue using array


```

1 #include <stdio.h>
2 #define SIZE 100
3
4 int queue[SIZE];
5 int front = -1, rear = -1;
6
7 void enqueue(int value) {
8     if(rear == SIZE - 1) {
9         printf("Queue Overflow\n");
10    } else {
11        if(front == -1)
12            front = 0;
13        rear++;
14        queue[rear] = value;
15        printf("Enqueued %d\n", value);
16    }
17 }
18 void dequeue() {
19     if(front == -1 || front > rear) {
20         printf("Queue Underflow\n");
21     } else {
22         printf("Dequeued %d\n", queue[front]);
23         front++;
24     }
25 }
26 void display() {
27     if(front == -1 || front > rear) {
28         printf("Queue is Empty\n");
29     } else {
30         printf("Queue elements:\n");
31         for(int i = front; i <= rear; i++)
32             printf("%d ", queue[i]);
33         printf("\n");
34     }
35 }
36

```

```

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 10
Enqueued 10

```

```

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 20
Enqueued 20

```

```

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements:
10 20

```

```

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

```

```

main.c
32     printf("%d ", queue[i]);
33     printf("\n");
34 }
35 }
36
37 int main() {
38     int choice, value;
39
40     while(1) {
41         printf("\nQueue Operations Menu:\n");
42         printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
43         printf("Enter your choice: ");
44         scanf("%d", &choice);
45
46         switch(choice) {
47             case 1:
48                 printf("Enter value to enqueue: ");
49                 scanf("%d", &value);
50                 enqueue(value);
51                 break;
52             case 2:
53                 dequeue();
54                 break;
55             case 3:
56                 display();
57                 break;
58             case 4:
59                 return 0;
60             default:
61                 printf("Invalid choice\n");
62         }
63     }
64
65     return 0;
66 }
67

```

```

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 10
Enqueued 10

```

```

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 20
Enqueued 20

```

```

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements:
10 20

```

```

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

```

=== Code Execution Successful ===

11. Write a program in C to implement operations on queue using linked list

```
main.c | 1 | #include <stdio.h>
      | 2 | #include <stdlib.h>
      | 3 |
      | 4 | struct Node {
      | 5 |     int data;
      | 6 |     struct Node* next;
      | 7 | };
      | 8 |
      | 9 | struct Node* front = NULL;
      |10 | struct Node* rear = NULL;
      |11 |
      |12 | void enqueue(int value) {
      |13 |     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
      |14 |     if (newNode == NULL) {
      |15 |         printf("Memory allocation failed\n");
      |16 |         return;
      |17 |     }
      |18 |     newNode->data = value;
      |19 |     newNode->next = NULL;
      |20 |     if (rear == NULL) {
      |21 |         front = rear = newNode;
      |22 |     } else {
      |23 |         rear->next = newNode;
      |24 |         rear = newNode;
      |25 |     }
      |26 |     printf("Enqueued %d\n", value);
      |27 | }
      |28 | void dequeue() {
      |29 |     if (front == NULL) {
      |30 |         printf("Queue Underflow\n");
      |31 |         return;
      |32 |     }
      |33 |     struct Node* temp = front;
      |34 |     printf("Dequeued %d\n", front->data);
      |35 |     front = front->next;
      |36 |     if (front == NULL)
      |37 |         return;
      |38 | }
```

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 10
Enqueued 10

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued 10

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 10
Invalid choice

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

Queue Operations Menu:
1. Enqueue

main.c

Share

Run

36

if (front == NULL)

37

rear = NULL;

38

free(temp);

39

}

40

41

void display() {

42

if (front == NULL) {

43

printf("Queue is Empty\n");

44

return;

45

}

46

struct Node* temp = front;

47

printf("Queue elements:\n");

48

while (temp != NULL) {

49

printf("%d ", temp->data);

50

temp = temp->next;

51

}

52

printf("\n");

53

}

54

55

int main() {

56

int choice, value;

57

58

while (1) {

59

printf("\nQueue Operations Menu:\n");

60

printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");

61

printf("Enter your choice: ");

62

scanf("%d", &choice);

63

64

switch (choice) {

65

case 1:

66

printf("Enter value to enqueue: ");

67

scanf("%d", &value);

68

enqueue(value);

69

break;

70

case 2:

71

dequeue();

72

73

case 3:

74

display();

75

break;

76

case 4:

77

return 0;

78

default:

79

printf("Invalid choice\n");

80

}

81

}

82

83

return 0;

84

}

85

Output

Enter your choice: 1
Enter value to enqueue: 10
Enqueued 10

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued 10

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 10
Invalid choice

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

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

=== Code Execution Successful ===

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

main.c

Share

Run

```
35 if (front == -1) {
36     printf("Queue is Empty\n");
37     return;
38 }
39
40 printf("Queue elements:\n");
41 int i = front;
42 while (1) {
43     printf("%d ", queue[i]);
44     if (i == rear)
45         break;
46     i = (i + 1) % SIZE;
47 }
48 printf("\n");
49 }
50
51 int main() {
52     int choice, value;
53
54     while (1) {
55         printf("\nCircular Queue Menu:\n");
56         printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
57         printf("Enter your choice: ");
58         scanf("%d", &choice);
59
60         switch (choice) {
61             case 1:
62                 printf("Enter value to enqueue: ");
63                 scanf("%d", &value);
64                 enqueue(value);
65                 break;
66             case 2:
67                 dequeue();
68                 break;
69             case 3:
70                 display();
71         }
72     }
73 }
```

Output

Enter your choice: 1
Enter value to enqueue: 10
Enqueued 10

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued 10

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Queue Underflow

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

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4

=== Code Execution Successful ===

```
main.c
46     i = (i + 1) % SIZE;
47 }
48 printf("\n");
49 }
50
51 int main() {
52     int choice, value;
53
54     while (1) {
55         printf("\nCircular Queue Menu:\n");
56         printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
57         printf("Enter your choice: ");
58         scanf("%d", &choice);
59
60         switch (choice) {
61             case 1:
62                 printf("Enter value to enqueue: ");
63                 scanf("%d", &value);
64                 enqueue(value);
65                 break;
66             case 2:
67                 dequeue();
68                 break;
69             case 3:
70                 display();
71                 break;
72             case 4:
73                 return 0;
74             default:
75                 printf("Invalid choice\n");
76         }
77     }
78
79     return 0;
80 }
81
```

Output

```
Enter value to enqueue: 10
Enqueued 10

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued 10

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Queue Underflow

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

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4

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

13. Write a program in C to implement insertion in a linked list(beg; mid; end)

main.c

Share

Run

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct Node {
4     int data;
5     struct Node* next;
6 };
7
8 struct Node* head = NULL;
9 void insertAtBeginning(int value) {
10     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
11     newNode->data = value;
12     newNode->next = head;
13     head = newNode;
14     printf("Inserted %d at beginning\n", value);
15 }
16
17 void insertAtEnd(int value) {
18     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
19     newNode->data = value;
20     newNode->next = NULL;
21
22     if (head == NULL) {
23         head = newNode;
24     } else {
25         struct Node* temp = head;
26         while (temp->next != NULL)
27             temp = temp->next;
28         temp->next = newNode;
29     }
30     printf("Inserted %d at end\n", value);
31 }
32
33 void insertAtPosition(int pos, int value) {
34     if (pos == 0) {
35         insertAtBeginning(value);
36         return;
37     }
38     struct Node* temp = head;
39     for (int i = 1; i < pos; i++)
40         temp = temp->next;
41     if (temp == NULL)
42         return;
43     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
44     newNode->data = value;
45     newNode->next = temp->next;
46     temp->next = newNode;
47 }
```

Output

Linked List Insertion Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 1
Enter value: 10
Inserted 10 at beginning

Linked List Insertion Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 2
Enter value: 20
Inserted 20 at end

Linked List Insertion Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 3
Enter position (starting from 0): 30
Enter value: 4
Invalid position

Linked List Insertion Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit

```
main.c
34 if (pos == 0) {
35     insertAtBeginning(value);
36     return;
37 }
38
39 struct Node* temp = head;
40 for (int i = 0; i < pos - 1 && temp != NULL; i++)
41     temp = temp->next;
42
43 if (temp == NULL) {
44     printf("Invalid position\n");
45     return;
46 }
47
48 struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
49 newNode->data = value;
50 newNode->next = temp->next;
51 temp->next = newNode;
52 printf("Inserted %d at position %d\n", value, pos);
53 }
54
55 void display() {
56     struct Node* temp = head;
57     printf("Linked List: ");
58     while (temp != NULL) {
59         printf("%d -> ", temp->data);
60         temp = temp->next;
61     }
62     printf("NULL\n");
63 }
64
65 int main() {
66     int choice, value, position;
67
68     while (1) {
69         printf("\nLinked List Insertion Menu:\n");
70         printf("1. Insert at Beginning\n");
71         printf("2. Insert at End\n");
72         printf("3. Insert at Position\n");
73         printf("4. Display\n");
74         printf("5. Exit\n");
75         printf("Enter choice: ");
76         int choice;
77         scanf("%d", &choice);
78         printf("Enter value: ");
79         int value;
80         scanf("%d", &value);
81         if (choice == 1) {
82             insertAtBeginning(value);
83         } else if (choice == 2) {
84             insertAtEnd(value);
85         } else if (choice == 3) {
86             insertAtPosition(value, position);
87         } else if (choice == 4) {
88             display();
89         } else if (choice == 5) {
90             break;
91         }
92     }
93 }
```

```

70     printf("1. Insert at Beginning\n2. Insert at End\n3. Insert at Position\n4. Display\n5.
    Exit\n");
71     printf("Enter choice: ");
72     scanf("%d", &choice);
73
74     switch (choice) {
75         case 1:
76             printf("Enter value: ");
77             scanf("%d", &value);
78             insertAtBeginning(value);
79             break;
80         case 2:
81             printf("Enter value: ");
82             scanf("%d", &value);
83             insertAtEnd(value);
84             break;
85         case 3:
86             printf("Enter position (starting from 0): ");
87             scanf("%d", &position);
88             printf("Enter value: ");
89             scanf("%d", &value);
90             insertAtPosition(position, value);
91             break;
92         case 4:
93             display();
94             break;
95         case 5:
96             return 0;
97         default:
98             printf("Invalid choice\n");
99     }
100 }
101
102 return 0;
103 }
104

```

Linked List Insertion Menu:

1. Insert at Beginning

2. Insert at End

3. Insert at Position

4. Display

5. Exit

Enter choice: 1

Enter value: 10

Inserted 10 at beginning

Linked List Insertion Menu:

1. Insert at Beginning

2. Insert at End

3. Insert at Position

4. Display

5. Exit

Enter choice: 2

Enter value: 20

Inserted 20 at end

Linked List Insertion Menu:

1. Insert at Beginning

2. Insert at End

3. Insert at Position

4. Display

5. Exit

Enter choice: 3

Enter position (starting from 0):

Enter value: 4

Invalid position

Linked List Insertion Menu:

1. Insert at Beginning

2. Insert at End

3. Insert at Position

14. Write a program in C to implement deletion from a linked list(beg; mid; end)

```
main.c | 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 | Run | Output
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct Node {
5     int data;
6     struct Node* next;
7 };
8
9 void printList(struct Node* head) {
10     struct Node* temp = head;
11     printf("List: ");
12     while (temp != NULL) {
13         printf("%d -> ", temp->data);
14         temp = temp->next;
15     }
16     printf("NULL\n");
17 }
18
19 void deleteBeginning(struct Node** head) {
20     if (*head == NULL) return;
21     struct Node* temp = *head;
22     *head = (*head)->next;
23     free(temp);
24 }
25
26 void deleteEnd(struct Node** head) {
27     if (*head == NULL) return;
28     if ((*head)->next == NULL) {
29         free(*head);
30         *head = NULL;
31         return;
32     }
33     struct Node* temp = *head;
34     while (temp->next->next != NULL)
35         temp = temp->next;
36     free(temp->next);
37     temp->next = NULL;
```

```
List: 10 -> 20 -> 30 -> 40 -> 50 -> NULL
List: 20 -> 30 -> 40 -> 50 -> NULL
List: 20 -> 40 -> 50 -> NULL
List: 20 -> 40 -> NULL
```

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

```
main.c | 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 | Run | Output
38 }
39
40 void deleteByValue(struct Node** head, int value) {
41     if (*head == NULL) return;
42     if ((*head)->data == value) {
43         deleteBeginning(head);
44         return;
45     }
46     struct Node* temp = *head;
47     while (temp->next != NULL && temp->next->data != value)
48         temp = temp->next;
49     if (temp->next == NULL) return;
50     struct Node* toDelete = temp->next;
51     temp->next = temp->next->next;
52     free(toDelete);
53 }
54
55 void insertEnd(struct Node** head, int data) {
56     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
57     newNode->data = data;
58     newNode->next = NULL;
59     if (*head == NULL) {
60         *head = newNode;
61         return;
62     }
63     struct Node* temp = *head;
64     while (temp->next != NULL)
65         temp = temp->next;
66     temp->next = newNode;
67 }
68
69 int main() {
70     struct Node* head = NULL;
71
72     insertEnd(&head, 10);
73     insertEnd(&head, 20);
```

```
List: 10 -> 20 -> 30 -> 40 -> 50 -> NULL
List: 20 -> 30 -> 40 -> 50 -> NULL
List: 20 -> 40 -> 50 -> NULL
List: 20 -> 40 -> NULL
```

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

```
main.c
50 struct Node* toDelete = temp->next;
51 temp->next = temp->next->next;
52 free(toDelete);
53 }
54
55 void insertEnd(struct Node** head, int data) {
56     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
57     newNode->data = data;
58     newNode->next = NULL;
59     if (*head == NULL) {
60         *head = newNode;
61         return;
62     }
63     struct Node* temp = *head;
64     while (temp->next != NULL)
65         temp = temp->next;
66     temp->next = newNode;
67 }
68
69 int main() {
70     struct Node* head = NULL;
71
72     insertEnd(&head, 10);
73     insertEnd(&head, 20);
74     insertEnd(&head, 30);
75     insertEnd(&head, 40);
76     insertEnd(&head, 50);
77     printList(head);
78
79     deleteBeginning(&head);
80     printList(head);
81     deleteByValue(&head, 30);
82     printList(head);
83     deleteEnd(&head);
84     printList(head);
85
86     return 0;
87 }
```

Output

```
List: 10 -> 20 -> 30 -> 40 -> 50 -> NULL
List: 20 -> 30 -> 40 -> 50 -> NULL
List: 20 -> 40 -> 50 -> NULL
List: 20 -> 40 -> NULL

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

15. Write a program in C to implement insertion in a circular linked list(beg; mid; end)

main.c	Run	Output
<pre>1 #include <stdio.h> 2 #include <stdlib.h> 3 struct Node { 4 int data; 5 struct Node* next; 6 }; 7 struct Node* createNode(int data) { 8 struct Node* newNode = (struct Node*)malloc(sizeof(struct Node)); 9 newNode->data = data; 10 newNode->next = NULL; 11 return newNode; 12 } 13 14 void insertEnd(struct Node** head, int data) { 15 struct Node* newNode = createNode(data); 16 if (*head == NULL) { 17 newNode->next = newNode; 18 *head = newNode; 19 return; 20 } 21 22 struct Node* temp = *head; 23 while (temp->next != *head) 24 temp = temp->next; 25 26 temp->next = newNode; 27 newNode->next = *head; 28 } 29 30 void insertBegin(struct Node** head, int data) { 31 struct Node* newNode = createNode(data); 32 if (*head == NULL) { 33 newNode->next = newNode; 34 *head = newNode; 35 return; 36 } 37 }</pre>	<div>Run</div>	<p>After inserting at end: 10 -> 20 -> 30 -> (back to 10) After inserting at beginning: 5 -> 10 -> 20 -> 30 -> (back to 5) After inserting 25 after 20: 5 -> 10 -> 20 -> 25 -> 30 -> (back to 5)</p> <p>=== Code Execution Successful ===</p>

main.c	Run	Output
<pre>38 struct Node* temp = *head; 39 while (temp->next != *head) 40 temp = temp->next; 41 42 newNode->next = *head; 43 temp->next = newNode; 44 *head = newNode; 45 } 46 47 void insertAfter(struct Node* head, int key, int data) { 48 struct Node* temp = head; 49 do { 50 if (temp->data == key) { 51 struct Node* newNode = createNode(data); 52 newNode->next = temp->next; 53 temp->next = newNode; 54 return; 55 } 56 temp = temp->next; 57 } while (temp != head); 58 59 printf("Node with value %d not found.\n", key); 60 } 61 62 void display(struct Node* head) { 63 if (head == NULL) { 64 printf("List is empty.\n"); 65 return; 66 } 67 struct Node* temp = head; 68 do { 69 printf("%d -> ", temp->data); 70 temp = temp->next; 71 } while (temp != head); 72 printf("(back to %d)\n", head->data); 73 } 74 75 int main() {</pre>	<div>Run</div>	<p>After inserting at end: 10 -> 20 -> 30 -> (back to 10) After inserting at beginning: 5 -> 10 -> 20 -> 30 -> (back to 5) After inserting 25 after 20: 5 -> 10 -> 20 -> 25 -> 30 -> (back to 5)</p> <p>=== Code Execution Successful ===</p>

```

54     return;
55 }
56 temp = temp->next;
57 } while (temp != head);
58
59 printf("Node with value %d not found.\n", key);
60 }
61 void display(struct Node* head) {
62     if (head == NULL) {
63         printf("List is empty.\n");
64         return;
65     }
66     struct Node* temp = head;
67     do {
68         printf("%d -> ", temp->data);
69         temp = temp->next;
70     } while (temp != head);
71     printf("(back to %d)\n", head->data);
72 }
73 int main() {
74     struct Node* head = NULL;
75
76     insertEnd(&head, 10);
77     insertEnd(&head, 20);
78     insertEnd(&head, 30);
79     printf("After inserting at end:\n");
80     display(head);
81
82     insertBegin(&head, 5);
83     printf("After inserting at beginning:\n");
84     display(head);
85
86     insertAfter(head, 20, 25);
87     printf("After inserting 25 after 20:\n");
88     display(head);
89
90     return 0;

```

After inserting at end:
 10 -> 20 -> 30 -> (back to 10)
 After inserting at beginning:
 5 -> 10 -> 20 -> 30 -> (back to 5)
 After inserting 25 after 20:
 5 -> 10 -> 20 -> 25 -> 30 -> (back to 5)

=== Code Execution Successful ===

16. Write a program in C to implement deletion from a circular linked list(beg; mid; end)


```
88     if (head == NULL) {
89         printf("List is empty.\n");
90         return;
91     }
92     struct Node* temp = head;
93     do {
94         printf("%d -> ", temp->data);
95         temp = temp->next;
96     } while (temp != head);
97     printf("(back to %d)\n", head->data);
98 }
99 int main() {
100     struct Node* head = NULL;
101
102     insertEnd(&head, 10);
103     insertEnd(&head, 20);
104     insertEnd(&head, 30);
105     insertEnd(&head, 40);
106     printf("Original list:\n");
107     display(head);
108
109     deleteBegin(&head);
110     printf("After deleting from beginning:\n");
111     display(head);
112
113     deleteEnd(&head);
114     printf("After deleting from end:\n");
115     display(head);
116
117     deleteByValue(&head, 20);
118     printf("After deleting 20 (middle):\n");
119     display(head);
120
121     return 0;
122 }
123
```

^ Original list:
10 -> 20 -> 30 -> 40 -> (back to 10)
After deleting from beginning:
20 -> 30 -> 40 -> (back to 20)
After deleting from end:
20 -> 30 -> (back to 20)
After deleting 20 (middle):
30 -> (back to 30)

=== Code Execution Successful ===

17. Write a program in C to implement insertion in a doubly linked

main.c

Share

Run

Output

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 struct Node {
4     int data;
5     struct Node* prev;
6     struct Node* next;
7 };
8 struct Node* createNode(int data) {
9     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
10    newNode->data = data;
11    newNode->prev = newNode->next = NULL;
12    return newNode;
13 }
14 void insertBegin(struct Node** head, int data) {
15     struct Node* newNode = createNode(data);
16     if (*head == NULL) {
17         *head = newNode;
18         return;
19     }
20     newNode->next = *head;
21     (*head)->prev = newNode;
22     *head = newNode;
23 }
24 void insertEnd(struct Node** head, int data) {
25     struct Node* newNode = createNode(data);
26     if (*head == NULL) {
27         *head = newNode;
28         return;
29     }
30     struct Node* temp = *head;
31     while (temp->next != NULL)
32         temp = temp->next;
33
34     temp->next = newNode;
35     newNode->prev = temp;
36 }

```

main.c

Share

Run

Output

```

47 struct Node* newNode = createNode(data);
48 newNode->next = temp->next;
49 newNode->prev = temp;
50 if (temp->next != NULL)
51     temp->next->prev = newNode;
52 temp->next = newNode;
53 }
54 void display(struct Node* head) {
55     struct Node* temp = head;
56     printf("Doubly Linked List: ");
57     while (temp != NULL) {
58         printf("%d <-> ", temp->data);
59         temp = temp->next;
60     }
61     printf("NULL\n");
62 }
63 int main() {
64     struct Node* head = NULL;
65
66     insertEnd(&head, 10);
67     insertEnd(&head, 20);
68     insertEnd(&head, 30);
69     printf("After inserting at end:\n");
70     display(head);
71
72     insertBegin(&head, 5);
73     printf("After inserting at beginning:\n");
74     display(head);
75
76     insertAfter(head, 20, 25);
77     printf("After inserting 25 after 20:\n");
78     display(head);
79
80     return 0;
81 }
82

```

main.c

Share

Run

Output

```

After inserting at end:
Doubly Linked List: 10 <-> 20 <-> 30 <-> NULL
After inserting at beginning:
Doubly Linked List: 5 <-> 10 <-> 20 <-> 30 <-> NULL
After inserting 25 after 20:
Doubly Linked List: 5 <-> 10 <-> 20 <-> 25 <-> 30 <-> NULL

=== Code Execution Successful ===

```

18. Write a program in C to implement deletion from a doubly linked list

```
main.c | Run | Output
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct Node {
4     int data;
5     struct Node* prev;
6     struct Node* next;
7 };
8 struct Node* createNode(int data) {
9     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
10    newNode->data = data;
11    newNode->prev = newNode->next = NULL;
12    return newNode;
13 }
14 void insertEnd(struct Node** head, int data) {
15     struct Node* newNode = createNode(data);
16     if (*head == NULL) {
17         *head = newNode;
18         return;
19     }
20     struct Node* temp = *head;
21     while (temp->next != NULL)
22         temp = temp->next;
23     temp->next = newNode;
24     newNode->prev = temp;
25 }
26 void deleteBegin(struct Node** head) {
27     if (*head == NULL) return;
28
29     struct Node* temp = *head;
30     *head = temp->next;
31     if (*head != NULL)
32         (*head)->prev = NULL;
33
34     free(temp);
35 }
36 void deleteEnd(struct Node** head) {
37     if (*head == NULL) return;
38     struct Node* temp = *head;
39     while (temp->next != NULL)
40         temp = temp->next;
41     temp->prev->next = NULL;
42     free(temp);
43 }
44 void deleteByValue(struct Node** head, int key) {
45     if (*head == NULL) return;
46     struct Node* temp = *head;
47     if (temp->data == key) {
48         deleteBegin(head);
49         return;
50     }
51     while (temp != NULL && temp->data != key)
52         temp = temp->next;
53     if (temp == NULL) {
54         printf("Node with value %d not found.\n", key);
55         return;
56     }
57     if (temp->next != NULL)
58         temp->next->prev = temp->prev;
59     if (temp->prev != NULL)
60         temp->prev->next = temp->next;
61     free(temp);
62 }
```

Original list:
Doubly Linked List: 10 <-> 20 <-> 30 <-> 40 <-> NULL
After deleting from beginning:
Doubly Linked List: 20 <-> 30 <-> 40 <-> NULL
After deleting from end:
Doubly Linked List: 20 <-> 30 <-> NULL
After deleting 20 (middle):
Doubly Linked List: 30 <-> NULL
=== Code Execution Successful ===

```
main.c | Run | Output
38
39 struct Node* temp = *head;
40 if (temp->next == NULL) {
41     free(temp);
42     *head = NULL;
43     return;
44 }
45 while (temp->next != NULL)
46     temp = temp->next;
47 temp->prev->next = NULL;
48 free(temp);
49 }
50 void deleteByValue(struct Node** head, int key) {
51     if (*head == NULL) return;
52     struct Node* temp = *head;
53     if (temp->data == key) {
54         deleteBegin(head);
55         return;
56     }
57     while (temp != NULL && temp->data != key)
58         temp = temp->next;
59     if (temp == NULL) {
60         printf("Node with value %d not found.\n", key);
61         return;
62     }
63     if (temp->next != NULL)
64         temp->next->prev = temp->prev;
65     if (temp->prev != NULL)
66         temp->prev->next = temp->next;
67     free(temp);
68 }
```

Original list:
Doubly Linked List: 10 <-> 20 <-> 30 <-> 40 <-> NULL
After deleting from beginning:
Doubly Linked List: 20 <-> 30 <-> 40 <-> NULL
After deleting from end:
Doubly Linked List: 20 <-> 30 <-> NULL
After deleting 20 (middle):
Doubly Linked List: 30 <-> NULL
=== Code Execution Successful ===

```
main.c
75 }
76 void display(struct Node* head) {
77     struct Node* temp = head;
78     printf("Doubly Linked List: ");
79     while (temp != NULL) {
80         printf("%d <-> ", temp->data);
81         temp = temp->next;
82     }
83     printf("NULL\n");
84 }
85
86 int main() {
87     struct Node* head = NULL;
88
89     insertEnd(&head, 10);
90     insertEnd(&head, 20);
91     insertEnd(&head, 30);
92     insertEnd(&head, 40);
93     printf("Original list:\n");
94     display(head);
95
96     deleteBegin(&head);
97     printf("After deleting from beginning:\n");
98     display(head);
99
100    deleteEnd(&head);
101    printf("After deleting from end:\n");
102    display(head);
103
104    deleteByValue(&head, 20);
105    printf("After deleting 20 (middle):\n");
106    display(head);
107
108    return 0;
109 }
110
```

Output

Original list:
Doubly Linked List: 10 <-> 20 <-> 30 <-> 40 <-> NULL
After deleting from beginning:
Doubly Linked List: 20 <-> 30 <-> 40 <-> NULL
After deleting from end:
Doubly Linked List: 20 <-> 30 <-> NULL
After deleting 20 (middle):
Doubly Linked List: 30 <-> NULL

=== Code Execution Successful ===

19. Write a program in C to implement insertion in Binary tree

main.c	Run	Output
<pre>1 #include <stdio.h> 2 #include <stdlib.h> 3 struct Node { 4 int data; 5 struct Node* left; 6 struct Node* right; 7 }; 8 struct Queue { 9 int front, rear, size; 10 struct Node** array; 11 }; 12 struct Node* createNode(int data) { 13 struct Node* newNode = (struct Node*)malloc(sizeof(struct Node)); 14 newNode->data = data; 15 newNode->left = newNode->right = NULL; 16 return newNode; 17 } 18 struct Queue* createQueue(int size) { 19 struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue)); 20 q->front = q->rear = -1; 21 q->size = size; 22 q->array = (struct Node**)malloc(size * sizeof(struct Node*)); 23 return q; 24 } 25 int isEmpty(struct Queue* q) { 26 return q->front == -1; 27 } 28 29 void enqueue(struct Queue* q, struct Node* node) { 30 if (q->rear == q->size - 1) return; 31 if (isEmpty(q)) q->front = 0; 32 q->array[++q->rear] = node; 33 } 34 35 struct Node* dequeue(struct Queue* q) { 36 if (isEmpty(q)) return NULL; 37 }</pre>	<div>Run</div>	<div>Inorder traversal of Binary Tree: 40 20 50 10 30</div> <div>=== Code Execution Successful ===</div>

main.c

Share

Run

```
36     if (isEmpty(q)) return NULL;
37     struct Node* temp = q->array[q->front];
38     if (q->front == q->rear)
39     {
40         q->front = q->rear = -1;
41     }
42     else
43     {
44         q->front++;
45     }
46     return temp;
47 }
48
49 void insert(struct Node** root, int data) {
50     struct Node* newNode = createNode(data);
51     if (*root == NULL) {
52         *root = newNode;
53         return;
54     }
55
56     struct Queue* q = createQueue(100);
57     enqueue(q, *root);
58
59     while (!isEmpty(q)) {
60         struct Node* temp = dequeue(q);
61
62         if (temp->left == NULL) {
63             temp->left = newNode;
64             break;
65         }
66         else {
67             enqueue(q, temp->left);
68         }
69
70         if (temp->right == NULL) {
71             temp->right = newNode;
72             break;
73         }
74         else {
75             enqueue(q, temp->right);
76         }
77     }
78 }
```

Output

^ Inorder traversal of Binary Tree:

40 20 50 10 30

=== Code Execution Successful ===

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

58     temp->left = newNode;
59     break;
60 } else {
61     enqueue(q, temp->left);
62 }
63
64 if (temp->right == NULL) {
65     temp->right = newNode;
66     break;
67 } else {
68     enqueue(q, temp->right);
69 }
70 }
71 }
72 void inorder(struct Node* root) {
73     if (root == NULL) return;
74     inorder(root->left);
75     printf("%d ", root->data);
76     inorder(root->right);
77 }
78
79 int main() {
80     struct Node* root = NULL;
81
82     insert(&root, 10);
83     insert(&root, 20);
84     insert(&root, 30);
85     insert(&root, 40);
86     insert(&root, 50);
87
88     printf("Inorder traversal of Binary Tree:\n");
89     inorder(root);
90
91     return 0;
92 }
93
```

Output

^ Inorder traversal of Binary Tree:
40 20 50 10 30

=== Code Execution Successful ===

20. Write a program in C to implement deletion from Binary tree

main.c	Run	Output
<pre>1 #include <stdio.h> 2 #include <stdlib.h> 3 struct Node { 4 int data; 5 struct Node* left; 6 struct Node* right; 7 }; 8 struct Node* createNode(int data) { 9 struct Node* newNode = (struct Node*)malloc(sizeof(struct Node)); 10 newNode->data = data; 11 newNode->left = newNode->right = NULL; 12 return newNode; 13 } 14 void inorder(struct Node* root) { 15 if (root == NULL) return; 16 inorder(root->left); 17 printf("%d ", root->data); 18 inorder(root->right); 19 } 20 void deleteDeepest(struct Node* root, struct Node* delNode) { 21 if (root == NULL) return; 22 23 if (root->left == delNode) { 24 root->left = NULL; 25 free(delNode); 26 return; 27 } else if (root->right == delNode) { 28 root->right = NULL; 29 free(delNode); 30 return; 31 } 32 33 if (root->left) 34 deleteDeepest(root->left, delNode); 35 if (root->right) 36 deleteDeepest(root->right, delNode); 37 }</pre>	<div>Run</div>	<p>Inorder traversal before deletion: 40 20 50 10 30</p> <p>Deleting node 20...</p> <p>Inorder traversal after deletion: 40 50 10 30</p> <p>=== Code Execution Successful ===</p>

main.c	Run	Output
<pre>35 if (root->right) 36 deleteDeepest(root->right, delNode); 37 } 38 void deleteNode(struct Node** root, int key) { 39 if (*root == NULL) return; 40 41 if ((*root)->left == NULL && (*root)->right == NULL) { 42 if ((*root)->data == key) { 43 free(*root); 44 *root = NULL; 45 } 46 return; 47 } 48 49 struct Node *keyNode = NULL, *temp; 50 struct Node* queue[100]; 51 int front = 0, rear = 0; 52 53 queue[rear++] = *root; 54 55 while (front < rear) { 56 temp = queue[front++]; 57 58 if (temp->data == key) 59 keyNode = temp; 60 61 if (temp->left) 62 queue[rear++] = temp->left; 63 if (temp->right) 64 queue[rear++] = temp->right; 65 } 66 67 if (keyNode) { 68 int x = temp->data; 69 deleteDeepest(*root, temp); 70 keyNode->data = x; 71 }</pre>	<div>Run</div>	<p>Inorder traversal before deletion: 40 20 50 10 30</p> <p>Deleting node 20...</p> <p>Inorder traversal after deletion: 40 50 10 30</p> <p>=== Code Execution Successful ===</p>

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

89  if (!temp->left) {
90      temp->left = newNode;
91      return;
92  } else {
93      queue[rear++] = temp->left;
94  }
95
96  if (!temp->right) {
97      temp->right = newNode;
98      return;
99  } else {
100     queue[rear++] = temp->right;
101  }
102  }
103  }
104  int main() {
105     struct Node* root = NULL;
106
107     insert(&root, 10);
108     insert(&root, 20);
109     insert(&root, 30);
110     insert(&root, 40);
111     insert(&root, 50);
112
113     printf("Inorder traversal before deletion:\n");
114     inorder(root);
115
116     printf("\n\nDeleting node 20...\n");
117     deleteNode(&root, 20);
118
119     printf("Inorder traversal after deletion:\n");
120     inorder(root);
121
122     return 0;
123  }
124
```

Output

^ Inorder traversal before deletion:
40 20 50 10 30

Deleting node 20...

Inorder traversal after deletion:
40 50 10 30

=== Code Execution Successful ===

21. Write a program in C to implement recursive tree traversals \

main.c



Share

Run

Output

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct Node {
4     int data;
5     struct Node* left;
6     struct Node* right;
7 };
8 struct Node* createNode(int data) {
9     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
10    newNode->data = data;
11    newNode->left = newNode->right = NULL;
12    return newNode;
13 }
14 void preorder(struct Node* root) {
15     if (root == NULL) return;
16     printf("%d ", root->data);
17     preorder(root->left);
18     preorder(root->right);
19 }
20
21 void inorder(struct Node* root) {
22     if (root == NULL) return;
23     inorder(root->left);
24     printf("%d ", root->data);
25     inorder(root->right);
26 }
27 void postorder(struct Node* root) {
28     if (root == NULL) return;
29     postorder(root->left);
30     postorder(root->right);
31     printf("%d ", root->data);
32 }
33 int main() {
34     struct Node* root = createNode(1);
35     root->left = createNode(2);
36     root->right = createNode(3);
37 }
```

Preorder traversal: 1 2 4 5 3
Inorder traversal: 4 2 5 1 3
Postorder traversal: 4 5 2 3 1

=== Code Execution Successful ===

main.c

Share

Run

```
19 }
20
21 void inorder(struct Node* root) {
22     if (root == NULL) return;
23     inorder(root->left);
24     printf("%d ", root->data);
25     inorder(root->right);
26 }
27 void postorder(struct Node* root) {
28     if (root == NULL) return;
29     postorder(root->left);
30     postorder(root->right);
31     printf("%d ", root->data);
32 }
33 int main() {
34     struct Node* root = createNode(1);
35     root->left = createNode(2);
36     root->right = createNode(3);
37     root->left->left = createNode(4);
38     root->left->right = createNode(5);
39
40     printf("Preorder traversal: ");
41     preorder(root);
42     printf("\n");
43
44     printf("Inorder traversal: ");
45     inorder(root);
46     printf("\n");
47
48     printf("Postorder traversal: ");
49     postorder(root);
50     printf("\n");
51
52     return 0;
53 }
54
```

Preorder traversal: 1 2 4 5 3
Inorder traversal: 4 2 5 1 3
Postorder traversal: 4 5 2 3 1

=== Code Execution Successful ===