

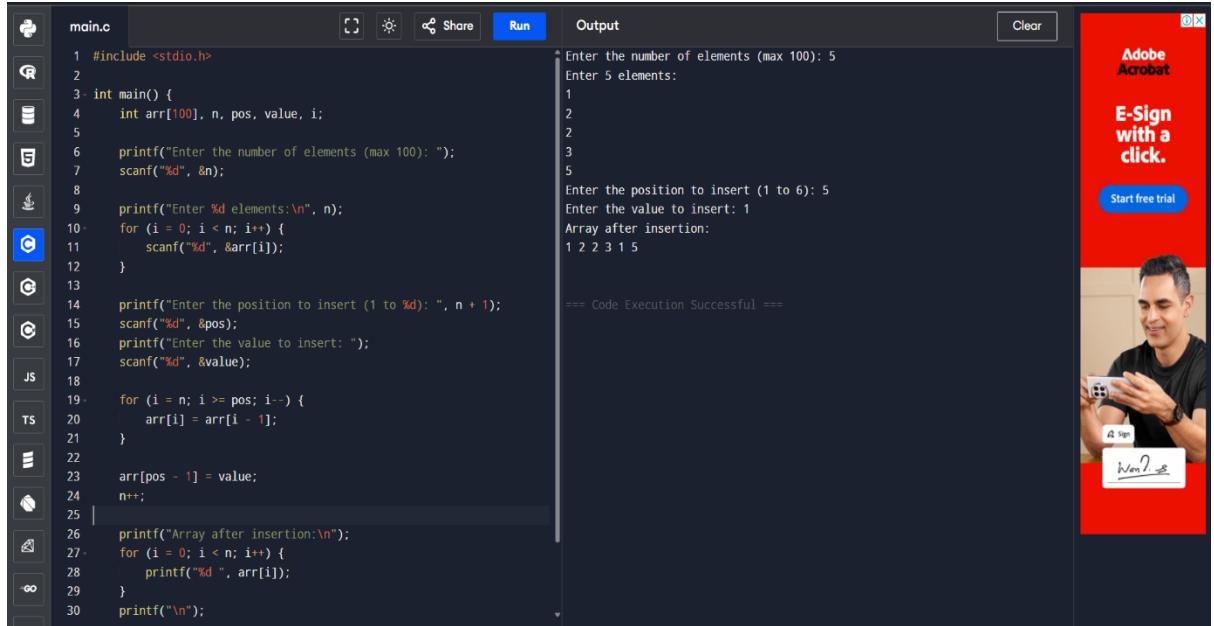
DATA STRUCTURE LAB (4.5CA251C01:)

#Questions for Programs:

1. Write a program in C to implement insertion in 1-D Arrays
2. Write a program in C to implement deletion in 1-D Arrays
3. Write a program in C to implement linear and binary searching in 1-D Arrays
4. Write a program in C to implement sorting in 1-D Arrays
5. Write a program in C to concatenate two arrays
6. Write a program in C to implement the following Operations on 2-D Array (addition;
subtraction; multiplication; transpose)
7. Write a program in C to implement operations on Stack using array
8. Write a program in C to implement operations on Stack using linked list
9. Write a program in C to implement applications of Stack
10. Write a program in C to implement operations on queue using array
11. Write a program in C to implement operations on queue using linked list
12. Write a program in C to implement operations on circular queue using array
13. Write a program in C to implement insertion in a linked list(beg;
mid; end)

14. Write a program in C to implement deletion from a linked list(beg; mid; end)
15. Write a program in C to implement insertion in a circular linked list(beg; mid; end)
16. Write a program in C to implement deletion from a circular linked list(beg; mid; end)
17. Write a program in C to implement insertion in a doubly linked list(beg; mid; end)
18. Write a program in C to implement deletion from a doubly linked list(beg; mid; end)
19. Write a program in C to implement insertion in Binary tree
20. Write a program in C to implement deletion from Binary tree
21. Write a program in C to implement recursive tree traversals (Inorder; Preorder; Postorder)
22. Write a program in C to Sort a list using Bubble Sort
23. Write a program in C to Sort a list using Selection Sort
24. Write a program in C to sort a list using Quick Sort
25. Write a program in C to sort a list using Merge Sort
26. Write a program in C to sort a list using Insertion Sort
27. Write a program in C to sort a list using Heap Sort

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

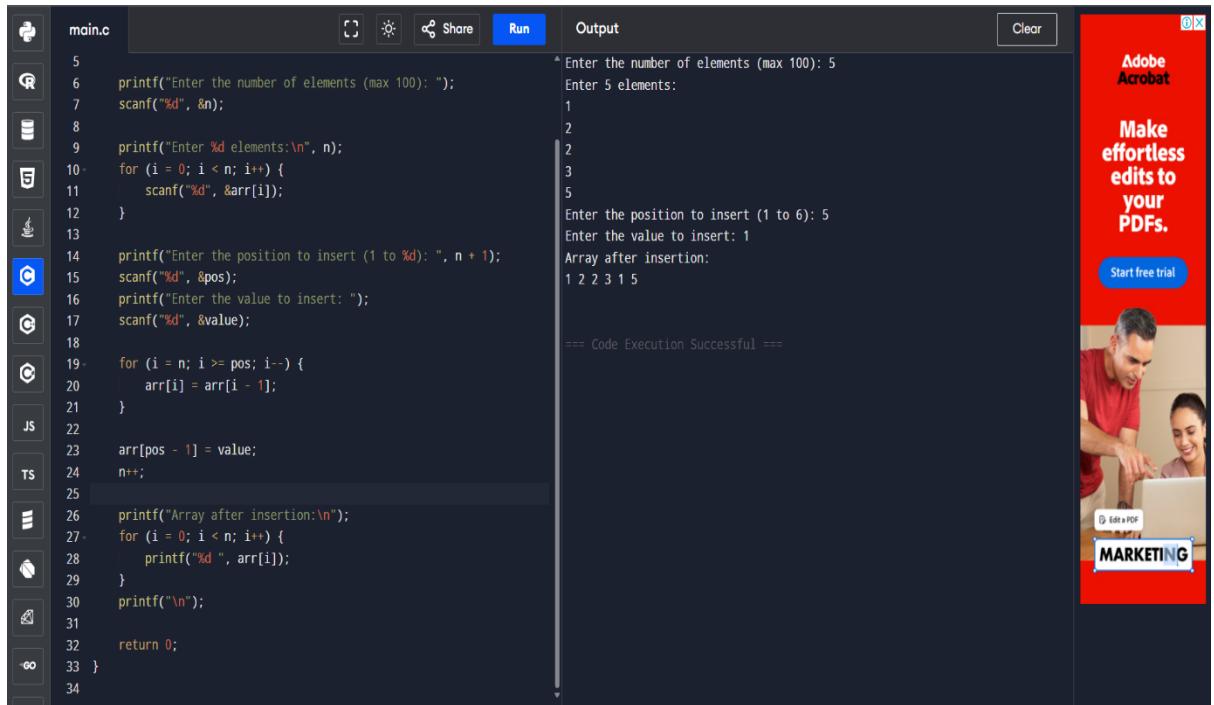


```
main.c
1 #include <stdio.h>
2
3 int main() {
4     int arr[100], n, pos, value, i;
5
6     printf("Enter the number of elements (max 100): ");
7     scanf("%d", &n);
8
9     printf("Enter %d elements:\n", n);
10    for (i = 0; i < n; i++) {
11        scanf("%d", &arr[i]);
12    }
13
14    printf("Enter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

Output

```
* Enter the number of elements (max 100): 5
Enter 5 elements:
1
2
3
5
Enter the position to insert (1 to 6): 1
Enter the value to insert: 1
Array after insertion:
1 2 2 3 1 5
*** Code Execution Successful ***
```

Right sidebar: Adobe Acrobat E-Sign with a click. Start free trial. A man holding a smartphone with a handwritten note "Wor2".



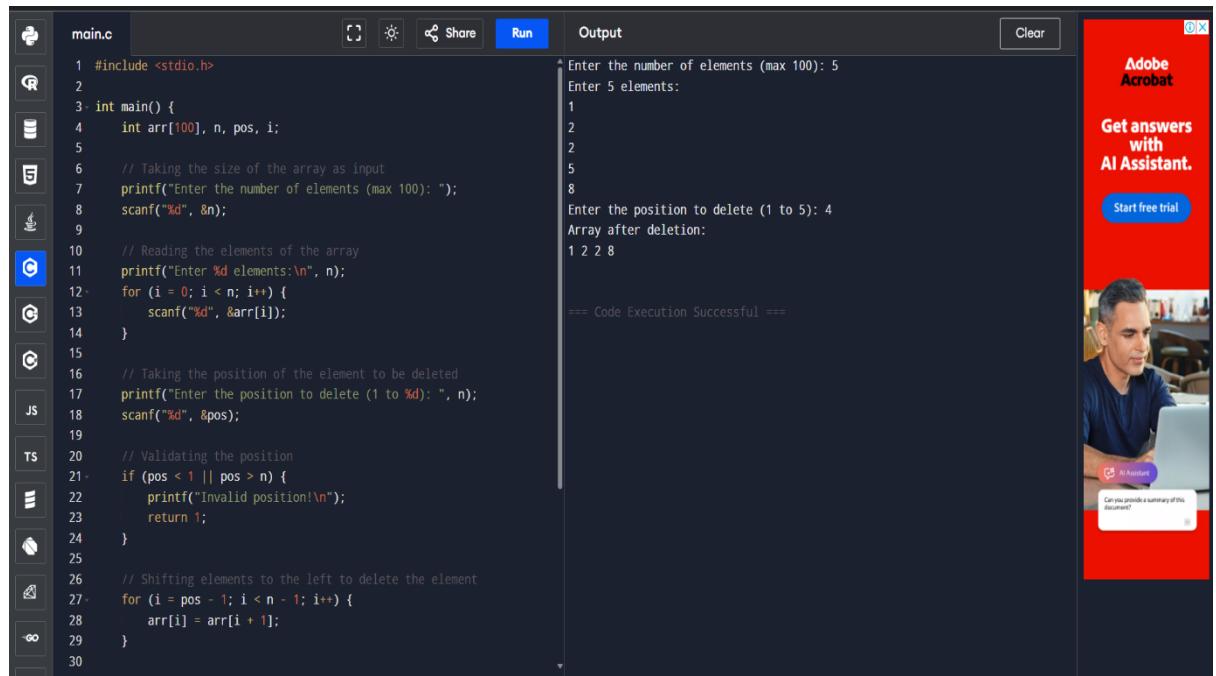
```
main.c
5
6     printf("Enter the number of elements (max 100): ");
7     scanf("%d", &n);
8
9     printf("Enter %d elements:\n", n);
10    for (i = 0; i < n; i++) {
11        scanf("%d", &arr[i]);
12    }
13
14    printf("Enter the position to insert (1 to %d): ", n + 1);
15    scanf("%d", &pos);
16    printf("Enter the value to insert: ");
17    scanf("%d", &value);
18
19    for (i = n; i >= pos; i--) {
20        arr[i] = arr[i - 1];
21    }
22
23    arr[pos - 1] = value;
24    n++;
25
26    printf("Array after insertion:\n");
27    for (i = 0; i < n; i++) {
28        printf("%d ", arr[i]);
29    }
30    printf("\n");
31
32    return 0;
33 }
```

Output

```
* Enter the number of elements (max 100): 5
Enter 5 elements:
1
2
3
5
Enter the position to insert (1 to 6): 5
Enter the value to insert: 1
Array after insertion:
1 2 2 3 1 5
*** Code Execution Successful ***
```

Right sidebar: Adobe Acrobat Make effortless edits to your PDFs. Start free trial. A man and a woman looking at a laptop screen with the word "MARKETING".

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

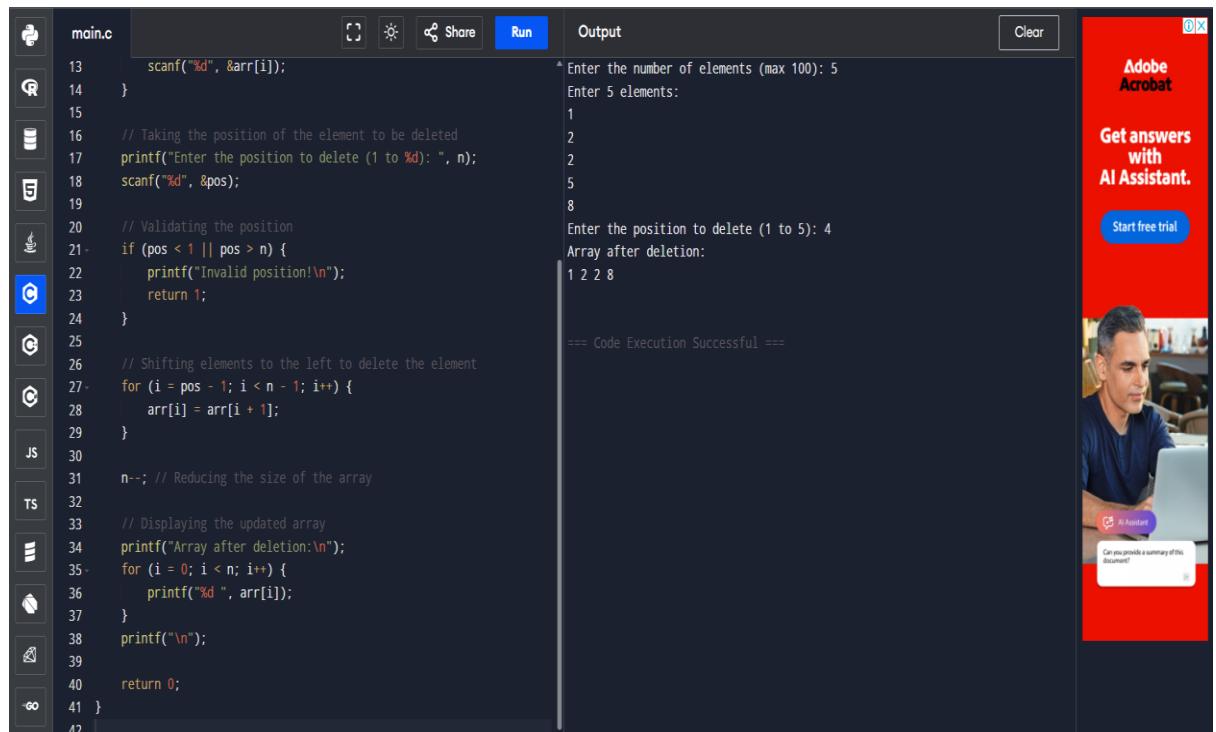


```
main.c
1 #include <stdio.h>
2
3 int main() {
4     int arr[100], n, pos, i;
5
6     // Taking the size of the array as input
7     printf("Enter the number of elements (max 100): ");
8     scanf("%d", &n);
9
10    // Reading the elements of the array
11    printf("Enter %d elements:\n", n);
12    for (i = 0; i < n; i++) {
13        scanf("%d", &arr[i]);
14    }
15
16    // Taking the position of the element to be deleted
17    printf("Enter the position to delete (1 to %d): ", n);
18    scanf("%d", &pos);
19
20    // Validating the position
21    if (pos < 1 || pos > n) {
22        printf("Invalid position!\n");
23        return 1;
24    }
25
26    // Shifting elements to the left to delete the element
27    for (i = pos - 1; i < n - 1; i++) {
28        arr[i] = arr[i + 1];
29    }
30
31    n--; // Reducing the size of the array
32
33    // Displaying the updated array
34    printf("Array after deletion:\n");
35    for (i = 0; i < n; i++) {
36        printf("%d ", arr[i]);
37    }
38    printf("\n");
39
40    return 0;
41 }
```

Output

```
Enter the number of elements (max 100): 5
Enter 5 elements:
1
2
2
5
8
Enter the position to delete (1 to 5): 4
Array after deletion:
1 2 2 8
==== Code Execution Successful ====
```

Adobe Acrobat
Get answers with AI Assistant.
Start free trial



```
main.c
13     scanf("%d", &arr[i]);
14 }
15
16 // Taking the position of the element to be deleted
17 printf("Enter the position to delete (1 to %d): ", n);
18 scanf("%d", &pos);
19
20 // Validating the position
21 if (pos < 1 || pos > n) {
22     printf("Invalid position!\n");
23     return 1;
24 }
25
26 // Shifting elements to the left to delete the element
27 for (i = pos - 1; i < n - 1; i++) {
28     arr[i] = arr[i + 1];
29 }
30
31 n--; // Reducing the size of the array
32
33 // Displaying the updated array
34 printf("Array after deletion:\n");
35 for (i = 0; i < n; i++) {
36     printf("%d ", arr[i]);
37 }
38 printf("\n");
39
40 return 0;
41 }
```

Output

```
Enter the number of elements (max 100): 5
Enter 5 elements:
1
2
2
5
8
Enter the position to delete (1 to 5): 4
Array after deletion:
1 2 2 8
==== Code Execution Successful ====
```

Adobe Acrobat
Get answers with AI Assistant.
Start free trial



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

The screenshot shows a C code editor interface with a dark theme. The left pane displays the source code for `main.c`, which includes functions for linear search and binary search. The right pane shows the output window where the user enters 4 elements (1, 2, 3, 4) and searches for element 1, resulting in both search methods finding it at position 1. A success message is displayed at the end.

```
1 #include <stdio.h>
2
3 // Function for Linear Search
4 void linearSearch(int arr[], int n, int key) {
5     for (int i = 0; i < n; i++) {
6         if (arr[i] == key) {
7             printf("Element %d found at position %d (Linear Search)\n", key, i + 1);
8             return;
9         }
10    }
11    printf("Element %d not found (Linear Search)\n", key);
12 }
13
14 // Function for Binary Search (Array must be sorted)
15 int binarySearch(int arr[], int n, int key) {
16     int left = 0, right = n - 1, mid;
17     while (left <= right) {
18         mid = (left + right) / 2;
19         if (arr[mid] == key) return mid; // Element found
20         else if (arr[mid] < key) left = mid + 1;
21         else right = mid - 1;
22     }
23     return -1; // Element not found
24 }
25
26 // Main Function
27 int main() {
28     int arr[100], n, key, i, pos;
29 }
```

Output:

```
Enter number of elements (max 100): 4
Enter 4 elements:
1
2
3
4
Enter the element to search: 1
Element 1 found at position 1 (Linear Search)
Element 1 found at position 1 (Binary Search)

== Code Execution Successful ==
```

The screenshot shows a C code editor interface with a dark theme. The left pane displays the source code for `main.c`, which includes input for the search key, sorting the array using bubble sort, and then performing linear and binary search. The right pane shows the output window where the user enters 4 elements (1, 2, 3, 4) and searches for element 1, resulting in both search methods finding it at position 1. A success message is displayed at the end. An advertisement for Amazon Prime featuring audio gadgets is visible on the right side.

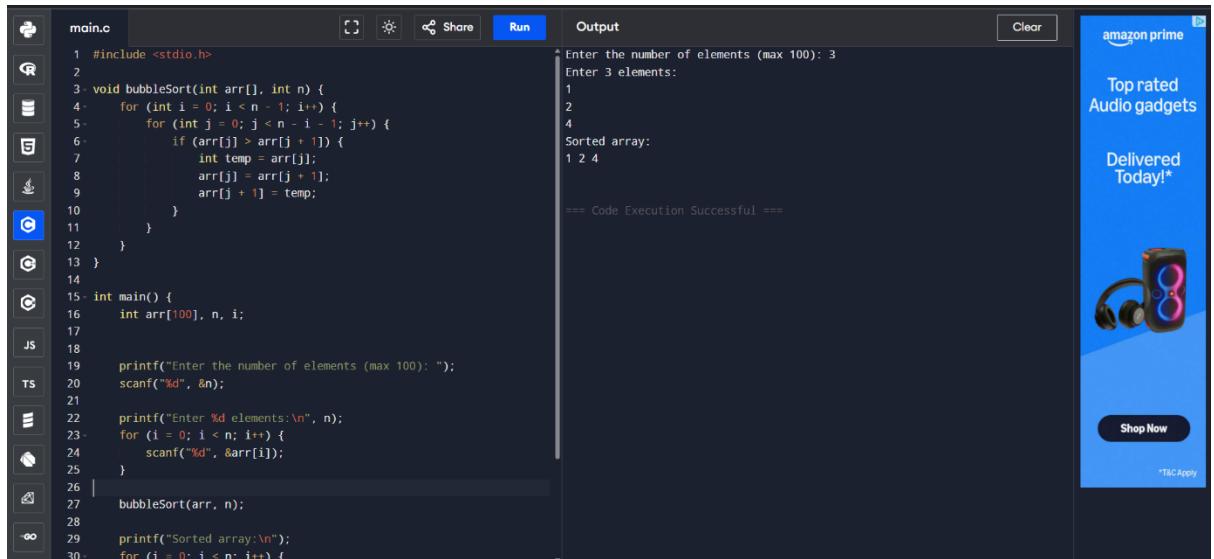
```
35
36 // Input search key
37 printf("Enter the element to search: ");
38 scanf("%d", &key);
39
40 // Linear Search
41 linearSearch(arr, n, key);
42
43 // Sorting array for Binary Search (Simple Bubble Sort)
44 for (i = 0; i < n - 1; i++) {
45     for (int j = 0; j < n - i - 1; j++) {
46         if (arr[j] > arr[j + 1]) {
47             int temp = arr[j];
48             arr[j] = arr[j + 1];
49             arr[j + 1] = temp;
50         }
51     }
52 }
53
54 // Binary Search
55 pos = binarySearch(arr, n, key);
56 if (pos != -1)
57     printf("Element %d found at position %d (Binary Search)\n",
58            key, pos + 1);
59 else
60     printf("Element %d not found (Binary Search)\n", key);
61
62 }
63 }
```

Output:

```
Enter number of elements (max 100): 4
Enter 4 elements:
1
2
3
4
Enter the element to search: 1
Element 1 found at position 1 (Linear Search)
Element 1 found at position 1 (Binary Search)

== Code Execution Successful ==
```

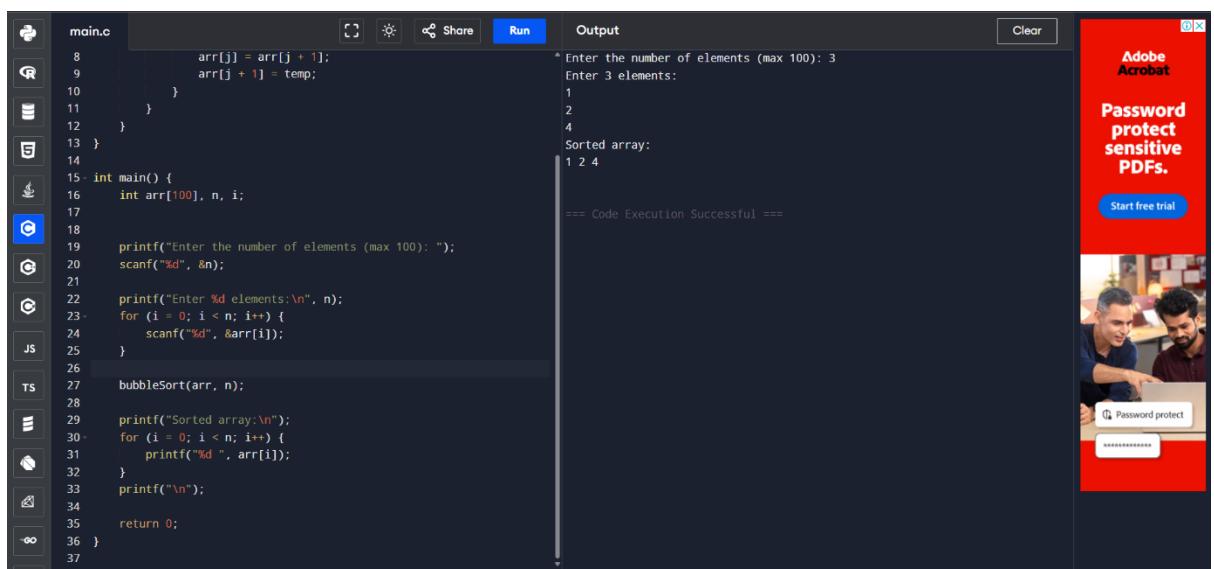
4. Write a program in C to implement sorting in 1-D Arrays.



```
main.c
1 #include <stdio.h>
2
3 void bubbleSort(int arr[], int n) {
4     for (int i = 0; i < n - i - 1; i++) {
5         for (int j = 0; j < n - i - 1; j++) {
6             if (arr[j] > arr[j + 1]) {
7                 int temp = arr[j];
8                 arr[j] = arr[j + 1];
9                 arr[j + 1] = temp;
10            }
11        }
12    }
13 }
14
15 int main() {
16     int arr[100], n, i;
17
18     printf("Enter the number of elements (max 100): ");
19     scanf("%d", &n);
20
21     printf("Enter %d elements:\n", n);
22     for (i = 0; i < n; i++) {
23         scanf("%d", &arr[i]);
24     }
25
26     bubbleSort(arr, n);
27
28     printf("Sorted array:\n");
29     for (i = 0; i < n; i++) {
30
31
32
33
34
35
36 }
```

Output

```
Enter the number of elements (max 100): 3
Enter 3 elements:
1
2
4
Sorted array:
1 2 4
== Code Execution Successful ==
```

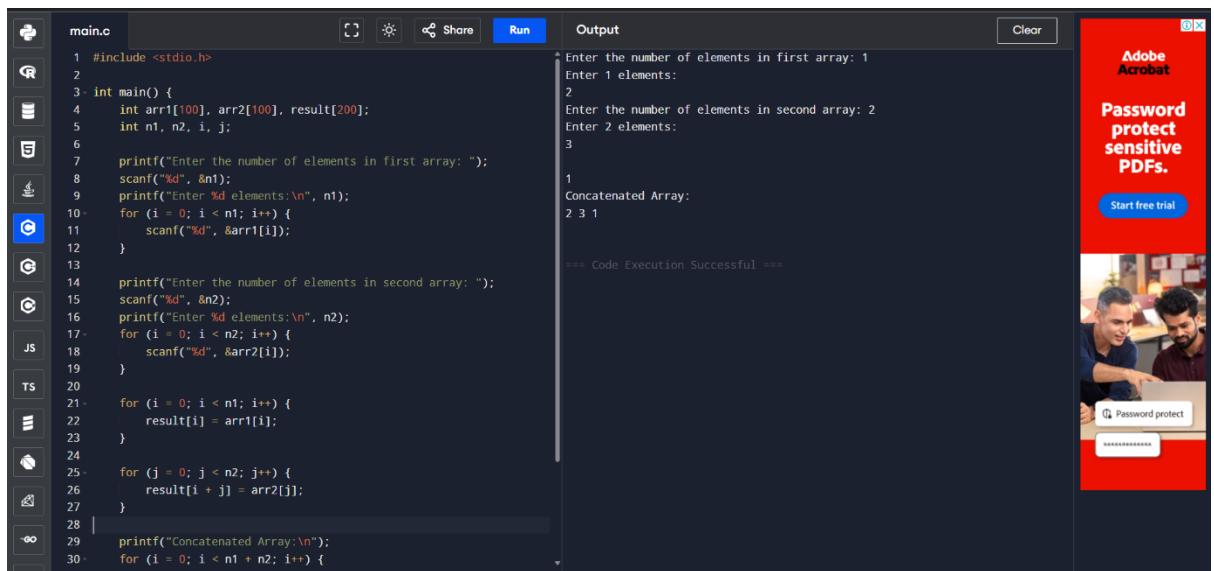


```
main.c
8         arr[j] = arr[j + 1];
9         arr[j + 1] = temp;
10    }
11 }
12 }
13 }
14
15 int main() {
16     int arr[100], n, i;
17
18     printf("Enter the number of elements (max 100): ");
19     scanf("%d", &n);
20
21     printf("Enter %d elements:\n", n);
22     for (i = 0; i < n; i++) {
23         scanf("%d", &arr[i]);
24     }
25
26     bubbleSort(arr, n);
27
28     printf("Sorted array:\n");
29     for (i = 0; i < n; i++) {
30
31
32
33
34
35
36 }
```

Output

```
Enter the number of elements (max 100): 3
Enter 3 elements:
1
2
4
Sorted array:
1 2 4
== Code Execution Successful ==
```

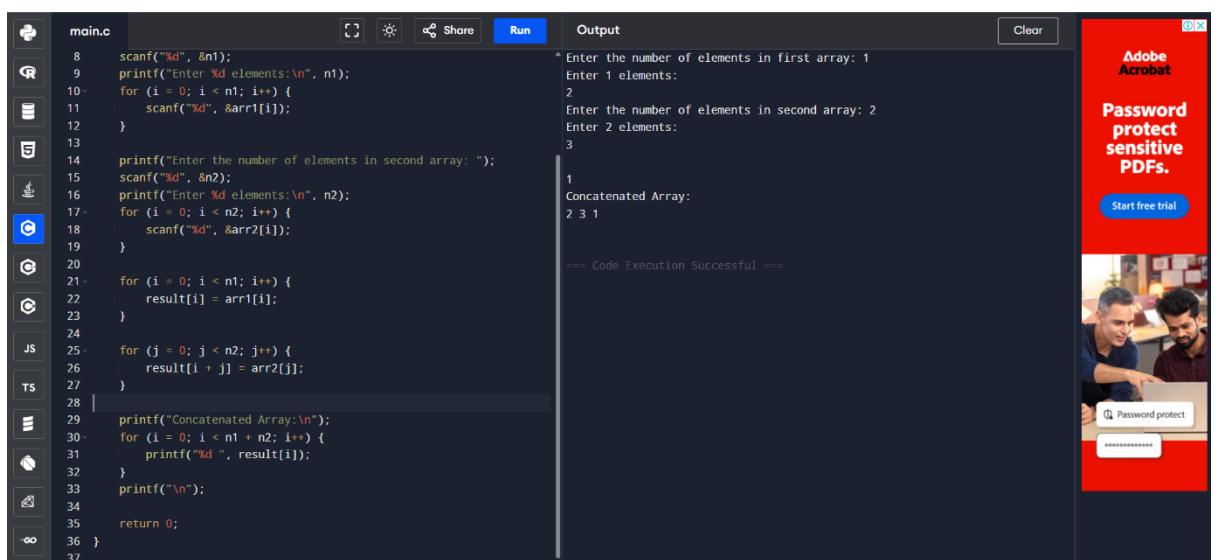
5. Write a program in C to concatenate two arrays.



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

Output

```
Enter the number of elements in first array: 1
Enter 1 elements:
2
Enter the number of elements in second array: 2
Enter 2 elements:
3
1
Concatenated Array:
2 3 1
*** Code Execution Successful ***
```

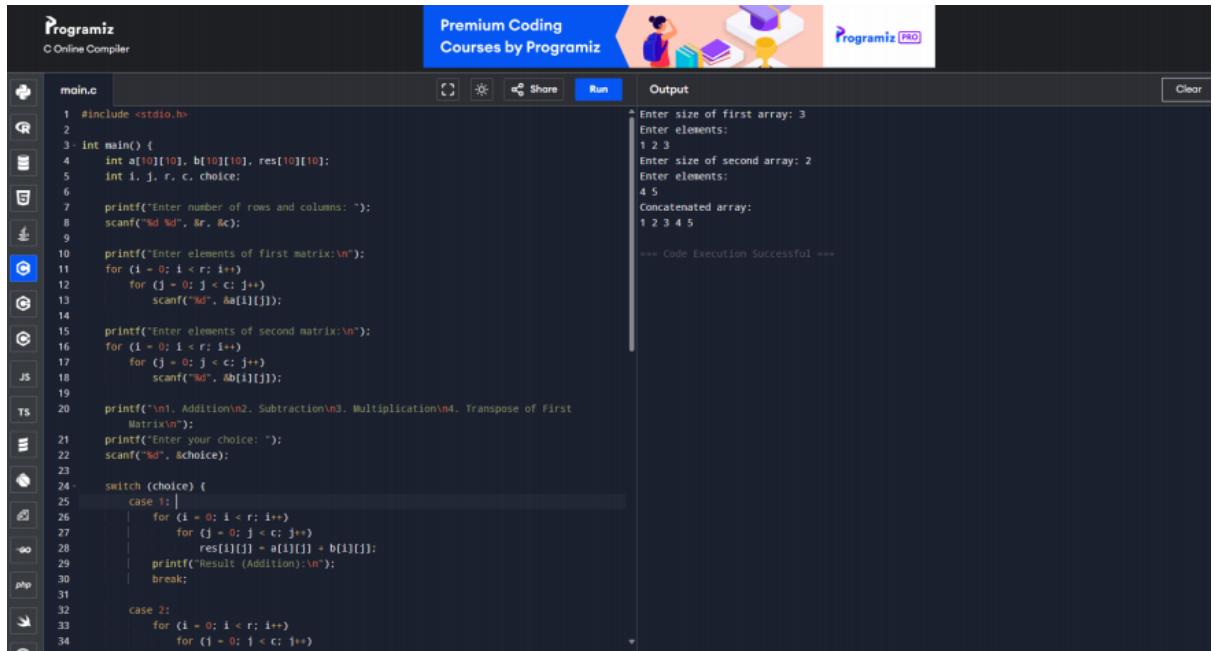


```
main.c
8     scanf("%d", &n1);
9     printf("Enter %d elements:\n", n1);
10    for (i = 0; i < n1; i++) {
11        scanf("%d", &arr1[i]);
12    }
13
14    printf("Enter the number of elements in second array: ");
15    scanf("%d", &n2);
16    printf("Enter %d elements:\n", n2);
17    for (i = 0; i < n2; i++) {
18        scanf("%d", &arr2[i]);
19    }
20
21    for (i = 0; i < n1; i++) {
22        result[i] = arr1[i];
23    }
24
25    for (j = 0; j < n2; j++) {
26        result[i + j] = arr2[j];
27    }
28
29    printf("Concatenated Array:\n");
30    for (i = 0; i < n1 + n2; i++) {
31        printf("%d ", result[i]);
32    }
33    printf("\n");
34
35    return 0;
36 }
```

Output

```
Enter the number of elements in first array: 1
Enter 1 elements:
2
Enter the number of elements in second array: 2
Enter 2 elements:
3
1
Concatenated Array:
2 3 1
*** Code Execution Successful ***
```

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

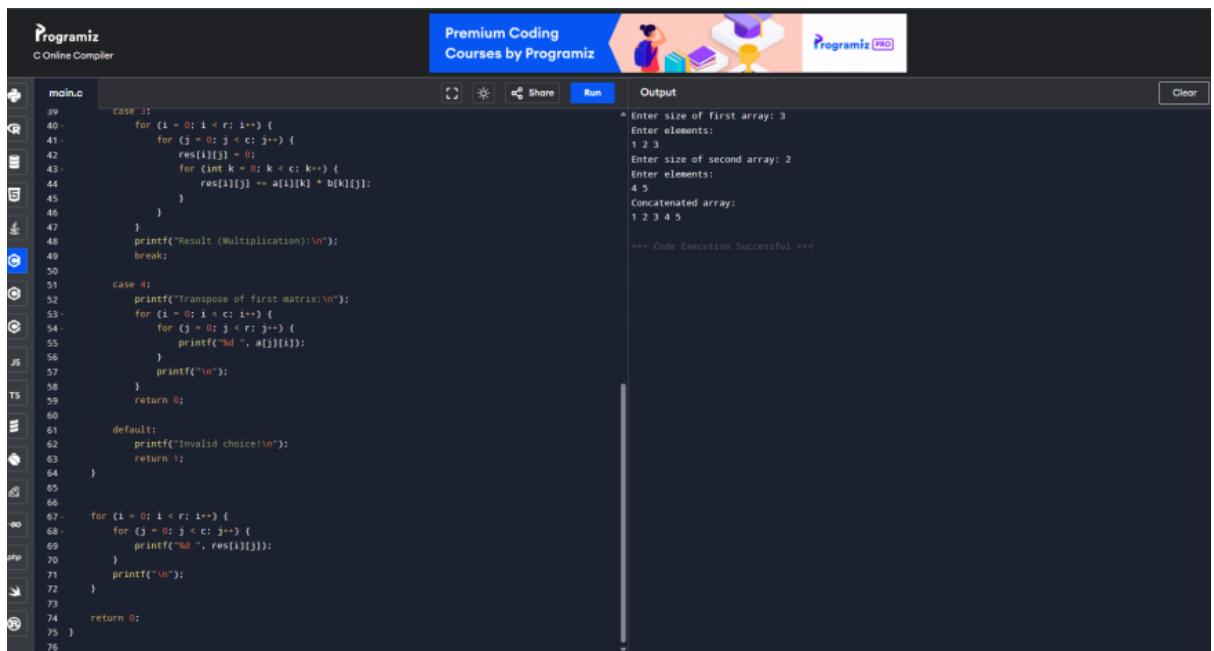


The screenshot shows the Programiz C Online Compiler interface. The code in main.c implements various operations on 2D arrays. The output window shows the execution of the program, prompting for matrix sizes and elements, and displaying the concatenated array and execution status.

```
main.c
1 #include <stdio.h>
2
3 int main() {
4     int a[10][10], b[10][10], res[10][10];
5     int i, j, r, c, choice;
6
7     printf("Enter number of rows and columns: ");
8     scanf("%d %d", &r, &c);
9
10    printf("Enter elements of first matrix:\n");
11    for (i = 0; i < r; i++) {
12        for (j = 0; j < c; j++)
13            scanf("%d", &a[i][j]);
14
15    printf("Enter elements of second matrix:\n");
16    for (i = 0; i < r; i++) {
17        for (j = 0; j < c; j++)
18            scanf("%d", &b[i][j]);
19
20    printf("\n1. Addition\n2. Subtraction\n3. Multiplication\n4. Transpose of First
Matrix\n");
21    printf("Enter your choice: ");
22    scanf("%d", &choice);
23
24    switch (choice) {
25        case 1:
26            for (i = 0; i < r; i++) {
27                for (j = 0; j < c; j++) {
28                    res[i][j] = a[i][j] + b[i][j];
29                }
30            }
31            break;
32
33        case 2:
34            for (i = 0; i < r; i++) {
35                for (j = 0; j < c; j++)
```

Output

```
Enter size of first array: 3
Enter elements:
1 2 3
Enter size of second array: 2
Enter elements:
4 5
Concatenated array:
1 2 3 4 5
*** Code Execution Successful ***
```



The screenshot shows the Programiz C Online Compiler interface. The code in main.c handles matrix operations based on user choice. The output window shows the execution of the program, prompting for matrix sizes and elements, and displaying the concatenated array and execution status.

```
main.c
39
40    case 3:
41        for (i = 0; i < r; i++) {
42            for (j = 0; j < c; j++) {
43                res[i][j] = 0;
44                for (int k = 0; k < c; k++) {
45                    res[i][j] += a[i][k] * b[k][j];
46                }
47            }
48        printf("Result (Multiplication):\n");
49        break;
50
51    case 4:
52        printf("Transpose of first matrix:\n");
53        for (i = 0; i < c; i++) {
54            for (j = 0; j < r; j++) {
55                printf("%d ", a[j][i]);
56            }
57        printf("\n");
58    }
59    return 0;
60
61    default:
62        printf("Invalid choice!\n");
63        return 1;
64    }
65
66
67    for (i = 0; i < r; i++) {
68        for (j = 0; j < c; j++) {
69            printf("%d ", res[i][j]);
70        }
71    printf("\n");
72}
73
74    return 0;
75 }
```

Output

```
Enter size of first array: 3
Enter elements:
1 2 3
Enter size of second array: 2
Enter elements:
4 5
Concatenated array:
1 2 3 4 5
*** Code Execution Successful ***
```

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

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is as follows:

```
#include <stdio.h>
#define SIZE 100
int stack[SIZE];
int top = -1;

void push(int val) {
    if (top == SIZE - 1) {
        printf("Stack overflow\n");
    } else {
        top++;
        stack[top] = val;
        printf("Pushed %d\n", val);
    }
}

void pop() {
    if (top == -1) {
        printf("Stack underflow\n");
    } else {
        printf("Popped %d\n", stack[top]);
        top--;
    }
}

void display() {
    if (top == -1) {
        printf("Stack is empty\n");
    } else {
        printf("Stack elements:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d\n", stack[i]);
        }
    }
}

int main() {
    int choice, value;
    while (1) {
        printf("1. Push\n2. Pop\n3. Display\n4. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice\n");
        }
    }
}
```

The output window shows the following interaction:

```
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 3
Stack is empty
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 2
Stack Underflow
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 4
--- Code Execution Successful ---
```

The screenshot shows the Programiz C Online Compiler interface with a modified version of the stack program. The code is as follows:

```
#include <stdio.h>
int stack[SIZE];
int top = -1;

void display() {
    if (stack == -1) {
        printf("Stack is empty\n");
    } else {
        printf("Stack elements:\n");
        for (int i = top; i >= 0; i--) {
            printf("%d\n", stack[i]);
        }
    }
}

int main() {
    int choice, value;
    while (1) {
        printf("1. Push\n2. Pop\n3. Display\n4. Exit\nEnter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice\n");
        }
    }
}
```

The output window shows the following interaction:

```
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 3
Stack is empty
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 2
Stack Underflow
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 4
--- Code Execution Successful ---
```

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

The screenshot shows a C online compiler interface on Programiz. The code in the editor is as follows:

```
main.c
1 #include <stdio.h>
2 #define SIZE 100
3
4 int stack[SIZE];
5 int top = -1;
6
7 // Push operation
8 void push(int val) {
9     if (top == SIZE - 1) {
10         printf("Stack Overflow\n");
11     } else {
12         top++;
13         stack[top] = val;
14         printf("Pushed %d\n", val);
15     }
16 }
17
18 // Pop operation
19 void pop() {
20     if (top == -1) {
21         printf("Stack Underflow\n");
22     } else {
23         printf("Popped %d\n", stack[top]);
24         top--;
25     }
26 }
27
28 // Display operation
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         }
37     }
38 }
```

The output window shows the execution of the program:

```
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 3
Stack is empty

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 2
Stack Underflow

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 4
--- Code Execution Successful ---
```

The screenshot shows a C online compiler interface on Programiz. The code in the editor is as follows:

```
main.c
21     printf("Stack is empty\n");
22 } else {
23     printf("Stack elements:\n");
24     for (int i = top; i >= 0; i--) {
25         printf("%d\n", stack[i]);
26     }
27 }
28
29 int main() {
30     int choice, value;
31
32     while (1) {
33         printf("1. Push\n2. Pop\n3. Display\n4. Exit\nEnter choice: ");
34         scanf("%d", &choice);
35         if (choice == 1) {
36             printf("Enter value to push: ");
37             scanf("%d", &value);
38             push(value);
39         } else if (choice == 2) {
40             pop();
41         } else if (choice == 3) {
42             display();
43         } else if (choice == 4) {
44             return 0;
45         } else {
46             printf("Invalid choice\n");
47         }
48     }
49 }
```

The output window shows the execution of the program:

```
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 3
Stack is empty

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 2
Stack underflow

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 4
--- Code Execution Successful ---
```

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

Programiz
C Online Compiler

Premium Coding Courses by Programiz

Output

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 struct Node {
6     char data;
7     struct Node* next;
8 }
9
10 struct Node* top = NULL;
11
12
13 void push(char value) {
14     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
15     if (newNode == NULL) {
16         printf("Stack Overflow\n");
17         return;
18     }
19     newNode->data = value;
20     newNode->next = top;
21     top = newNode;
22 }
23
24
25 char pop() {
26     if (top == NULL) {
27         return -1;
28     }
29     struct Node* temp = top;
30     top = top->next;
31     char popped = temp->data;
32     free(temp);
33     return popped;
34 }
35
36
37 int isBalanced(char* expr) {
38     for (int i = 0; expr[i] != '\0'; i++) {
39         if (expr[i] == '(' || expr[i] == '[' || expr[i] == '{') {
40             push(expr[i]);
41         } else if (expr[i] == ')' || expr[i] == ']' || expr[i] == '}') {
42             char topChar = pop();
43             if ((expr[i] == ')' && topChar != '(') ||
44                 (expr[i] == ']' && topChar != '[') ||
45                 (expr[i] == '}' && topChar != '{')) {
46                 return 0; // Not balanced
47             }
48         }
49     }
50     return top == NULL;
51 }
52
53
54 int main() {
55     char expr[100];
56
57     printf("Enter an expression with parentheses: ");
58     scanf("%s", expr);
59
60     if (isBalanced(expr)) {
61         printf("The parentheses are balanced.\n");
62     } else {
63         printf("The parentheses are not balanced.\n");
64     }
65
66     return 0;
67 }
```

Enter an expression with parentheses: ((a+b)*(c/d))
The parentheses are balanced.
*** Code Execution Successful ***

Programiz
C Online Compiler

Premium Coding Courses by Programiz

Output

```
51     char popped = temp->data;
52     free(temp);
53     return popped;
54 }
55
56
57 int isBalanced(char* expr) {
58     for (int i = 0; expr[i] != '\0'; i++) {
59         if (expr[i] == '(' || expr[i] == '[' || expr[i] == '{') {
60             push(expr[i]);
61         } else if (expr[i] == ')' || expr[i] == ']' || expr[i] == '}') {
62             char topChar = pop();
63             if ((expr[i] == ')' && topChar != '(') ||
64                 (expr[i] == ']' && topChar != '[') ||
65                 (expr[i] == '}' && topChar != '{')) {
66                 return 0; // Not balanced
67             }
68         }
69     }
70     return top == NULL;
71 }
72
73
74 int main() {
75     char expr[100];
76
77     printf("Enter an expression with parentheses: ");
78     scanf("%s", expr);
79
80     if (isBalanced(expr)) {
81         printf("The parentheses are balanced.\n");
82     } else {
83         printf("The parentheses are not balanced.\n");
84     }
85
86     return 0;
87 }
```

Enter an expression with parentheses: ((a+b)*(c/d))
The parentheses are balanced.
*** Code Execution Successful ***

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

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is as follows:

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 struct Node {
6     int data;
7     struct Node* next;
8 };
9
10 struct Node* stack1 = NULL;
11 struct Node* stack2 = NULL;
12
13
14 void push(struct Node** top, int value) {
15     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
16     if (newNode == NULL) {
17         printf("Memory error\n");
18         return;
19     }
20     newNode->data = value;
21     newNode->next = *top;
22     *top = newNode;
23 }
24
25
26 int pop(struct Node** top) {
27     if (*top == NULL) {
28         return -1;
29     }
30     struct Node* temp = *top;
31     *top = (*top)->next;
32     int popped = temp->data;
33     free(temp);
34     return popped;
35 }
36
37 // Enqueue operation (push to stack1)
38 void enqueue(int value) {
39     push(&stack1, value);
40     printf("Enqueued %d", value);
41 }
42
43 // Dequeue operation (pop from stack1)
44 int dequeue() {
45     if (stack1 == NULL && stack2 == NULL) {
46         printf("Queue is empty\n");
47         return -1;
48     }
49
50     // If stack2 is empty, move elements from stack1 to stack2
51     if (stack2 == NULL) {
52         while (stack1 != NULL) {
53             int value = pop(&stack1);
54             push(&stack2, value);
55         }
56     }
57
58     return pop(&stack2); // Pop from stack2 (FIFO order)
59 }
60
61 // Display operation (for debugging)
62 void displayQueue() {
63     if (stack1 == NULL && stack2 == NULL) {
64         printf("Queue is empty\n");
65         return;
66     }
67
68     struct Node* temp = stack2;
69     printf("Queue elements: ");
70     while (temp != NULL) {
71         printf("%d ", temp->data);
72         temp = temp->next;
73     }
74 }
```

The output window shows the following interaction:

```
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter choice: 1
Enter value to enqueue: 5
Enqueued 5

1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter choice: 4
*** Code Execution Successful ***
```

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is as follows:

```
main.c
1 // Enqueue operation (push to stack1)
2 void enqueue(int value) {
3     push(&stack1, value);
4     printf("Enqueued %d", value);
5 }
6
7 // Dequeue operation (pop from stack1)
8 int dequeue() {
9     if (stack1 == NULL && stack2 == NULL) {
10         printf("Queue is empty\n");
11         return -1;
12     }
13
14     // If stack2 is empty, move elements from stack1 to stack2
15     if (stack2 == NULL) {
16         while (stack1 != NULL) {
17             int value = pop(&stack1);
18             push(&stack2, value);
19         }
20     }
21
22     return pop(&stack2); // Pop from stack2 (FIFO order)
23 }
24
25 // Display operation (for debugging)
26 void displayQueue() {
27     if (stack1 == NULL && stack2 == NULL) {
28         printf("Queue is empty\n");
29         return;
30     }
31
32     struct Node* temp = stack2;
33     printf("Queue elements: ");
34     while (temp != NULL) {
35         printf("%d ", temp->data);
36         temp = temp->next;
37     }
38 }
```

The output window shows the following interaction:

```
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter choice: 1
Enter value to enqueue: 5
Enqueued 5

1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter choice: 4
*** Code Execution Successful ***
```

Programiz
C Online Compiler

Premium Coding
Courses by Programiz

Programiz PRO

Run Clear

```
main.c
66     temp = temp / 1000;
67 }
68
69     printf("\n");
70 }
71
72 int main() {
73     int choice, value;
74
75     while (1) {
76         printf("1. Enqueue\n2. Dequeue\n3. Display Queue\n4. Exit\nEnter choice: ");
77         scanf("%d", &choice);
78
79         switch (choice) {
80             case 1:
81                 printf("Enter value to enqueue: ");
82                 scanf("%d", &value);
83                 enqueue(value);
84                 break;
85             case 2:
86                 value = dequeue();
87                 if (value != -1) {
88                     printf("Dequeued %d\n", value);
89                 }
90                 break;
91             case 3:
92                 displayQueue();
93                 break;
94             case 4:
95                 return 0;
96             default:
97                 printf("Invalid choice\n");
98         }
99     }
100
101     return 0;
102 }
```

1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter choice: 1
Enter value to enqueue: 5
Enqueued 5
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter choice: 4
==== Code Execution Successful ===

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

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is as follows:

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 struct Node {
6     int data;
7     struct Node* next;
8 };
9
10 struct Node* front = NULL;
11 struct Node* rear = NULL;
12
13
14 void enqueue(int value) {
15     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
16     if (newNode == NULL) {
17         printf("Memory error\n");
18         return;
19     }
20     newNode->data = value;
21     newNode->next = NULL;
22
23     if (rear == NULL) {
24         front = rear = newNode;
25     } else {
26         rear->next = newNode;
27         rear = newNode;
28     }
29     printf("Enqueued %d\n", value);
30 }
31
32
33 int dequeue() {
34     if (front == NULL) {
35         printf("Queue is empty\n");
36         return -1;
37     }
38     struct Node* temp = front;
39     int value = temp->data;
40     front = front->next;
41     if (front == NULL) {
42         rear = NULL;
43     }
44     free(temp);
45     return value;
46 }
47
48
49 void display() {
50     if (front == NULL) {
51         printf("Queue is empty\n");
52         return;
53     }
54     struct Node* temp = front;
55     printf("%d elements:\n");
56     while (temp != NULL) {
57         printf("%d ", temp->data);
58         temp = temp->next;
59     }
60     printf("\n");
61 }
62
63 int main() {
64     int choice, value;
65
66     while (1) {
67         printf("1. Enqueue\n2. Dequeue\n3. Display Queue\n4. Exit\nEnter choice: ");
68         scanf("%d", &choice);
69
70         switch (choice) {
71             case 1:
72                 printf("Enter value to enqueue: ");
73                 scanf("%d", &value);
74                 enqueue(value);
```

The output window shows the following interaction:

- 1. Enqueue
- 2. Dequeue
- 3. Display Queue
- 4. Exit

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

Enter choice: 1
Enter value to enqueue: 20
Enqueued 20

Enter choice: 3
Queue elements: 10 20

Enter choice: 2
Dequeued 10

Enter choice: 3
Queue is empty

Enter choice: 4

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is as follows:

```
main.c
38     struct Node* temp = front;
39     int value = temp->data;
40     front = front->next;
41     if (front == NULL) {
42         rear = NULL;
43     }
44     free(temp);
45     return value;
46 }
47
48
49 void display() {
50     if (front == NULL) {
51         printf("Queue is empty\n");
52         return;
53     }
54     struct Node* temp = front;
55     printf("%d elements:\n");
56     while (temp != NULL) {
57         printf("%d ", temp->data);
58         temp = temp->next;
59     }
60     printf("\n");
61 }
62
63 int main() {
64     int choice, value;
65
66     while (1) {
67         printf("1. Enqueue\n2. Dequeue\n3. Display Queue\n4. Exit\nEnter choice: ");
68         scanf("%d", &choice);
69
70         switch (choice) {
71             case 1:
72                 printf("Enter value to enqueue: ");
73                 scanf("%d", &value);
74                 enqueue(value);
```

The output window shows the following interaction:

- 1. Enqueue
- 2. Dequeue
- 3. Display Queue
- 4. Exit

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

Enter choice: 1
Enter value to enqueue: 20
Enqueued 20

Enter choice: 3
Queue elements: 10 20

Enter choice: 2
Dequeued 10

Enter choice: 3
Queue elements: 20

Enter choice: 2
Dequeued 20

Enter choice: 3
Queue is empty

Enter choice: 4

Programiz
C Online Compiler

Premium Coding Courses by Programiz

Run | Clear

```
main.c
57     printf("%d\n", temp->data);
58     temp = temp->next;
59 }
60 printf("\n");
61 }
62
63 int main() {
64     int choice, value;
65
66     while (1) {
67         printf("1. Enqueue\n2. Dequeue\n3. Display Queue\n4. Exit\nEnter choice: ");
68         scanf("%d", &choice);
69
70         switch (choice) {
71             case 1:
72                 printf("Enter value to enqueue: ");
73                 scanf("%d", &value);
74                 enqueue(value);
75                 break;
76             case 2:
77                 value = dequeue();
78                 if (value != -1) {
79                     printf("Dequeued %d\n", value);
80                 }
81                 break;
82             case 3:
83                 display();
84                 break;
85             case 4:
86                 return 0;
87             default:
88                 printf("Invalid choice\n");
89         }
90     }
91
92 }
93 }
```

Output

- 1. Enqueue
- 2. Dequeue
- 3. Display Queue
- 4. exit

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

Enter choice: 1
Enter value to enqueue: 20
Enqueued 20

Enter choice: 3
Queue elements: 10 20

Enter choice: 2
Dequeued 10

Enter choice: 3
Queue elements: 20

Enter choice: 2
Dequeued 20

Enter choice: 3
Queue is empty

Enter choice: 4

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

The screenshot shows the Programiz C Online Compiler interface. The code editor contains a C program for a queue implementation using arrays. The output window shows the execution of the program, demonstrating enqueue and dequeue operations.

```
#include <stdio.h>
#define SIZE 5
int queue[SIZE];
int front = -1, rear = -1;
void enqueue(int value) {
    if ((front == -1) && (rear == SIZE - 1)) || (rear + 1 == front)) {
        printf("Queue is full\n");
        return;
    }
    if (front == -1) {
        front = rear = 0;
    } else if (rear == SIZE - 1 && front != 0) {
        rear = 0;
    } else {
        rear++;
    }
    queue[rear] = value;
    printf("Enqueued %d\n", value);
}
void dequeue() {
    if (front == -1) {
        printf("Queue is empty\n");
        return;
    }
    printf("Dequeued %d\n", queue[front]);
    if (front == rear) {
        front = rear = -1;
    } else if (front == SIZE - 1) {
        front = 0;
    }
}
```

Output:

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 1
Enter value to enqueue: 10
Enqueued 10

Enter choice: 1
Enter value to enqueue: 20
Enqueued 20

Enter choice: 3
Queue elements: 10 20

Enter choice: 2
Dequeued 10

Enter choice: 3
Queue Elements: 20

Enter choice: 4
```

Programiz

C Online Compiler

Premium Coding Courses by Programiz



Programiz PRO

main.c

```
39 } else {
40     front++;
41 }
42 }
43
44 void display() {
45     if (front == -1) {
46         printf("Queue is empty\n");
47         return;
48     }
49 }
50
51 printf("Queue elements: ");
52 int i = front;
53 while (i) {
54     printf("%d ", queue[i]);
55     if (i == rear)
56         break;
57     i = (i + 1) % SIZE;
58 }
59 printf("\n");
60 }
61
62 int main() {
63     int choice, value;
64
65     while (1) {
66         printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\nEnter choice: ");
67         scanf("%d", &choice);
68
69         switch (choice) {
70             case 1:
71                 printf("Enter value to enqueue: ");
72                 scanf("%d", &value);
73                 enqueue(value);
74                 break;
75             case 2:
76                 dequeue();
77                 break;
78             case 3:
79                 display();
80                 break;
81             case 4:
82                 exit(0);
83         }
84     }
85 }
```

Output

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 1
Enter value to enqueue: 10
Enqueued 10

Enter choice: 1
Enter value to enqueue: 20
Enqueued 20

Enter choice: 3
Queue elements: 10 20

Enter choice: 2
Dequeued 10

Enter choice: 3
Queue elements: 20

Enter choice: 4
```

Programiz
C Online Compiler

Premium Coding
Courses by Programiz

Programiz PRO

```
main.c
53     printf("%d ", queue[i]);
54     if (i == rear)
55         break;
56     i = (i + 1) % SIZE;
57 }
58 printf("\n");
59
60 }
61
62 int main() {
63     int choice, value;
64
65     while (1) {
66         printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\nEnter choice: ");
67         scanf("%d", &choice);
68
69         switch (choice) {
70             case 1:
71                 printf("Enter value to enqueue: ");
72                 scanf("%d", &value);
73                 enqueue(value);
74                 break;
75             case 2:
76                 dequeue();
77                 break;
78             case 3:
79                 display();
80                 break;
81             case 4:
82                 return 0;
83             default:
84                 printf("Invalid choice!\n");
85         }
86     }
87
88     return 0;
89 }
```

Run

Output:

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

Enter choice: 1
Enter value to enqueue: 20
Enqueued 20

Enter choice: 3
Queue elements: 10 20

Enter choice: 2
Dequeued 10

Enter choice: 3
Queue elements: 20

Enter choice: 4

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

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is for inserting at the beginning of a linked list. The output window shows the program's flow: it asks for a choice (1 for insert at beginning), enters value 10, and inserts it at the beginning, resulting in the linked list 10 -> NULL. It then asks for another choice, enters value 30, and inserts it at the beginning, resulting in the linked list 30 -> NULL.

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

Output:

- 1. Insert at Beginning
- 2. Insert at End
- 3. Insert at Position
- 4. Display
- 5. Exit

Enter your choice: 1
Enter value: 10
Inserted 10 at beginning

Enter your choice: 2
Enter value: 30
Inserted 30 at end

Enter your choice: 3
Enter value and position: 20 1
Inserted 20 after position 1

Enter your choice: 4
Linked List: 10 -> 20 -> 30 -> NULL

The screenshot shows the Programiz C Online Compiler interface. The code has been modified to allow insertion at a specific position. The output window shows the program's flow: it asks for a choice (3 for insert at position), enters value 10 and position 1, and inserts it at position 1, resulting in the linked list 10 -> NULL. It then asks for another choice, enters value 30 and position 1, and inserts it at position 1, resulting in the linked list 30 -> NULL.

```
main.c
39 void insertAtPosition(int value, int position) {
40     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
41     newNode->data = value;
42
43     struct Node* temp = head;
44     for (int i = 1; i < position && temp != NULL; i++) {
45         temp = temp->next;
46     }
47
48     if (temp == NULL) {
49         printf("Position not found.\n");
50         return;
51     }
52
53     newNode->next = temp->next;
54     temp->next = newNode;
55     printf("Inserted %d after position %d\n", value, position);
56 }
57
58 void display() {
59     struct Node* temp = head;
60     printf("Linked List: ");
61     while (temp != NULL) {
62         printf("%d -> ", temp->data);
63         temp = temp->next;
64     }
65     printf("NULL\n");
66 }
67
68 int main() {
69     int choice, value, position;
70
71     while (1) {
72         printf("\n1. Insert at Beginning\n2. Insert at End\n3. Insert at Position\n4. Display\n5. Exit\nEnter your choice: ");
73         scanf("%d", &choice);
74     }
75 }
```

Output:

- 1. Insert at Beginning
- 2. Insert at End
- 3. Insert at Position
- 4. Display
- 5. Exit

Enter your choice: 1
Enter value: 10
Inserted 10 at beginning

Enter your choice: 2
Enter value: 30
Inserted 30 at end

Enter your choice: 3
Enter value and position: 20 1
Inserted 20 after position 1

Enter your choice: 4
Linked List: 10 -> 20 -> 30 -> NULL

Programiz
C Online Compiler

Premium Coding
Courses by Programiz

Programiz PRO

The screenshot shows a C programming environment on the Programiz website. The code in the editor is for a linked list insertion program. The output window shows the execution of the program, starting with a menu of options (1. Insert at Beginning, 2. Insert at End, etc.) and then demonstrating the insertion of nodes with values 10, 20, and 30 at various positions.

```
main.c
66
67- int main() {
68    int choice, value, position;
69
70    while (1) {
71        printf("1. Insert at Beginning\n2. Insert at End\n3. Insert at Position\n4. Display\n5. Exit\nEnter your 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 value and position: ");
87                scanf("%d %d", &value, &position);
88                insertAtPosition(value, position);
89                break;
90            case 4:
91                display();
92                break;
93            case 5:
94                return 0;
95            default:
96                printf("Invalid choice\n");
97        }
98    }
99}
100}
101
102 return 0;
103}
104}
```

Output

- 1. Insert at Beginning
- 2. Insert at End
- 3. Insert at Position
- 4. Display
- 5. Exit

Enter your choice: 1
Enter value: 10
Inserted 10 at beginning

Enter your choice: 2
Enter value: 30
Inserted 30 at end

Enter your choice: 3
Enter value and position: 20 1
Inserted 20 after position 1

Enter your choice: 4
Linked List: 10 -> 20 -> 30 -> NULL

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

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is for a linked list implementation and includes functions for insertion at the beginning, deletion from the beginning, and printing the list. The output window shows the execution of the program, starting with a menu of options (Delete from Beginning, Delete from End, etc.) and then performing specific deletions (Delete 10 from beginning, Delete 40 from end, Delete 30 from position 2), which results in the linked list being modified.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 struct Node {
6     int data;
7     struct Node* next;
8 };
9
10 struct Node* head = NULL;
11
12
13 void insertAtEnd(int value) {
14     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
15     newNode->data = value;
16     newNode->next = NULL;
17
18     if (head == NULL)
19         head = newNode;
20     else {
21         struct Node* temp = head;
22         while (temp->next != NULL)
23             temp = temp->next;
24         temp->next = newNode;
25     }
26 }
27
28
29 void deleteFromBeginning() {
30     if (head == NULL) {
31         printf("List is empty\n");
32         return;
33     }
34     struct Node* temp = head;
35     head = head->next;
36     printf("Deleted %d from beginning\n", temp->data);
37     free(temp);
38 }
39
40
41 void deleteFromEnd() {
42     if (head == NULL) {
43         printf("List is empty\n");
44         return;
45     }
46     struct Node* temp = head;
47     struct Node* prev = NULL;
48
49     while (temp->next != NULL) {
50         prev = temp;
51         temp = temp->next;
52     }
53
54     if (prev == NULL)
55         head = NULL;
56     else
57         prev->next = NULL;
58
59     printf("Deleted %d from end\n", temp->data);
60     free(temp);
61 }
62
63
64 void deleteFromPosition(int position) {
65     if (head == NULL) {
66         printf("List is empty\n");
67         return;
68     }
69
70     if (position == 1) {
71         deleteFromBeginning();
72         return;
73     }
74 }
```

Output:

- 1. Delete from Beginning
- 2. Delete from End
- 3. Delete from Position
- 4. Display
- 5. Exit

Enter your choice: 4
Linked List: 10 -> 20 -> 30 -> 40 -> NULL

Enter your choice: 1
Deleted 10 from beginning

Enter your choice: 2
Deleted 40 from end

Enter your choice: 3
Enter position to delete: 2
Deleted 30 from position 2

Enter your choice: 4
Linked List: 20 -> NULL

The screenshot shows the Programiz C Online Compiler interface. The code is identical to the one in the first screenshot, but the output window shows a different sequence of operations: first, it prints the initial linked list (10 -> 20 -> 30 -> 40 -> NULL). Then, it performs a deletion from the beginning (Delete 10 from beginning), resulting in the list 20 -> 30 -> 40 -> NULL. Next, it performs a deletion from the end (Delete 40 from end), resulting in the list 20 -> 30 -> NULL. Finally, it performs a deletion from position 2 (Delete 30 from position 2), resulting in an empty list (NULL).

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 struct Node {
6     int data;
7     struct Node* next;
8 };
9
10 struct Node* head = NULL;
11
12
13 void insertAtEnd(int value) {
14     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
15     newNode->data = value;
16     newNode->next = NULL;
17
18     if (head == NULL)
19         head = newNode;
20     else {
21         struct Node* temp = head;
22         while (temp->next != NULL)
23             temp = temp->next;
24         temp->next = newNode;
25     }
26 }
27
28
29 void deleteFromBeginning() {
30     if (head == NULL) {
31         printf("List is empty\n");
32         return;
33     }
34     struct Node* temp = head;
35     head = head->next;
36     printf("Deleted %d from beginning\n", temp->data);
37     free(temp);
38 }
39
40
41 void deleteFromEnd() {
42     if (head == NULL) {
43         printf("List is empty\n");
44         return;
45     }
46     struct Node* temp = head;
47     struct Node* prev = NULL;
48
49     while (temp->next != NULL) {
50         prev = temp;
51         temp = temp->next;
52     }
53
54     if (prev == NULL)
55         head = NULL;
56     else
57         prev->next = NULL;
58
59     printf("Deleted %d from end\n", temp->data);
60     free(temp);
61 }
62
63
64 void deleteFromPosition(int position) {
65     if (head == NULL) {
66         printf("List is empty\n");
67         return;
68     }
69
70     if (position == 1) {
71         deleteFromBeginning();
72         return;
73     }
74 }
```

Output:

Linked List: 10 -> 20 -> 30 -> 40 -> NULL

Deleted 10 from beginning

Deleted 40 from end

Deleted 30 from position 2

Linked List: 20 -> NULL

Programiz
C Online Compiler

Premium Coding Courses by Programiz

Share Run Output Clear

```
main.c
1. Delete from Beginning
2. Delete from End
3. Delete from Position
4. Display
5. Exit
Enter your choice: 4
Linked List: 10 -> 20 -> 30 -> 40 -> NULL

Enter your choice: 1
Deleted 10 from beginning

Enter your choice: 2
Deleted 40 from end

Enter your choice: 3
Enter position to delete: 2
Deleted 30 from position 2

Enter your choice: 4
Linked List: 20 -> NULL
```

File Editor Language: C Compiler: g++ Version: 11.2.0 Build: 2023-09-14 10:00:00 UTC

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 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 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141

Programiz
C Online Compiler

SHARPEN YOUR SKILLS
Adelaide University
Access Now

Share Run Output Clear

```
main.c
1. Delete From Beginning
2. Delete From End
3. Delete From Position
4. Display
5. Exit
Enter your choice: 4
Linked List: 10 -> 20 -> 30 -> 40 -> NULL

Enter your choice: 1
Deleted 10 from beginning

Enter your choice: 2
Deleted 40 from end

Enter your choice: 3
Enter position to delete: 2
Deleted 30 from position 2

Enter your choice: 4
Linked List: 20 -> NULL
```

File Editor Language: C Compiler: g++ Version: 11.2.0 Build: 2023-09-14 10:00:00 UTC

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 38 39 39 40 41 42 43 44 45 46 47 48 49 49 50 51 52 53 54 55 56 57 58 59 59 60 61 62 63 64 65 66 67 68 69 69 70 71 72 73 74 75 76 77 78 79 79 80 81 82 83 84 85 86 87 88 89 89 90 91 92 93 94 95 96 97 98 99 99 100 101 102 103 104 105 106 107 108 109 109 110 111 112 113 114 115 116 117 118 119 119 120 121 122 123 124 125 126 127 128 129 129 130 131 132 133 134 135 136 137 138 139 139 140 141

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

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is for a circular linked list implementation. It includes functions for displaying the list, inserting at the beginning, and inserting at the end. The output window shows the execution of the program, starting with a menu of options (1. Insert at Beginning, 2. Insert at End, etc.). It then inserts the value 10 at the beginning, followed by 30 at the end, and 20 at position 1. Finally, it displays the circular linked list structure.

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

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

struct Node* last = NULL;

void display() {
    if (last == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* temp = last->next;
    do {
        printf("%d -> ", temp->data);
        temp = temp->next;
    } while (temp != last->next);
    printf("(head)\n");
}

void insertAtBeginning(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;

    if (last == NULL) {
        newNode->next = newNode;
        last = newNode;
    } else {
        newNode->next = last->next;
        last->next = newNode;
    }
}

int main() {
    // Your main code here
}
```

Output:

- 1. Insert at Beginning
- 2. Insert at End
- 3. Insert at Position
- 4. Display
- 5. Exit

Enter your choice: 1
Enter value: 10
Inserted 10 at beginning

Enter your choice: 2
Enter value: 30
Inserted 30 at end

Enter your choice: 3
Enter value and position: 20 1
Inserted 20 after position 1

Enter your choice: 4
Circular Linked List: 10 -> 20 -> 30 -> (head)

Enter your choice: 5

This screenshot shows the same C program running on the Programiz compiler, but with a different sequence of user inputs. The user first inserts 10 at the beginning, then 30 at the end, and finally 20 at position 1. The output shows the resulting circular linked list structure.

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

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

struct Node* last = NULL;

void display() {
    if (last == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* temp = last->next;
    do {
        printf("%d -> ", temp->data);
        temp = temp->next;
    } while (temp != last->next);
    printf("(head)\n");
}

void insertAtBeginning(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;

    if (last == NULL) {
        newNode->next = newNode;
        last = newNode;
    } else {
        newNode->next = last->next;
        last->next = newNode;
    }
}

int main() {
    // Your main code here
}
```

Output:

- 1. Insert at Beginning
- 2. Insert at End
- 3. Insert at Position
- 4. Display
- 5. Exit

Enter your choice: 1
Enter value: 10
Inserted 10 at beginning

Enter your choice: 2
Enter value: 30
Inserted 30 at end

Enter your choice: 3
Enter value and position: 20 1
Inserted 20 after position 1

Enter your choice: 4
Circular Linked List: 10 -> 20 -> 30 -> (head)

Enter your choice: 5

Programiz
C Online Compiler

Premium Coding Courses by Programiz

Output

```
main.c
64
65- int main() {
66    int choice, value, position;
67
68    while (1) {
69        printf("1. Insert at Beginning\n2. Insert at End\n3. Insert at Position\n4. Display\n5. Exit\nEnter your choice: ");
70        scanf("%d", &choice);
71
72        switch(choice) {
73            case 1:
74                printf("Enter value: ");
75                scanf("%d", &value);
76                insertAtBeginning(value);
77                break;
78            case 2:
79                printf("Enter value: ");
80                scanf("%d", &value);
81                insertAtEnd(value);
82                break;
83            case 3:
84                printf("Enter value and position: ");
85                scanf("%d %d", &value, &position);
86                insertAtPosition(value, position);
87                break;
88            case 4:
89                display();
90                break;
91            case 5:
92                return 0;
93            default:
94                printf("Invalid choice\n");
95        }
96
97    }
98
99    return 0;
100 }
```

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

Enter your choice: 2
Enter value: 20
Inserted 20 at end

Enter your choice: 3
Enter value and position: 20 1
Inserted 20 after position 1

Enter your choice: 4
Circular Linked List: 10 -> 20 -> 30 -> (head)

Enter your choice: 5

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

The screenshot shows the Programiz C Online Compiler interface. The code in main.c implements a circular linked list with functions for insertion at the end, displaying the list, and deleting from the beginning. The output window shows the menu options and the results of selecting 'Delete from Beginning'.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct Node {
5     int data;
6     struct Node* next;
7 };
8
9 struct Node* last = NULL;
10
11
12 void insertAtEnd(int value) {
13     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
14     newNode->data = value;
15     if (last == NULL) {
16         newNode->next = newNode;
17         last = newNode;
18     } else {
19         newNode->next = last->next;
20         last->next = newNode;
21         last = newNode;
22     }
23 }
24
25
26 void display() {
27     if (last == NULL) {
28         printf("List is empty\n");
29         return;
30     }
31     struct Node* temp = last->next;
32     printf("Circular Linked List: ");
33     do {
34         printf("%d -> ", temp->data);
35         temp = temp->next;
36     } while (temp != last->next);
37     printf("(head)\n");
38 }
```

Output:

- 1. Delete from Beginning
- 2. Delete from End
- 3. Delete from Position
- 4. Display
- 5. Exit

Enter your choice: 4
Circular Linked List: 10 -> 20 -> 30 -> 40 -> (head)

Enter your choice: 1
Deleted 10 from beginning

Enter your choice: 2
Deleted 40 from end

Enter your choice: 3
Enter position to delete: 2
Deleted 30 from position 2

Enter your choice: 4
Circular Linked List: 20 -> (head)

The screenshot shows the Programiz C Online Compiler interface. The code in main.c adds functions for deleting from the end and from a specific position. The output window shows the menu options and the results of selecting 'Delete from End'.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct Node {
5     int data;
6     struct Node* next;
7 };
8
9 struct Node* last = NULL;
10
11
12 void insertAtEnd(int value) {
13     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
14     newNode->data = value;
15     if (last == NULL) {
16         newNode->next = newNode;
17         last = newNode;
18     } else {
19         newNode->next = last->next;
20         last->next = newNode;
21         last = newNode;
22     }
23 }
24
25
26 void display() {
27     if (last == NULL) {
28         printf("List is empty\n");
29         return;
30     }
31     struct Node* temp = last->next;
32     printf("Circular Linked List: ");
33     do {
34         printf("%d -> ", temp->data);
35         temp = temp->next;
36     } while (temp != last->next);
37     printf("(head)\n");
38 }
39
40
41 void deleteFromBeginning() {
42     if (last == NULL) {
43         printf("List is empty\n");
44         return;
45     }
46     struct Node* temp = last->next;
47
48     if (last == last->next) { // Only one node
49         printf("Deleted %d from beginning\n", temp->data);
50         free(temp);
51         last = NULL;
52     } else {
53         last->next = temp->next;
54         printf("Deleted %d from beginning\n", temp->data);
55         free(temp);
56     }
57 }
58
59
60 void deleteFromEnd() {
61     if (last == NULL) {
62         printf("List is empty\n");
63         return;
64     }
65     struct Node* temp = last->next;
66     if (last == last->next) { // Only one node
67         printf("Deleted %d from end\n", last->data);
68         free(last);
69         last = NULL;
70     } else {
71         struct Node* prev = NULL;
72         for (temp = last->next; temp != last; prev = temp)
73             temp = temp->next;
74         prev->next = last;
75         free(temp);
76     }
77 }
```

Output:

- 1. Delete from Beginning
- 2. Delete from End
- 3. Delete from Position
- 4. Display
- 5. Exit

Enter your choice: 4
Circular Linked List: 10 -> 20 -> 30 -> 40 -> (head)

Enter your choice: 1
Deleted 10 from beginning

Enter your choice: 2
Deleted 40 from end

Enter your choice: 3
Enter position to delete: 2
Deleted 30 from position 2

Enter your choice: 4
Circular Linked List: 20 -> (head)

Programiz
C Online Compiler

Premium Coding Courses by Programiz

Run Output

```
main.c
74-     struct Node* prev = NULL;
75-     while (temp->next != last->next) {
76-         prev = temp;
77-         temp = temp->next;
78-     }
79-     prev->next = last->next;
80-     printf("Deleted %d from end\n", last->data);
81-     free(last);
82-     last = prev;
83- }
84-
85- void deleteFromPosition(int position) {
86-     if (last == NULL) {
87-         printf("List is empty\n");
88-         return;
89-     }
90-
91-     if (position == 1) {
92-         deleteFromBeginning();
93-         return;
94-     }
95-
96-     struct Node* temp = last->next;
97-     struct Node* prev = NULL;
98-     for (int i = 1; i < position && temp != last; i++) {
99-         prev = temp;
100-        temp = temp->next;
101-    }
102-
103-    if (temp == last && position != 1) {
104-        deleteFromEnd();
105-    } else if (temp == last->next) {
106-        prev->next = temp->next;
107-        printf("Deleted %d from position %d\n", temp->data, position);
108-        free(temp);
109-    } else {
110-    }
```

1. Delete from Beginning
2. Delete from End
3. Delete from Position
4. Display
5. Exit
Enter your choice: 4
Circular Linked List: 10 -> 20 -> 30 -> 40 -> (head)

Enter your choice: 1
Deleted 10 from beginning

Enter your choice: 2
Deleted 40 from end

Enter your choice: 3
Enter position to delete: 2
Deleted 30 from position 2

Enter your choice: 4
Circular Linked List: 20 -> (head)

Programiz
C Online Compiler

Premium Coding Courses by Programiz

Run Output

```
main.c
110- int main() {
111-     int choice, position;
112-
113-     insertAtEnd(10);
114-     insertAtEnd(20);
115-     insertAtEnd(30);
116-     insertAtEnd(40);
117-
118-     while (1) {
119-         printf("\n1. Delete from Beginning\n2. Delete from End\n3. Delete from Position\n4. Display\n5. Exit\nEnter your choice: ");
120-         scanf("%d", &choice);
121-
122-         switch (choice) {
123-             Case 1:
124-                 deleteFromBeginning();
125-                 break;
126-             Case 2:
127-                 deleteFromEnd();
128-                 break;
129-             Case 3:
130-                 printf("Enter position to delete: ");
131-                 scanf("%d", &position);
132-                 deleteFromPosition(position);
133-                 break;
134-             Case 4:
135-                 display();
136-                 break;
137-             Case 5:
138-                 return 0;
139-             default:
140-                 printf("Invalid choice\n");
141-         }
142-     }
143-     return 0;
144- }
```

1. Delete from Beginning
2. Delete from End
3. Delete from Position
4. Display
5. Exit
Enter your choice: 4
Circular Linked List: 10 -> 20 -> 30 -> 40 -> (head)

Enter your choice: 1
Deleted 10 From beginning

Enter your choice: 2
Deleted 40 From end

Enter your choice: 3
Enter position to delete: 2
Deleted 30 From position 2

Enter your choice: 4
Circular Linked List: 20 -> (head)

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

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is as follows:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct Node {
5     int data;
6     struct Node* prev;
7     struct Node* next;
8 };
9
10 struct Node* head = NULL;
11
12
13 void display() {
14     struct Node* temp = head;
15     printf("Doubly Linked List: ");
16     while (temp != NULL) {
17         printf("%d <-> ", temp->data);
18         temp = temp->next;
19     }
20     printf("NULL\n");
21 }
22
23
24 void insertAtBeginning(int value) {
25     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
26     newNode->data = value;
27     newNode->prev = NULL;
28     newNode->next = head;
29
30     if (head != NULL)
31         head->prev = newNode;
32
33     head = newNode;
34
35     printf("Inserted %d at beginning\n", value);
36 }
37
```

The output window shows the execution of the program:

- 1. Insert at Beginning
- 2. Insert at End
- 3. Insert at Position
- 4. Display
- 5. Exit

Enter your choice: 1
Enter value: 10
Inserted 10 at beginning

Enter your choice: 2
Enter value: 30
Inserted 30 at end

Enter your choice: 3
Enter value and position: 20 1
Inserted 20 after position 1

Enter your choice: 4
Doubly Linked List: 10 <-> 20 <-> 30 <-> NULL

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is as follows:

```
1
2
39 void insertAtEnd(int value) {
40     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
41     newNode->data = value;
42     newNode->next = NULL;
43
44     if (head == NULL) {
45         newNode->prev = NULL;
46         head = newNode;
47         printf("Inserted %d at end\n", value);
48         return;
49     }
50
51     struct Node* temp = head;
52     while (temp->next != NULL)
53         temp = temp->next;
54
55     temp->next = newNode;
56     newNode->prev = temp;
57
58     printf("Inserted %d at end\n", value);
59 }
60
61
62 void insertAtPosition(int value, int position) {
63     if (head == NULL || position == 0) {
64         insertAtBeginning(value);
65         return;
66     }
67
68     struct Node* temp = head;
69     for (int i = 1; i < position && temp->next != NULL; i++) {
70         temp = temp->next;
71     }
72
73     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

The output window shows the execution of the program:

- 1. Insert at Beginning
- 2. Insert at End
- 3. Insert at Position
- 4. Display
- 5. Exit

Enter your choice: 1
Enter value: 10
Inserted 10 at beginning

Enter your choice: 2
Enter value: 30
Inserted 30 at end

Enter your choice: 3
Enter value and position: 20 1
Inserted 20 after position 1

Enter your choice: 4
Doubly Linked List: 10 <-> 20 <-> 30 <-> NULL

*** Session Ended. Please Run the code again ***

Programiz
C Online Compiler

Premium Coding
Courses by Programiz

Programiz PRO

main.c

```
72 struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
73 newNode->data = value;
74
75 newNode->next = temp->next;
76 newNode->prev = temp;
77
78 if (temp->next != NULL)
79     temp->next->prev = newNode;
80
81 temp->next = newNode;
82
83 printf("Inserted %d after position %d\n", value, position);
84 }
85
86
87 int main() {
88     int choice, value, position;
89
90     while (1) {
91         printf("1. Insert at Beginning\n2. Insert at End\n3. Insert at Position\n4. Display\n");
92         printf("Enter your choice: ");
93         scanf("%d", &choice);
94
95         switch (choice) {
96             case 1:
97                 printf("Enter value: ");
98                 scanf("%d", &value);
99                 insertAtBeginning(value);
100                break;
101            case 2:
102                 printf("Enter value: ");
103                 scanf("%d", &value);
104                 insertAtEnd(value);
105                break;
106            case 3:
107                 printf("Enter value and position: ");
108                 scanf("%d %d", &value, &position);
109                 insertAtPosition(value, position);
110                break;
111            case 4:
112                 display();
113                 break;
114            case 5:
115                 return 0;
116            default:
117                 printf("Invalid choice\n");
118         }
119     }
120 }
```

Output

```
* 1. Insert at Beginning
  2. Insert at End
  3. Insert at Position
  4. Display
  5. Exit
Enter your choice: 1
Enter value: 10
Inserted 10 at beginning

Enter your choice: 2
Enter value: 30
Inserted 30 at end

Enter your choice: 3
Enter value and position: 20 1
Inserted 20 after position 1

Enter your choice: 4
Doubly Linked List: 10 <-> 20 <-> 30 <-> NULL
*** Session Ended. Please Run the code again ***
```

Programiz
C Online Compiler

Premium Coding
Courses by Programiz

Programiz PRO

main.c

```
83 }
84
85
86 int main() {
87     int choice, value, position;
88
89     while (1) {
90         printf("1. Insert at Beginning\n2. Insert at End\n3. Insert at Position\n4. Display\n");
91         printf("Enter your choice: ");
92         scanf("%d", &choice);
93
94         switch (choice) {
95             case 1:
96                 printf("Enter value: ");
97                 scanf("%d", &value);
98                 insertAtBeginning(value);
99                 break;
100            case 2:
101                 printf("Enter value: ");
102                 scanf("%d", &value);
103                 insertAtEnd(value);
104                 break;
105            case 3:
106                 printf("Enter value and position: ");
107                 scanf("%d %d", &value, &position);
108                 insertAtPosition(value, position);
109                 break;
110            case 4:
111                 display();
112                 break;
113            case 5:
114                 return 0;
115            default:
116                 printf("Invalid choice\n");
117         }
118     }
119 }
```

Output

```
* 1. Insert at Beginning
  2. Insert at End
  3. Insert at Position
  4. Display
  5. Exit
Enter your choice: 1
Enter value: 10
Inserted 10 at beginning

Enter your choice: 2
Enter value: 30
Inserted 30 at end

Enter your choice: 3
Enter value and position: 20 1
Inserted 20 after position 1

Enter your choice: 4
Doubly Linked List: 10 <-> 20 <-> 30 <-> NULL
*** Session Ended. Please Run the code again ***
```

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

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is as follows:

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct Node {
5     int data;
6     struct Node *prev, *next;
7 };
8
9 struct Node *head = NULL;
10
11 // Insert at end for testing
12 void insert(int value) {
13     struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
14     newNode->data = value;
15     newNode->next = NULL;
16
17     if (head == NULL) {
18         newNode->prev = NULL;
19         head = newNode;
20     } else {
21         struct Node *temp = head;
22         while (temp->next) temp = temp->next;
23         temp->next = newNode;
24         newNode->prev = temp;
25     }
26 }
27
28 // Display list
29 void display() {
30     struct Node *temp = head;
31     printf("List: ");
32     while (temp) {
33         printf("%d <-> ", temp->data);
34         temp = temp->next;
35     }
36     printf("NULL\n");
37 }
38
```

The output window shows the menu options and the execution of the program:

- 1. Delete Beginning
- 2. Delete End
- 3. Delete at Position
- 4. Display
- 5. Exit

Enter choice: 4
List: 10 <-> 20 <-> 30 <-> 40 <-> NULL

Enter choice: 1
Deleted 10 from beginning

Enter choice: 3
Enter position: 2
Deleted 30 from position 2

Enter choice: 2
Deleted 40 from end

Enter choice: 4
List: 20 <-> NULL

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is as follows:

```
main.c
39 // Delete from beginning
40 void deleteBeginning() {
41     if (!head) {
42         printf("List is empty\n");
43         return;
44     }
45     struct Node *temp = head;
46     head = head->next;
47     if (head) head->prev = NULL;
48     printf("Deleted %d from beginning\n", temp->data);
49     free(temp);
50 }
51
52 // Delete from end
53 void deleteEnd() {
54     if (!head) {
55         printf("List is empty\n");
56         return;
57     }
58     struct Node *temp = head;
59     if (!temp->next) {
60         printf("Deleted %d from end\n", temp->data);
61         free(temp);
62         head = NULL;
63         return;
64     }
65     while (temp->next) temp = temp->next;
66     printf("Deleted %d from end\n", temp->data);
67     temp->prev->next = NULL;
68     free(temp);
69 }
70
71 // Delete from position
72 void deletePosition(int pos) {
73     if (!head || pos <= 0) {
74         printf("Invalid position\n");
75         return;
76     }
```

The output window shows the menu options and the execution of the program:

- 1. Delete Beginning
- 2. Delete End
- 3. Delete at Position
- 4. Display
- 5. Exit

Enter choice: 4
List: 10 <-> 20 <-> 30 <-> 40 <-> NULL

Enter choice: 1
Deleted 10 from beginning

Enter choice: 3
Enter position: 2
Deleted 30 from position 2

Enter choice: 2
Deleted 40 from end

Enter choice: 4
List: 20 <-> NULL

*** Session Ended. Please Run the code again ***

Programiz
C Online Compiler

Premium Coding Courses by Programiz

Output

```
main.c
14     printf("Invalid position\n");
15     return;
16 }
17 if (pos == 1) {
18     deleteBeginning();
19     return;
20 }
21 struct Node *temp = head;
22 for (int i = 1; temp->next && i < pos; i++)
23     temp = temp->next;
24
25 if (!temp) {
26     printf("Position out of range\n");
27     return;
28 }
29
30 if (temp->next)
31     temp->next->prev = temp->prev;
32 if (temp->prev)
33     temp->prev->next = temp->next;
34
35 printf("Deleted %d from position %d\n", temp->data, pos);
36 free(temp);
37 }
38
39 // Main
40 int main() {
41     insert(10);
42     insert(20);
43     insert(30);
44     insert(40);
45
46     int choice, pos;
47     while (1) {
48         printf("\n1. Delete Beginning\n2. Delete End\n3. Delete at Position\n4. Display\n");
49         EnterUser choice: "3";
50         scanf("%d", &choice);
51         switch (choice) {
52             case 1: deletedBeginning(); break;
53             case 2: deletedEnd(); break;
54             case 3:
55                 printf("Enter position: ");
56                 scanf("%d", &pos);
57                 deletePosition(pos);
58                 break;
59             case 4: display(); break;
60             case 5: return 0;
61             default: printf("Invalid choice\n");
62         }
63     }
64 }
```

1. Delete Beginning
2. Delete End
3. Delete at Position
4. Display
5. Exit
Enter choice: 4
List: 10 <-> 20 <-> 30 <-> 40 <-> NULL
Enter choice: 1
Deleted 10 from beginning
Enter choice: 3
Enter position: 2
Deleted 30 from position 2
Enter choice: 2
Deleted 40 from end
Enter choice: 4
List: 20 <-> NULL
*** Session Ended. Please Run the code again ***

Programiz
C Online Compiler

Premium Coding Courses by Programiz

Output

```
main.c
46 }
47
48 if (temp->next)
49     temp->next->prev = temp->prev;
50 if (temp->prev)
51     temp->prev->next = temp->next;
52
53 printf("Deleted %d from position %d\n", temp->data, pos);
54 free(temp);
55 }
56
57 // Main
58 int main() {
59     insert(10);
60     insert(20);
61     insert(30);
62     insert(40);
63
64     int choice, pos;
65     while (1) {
66         printf("\n1. Delete Beginning\n2. Delete End\n3. Delete at Position\n4. Display\n");
67         EnterUser choice: "3";
68         scanf("%d", &choice);
69         switch (choice) {
70             case 1: deletedBeginning(); break;
71             case 2: deletedEnd(); break;
72             case 3:
73                 printf("Enter position: ");
74                 scanf("%d", &pos);
75                 deletePosition(pos);
76                 break;
77             case 4: display(); break;
78             case 5: return 0;
79             default: printf("Invalid choice\n");
80         }
81     }
82 }
```

1. Delete Beginning
2. Delete End
3. Delete at Position
4. Display
5. Exit
Enter choice: 4
List: 10 <-> 20 <-> 30 <-> 40 <-> NULL
Enter choice: 1
Deleted 10 from beginning
Enter choice: 3
Enter position: 2
Deleted 30 from position 2
Enter choice: 2
Deleted 40 from end
Enter choice: 4
List: 20 <-> NULL
*** Session Ended. Please Run the code again ***

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

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is as follows:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 struct Node {
6     int data;
7     struct Node* left;
8     struct Node* right;
9 };
10
11 struct Node* createNode(int value) {
12     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
13     newNode->data = value;
14     newNode->left = newNode->right = NULL;
15     return newNode;
16 }
17
18
19 void insert(struct Node** root, int value) {
20     struct Node* newNode = createNode(value);
21     if (*root == NULL) {
22         *root = newNode;
23         return;
24     }
25
26     struct Node* queue[100];
27     int front = 0, rear = 0;
28
29     queue[rear++] = *root;
30
31     while (front != rear) {
32         struct Node* temp = queue[front++];
33
34         if (temp->left == NULL) {
35             temp->left = newNode;
36             return;
37         }
38     }
39
40     if (temp->right == NULL) {
41         temp->right = newNode;
42         return;
43     }
44     else {
45         queue[rear++] = temp->right;
46     }
47 }
48
49
50
51
52 void inorder(struct Node* root) {
53     if (root != NULL) {
54         inorder(root->left);
55         printf("%d ", root->data);
56         inorder(root->right);
57     }
58 }
59
60
61 int main() {
62     struct Node* root = NULL;
63
64     insert(&root, 1);
65     insert(&root, 2);
66     insert(&root, 3);
67     insert(&root, 4);
68     insert(&root, 5);
69
70     printf("In-order traversal of the Binary Tree: ");
71     inorder(root); // Output the tree structure in in-order
72     printf("\n");
73
74     return 0;
75 }
```

The output window shows the result of running the code: "In-order traversal of the Binary Tree: 4 2 5 1 3" followed by "*** Code Execution Successful ***".

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is as follows:

```
38     } else {
39         queue[rear++] = temp->left;
40     }
41
42     if (temp->right == NULL) {
43         temp->right = newNode;
44         return;
45     }
46     else {
47         queue[rear++] = temp->right;
48     }
49 }
50
51
52 void inorder(struct Node* root) {
53     if (root != NULL) {
54         inorder(root->left);
55         printf("%d ", root->data);
56         inorder(root->right);
57     }
58 }
59
60
61 int main() {
62     struct Node* root = NULL;
63
64     insert(&root, 1);
65     insert(&root, 2);
66     insert(&root, 3);
67     insert(&root, 4);
68     insert(&root, 5);
69
70     printf("In-order traversal of the Binary Tree: ");
71     inorder(root); // Output the tree structure in in-order
72     printf("\n");
73
74     return 0;
75 }
```

The output window shows the result of running the code: "In-order traversal of the Binary Tree: 4 2 5 1 3" followed by "*** Code Execution Successful ***".

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

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is as follows:

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 struct Node {
6     int data;
7     struct Node* left;
8     struct Node* right;
9 };
10
11
12 struct Node* createNode(int value) {
13     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
14     newNode->data = value;
15     newNode->left = NULL;
16     newNode->right = NULL;
17 }
18
19
20 struct Node* insert(struct Node* root, int value) {
21     if (root == NULL)
22         return createNode(value);
23
24     if (value < root->data)
25         root->left = insert(root->left, value);
26     else if (value > root->data)
27         root->right = insert(root->right, value);
28
29     return root;
30 }
31
32
33 struct Node* minValueNode(struct Node* node) {
34     struct Node* current = node;
35
36     while (current && current->left != NULL)
37         current = current->left;
38 }
```

The output window shows the execution results:

```
Inorder traversal of the original tree:
20 30 40 50 60 70 80
Deleting 50
Inorder traversal after deletion:
20 30 40 60 70 80
*** Code Execution Successful ***
```

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is as follows:

```
main.c
38
39     return current;
40 }
41
42
43 struct Node* deleteNode(struct Node* root, int key) {
44     if (root == NULL) return root;
45
46
47     if (key < root->data)
48         root->left = deleteNode(root->left, key);
49     else if (key > root->data)
50         root->right = deleteNode(root->right, key);
51     else {
52
53         if (root->left == NULL) {
54             struct Node* temp = root->right;
55             free(root);
56             return temp;
57         }
58         else if (root->right == NULL) {
59             struct Node* temp = root->left;
60             free(root);
61             return temp;
62         }
63
64         struct Node* temp = minValueNode(root->right);
65
66         root->data = temp->data;
67
68         root->right = deleteNode(root->right, temp->data);
69
70     }
71
72     return root;
73 }
```

The output window shows the execution results:

```
Inorder traversal of the original tree:
20 30 40 50 60 70 80
Deleting 50
Inorder traversal after deletion:
20 30 40 60 70 80
*** Code Execution Successful ***
```

Programiz
C Online Compiler

Premium Coding Courses by Programiz

Output

Inorder traversal of the original tree:
20 30 40 50 60 70 80

Deleting 50
Inorder traversal after deletion:
20 30 40 60 70 80

--- Code Execution Successful ---

```
main.c
72 }
73 return root;
74 }
75 }
76 }
77 }
78 void inorder(struct Node* root) {
79     if (root != NULL) {
80         inorder(root->left);
81         printf("%d ", root->data);
82         inorder(root->right);
83     }
84 }
85 }
86 }
87 int main() {
88     struct Node* root = NULL;
89
90     root = insert(root, 50);
91     root = insert(root, 30);
92     root = insert(root, 70);
93     root = insert(root, 20);
94     root = insert(root, 40);
95     root = insert(root, 60);
96     root = insert(root, 80);
97
98     printf("Inorder traversal of the original tree:\n");
99     inorder(root);
100
101     printf("\nDeleting 50\n");
102     root = deleteNode(root, 50);
103
104     printf("Inorder traversal after deletion:\n");
105     inorder(root);
106
107
108     return 0;
109 }
```

Programiz
C Online Compiler

Premium Coding Courses by Programiz

Output

Inorder traversal of the original tree:
20 30 40 50 60 70 80

Deleting 50
Inorder traversal after deletion:
20 30 40 60 70 80

--- Code Execution Successful ---

```
main.c
74 return root;
75 }
76 }
77 }
78 void inorder(struct Node* root) {
79     if (root != NULL) {
80         inorder(root->left);
81         printf("%d ", root->data);
82         inorder(root->right);
83     }
84 }
85 }
86 }
87 int main() {
88     struct Node* root = NULL;
89
90     root = insert(root, 50);
91     root = insert(root, 30);
92     root = insert(root, 70);
93     root = insert(root, 20);
94     root = insert(root, 40);
95     root = insert(root, 60);
96     root = insert(root, 80);
97
98     printf("Inorder traversal of the original tree:\n");
99     inorder(root);
100
101     printf("\nDeleting 50\n");
102     root = deleteNode(root, 50);
103
104     printf("Inorder traversal after deletion:\n");
105     inorder(root);
106
107
108     return 0;
109 }
```

21. Write a program in C to implement recursive tree traversals (Inorder; Preorder; Postorder).

The screenshot shows the Programiz C Online Compiler interface. The code editor contains `main.c` with the following content:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 struct Node {
6     int data;
7     struct Node* left;
8     struct Node* right;
9 };
10
11 struct Node* createNode(int value) {
12     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
13     newNode->data = value;
14     newNode->left = newNode->right = NULL;
15     return newNode;
16 }
17
18
19 void inorder(struct Node* root) {
20     if (root != NULL) {
21         inorder(root->left);
22         printf("%d ", root->data);
23         inorder(root->right);
24     }
25 }
26
27 void preorder(struct Node* root) {
28     if (root != NULL) {
29         printf("%d ", root->data);
30         preorder(root->left);
31         preorder(root->right);
32     }
33 }
34
35
36 void postorder(struct Node* root) {
37     if (root != NULL) {
38         postorder(root->left);
39         postorder(root->right);
40         printf("%d ", root->data);
41     }
42 }
43
44
45
46 int main() {
47
48     struct Node* root = createNode(1);
49     root->left = createNode(2);
50     root->right = createNode(3);
51     root->left->left = createNode(4);
52     root->left->right = createNode(5);
53     root->right->left = createNode(6);
54     root->right->right = createNode(7);
55
56     printf("Inorder Traversal: ");
57     inorder(root);
58     printf("\n");
59
60     printf("Preorder Traversal: ");
61     preorder(root);
62     printf("\n");
63
64     printf("Postorder Traversal: ");
65     postorder(root);
66     printf("\n");
67
68     return 0;
69 }
```

The output window shows the results of the code execution:

```
Inorder Traversal: 4 2 5 1 6 3 7
Preorder Traversal: 1 2 4 5 3 6 7
Postorder Traversal: 4 5 2 6 7 3 1
--- Code Execution successful ---
```

The screenshot shows the Programiz C Online Compiler interface. The code editor contains `main.c` with the following content:

```
34 }
35
36
37 void postorder(struct Node* root) {
38     if (root != NULL) {
39         postorder(root->left);
40         postorder(root->right);
41         printf("%d ", root->data);
42     }
43 }
44
45
46 int main() {
47
48     struct Node* root = createNode(1);
49     root->left = createNode(2);
50     root->right = createNode(3);
51     root->left->left = createNode(4);
52     root->left->right = createNode(5);
53     root->right->left = createNode(6);
54     root->right->right = createNode(7);
55
56     printf("Inorder Traversal: ");
57     inorder(root);
58     printf("\n");
59
60     printf("Preorder Traversal: ");
61     preorder(root);
62     printf("\n");
63
64     printf("Postorder Traversal: ");
65     postorder(root);
66     printf("\n");
67
68     return 0;
69 }
```

The output window shows the results of the code execution:

```
Inorder Traversal: 4 2 5 1 6 3 7
Preorder Traversal: 1 2 4 5 3 6 7
Postorder Traversal: 4 5 2 6 7 3 1
--- Code Execution successful ---
```

22. Write a program in C to Sort a list using Bubble Sort.

The screenshot shows the Programiz Online Compiler interface. The code editor contains a C program named main.c. The code implements a bubble sort algorithm to sort an array of integers. It includes functions for displaying the array and performing the sort. The output window shows the original array [64, 34, 25, 12, 22, 11, 90] and the sorted array [11, 12, 22, 25, 34, 64, 90].

```
1 #include <stdio.h>
2
3 void bubbleSort(int arr[], int n) {
4     for (int i = 0; i < n-1; i++) {
5         for (int j = 0; j < n-i-1; j++) {
6             if (arr[j] > arr[j+1]) {
7                 int temp = arr[j];
8                 arr[j] = arr[j+1];
9                 arr[j+1] = temp;
10            }
11        }
12    }
13}
14
15
16 void displayArray(int arr[], int n) {
17     for (int i = 0; i < n; i++) {
18         printf("%d ", arr[i]);
19     }
20     printf("\n");
21 }
22
23
24
25
26 int main() {
27     int arr[] = {64, 34, 25, 12, 22, 11, 90}; // Example array
28     int n = sizeof(arr) / sizeof(arr[0]);
29
30     printf("Original Array: ");
31     displayArray(arr, n);
32
33     bubbleSort(arr, n);
34
35     printf("Sorted Array: ");
36     displayArray(arr, n);
37
38     return 0;
39 }
```

This screenshot shows the same C program for bubble sort in the Programiz Online Compiler. The code and output are identical to the first screenshot, demonstrating the execution of the bubble sort algorithm on the provided example array.

```
1 #include <stdio.h>
2
3 void bubbleSort(int arr[], int n) {
4     for (int i = 0; i < n-1; i++) {
5         for (int j = 0; j < n-i-1; j++) {
6             if (arr[j] > arr[j+1]) {
7                 int temp = arr[j];
8                 arr[j] = arr[j+1];
9                 arr[j+1] = temp;
10            }
11        }
12    }
13}
14
15
16 void displayArray(int arr[], int n) {
17     for (int i = 0; i < n; i++) {
18         printf("%d ", arr[i]);
19     }
20     printf("\n");
21 }
22
23
24
25
26 int main() {
27     int arr[] = {64, 34, 25, 12, 22, 11, 90}; // Example array
28     int n = sizeof(arr) / sizeof(arr[0]);
29
30     printf("Original Array: ");
31     displayArray(arr, n);
32
33     bubbleSort(arr, n);
34
35     printf("Sorted Array: ");
36     displayArray(arr, n);
37
38     return 0;
39 }
```

23. Write a program in C to Sort a list using Selection Sort.

The screenshot shows the Programiz C Online Compiler interface. The code editor contains the following C program:

```
1 #include <stdio.h>
2
3 void selectionSort(int arr[], int n) {
4     for (int i = 0; i < n-1; i++) {
5         int minIndex = i;
6         for (int j = i+1; j < n; j++) {
7             if (arr[j] < arr[minIndex]) {
8                 minIndex = j;
9             }
10        }
11
12        if (minIndex != i) {
13            int temp = arr[i];
14            arr[i] = arr[minIndex];
15            arr[minIndex] = temp;
16        }
17    }
18 }
19
20
21 void displayArray(int arr[], int n) {
22     for (int i = 0; i < n; i++) {
23         printf("%d ", arr[i]);
24     }
25     printf("\n");
26 }
27
28
29
30 int main() {
31     int arr[] = {64, 34, 25, 12, 22, 11, 90};
32     int n = sizeof(arr) / sizeof(arr[0]);
33
34     printf("Original Array: ");
35     displayArray(arr, n);
36
37     selectionSort(arr, n);
38 }
```

The output window shows the original array [64, 34, 25, 12, 22, 11, 90] and the sorted array [11, 12, 22, 25, 34, 64, 90]. A message "Code Execution Successful" is displayed.

This screenshot is identical to the one above, showing the same C program for selection sort and its execution output. The original array is [64, 34, 25, 12, 22, 11, 90] and the sorted array is [11, 12, 22, 25, 34, 64, 90]. The message "Code Execution Successful" is present.

24. Write a program in C to sort a list using Quick Sort.

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is:

```
main.c
1 #include <stdio.h>
2
3
4 void swap(int* a, int* b) {
5     int temp = *a;
6     *a = *b;
7     *b = temp;
8 }
9
10
11 int partition(int arr[], int low, int high) {
12     int pivot = arr[high];
13     int i = (low - 1);
14
15     for (int j = low; j <= high - 1; j++) {
16
17         if (arr[j] < pivot) {
18             i++;
19             swap(&arr[i], &arr[j]);
20         }
21     }
22
23     swap(&arr[i + 1], &arr[high]);
24     return (i + 1);
25 }
26
27
28 void quickSort(int arr[], int low, int high) {
29     if (low < high) {
30
31         int pi = partition(arr, low, high);
32
33         quickSort(arr, low, pi - 1);
34         quickSort(arr, pi + 1, high);
35     }
36 }
37 }
```

The output window shows the execution results:

Original Array: 64 34 25 12 22 11 90
Sorted Array: 11 12 22 25 34 64 90
--- Code Execution Successful ---

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is:

```
main.c
24     return 0;
25 }
26
27
28 void quickSort(int arr[], int low, int high) {
29     if (low < high) {
30
31         int pi = partition(arr, low, high);
32
33         quickSort(arr, low, pi - 1);
34         quickSort(arr, pi + 1, high);
35     }
36 }
37
38
39 /
40 void displayArray(int arr[], int n) {
41     for (int i = 0; i < n; i++) {
42         printf("%d ", arr[i]);
43     }
44     printf("\n");
45 }
46
47 int main() {
48     int arr[] = {64, 34, 25, 12, 22, 11, 90};
49     int n = sizeof(arr) / sizeof(arr[0]);
50
51     printf("Original Array: ");
52     displayArray(arr, n);
53
54     quickSort(arr, 0, n - 1);
55
56     printf("Sorted Array: ");
57     displayArray(arr, n);
58
59     return 0;
60 }
```

The output window shows the execution results:

Original Array: 64 34 25 12 22 11 90
Sorted Array: 11 12 22 25 34 64 90
--- Code Execution Successful ---

25. Write a program in C to sort a list using Merge Sort.

The screenshot shows the Programiz C Online Compiler interface. The code editor contains the following C code for Merge Sort:

```
1 #include <stdio.h>
2
3 void merge(int arr[], int left, int mid, int right) {
4     int i, j, k;
5     int n1 = mid - left + 1;
6     int n2 = right - mid;
7
8     int L[n1], R[n2];
9
10    for (i = 0; i < n1; i++)
11        L[i] = arr[left + i];
12    for (j = 0; j < n2; j++)
13        R[j] = arr[mid + 1 + j];
14
15    i = 0, j = 0, k = left;
16    while (i < n1 && j < n2) {
17        if (L[i] <= R[j]) {
18            arr[k] = L[i++];
19        } else {
20            arr[k] = R[j++];
21        }
22    }
23
24    while (i < n1)
25        arr[k++] = L[i++];
26
27    while (j < n2)
28        arr[k++] = R[j++];
29
30}
31
32 void mergeSort(int arr[], int left, int right) {
33    if (left < right) {
34        int mid = (left + right) / 2;
35
36        mergeSort(arr, left, mid);
37        mergeSort(arr, mid + 1, right);
38
39        merge(arr, left, mid, right);
40    }
41
42}
43
44
45 int main() {
46    int arr[] = {64, 34, 25, 12, 22, 11, 90};
47    int size = sizeof(arr) / sizeof(arr[0]);
48
49    printf("Original Array: ");
50    displayArray(arr, size);
51    printf("\n");
52
53    mergeSort(arr, 0, size - 1);
54
55    printf("Sorted Array: ");
56    displayArray(arr, size);
57
58    return 0;
59}
```

The output window shows the execution results:

```
Original Array: 64 34 25 12 22 11 90
Sorted Array: 11 12 22 25 34 64 90
*** Code Execution Successful ***
```

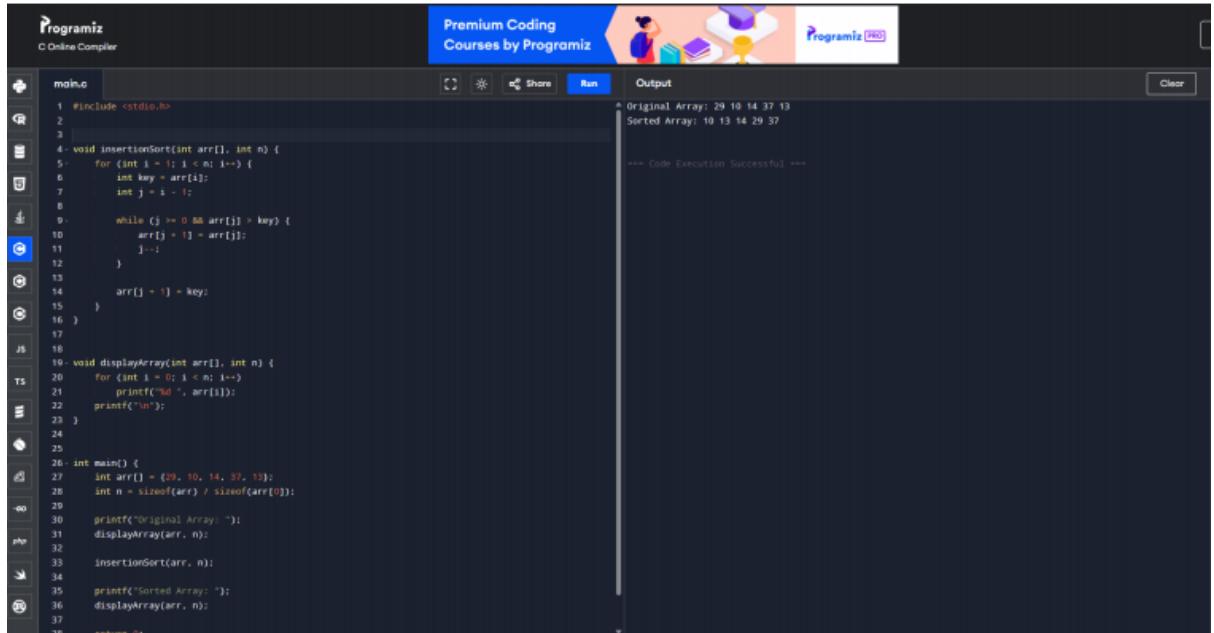
The screenshot shows the Programiz C Online Compiler interface. The code editor contains the full C program for Merge Sort, including the `main()` function:

```
1 #include <stdio.h>
2
3 void merge(int arr[], int left, int mid, int right) {
4     int i, j, k;
5     int n1 = mid - left + 1;
6     int n2 = right - mid;
7
8     int L[n1], R[n2];
9
10    for (i = 0; i < n1; i++)
11        L[i] = arr[left + i];
12    for (j = 0; j < n2; j++)
13        R[j] = arr[mid + 1 + j];
14
15    i = 0, j = 0, k = left;
16    while (i < n1 && j < n2) {
17        if (L[i] <= R[j]) {
18            arr[k] = L[i++];
19        } else {
20            arr[k] = R[j++];
21        }
22    }
23
24    while (i < n1)
25        arr[k++] = L[i++];
26
27    while (j < n2)
28        arr[k++] = R[j++];
29
30}
31
32 void mergeSort(int arr[], int left, int right) {
33    if (left < right) {
34        int mid = (left + right) / 2;
35
36        mergeSort(arr, left, mid);
37        mergeSort(arr, mid + 1, right);
38
39        merge(arr, left, mid, right);
40    }
41
42}
43
44
45 int main() {
46    int arr[] = {64, 34, 25, 12, 22, 11, 90};
47    int size = sizeof(arr) / sizeof(arr[0]);
48
49    printf("Original Array: ");
50    displayArray(arr, size);
51    printf("\n");
52
53    mergeSort(arr, 0, size - 1);
54
55    printf("Sorted Array: ");
56    displayArray(arr, size);
57
58    return 0;
59}
```

The output window shows the execution results:

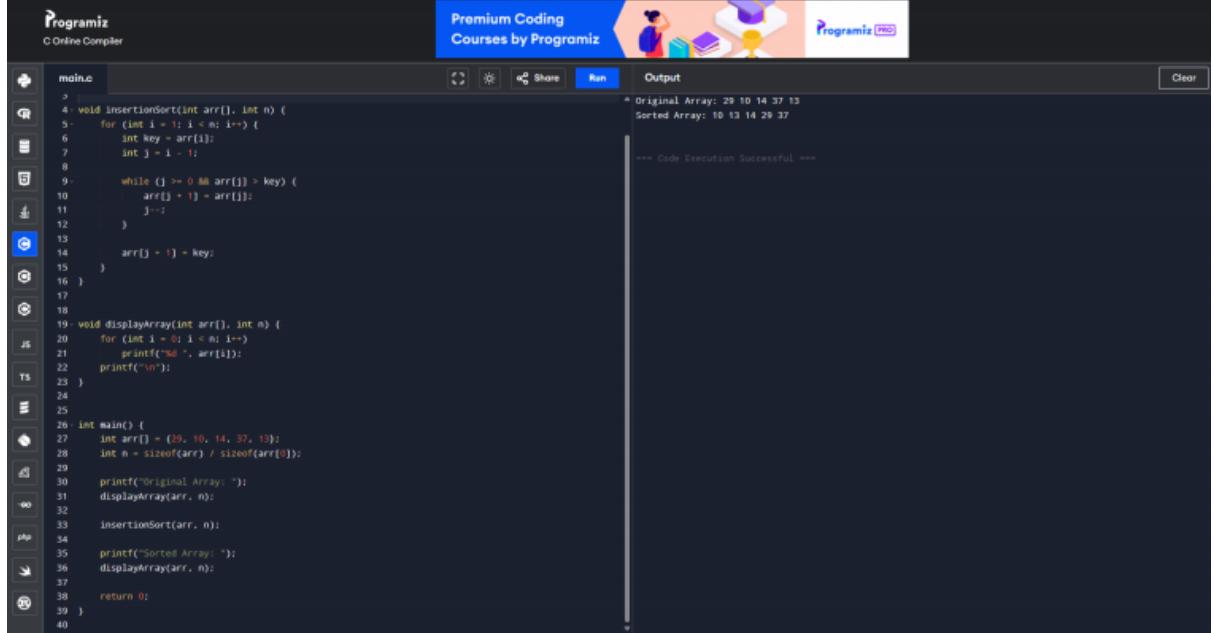
```
Original Array: 64 34 25 12 22 11 90
Sorted Array: 11 12 22 25 34 64 90
*** Code Execution Successful ***
```

26. Write a program in C to sort a list using Insertion Sort.



```
main.c
1 #include <stdio.h>
2
3
4 void insertionSort(int arr[], int n) {
5     for (int i = 1; i < n; i++) {
6         int key = arr[i];
7         int j = i - 1;
8
9         while (j >= 0 && arr[j] > key) {
10             arr[j + 1] = arr[j];
11             j--;
12         }
13         arr[j + 1] = key;
14     }
15 }
16
17 void displayArray(int arr[], int n) {
18     for (int i = 0; i < n; i++) {
19         printf("%d ", arr[i]);
20         printf("\n");
21     }
22 }
23
24
25 int main() {
26     int arr[] = {29, 10, 14, 37, 13};
27     int n = sizeof(arr) / sizeof(arr[0]);
28
29     printf("Original Array: ");
30     displayArray(arr, n);
31
32     insertionSort(arr, n);
33
34     printf("Sorted Array: ");
35     displayArray(arr, n);
36
37     return 0;
38 }
```

Original Array: 29 10 14 37 13
Sorted Array: 10 13 14 29 37
--- Code Execution Successful ---



```
main.c
1
2 void insertionSort(int arr[], int n) {
3     for (int i = 1; i < n; i++) {
4         int key = arr[i];
5         int j = i - 1;
6
7         while (j >= 0 && arr[j] > key) {
8             arr[j + 1] = arr[j];
9             j--;
10        }
11        arr[j + 1] = key;
12    }
13 }
14
15 void displayArray(int arr[], int n) {
16     for (int i = 0; i < n; i++) {
17         printf("%d ", arr[i]);
18         printf("\n");
19     }
20 }
21
22
23 int main() {
24     int arr[] = {29, 10, 14, 37, 13};
25     int n = sizeof(arr) / sizeof(arr[0]);
26
27     printf("Original Array: ");
28     displayArray(arr, n);
29
30     insertionSort(arr, n);
31
32     printf("Sorted Array: ");
33     displayArray(arr, n);
34
35     return 0;
36 }
```

Original Array: 29 10 14 37 13
Sorted Array: 10 13 14 29 37
--- Code Execution Successful ---

27. Write a program in C to sort a list using Heap Sort.

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is:

```
main.c
1 #include <stdio.h>
2
3
4 void heapify(int arr[], int n, int i) {
5     int largest = i;
6     int left = 2 * i + 1;
7     int right = 2 * i + 2;
8
9     if (left < n && arr[left] > arr[largest])
10        largest = left;
11
12
13     if (right < n && arr[right] > arr[largest])
14        largest = right;
15
16
17     if (largest != i) {
18         int temp = arr[i];
19         arr[i] = arr[largest];
20         arr[largest] = temp;
21
22
23         heapify(arr, n, largest);
24     }
25 }
26
27
28
29 void heapSort(int arr[], int n) {
30
31     for (int i = n / 2 - 1; i >= 0; i--)
32         heapify(arr, n, i);
33
34
35     for (int i = n - 1; i >= 0; i--) {
36
37         int temp = arr[0];
38         arr[0] = arr[i];
39         arr[i] = temp;
40
41
42         heapify(arr, i, 0);
43     }
44 }
45
46
47 void displayArray(int arr[], int n) {
48     for (int i = 0; i < n; i++)
49         printf("%d ", arr[i]);
50     printf("\n");
51 }
52
53 // Driver code
54 int main() {
55     int arr[] = {12, 11, 13, 5, 6, 7};
56     int n = sizeof(arr) / sizeof(arr[0]);
57
58     printf("Original array: ");
59     displayArray(arr, n);
60
61     heapSort(arr, n);
62
63     printf("Sorted array: ");
64     displayArray(arr, n);
65
66
67 }
68
```

The output window shows the execution results:

```
Original array: 12 11 13 5 6 7
Sorted array: 5 6 7 11 12 13
--- Code Execution Successful ---
```

The screenshot shows the Programiz C Online Compiler interface. The code in the editor is:

```
main.c
1 for (int i = n / 2 - 1; i >= 0; i--) {
2     heapify(arr, n, i);
3
4
5     for (int i = n - 1; i >= 0; i--) {
6
7         int temp = arr[0];
8         arr[0] = arr[i];
9         arr[i] = temp;
10
11
12         heapify(arr, i, 0);
13     }
14 }
15
16
17 void displayArray(int arr[], int n) {
18     for (int i = 0; i < n; i++)
19         printf("%d ", arr[i]);
20     printf("\n");
21 }
22
23 // Driver code
24 int main() {
25     int arr[] = {12, 11, 13, 5, 6, 7};
26     int n = sizeof(arr) / sizeof(arr[0]);
27
28     printf("Original array: ");
29     displayArray(arr, n);
30
31     heapSort(arr, n);
32
33     printf("Sorted array: ");
34     displayArray(arr, n);
35
36
37 }
38
```

The output window shows the execution results:

```
Original array: 12 11 13 5 6 7
Sorted array: 5 6 7 11 12 13
--- Code Execution Successful ---
```