

main.c

Share

Run

```
1 #include <stdio.h>
2 int main() {
3     int m1[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
4     int m2[3][3] = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};
5     int result[3][3] = {0};
6     for (int i = 0; i < 3; i++) {
7         for (int j = 0; j < 3; j++) {
8             for (int k = 0; k < 3; k++) {
9                 result[i][j] += m1[i][k] * m2[k][j];
10            }
11        }
12    }
13    printf("Result of Matrix Multiplication:\n");
14    for (int i = 0; i < 3; i++) {
15        for (int j = 0; j < 3; j++) {
16            printf("%d ", result[i][j]);
17        }
18        printf("\n");
19    }
20    return 0;
21 }
```

Output

Clear

```
/tmp/7314e0ReUg.o
Result of Matrix Multiplication:
30 24 18
84 69 54
138 114 90

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



Learn More

## LOOKING TO LEARN PROGRAMMING?

Start your programming journey with Programiz **AT NO COST.**



Programiz PRO >



main.c



Share

Run

Output

Clear

```
1 #include <stdio.h>
2 int main() {
3     int numbers[] = {2, 5, 8, 11, 14};
4     int size = sizeof(numbers) / sizeof(numbers[0]);
5     for (int i = 0; i < size; i++) {
6         if (numbers[i] % 2 == 0) {
7             printf("%d is even.\n", numbers[i]);
8         } else {
9             printf("%d is odd.\n", numbers[i]);
10        }
11    }
12    return 0;
13 }
14
```

/tmp/fUxSf325kz.o

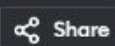
2 is even.  
5 is odd.  
8 is even.  
11 is odd.  
14 is even.

=== Code Execution Successful ===





main.c



Run

Output

Clear

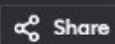
```
1 #include <stdio.h>
2 int fibonacci(int n) {
3     int a = 0, b = 1, c, i;
4     if (n == 0)
5         return a;
6     for (i = 2; i <= n; i++) {
7         c = a + b;
8         a = b;
9         b = c;
10    }
11    return b;
12 }
13 int main() {
14     int num;
15     printf("Enter the number to find Fibonacci: ");
16     scanf("%d", &num);
17     printf("Fibonacci number at position %d is: %d", num, fibonacci(num));
18     return 0;
19 }
20
```

```
/tmp/Xo1jMaoU8X.o
Enter the number to find Fibonacci: 7
Fibonacci number at position 7 is: 13

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



main.c



Run

Output

Clear

```
1 #include <stdio.h>
2 int factorial(int n) {
3     if (n == 0) {
4         return 1;
5     } else {
6         return n * factorial(n - 1);
7     }
8 }
9 int main() {
10     int number;
11     printf("Enter a non-negative integer: ");
12     scanf("%d", &number);
13     if (number < 0) {
14         printf("Factorial is not defined for negative numbers.\n");
15     } else {
16         int result = factorial(number);
17         printf("Factorial of %d = %d\n", number, result);
18     }
19     return 0;
20 }
21
```

```
/tmp/HiFr4U1Q0N.o
Enter a non-negative integer: 9
Factorial of 9 = 362880
```

=== Code Execution Successful ===

main.c



Share

Run

Output

Clear

```
1 #include <stdio.h>
2 int fibonacci(int n) {
3     if (n <= 1)
4         return n;
5     return fibonacci(n - 1) + fibonacci(n - 2);
6 }
7 int main() {
8     int n, i;
9     printf("Enter the number of terms: ");
10    scanf("%d", &n);
11    printf("Fibonacci Series: ");
12    for (i = 0; i < n; i++) {
13        printf("%d ", fibonacci(i));
14    }
15    return 0;
16 }
17
```

```
/tmp/HiFr4U1Q0N.o
Enter a non-negative integer: 9
Factorial of 9 = 362880

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



Learn More

## LOOKING TO LEARN PROGRAMMING?

Start your programming journey with Programiz **AT NO COST.**



Programiz PRO >



main.c



Share

Run

Output

Clear

```
1 #include <stdio.h>
2 int main() {
3     int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
4     int size = sizeof(arr) / sizeof(arr[0]);
5     printf("Alternate values in the array: ");
6     for (int i = 0; i < size; i += 2) {
7         printf("%d ", arr[i]);
8     }
9     return 0;
10 }
11
```

/tmp/FmMuVPGRuK.o

Alternate values in the array: 1 3 5 7 9

=== Code Execution Successful ===



Search



main.c



Share

Run

Output

Clear

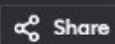
```
1 #include <stdio.h>
2 int main() {
3     int arr[] = {2, 4, 6, 8, 10};
4     int n = sizeof(arr) / sizeof(arr[0]);
5     int searchElement = 6;
6     int found = 0;
7     for (int i = 0; i < n; i++) {
8         if (arr[i] == searchElement) {
9             printf("Element found at index: %d\n", i);
10            found = 1;
11            break;
12        }
13    }
14    if (!found) {
15        printf("Element not found\n");
16    }
17    return 0;
18 }
19
```

```
/tmp/7alp05Q8uE.o
Element found at index: 2
```

=== Code Execution Successful ===



main.c



Run

Output

Clear

```
1 #include <stdio.h>
2 void rotateArray(int arr[], int n, int k) {
3     k = k % n;
4     for (int i = 0; i < n; i++) {
5         if (i < k) {
6             printf("%d ", arr[n + i - k]);
7         } else {
8             printf("%d ", arr[i - k]);
9         }
10    }
11 }
12 int main() {
13     int arr[] = {1, 2, 3, 4, 5};
14     int n = sizeof(arr) / sizeof(arr[0]);
15     int k = 2;
16     rotateArray(arr, n, k);
17     return 0;
18 }
19
```

/tmp/byqD2YdCCh.o

4 5 1 2 3

=== Code Execution Successful ===



main.c



Run

Output

Clear

```
1 #include <stdio.h>
2 int maxSubArraySum(int arr[], int n) {
3     int max_sum = arr[0];
4     int current_sum = arr[0];
5     for (int i = 1; i < n; i++) {
6         current_sum = (current_sum + arr[i] > arr[i]) ? current_sum +
            arr[i] : arr[i];
7         max_sum = (current_sum > max_sum) ? current_sum : max_sum;
8     }
9     return max_sum;
10 }
11 int main() {
12     int arr[] = { -2, 1, -3, 4, -1, 2, 1, -5, 4 };
13     int n = sizeof(arr) / sizeof(arr[0]);
14     int max_sum = maxSubArraySum(arr, n);
15     printf("The largest sum of contiguous subarray is: %d", max_sum);
16     return 0;
17 }
18
```

```
/tmp/E60J9Gpa4c.o
The largest sum of contiguous subarray is: 6

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

main.c

Share

Run

Output

Clear

```
1 #include <stdio.h>
2 int main() {
3     int arr1[] = {1, 2, 3, 4};
4     int arr2[] = {5, 6, 7, 8};
5     int size1 = sizeof(arr1) / sizeof(arr1[0]);
6     int size2 = sizeof(arr2) / sizeof(arr2[0]);
7     int size3 = size1 + size2;
8     int merged[size3];
9     for (int i = 0; i < size1; i++) {
10         merged[i] = arr1[i];
11     }
12     for (int i = 0; i < size2; i++) {
13         merged[size1 + i] = arr2[i];
14     }
15     for (int i = 0; i < size3; i++) {
16         printf("%d ", merged[i]);
17     }
18     return 0;
19 }
20
```

```
/tmp/GGVEkAcEjG.o
1 2 3 4 5 6 7 8

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



SCHOLARSHIP EXAM TO FULFIL  
YOUR CHILD'S IIT-JEE  
**DREAM**



Register for:  
**NSAT**  
2024  
ASPIRE \* ASSESS \* ACHIEVE

Register Now

Programiz PRO >



main.c



Share

Run

Output

Clear

```
1 #include <stdio.h>
2 void rearrangeArray(int arr[], int n) {
3     int max_idx = n - 1;
4     int min_idx = 0;
5     int max_elem = arr[max_idx] + 1;
6     for (int i = 0; i < n; i++) {
7         if (i % 2 == 0) {
8             arr[i] += (arr[max_idx] % max_elem) * max_elem;
9             max_idx--;
10        } else {
11            arr[i] += (arr[min_idx] % max_elem) * max_elem;
12            min_idx++;
13        }
14    }
15    for (int i = 0; i < n; i++) {
16        arr[i] = arr[i] / max_elem;
17    }
18 }
19 int main() {
20     int arr[] = {1, 2, 3, 4, 5, 6, 7};
21     int n = sizeof(arr) / sizeof(arr[0]);
22     rearrangeArray(arr, n);
23     printf("Rearranged Array: ");
24     for (int i = 0; i < n; i++) {
25         printf("%d ", arr[i]);
26     }
27     return 0;
28 }
29
```

/tmp/CbpIfSxaxN.o  
Rearranged Array: 7 1 6 2 5 3 4

=== Code Execution Successful ===





SCHOLARSHIP EXAM TO FULFIL  
YOUR CHILD'S IIT-JEE  
**DREAM**



Register for:  
**NSAT**  
2024  
ASPIRE • ASSESS • ACHIEVE

Register Now

Programiz PRO >



main.c



Share

Run

Output

Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct Student {
5     int rollno;
6     char name[50];
7 };
8
9 int main() {
10     struct Student *ptr = (struct Student*)malloc(sizeof(struct Student));
11
12     printf("Enter Roll Number: ");
13     scanf("%d", &ptr->rollno);
14
15     printf("Enter Name: ");
16     scanf("%s", ptr->name);
17
18     printf("\nStudent Details\n");
19     printf("Roll Number: %d\n", ptr->rollno);
20     printf("Name: %s\n", ptr->name);
21
22     free(ptr);
23     return 0;
24 }
25
```

```
/tmp/IsjvYof1ZW.o
Enter Roll Number: 1912
Enter Name: govardhan

Student Details
Roll Number: 1912
Name: govardhan

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

Programiz

C Online Compiler



LOOKING TO LEARN PROGRAMMING?

Start your programming journey with Programiz AT NO COST.



Programiz PRO >



main.c



Output



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 typedef struct Node {int data; struct Node* next;} Node;
4 Node* createNode(int data) {
5     Node* newNode = (Node*)malloc(sizeof(Node));
6     newNode->data = data;
7     newNode->next = NULL;
8     return newNode;
9 }
10 void insertAtBeginning(Node** head, int data) {
11     Node* newNode = createNode(data);
12     newNode->next = *head;
13     *head = newNode;
14 }
15 void printList(Node* head) {
16     Node* temp = head;
17     while (temp != NULL) {
18         printf("%d->", temp->data);
19         temp = temp->next;
20     }
21     printf("NULL\n");
22 }
23 int main() {
24     Node* head = createNode(22);
25     head->next = createNode(77);
26     head->next->next = createNode(55);
27     head->next->next->next = createNode(88);
28     printf("List before insertion: ");
29     printList(head);
30     insertAtBeginning(&head, 100);
31     printf("List after insertion: ");
32     printList(head);
33     return 0;
34 }
```

```
/tmp/a1QNK17zfu.o
List before insertion: 22->77->55->88->NULL
List after insertion: 100->22->77->55->88->NULL
```

=== Code Execution Successful ===

**LOOKING TO LEARN PROGRAMMING?**  
Start your programming journey with Programiz **AT NO COST.**

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 typedef struct Node {
4     int data;
5     struct Node* next;
6 } Node;
7 Node* createNode(int data) {
8     Node* newNode = (Node*)malloc(sizeof(Node));
9     newNode->data = data;
10    newNode->next = NULL;
11    return newNode;
12 }
13 void insertAtEnd(Node** head, int data) {
14     Node* newNode = createNode(data);
15     if (*head == NULL) {
16         *head = newNode;
17     } else {
18         Node* temp = *head;
19         while (temp->next != NULL) {
20             temp = temp->next;
21         }
22         temp->next = newNode;
23     }
24 }
25 void printList(Node* head) {
26     while (head) {
27         printf("%d->", head->data);
28         head = head->next;
29     }
30     printf("NULL\n");
31 }
32 int main() {
33     Node* head = createNode(22);
34     head->next = createNode(77);
35     head->next->next = createNode(55);
```

Output

/tmp/Ar7Nh3gqf1.o  
List before insertion: 22->77->55->88->NULL  
List after insertion: 22->77->55->88->110->NULL

=== Code Execution Successful ===

**LOOKING TO LEARN PROGRAMMING?**  
Start your programming journey with Programiz **AT NO COST.**

Programiz PRO >

```
main.c
17 } else {
18     Node* temp = *head;
19     while (temp->next != NULL) {
20         temp = temp->next;
21     }
22     temp->next = newNode;
23 }
24 }
25 void printList(Node* head) {
26     while (head) {
27         printf("%d->", head->data);
28         head = head->next;
29     }
30     printf("NULL\n");
31 }
32 int main() {
33     Node* head = createNode(22);
34     head->next = createNode(77);
35     head->next->next = createNode(55);
36     head->next->next->next = createNode(88);
37     printf("List before insertion: ");
38     printList(head);
39     insertAtEnd(&head, 110);
40     printf("List after insertion: ");
41     printList(head);
42     Node* temp;
43     while (head) {
44         temp = head;
45         head = head->next;
46         free(temp);
47     }
48     return 0;
49 }
50
```

Output

/tmp/Ar7Nh3gqfi.o  
List before insertion: 22->77->55->88->NULL  
List after insertion: 22->77->55->88->110->NULL

=== Code Execution Successful ===

Clear





SCHOLARSHIP EXAM TO FULFIL  
YOUR CHILD'S IIT-JEE  
**DREAM**



Register for:  
**NSAT**  
2024  
ASPIRE • ASSESS • ACHIEVE

Register Now

Programiz PRO >



main.c



Share

Run

Output

Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct Node {
4     int data;
5     struct Node* next;
6 };
7 void insertNode(struct Node** head_ref, int new_data, int position) {
8     struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
9     new_node->data = new_data;
10    if (position == 1) {
11        new_node->next = *head_ref;
12        *head_ref = new_node;
13    } else {
14        struct Node* temp = *head_ref;
15        for (int i = 1; i < position - 1 && temp != NULL; i++) {
16            temp = temp->next;
17        }
18        if (temp == NULL) {
19            printf("Position out of range\n");
20            return;
21        }
22        new_node->next = temp->next;
23        temp->next = new_node;
24    }
25 }
26 void printList(struct Node* node) {
27     while (node != NULL) {
28         printf("%d->", node->data);
29         node = node->next;
30     }
```

```
/tmp/kCAyrUHR1.o
Input: 5->77->88->99->22->
Output: 5->77->88->101->99->22->
```

=== Code Execution Successful ===



SCHOLARSHIP EXAM TO FULFIL  
YOUR CHILD'S IIT-JEE  
**DREAM**



Register for:  
**NSAT**  
2024  
ASPIRE • ASSESS • ACHIEVE

Register Now

Programiz PRO >



```
main.c
18 // temp == NULL
19     printf("Position out of range\n");
20     return;
21 }
22 new_node->next = temp->next;
23 temp->next = new_node;
24 }
25 }
26 void printList(struct Node* node) {
27     while (node != NULL) {
28         printf("%d->", node->data);
29         node = node->next;
30     }
31     printf("\n");
32 }
33 int main() {
34     struct Node* head = NULL;
35     insertNode(&head, 55, 1);
36     insertNode(&head, 77, 2);
37     insertNode(&head, 88, 3);
38     insertNode(&head, 99, 4);
39     insertNode(&head, 22, 5);
40     printf("Input: ");
41     printList(head);
42     insertNode(&head, 101, 4);
43     printf("Output: ");
44     printList(head);
45     return 0;
46 }
47
```



Share

Run

Output

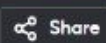
Clear

```
/tmp/kCAyrUHR1.o
Input: 5->77->88->99->22->
Output: 5->77->88->101->99->22->
```

=== Code Execution Successful ===



main.c



Output

Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct Node {
4     int data;
5     struct Node* next;
6 };
7 void deleteFirstNode(struct Node** head) {
8     if (*head == NULL) {
9         printf("List is empty. No node to delete.\n");
10        return;
11    }
12    struct Node* temp = *head;
13    *head = temp->next;
14    free(temp);
15 }
16 void printList(struct Node* head) {
17     struct Node* temp = head;
18     while (temp != NULL) {
19         printf("%d -> ", temp->data);
20         temp = temp->next;
21     }
22     printf("NULL\n");
23 }
24 int main() {
25     struct Node* head = NULL;
26     struct Node* second = NULL;
27     struct Node* third = NULL;
28     head = (struct Node*)malloc(sizeof(struct Node));
29     second = (struct Node*)malloc(sizeof(struct Node));
30     third = (struct Node*)malloc(sizeof(struct Node));
```

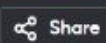
```
/tmp/XTK1Xw4QKF.o
Original List: 22 -> 55 -> 77 -> NULL
List after deleting the first node: 55 -> 77 -> NULL

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





main.c



Output

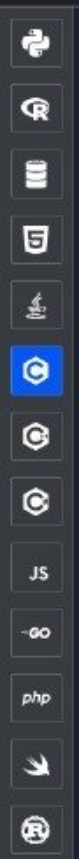
Clear

```
16 void printlist(struct Node* head) {
17     struct Node* temp = head;
18     while (temp != NULL) {
19         printf("%d -> ", temp->data);
20         temp = temp->next;
21     }
22     printf("NULL\n");
23 }
24 int main() {
25     struct Node* head = NULL;
26     struct Node* second = NULL;
27     struct Node* third = NULL;
28     head = (struct Node*)malloc(sizeof(struct Node));
29     second = (struct Node*)malloc(sizeof(struct Node));
30     third = (struct Node*)malloc(sizeof(struct Node));
31     head->data = 22;
32     head->next = second;
33     second->data = 55;
34     second->next = third;
35     third->data = 77;
36     third->next = NULL;
37     printf("Original List: ");
38     printlist(head);
39     deleteFirstNode(&head);
40     printf("List after deleting the first node: ");
41     printlist(head);
42     return 0;
43 }
44
```

```
/tmp/XTK1Xw4QKF.o
Original List: 22 -> 55 -> 77 -> NULL
List after deleting the first node: 55 -> 77 -> NULL

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





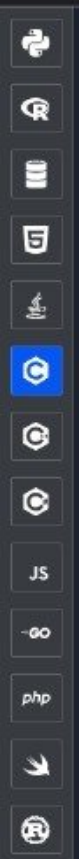
main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct Node {
4     int data;
5     struct Node* next;
6 };
7 void deleteEndNode(struct Node** head) {
8     if (*head == NULL || (*head)->next == NULL) {
9         return;
10    }
11    struct Node* secondLast = *head;
12    while (secondLast->next->next != NULL) {
13        secondLast = secondLast->next;
14    }
15    free(secondLast->next);
16    secondLast->next = NULL;
17 }
18 void printlist(struct Node* node) {
19     while (node != NULL) {
20         printf("%d -> ", node->data);
21         node = node->next;
22     }
23     printf("\n");
24 }
25 int main() {
26     struct Node* head = NULL;
27     struct Node* second = NULL;
28     struct Node* third = NULL;
29     struct Node* fourth = NULL;
30     head = (struct Node*)malloc(sizeof(struct Node));
```



Run

Output

/tmp/sbEZg8mcE7.o  
Original Linked List: 11 -> 33 -> 44 -> 22 ->  
Linked List after deleting the end node: 11 -> 33 -> 44 ->  
  
=== Code Execution Successful ===



```
main.c
20 printf("%d -> ", node->data);
21 node = node->next;
22 }
23 printf("\n");
24 }
25 int main() {
26     struct Node* head = NULL;
27     struct Node* second = NULL;
28     struct Node* third = NULL;
29     struct Node* fourth = NULL;
30     head = (struct Node*)malloc(sizeof(struct Node));
31     second = (struct Node*)malloc(sizeof(struct Node));
32     third = (struct Node*)malloc(sizeof(struct Node));
33     fourth = (struct Node*)malloc(sizeof(struct Node));
34     head->data = 11;
35     head->next = second;
36     second->data = 33;
37     second->next = third;
38     third->data = 44;
39     third->next = fourth;
40     fourth->data = 22;
41     fourth->next = NULL;
42     printf("Original Linked List: ");
43     printList(head);
44     deleteEndNode(&head);
45     printf("Linked List after deleting the end node: ");
46     printList(head);
47     return 0;
48 }
49
```

   Share [Run](#)

Output [Clear](#)

```
/tmp/sbEZg8mcE7.o
Original Linked List: 11 -> 33 -> 44 -> 22 ->
Linked List after deleting the end node: 11 -> 33 -> 44 ->

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

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct Node {
4     int data;
5     struct Node* next;
6 };
7 void deleteNode(struct Node** head_ref, int position) {
8     if (*head_ref == NULL)
9         return;
10    struct Node* temp = *head_ref;
11    if (position == 0) {
12        *head_ref = temp->next;
13        free(temp);
14        return;
15    }
16    for (int i = 0; temp != NULL && i < position - 1; i++)
17        temp = temp->next;
18    if (temp == NULL || temp->next == NULL)
19        return;
20    struct Node* next = temp->next->next;
21    free(temp->next);
22    temp->next = next;
23 }
24 void printList(struct Node* node) {
25     while (node != NULL) {
26         printf("%d -> ", node->data);
27         node = node->next;
28     }
29     printf("NULL\n");
30 }
```

Output

Clear

/tmp/f5xw1wLJok.o  
Original Linked List: 22 -> 77 -> 99 -> 22 -> NULL  
Linked List after deleting node at position 4: 22 -> 77 -> 99 -> NULL

=== Code Execution Successful ===



- Python
- Java
- C
- C++
- C#
- JavaScript
- Go
- PHP
- Swift
- Ruby

```
main.c
26     printf("%d -> ", node->data);
27     node = node->next;
28 }
29 printf("NULL\n");
30 }
31 int main() {
32     struct Node* head = NULL;
33     struct Node* second = NULL;
34     struct Node* third = NULL;
35     struct Node* fourth = NULL;
36     head = (struct Node*)malloc(sizeof(struct Node));
37     second = (struct Node*)malloc(sizeof(struct Node));
38     third = (struct Node*)malloc(sizeof(struct Node));
39     fourth = (struct Node*)malloc(sizeof(struct Node));
40     head->data = 22;
41     head->next = second;
42     second->data = 77;
43     second->next = third;
44     third->data = 99;
45     third->next = fourth;
46     fourth->data = 22;
47     fourth->next = NULL;
48     printf("Original Linked List: ");
49     printList(head);
50     deleteNode(&head, 3);
51     printf("Linked List after deleting node at position 4: ");
52     printList(head);
53     return 0;
54 }
55
```

Share Run

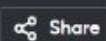
Output Clear

/tmp/f5xw1wLJok.o  
Original Linked List: 22 -> 77 -> 99 -> 22 -> NULL  
Linked List after deleting node at position 4: 22 -> 77 -> 99 -> NULL  
  
=== Code Execution Successful ===





main.c



Output

Clear

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct Node {
4     int data;
5     struct Node* next;
6 };
7 int findDifference(struct Node* head) {
8     if (head == NULL) {
9         return 0;
10    }
11    int min = head->data;
12    int max = head->data;
13    struct Node* current = head;
14    while (current != NULL) {
15        if (current->data < min) {
16            min = current->data;
17        }
18        if (current->data > max) {
19            max = current->data;
20        }
21        current = current->next;
22    }
23    return max - min;
24 }
25 int main() {
26     struct Node* head = NULL;
27     struct Node* second = NULL;
28     struct Node* third = NULL;
29     struct Node* fourth = NULL;
30     struct Node* fifth = NULL;
```

/tmp/W6wQrIvHs8.o  
Difference = 77

=== Code Execution Successful ===

main.c

21

current = current->next;

22

}

23

return max - min;

24

}

25

int main() {

26

struct Node\* head = NULL;

27

struct Node\* second = NULL;

28

struct Node\* third = NULL;

29

struct Node\* fourth = NULL;

30

struct Node\* fifth = NULL;

31

head = (struct Node\*)malloc(sizeof(struct Node));

32

second = (struct Node\*)malloc(sizeof(struct Node));

33

third = (struct Node\*)malloc(sizeof(struct Node));

34

fourth = (struct Node\*)malloc(sizeof(struct Node));

35

fifth = (struct Node\*)malloc(sizeof(struct Node));

36

head->data = 56;

37

head->next = second;

38

second->data = 99;

39

second->next = third;

40

third->data = 66;

41

third->next = fourth;

42

fourth->data = 22;

43

fourth->next = fifth;

44

fifth->data = 33;

45

fifth->next = NULL;

46

printf("Difference = %d\n", findDifference(head));

47

return 0;

48

}

49

Run

Share

Output

Clear

/tmp/W6wQrIvHs8.o

Difference = 77

=== Code Execution Successful ===

main.c

```
1 #include <stdio.h>
2 #define SIZE 5
3 int queue[SIZE];
4 int front = -1, rear = -1;
5 void enqueue(int value) {
6     if ((front == 0 && rear == SIZE - 1) || (rear == (front - 1) % (SIZE - 1))) {
7         printf("Queue is full\n");
8     } else if (front == -1) {
9         front = rear = 0;
10        queue[rear] = value;
11    } else if (rear == SIZE - 1 && front != 0) {
12        rear = 0;
13        queue[rear] = value;
14    } else {
15        rear++;
16        queue[rear] = value;
17    }
18 }
19 int dequeue() {
20     if (front == -1) {
21         printf("Queue is empty\n");
22         return -1;
23     }
24     int data = queue[front];
25     queue[front] = -1;
26     if (front == rear) {
27         front = rear = -1;
28     } else if (front == SIZE - 1) {
29         front = 0;
30     } else {
31         front++;
32     }
33     return data;
34 }
35 void display() {
```

Output

```
/tmp/46Acq7j0Sb.o
Elements in Circular Queue are: 1 2 3 4 5
Elements in Circular Queue are: 3 4 5
Elements in Circular Queue are: 3 4 5 6 7
```

=== Code Execution Successful ===

Clear

```
main.c
35 void display() {
36     if (front == -1) {
37         printf("Queue is empty\n");
38         return;
39     }
40     printf("Elements in Circular Queue are: ");
41     if (rear >= front) {
42         for (int i = front; i <= rear; i++) {
43             printf("%d ", queue[i]);
44         }
45     } else {
46         for (int i = front; i < SIZE; i++) {
47             printf("%d ", queue[i]);
48         }
49         for (int i = 0; i <= rear; i++) {
50             printf("%d ", queue[i]);
51         }
52     }
53     printf("\n");
54 }
55 int main() {
56     enqueue(1);
57     enqueue(2);
58     enqueue(3);
59     enqueue(4);
60     enqueue(5);
61     display();
62     dequeue();
63     dequeue();
64     display();
65     enqueue(6);
66     enqueue(7);
67     display();
68     return 0;
69 }
```

Output

Elements in Circular Queue are: 1 2 3 4 5  
Elements in Circular Queue are: 3 4 5  
Elements in Circular Queue are: 3 4 5 6 7

=== Code Execution Successful ===



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

Output

/tmp/s6s2pPzGf3.o

After enqueueing 10, 20, 30:  
Queue: 30 20 10  
Queue underflow

=== Code Exited With Errors ===

Clear

**LOOKING TO LEARN PROGRAMMING?**  
Start your programming journey with Programiz **AT NO COST.**

Programiz PRO >

```
main.c
34- int isEmpty(Stack* stack) {
35-     return stack->top == NULL;
36- }
37- void display(Stack* stack) {
38-     Node* temp = stack->top;
39-     while (temp != NULL) {
40-         printf("%d ", temp->data);
41-         temp = temp->next;
42-     }
43-     printf("\n");
44- }
45- typedef struct {
46-     Stack s1;
47-     Stack s2;
48- } Queue;
49- void initQueue(Queue* queue) {
50-     initStack(&queue->s1);
51-     initStack(&queue->s2);
52- }
53- void enqueue(Queue* queue, int value) {
54-     push(&queue->s1, value);
55- }
56- int dequeue(Queue* queue) {
57-     if (isEmpty(&queue->s1) && isEmpty(&queue->s2)) {
58-         printf("Queue underflow\n");
59-         exit(1);
60-     }
61-     if (isEmpty(&queue->s2)) {
62-         while (!isEmpty(&queue->s1)) {
63-             push(&queue->s2, pop(&queue->s1));
64-         }
65-     }
66-     return pop(&queue->s2);
67- }
```

Output

After enqueueing 10, 20, 30:  
Queue: 30 20 10  
Queue underflow

=== Code Exited With Errors ===

**LOOKING TO LEARN PROGRAMMING?**  
Start your programming journey with Programiz **AT NO COST.**

Programiz PRO >

```
main.c
71 initStack(&temp);
72 while (!isEmpty(&queue->s2)) {
73     int value = pop(&queue->s2);
74     printf("%d ", value);
75     push(&temp, value);
76 }
77 while (!isEmpty(&temp)) {
78     push(&queue->s2, pop(&temp));
79 }
80 while (!isEmpty(&queue->s1)) {
81     printf("%d ", pop(&queue->s1));
82 }
83 while (!isEmpty(&temp)) {
84     push(&queue->s1, pop(&temp));
85 }
86 printf("\n");
87 }
88 int main() {
89     Queue queue;
90     initQueue(&queue);
91     enqueue(&queue, 10);
92     enqueue(&queue, 20);
93     enqueue(&queue, 30);
94     printf("After enqueueing 10, 20, 30:\n");
95     displayQueue(&queue);
96     printf("Dequeued: %d\n", dequeue(&queue));
97     printf("After dequeue:\n");
98     displayQueue(&queue);
99     enqueue(&queue, 40);
100    enqueue(&queue, 50);
101    printf("After enqueueing 40, 50:\n");
102    displayQueue(&queue);
103    return 0;
104 }
105
```

Output

After enqueueing 10, 20, 30:  
Queue: 30 20 10  
Queue underflow

=== Code Exited With Errors ===

**LOOKING TO LEARN PROGRAMMING?**  
Start your programming journey with Programiz **AT NO COST.**

Programiz PRO >

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdbool.h>
4 #define MAX 100
5 typedef struct {
6     char arr[MAX];
7     int top;
8 } Stack;
9 void initStack(Stack* stack) {
10     stack->top = -1;
11 }
12 bool isEmpty(Stack* stack) {
13     return stack->top == -1;
14 }
15 void push(Stack* stack, char c) {
16     stack->arr[++(stack->top)] = c;
17 }
18 char pop(Stack* stack) {
19     return stack->arr[(stack->top)--];
20 }
21 bool areParenthesesBalanced(const char* str) {
22     Stack stack;
23     initStack(&stack);
24     for (int i = 0; str[i] != '\0'; i++) {
25         char ch = str[i];
26         if (ch == '(' || ch == '[' || ch == '{') {
27             push(&stack, ch);
28         } else if (ch == ')' || ch == ']' || ch == '}') {
29             if (isEmpty(&stack)) return false;
30             char top = pop(&stack);
31             if ((ch == ')' && top != '(') ||
32                 (ch == ']' && top != '[') ||
33                 (ch == '}' && top != '{')) {
34                 return false;
35             }
36         }
37     }
38     return isEmpty(&stack);
39 }
```

Output

/tmp/0eyMoJyaph.o  
Not Balanced

=== Code Execution Successful ===

Clear



**LOOKING TO LEARN PROGRAMMING?**  
Start your programming journey with Programiz **AT NO COST.**

Programiz PRO >

```
main.c
20 }
21 bool areParenthesesBalanced(const char* str) {
22     Stack stack;
23     initStack(&stack);
24     for (int i = 0; str[i] != '\0'; i++) {
25         char ch = str[i];
26         if (ch == '(' || ch == '[' || ch == '{') {
27             push(&stack, ch);
28         } else if (ch == ')' || ch == ']' || ch == '}') {
29             if (isEmpty(&stack)) return false;
30             char top = pop(&stack);
31             if ((ch == ')' && top != '(') ||
32                 (ch == ']' && top != '[') ||
33                 (ch == '}' && top != '{')) {
34                 return false;
35             }
36         }
37     }
38
39     return isEmpty(&stack);
40 }
41
42 int main() {
43     const char* expr = "(A+B)*(C-D))/E^2";
44
45     if (areParenthesesBalanced(expr)) {
46         printf("Balanced\n");
47     } else {
48         printf("Not Balanced\n");
49     }
50
51     return 0;
52 }
53
```

Output

/tmp/0eyMoJyaph.o  
Not Balanced

=== Code Execution Successful ===

Clear

Programiz

C Online Compiler



LOOKING TO LEARN PROGRAMMING?

Start your programming journey with Programiz AT NO COST.



Programiz PRO >



main.c



Output



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <ctype.h>
5 typedef struct {
6     char* arr;
7     int top;
8     int capacity;
9 } Stack;
10 Stack* createStack(int capacity) {
11     Stack* stack = (Stack*)malloc(sizeof(Stack));
12     stack->capacity = capacity;
13     stack->top = -1;
14     stack->arr = (char*)malloc(capacity * sizeof(char));
15     return stack;
16 }
17 int isEmpty(Stack* stack) {
18     return stack->top == -1;
19 }
20 void push(Stack* stack, char c) {
21     stack->arr[++(stack->top)] = c;
22 }
23 char pop(Stack* stack) {
24     return stack->arr[(stack->top)--];
25 }
26 char peek(Stack* stack) {
27     return stack->arr[stack->top];
28 }
29 int precedence(char op) {
30     if (op == '^') return 3;
31     if (op == '*' || op == '/') return 2;
32     if (op == '+' || op == '-') return 1;
33     return 0;
34 }
35 void infixToPostfix(const char* infix, char* postfix) {
```

```
/tmp/gLHKdgrVvR.o
Infix to Postfix: AB+CD-*E2^/
Infix to Prefix: *+AB/-CD^E2
```

=== Code Execution Successful ===

**LOOKING TO LEARN PROGRAMMING?**  
Start your programming journey with Programiz **AT NO COST.**

Programiz PRO >

```
main.c
34 }
35 void infixToPostfix(const char* infix, char* postfix) {
36     Stack* stack = createStack(strlen(infix));
37     int k = 0;
38     for (int i = 0; infix[i] != '\0'; i++) {
39         char c = infix[i];
40         if (isdigit(c)) {
41             postfix[k++] = c;
42         } else if (c == '(') {
43             push(stack, c);
44         } else if (c == ')') {
45             while (!isEmpty(stack) && peek(stack) != '(') {
46                 postfix[k++] = pop(stack);
47             }
48             pop(stack);
49         } else {
50             while (!isEmpty(stack) && precedence(peek(stack)) >= precedence(c)) {
51                 postfix[k++] = pop(stack);
52             }
53             push(stack, c);
54         }
55     }
56     while (!isEmpty(stack)) {
57         postfix[k++] = pop(stack);
58     }
59     postfix[k] = '\0';
60     free(stack->arr);
61     free(stack);
62 }
63 void reverseString(char* str) {
64     int len = strlen(str);
65     for (int i = 0; i < len / 2; i++) {
66         char temp = str[i];
67         str[i] = str[len - 1 - i];
68         str[len - 1 - i] = temp;
69     }
70 }
```

Output

/tmp/gLHKdgrVvR.o  
Infix to Postfix: AB+CD-\*E2^/  
Infix to Prefix: \*-AB/-CD^E2

=== Code Execution Successful ===

Clear

**LOOKING TO LEARN PROGRAMMING?**  
Start your programming journey with Programiz **AT NO COST.**

Programiz PRO >

```
main.c
62 }
63 void reverseString(char* str) {
64     int len = strlen(str);
65     for (int i = 0; i < len / 2; i++) {
66         char temp = str[i];
67         str[i] = str[len - 1 - i];
68         str[len - 1 - i] = temp;
69     }
70 }
71 void infixToPrefix(const char* infix, char* prefix) {
72     char* reversedInfix = strdup(infix);
73     reverseString(reversedInfix);
74     for (int i = 0; reversedInfix[i]; i++) {
75         if (reversedInfix[i] == '(') reversedInfix[i] = ')';
76         else if (reversedInfix[i] == ')') reversedInfix[i] = '(';
77     }
78     char* reversedPrefix = (char*)malloc(strlen(infix) + 1);
79     infixToPostfix(reversedInfix, reversedPrefix);
80     reverseString(reversedPrefix);
81     strcpy(prefix, reversedPrefix);
82     free(reversedInfix);
83     free(reversedPrefix);
84 }
85 int main() {
86     const char* infix = "(A+B)*(C-D)/E^2";
87     char postfix[100];
88     char prefix[100];
89     infixToPostfix(infix, postfix);
90     printf("Infix to Postfix: %s\n", postfix);
91     infixToPrefix(infix, prefix);
92     printf("Infix to Prefix: %s\n", prefix);
93     return 0;
94 }
95
```

Output

/tmp/gLHKdgrVyR.o  
Infix to Postfix: AB+CD-\*E2^/  
Infix to Prefix: \*+AB/-CD^E2

=== Code Execution Successful ===

Clear



**LOOKING TO LEARN PROGRAMMING?**  
Start your programming journey with Programiz **AT NO COST.**

Programiz PRO >

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #define MAX 100
4 typedef struct {
5     int arr[MAX];
6     int top;
7 } Stack;
8 void initStack(Stack* stack);
9 int isEmpty(Stack* stack);
10 int isFull(Stack* stack);
11 void push(Stack* stack, int value);
12 int pop(Stack* stack);
13 int peek(Stack* stack);
14 void display(Stack* stack);
15 int main() {
16     Stack stack;
17     initStack(&stack);
18     push(&stack, 10);
19     push(&stack, 20);
20     push(&stack, 30);
21     printf("Stack after PUSH operations:\n");
22     display(&stack);
23     printf("Top element (PEEK): %d\n", peek(&stack));
24     printf("Popped element: %d\n", pop(&stack));
25     printf("Stack after POP operation:\n");
26     display(&stack);
27     return 0;
28 }
29 void initStack(Stack* stack) {
30     stack->top = -1;
31 }
32 int isEmpty(Stack* stack) {
33     return stack->top == -1;
34 }
35 int isFull(Stack* stack) {
```

Output

```
/tmp/2A0SAuYwp8.o
Stack after PUSH operations:
Stack elements are:
30
20
10
Top element (PEEK): 30
Popped element: 30
Stack after POP operation:
Stack elements are:
20
10

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

main.c

40

printf("Stack overflow\n");

41

return;

42

}

43

stack->arr[++(stack->top)] = value;

44

}

45

int pop(Stack\* stack) {

46

if (isEmpty(stack)) {

47

printf("Stack underflow\n");

48

exit(1);

49

}

50

return stack->arr[(stack->top)--];

51

}

52

53

// Function to peek the top element of the stack

54

int peek(Stack\* stack) {

55

if (isEmpty(stack)) {

56

printf("Stack is empty\n");

57

exit(1); // Exit the program if stack is empty

58

}

59

return stack->arr[stack->top];

60

}

61

62

// Function to display all elements in the stack

63

void display(Stack\* stack) {

64

if (isEmpty(stack)) {

65

printf("Stack is empty\n");

66

return;

67

}

68

printf("Stack elements are:\n");

69

for (int i = stack->top; i >= 0; i--) {

70

printf("%d\n", stack->arr[i]);

71

}

72

}

73

Share

Run

Output

Clear

/tmp/2A0SAuYwp8.o

Stack after PUSH operations:

Stack elements are:

30

20

10

Top element (PEEK): 30

Popped element: 30

Stack after POP operation:

Stack elements are:

20

10

=== Code Execution Successful ===



main.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 typedef struct Node {
4     int data;
5     struct Node* next;
6 } Node;
7 Node* createNode(int data) {
8     Node* newNode = (Node*)malloc(sizeof(Node));
9     newNode->data = data;
10    newNode->next = NULL;
11    return newNode;
12 }
13 int search(Node* head, int key) {
14     Node* current = head;
15     while (current != NULL) {
16         if (current->data == key) {
17             return 1;
18         }
19         current = current->next;
20     }
21     return 0;
22 }
23 void display(Node* head) {
24     Node* current = head;
25     while (current != NULL) {
26         printf("%d->", current->data);
27         current = current->next;
28     }
29     printf("NULL\n");
30 }
31 int main() {
32     Node* head = createNode(55);
33     head->next = createNode(99);
34     head->next->next = createNode(66);
35     head->next->next->next = createNode(22);
```



Run

Output

Clear

```
/tmp/K8PkmJ66U5.o
Linked List: 55->99->66->22->NULL
Key 222 NOT FOUND
```

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



```
main.c
20 }
21 return 0;
22 }
23 void display(Node* head) {
24     Node* current = head;
25     while (current != NULL) {
26         printf("%d->", current->data);
27         current = current->next;
28     }
29     printf("NULL\n");
30 }
31 int main() {
32     Node* head = createNode(55);
33     head->next = createNode(99);
34     head->next->next = createNode(66);
35     head->next->next->next = createNode(22);
36     printf("Linked List: ");
37     display(head);
38     int key = 222;
39     if (search(head, key)) {
40         printf("Key %d found in the list.\n", key);
41     } else {
42         printf("Key %d NOT FOUND\n", key);
43     }
44     Node* current = head;
45     Node* nextNode;
46     while (current != NULL) {
47         nextNode = current->next;
48         free(current);
49         current = nextNode;
50     }
51     return 0;
52 }
53 }
```



Output

```
/tmp/K8PkmJ66U5.o
Linked List: 55->99->66->22->NULL
Key 222 NOT FOUND
```

=== Code Execution Successful ===