main.c

Share Run

```c
#include <stdio.h>
#define SIZE 5
void enQueue(int);
void deQueue();
void display();
int items[SIZE], front = -1, rear = -1;
int main()
{
  deQueue();
  enQueue(1);
  enQueue(2);
  enQueue(3);
  enQueue(4);
  enQueue(5);
  enQueue(6);
  display();
  deQueue();
  display();
  return 0;
}
void enQueue(int value)
{
  if (rear == SIZE - 1)
    printf("\nQueue is Full!!");
  else
  {
```

Output

Clear

```
/tmp/ImyZhC1vNI.o

Queue is Empty!!
Inserted -> 1
Inserted -> 2
Inserted -> 3
Inserted -> 4
Inserted -> 5
Queue is Full!!
Queue elements are:
1  2  3  4  5

Deleted : 1
Queue elements are:
2  3  4  5


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

main.c

Share Run

```c
#include <stdio.h>
void insertElement(int arr[], int n, int pos, int value) {
    for (int i = n; i > pos; i--) {
        arr[i] = arr[i - 1];
    }
    arr[pos] = value;
}
void deleteElement(int arr[], int n, int pos) {
    for (int i = pos; i < n - 1; i++) {
        arr[i] = arr[i + 1];
    }
}
int main() {
    int arr[100] = {1, 2, 3, 4, 5};
    int n = 5;
    insertElement(arr, n, 2, 99);
    n++;
    printf("Array after insertion: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    deleteElement(arr, n, 3);
    n--;
    printf("Array after deletion: ");
    for (int i = 0; i < n; i++) {
```

Output Clear

```
/tmp/3eNlbcXrdq.o
Array after insertion: 1 2 99 3 4 5
Array after deletion: 1 2 99 4 5

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

main.c

Share Run

Output Clear

```c
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
struct Stack
{
    int top;
    unsigned capacity;
    int* array;
};
struct Stack* createStack(unsigned capacity)
{
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack
        ));
    stack->capacity = capacity;
    stack->top = -1;
    stack->array = (int*)malloc(stack->capacity * sizeof(int));
    return stack;
}
int isFull(struct Stack* stack)
{
    return stack->top == stack->capacity - 1;
}
int isEmpty(struct Stack* stack)
{
    return stack->top == -1;
}
```

```
/tmp/l1YJBRjOeY.o
10 pushed to stack
20 pushed to stack
30 pushed to stack
30 popped from stack


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