AJAY KUMAR GARG ENGINEERING COLLEGE, GHAZIABAD

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

LIST OF EXPERIMENTS

COURSE:B TECH SEMESTER:3 SESSION: 2024-245 (ODD SEM)
BRANCH: CS
SUBJECT CODE & NAME: BCS-351, DATA STRUCTURE LAB

SI. No.	NAME OF EXPERIMENT	Date	Faculty Signature
1	Write a program in C to implement Linear Search.		
2	Write a program in C to implement Binary Search.		
3	Write a program in C to do Insertion (Beginning & End) in Single Linked list.		
4	Write a program in C to do Deletion (Beginning & End) in Single Linked list.		
5	Write a program in C to Insertion & Deletion in the Array.		
6	Write a program in C to implement Stack Using Array.		
7	Write a program in C to implement queue using array.		
8	Write a program in C to implement Stack using Linked List.		
9	Write a program in C to implement queue using Linked List.		

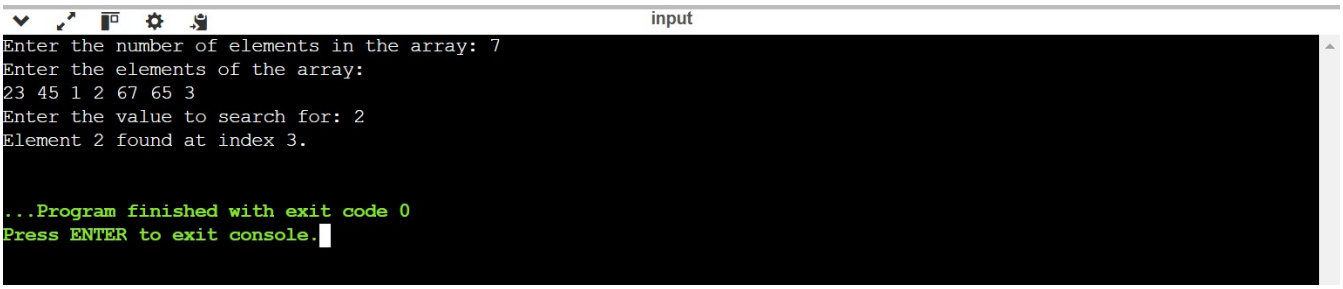
PROGRAM-01

OBJECTIVE : Write a program in C to implement Linear Search.

CODE :

```
#include <stdio.h>
  int linearSearch(int arr[], int size, int target) {
      for (int i = 0; i < size; i++) {
            if (arr[i] == target) {
                  return i; // Return the index if the target is found
            }
      }
      return -1; // Return -1 if the target is not found
}
   int main() {
      int n, target, result;
   // Input the number of elements in the array
      printf("Enter the number of elements in the array: ");
      scanf("%d", &n);

      int arr[n]; // Declare the array

      // Input the elements of the array
      printf("Enter the elements of the array:\n");
      for (int i = 0; i < n; i++) {
            scanf("%d", &arr[i]);
   }
   // Input the target value to search for
      printf("Enter the value to search for: ");
      scanf("%d", &target);
   // Perform linear search
      result = linearSearch(arr, n, target);

      // Output the result
      if (result != -1) {
            printf("Element %d found at index %d.\n", target, result);
      } else {
            printf("Element %d not found in the array.\n", target);
      }
return 0;
}
```

OUTPUT :


input
Enter the number of elements in the array: 7
Enter the elements of the array:
23 45 1 2 67 65 3
Enter the value to search for: 2
Element 2 found at index 3.
...Program finished with exit code 0
Press ENTER to exit console.

PROGRAM-02

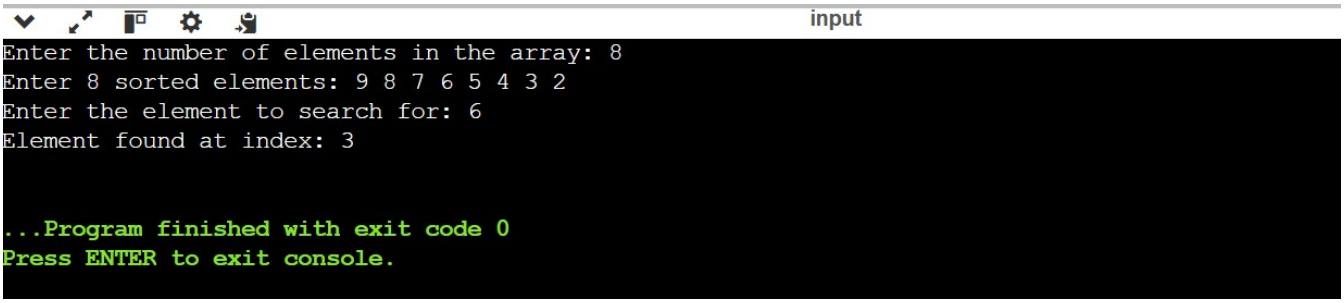
OBJECTIVE : Write a program in C to implement Binary Search.

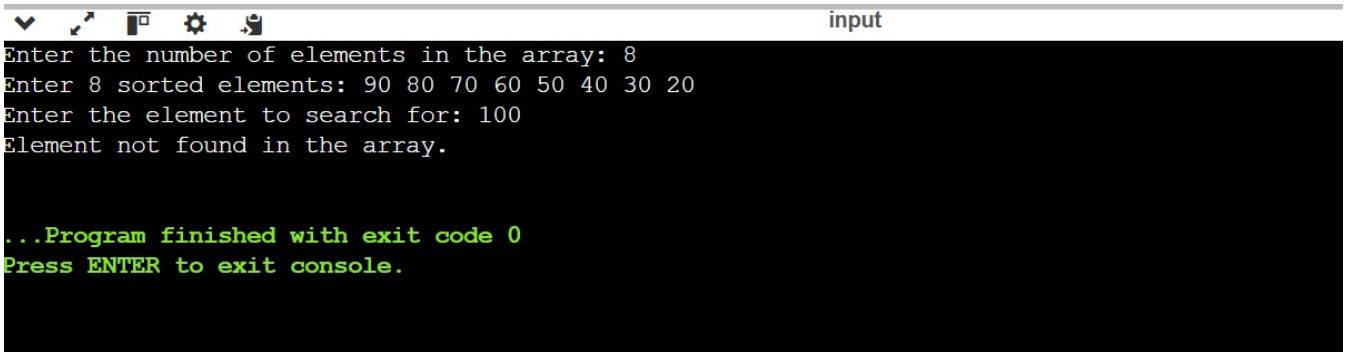
CODE :

```
#include <stdio.h>

// Function to perform binary search
int binarySearch(int arr[], int size, int target) {
        int left = 0;
        int right = size - 1;

        while (left <= right) {
                int mid = left + (right - left) / 2;

                // Check if the target is present at mid
                if (arr[mid] == target) {
                        return mid; // Element found
                }

                // If target is greater, ignore the left half
                if (arr[mid] < target) {
                        left = mid + 1;
                }
                // If target is smaller, ignore the right half
                else {
                        right = mid - 1;
                }
        }
  return -1; // Element not found
}

int main() {
        int n, target, result;
    // Input size of the array
        printf("Enter the number of elements in the array: ");
        scanf("%d", &n);
       int arr[n];

        // Input elements of the sorted array
        printf("Enter %d sorted elements: ", n);
```

```
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    // Input the target element to search for
    printf("Enter the element to search for: ");
    scanf("%d", &target);
    // Perform binary search
    result = binarySearch(arr, n, target);
    // Output the result
    if (result != -1) {
        printf("Element found at index: %d\n", result);
    } else {
        printf("Element not found in the array.\n");
    }
  return 0;
}
```

OUTPUT :





PROGRAM-03

OBJECTIVE : Write a program in C to do Insertion (Beg & End) in Single Linked list.

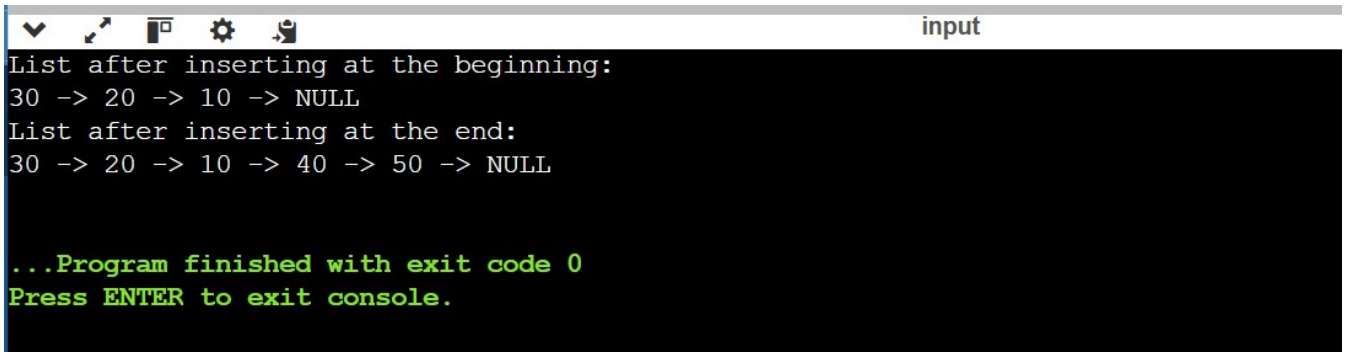
CODE :

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

// Define the structure for a node in the linked list
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node at the beginning of the list
void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head; // Link the new node to the old head
    *head = newNode;            // Update head to point to the new node
}

// Function to insert a node at the end of the list
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode; // If the list is empty, make the new node the head
        return;
    }
```

```
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next; // Traverse to the last node
    }
    temp->next = newNode; // Link the last node to the new node
}

// Function to print the linked list
void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}


int main() {
    struct Node* head = NULL; // Initialize the head of the list

    // Insert nodes at the beginning
    insertAtBeginning(&head, 10);
    insertAtBeginning(&head, 20);
    insertAtBeginning(&head, 30);
    printf("List after inserting at the beginning:\n");
    printList(head);

    // Insert nodes at the end
    insertAtEnd(&head, 40);
    insertAtEnd(&head, 50);
    printf("List after inserting at the end:\n");
    printList(head);


    struct Node* current = head;
    struct Node* nextNode;
```

```
    while (current != NULL) {
        nextNode = current->next;
        free(current);
        current = nextNode;
    }

    return 0;
}
```

OUTPUT :



PROGRAM-04

OBJECTIVE : Write a program in C to do Deletion (Beg & End) in Single Linked list.

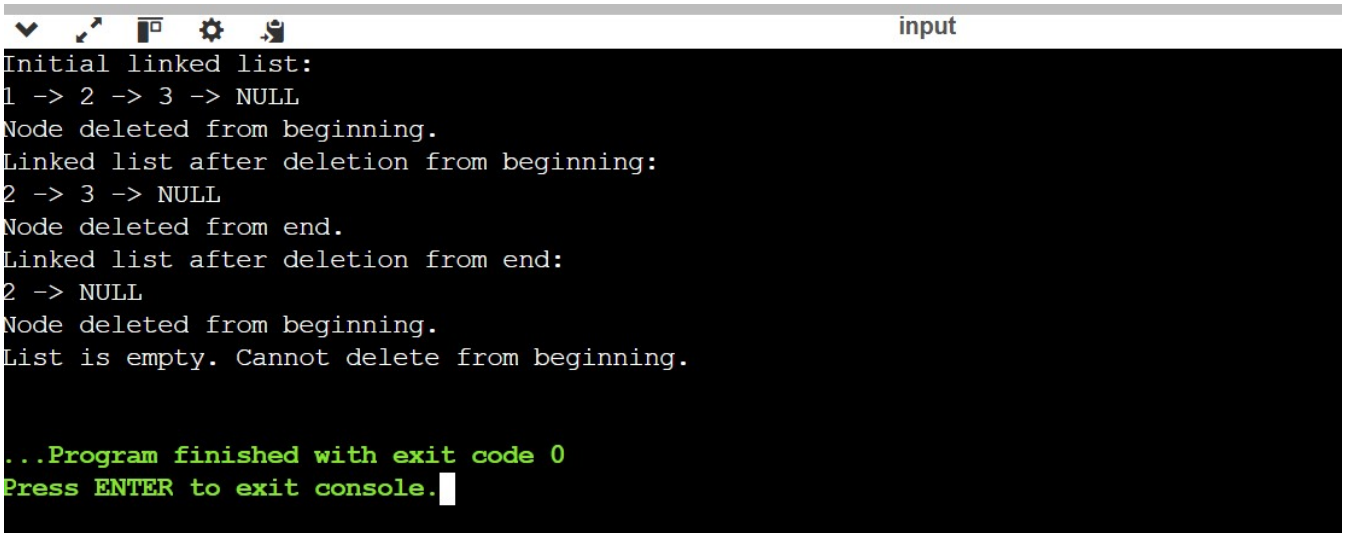
CODE :

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

// Define the structure for a node in the linked list
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to delete a node from the beginning of the linked list
void deleteFromBeginning(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty. Cannot delete from beginning.\n");
        return;
    }
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
    printf("Node deleted from beginning.\n");
}

// Function to delete a node from the end of the linked list
void deleteFromEnd(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty. Cannot delete from end.\n");
        return;
```

```
    }

    struct Node* temp = *head;
    // If there's only one node
    if (temp->next == NULL) {
        free(temp);
        *head = NULL;
        printf("Node deleted from end.\n");
        return;
    }

    // Traverse to the second last node
    while (temp->next->next != NULL) {
        temp = temp->next;
    }
    free(temp->next);
    temp->next = NULL;
    printf("Node deleted from end.\n");
}

// Function to print the linked list
void printList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}


int main() {
    struct Node* head = NULL;

    // Adding some nodes to the linked list for demonstration
    head = createNode(1);
    head->next = createNode(2);
    head->next->next = createNode(3);
```

```
    printf("Initial linked list:\n");
    printList(head);

    // Deleting from the beginning
    deleteFromBeginning(&head);
    printf("Linked list after deletion from beginning:\n");
    printList(head);

    // Deleting from the end
    deleteFromEnd(&head);
    printf("Linked list after deletion from end:\n");
    printList(head);

    // Clean up remaining nodes
    deleteFromBeginning(&head);
    deleteFromBeginning(&head);

    return 0;
}
```

OUTPUT :



PROGRAM-05

OBJECTIVE : Write a program in C to Insertion & Deletion in the Array

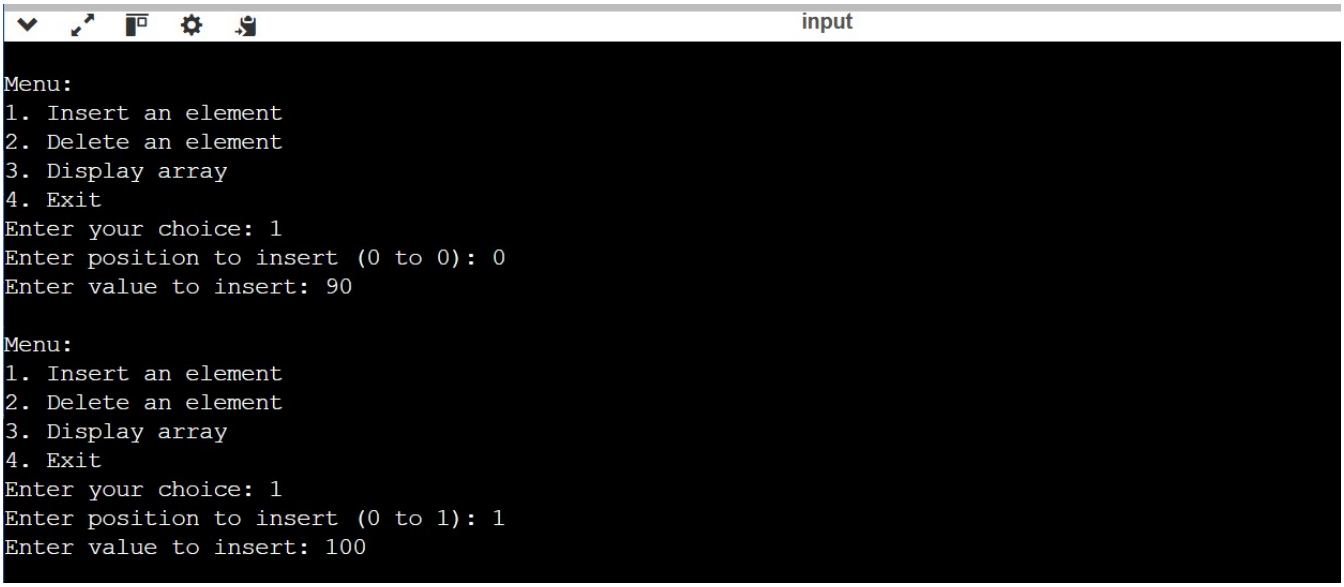
CODE :

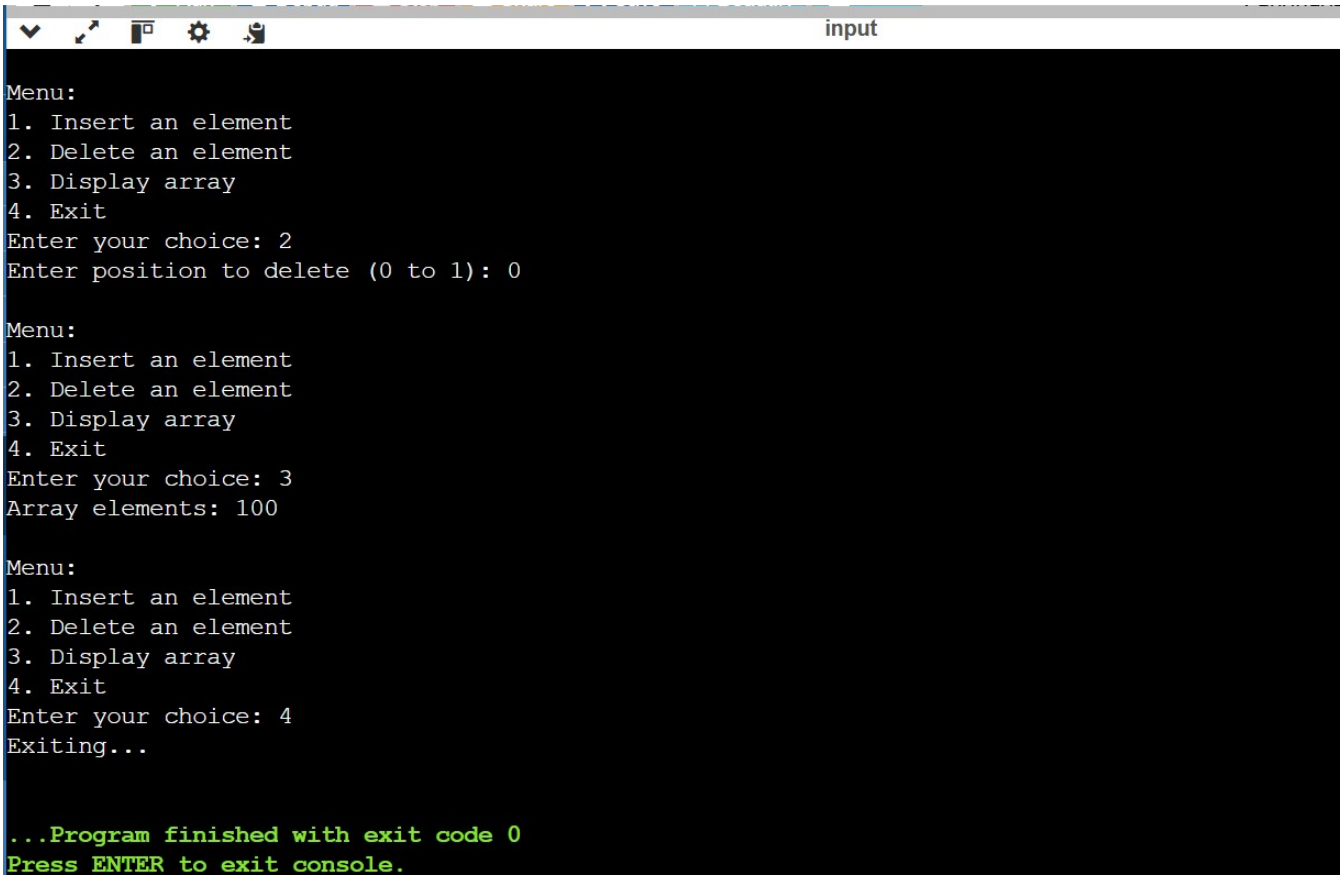
```
#include <stdio.h>

#define MAX_SIZE 100

void insert(int arr[], int *n, int pos, int value) {
    if (*n >= MAX_SIZE) {
        printf("Array is full. Cannot insert more elements.\n");
        return;
    }
    if (pos < 0 || pos > *n) {
        printf("Invalid position. Please enter a position between 0 and %d.\n", *n);
        return;
    }

    // Shift elements to the right to make space for the new element
    for (int i = *n; i > pos; i--) {
        arr[i] = arr[i - 1];
    }
    arr[pos] = value; // Insert the new element
    (*n)++; // Increase the size of the array
}

void delete(int arr[], int *n, int pos) {
    if (*n == 0) {
        printf("Array is empty. Cannot delete any elements.\n");
        return;
    }
    if (pos < 0 || pos >= *n) {
        printf("Invalid position. Please enter a position between 0 and %d.\n", *n - 1);
        return;
    }

    // Shift elements to the left to fill the gap
    for (int i = pos; i < *n - 1; i++) {
        arr[i] = arr[i + 1];
```

```
    }
    (*n)--; // Decrease the size of the array
}

void display(int arr[], int n) {
    if (n == 0) {
        printf("Array is empty.\n");
        return;
    }
    printf("Array elements: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[MAX_SIZE];
    int n = 0; // Current size of the array
    int choice, pos, value;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Insert an element\n");
        printf("2. Delete an element\n");
        printf("3. Display array\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter position to insert (0 to %d): ", n);
                scanf("%d", &pos);
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert(arr, &n, pos, value);
                break;

            case 2:
                printf("Enter position to delete (0 to %d): ", n - 1);
                scanf("%d", &pos);
```

```
                delete(arr, &n, pos);
                break;

            case 3:
                display(arr, n);
                break;

            case 4:
                printf("Exiting...\n");
                return 0;

            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

OUTPUT :


input
Menu:
1. Insert an element
2. Delete an element
3. Display array
4. Exit
Enter your choice: 1
Enter position to insert (0 to 0): 0
Enter value to insert: 90
Menu:
1. Insert an element
2. Delete an element
3. Display array
4. Exit
Enter your choice: 1
Enter position to insert (0 to 1): 1
Enter value to insert: 100


input
Menu:
1. Insert an element
2. Delete an element
3. Display array
4. Exit
Enter your choice: 2
Enter position to delete (0 to 1): 0
Menu:
1. Insert an element
2. Delete an element
3. Display array
4. Exit
Enter your choice: 3
Array elements: 100
Menu:
1. Insert an element
2. Delete an element
3. Display array
4. Exit
Enter your choice: 4
Exiting...
...Program finished with exit code 0
Press ENTER to exit console.

PROGRAM-06

OBJECTIVE : Write a program in C to implement Stack Using Array.

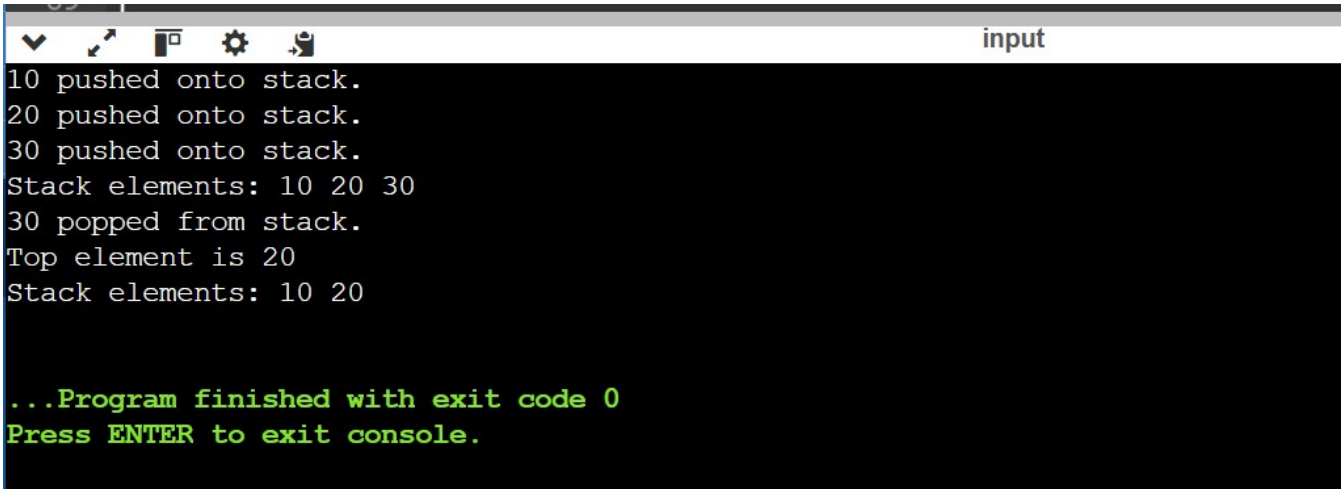
CODE :

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

#define MAX 100 // Define the maximum size of the stack

// Structure to represent a stack
struct Stack {
	int arr[MAX];
	int top;
};

// Function to create a stack and initialize its top
struct Stack* createStack() {
	struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
	stack->top = -1; // Stack is initially empty
	return stack;
}

// Function to check if the stack is full
bool isFull(struct Stack* stack) {
	return stack->top == MAX - 1;
}

// Function to check if the stack is empty
bool isEmpty(struct Stack* stack) {
	return stack->top == -1;
}

// Function to add an item to the stack
void push(struct Stack* stack, int item) {
	if (isFull(stack)) {
		printf("Stack overflow! Cannot push %d onto stack.\n", item);
		return;
	}
```

```
    stack->arr[++stack->top] = item;
    printf("%d pushed onto stack.\n", item);
}

// Function to remove an item from the stack
int pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack underflow! Cannot pop from empty stack.\n");
        return -1; // Return an invalid value
    }
    return stack->arr[stack->top--];
}

// Function to get the top item of the stack
int peek(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty! Cannot peek.\n");
        return -1; // Return an invalid value
    }
    return stack->arr[stack->top];
}

// Function to display the stack
void display(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty!\n");
        return;
    }
    printf("Stack elements: ");
    for (int i = 0; i <= stack->top; i++) {
        printf("%d ", stack->arr[i]);
    }
    printf("\n");
}

// Main function to demonstrate stack operations
int main() {
    struct Stack* stack = createStack();

    push(stack, 10);
    push(stack, 20);
    push(stack, 30);
```

```
    display(stack);

    printf("%d popped from stack.\n", pop(stack));
    printf("Top element is %d\n", peek(stack));
    display(stack);

    // Clean up
    free(stack);
    return 0;
}
```

OUTPUT :



PROGRAM-07

OBJECTIVE : Write a program in C to implement queue using array.

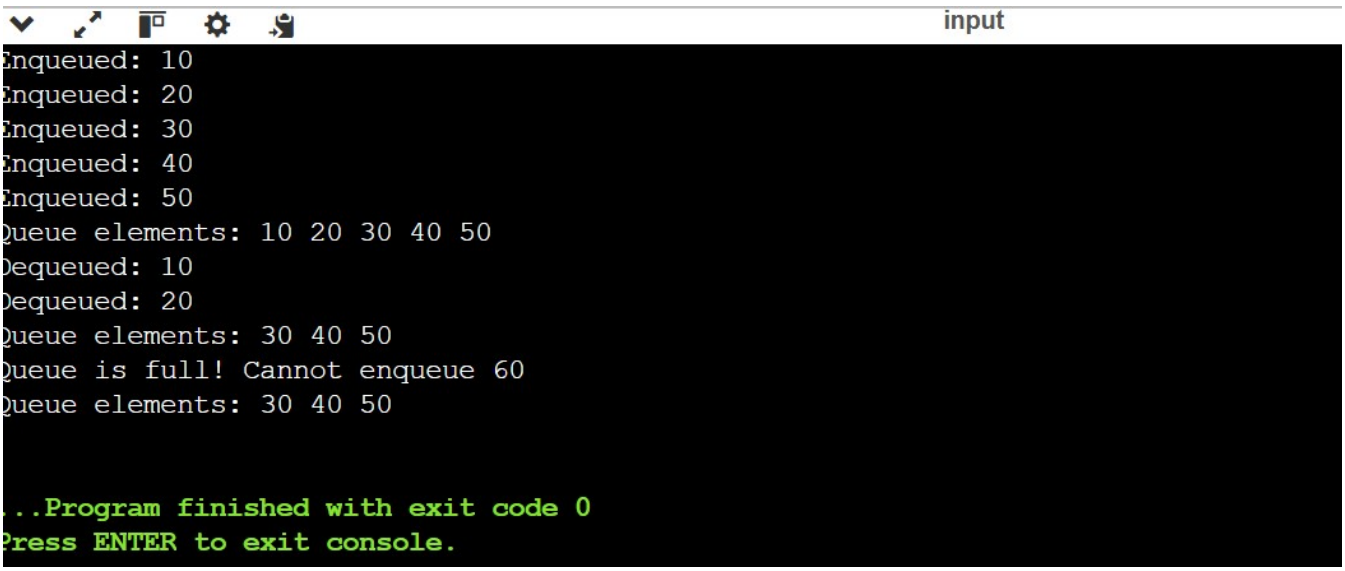
CODE :

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

#define MAX 5 // Maximum size of the queue

typedef struct Queue {
    int items[MAX];
    int front;
    int rear;
} Queue;

// Function to create a queue
Queue* createQueue() {
    Queue* q = (Queue*)malloc(sizeof(Queue));
    q->front = -1;
    q->rear = -1;
    return q;
}

// Function to check if the queue is full
int isFull(Queue* q) {
    return (q->rear == MAX - 1);
}

// Function to check if the queue is empty
int isEmpty(Queue* q) {
    return (q->front == -1 || q->front > q->rear);
}

// Function to add an element to the queue
void enqueue(Queue* q, int value) {
    if (isFull(q)) {
        printf("Queue is full! Cannot enqueue %d\n", value);
    } else {
        if (q->front == -1) {
```

```
            q->front = 0; // Initialize front on first enqueue
        }
        q->rear++;
        q->items[q->rear] = value;
        printf("Enqueued: %d\n", value);
    }
}

// Function to remove an element from the queue
int dequeue(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty! Cannot dequeue.\n");
        return -1; // Return -1 to indicate an error
    } else {
        int dequeuedValue = q->items[q->front];
        q->front++;
        // Reset front and rear if the queue becomes empty
        if (q->front > q->rear) {
            q->front = q->rear = -1;
        }
        printf("Dequeued: %d\n", dequeuedValue);
        return dequeuedValue;
    }
}

// Function to display the queue
void display(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty!\n");
    } else {
        printf("Queue elements: ");
        for (int i = q->front; i <= q->rear; i++) {
            printf("%d ", q->items[i]);
        }
        printf("\n");
    }
}

// Main function to test the queue implementation
int main() {
    Queue* q = createQueue();
```

```
    enqueue(q, 10);
    enqueue(q, 20);
    enqueue(q, 30);
    enqueue(q, 40);
    enqueue(q, 50);
    display(q);

    dequeue(q);
    dequeue(q);
    display(q);

    enqueue(q, 60);
    display(q);

    // Clean up
    free(q);
    return 0;
}
```

OUTPUT :



PROGRAM-08

OBJECTIVE : Write a program in C to implement Stack using Linked List.

CODE :

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

// Define the structure for a stack node
struct Node {
	int data;
	struct Node* next;
};

// Define the structure for the stack
struct Stack {
	struct Node* top;
};

// Function to create a new stack
struct Stack* createStack() {
	struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
	stack->top = NULL;
	return stack;
}

// Function to check if the stack is empty
int isEmpty(struct Stack* stack) {
	return stack->top == NULL;
}

// Function to push an element onto the stack
void push(struct Stack* stack, int data) {
	struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
	newNode->data = data;
	newNode->next = stack->top;
	stack->top = newNode;
	printf("%d pushed onto stack\n", data);
}
```

```
// Function to pop an element from the stack
int pop(struct Stack* stack) {
	if (isEmpty(stack)) {
		printf("Stack underflow! Cannot pop from empty stack.\n");
		return -1; // Return -1 to indicate stack underflow
	}
	struct Node* temp = stack->top;
	int poppedData = temp->data;
	stack->top = stack->top->next;
	free(temp);
	return poppedData;
}

// Function to peek at the top element of the stack
int peek(struct Stack* stack) {
	if (isEmpty(stack)) {
		printf("Stack is empty! Cannot peek.\n");
		return -1; // Return -1 to indicate stack is empty
	}
	return stack->top->data;
}

// Function to display the stack elements
void display(struct Stack* stack) {
	if (isEmpty(stack)) {
		printf("Stack is empty!\n");
		return;
	}
	struct Node* current = stack->top;
	printf("Stack elements: ");
	while (current != NULL) {
		printf("%d ", current->data);
		current = current->next;
	}
	printf("\n");
}

// Main function to demonstrate stack operations
int main() {
	struct Stack* stack = createStack();
```

```
    push(stack, 10);
    push(stack, 20);
    push(stack, 30);

    display(stack);

    printf("%d popped from stack\n", pop(stack));
    printf("Top element is %d\n", peek(stack));

    display(stack);

    return 0;
}
```

OUTPUT :

PROGRAM-09

OBJECTIVE : Write a program in C to implement queue using Linked List.

CODE :

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

// Define a structure for the queue node
struct Node {
    int data;
    struct Node* next;
};

// Define a structure for the queue
struct Queue {
    struct Node* front;
    struct Node* rear;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to create an empty queue
struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = queue->rear = NULL;
    return queue;
}

// Function to add an item to the queue
void enqueue(struct Queue* queue, int data) {
    struct Node* newNode = createNode(data);
    if (queue->rear == NULL) {
```

```
        // If the queue is empty, both front and rear point to the new node
        queue->front = queue->rear = newNode;
        return;
    }
    // Add the new node at the end of the queue and update the rear pointer
    queue->rear->next = newNode;
    queue->rear = newNode;
}

// Function to remove an item from the queue
int dequeue(struct Queue* queue) {
    if (queue->front == NULL) {
        printf("Queue is empty!\n");
        return -1; // Indicate that the queue is empty
    }
    struct Node* temp = queue->front;
    int data = temp->data;
    queue->front = queue->front->next;

    // If the front becomes NULL, then change rear also to NULL
    if (queue->front == NULL) {
        queue->rear = NULL;
    }
    free(temp);
    return data;
}

// Function to display the queue
void displayQueue(struct Queue* queue) {
    if (queue->front == NULL) {
        printf("Queue is empty!\n");
        return;
    }
    struct Node* temp = queue->front;
    printf("Queue elements: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
```

```
// Main function to demonstrate the queue operations
int main() {
    struct Queue* queue = createQueue();

    enqueue(queue, 10);
    enqueue(queue, 20);
    enqueue(queue, 30);

    displayQueue(queue);

    printf("Dequeued: %d\n", dequeue(queue));
    displayQueue(queue);

    printf("Dequeued: %d\n", dequeue(queue));
    displayQueue(queue);

    printf("Dequeued: %d\n", dequeue(queue));
    displayQueue(queue);

    printf("Dequeued: %d\n", dequeue(queue)); // Trying to dequeue from an empty queue

    return 0;
}
```

OUTPUT :