Q1. Write a menu driven program to implement the following operations on stack.

a. PUSH()

b. POP()

c. Display()

Algorithm:

Define a Node:

- Create a structure node with fields data (to store stack elements) and link (to point to the next node).

PUSH Operation:

- Allocate memory for a new node.
- Assign the input value to the data field of the node.
- Set the link of the new node to the current top of the stack (start).
- Update start to point to the new node.

POP Operation:

- If start is NULL, print "Stack underflow" and exit the operation.
- Otherwise, store the current top node in a temporary pointer.
- Move start to the next node.
- Free the memory of the temporary pointer and print a message.

Display Operation:

- Initialize a pointer to the top of the stack (start).
- Traverse through the stack until the pointer is NULL, printing the value of each node.

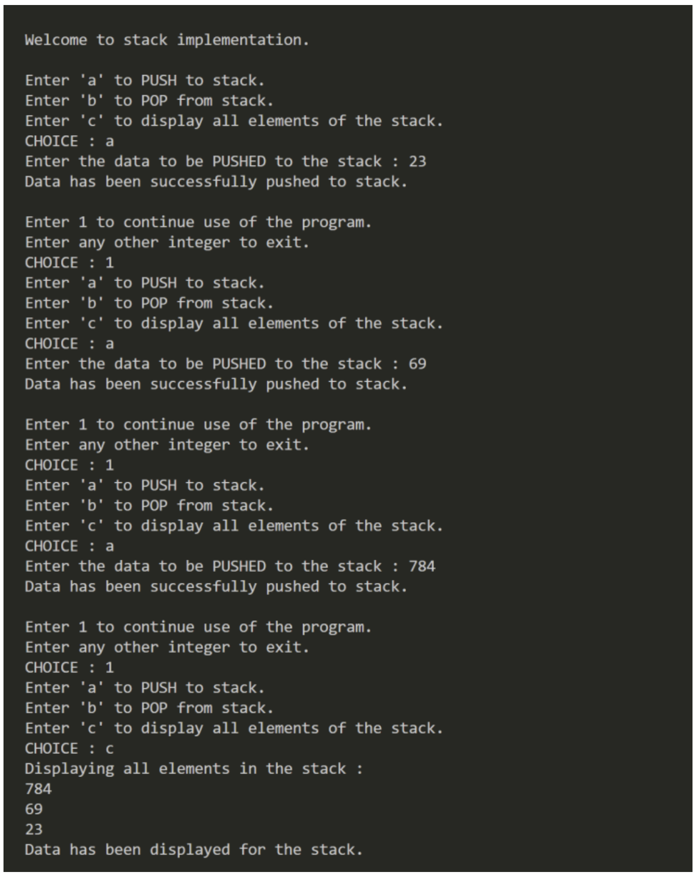
Main Function:

- Initialize start as NULL.
- Use a loop to display a menu with options:
    - a: Call the PUSH function with input data.
    - b: Call the POP function.
    - c: Call the display function to print stack elements.
    - Default: Print an invalid choice message.
- Allow the user to exit or continue based on input.

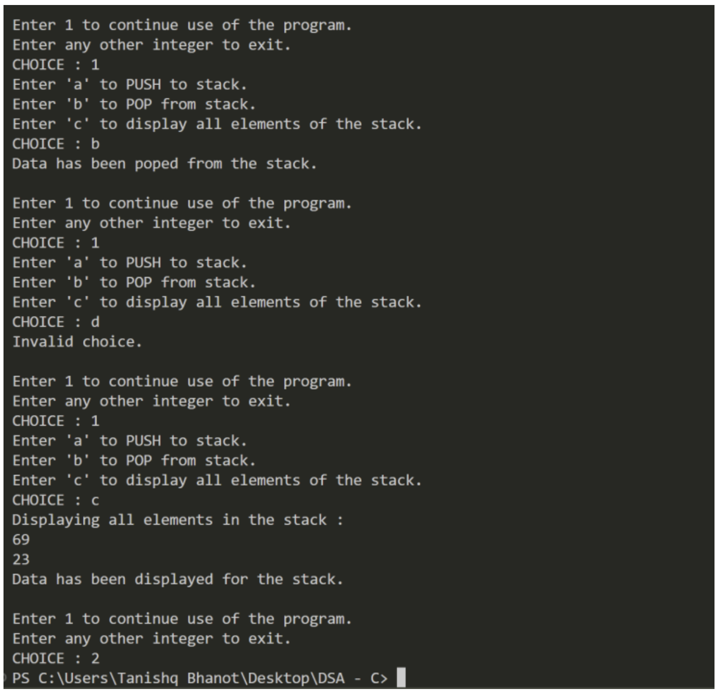
Program:

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

struct node {
   int data;
   struct node* link;
};

struct node* PUSH(struct node* start, int data) {
   struct node* temp = (struct node*)malloc(sizeof(struct node));
   temp->data = data;
   temp->link = start;
   start = temp;
   return start;
}

struct node* POP(struct node* start) {
   if(start == NULL) {
      printf("No data to be popped. Stack underflow.\n\n");
      return start;
   }
   struct node* temp = start;
   start = start->link;
   printf("Data has been popped from the stack.\n\n");
   free(temp);
   return start;
}

void display(struct node* start) {
   struct node* p = start;
   while(p != NULL) {
        printf("%d\n", p->data);
        p = p->link;
    }
    return;
}

int main() {
    struct node* start = NULL;
    printf("\n\nWelcome to stack implementation.\n\n");
    int choice = 1;
    do {
        char x;
        printf("Enter 'a' to PUSH to stack.\nEnter 'b' to POP from stack.\nEnter 'c' to display all elements of the
stack.\nCHOICE : ");
        scanf("%s", &x);

        switch(x){
            case 'a' : {
                int dat;
                printf("Enter the data to be PUSHED to the stack : ");
                scanf("%d", &dat);
                start = PUSH(start, dat);
                printf("Data has been successfully pushed to stack.\n\n");
                break;
            }

            case 'b' : {
                start = POP(start);
                break;
            }

            case 'c' : {
                printf("Displaying all elements in the stack : \n");
            display(start);

            printf("Data has been displayed for the stack.\n\n");

            break;

          }


          default : {

            printf("Invalid choice.\n\n");

          }

        }


        printf("Enter 1 to continue use of the program.\nEnter any other integer to exit.\nCHOICE : ");

        scanf("%d", &choice);

    }while(choice == 1);

    return 0;

}
```

Output:



```
Welcome to stack implementation.

Enter 'a' to PUSH to stack.
Enter 'b' to POP from stack.
Enter 'c' to display all elements of the stack.
CHOICE : a
Enter the data to be PUSHED to the stack : 23
Data has been successfully pushed to stack.

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 'a' to PUSH to stack.
Enter 'b' to POP from stack.
Enter 'c' to display all elements of the stack.
CHOICE : a
Enter the data to be PUSHED to the stack : 69
Data has been successfully pushed to stack.

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 'a' to PUSH to stack.
Enter 'b' to POP from stack.
Enter 'c' to display all elements of the stack.
CHOICE : a
Enter the data to be PUSHED to the stack : 784
Data has been successfully pushed to stack.

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 'a' to PUSH to stack.
Enter 'b' to POP from stack.
Enter 'c' to display all elements of the stack.
CHOICE : c
Displaying all elements in the stack :
784
69
23
Data has been displayed for the stack.
```



```
Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 'a' to PUSH to stack.
Enter 'b' to POP from stack.
Enter 'c' to display all elements of the stack.
CHOICE : b
Data has been poped from the stack.

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 'a' to PUSH to stack.
Enter 'b' to POP from stack.
Enter 'c' to display all elements of the stack.
CHOICE : d
Invalid choice.

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 'a' to PUSH to stack.
Enter 'b' to POP from stack.
Enter 'c' to display all elements of the stack.
CHOICE : c
Displaying all elements in the stack :
69
23
Data has been displayed for the stack.

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 2
PS C:\Users\Tanishq Bhanot\Desktop\DSA - C>
```

Q2. Write a menu driven program to implement the following operations on Queue:

a. Enqueue()

b. Dequeue()

c. Display()

Algorithm:

Define a Node:

- Create a structure node with three fields:
  - prev: Pointer to the previous node.
  - data: To store the value of the node.
  - next: Pointer to the next node.

Create New Node:

- Allocate memory for a new node using malloc.
- Assign the input value to the data field.
- Set prev and next pointers to NULL.
- Return the new node.

Enqueue Operation:

- Input: head, tail, and the data to enqueue.
- Create a new node using the createNewNode function.

- If the queue is empty (head == NULL):
    - Set both head and tail to the new node.
- Otherwise:
    - Set tail->next to the new node.
    - Set newNode->prev to tail.
    - Update tail to point to the new node.

Dequeue Operation:

- Input: head and tail.
- If the queue is empty (head == NULL):
    - Print "Queue is empty" and exit.
- Otherwise:
    - Store the head in a temporary pointer.
    - Move head to the next node (head = head->next).
    - If the new head is not NULL, set head->prev to NULL.
    - If the new head is NULL, set tail to NULL (queue becomes empty).
    - Free the memory of the temporary node.

Display Operation:

- Input: head.
- Initialize a pointer p to head.
- While p is not NULL:
    - Print the value of p->data.
    - Move p to the next node (p = p->next).

Main Function:

- Initialize head and tail as NULL.
- Print a welcome message.
- Enter a loop to display the menu and handle user input:
    - Print options:
        - 'a': Enqueue a value using the Enqueue function.
        - 'b': Dequeue a value using the Dequeue function.
        - 'c': Display all elements using the Display function.
        - Default: Print an invalid choice message.
    - Allow the user to exit or continue based on input (choice).

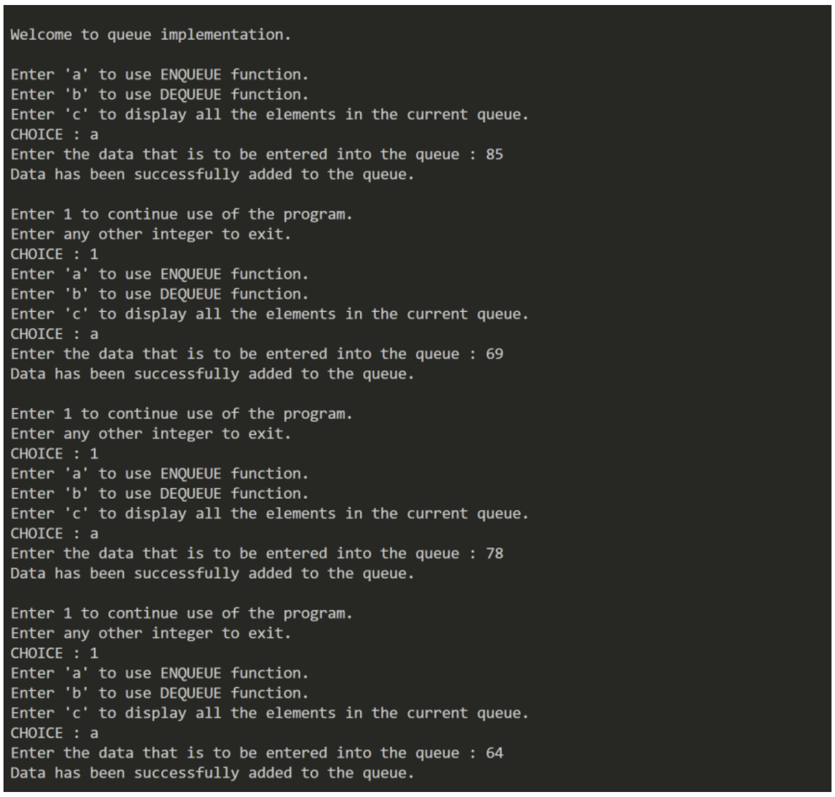
Program :

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

struct node {
   struct node* prev;
   int data;
   struct node* next;
};

struct node* createNewNode(int data) {
   struct node* newNode = (struct node*)malloc(sizeof(struct node));
   newNode->data = data;
   newNode->prev = NULL;
   newNode->next = NULL;
   return newNode;
}

void Enqueue(struct node** head, struct node** tail, int data) {
   struct node *newNode = createNewNode(data);
   if(*head == NULL) {
      *head = newNode;
      *tail = newNode;
      return;
   }
   (*tail)->next = newNode;
   newNode->prev = *tail;
   *tail = newNode;
   return;
}
```

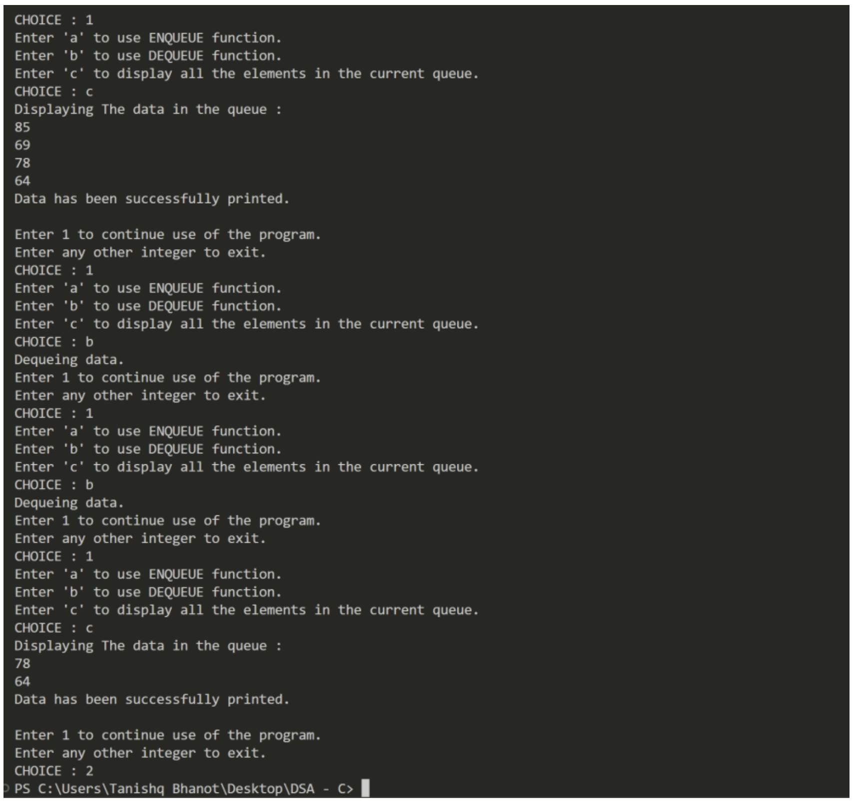
```c
void Dequeue(struct node** head, struct node** tail) {
   if (*head == NULL) {
      printf("List is empty. Nothing to delete.\n\n");
      return;
   }
   struct node* temp = *head;
   *head = (*head)->next;
   if (*head != NULL) {
      (*head)->prev = NULL;
   }
   else {
      *tail = NULL;
   }
   free(temp);
}

void Display(struct node* head) {
   struct node *p = head;
   while(p != NULL) {
      printf("%d\n", p->data);
      p = p->next;
   }
   return;
}

int main() {
   struct node* head = NULL;
   struct node* tail = NULL;
   printf("\n\nWelcome to queue implementation.\n\n");
   int choice = 1;
   do {
      char x;
```

```
        printf("Enter 'a' to use ENQUEUE function.\nEnter 'b' to use DEQUEUE function.\nEnter 'c' to display all the elements in the current queue.\nCHOICE : ");

        scanf("%s", &x);


        switch(x){

          case 'a' : {

            int dat;

            printf("Enter the data that is to be entered into the queue : ");

            scanf("%d", &dat);

            Enqueue(&head, &tail, dat);

            printf("Data has been successfully added to the queue.\n\n");

            break;

          }


          case 'b' : {

            printf("Dequeing data.\n");

            Dequeue(&head, &tail);

            break;

          }


          case 'c' : {

            printf("Displaying The data in the queue : \n");

            Display(head);

            printf("Data has been successfully printed.\n\n");

            break;

          }


          default : {

            printf("Invalid choice.\n\n");

          }

        }


        printf("Enter 1 to continue use of the program.\nEnter any other integer to exit.\nCHOICE : ");
        scanf("%d", &choice);
    }while(choice == 1);
    return 0;
}
```

Output:



```
Welcome to queue implementation.

Enter 'a' to use ENQUEUE function.
Enter 'b' to use DEQUEUE function.
Enter 'c' to display all the elements in the current queue.
CHOICE : a
Enter the data that is to be entered into the queue : 85
Data has been successfully added to the queue.

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 'a' to use ENQUEUE function.
Enter 'b' to use DEQUEUE function.
Enter 'c' to display all the elements in the current queue.
CHOICE : a
Enter the data that is to be entered into the queue : 69
Data has been successfully added to the queue.

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 'a' to use ENQUEUE function.
Enter 'b' to use DEQUEUE function.
Enter 'c' to display all the elements in the current queue.
CHOICE : a
Enter the data that is to be entered into the queue : 78
Data has been successfully added to the queue.

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 'a' to use ENQUEUE function.
Enter 'b' to use DEQUEUE function.
Enter 'c' to display all the elements in the current queue.
CHOICE : a
Enter the data that is to be entered into the queue : 64
Data has been successfully added to the queue.
```



```
CHOICE : 1
Enter 'a' to use ENQUEUE function.
Enter 'b' to use DEQUEUE function.
Enter 'c' to display all the elements in the current queue.
CHOICE : c
Displaying The data in the queue :
85
69
78
64
Data has been successfully printed.

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 'a' to use ENQUEUE function.
Enter 'b' to use DEQUEUE function.
Enter 'c' to display all the elements in the current queue.
CHOICE : b
Dequeing data.
Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 'a' to use ENQUEUE function.
Enter 'b' to use DEQUEUE function.
Enter 'c' to display all the elements in the current queue.
CHOICE : b
Dequeing data.
Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 'a' to use ENQUEUE function.
Enter 'b' to use DEQUEUE function.
Enter 'c' to display all the elements in the current queue.
CHOICE : c
Displaying The data in the queue :
78
64
Data has been successfully printed.

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 2
PS C:\Users\Tanishq Bhanot\Desktop\DSA - C>
```

Q3. Write a menu driven program to implement the following operations on circular Queue:

> **a. Enqueue()**
>
> **b. Dequeue()**
>
> **c. Display()**

Algorithm:

Define the Node Structure (Node):

- A Node structure with three fields:
  - data: Stores the value of the node.
  - next: Points to the next node in the list.
  - prev: Points to the previous node in the list.

Define the Queue Structure (CircularQueue):

- A CircularQueue structure with two pointers:
  - front: Points to the front node in the queue.
  - rear: Points to the rear node in the queue.

Initialize the Queue (initialize):

- Set both front and rear to NULL, indicating that the queue is initially empty.

Enqueue Operation (enqueue):

- Input: A CircularQueue and a value to enqueue.
- Create a new node and set its data field to the provided value.
- If the queue is empty (front == NULL), set both front and rear to the new node, and set the new node’s next and prev to point to itself (since the queue is circular).
- If the queue is not empty, update the next pointer of the new node to point to the front node, and the prev pointer to point to the rear node. Set the next pointer of rear to the new node and prev pointer of front to the new node. Finally, update the rear pointer to the new node.
- Print a message confirming the enqueued value.

Dequeue Operation (dequeue):

- Input: A CircularQueue.
- If the queue is empty (front == NULL), print an error message and return.
- Otherwise, store the data from the front node and free the memory of the front node.
- If the queue has only one node (front == rear), set both front and rear to NULL.

- If the queue has more than one node, update the front pointer to the next node, adjust the prev pointer of the new front to point to the rear, and set the next pointer of the new rear to point to the new front.
- Print a message confirming the dequeued value.

Display the Queue (display):

- Input: A CircularQueue.
- If the queue is empty (front == NULL), print an error message.
- Otherwise, traverse the queue starting from front, printing the data of each node until reaching the front again (to complete the circular traversal).
- Print the queue contents.

Main Function:

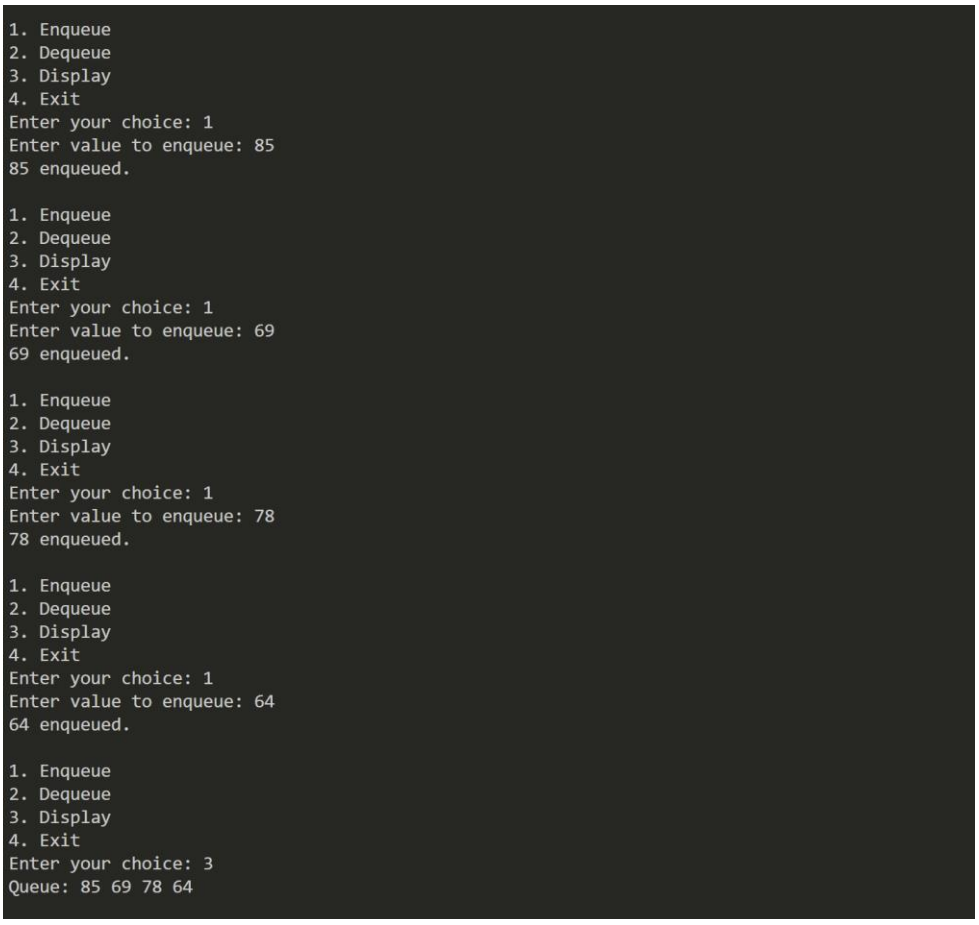
- Initialize the CircularQueue by calling initialize.
- Provide a menu with options:
    - 1: Enqueue operation.
    - 2: Dequeue operation.
    - 3: Display the queue.
    - 4: Exit the program.
- Perform the corresponding operation based on user input.
- Continuously prompt for the next operation until the user chooses to exit.

Program:

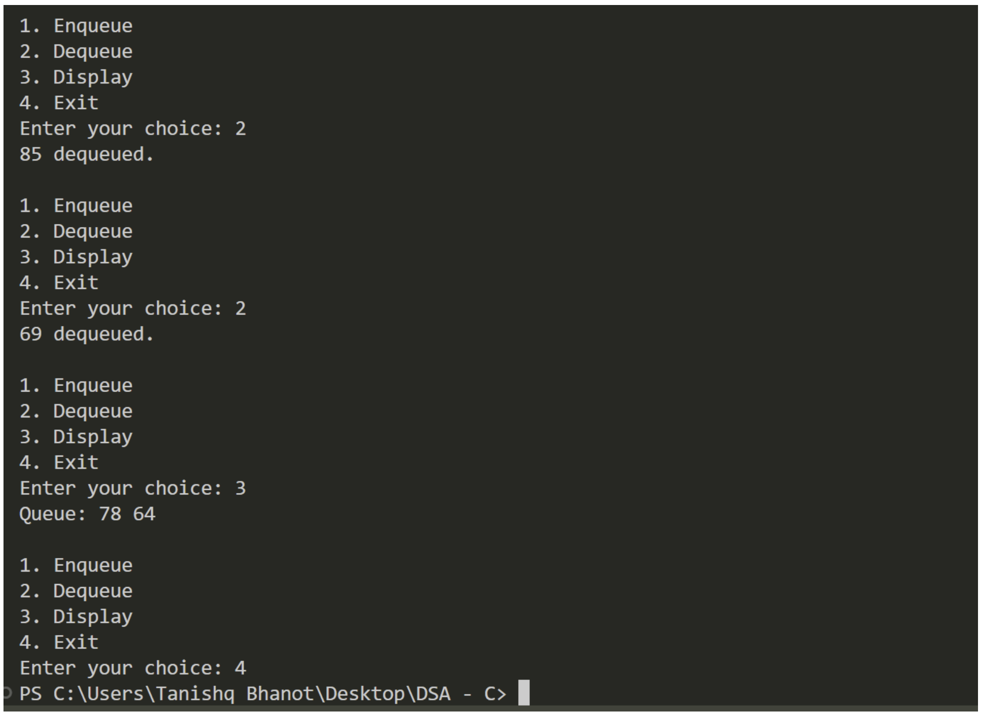
```
#include <stdio.h>

#include <stdlib.h>


typedef struct Node {

   int data;

   struct Node* next;

   struct Node* prev;

} Node;


typedef struct CircularQueue {

   Node* front;
```

```c
    Node* rear;
} CircularQueue;

void initialize(CircularQueue* q) {
    q->front = NULL;
    q->rear = NULL;
}

void enqueue(CircularQueue* q, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    if (q->front == NULL) {
        q->front = q->rear = newNode;
        newNode->next = newNode->prev = newNode;
    } else {
        newNode->next = q->front;
        newNode->prev = q->rear;
        q->rear->next = newNode;
        q->front->prev = newNode;
        q->rear = newNode;
    }
    printf("%d enqueued.\n", value);
}

void dequeue(CircularQueue* q) {
    if (q->front == NULL) {
        printf("Queue is empty.\n");
        return;
    }
    Node* temp = q->front;
    int value = temp->data;
    if (q->front == q->rear) {
        q->front = q->rear = NULL;
    } else {
        q->front = q->front->next;
        q->front->prev = q->rear;
        q->rear->next = q->front;
    }
    free(temp);
    printf("%d dequeued.\n", value);
}

void display(CircularQueue* q) {
    if (q->front == NULL) {
        printf("Queue is empty.\n");
        return;
    }
    Node* temp = q->front;
    printf("Queue: ");
    do {
        printf("%d ", temp->data);
        temp = temp->next;
    } while (temp != q->front);
    printf("\n");
}

int main() {
    CircularQueue q;
    initialize(&q);
    int choice, value;

    while (1) {
        printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {

          case 1: {
            printf("Enter value to enqueue: ");
            scanf("%d", &value);
            enqueue(&q, value);
            break;
            }

          case 2: {
            dequeue(&q);
            break;
            }

          case 3: {
            display(&q);
            break;
            }

          case 4: {
            exit(0);
            }

          default:
            printf("Invalid choice.\n");
        }
    }
    return 0;
}
```

Output:



```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 85
85 enqueued.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 69
69 enqueued.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 78
78 enqueued.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 64
64 enqueued.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue: 85 69 78 64
```



```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
85 dequeued.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
69 dequeued.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue: 78 64

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
PS C:\Users\Tanishq Bhanot\Desktop\DSA - C>
```

Q4. Write a menu driven program to implement the following operations on singly linked list:

a. Insertion()

i. Beginning

ii. End

iii. At a given position

b. Deletion()

i. Beginning

ii. End

iii. At a given position

c. Search(): search for the given element on the list

Algorithm:

Define a Node:

- Create a structure node with two fields:
  - data: Stores the value of the node.
  - next: Points to the next node in the list.

Insert at Beginning (insertAtBeg):

- Input: start (head of the list), data (data to be inserted).
- Allocate memory for a new node.
- Set the new node's data to the input data.
- Set the new node's next to point to the current start.
- Update start to point to the new node.
- Return the updated start.

Insert at End (insertAtEnd):

- Input: start (head of the list), data (data to be inserted).
- Create a new node with the input data.
- Traverse the list to find the last node (node where next is NULL).
- Set the last node's next to the new node.
- Set the new node's next to NULL.
- Return the updated start.

Insert at Position (insertAtPos):

- Input: start (head of the list), data (data to be inserted), pos (position).

- If the position is 0, call insertAtBeg to insert at the beginning and return.
- Traverse the list to find the node at position pos-1.
- If the node is NULL, print an error message and return.
- Insert the new node after the node at pos-1 by updating the pointers.
- Return the updated start.

Delete at Beginning (deleteAtBeg):

- Input: start (head of the list).
- If the list is empty (start == NULL), print an error message and return.
- Store the current start in a temporary pointer.
- Set start to the next node (start->next).
- Free the memory of the temporary node.
- Return the updated start.

Delete at End (deleteAtEnd):

- Input: start (head of the list).
- If the list is empty (start == NULL), print an error message and return.
- If the list has only one node (start->next == NULL), free the node and set start to NULL.
- Otherwise, traverse the list to find the second last node.
- Set the second last node's next to NULL and free the last node.
- Return the updated start.

Delete at Position (deleteAtPos):

- Input: start (head of the list), pos (position of the node to delete).
- If the list is empty (start == NULL), print an error message and return.
- If the position is 0, call deleteAtBeg to delete the first node and return.
- Traverse the list to find the node at position pos.
- If the node is NULL, print an error message and return.
- Update the next pointer of the previous node to skip the node to be deleted.
- Free the memory of the deleted node.
- Return the updated start.

Search Operation:

- Input: start (head of the list), data (data to search).
- Traverse the list, checking each node's data.
- If the data is found, print the position and return.

- If the end of the list is reached without finding the data, print an error message.

Display the List:

- Input: start (head of the list).
- Traverse the list, printing each node's data.
- End the display with NULL to indicate the end of the list.

Main Function:

- Initialize start as NULL.
- Provide a menu of operations:
    - 1: Insert at beginning.
    - 2: Insert at end.
    - 3: Insert at a specific position.
    - 4: Delete the first node.
    - 5: Delete the last node.
    - 6: Delete at a specific position.
    - 7: Search for an element.
- Execute the corresponding function based on user input.
- Display the updated list after each operation.
- Allow the user to continue or exit based on input.

Program:

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

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

struct node* insertAtBeg(struct node* start, int data) {
   struct node* temp = (struct node*)malloc(sizeof(struct node));
   temp->data = data;
   temp->next = start;
```

```
    start = temp;
    return start;
}

struct node* insertAtEnd(struct node* start, int data) {
    struct node* temp = (struct node*)malloc(sizeof(struct node));
    struct node *p = start;
    while(p->next != NULL) {
        p = p->next;
    }
    temp->data = data;
    p->next = temp;
    temp->next = NULL;
    return start;
}

struct node* insertAtPos(struct node* start, int data, int pos) {
    struct node* temp = (struct node*)malloc(sizeof(struct node));
    if(pos == 0) {
        start = insertAtBeg(start, data);
        return start;
    }
    struct node *p = start;
    for(int i=0; i<pos-1 && p != NULL; i++) {
        p = p->next;
    }
    if(p == NULL) {
        printf("The list is not big enough to fit the particular data at the provided position.\n\n");
        return start;
    }
    temp->data = data;
    temp->next = p->next;
    p->next = temp;
    return start;
}

struct node* deleteAtBeg(struct node* start) {
    if(start == NULL) {
        printf("The given list is empty. No element can be deleted.\n\n");
        return start;
    }
    struct node* temp = start;
    start = start->next;
    free(temp);
    return start;
}

struct node* deleteAtEnd(struct node* start) {
    if(start == NULL) {
        printf("The given list is empty. No element can be deleted.\n\n");
        return start;
    }
    if(start->next == NULL) {
        free(start);
        start = NULL;
        return start;
    }
    struct node* p = start;
    while(p->next->next != NULL) {
        p = p->next;
    }
    free(p->next);
    p->next = NULL;
    return start;
```

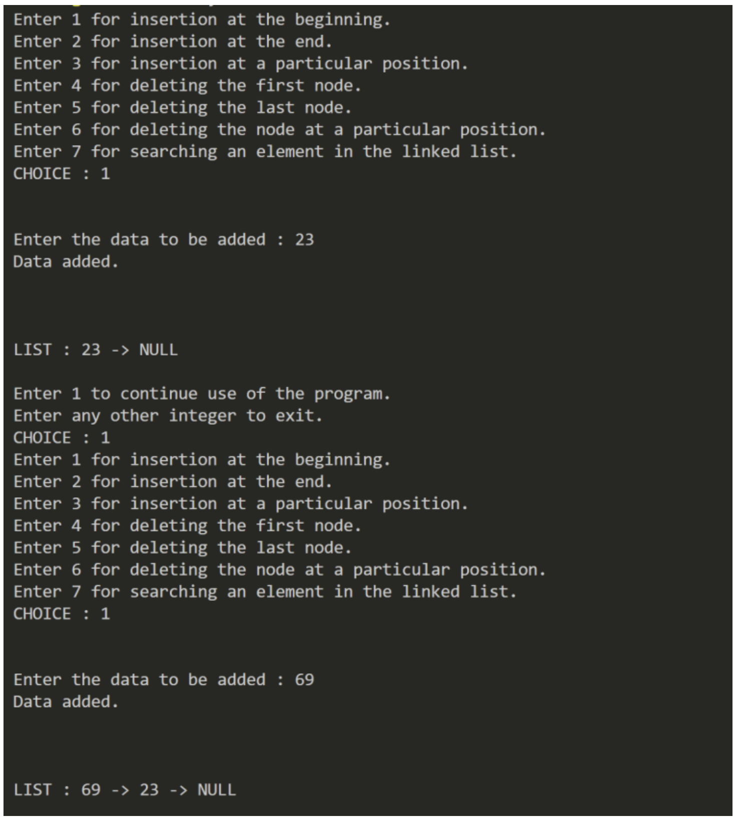
```
}

struct node* deleteAtPos(struct node* start, int pos) {
    if(start == NULL) {
        printf("The linked list is empty.\n\n");
        return start;
    }
    struct node* p = start;
    struct node* prev = NULL;
    if(pos == 0) {
        start = deleteAtBeg(start);
        return start;
    }
    for(int i=0; i<pos && p != NULL; i++) {
        prev = p;
        p = p->next;
    }
    if(p == NULL) {
        printf("The linked list isn't long enough.\n\n");
        return start;
    }
    prev->next = p->next;
    free(p);
    return start;
}

void search(struct node* start, int data) {
    struct node* p = start;
    int counter = 0;
    while(p != NULL) {
        counter++;
        if(p->data == data) {
            printf("The provided data exists in the linked list at %d position.\n\n", counter);
            return;
        }
        p = p->next;
    }
    printf("The given data does not exist in the linked list.\n\n");
    return;
}
void display(struct node* start) {
    struct node* p = start;
    printf("\n\nLIST : ");
    while(p != NULL) {
        printf("%d -> ", p->data);
        p = p->next;
    }
    printf("NULL\n\n");
    return;
}


int main() {
    struct node* start = NULL;
    int choice = 1;
    do {
        int x;
        printf("Enter 1 for insertion at the beginning.\nEnter 2 for insertion at the end.\nEnter 3 for insertion
at a particular position.\nEnter 4 for deleting the first node.\nEnter 5 for deleting the last node.\nEnter 6
for deleting the node at a particular position.\nEnter 7 for searching an element in the linked list.\nCHOICE
: ");
        scanf("%d", &x);
        printf("\n\n");


        switch(x) {
```

```
case 1 : {
   int data;
   printf("Enter the data to be added : ");
   scanf("%d", &data);
   start = insertAtBeg(start, data);
   printf("Data added.\n\n");
   break;
}

case 2 : {
   int data;
   printf("Enter the data to be added : ");
   scanf("%d", &data);
   start = insertAtEnd(start, data);
   printf("Data added.\n\n");
   break;
}

case 3 : {
   int data, pos;
   printf("Enter the data to be added : ");
   scanf("%d", &data);
   printf("Enter the position for the data to be added : ");
   scanf("%d", &pos);
   start = insertAtPos(start, data, pos-1);
   printf("Data added.\n\n");
   break;
}

case 4 : {
   start = deleteAtBeg(start);
   printf("Data deleted.\n\n");
```

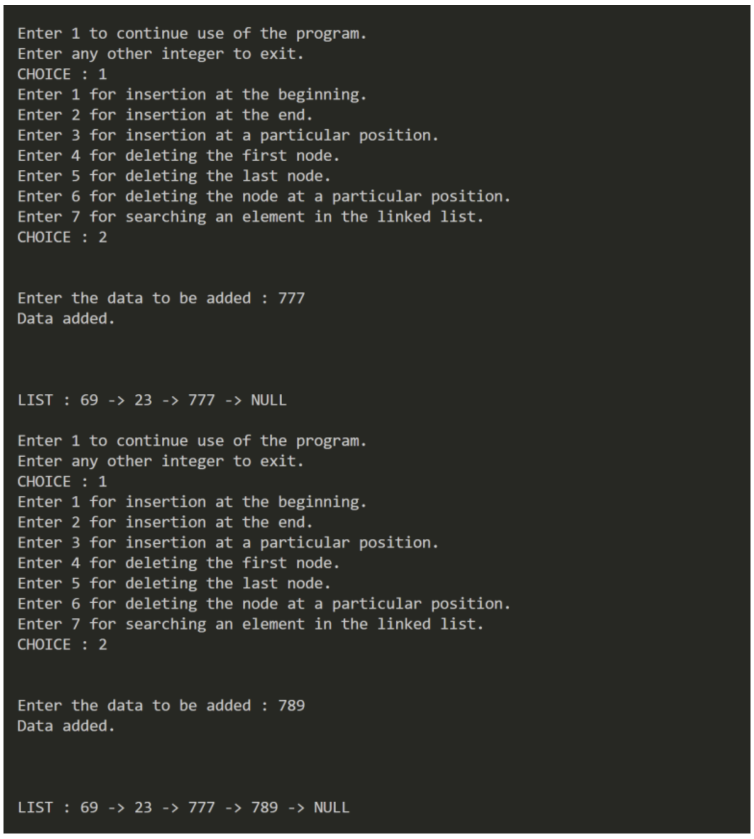
```
        break;
    }

    case 5 : {
        start = deleteAtEnd(start);
        printf("Data deleted.\n\n");
        break;
    }

    case 6 : {
        int pos;
        printf("Enter the position for the data to be deleted : ");
        scanf("%d", &pos);
        start = deleteAtPos(start, pos-1);
        printf("Data deleted.\n\n");
        break;
    }

    case 7 : {
        int data;
        printf("Enter the data to be searched for : ");
        scanf("%d", &data);
        search(start, data);
        break;
    }

    default : {
        printf("Invalid choice.\n\n");
    }
}
display(start);
printf("Enter 1 to continue use of the program.\nEnter any other integer to exit.\nCHOICE : ");
```

```
        scanf("%d", &choice);

    } while(choice == 1);

    return 0;

}
```

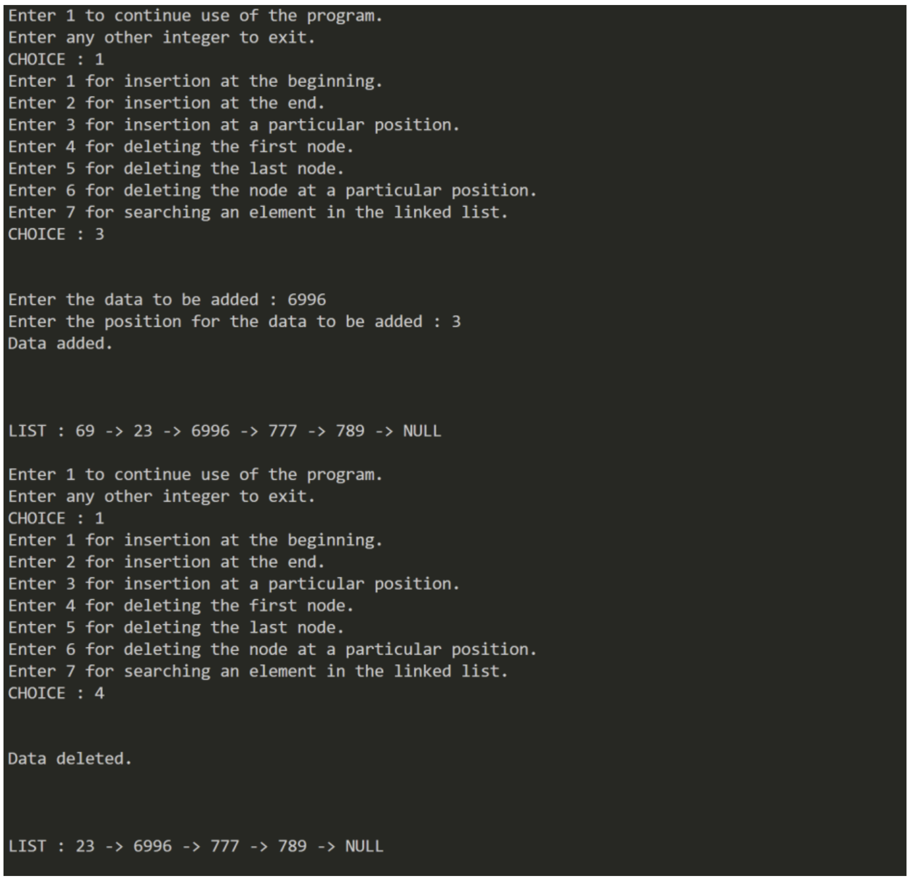
Output:



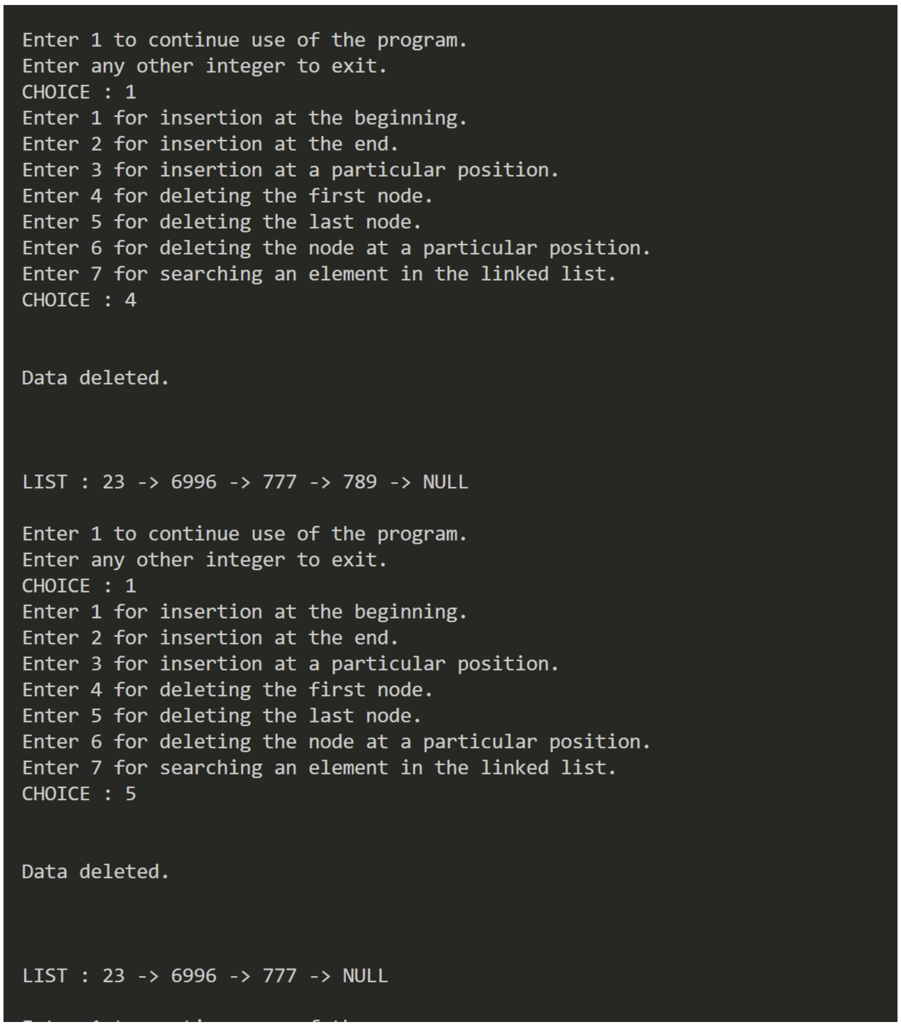
```
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 1


Enter the data to be added : 23
Data added.



LIST : 23 -> NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 1


Enter the data to be added : 69
Data added.



LIST : 69 -> 23 -> NULL
```



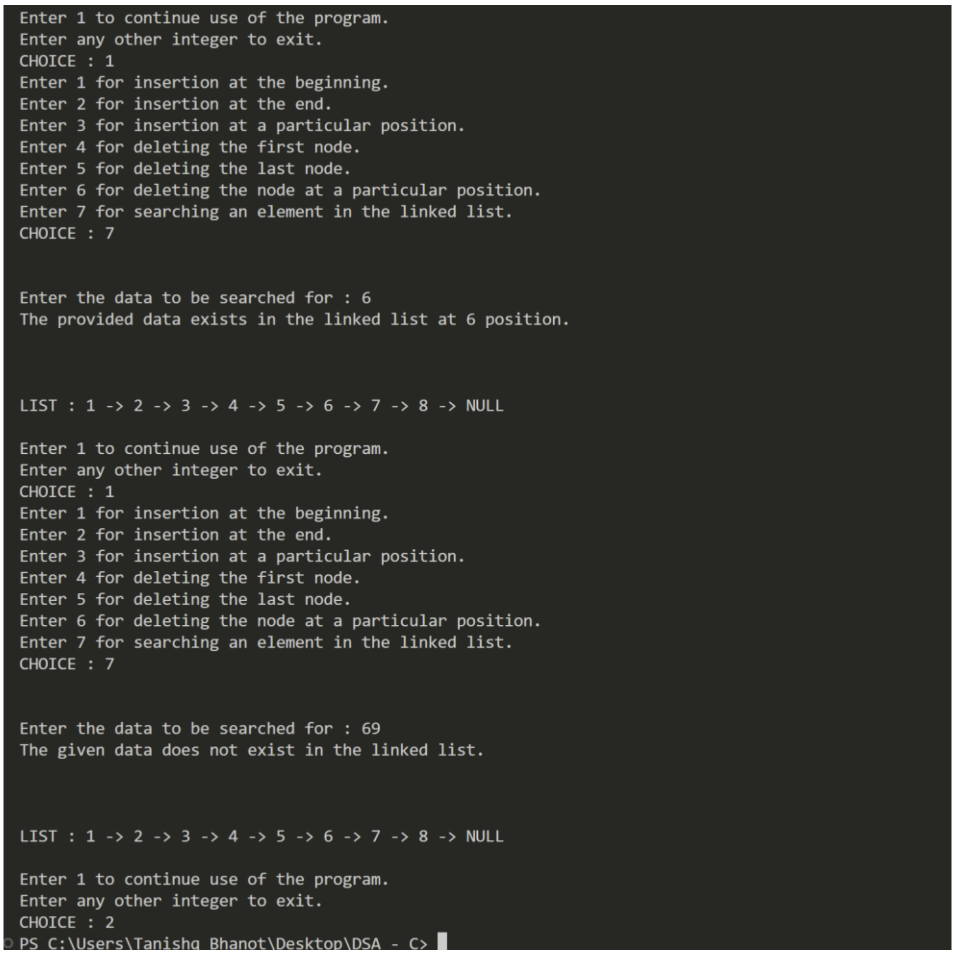
```
Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 2


Enter the data to be added : 777
Data added.



LIST : 69 -> 23 -> 777 -> NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 2


Enter the data to be added : 789
Data added.



LIST : 69 -> 23 -> 777 -> 789 -> NULL
```



```
Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 3


Enter the data to be added : 6996
Enter the position for the data to be added : 3
Data added.



LIST : 69 -> 23 -> 6996 -> 777 -> 789 -> NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 4


Data deleted.



LIST : 23 -> 6996 -> 777 -> 789 -> NULL
```



```
Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 4


Data deleted.



LIST : 23 -> 6996 -> 777 -> 789 -> NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 5


Data deleted.



LIST : 23 -> 6996 -> 777 -> NULL
```



```
Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 7


Enter the data to be searched for : 6
The provided data exists in the linked list at 6 position.



LIST : 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 7


Enter the data to be searched for : 69
The given data does not exist in the linked list.



LIST : 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 2
PS C:\Users\Tanishq Bhanot\Desktop\DSA - C>
```

Q5. Write a menu driven program to implement the following operations on Doubly linked list:

a. Insertion()

i. Beginning

ii. End

iii. At a given position

b. Deletion()

i. Beginning

ii. End

iii. At a given position

c. Search(): search for the given element on the list

Algorithm:

Define a Node:

- A node structure with three fields:
  - data: Stores the value of the node.

- prev: Points to the previous node in the list.
- next: Points to the next node in the list.

Create a New Node (createNewNode):

- Allocate memory for a new node.
- Set the node's data, prev, and next to the provided values (prev = NULL, next = NULL).
- Return the new node.

Insert at Beginning (insertAtBeg):

- Input: start (head of the list), tail (tail of the list), data (data to insert).
- Allocate memory for a new node.
- If the list is empty (start == NULL), set both start and tail to the new node.
- Otherwise, set the new node's next to start, and update the prev pointer of the current start to point to the new node.
- Set start to the new node.

Insert at End (insertAtEnd):

- Input: start (head of the list), tail (tail of the list), data (data to insert).
- Allocate memory for a new node.
- If the list is empty (start == NULL), set both start and tail to the new node.
- Otherwise, set the current tail's next to the new node and the new node's prev to the current tail.
- Set tail to the new node.

Insert at Position (insertAtPos):

- Input: start (head of the list), tail (tail of the list), data (data to insert), pos (position).
- If the position is 0, call insertAtBeg to insert at the beginning and return.
- Traverse the list until reaching the node at position pos-1.
- If the node is NULL, print an error and free the new node.
- Insert the new node after the node at position pos-1, updating the prev and next pointers of adjacent nodes.

Delete at Beginning (deleteAtBeg):

- Input: start (head of the list), tail (tail of the list).
- If the list is empty (start == NULL), print an error and return.
- Otherwise, set start to the next node, and update the prev pointer of the new start to NULL.
- If the list becomes empty (start == NULL), set tail to NULL.
- Free the old start node.

Delete at End (deleteAtEnd):

- Input: start (head of the list), tail (tail of the list).
- If the list is empty (tail == NULL), print an error and return.
- If the list has only one node (start == tail), set both start and tail to NULL.
- Otherwise, set tail to the previous node and set its next pointer to NULL.
- Free the old tail node.

Delete at Position (deleteAtPos):

- Input: start (head of the list), tail (tail of the list), pos (position).
- If the list is empty (start == NULL), print an error and return.
- If the position is 0, call deleteAtBeg to delete the first node and return.
- Traverse the list until reaching the node at position pos.
- If the node is NULL, print an error.
- If the node is the first (start), delete it using deleteAtBeg.
- If the node is the last (tail), delete it using deleteAtEnd.
- Otherwise, update the prev and next pointers of adjacent nodes and free the current node.

Search Operation (search):

- Input: start (head of the list), data (data to search).
- Traverse the list, checking each node's data.
- If the data is found, print the position and return.
- If the end of the list is reached without finding the data, print an error.

Display the List (display):

- Input: start (head of the list).
- Traverse the list and print each node's data from start to tail.
- Also display the list in reverse order, starting from tail to start.

Main Function:

- Initialize start and tail as NULL.
- Provide a menu with options:
    - 1: Insert at beginning.
    - 2: Insert at end.
    - 3: Insert at a specific position.
    - 4: Delete the first node.
    - 5: Delete the last node.
    - 6: Delete at a specific position.

    - 7: Search for an element.
- Execute the corresponding function based on user input.
- Display the updated list after each operation.
- Allow the user to continue or exit based on input.

Program:

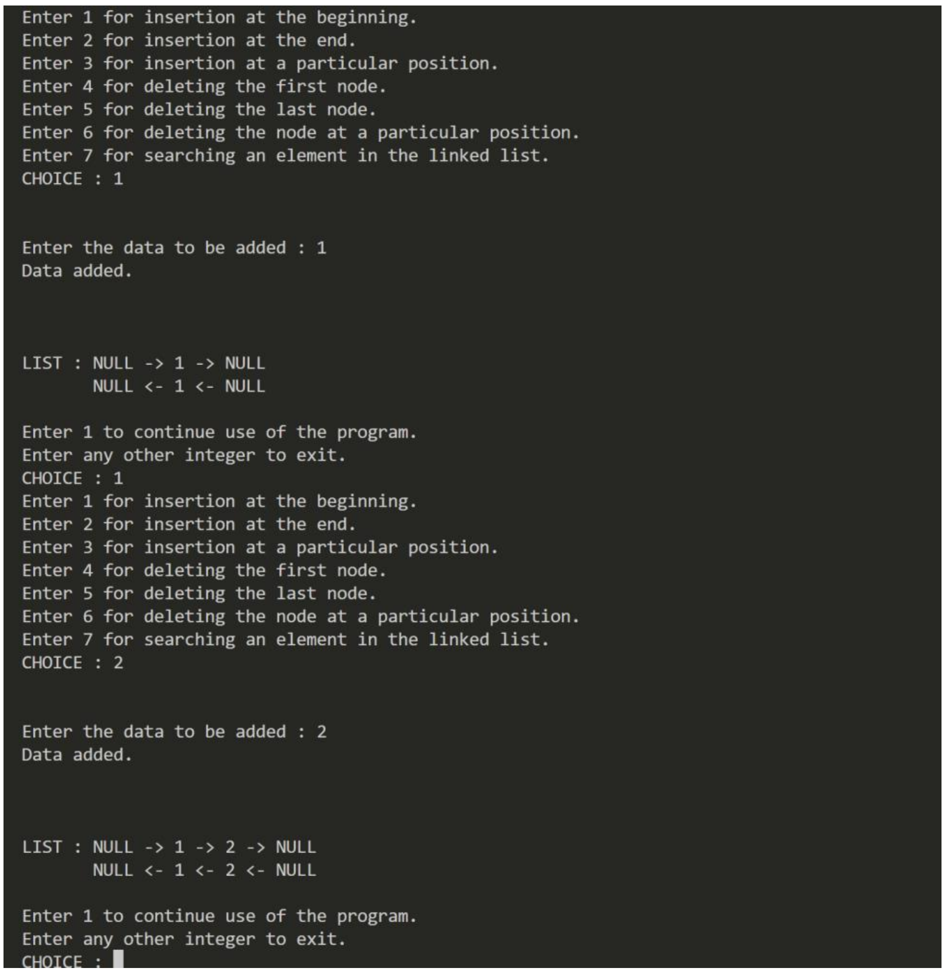
```
# include <stdio.h>
# include <stdlib.h>

struct node {
   struct node* prev;
   int data;
   struct node* next;
};

struct node* createNewNode(int data) {
   struct node* newNode = (struct node*)malloc(sizeof(struct node));
   newNode->data = data;
   newNode->prev = NULL;
   newNode->next = NULL;
   return newNode;
}

void insertAtBeg(struct node** start, struct node** tail, int data) {
   struct node *newNode = createNewNode(data);
   if(*start == NULL) {
      *start = newNode;
      *tail = newNode;
      return;
   }
   newNode->next = *start;
   (*start)->prev = newNode;
```

```
    *start = newNode;
    return;
}

void insertAtEnd(struct node** start, struct node** tail, int data) {
    struct node *newNode = createNewNode(data);
    if(*start == NULL) {
        *start = newNode;
        *tail = newNode;
        return;
    }
    (*tail)->next = newNode;
    newNode->prev = *tail;
    *tail = newNode;
    return;
}

void insertAtPos(struct node** start, struct node** tail, int data, int pos) {
    struct node *newNode = createNewNode(data);
    if(pos == 0) {
        insertAtBeg(start, tail, data);
        return;
    }
    struct node* p = *start;
    for(int i=0; i<pos-1 && p != NULL; i++) {
        p = p->next;
    }
    if(p == NULL) {
        printf("There is not enough spaces in the linked list.\n\n");
        free(newNode);
        return;
    }
    newNode->next = p->next;
    newNode->prev = p;
    p->next->prev = newNode;
    p->next = newNode;
    return;
}

void deleteAtBeg(struct node** start, struct node** tail) {
    if (*start == NULL) {
        printf("List is empty. Nothing to delete.\n\n");
        return;
    }
    struct node* temp = *start;
    *start = (*start)->next;
    if (*start != NULL) {
        (*start)->prev = NULL;
    }
    else {
        *tail = NULL;
    }
    free(temp);
    printf("First node deleted.\n\n");
}

void deleteAtEnd(struct node** start, struct node** tail) {
    if (*tail == NULL) {
        printf("List is empty. Nothing to delete.\n\n");
        return;
    }
    struct node* temp = *tail;
    if (*tail == *start) {
        *start = NULL;
        *tail = NULL;
    } else {
        *tail = (*tail)->prev;
        (*tail)->next = NULL;
    }
    free(temp);
    printf("Last node deleted.\n\n");
}

void deleteAtPos(struct node** start, struct node** tail, int pos) {
    if (*start == NULL) {
        printf("List is empty. Nothing to delete.\n");
        return;
    }
    struct node* temp = *start;
    int curr = 1;
    while (temp != NULL && curr < pos) {
        temp = temp->next;
        curr++;
    }
    if (temp == NULL) {
        printf("Invalid position %d. No node found.\n", pos);
        return;
    }
    if (temp == *start) {
        deleteAtBeg(start, tail);
    }
    else if (temp == *tail) {
        deleteAtEnd(start, tail);
    }
    else {
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
    }

    free(temp);
    printf("Node at position %d deleted.\n", pos);
}

void search(struct node* start, int data) {
    struct node* p = start;
    int counter = 0;
    while(p != NULL) {
        counter++;
        if(p->data == data) {
            printf("The provided data exists in the linked list at %d position.\n\n", counter);
            return;
        }
        p = p->next;
    }
    printf("The given data does not exist in the linked list.\n\n");
    return;
}

void display(struct node* start) {
    struct node* p = start;
    printf("\n\nLIST : NULL -> ");
    while(p != NULL) {
        printf("%d -> ", p->data);
        p = p->next;
    }
    printf("NULL\n");
    p = start;
    printf("       NULL <- ");
    while(p != NULL) {

        printf("%d <- ", p->data);

        p = p->next;

    }

    printf("NULL\n\n");

    return;

}


int main() {

    struct node *start = NULL;

    struct node *tail = NULL;

    int choice = 1;

    do {

        int x;

        printf("Enter 1 for insertion at the beginning.\nEnter 2 for insertion at the end.\nEnter 3 for insertion
at a particular position.\nEnter 4 for deleting the first node.\nEnter 5 for deleting the last node.\nEnter 6
for deleting the node at a particular position.\nEnter 7 for searching an element in the linked list.\nCHOICE
: ");

        scanf("%d", &x);

        printf("\n\n");


        switch(x) {


            case 1 : {

                int data;

                printf("Enter the data to be added : ");

                scanf("%d", &data);

                insertAtBeg(&start, &tail, data);

                printf("Data added.\n\n");

                break;

            }


            case 2 : {
```

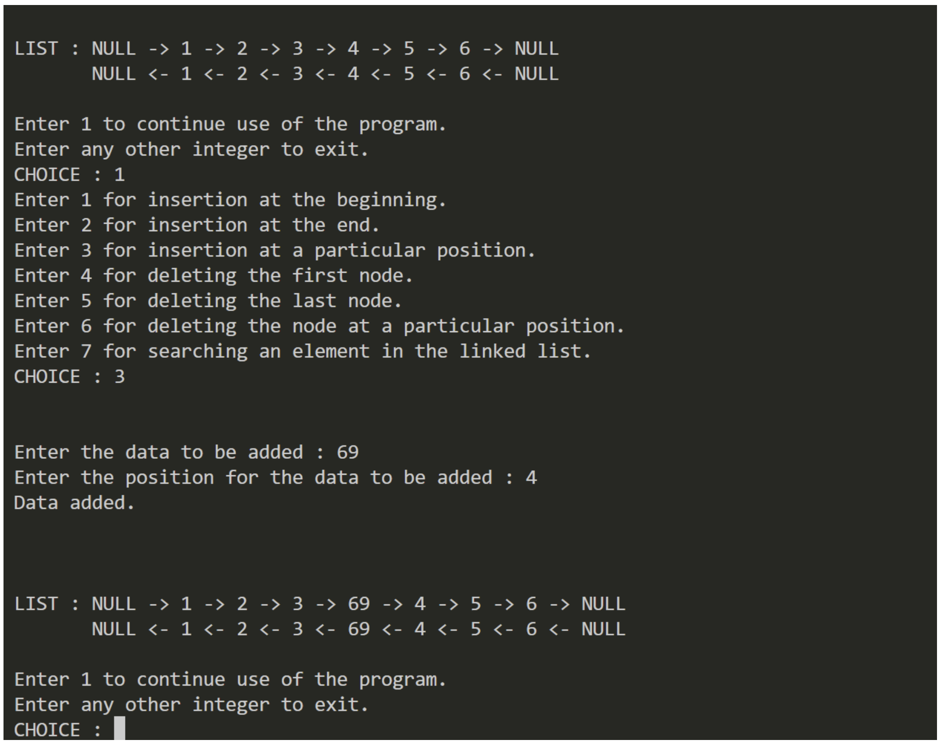
```
   int data;
   printf("Enter the data to be added : ");
   scanf("%d", &data);
   insertAtEnd(&start, &tail, data);
   printf("Data added.\n\n");
   break;
}

case 3 : {
   int data, pos;
   printf("Enter the data to be added : ");
   scanf("%d", &data);
   printf("Enter the position for the data to be added : ");
   scanf("%d", &pos);
   insertAtPos(&start, &tail, data, pos-1);
   printf("Data added.\n\n");
   break;
}

case 4 : {
   deleteAtBeg(&start, &tail);
   printf("Data deleted.\n\n");
   break;
}

case 5 : {
   deleteAtEnd(&start, &tail);
   printf("Data deleted.\n\n");
   break;
}

case 6 : {
```

```
            int pos;
            printf("Enter the position for the data to be deleted : ");
            scanf("%d", &pos);
            deleteAtPos(&start, &tail, pos);
            printf("Data deleted.\n\n");
            break;
        }

        case 7 : {
            int data;
            printf("Enter the data to be searched for : ");
            scanf("%d", &data);
            search(start, data);
            break;
        }

        default : {
            printf("Invalid choice.\n\n");
        }
      }
      display(start);
      printf("Enter 1 to continue use of the program.\nEnter any other integer to exit.\nCHOICE : ");
      scanf("%d", &choice);
    } while(choice == 1);
    return 0;
}
```

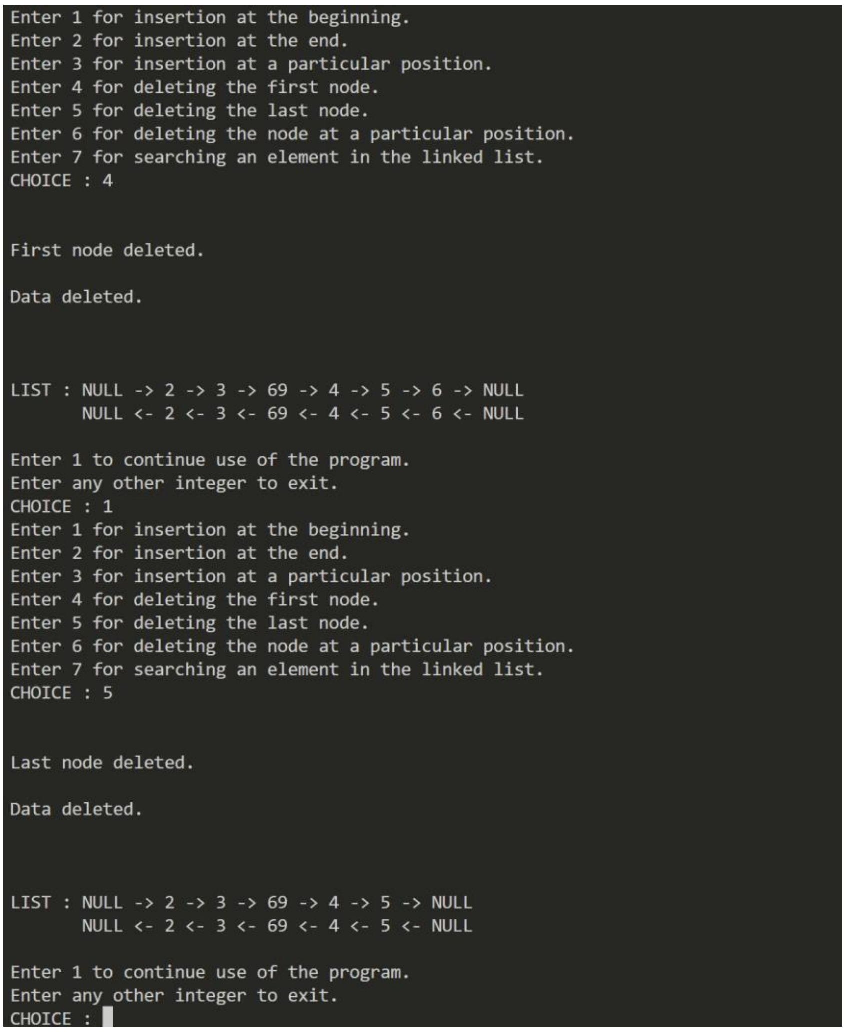
Output:



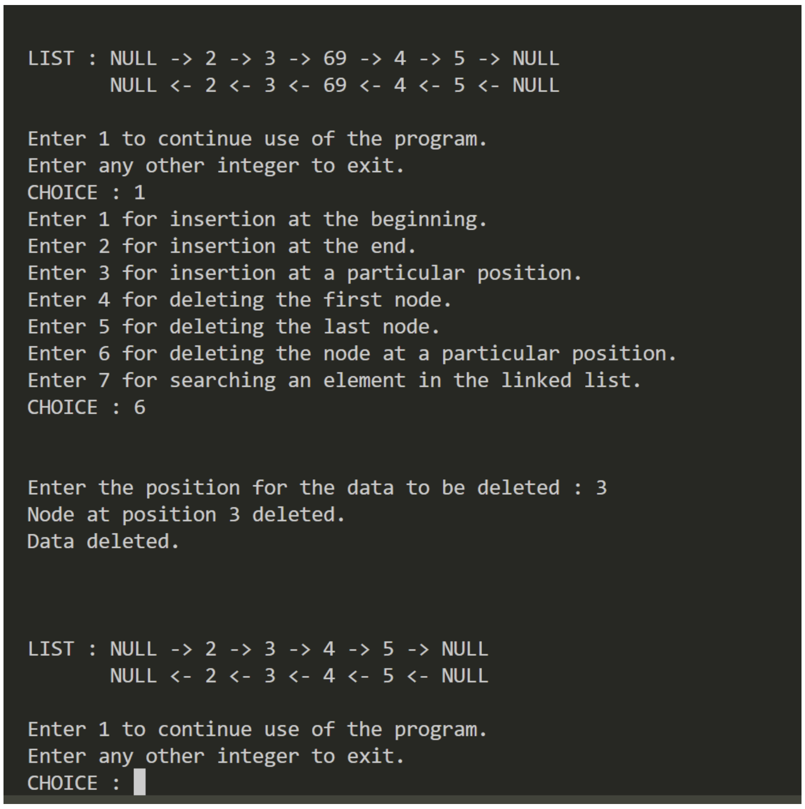
```
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 1


Enter the data to be added : 1
Data added.



LIST : NULL -> 1 -> NULL
       NULL <- 1 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 2


Enter the data to be added : 2
Data added.



LIST : NULL -> 1 -> 2 -> NULL
       NULL <- 1 <- 2 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE :
```



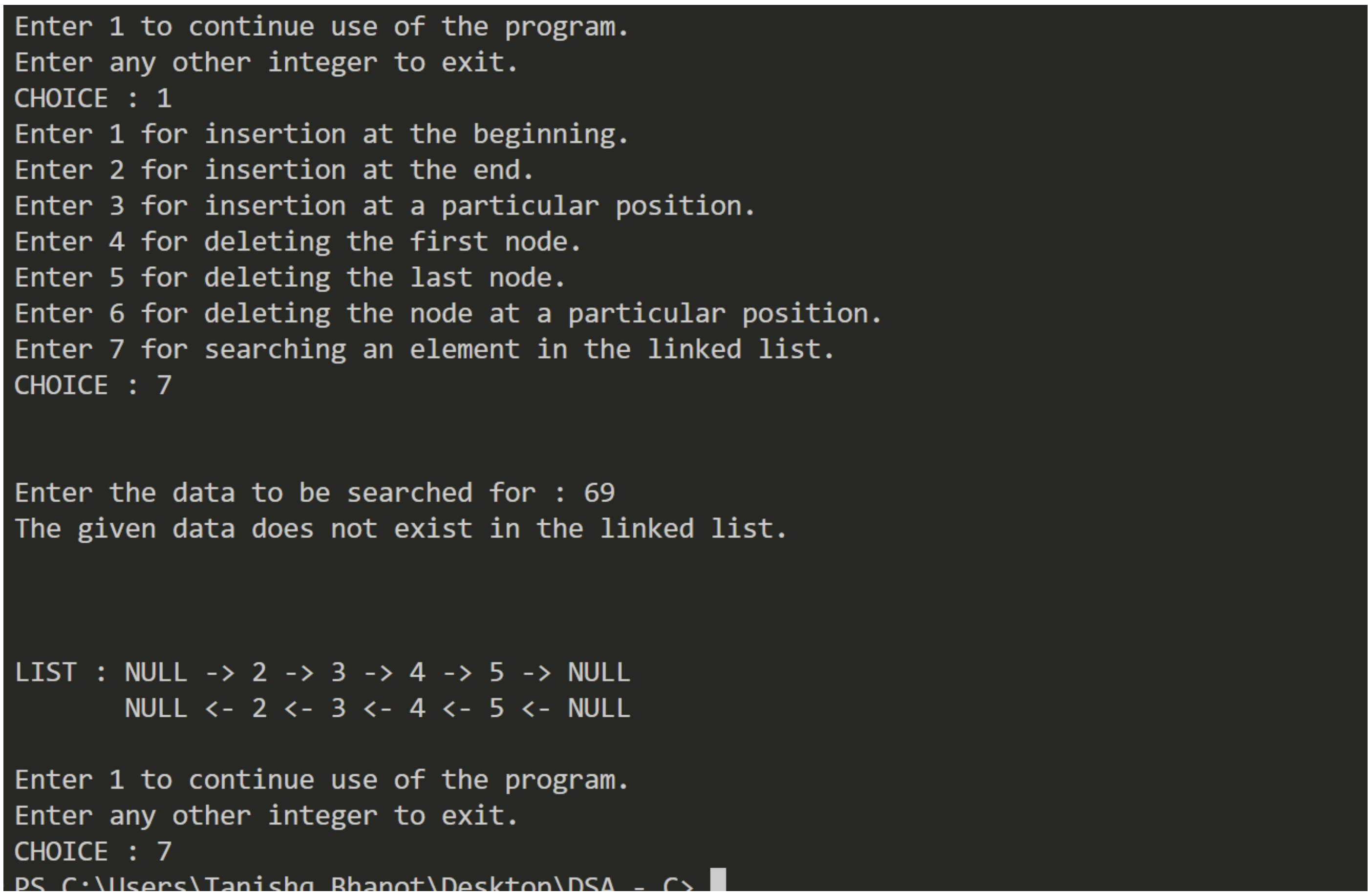
```
LIST : NULL -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL
       NULL <- 1 <- 2 <- 3 <- 4 <- 5 <- 6 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 3


Enter the data to be added : 69
Enter the position for the data to be added : 4
Data added.



LIST : NULL -> 1 -> 2 -> 3 -> 69 -> 4 -> 5 -> 6 -> NULL
       NULL <- 1 <- 2 <- 3 <- 69 <- 4 <- 5 <- 6 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE :
```



```
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 4


First node deleted.

Data deleted.



LIST : NULL -> 2 -> 3 -> 69 -> 4 -> 5 -> 6 -> NULL
       NULL <- 2 <- 3 <- 69 <- 4 <- 5 <- 6 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 5


Last node deleted.

Data deleted.



LIST : NULL -> 2 -> 3 -> 69 -> 4 -> 5 -> NULL
       NULL <- 2 <- 3 <- 69 <- 4 <- 5 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE :
```



```
LIST : NULL -> 2 -> 3 -> 69 -> 4 -> 5 -> NULL
       NULL <- 2 <- 3 <- 69 <- 4 <- 5 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 6


Enter the position for the data to be deleted : 3
Node at position 3 deleted.
Data deleted.



LIST : NULL -> 2 -> 3 -> 4 -> 5 -> NULL
       NULL <- 2 <- 3 <- 4 <- 5 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE :
```



```
Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 7


Enter the data to be searched for : 69
The given data does not exist in the linked list.




LIST : NULL -> 2 -> 3 -> 4 -> 5 -> NULL
       NULL <- 2 <- 3 <- 4 <- 5 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 7
PS C:\Users\Tanishq Bhanot\Desktop\DSA - C>
```