Data Structures and Algorithms - CBS1003

Assessment - I

SAKSHAM DUBEY - 24BBS0081

Question 1

Algorithm

PUSH( )
- if(‘top’ == ‘MAX’ - 1)
  - Print “stack overflow”
- else
  - Input ‘num’
  - ‘top’ = ‘top’ + 1
  - Insert ‘num’ at ‘top’ index in stack

POP( )
- if (‘top’ == -1)
  - print “Stack Underflow”
- Else
  - print element at index ‘top’ of Stack
  - ‘top’ = ‘top’ - 1

Display( )
- if (‘top’ == -1)
  - print “stack empty”
- Else
  - For i from ‘top’ to 0
    - Print stack[i]

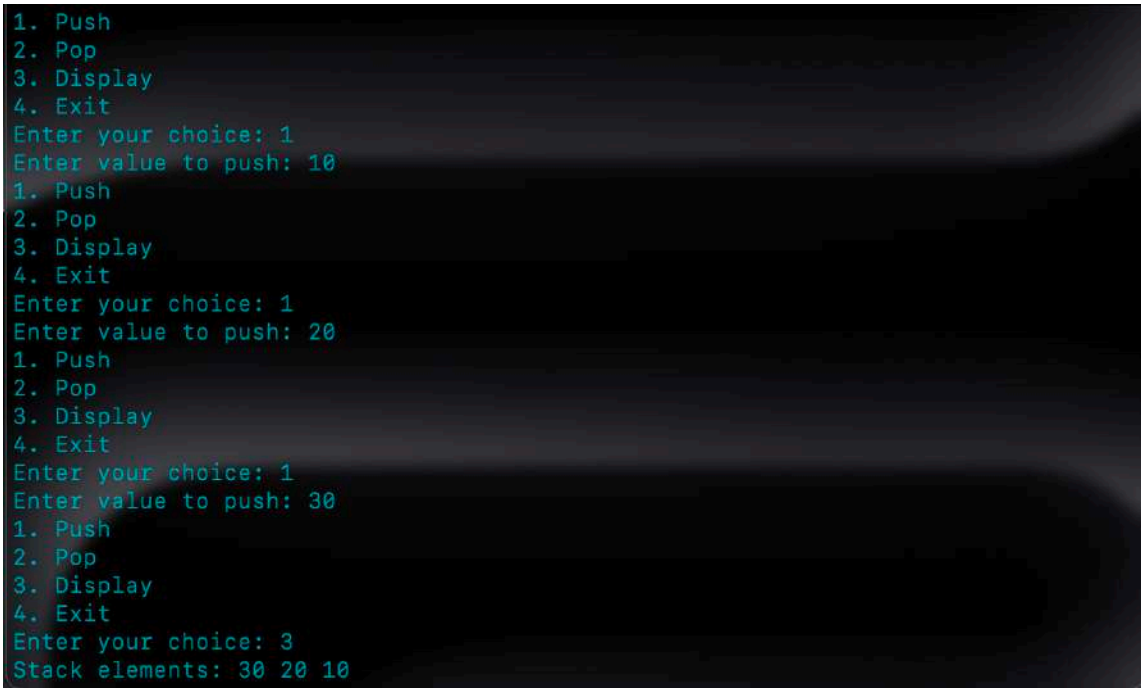
C Program

```
#include <stdio.h>
#define MAX 5
int stack[MAX], top = -1;
void push() {
    int value;
    if (top == MAX - 1)
        printf("Stack Overflow\n");
    else {
        printf("Enter value to push: ");
        scanf("%d", &value);
        stack[++top] = value;
    }
}
void pop() {
    if (top == -1)
        printf("Stack Underflow\n");
    else
        printf("Popped: %d\n", stack[top--]);
}
void display() {
    if (top == -1)
        printf("Stack Empty\n");
    else {
        printf("Stack elements: ");
        for (int i = top; i >= 0; i--)
            printf("%d ", stack[i]);
```

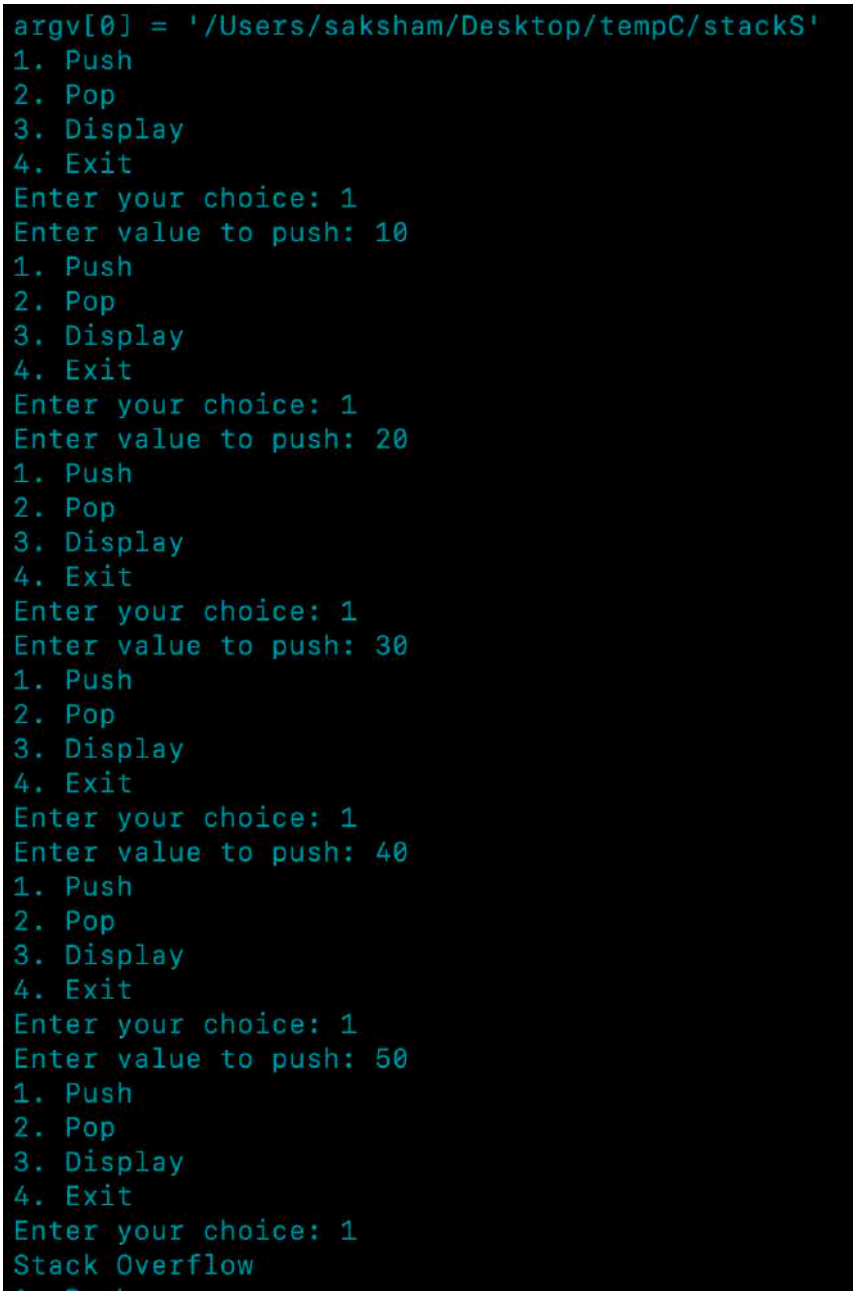
```
        printf("\n");
    }
}
int main() {
    int c;
    do {
        printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &c);
        switch (c) {
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Thank You\n");
                break;
        }
    } while (c != 4);
}
```

Test Case:

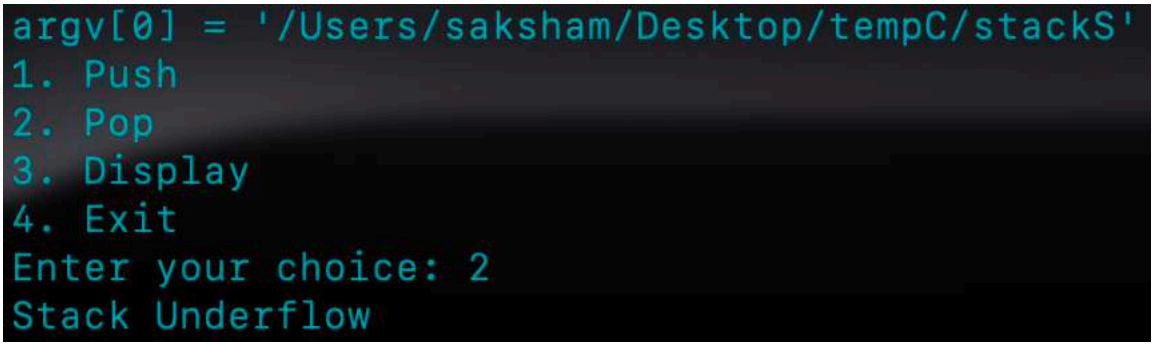
General Condition:



Boundary Condition: Stack Overflow



Boundary Condition: Stack Underflow



Question 2

Algorithm

Enqueue( )
- Check if ‘rear’ is equal to ‘MAX’ - 1
- If True, print “Queue Overflow”
- Else, check if front is equal to -1
- If true, increment front by 1
- Input an element
- Increment ‘rear’
- Insert element at queue of index ‘rear’

Dequeue( )
- Check if ‘front’ is equal to -1 or front is greater than rear
- If True, print “Queue Underflow”
- Else, print element in queue at index ‘front’
- Increment ‘front’ by 1

Display( )
- Check if ‘front’ is equal to -1 or front is greater than rear
- If true, print “Queue empty”
- Else, traverse from index front to rear in queue and print elements

C Program

```
#include <stdio.h>
#define MAX 5

int queue[MAX], front = -1, rear = -1;

void enqueue() {
    int value;
    if (rear == MAX - 1)
        printf("Queue Overflow\n");
    else {
        if (front == -1){
            ++front;
        }
        printf("Enter value to enqueue: ");
        scanf("%d", &value);
        queue[++rear] = value;
    }
}

void dequeue() {
    if (front == -1 || front > rear){
        printf("Queue Underflow\n");
    }
    else{
        printf("Dequeued: %d\n", queue[front++]);
    }
}
```

```
void display() {
    if (front == -1 || front > rear){
        printf("Queue is Empty\n");
    }
    else {
        printf("Queue elements: ");
        for (int i = front; i <= rear; i++)
            printf("%d ", queue[i]);
        printf("\n");
    }
}

int main() {
    int c;
    do {
        printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &c);
        switch (c) {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Thank you\n");
                break;
        }
    } while (c != 4);
}
```

Test Case:

General condition

```
argv[0] = '/Users/saksham/Desktop/tempC/queueQ'
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 10
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued: 10
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue is Empty
```

Boundary condition: queue overflow

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 30
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 40
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 50
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Queue Overflow
```

Boundary condition: queue underflow

Question 3

Algorithm

Enqueue( )
- Check if (‘rear’ + 1) % ‘MAX’ is equal to front
- If true, print “Queue overflow”
- Else, input element
- Assign rear as ('rear’ + 1) % ‘MAX’
- Insert element at index ‘rear’ of queue

Dequeue( )
- Check if ‘front’ is equal to -1
- If true, print “Queue underflow”
- Else, print element at index ‘front’ of queue
- Check if ‘front’ is equal to ‘rear’
- If true, assign -1 to ‘front’ and ‘rear’
- Else, assign ‘front’ as (‘front’ + 1) % ‘MAX’

Display( )
- Check if ‘front’ is equal to -1
- If true, print “Queue empty”
- Else, assign ‘i’ as ‘front’
- While i is not equal to rear
  - Print element at index ‘I’
  - Assign i as (‘i’ + 1) % ‘MAX’
- Print element at index ‘rear’

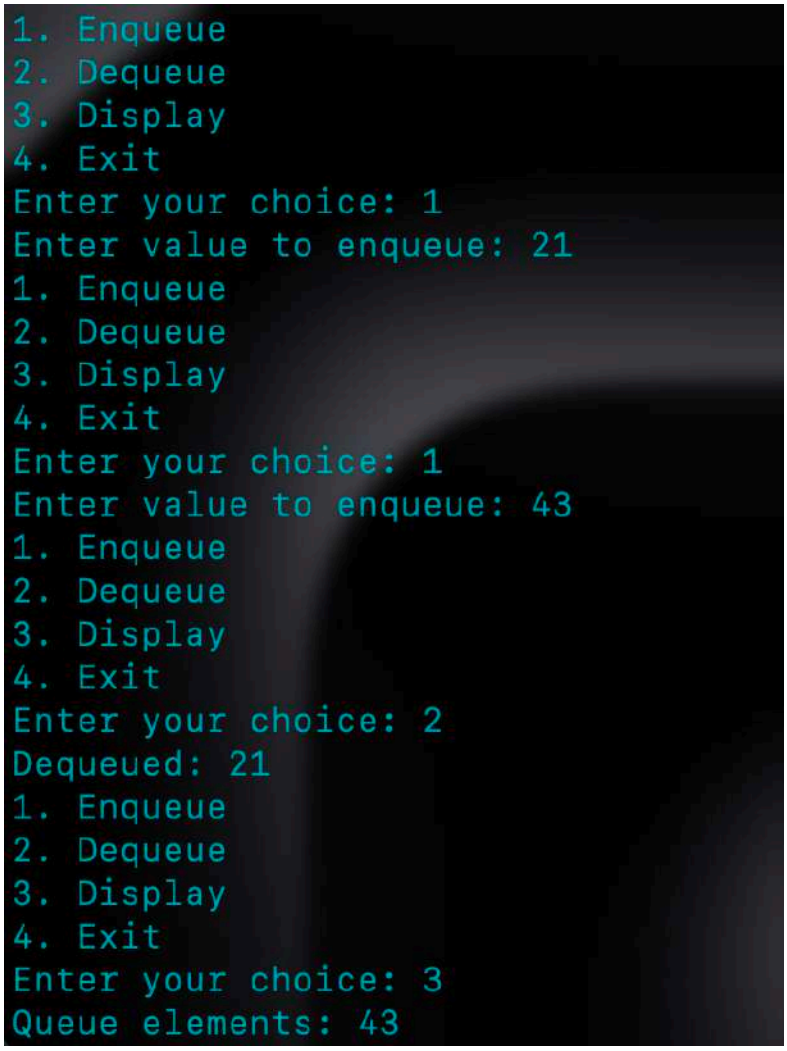
C Program

```
#include <stdio.h>
#define MAX 5
int queue[MAX], front = -1, rear = -1;
void enqueue() {
    int value;
    if ((rear + 1) % MAX == front)
        printf("Queue Overflow\n");
    else {
        if (front == -1) front = 0;
        printf("Enter value to enqueue: ");
        scanf("%d", &value);
        rear = (rear + 1) % MAX;
        queue[rear] = value;
    }
}
void dequeue() {
    if (front == -1)
        printf("Queue Underflow\n");
    else {
        printf("Dequeued: %d\n", queue[front]);
        if (front == rear) {
            front = rear = -1;
        } else {
            front = (front + 1) % MAX;
```

```
        }
    }
}
void display() {
    if (front == -1)
        printf("Queue is Empty\n");
    else {
        int i = front;
        printf("Queue elements: ");
        while (i != rear) {
            printf("%d ", queue[i]);
            i = (i + 1) % MAX;
        }
        printf("%d\n", queue[rear]);
    }
}
int main() {
    int c;
    do {
        printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &c);
        switch (c) {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Thank you\n");
                break;
        }
    } while (c != 4);
}
```

Test Case:

General condition



Boundary Conditions: Queue Overflow

Boundary Conditions: Queue Underflow

Question 4

Algorithm

1) Insertion

insertAtBeginning(int num)

- struct node* NewNode = createNode(num)
- Set NewNode -> next = ‘head’
- ‘head’ = NewNode

insertAtEnd(int num)

- struct node* NewNode = createNode(num)
- struct node* temp = head
- while(temp -> next != NULL)
  - Set temp = temp->next
- Set temp->next = NewNode

insertAtPosition(int num, int position)

- if(position == 0)
  - Call insertAtBeginning(num)
- Else
  - struct Node* NewNode = createNode(num)
  - struct Node* temp = head
  - For i from 0 to position - 1 && temp !=NULL
    - temp=temp->next
  - if(temp==NULL)
    - Print “Invalid position”
  - Else
    - Set NewNode->next = temp->next
    - Temp->next = NewNode

2) Deletion

deleteAtBeginning( )

- if(head == NULL)
  - Print “List Empty”
- Else
  - Struct Node* temp = head
  - Set head = head -> next
  - free(temp)

deleteAtEnd( )

- if(head == NULL)
  - Print “list empty”
- Else
  - Struct Node* temp = head
  - while(temp->next->next != NULL)
    - temp = temp->next
  - free(temp->next)
  - temp->next = NULL

deleteAtPosition(int position)

- if(position == 0)
  - Call deleteAtBeginning( )
- Else
  - Struct Node* current = head
  - For i from 0 to position -1 and current != NULL
    - Current = current -> next
  - if(current == NULL or current->next == NULL)
    - Print “invalid position”
  - Else

- Struct node* temp = current->next
- Current->next = temp->next
- free(temp)

3) search(int num)
- Initialize position = 0
- Struct node* temp = head
- while(temp!=NULL)
  - if(temp->data == num)
    - Print position
    - return
  - temp = temp->next
  - position++
- Print “Element not found”

C Program

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

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

struct Node* head = NULL;

struct Node* createNode(int num){
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = num;
    temp->next = NULL;
    return temp;
}

void insertAtBeginning(int num){
    struct Node* NewNode = createNode(num);
    NewNode->next=head;
    head = NewNode;
}

void insertAtEnd(int num){
    struct Node* NewNode = createNode(num);
    struct Node* temp = head;
    while(temp->next!=NULL){
        temp = temp->next;
    }
    temp->next = NewNode;
}

void insertAtPosition(int num, int position){
    if(position == 0){
        insertAtBeginning(num);
        return;
    }
```

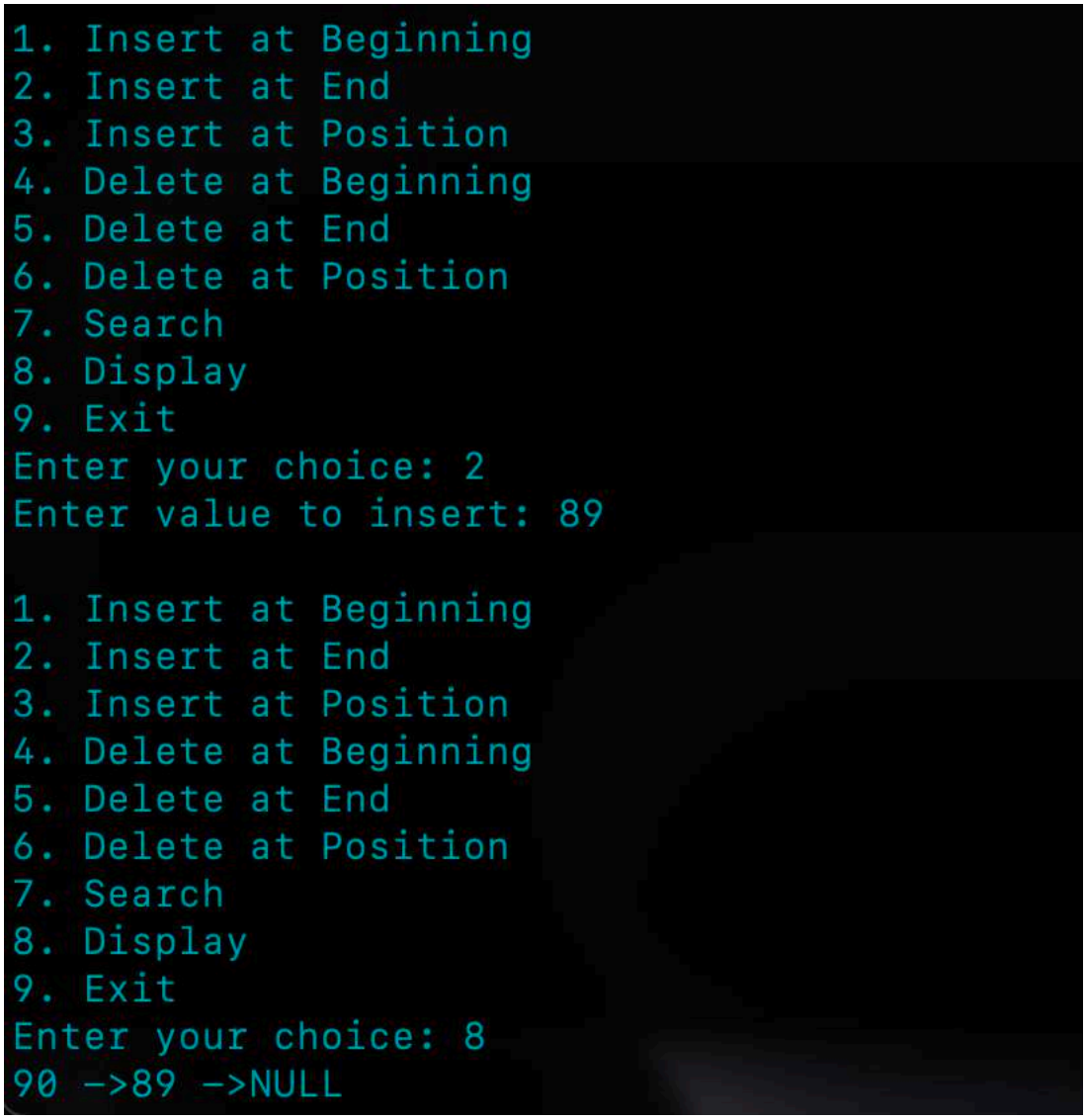
```
    struct Node* NewNode = createNode(num);
    struct Node* temp = head;
    for(int i = 0; i < position - 1 && temp != NULL; i++){
        temp = temp->next;
    }
    if(temp==NULL){
        printf("Invalid Position\n");
        return;
    }
    NewNode->next = temp->next;
    temp->next = NewNode;
}

void deleteAtBeginning(){
    if(head==NULL){
        printf("List Empty\n");
    } else{
        struct Node* temp = head;
        head = head->next;
        free(temp);
    }
}

void deleteAtEnd(){
    if(head == NULL){
        printf("List Empty\n");
    }
    else {
        struct Node* temp = head;
        while(temp->next->next != NULL){
            temp = temp->next;
        }
        free(temp->next);
        temp->next = NULL;
    }
}

void deleteAtPosition(int position){
    if(position == 0){
        deleteAtBeginning();
        return;
    }
    struct Node* current = head;
    for(int i=0;i<position - 1 && current!=NULL;i++){
        current = current->next;
    }
    if(current == NULL || current->next == NULL){
        printf("Invalid position\n");
    } else{
        struct Node* temp = current->next;
        current->next = temp->next;
        free(temp);
```

```
    }
}

void search(int num){
    int position=0;
    struct Node* temp = head;
    while(temp != NULL){
        if(temp->data == num){
            printf("Element found at position: %d\n", position);
            return;
        }
        temp = temp->next;
        position++;
    }
    printf("Element not found\n");
}

void display(){
    struct Node* temp = head;
    while(temp != NULL){
        printf("%d ->", temp->data);
        temp = temp->next;
    }
    printf("NULL");
}

int main() {
    int c, value, position;
    do {
        printf("\n1. Insert at Beginning\n2. Insert at End\n3.
Insert at Position\n");
        printf("4. Delete at Beginning\n5. Delete at End\n6.
Delete at Position\n");
        printf("7. Search\n8. Display\n9. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &c);

        switch (c) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insertAtBeginning(value);
                break;
            case 2:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insertAtEnd(value);
                break;
            case 3:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                printf("Enter position: ");
```

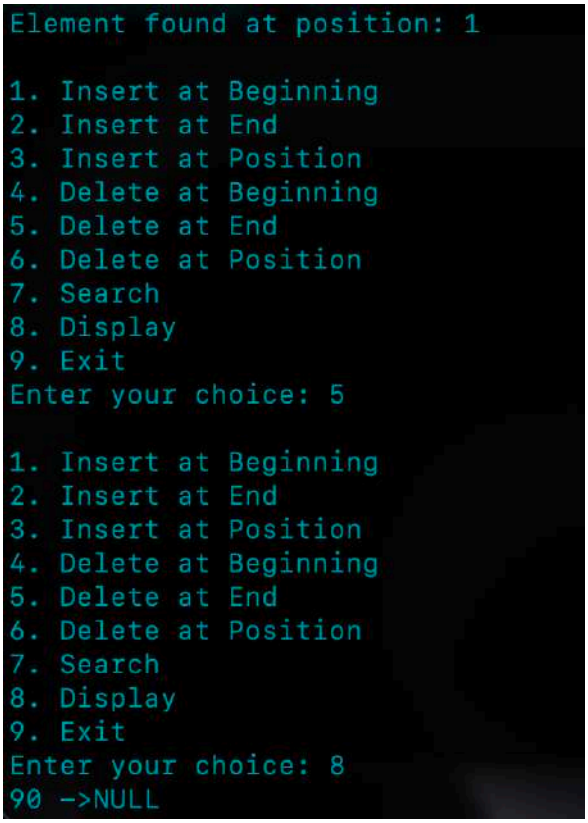
```
                scanf("%d", &position);
                insertAtPosition(value, position);
                break;
            case 4:
                deleteAtBeginning();
                break;
            case 5:
                deleteAtEnd();
                break;
            case 6:
                printf("Enter position to delete: ");
                scanf("%d", &position);
                deleteAtPosition(position);
                break;
            case 7:
                printf("Enter value to search: ");
                scanf("%d", &value);
                search(value);
                break;
            case 8:
                display();
                break;
            case 9:
                printf("Thank you\n");
                break;
        }
    } while (c != 9);
}
```

Test Case

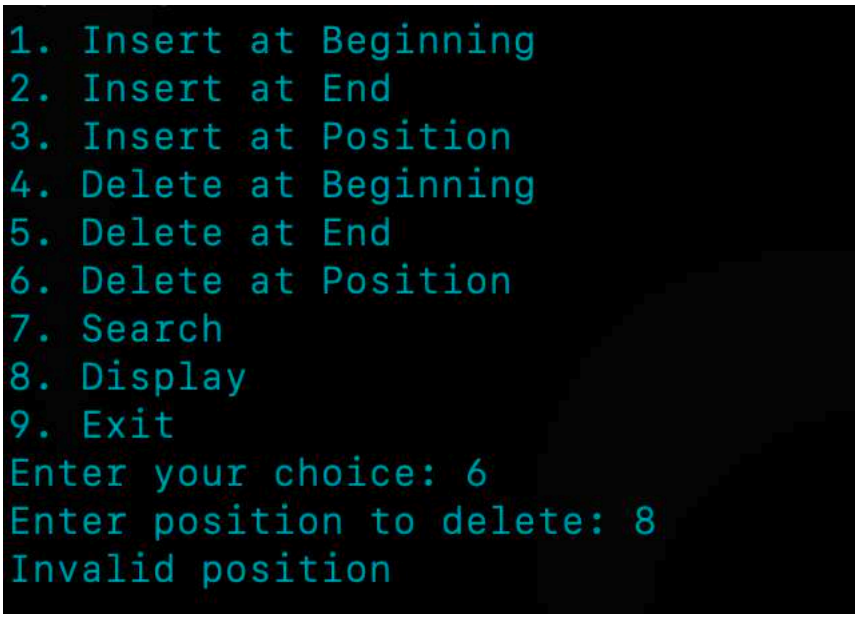
Test 1: General Condition



Test 2: Delete and search



Test 3: Invalid Position



Question 5

Algorithm

1) Insertion

insertAtBeginning(int num)

- struct node* NewNode = createNode(num)
- if(head == NULL)
  - Head = NewNode
- Else
  - Set newNode->next = head
  - Set Head->prev = newNode
  - Set head = newNode

insertAtEnd(int num)

- struct node* NewNode = createNode(num)
- struct node* temp = head
- while(temp -> next != NULL)
  - Set temp = temp->next
- Set temp->next = NewNode
- Set newNode->prev = temp

insertAtPosition(int num, int position)

- if(position == 0)
  - Call insertAtBeginning(num)
- Else
  - struct Node* NewNode = createNode(num)
  - struct Node* temp = head
  - For i from 0 to position - 1 && temp !=NULL
    - Set temp=temp->next
  - if(temp==NULL)
    - Print “Invalid position”
  - Else
    - Set NewNode->next = temp->next
    - Set newNode->prev = temp
    - if(temp->next != NULL)
      - Temp->next->prev = newNode
    - Temp->next = newNode

2) Deletion

deleteAtBeginning( )

- if(head == NULL)
  - Print “List Empty”
- Else
  - Struct Node* temp = head
  - Set head = head -> next
  - if(head!=NULL)
    - Set Head->prev = NULL
  - free(temp)

deleteAtEnd( )

- if(head == NULL)
  - Print “list empty”
- Else
  - Struct Node* temp = head
  - while(temp->next != NULL)
    - Set temp = temp->next
  - if(temp->prev!=NULL)
    - Set Temp->prev->next = NULL
  - Else

- Set head = NULL
- free(temp)

deleteAtPosition(int position)
- if(position == 0)
  - Call deleteAtBeginning( )
- Else
  - Struct Node* current = head
  - For i from 0 to position and current != NULL
    - Current = current -> next
  - if(current == NULL)
    - Print “invalid position”
    - return
  - if(current->prev != NULL)
    - current->prev->next = current->next
  - if(current->next != NULL)
    - Current->next->prev = current->prev
  - free(current)

3) search(int num)
- Initialize position = 0
- Struct node* temp = head
- while(temp!=NULL)
  - if(temp->data == num)
    - Print position
    - return
  - temp = temp->next
  - position++
- Print “Element not found”

C Program

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

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

struct Node* head = NULL;

struct Node* createNode(int num) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    newNode->data = num;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

void insertAtBeginning(int num) {
    struct Node* newNode = createNode(num);
    if (head == NULL) {
```

```
        head = newNode;
    } else {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
}

void insertAtEnd(int num) {
    struct Node* newNode = createNode(num);
    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}

void insertAtPosition(int num, int position) {
    if (position == 0) {
        insertAtBeginning(num);
        return;
    }
    struct Node* newNode = createNode(num);
    struct Node* temp = head;
    for (int i = 0; i < position - 1 && temp != NULL; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Invalid Position\n");
        free(newNode);
        return;
    }
    newNode->next = temp->next;
    newNode->prev = temp;
    if (temp->next != NULL) {
        temp->next->prev = newNode;
    }
    temp->next = newNode;
}

void deleteAtBeginning() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    head = head->next;
```

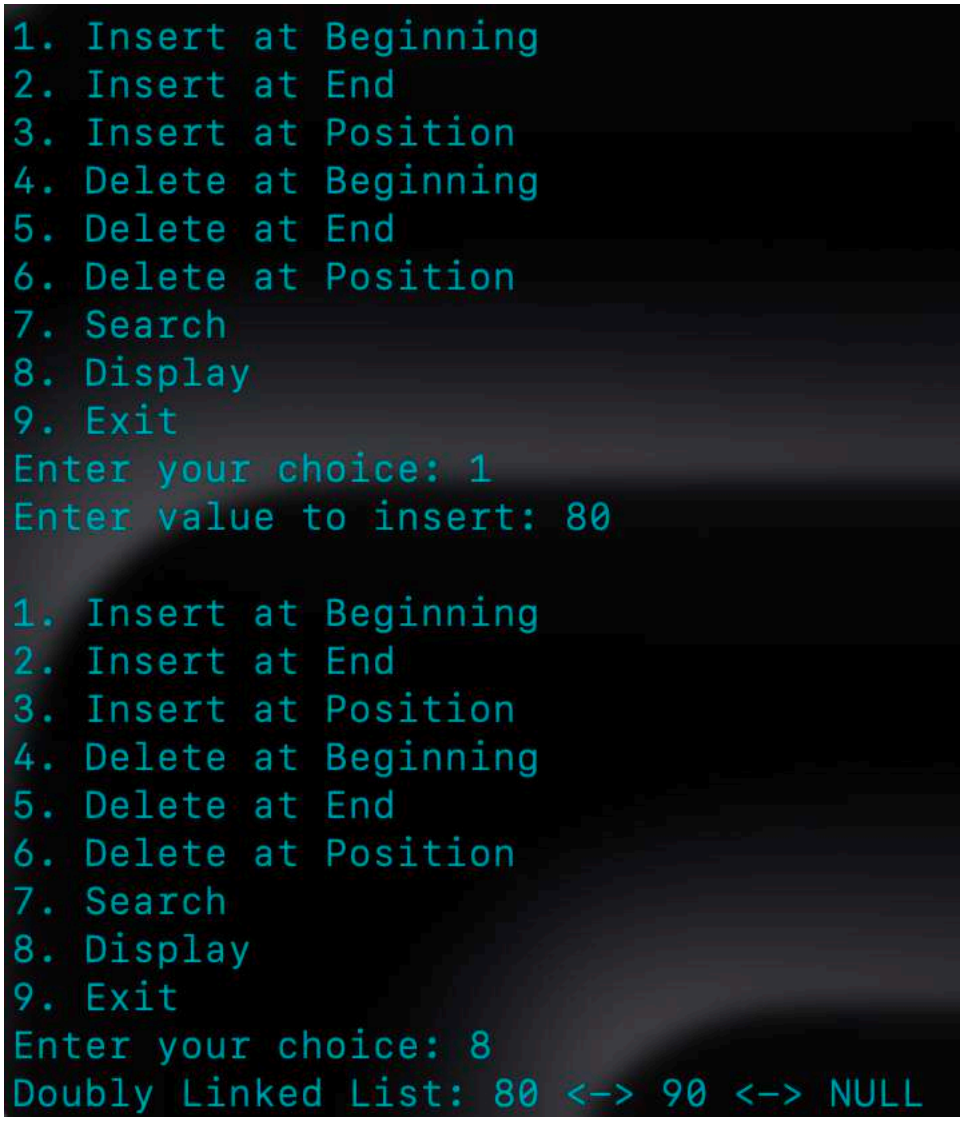
```
    if (head != NULL) {
        head->prev = NULL;
    }
    free(temp);
}

void deleteAtEnd() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    if (temp->prev != NULL) {
        temp->prev->next = NULL;
    } else {
        head = NULL;
    }
    free(temp);
}

void deleteAtPosition(int position) {
    if (position == 0) {
        deleteAtBeginning();
        return;
    }
    struct Node* temp = head;
    for (int i = 0; i < position && temp != NULL; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Invalid Position\n");
        return;
    }
    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    }
    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }
    free(temp);
}

void search(int num) {
    int position = 0;
    struct Node* temp = head;
    while (temp != NULL) {
        if (temp->data == num) {
            printf("Element found at position: %d\n", position);
            return;
```

```
        }
        temp = temp->next;
        position++;
    }
    printf("Element not found\n");
}

void display() {
    struct Node* temp = head;
    if (temp == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("Doubly Linked List: ");
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    int c, value, position;
    do {
        printf("\n1. Insert at Beginning\n2. Insert at End\n3.
Insert at Position\n");
        printf("4. Delete at Beginning\n5. Delete at End\n6.
Delete at Position\n");
        printf("7. Search\n8. Display\n9. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &c);

        switch (c) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insertAtBeginning(value);
                break;
            case 2:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insertAtEnd(value);
                break;
            case 3:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                printf("Enter position: ");
                scanf("%d", &position);
                insertAtPosition(value, position);
                break;
            case 4:
                deleteAtBeginning();
```

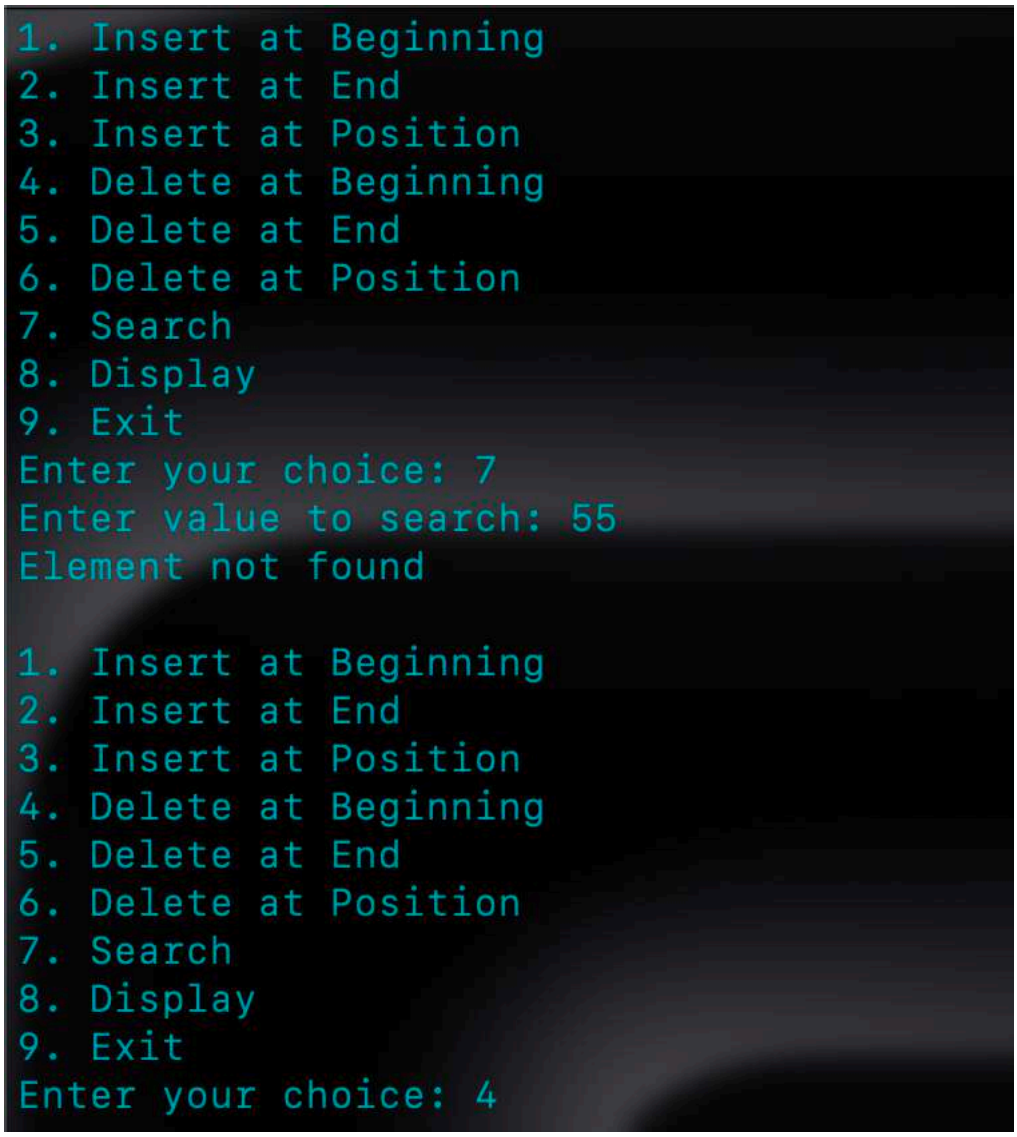
```
                break;
            case 5:
                deleteAtEnd();
                break;
            case 6:
                printf("Enter position to delete: ");
                scanf("%d", &position);
                deleteAtPosition(position);
                break;
            case 7:
                printf("Enter value to search: ");
                scanf("%d", &value);
                search(value);
                break;
            case 8:
                display();
                break;
            case 9:
                printf("Thank you\n");
                break;
        }
    } while (c != 9);
}
```

Test Case

Case 1: General



Case 2: search and delete



Case 3: Invalid position