DSA - LAB - ASSESSEMENT-1

24BBS0163 - KAPISH TICKOO

1.)**PSEUDOCODE-**

```
Define MAX == 4
Declare stack[MAX] as array
Declare top == -1

Function push(val):
   If top == MAX - 1:
      Print "Stack Overflow"
   Else:
      top = top + 1
      stack[top] = val

Function pop():
   If top == -1:
      Print "Stack Underflow"
   Else:
      Print "Popped: stack[top]"
      top = top - 1

Function display():
   If top == -1:
      Print "Stack is empty"
   Else:
      For i == 0 to top:
         Print stack[i]

Main:
   Declare choice, val
   Do:
      Print menu options:
         1. Push
         2. Pop
         3. Display
         0. Exit
      Input choice
      Switch(choice):
         Case 1:
            Print "Enter value to push"
            Input val
            Call push(val)
         Case 2:
            Call pop()
```

Case 3:
Call display()
Case 0:
Print "Exiting..."
Default:
Print "Invalid choice, try again"
While choice != 0

PROGRAM-

```c
#include<stdio.h>
#define MAX 4
int stack[MAX], top = -1;
void push(int val){
    if (top < MAX - 1){
        stack[++top] = val;
    } else{
        printf("Stack Overflow\n");
    }
}
void pop() {
    if (top > -1) {
        printf("Popped: %d\n", stack[top--]);
    } else {
        printf("Stack Underflow\n");
    }
}
void display(){
    if (top == -1){
        printf("Stack is empty\n");
    } else{
        for (int i = 0; i <= top; i++) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}
int main(){
    int choice, val;
    do {
        printf("\nMenu:\n");
        printf("1. Push\n2. Pop\n3. Display\n0. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice){
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &val);
                push(val);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 0:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice, try again.\n");
        }
    } while (choice != 0);
```

```
    return 0;
}
```

Testcase-1)

```
Menu:
1. Push
2. Pop
3. Display
0. Exit
Enter your choice: 1
Enter value to push: 10

Menu:
1. Push
2. Pop
3. Display
0. Exit
Enter your choice: 1
Enter value to push: 20

Menu:
1. Push
2. Pop
3. Display
0. Exit
Enter your choice: 1
Enter value to push: 30

Menu:
1. Push
2. Pop
3. Display
0. Exit
Enter your choice: 3
10 20 30

Menu:
1. Push
2. Pop
3. Display
0. Exit
Enter your choice: 0
Exiting...
PS C:\Users\Kapish\AppData\Local\Temp>
```

Testcase-2)

```
Menu:
1. Push
2. Pop
3. Display
0. Exit
Enter your choice: 2
Stack Underflow

Menu:
1. Push
2. Pop
3. Display
0. Exit
Enter your choice: 3
Stack is empty

Menu:
1. Push
2. Pop
3. Display
0. Exit
Enter your choice: 0
Exiting...
```

Testcase-3)

```
Menu:
1. Push
2. Pop
3. Display
0. Exit
Enter your choice: 1
Enter value to push: 5

Menu:
1. Push
2. Pop
3. Display
0. Exit
Enter your choice: 1
Enter value to push: 6

Menu:
1. Push
2. Pop
3. Display
0. Exit
Enter your choice: 1
Enter value to push: 7

Menu:
1. Push
2. Pop
3. Display
0. Exit
Enter your choice: 1
Enter value to push: 8

Menu:
1. Push
2. Pop
3. Display
0. Exit
Enter your choice: 1
Enter value to push: 9
Stack Overflow
```

2) PSEUDOCODE-

Define MAX == 4

```
Declare queue[MAX] as array
Declare front == -1, rear == -1

Function enqueue(val):
  If rear == MAX - 1:
    Print "Queue Overflow"
  Else:
    If front == -1:
      front = 0
    rear = rear + 1
    queue[rear] = val

Function dequeue():
  If front == -1 OR front > rear:
    Print "Queue Underflow"
  Else:
    Print "Dequeued: queue[front]"
    front = front + 1
    If front > rear:
      front = rear = -1

Function display():
  If front == -1:
    Print "Queue is empty"
  Else:
    For i == front to rear:
      Print queue[i]

Main:
  Declare choice, val
  Do:
    Print menu options:
      1. Enqueue
      2. Dequeue
      3. Display
      0. Exit
    Input choice
    Switch(choice):
      Case 1:
        Print "Enter value to enqueue"
        Input val
        Call enqueue(val)
      Case 2:
        Call dequeue()
      Case 3:
        Call display()
      Case 0:
        Print "Exiting..."
      Default:
        Print "Invalid choice, try again"
  While choice != 0
```

CODE-

```c
#include<stdio.h>
#define MAX 4

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

```c
void enqueue(int val){
    if(rear == MAX - 1){
        printf("Queue Overflow\n");
    }else{
        if(front == -1) front = 0;
        queue[++rear] = val;
    }
}
void dequeue(){
    if(front == -1 || front > rear){
        printf("Queue Underflow\n");
    }else{
        printf("Dequeued: %d\n", queue[front++]);
        if(front > rear) front = rear = -1; // Reset queue when empty
    }
}
void display(){
    if(front == -1){
        printf("Queue is empty\n");
    }else{
        for(int i = front; i <= rear; i++){
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}
int main(){
    int choice, val;
    do{
        printf("\nMenu:\n");
        printf("1. Enqueue\n2. Dequeue\n3. Display\n0. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d", &val);
                enqueue(val);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 0:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice, try again.\n");
        }
    }while(choice != 0);
    return 0;
}
```

Testcase-1)

```
Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 1
Enter value to enqueue: 10

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 1
Enter value to enqueue: 20

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 1
Enter value to enqueue: 30

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 3
10 20 30

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 2
Dequeued: 10
```

```
Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 3
20 30

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 0
Exiting...
```

Testcase-2)

```
Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 2
Queue Underflow

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 3
Queue is empty

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 0
Exiting...
PS C:\Users\Kapish\AppData\Local\Temp>
```

Testcase-3)

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

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 1
Enter value to enqueue: 5

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 1
Enter value to enqueue: 6

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 1
Enter value to enqueue: 7

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 1
Enter value to enqueue: 8
Queue Overflow
```

3)**PSEUDOCODE-**

```
Define MAX == 4
Declare queue[MAX] as array
Declare front == -1, rear == -1

Function enqueue(val):
  If (front == 0 AND rear == MAX-1) OR (rear == (front-1)%(MAX-1)):
    Print "Queue Overflow"
  Else If front == -1:
    front = rear = 0
    queue[rear] = val
  Else If rear == MAX-1 AND front != 0:
    rear = 0
    queue[rear] = val
  Else:
    rear = rear + 1
    queue[rear] = val

Function dequeue():
  If front == -1:
    Print "Queue Underflow"
  Else:
    Print "Dequeued: queue[front]"
    If front == rear:
      front = rear = -1
    Else If front == MAX-1:
      front = 0
    Else:
      front = front + 1

Function display():
  If front == -1:
    Print "Queue is empty"
  Else:
    If rear >= front:
      For i == front to rear:
        Print queue[i]
    Else:
      For i == front to MAX-1:
        Print queue[i]
      For i == 0 to rear:
        Print queue[i]

Main:
  Declare choice, val
  Do:
    Print menu options:
      1. Enqueue
      2. Dequeue
      3. Display
      0. Exit
    Input choice
    Switch(choice):
      Case 1:
        Print "Enter value to enqueue"
        Input val
        Call enqueue(val)
```

Case 2:
Call dequeue()
Case 3:
Call display()
Case 0:
Print "Exiting..."
Default:
Print "Invalid choice, try again"
While choice != 0

CODE-

```c
#include<stdio.h>
#define MAX 4

int queue[MAX], front = -1, rear = -1;
void enqueue(int val){
    if((front == 0 && rear == MAX-1) || (rear == (front-1)%(MAX-1))){
        printf("Queue Overflow\n");
    }else if(front == -1){
        front = rear = 0;
        queue[rear] = val;
    }else if(rear == MAX-1 && front != 0){
        rear = 0;
        queue[rear] = val;
    }else{
        queue[++rear] = val;
    }
}
void dequeue(){
    if(front == -1){
        printf("Queue Underflow\n");
    }else{
        printf("Dequeued: %d\n", queue[front]);
        if(front == rear){
            front = rear = -1;
        }else if(front == MAX-1){
            front = 0;
        }else{
            front++;
        }
    }
}
void display(){
    if(front == -1){
        printf("Queue is empty\n");
    }else{
        if(rear >= front){
            for(int i = front; i <= rear; i++){
                printf("%d ", queue[i]);
            }
        }else{
            for(int i = front; i < MAX; i++){
                printf("%d ", queue[i]);
            }
            for(int i = 0; i <= rear; i++){
                printf("%d ", queue[i]);
            }
        }
        printf("\n");
    }
}
```

```c
int main(){
    int choice, val;
    do{
        printf("\nMenu:\n");
        printf("1. Enqueue\n2. Dequeue\n3. Display\n0. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d", &val);
                enqueue(val);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 0:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice, try again.\n");
        }
    }while(choice != 0);
    return 0;
}
```

Testcase-1)

```
Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 1
Enter value to enqueue: 10

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 1
Enter value to enqueue: 20

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 2
Dequeued: 10

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 3
20
```

Testcase 2)

```
Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 1
Enter value to enqueue: 5

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 1
Enter value to enqueue: 6

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 1
Enter value to enqueue: 7

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 1
Enter value to enqueue: 8

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 1
Enter value to enqueue: 9
Queue Overflow
```

Testcase-3)

```
Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 1
Enter value to enqueue: 10

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 2
Dequeued: 10

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 2
Queue Underflow

Menu:
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter your choice: 3
Queue is empty
```

4)**PSEUDOCODE-**

```
Define Node as structure:
   data: integer
   next: pointer to Node

Declare head as pointer to Node initialized to NULL

Function insertAtBeginning(val):
   Create newNode
   newNode.data = val
   newNode.next = head
   head = newNode
```

```
Function insertAtEnd(val):
  Create newNode
  newNode.data = val
  newNode.next = NULL
  If head is NULL:
    head = newNode
  Else:
    temp = head
    While temp.next is not NULL:
      temp = temp.next
    temp.next = newNode

Function insertAtPosition(val, pos):
  Create newNode
  newNode.data = val
  If pos == 1:
    newNode.next = head
    head = newNode
    return
  temp = head
  For i from 1 to pos-1 and temp is not NULL:
    temp = temp.next
  If temp is NULL:
    Print "Position out of range"
    return
  newNode.next = temp.next
  temp.next = newNode

Function deleteFromBeginning():
  If head is NULL:
    Print "List is empty"
    return
  temp = head
  head = head.next
  Free temp

Function deleteFromEnd():
  If head is NULL:
    Print "List is empty"
    return
  If head.next is NULL:
    Free head
    head = NULL
    return
  temp = head
  While temp.next.next is not NULL:
    temp = temp.next
  Free temp.next
  temp.next = NULL

Function deleteFromPosition(pos):
  If head is NULL:
    Print "List is empty"
    return
  temp = head
  If pos == 1:
    head = head.next
```

```
    Free temp
    return
  prev = NULL
  For i from 1 to pos and temp is not NULL:
    prev = temp
    temp = temp.next
  If temp is NULL:
    Print "Position out of range"
    return
  prev.next = temp.next
  Free temp

Function search(val):
  temp = head
  pos = 1
  While temp is not NULL:
    If temp.data == val:
      Print "Element found at position pos"
      return
    temp = temp.next
    pos = pos + 1
  Print "Element not found"

Function display():
  temp = head
  While temp is not NULL:
    Print temp.data ->
    temp = temp.next
  Print NULL

Main:
  Declare choice, val, pos
  Do:
    Print menu options:
      1. Insert at Beginning
      2. Insert at End
      3. Insert at Position
      4. Delete from Beginning
      5. Delete from End
      6. Delete from Position
      7. Search
      8. Display
      0. Exit
    Input choice
    Switch(choice):
      Case 1:
        Print "Enter value to insert at beginning"
        Input val
        Call insertAtBeginning(val)
      Case 2:
        Print "Enter value to insert at end"
        Input val
        Call insertAtEnd(val)
      Case 3:
        Print "Enter value to insert"
        Input val
        Print "Enter position to insert"
```

```
            Input pos
            Call insertAtPosition(val, pos)
        Case 4:
            Call deleteFromBeginning()
        Case 5:
            Call deleteFromEnd()
        Case 6:
            Print "Enter position to delete from"
            Input pos
            Call deleteFromPosition(pos)
        Case 7:
            Print "Enter value to search"
            Input val
            Call search(val)
        Case 8:
            Call display()
        Case 0:
            Print "Exiting..."
        Default:
            Print "Invalid choice, try again"
    While choice != 0
```

CODE-

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

struct Node {
    int data;
    struct Node* next;
};
struct Node* head = NULL;
void insertAtBeginning(int val){
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = head;
    head = newNode;
}
void insertAtEnd(int val){
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = NULL;
    if(head == NULL){
        head = newNode;
    } else {
        struct Node* temp = head;
        while(temp->next != NULL) temp = temp->next;
        temp->next = newNode;
    }
}
void insertAtPosition(int val, int pos){
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = val;
    if(pos == 1){
        newNode->next = head;
        head = newNode;
        return;
    }
    struct Node* temp = head;
    for(int i = 1; i < pos-1 && temp != NULL; i++) temp = temp->next;
    if(temp == NULL){
```

```
        printf("Position out of range\n");
        return;
    }
    newNode->next = temp->next;
    temp->next = newNode;
}
void deleteFromBeginning(){
    if(head == NULL){
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    head = head->next;
    free(temp);
}
void deleteFromEnd(){
    if(head == NULL){
        printf("List is empty\n");
        return;
    }
    if(head->next == NULL){
        free(head);
        head = NULL;
        return;
    }
    struct Node* temp = head;
    while(temp->next->next != NULL) temp = temp->next;
    free(temp->next);
    temp->next = NULL;
}
void deleteFromPosition(int pos){
    if(head == NULL){
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    if(pos == 1){
        head = head->next;
        free(temp);
        return;
    }
    struct Node* prev = NULL;
    for(int i = 1; i < pos && temp != NULL; i++){
        prev = temp;
        temp = temp->next;
    }
    if(temp == NULL){
        printf("Position out of range\n");
        return;
    }
    prev->next = temp->next;
    free(temp);
}
void search(int val){
    struct Node* temp = head;
    int pos = 1;
    while(temp != NULL){
        if(temp->data == val){
            printf("Element found at position %d\n", pos);
            return;
        }
```

```
        temp = temp->next;
        pos++;
    }
    printf("Element not found\n");
}
void display(){
    struct Node* temp = head;
    while(temp != NULL){
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
int main(){
    int choice, val, pos;
    do{
        printf("\nMenu:\n");
        printf("1. Insert at Beginning\n2. Insert at End\n3. Insert at Position\n4. Delete
from Beginning\n5. Delete from End\n6. Delete from Position\n7. Search\n8. Display\n0.
Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1:
                printf("Enter value to insert at beginning: ");
                scanf("%d", &val);
                insertAtBeginning(val);
                break;
            case 2:
                printf("Enter value to insert at end: ");
                scanf("%d", &val);
                insertAtEnd(val);
                break;
            case 3:
                printf("Enter value to insert: ");
                scanf("%d", &val);
                printf("Enter position to insert: ");
                scanf("%d", &pos);
                insertAtPosition(val, pos);
                break;
            case 4:
                deleteFromBeginning();
                break;
            case 5:
                deleteFromEnd();
                break;
            case 6:
                printf("Enter position to delete from: ");
                scanf("%d", &pos);
                deleteFromPosition(pos);
                break;
            case 7:
                printf("Enter value to search: ");
                scanf("%d", &val);
                search(val);
                break;
            case 8:
                display();
                break;
            case 0:
                printf("Exiting...\n");
                break;
```

```
            default:
                printf("Invalid choice, try again.\n");
        }
    }while(choice != 0);
    return 0;
}
```

Testcases

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 1
Enter value to insert at beginning: 20

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 2
Enter value to insert at end: 10

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 7
Enter value to search: 10
Element found at position 2
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 3
Enter value to insert: 30
Enter position to insert: 2

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 8
20 -> 30 -> 10 -> NULL
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 1
Enter value to insert at beginning: 50

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 1
Enter value to insert at beginning: 40

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 4
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 4

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 4
List is empty

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 8
NULL
```

5) Pseudocode

Define Node as structure:
  data: integer
  prev: pointer to Node
  next: pointer to Node

Declare head as pointer to Node initialized to NULL

Function insertAtBeginning(val):
  Create newNode
  newNode.data = val
  newNode.prev = NULL
  newNode.next = head
  If head is not NULL:
    head.prev = newNode
  head = newNode

```
Function insertAtEnd(val):
    Create newNode
    newNode.data = val
    newNode.next = NULL
    If head is NULL:
        newNode.prev = NULL
        head = newNode
    Else:
        temp = head
        While temp.next is not NULL:
            temp = temp.next
        temp.next = newNode
        newNode.prev = temp

Function insertAtPosition(val, pos):
    Create newNode
    newNode.data = val
    If pos == 1:
        newNode.next = head
        newNode.prev = NULL
        If head is not NULL:
            head.prev = newNode
        head = newNode
        return
    temp = head
    For i from 1 to pos-1 and temp is not NULL:
        temp = temp.next
    If temp is NULL:
        Print "Position out of range"
        return
    newNode.next = temp.next
    newNode.prev = temp
    If temp.next is not NULL:
        temp.next.prev = newNode
    temp.next = newNode

Function deleteFromBeginning():
    If head is NULL:
        Print "List is empty"
        return
    temp = head
    head = head.next
    If head is not NULL:
        head.prev = NULL
    Free temp

Function deleteFromEnd():
    If head is NULL:
        Print "List is empty"
        return
    temp = head
    While temp.next is not NULL:
        temp = temp.next
    If temp.prev is not NULL:
        temp.prev.next = NULL
    Else:
```

```
        head = NULL
    Free temp

Function deleteFromPosition(pos):
    If head is NULL:
        Print "List is empty"
        return
    temp = head
    If pos == 1:
        head = head.next
        If head is not NULL:
            head.prev = NULL
        Free temp
        return
    For i from 1 to pos and temp is not NULL:
        temp = temp.next
    If temp is NULL:
        Print "Position out of range"
        return
    If temp.prev is not NULL:
        temp.prev.next = temp.next
    If temp.next is not NULL:
        temp.next.prev = temp.prev
    Free temp

Function search(val):
    temp = head
    pos = 1
    While temp is not NULL:
        If temp.data == val:
            Print "Element found at position pos"
            return
        temp = temp.next
        pos = pos + 1
    Print "Element not found"

Function display():
    temp = head
    While temp is not NULL:
        Print temp.data <->
        temp = temp.next
    Print NULL

Main:
    Declare choice, val, pos
    Do:
        Print menu options:
            1. Insert at Beginning
            2. Insert at End
            3. Insert at Position
            4. Delete from Beginning
            5. Delete from End
            6. Delete from Position
            7. Search
            8. Display
            0. Exit
        Input choice
```

Switch(choice):
  Case 1:
    Print "Enter value to insert at beginning"
    Input val
    Call insertAtBeginning(val)
  Case 2:
    Print "Enter value to insert at end"
    Input val
    Call insertAtEnd(val)
  Case 3:
    Print "Enter value to insert"
    Input val
    Print "Enter position to insert"
    Input pos
    Call insertAtPosition(val, pos)
  Case 4:
    Call deleteFromBeginning()

Code-

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

struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};
struct Node* head = NULL;
void insertAtBeginning(int val){
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->prev = NULL;
    newNode->next = head;
    if(head != NULL) head->prev = newNode;
    head = newNode;
}
void insertAtEnd(int val){
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = val;
    newNode->next = NULL;
    if(head == NULL){
        newNode->prev = NULL;
        head = newNode;
    } else {
        struct Node* temp = head;
        while(temp->next != NULL) temp = temp->next;
        temp->next = newNode;
        newNode->prev = temp;
    }
}
void insertAtPosition(int val, int pos){
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = val;
    if(pos == 1){
        newNode->next = head;
        newNode->prev = NULL;
        if(head != NULL) head->prev = newNode;
        head = newNode;
        return;
    }
```

```
    struct Node* temp = head;
    for(int i = 1; i < pos-1 && temp != NULL; i++) temp = temp->next;
    if(temp == NULL){
        printf("Position out of range\n");
        return;
    }
    newNode->next = temp->next;
    newNode->prev = temp;
    if(temp->next != NULL) temp->next->prev = newNode;
    temp->next = newNode;
}
void deleteFromBeginning(){
    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 deleteFromEnd(){
    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; // Only one element was present
    free(temp);
}
void deleteFromPosition(int pos){
    if(head == NULL){
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    if(pos == 1){
        head = head->next;
        if(head != NULL) head->prev = NULL;
        free(temp);
        return;
    }
    for(int i = 1; i < pos && temp != NULL; i++) temp = temp->next;
    if(temp == NULL){
        printf("Position out of range\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 val){
    struct Node* temp = head;
    int pos = 1;
    while(temp != NULL){
        if(temp->data == val){
            printf("Element found at position %d\n", pos);
            return;
        }
```

```
        temp = temp->next;
        pos++;
    }
    printf("Element not found\n");
}
void display(){
    struct Node* temp = head;
    while(temp != NULL){
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
int main(){
    int choice, val, pos;
    do{
        printf("\nMenu:\n");
        printf("1. Insert at Beginning\n2. Insert at End\n3. Insert at Position\n4. Delete
from Beginning\n5. Delete from End\n6. Delete from Position\n7. Search\n8. Display\n0.
Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1:
                printf("Enter value to insert at beginning: ");
                scanf("%d", &val);
                insertAtBeginning(val);
                break;
            case 2:
                printf("Enter value to insert at end: ");
                scanf("%d", &val);
                insertAtEnd(val);
                break;
            case 3:
                printf("Enter value to insert: ");
                scanf("%d", &val);
                printf("Enter position to insert: ");
                scanf("%d", &pos);
                insertAtPosition(val, pos);
                break;
            case 4:
                deleteFromBeginning();
                break;
            case 5:
                deleteFromEnd();
                break;
            case 6:
                printf("Enter position to delete from: ");
                scanf("%d", &pos);
                deleteFromPosition(pos);
                break;
            case 7:
                printf("Enter value to search: ");
                scanf("%d", &val);
                search(val);
                break;
            case 8:
                display();
                break;
            case 0:
                printf("Exiting...\n");
                break;
```

```
            default:
                printf("Invalid choice, try again.\n");
        }
    }while(choice != 0);
    return 0;
}
```

Testcases-

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 1
Enter value to insert at beginning: 10

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 1
Enter value to insert at beginning: 20

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 8
20 <-> 10 <-> NULL
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 2
Enter value to insert at end: 20

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 2
Enter value to insert at end: 30

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 2
Enter value to insert at end: 50
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 5

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete from Position
7. Search
8. Display
0. Exit
Enter your choice: 8
20 <-> 30 <-> NULL
```