Course Code: CBS1003

Coure Name: Data Structures and Algorithms

Assessment-1

Name: Rijul Kothawade

Registration No: 24BBS0230

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

a. PUSH()

b. POP()

c. Display()

Pseudocode:

```
BEGIN
Create an empty stack S and set TOP=-1
Define Max_size as size of stack

Function PUSH():
If TOP== Max_size -1 then
Print “Stack Overflow”
Else
Print "Enter the element to push: "
        Read element
        TOP = TOP + 1
        STACK[TOP] = element
        Print "Element pushed successfully"

  FUNCTION POP():
     If TOP == -1 then
```

```
        Print "Stack Underflow"
    Else
        Print "Popped element: ", STACK[TOP]
        TOP = TOP - 1

FUNCTION DISPLAY():
    IF TOP == -1 then
        Print "Stack is empty"
    Else
        Print "Stack elements are: "
        FOR i = TOP DOWNTO 0 DO
            Print STACK[i]

REPEAT
    Print "Menu:"
    Print "1. PUSH"
    Print "2. POP"
    Print "3. DISPLAY"
    Print "4. EXIT"
    Print "Enter your choice: "
    Read CHOICE

    SWITCH (CHOICE)
        CASE 1:
```

```
            CALL PUSH()
        CASE 2:
            CALL POP()
        CASE 3:
            CALL DISPLAY()
        CASE 4:
            Print "Exiting program"
            EXIT
        DEFAULT:
            Print "Invalid choice! Please try again."
    END SWITCH
  UNTIL FALSE
END
```

CODE

```
#include<stdio.h>
#define MAX 100

int stack[MAX];
int top=-1;

void push(){
   if (top==MAX-1){
```

```
        Printf("Stack overflow. \n");
    } else {
        int element;
        Printf("Enter the element to enter: ");
        scanf("%d", &element);

        top++;
        stack[top]=element;
        Printf("Element pushed succesfully. \n");
    }
}

void pop() {
    if (top == -1) {
        Printf("Stack Underflow\n");
    } else {
        Printf("Popped element: %d\n", stack[top]);
        top--;
    }
}

void display() {
    if (top == -1) {
        Printf("Stack is empty\n");
```

```
    } else {
        Printf("The elements in stack are: ");
        for (int i = top; i >= 0; i--) {
            Printf("%d ", stack[i]);
        }
        Printf("\n");
    }
}

int main(){
    int choice;
    do {
        Printf("\nMenu:\n");
        Printf("1. PUSH\n");
        Printf("2. POP\n");
        Printf("3. DISPLAY\n");
        Printf("4. EXIT\n");
        Printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                push();
                break;
```

```
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                Printf("Exiting program\n");
                break;
            default:
                Printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

TestCases:

```
PS C:\Users\rijul\OneDrive\Desktop\c programs> cd "c:\Users\rijul\OneDrive\Desktop\c programs\" ; if ($?) { gcc stack.c -o sta
ck } ; if ($?) { .\stack }

Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the element to enter: 10
Element pushed succesfully.

Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the element to enter: 14
Element pushed succesfully.

Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the element to enter: 72
Element pushed succesfully.
```

```
Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 3
The elements in stack are: 72 14 10

Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice:
2
Popped element: 72
```

```
Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 2
Popped element: 14

Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 2
Popped element: 10

Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 2
Stack Underflow

Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 3
Stack is empty
```

```
Menu:
1. PUSH
2. POP
3. DISPLAY
4. EXIT
Enter your choice: 4
Exiting program
```

Q2. Write a menu driven program to implement the following operations on Queue: a. Enqueue() b. Dequeue() c. Display()

Pseudocode:

```
BEGIN
   Create an empty queue (QUEUE) with size MAX_SIZE
   Initialize FRONT = -1 and REAR = -1

   FUNCTION ENQUEUE():
      IF REAR == MAX_SIZE - 1 Then
         Print "Queue Overflow"
      ELSE
         Print "Enter the element to enqueue: "
         READ ELEMENT
         IF FRONT == -1 Then
            FRONT = 0  // Initialize FRONT if it's the first element
         END IF
         REAR = REAR + 1
         QUEUE[REAR] = ELEMENT
         Print "Element enqueued successfully"
      END IF
```

```
FUNCTION DEQUEUE():
   IF FRONT == -1 OR FRONT > REAR Then
      Print "Queue Underflow"
   ELSE
      Print "Dequeued element: ", QUEUE[FRONT]
      FRONT = FRONT + 1
      IF FRONT > REAR THEN
         FRONT = -1
         REAR = -1
      END IF
   END IF

FUNCTION DISPLAY():
   IF FRONT == -1 OR FRONT > REAR THEN
      Print "Queue is empty"
   ELSE
      Print "Queue elements are: "
      FOR i = FRONT TO REAR DO
         Print QUEUE[i]
      END FOR
   END IF

REPEAT
```

```
        Print "Menu:"
        Print "1. ENQUEUE"
        Print "2. DEQUEUE"
        Print "3. DISPLAY"
        Print "4. EXIT"
        Print "Enter your choice: "
        READ CHOICE

        SWITCH (CHOICE)
            CASE 1:
                CALL ENQUEUE()
            CASE 2:
                CALL DEQUEUE()
            CASE 3:
                CALL DISPLAY()
            CASE 4:
                Print "Exiting program"
                EXIT
            DEFAULT:
                Print "Invalid choice! Please try again."
        END SWITCH
    UNTIL FALSE
END
```

CODE:

```
#include <stdio.h>
#define MAX_SIZE 10

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

void enqueue() {
    if (rear == MAX_SIZE - 1) {
        Printf("Queue Overflow\n");
    } else {
        int element;
        Printf("Enter the element to enqueue: ");
        scanf("%d", &element);
        if (front == -1) {
            front = 0;
        }
        rear++;
        queue[rear] = element;
        Printf("Element enqueued successfully\n");
    }
}

void dequeue() {
```

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

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

```
int main() {
    int choice;
    do {
        Printf("\nMenu:\n");
        Printf("1. ENQUEUE\n");
        Printf("2. DEQUEUE\n");
        Printf("3. DISPLAY\n");
        Printf("4. EXIT\n");
        Printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                Printf("Exiting program\n");
                break;
```

```
        default:
            Printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

TESTCASES:

```
PS C:\Users\rijul\OneDrive\Desktop\c programs> cd "c:\Users\rijul\OneDrive\Desktop\c programs\" ; if ($?) { gcc queue.c -o que
ue } ; if ($?) { .\queue }

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the element to enqueue: 10
Element enqueued successfully

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the element to enqueue: 12
Element enqueued successfully

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the element to enqueue: 14
Element enqueued successfully
```

```
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the element to enqueue: 20
Element enqueued successfully

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 3
Queue elements are: 10 12 14 20

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Dequeued element: 10
```

```
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the element to enqueue: 23
Element enqueued successfully

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 3
Queue elements are: 12 14 20 23

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Dequeued element: 12
```

```
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Dequeued element: 14

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Dequeued element: 20

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Dequeued element: 23
```

```
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Queue Underflow

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 3
Queue is empty

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 4
Exiting program
```

Q3) Write a menu driven program to implement the following operations on circular Queue: a. Enqueue() b. Dequeue() c. Display()

PSEUDOCODE:

```
BEGIN
   Create an empty circular queue (QUEUE) with size MAX_SIZE
   Initialize FRONT = -1 and REAR = -1


   FUNCTION ENQUEUE():
      IF (FRONT == 0 AND REAR == MAX_SIZE - 1) OR (REAR + 1 == FRONT) Then
         Print "Queue Overflow"
      ELSE
         Print "Enter the element to enqueue: "
         READ ELEMENT
         IF FRONT == -1 THEN
            FRONT = 0
         END IF
         REAR = (REAR + 1) MOD MAX_SIZE
         QUEUE[REAR] = ELEMENT
         Print "Element enqueued successfully"
      END IF
```

```
FUNCTION DEQUEUE():
   IF FRONT == -1 THEN
      Print "Queue Underflow"
   ELSE
      Print "Dequeued element: ", QUEUE[FRONT]
      IF FRONT == REAR THEN
         FRONT = -1
         REAR = -1
      ELSE
         FRONT = (FRONT + 1) MOD MAX_SIZE
      END IF
   END IF

FUNCTION DISPLAY():
   IF FRONT == -1 THEN
      Print "Queue is empty"
   ELSE
      Print "Queue elements are: "
      i = FRONT
      REPEAT
         Print QUEUE[i]
         i = (i + 1) MOD MAX_SIZE
      UNTIL i == (REAR + 1) MOD MAX_SIZE
      Print "\n"
```

```
    END IF

REPEAT
    Print "Menu:"
    Print "1. ENQUEUE"
    Print "2. DEQUEUE"
    Print "3. DISPLAY"
    Print "4. EXIT"
    Print "Enter your choice: "
    READ CHOICE

    SWITCH (CHOICE)
        CASE 1:
            CALL ENQUEUE()
        CASE 2:
            CALL DEQUEUE()
        CASE 3:
            CALL DISPLAY()
        CASE 4:
            Print "Exiting program"
            EXIT
        DEFAULT:
            Print "Invalid choice! Please try again."
    END SWITCH
```

```
    UNTIL FALSE
END
```

CODE:

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

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

void enqueue() {
    if ((front == 0 && rear == MAX_SIZE - 1) || (rear + 1 == front)) {
        Printf("Queue Overflow\n");
    } else {
        int element;
        Printf("Enter the element to enqueue: ");
        scanf("%d", &element);
        if (front == -1) {
            front = 0;
        }
        rear = (rear + 1) % MAX_SIZE;
        queue[rear] = element;
        Printf("Element enqueued successfully\n");
```

```
    }
}

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

void display() {
    if (front == -1) {
        Printf("Circular Queue is empty\n");
    } else {
        Printf("Circular Queue elements are: ");
        int i = front;
        do {
```

```
            Printf("%d ", queue[i]);
            i = (i + 1) % MAX_SIZE;
        } while (i != (rear + 1) % MAX_SIZE);
        Printf("\n");
    }
}

int main() {
    int choice;
    do {
        Printf("\nMenu:\n");
        Printf("1. ENQUEUE\n");
        Printf("2. DEQUEUE\n");
        Printf("3. DISPLAY\n");
        Printf("4. EXIT\n");
        Printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
```

```
            break;
        case 3:
            display();
            break;
        case 4:
            Printf("Exiting program\n");
            break;
        default:
            Printf("Invalid choice! Please try again.\n");
      }
   } while (choice != 4);

   return 0;
}
```

TESTCASES:

```
PS C:\Users\rijul\OneDrive\Desktop\c programs> cd "c:\Users\rijul\OneDrive\Desktop\c programs\" ; if ($?) { gcc circular_queue
.c -o circular_queue } ; if ($?) { .\circular_queue }

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the element to enqueue: 10
Element enqueued successfully

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the element to enqueue: 23
Element enqueued successfully
```

```
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the element to enqueue: 67
Element enqueued successfully

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the element to enqueue: 45
Element enqueued successfully

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the element to enqueue: 5
Element enqueued successfully
```

```
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Queue Overflow

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 3
Circular Queue elements are: 10 23 67 45 5

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Dequeued element: 10

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter the element to enqueue: 80
Element enqueued successfully
```

```
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 3
Circular Queue elements are: 23 67 45 5 80

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Dequeued element: 23

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Dequeued element: 67

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Dequeued element: 45
```

```
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Dequeued element: 5

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Dequeued element: 80

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 2
Queue Underflow

Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 3
Circular Queue is empty
```

```
Menu:
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 4
```

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.

PSEUDOCODE:

```
BEGIN
  Define a NODE structure with:
    DATA: to store the value
    NEXT: to store the address of the next node
  Initialize HEAD = NULL (empty linked list)

  FUNCTION INSERT_BEGINNING(ELEMENT):
    Create a new NODE
    SET NODE.DATA = ELEMENT
    SET NODE.NEXT = HEAD
    HEAD = NODE
    Print "Element inserted at the beginning"

  FUNCTION INSERT_END(ELEMENT):
    Create a new NODE
    SET NODE.DATA = ELEMENT
```

```
    SET NODE.NEXT = NULL
    IF HEAD == NULL Then
        HEAD = NODE
    ELSE
        SET TEMP = HEAD
        WHILE TEMP.NEXT != NULL DO
            TEMP = TEMP.NEXT
        END WHILE
        TEMP.NEXT = NODE
    END IF
    Print "Element inserted at the end"

FUNCTION INSERT_AT_POSITION(ELEMENT, POSITION):
    Create a new NODE
    SET NODE.DATA = ELEMENT
    IF POSITION == 1 Then
        SET NODE.NEXT = HEAD
        HEAD = NODE
    ELSE
        SET TEMP = HEAD
        FOR i = 1 TO POSITION - 2 DO
            IF TEMP == NULL Then
                Print "Invalid position"
                RETURN
```

```
            END IF
            TEMP = TEMP.NEXT
        END FOR
        SET NODE.NEXT = TEMP.NEXT
        TEMP.NEXT = NODE
    END IF
    Print "Element inserted at position ", POSITION

FUNCTION DELETE_BEGINNING():
    IF HEAD == NULL Then
        Print "List is empty"
    ELSE
        SET TEMP = HEAD
        HEAD = HEAD.NEXT
        FREE TEMP
        Print "Element deleted from the beginning"
    END IF

FUNCTION DELETE_END():
    IF HEAD == NULL Then
        Print "List is empty"
    ELSE IF HEAD.NEXT == NULL Then
        FREE HEAD
        HEAD = NULL
```

```
    ELSE
        SET TEMP = HEAD
        WHILE TEMP.NEXT.NEXT != NULL DO
            TEMP = TEMP.NEXT
        END WHILE
        FREE TEMP.NEXT
        TEMP.NEXT = NULL
    END IF
    Print "Element deleted from the end"

FUNCTION DELETE_AT_POSITION(POSITION):
    IF HEAD == NULL Then
        Print "List is empty"
    ELSE IF POSITION == 1 Then
        SET TEMP = HEAD
        HEAD = HEAD.NEXT
        FREE TEMP
    ELSE
        SET TEMP = HEAD
        FOR i = 1 TO POSITION - 2 DO
            IF TEMP == NULL OR TEMP.NEXT == NULL Then
                Print "Invalid position"
                RETURN
            END IF
```

```
            TEMP = TEMP.NEXT
        END FOR
        SET DELETE_NODE = TEMP.NEXT
        TEMP.NEXT = DELETE_NODE.NEXT
        FREE DELETE_NODE
    END IF
    Print "Element deleted at position ", POSITION

FUNCTION SEARCH(ELEMENT):
    SET TEMP = HEAD
    SET POSITION = 1
    WHILE TEMP != NULL DO
        IF TEMP.DATA == ELEMENT Then
            Print "Element found at position ", POSITION
            RETURN
        END IF
        TEMP = TEMP.NEXT
        POSITION = POSITION + 1
    END WHILE
    Print "Element not found in the list"

FUNCTION DISPLAY():
    IF HEAD == NULL Then
        Print "List is empty"
```

```
    ELSE
        SET TEMP = HEAD
        Print "Elements in the list: "
        WHILE TEMP != NULL DO
            Print TEMP.DATA
            TEMP = TEMP.NEXT
        END WHILE
    END IF

REPEAT
    Print "Menu:"
    Print "1. INSERT at Beginning"
    Print "2. INSERT at End"
    Print "3. INSERT at a Position"
    Print "4. DELETE from Beginning"
    Print "5. DELETE from End"
    Print "6. DELETE at a Position"
    Print "7. SEARCH for an Element"
    Print "8. DISPLAY the List"
    Print "9. EXIT"
    Print "Enter your choice: "
    READ CHOICE

    SWITCH (CHOICE)
```

```
CASE 1:
   Print "Enter the element: "
   READ ELEMENT
   CALL INSERT_BEGINNING(ELEMENT)
CASE 2:
   Print "Enter the element: "
   READ ELEMENT
   CALL INSERT_END(ELEMENT)
CASE 3:
   Print "Enter the element: "
   READ ELEMENT
   Print "Enter the position: "
   READ POSITION
   CALL INSERT_AT_POSITION(ELEMENT, POSITION)
CASE 4:
   CALL DELETE_BEGINNING()
CASE 5:
   CALL DELETE_END()
CASE 6:
   Print "Enter the position: "
   READ POSITION
   CALL DELETE_AT_POSITION(POSITION)
CASE 7:
   Print "Enter the element to search: "
```

```
            READ ELEMENT
            CALL SEARCH(ELEMENT)
        CASE 8:
            CALL DISPLAY()
        CASE 9:
            Print "Exiting program"
            EXIT
        DEFAULT:
            Print "Invalid choice! Please try again."
    END SWITCH
  UNTIL FALSE
END
```

CODE:

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

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

struct Node* head = NULL;
```

```
void insertAtBeginning(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
   newNode->data = value;
   newNode->next = head;
   head = newNode;
   Printf("Element inserted at the beginning\n");
}

void insertAtEnd(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
   newNode->data = value;
   newNode->next = NULL;

   if (head == NULL) {
      head = newNode;
   } else {
      struct Node* temp = head;
      while (temp->next != NULL) {
         temp = temp->next;
      }
      temp->next = newNode;
   }
```

```
    Printf("Element inserted at the end\n");
}

void insertAtPosition(int value, int position) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    newNode->data = value;

    if (position == 1) {
        newNode->next = head;
        head = newNode;
    } else {
        struct Node* temp = head;
        for (int i = 1; i < position - 1; i++) {
            if (temp == NULL) {
                Printf("Invalid position\n");
                free(newNode);
                return;
            }
            temp = temp->next;
        }
        newNode->next = temp->next;
        temp->next = newNode;
    }
```

```
    Printf("Element inserted at position %d\n", position);
}

void deleteFromBeginning() {
    if (head == NULL) {
        Printf("List is empty\n");
    } else {
        struct Node* temp = head;
        head = head->next;
        free(temp);
        Printf("Element deleted from the beginning\n");
    }
}

void deleteFromEnd() {
    if (head == NULL) {
        Printf("List is empty\n");
    } else if (head->next == NULL) {
        free(head);
        head = NULL;
        Printf("Element deleted from the end\n");
    } else {
        struct Node* temp = head;
        while (temp->next->next != NULL) {
```

```
            temp = temp->next;
        }
        free(temp->next);
        temp->next = NULL;
        Printf("Element deleted from the end\n");
    }
}

void deleteAtPosition(int position) {
    if (head == NULL) {
        Printf("List is empty\n");
    } else if (position == 1) {
        struct Node* temp = head;
        head = head->next;
        free(temp);
        Printf("Element deleted from position %d\n", position);
    } else {
        struct Node* temp = head;
        for (int i = 1; i < position - 1; i++) {
            if (temp == NULL || temp->next == NULL) {
                Printf("Invalid position\n");
                return;
            }
            temp = temp->next;
```

```
        }
        struct Node* deleteNode = temp->next;
        if (deleteNode == NULL) {
            Printf("Invalid position\n");
        } else {
            temp->next = deleteNode->next;
            free(deleteNode);
            Printf("Element deleted from position %d\n", position);
        }
    }
}

void search(int value) {
    struct Node* temp = head;
    int position = 1;
    while (temp != NULL) {
        if (temp->data == value) {
            Printf("Element %d found at position %d\n", value, position);
            return;
        }
        temp = temp->next;
        position++;
    }
    Printf("Element %d not found in the list\n", value);
```

```
}

void display() {
    if (head == NULL) {
        Printf("List is empty\n");
    } else {
        struct Node* temp = head;
        Printf("Elements in the list: ");
        while (temp != NULL) {
            Printf("%d ", temp->data);
            temp = temp->next;
        }
        Printf("\n");
    }
}

int main() {
    int choice, value, position;

    do {
        Printf("\nMenu:\n");
        Printf("1. Insert at Beginning\n");
        Printf("2. Insert at End\n");
        Printf("3. Insert at a Position\n");
```

```
Printf("4. Delete from Beginning\n");
Printf("5. Delete from End\n");
Printf("6. Delete at a Position\n");
Printf("7. Search for an Element\n");
Printf("8. Display the List\n");
Printf("9. Exit\n");
Printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
   case 1:
      Printf("Enter the element: ");
      scanf("%d", &value);
      insertAtBeginning(value);
      break;
   case 2:
      Printf("Enter the element: ");
      scanf("%d", &value);
      insertAtEnd(value);
      break;
   case 3:
      Printf("Enter the element: ");
      scanf("%d", &value);
      Printf("Enter the position: ");
```

```
    scanf("%d", &position);
    insertAtPosition(value, position);
    break;
case 4:
    deleteFromBeginning();
    break;
case 5:
    deleteFromEnd();
    break;
case 6:
    Printf("Enter the position: ");
    scanf("%d", &position);
    deleteAtPosition(position);
    break;
case 7:
    Printf("Enter the element to search: ");
    scanf("%d", &value);
    search(value);
    break;
case 8:
    display();
    break;
case 9:
    Printf("Exiting program\n");
```

```
            break;
        default:
            Printf("Invalid choice! Please try again.\n");
    }
} while (choice != 9);

    return 0;
}
```

TESTCASES:

```
PS C:\Users\rijul\OneDrive\Desktop\c programs> cd "c:\Users\rijul\OneDrive\Desktop\c programs\" ; if ($?) { gcc singlelinkedli
sts.c -o singlelinkedlists } ; if ($?) { .\singlelinkedlists }

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 1
Enter the element: 10
Element inserted at the beginning

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 1
Enter the element: 23
Element inserted at the beginning
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 2
Enter the element: 15
Element inserted at the end
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 8
Elements in the list: 23 10 15

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 3
Enter the element: 3
Enter the position: 3
Element inserted at position 3
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 4
Element deleted from the beginning

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 8
Elements in the list: 10 3 15
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 1
Enter the element: 18
Element inserted at the beginning

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 7
Enter the element to search: 3
Element 3 found at position 3
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 7
Enter the element to search: 10
Element 10 found at position 2

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 7
Enter the element to search: 40
Element 40 not found in the list
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 3
Enter the element: 6
Enter the position: 8
Invalid position

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 8
Elements in the list: 18 10 3 15
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 6
Enter the position: 6
Invalid position

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 8
Elements in the list: 18 10 3 15
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 6
Enter the position: 3
Element deleted from position 3

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 8
Elements in the list: 18 10 15
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 5
Element deleted from the end

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 4
Element deleted from the beginning
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 4
Element deleted from the beginning

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 4
List is empty
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 8
List is empty

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 9
Exiting program
```

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

PSEUDOCODE:

```
BEGIN
  Define NODE structure:
    DATA: to store the value
    NEXT: to store the address of the next node
    PREV: to store the address of the previous node
  Initialize HEAD = NULL (empty list)

  FUNCTION INSERT_BEGINNING(ELEMENT):
    Create a new NODE
    SET NODE.DATA = ELEMENT
    SET NODE.PREV = NULL
    SET NODE.NEXT = HEAD
    IF HEAD != NULL Then
      HEAD.PREV = NODE
    END IF
    HEAD = NODE
    Print "Element inserted at the beginning"
```

```
FUNCTION INSERT_END(ELEMENT):
    Create a new NODE
    SET NODE.DATA = ELEMENT
    SET NODE.NEXT = NULL
    IF HEAD == NULL Then
        SET NODE.PREV = NULL
        HEAD = NODE
    ELSE
        SET TEMP = HEAD
        WHILE TEMP.NEXT != NULL DO
            TEMP = TEMP.NEXT
        END WHILE
        TEMP.NEXT = NODE
        NODE.PREV = TEMP
    END IF
    Print "Element inserted at the end"

FUNCTION INSERT_AT_POSITION(ELEMENT, POSITION):
    Create a new NODE
    SET NODE.DATA = ELEMENT
    IF POSITION == 1 THEN
        CALL INSERT_BEGINNING(ELEMENT)
    ELSE
```

```
        SET TEMP = HEAD
        FOR i = 1 TO POSITION - 2 DO
            IF TEMP == NULL Then
                Print "Invalid position"
                RETURN
            END IF
            TEMP = TEMP.NEXT
        END FOR
        IF TEMP == NULL Then
            Print "Invalid position"
            RETURN
        END IF
        SET NODE.NEXT = TEMP.NEXT
        SET NODE.PREV = TEMP
        IF TEMP.NEXT != NULL Then
            TEMP.NEXT.PREV = NODE
        END IF
        TEMP.NEXT = NODE
    END IF
    Print "Element inserted at position ", POSITION

FUNCTION DELETE_BEGINNING():
    IF HEAD == NULL Then
        Print "List is empty"
```

```
    ELSE
        SET TEMP = HEAD
        IF HEAD.NEXT != NULL Then
            HEAD.NEXT.PREV = NULL
        END IF
        HEAD = HEAD.NEXT
        FREE TEMP
        Print "Element deleted from the beginning"
    END IF

FUNCTION DELETE_END():
    IF HEAD == NULL Then
        Print "List is empty"
    ELSE IF HEAD.NEXT == NULL Then
        FREE HEAD
        HEAD = NULL
    ELSE
        SET TEMP = HEAD
        WHILE TEMP.NEXT != NULL DO
            TEMP = TEMP.NEXT
        END WHILE
        TEMP.PREV.NEXT = NULL
        FREE TEMP
    END IF
```

```
    Print "Element deleted from the end"


FUNCTION DELETE_AT_POSITION(POSITION):
    IF HEAD == NULL Then
        Print "List is empty"
    ELSE IF POSITION == 1 THEN
        CALL DELETE_BEGINNING()
    ELSE
        SET TEMP = HEAD
        FOR i = 1 TO POSITION - 1 DO
            IF TEMP == NULL Then
                Print "Invalid position"
                RETURN
            END IF
            TEMP = TEMP.NEXT
        END FOR
        IF TEMP == NULL Then
            PRINT "Invalid position"
            RETURN
        END IF
        IF TEMP.NEXT != NULL Then
            TEMP.NEXT.PREV = TEMP.PREV
        END IF
        IF TEMP.PREV != NULL Then
```

```
            TEMP.PREV.NEXT = TEMP.NEXT
        END IF
        FREE TEMP
        Print "Element deleted at position ", POSITION
    END IF

FUNCTION SEARCH(ELEMENT):
    SET TEMP = HEAD
    SET POSITION = 1
    WHILE TEMP != NULL DO
        IF TEMP.DATA == ELEMENT Then
            Print "Element ", ELEMENT, " found at position ", POSITION
            RETURN
        END IF
        TEMP = TEMP.NEXT
        POSITION = POSITION + 1
    END WHILE
    Print "Element ", ELEMENT, " not found in the list"

FUNCTION DISPLAY():
    IF HEAD == NULL Then
        Print "List is empty"
    ELSE
        SET TEMP = HEAD
```

```
        Print "Elements in the list: "
        WHILE TEMP != NULL DO
            Print TEMP.DATA
            TEMP = TEMP.NEXT
        END WHILE
    END IF

REPEAT
    Print "Menu:"
    Print "1. INSERT at Beginning"
    Print "2. INSERT at End"
    Print "3. INSERT at a Position"
    Print "4. DELETE from Beginning"
    Print "5. DELETE from End"
    Print "6. DELETE at a Position"
    Print "7. SEARCH for an Element"
    Print "8. DISPLAY the List"
    Print "9. EXIT"
    Print "Enter your choice: "
    READ CHOICE

    SWITCH (CHOICE)
        CASE 1:
            Print "Enter the element: "
```

```
    READ ELEMENT
    CALL INSERT_BEGINNING(ELEMENT)
CASE 2:
    Print "Enter the element: "
    READ ELEMENT
    CALL INSERT_END(ELEMENT)
CASE 3:
    Print "Enter the element: "
    READ ELEMENT
    Print "Enter the position: "
    READ POSITION
    CALL INSERT_AT_POSITION(ELEMENT, POSITION)
CASE 4:
    CALL DELETE_BEGINNING()
CASE 5:
    CALL DELETE_END()
CASE 6:
    Print "Enter the position: "
    READ POSITION
    CALL DELETE_AT_POSITION(POSITION)
CASE 7:
    Print "Enter the element to search: "
    READ ELEMENT
    CALL SEARCH(ELEMENT)
```

```
        CASE 8:
            CALL DISPLAY()
        CASE 9:
            Print "Exiting program"
            EXIT
        DEFAULT:
            Print "Invalid choice! Please try again."
    END SWITCH
  UNTIL FALSE
END
```

CODE:

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

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

struct Node* head = NULL;
```

```
void insertAtBeginning(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
   newNode->data = value;
   newNode->prev = NULL;
   newNode->next = head;
   if (head != NULL) {
      head->prev = newNode;
   }
   head = newNode;
   printf("Element inserted at the beginning\n");
}

void insertAtEnd(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
   newNode->data = value;
   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;
    }
    printf("Element inserted at the end\n");
}

void insertAtPosition(int value, int position) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    newNode->data = value;

    if (position == 1) {
        insertAtBeginning(value);
        return;
    }

    struct Node* temp = head;
    for (int i = 1; i < position - 1; i++) {
        if (temp == NULL) {
            printf("Invalid position\n");
            free(newNode);
```

```
            return;
        }
        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;
    printf("Element inserted at position %d\n", position);
}

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);
    printf("Element deleted from the beginning\n");
}

void deleteFromEnd() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    if (head->next == NULL) {
        free(head);
        head = NULL;
        printf("Element deleted from the end\n");
        return;
    }
```

```
    struct Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->prev->next = NULL;
    free(temp);
    printf("Element deleted from the end\n");
}

void deleteAtPosition(int position) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    if (position == 1) {
        deleteFromBeginning();
        return;
    }

    struct Node* temp = head;
    for (int i = 1; i < position; i++) {
        if (temp == NULL) {
            printf("Invalid position\n");
```

```
            return;
        }
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Invalid position\n");
        return;
    }

    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }
    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    }
    free(temp);
    printf("Element deleted at position %d\n", position);
}

void search(int value) {
    struct Node* temp = head;
    int position = 1;
```

```
    while (temp != NULL) {
        if (temp->data == value) {
            printf("Element %d found at position %d\n", value, position);
            return;
        }
        temp = temp->next;
        position++;
    }
    printf("Element %d not found in the list\n", value);
}

void display() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* temp = head;
    printf("Elements in the list: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
```

```
}

int main() {
    int choice, value, position;

    do {
        printf("\nMenu:\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Insert at a Position\n");
        printf("4. Delete from Beginning\n");
        printf("5. Delete from End\n");
        printf("6. Delete at a Position\n");
        printf("7. Search for an Element\n");
        printf("8. Display the List\n");
        printf("9. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the element: ");
                scanf("%d", &value);
                insertAtBeginning(value);
```

```
            break;
        case 2:
            printf("Enter the element: ");
            scanf("%d", &value);
            insertAtEnd(value);
            break;
        case 3:
            printf("Enter the element: ");
            scanf("%d", &value);
            printf("Enter the position: ");
            scanf("%d", &position);
            insertAtPosition(value, position);
            break;
        case 4:
            deleteFromBeginning();
            break;
        case 5:
            deleteFromEnd();
            break;
        case 6:
            printf("Enter the position: ");
            scanf("%d", &position);
            deleteAtPosition(position);
            break;
```

```
            case 7:
                printf("Enter the element to search: ");
                scanf("%d", &value);
                search(value);
                break;
            case 8:
                display();
                break;
            case 9:
                printf("Exiting program\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 9);

    return 0;
}
```

TESTCASE:

```
PS C:\Users\rijul\OneDrive\Desktop\c programs> cd "c:\Users\rijul\OneDrive\Desktop\c programs\" ; if ($?) { gcc doublelinkedli
sts.c -o doublelinkedlists } ; if ($?) { .\doublelinkedlists }

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 1
Enter the element: 14
Element inserted at the beginning

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 1
Enter the element: 67
Element inserted at the beginning
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 1
Enter the element: 98
Element inserted at the beginning

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 3
Enter the element: 45
Enter the position: 2
Element inserted at position 2
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 3
Enter the element: 88
Enter the position: 3
Element inserted at position 3

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 8
Elements in the list: 98 45 88 67 14
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 7
Enter the element to search: 45
Element 45 found at position 2

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 5
Element deleted from the end
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 4
Element deleted from the beginning

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 8
Elements in the list: 45 88 67
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 6
Enter the position: 3
Element deleted at position 3

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 3
Enter the element: 66
Enter the position: 7
Invalid position
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 6
Enter the position: 8
Invalid position

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 4
Element deleted from the beginning
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 5
Element deleted from the end

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 4
List is empty
```

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 8
List is empty

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
```

```
5. Delete from End
6. Delete at a Position
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 7
Enter the element to search: 23
Element 23 not found in the list

Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete at a Position
7. Search for an Element
8. Display the List
9. Exit
Enter your choice: 9
Exiting program
```