Assessment- 1

Name-Pranav Dwivedi
Registration No.: 24BBS0109

Q1. Write a menu driven program to implement the following operations on stack:
a. PUSH()
b. POP()
c. Display()

A1.

PSEUDO CODE:

1. Start
2. Initialize stack[] and top = -1
3. Display menu options:
   1: PUSH 2: POP 3: DISPLAY 4: EXIT
4. **If** user selects PUSH:
   - Check if top == MAX-1 (stack overflow)
   - **If** stack is not full, increment top and insert the element into stack[top]
5. **If** user selects POP:
   - Check if top == -1 (stack underflow)
   - **If** stack is not empty, display and remove the element from stack[top], decrement top
6. **If** user selects DISPLAY:
   - Check if top == -1 (stack is empty)
   - **If** not empty, display all elements from stack[0] to stack[top]
7. Repeat steps 3-6 until the user chooses EXIT
8. **End**

CODE :

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

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

void push() {
    int value;
    if (top == MAX - 1) {
        printf("Stack Overflow\n");
    } else {
        printf("Enter value to push: ");
        scanf("%d", &value);
        top++;
        stack[top] = value;
        printf("Value pushed successfully\n");
    }
}

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

```
void display() {
    if (top == -1) {
        printf("Stack is empty\n");
    } else {
        printf("Stack elements: \n");
        for (int i = 0; i <= top; i++) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}

int main() {
    int choice;
    while (1) {
        printf("\n1. Push\n2. Pop\n3. Display\n4. 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: return 0;
            default: printf("Invalid choice\n");
        }
    }
    return 0;
}
```

OUTPUT:

Input:

```
1. PUSH
2. POP
3. DISPLAY
4. EXIT

Enter your choice: 1
Enter value to push: 10

Enter your choice: 1
Enter value to push: 20

Enter your choice: 3
```

Output:

```
Pushed 10
Pushed 20
Stack: 20 10
```

Input:

```
Enter your choice: 2
```

Output:

```
Popped value: 20
```

Input:

```
Enter your choice: 3
```

Output:

```
Stack: 10
```

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

A2.

PSEUDO CODE:

1. Start
2. Initialise queue[], front = -1, rear = -1
3. Display menu options:
   - 1: ENQUEUE
   - 2: DEQUEUE
   - 3: DISPLAY
   - 4: EXIT
4. **If** user selects ENQUEUE:
   - Check if rear == MAX-1 (queue overflow)
   - **If** queue is not full, increment rear and insert the element into queue[rear]
   - If front == -1, set front = 0
5. **If** user selects DEQUEUE:
   - Check if front == -1 or front > rear (queue underflow)
   - **If** queue is not empty, display and remove the element from queue[front], increment front
6. **If** user selects DISPLAY:
   - Check if front == -1 or front > rear (queue is empty)
   - **If** not empty, display all elements from queue[front] to queue[rear]
7. Repeat steps 3-6 until the user chooses EXIT
8. **End**

CODE

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

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

void enqueue() {
    int value;
    if (rear == MAX - 1) {
        printf("Queue Overflow\n");
    } else {
        printf("Enter value to enqueue: ");
        scanf("%d", &value);
        if (front == -1) front = 0;
        rear++;
        queue[rear] = value;
        printf("Value enqueued successfully\n");
    }
}

void dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue Underflow\n");
    } else {
```

```
        printf("Dequeued value: %d\n", queue[front]);
        front++;
    }
}

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

int main() {
    int choice;
    while (1) {
        printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. 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: return 0;
            default: printf("Invalid choice\n");
        }
    }
    return 0;
}
```

OUTPUT :

Input:

```
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT

Enter your choice: 1
Enter value to enqueue: 5

Enter your choice: 1
Enter value to enqueue: 10

Enter your choice: 3
```

Output:

```
Enqueued 5
Enqueued 10
Queue: 5 10
```

Input:

```
Enter your choice: 2
```

Output:

```
Dequeued value: 5
```

Input:

```
Enter your choice: 3
```

Output:

```
Queue: 10
```

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

A3.
PSEUDO CODE :
1. Start
2. Initialize queue[], front = -1, rear = -1
3. Display menu options:
   - 1: ENQUEUE
   - 2: DEQUEUE
   - 3: DISPLAY
   - 4: EXIT
4. **If** user selects ENQUEUE:
   - Check if the queue is full ((front == 0 && rear == MAX - 1) or (rear == front-1) for circular condition)
   - If not full, increment rear in a circular manner and insert element into queue[rear]
   - If front == -1, set front = 0
5. **If** user selects DEQUEUE:
   - Check if the queue is empty (front == -1)
   - Display and remove element from queue[front] and update front in a circular manner
6. **If** user selects DISPLAY:
   - Check if queue is empty (front == -1)
   - Display all elements from queue[front] to queue[rear]
7. Repeat until EXIT is selected
8. **End**

CODE:

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

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

void enqueue() {
    int value;
    if ((front == 0 && rear == MAX - 1) || (rear == (front - 1) % (MAX - 1))) {
        printf("Queue Overflow\n");
    } else {
        printf("Enter value to enqueue: ");
        scanf("%d", &value);
        if (front == -1) {
```

```
            front = rear = 0;
        } else if (rear == MAX - 1 && front != 0) {
            rear = 0;
        } else {
            rear++;
        }
        queue[rear] = value;
        printf("Value enqueued successfully\n");
    }
}

void dequeue() {
    if (front == -1) {
        printf("Queue Underflow\n");
    } else {
        printf("Dequeued value: %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 {
        printf("Queue elements: \n");
        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");
    }
}

int main() {
    int choice;
    while (1) {
        printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. 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: return 0;
            default: printf("Invalid choice\n");
        }
    }
    return 0;
}
```

OUTPUT :

Input:

```
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT

Enter your choice: 1
Enter value to enqueue: 15

Enter your choice: 1
Enter value to enqueue: 25

Enter your choice: 3
```

Output:

```
Enqueued 15
Enqueued 25
Queue: 15 25
```

Input:

```
Enter your choice: 2
```

Output:

```
Dequeued value: 15
```

Input:

```
Enter your choice: 3
```

Output:

```
Queue: 25
```

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

A4.
PSEUDO CODE :

1. Start
2. Initialize head = NULL
3. Repeat until Exit:
- Display menu
- If Insert at Beginning:
    - Create new node
    - Set newNode->next = head
    - Set head = newNode
- If Insert at End:
    - Create new node
    - Traverse to last node
    - Set lastNode->next = newNode
- If Insert at Position:
    - Traverse to (position-1) node

- Set newNode->next = prevNode->next
- Set prevNode->next = newNode

- If Delete from Beginning:
  - Set temp = head
  - Set head = head->next
  - Free temp
- If Delete from End:
  - Traverse to second-last node
  - Set secondLastNode->next = NULL
  - Free last node
- If Delete from Position:
  - Traverse to (position-1) node
  - Set prevNode->next = targetNode->next
  - Free target node
- If Search:
  - Traverse list and compare node values
- If Display:
  - Print all node values

4. 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("Node 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("Node inserted at the end\n");
}

void insertAtPosition(int value, int pos) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    if (pos == 1) {
        newNode->next = head;
        head = newNode;
        return;
    }
```

```
    struct Node* temp = head;
    for (int i = 1; i < pos - 1; i++) {
        if (temp != NULL) temp = temp->next;
    }
    newNode->next = temp->next;
    temp->next = newNode;
    printf("Node inserted at position %d\n", pos);
}

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

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

void deleteAtPosition(int pos) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    if (pos == 1) {
        head = head->next;
        free(temp);
        return;
    }
    for (int i = 1; i < pos - 1; i++) {
        if (temp != NULL) temp = temp->next;
    }
    struct Node* deleteNode = temp->next;
    temp->next = temp->next->next;
    free(deleteNode);
    printf("Node deleted from position %d\n", pos);
}

void search(int value) {
    struct Node* temp = head;
    int pos = 1;
    while (temp != NULL) {
        if (temp->data == value) {
            printf("Element found at position %d\n", pos);
            return;
```

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

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

int main() {
    int choice, value, pos;
    while (1) {
        printf("\n1. 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\n9. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value: ");
                scanf("%d", &value);
                insertAtBeginning(value);
                break;
            case 2:
                printf("Enter value: ");
                scanf("%d", &value);
                insertAtEnd(value);
                break;
            case 3:
                printf("Enter value and position: ");
                scanf("%d%d", &value, &pos);
                insertAtPosition(value, pos);
                break;
            case 4: deleteAtBeginning(); break;
            case 5: deleteAtEnd(); break;
            case 6:
                printf("Enter position: ");
                scanf("%d", &pos);
                deleteAtPosition(pos);
                break;
            case 7:
                printf("Enter value to search: ");
                scanf("%d", &value);
                search(value);
                break;
            case 8: display(); break;
            case 9: return 0;
            default: printf("Invalid choice\n");
        }
    }
}
```

OUTPUT :

Input:

```
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
9. Exit

Enter your choice: 1
Enter value to insert at beginning: 30

Enter your choice: 2
Enter value to insert at end: 50

Enter your choice: 8
```

Output:

```
Node inserted at the beginning
Node inserted at the end
Linked List: 30 -> 50 -> NULL
```

Input:

```
Enter your choice: 4
```

Output:

```
Deleted node from the beginning
```

Input:

```
Enter your choice: 8
```

Output:

```
Linked List: 50 -> NULL
```

Input:

```
Enter your choice: 7
Enter value to search: 50
```

Output:

```
Element found at position 1
```

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

A5.
PSEUDO CODE:

1. Start
2. Initialize head = NULL
3. Repeat until Exit:
- Display menu
- If Insert at Beginning:
    - Create new node
    - Set newNode->next = head
    - Set head->prev = newNode
    - Set head = newNode
- If Insert at End:
    - Create new node
    - Traverse to last node

        - Set lastNode->next = newNode
        - Set newNode->prev = lastNode
- If Insert at Position:
- Traverse to (position-1) node
    - Set newNode->next = prevNode->next
    - Set prevNode->next = newNode
    - Set newNode->prev = prevNode
    - Set nextNode->prev = newNode
- If Delete from Beginning:
    - Set temp = head
    - Set head = head->next
    - Set head->prev = NULL
    - Free temp
- If Delete from End:
    - Traverse to last node
    - Set secondLastNode->next = NULL
    - Free last node
- If Delete from Position:
    - Traverse to (position-1) node
    - Set prevNode->next = targetNode->next
    - Set nextNode->prev = prevNode
    - Free target node
- If Search:
    - Traverse list and compare node values
- If Display:
    - Print all node values

4. 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->next = head;
    newNode->prev = NULL;
    if (head != NULL) {
        head->prev = newNode;
    }
    head = newNode;
    printf("Node 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("Node inserted at the end\n");
}

void insertAtPosition(int value, int pos) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    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; i++) {
        temp = temp->next;
    }
    newNode->next = temp->next;
    newNode->prev = temp;
    if (temp->next != NULL) {
        temp->next->prev = newNode;
    }
    temp->next = newNode;
    printf("Node inserted at position %d\n", pos);
}

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

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

void deleteAtPosition(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 - 1; i++) {
        temp = temp->next;
    }
    struct Node* deleteNode = temp->next;
    temp->next = temp->next->next;
    if (temp->next != NULL) {
        temp->next->prev = temp;
    }
    free(deleteNode);
    printf("Node deleted from position %d\n", pos);
}

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

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

int main() {
    int choice, value, pos;
    while (1) {
        printf("\n1. 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\n9. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value: ");
                scanf("%d", &value);
                insertAtBeginning(value);
                break;
            case 2:
```

```
            printf("Enter value: ");
            scanf("%d", &value);
            insertAtEnd(value);
            break;
        case 3:
            printf("Enter value and position: ");
            scanf("%d%d", &value, &pos);
            insertAtPosition(value, pos);
            break;
        case 4: deleteAtBeginning(); break;
        case 5: deleteAtEnd(); break;
        case 6:
            printf("Enter position: ");
            scanf("%d", &pos);
            deleteAtPosition(pos);
            break;
        case 7:
            printf("Enter value to search: ");
            scanf("%d", &value);
            search(value);
            break;
        case 8: display(); break;
        case 9: return 0;
        default: printf("Invalid choice\n");
    }
  }
}
```

OUTPUT :

Input:

```
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
9. Exit

Enter your choice: 1
Enter value to insert at beginning: 100

Enter your choice: 2
Enter value to insert at end: 200

Enter your choice: 8
```

Output:

```
Node inserted at the beginning
Node inserted at the end
Doubly Linked List: 100 <-> 200 <-> NULL
```

Input:

```
Enter your choice: 4
```

Output:

```
Deleted node from the beginning
```

Input:

```
Enter your choice: 8
```

Output:

```
Doubly Linked List: 200 <-> NULL
```

Input:

```
Enter your choice: 7
Enter value to search: 200
```

Output:

```
Element found at position 1
```