Assessment 1

Regno.- 24BBS0213

Name – Deepujjwal Singh

Question 1- STACK

Pseudocode:

1)Initialize stack with a maximum size and set top = -1.

2) Define push(value):

- Check if top == SIZE - 1, print "Overflow" if true.
- Otherwise, increment top and store value at stack[top].

3) Define pop():

- Check if top == -1, print "Underflow" if true.
- Otherwise, print and decrement top.

4) Define display():

- If top == -1, print "Empty stack".
- Otherwise, iterate from top to 0 and print elements.

5) In main, provide menu options: Push, Pop, Display, Exit.

Code-

```
#include <stdio.h>
#define SIZE 5
int stack[SIZE], top = -1;

void push(int value) {
    if (top == SIZE - 1)
        printf("Stack Overflow\n");
    else
        stack[++top] = value;
}

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

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

```
int main() {
    int choice, value;
    do {
        printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice\n");
        }
    } while (choice != 4);
    return 0;
}
```

Test Cases:

Test 1-

Input: 1 (Push), 10; 1 (Push), 20; 3 (Display);4(Exit)
Output: Stack elements: 20 10

```
PS D:\Python codes> cd "d:\Python codes\DSAcodes\DSA DA\" ;
 if ($?) { gcc Stack.c -o Stack } ; if ($?) { .\Stack }
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter value to push: 10
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter value to push: 20
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 3
Stack elements:
20
10
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 4
Exiting...
PS D:\Python codes\DSAcodes\DSA DA>
```

Test 2 –

Input: 1 (Push), 5; 2 (Pop); 2 (Pop)
Output: Popped: 5; Stack Underflow

```
PS D:\Python codes> cd "d:\Python codes\DSAcodes\DSA DA\" ;
 if ($?) { gcc Stack.c -o Stack } ; if ($?) { .\Stack }
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter value to push: 5
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 2
Popped: 5
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 2
Stack Underflow
```

Test case 3-Input: 1 (Push), 10 (5 times)

Output: Stack Overflow

```
PS D:\Python codes> cd "d:\Python codes\DSAcodes\DSA DA\" ;
 if ($?) { gcc Stack.c -o Stack } ; if ($?) { .\Stack }
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter value to push: 10
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter value to push: 10
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter value to push: 10
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter value to push: 10
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter value to push: 10
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter value to push: 10
Stack Overflow
```

Question 2 – Queue

Pseudocode-

1)Initialize queue with front = -1 and rear = -1.

2) Define enqueue(value):

- Check if rear == SIZE - 1, print "Overflow" if true.
- Otherwise, increment rear, set queue[rear] = value.
- If front == -1, set front = 0.

3) Define dequeue ():

- Check if front > rear or front == -1, print "Underflow" if true.
- Otherwise, print queue[front] and increment front.

4) Define display ():

- If queue is empty, print "Empty".
- Otherwise, iterate from front to rear and print elements.

5) In main, provide menu options: Enqueue, Dequeue, Display, Exit.

CODE-

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

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

```
        printf("\n");
    }
}

int main() {
    int choice, value;
    while (1) {
        printf("\nQueue Menu:\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d", &value);
                enqueue(value);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display_queue();
                break;
            case 4:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

Test Cases-

TEST 1:

Input: 1 (Enqueue), 5; 1 (Enqueue), 10; 3 (Display)
Output: Queue elements: 5 10

```
PS D:\Python codes> cd "d:\Python codes\DSAcodes\DSA DA\" ;
 if ($?) { gcc tempCodeRunnerFile.c -o tempCodeRunnerFile }
 ; if ($?) { .\tempCodeRunnerFile }

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 5
5 enqueued to queue

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 10
10 enqueued to queue

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 5 10
```

TEST 2:

Input: 1 (Enqueue), 15; 2 (Dequeue); 3 (Display)
Output: Dequeued: 15; Queue is empty

```
PS D:\Python codes\DSAcodes\DSA DA> cd "d:\Python codes\DSA
codes\DSA DA\" ; if ($?) { gcc tempCodeRunnerFile.c -o temp
CodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 15
15 enqueued to queue

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued: 15

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

TEST 3:

Input: 1 (Enqueue), 15; 1 (Enqueue), 45; 2 (Dequeue); 3 (Display)
Output: Dequeued: 15; Queue elements: 45

```
PS D:\Python codes> cd "d:\Python codes\DSAcodes\DSA DA\" ; if ($?) { gcc Queue.c -o Queue } ;
if ($?) { .\Queue }

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 15
15 enqueued to queue

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 45
45 enqueued to queue

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued: 15

Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 45
```

Question – Circular Queue

Pseudocode-

1. Initialize cqueue with cfront = -1 and crear = -1.
2. Define cenqueue(value):
    - Check if (crear + 1) % SIZE == cfront, print "Overflow" if true.
    - Otherwise, increment crear circularly, store value at cqueue[crear].
    - If cfront == -1, set cfront = 0.
3. Define cdequeue():
    - Check if cfront == -1, print "Underflow" if true.
    - Otherwise, print cqueue[cfront] and increment cfront circularly.
    - If cfront == crear, reset both to -1.
4. Define display():
    - If empty, print "Empty".
    - Otherwise, iterate circularly and print elements.
5. In main, provide menu options: Enqueue, Dequeue, Display, Exit.

CODE:

```
#include <stdio.h>
#define SIZE 5
int cqueue[SIZE], cfront = -1, crear = -1;
void cenqueue(int value) {
   if ((crear + 1) % SIZE == cfront) {
      printf("Circular Queue Overflow\n");
   } else { if (cfront == -1) cfront = 0;
      crear = (crear + 1) % SIZE;
      cqueue[crear] = value;
      printf("%d enqueued to circular queue\n", value);
   }
}
void cdequeue() {
   if (cfront == -1) {
      printf("Circular Queue Underflow\n");
   } else {
      printf("Dequeued: %d\n", cqueue[cfront]);
      if (cfront == crear) {
         cfront = crear = -1;
      } else {
         cfront = (cfront + 1) % SIZE;
      }
   }
}
void display_cqueue() {
   if (cfront == -1) {
      printf("Circular Queue is empty\n");
   } else {       printf("Circular Queue elements: ");
```

```
        int i = cfront;
        while (1) {
            printf("%d ", cqueue[i]);
            if (i == crear) break;
            i = (i + 1) % SIZE;
        }
        printf("\n");
    }
}
int main() {
    int choice, value;
    while (1) {
        printf("\nCircular Queue Menu:\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:  printf("Enter value to enqueue: ");
                scanf("%d", &value);
                cenqueue(value);
                break;
            case 2: cdequeue();
                break;
            case 3: display_cqueue();
                break;
            case 4: printf("Exiting...\n");
                return 0;
            default: printf("Invalid choice!\n");
        }
    }
}
```

TEST CASES:

Test 1-

Input: 1 (Enqueue), 5; 1 (Enqueue), 10; 1 (Enqueue), 15; 3 (Display)
Output: Circular Queue elements: 5 10 15

```
PS D:\Python codes> cd "d:\Python codes\DSAcodes\DSA DA\" ;
 if ($?) { gcc circularqueue.c -o circularqueue } ; if ($?)
 { .\circularqueue }

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 5
5 enqueued to circular queue

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 10
10 enqueued to circular queue

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 15
15 enqueued to circular queue

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Circular Queue elements: 5 10 15
```

TEST 2:

Input: 1 (Enqueue), 5;2 (Dequeue);2(Dequeue)
Output: Dequeued: 5, Circular Queue Underflow

```
PS D:\Python codes> cd "d:\Python codes\DSAcodes\DSA DA\" ;
 if ($?) { gcc circularqueue.c -o circularqueue } ; if ($?)
 { .\circularqueue }

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 5
5 enqueued to circular queue

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued: 5

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

TEST 3-

Input: 1 (Enqueue), 5;2 (Dequeue);1(Enqueue), 65;3(Display)
Output: Dequeued: 5, Circular Queue elements: 65

```
PS D:\Python codes> cd "d:\Python codes\DSAcodes\DSA DA\" ;
 if ($?) { gcc circularqueue.c -o circularqueue } ; if ($?)
 { .\circularqueue }

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 5
5 enqueued to circular queue

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued: 5

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 65
65 enqueued to circular queue

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Circular Queue elements: 65
```

Question 4 – Singly Linked List

Pseudocode-

1. Insert at Beginning (insert_begin):

1. Create a new node.
2. Set newNode.data = value.
3. Set newNode.next = head.
4. Update head = newNode.

2. Insert at End (insert_end):

1. Create a new node.
2. Set newNode.data = value.
3. Set newNode.next = NULL.
4. If head == NULL:
    - Set head = newNode.
5. Else:
    - Traverse to the last node using a pointer temp.
    - Set temp.next = newNode.

3. Insert at Position (insert_at_position):

1. If position == 1, call insert_begin(value).
2. Else:
    - Create a new node.
    - Traverse to position - 1 using a pointer temp.
    - If temp == NULL, print "Invalid position".
    - Else:
        - Set newNode.next = temp.next.
        - Set temp.next = newNode.

4. Delete from Beginning (delete_begin):

1. If head == NULL, print "List is empty".
2. Else:
    - Set temp = head.

- Update head = head.next.
- Free temp.

5. Delete from End (delete_end):

1. If head == NULL, print "List is empty".
2. Else if head.next == NULL:
    - Free head.
    - Set head = NULL.
3. Else:
    - Traverse to the second last node using temp.
    - Free temp.next.
    - Set temp.next = NULL.

6. Delete at Position (delete_at_position):

1. If position == 1, call delete_begin().
2. Else:
    - Traverse to position - 1 using a pointer temp.
    - If temp == NULL or temp.next == NULL, print "Invalid position".
    - Else:
        - Set toDelete = temp.next.
        - Set temp.next = toDelete.next.
        - Free toDelete.

7. Display (display_list):

1. If head == NULL, print "List is empty".
2. Else:
    - Traverse the list using temp.
    - Print temp.data for each node.

CODE:

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

struct Node {
   int data;
   struct Node* next;
};
struct Node* head = NULL;

void insert_begin(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
   newNode->data = value;
   newNode->next = head;
   head = newNode;
   printf("%d inserted at beginning\n", value);
}

void insert_end(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("%d inserted at end\n", value);
}

void insert_at_position(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 && temp != NULL; i++) {
            temp = temp->next;
        }
```

```
        if (temp == NULL) {
            printf("Invalid position\n");
        } else {
            newNode->next = temp->next;
            temp->next = newNode;
        }
    }
    printf("%d inserted at position %d\n", value, position);
}

void delete_begin() {
    if (head == NULL) {
        printf("List is empty\n");
    } else {
        struct Node* temp = head;
        head = head->next;
        printf("Deleted: %d\n", temp->data);
        free(temp);
    }
}
void delete_end() {
    if (head == NULL) {
        printf("List is empty\n");
    } else if (head->next == NULL) {
```

```
        printf("Deleted: %d\n", head->data);
        free(head);
        head = NULL;
    } else {
        struct Node* temp = head;
        while (temp->next->next != NULL) {
            temp = temp->next;
        }
        printf("Deleted: %d\n", temp->next->data);
        free(temp->next);
        temp->next = NULL;
    }
}
void delete_at_position(int position) {
    if (head == NULL) {
        printf("List is empty\n");
    } else if (position == 1) {
        struct Node* temp = head;
        head = head->next;
        printf("Deleted: %d\n", temp->data);
        free(temp);
    } else {
        struct Node* temp = head;
        struct Node* prev = NULL;
```

```
        for (int i = 1; i < position && temp != NULL; i++) {
            prev = temp;
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Invalid position\n");
        } else {
            prev->next = temp->next;
            printf("Deleted: %d\n", temp->data);
            free(temp);
        }
    }
}
void display_list() {
    if (head == NULL) {
        printf("List is empty\n");
    } else {
        struct Node* temp = head;
        printf("List elements: ");
        while (temp != NULL) {
            printf("%d ", temp->data);
            temp = temp->next;
        }
        printf("\n");
```

```
    }
}
int main() {
    int choice, value, position;
    while (1) {
        printf("\nSingly Linked List Menu:\n1. Insert at Beginning\n2. Insert at End\n3. Insert at Position\n4. Delete from Beginning\n5. Delete at Position\n6. Delete from End\n7. Display\n8. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert_begin(value);
                break;
            case 2:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert_end(value);
                break;
            case 3:
                printf("Enter value to insert and position: ");
                scanf("%d %d", &value, &position);
                insert_at_position(value, position);
```

```
            break;
        case 4:
            delete_begin();
            break;
        case 5:
            printf("Enter position to delete: ");
            scanf("%d", &position);
            delete_at_position(position);
            break;
        case 6:
            delete_end();
            break;
        case 7:
            display_list();
            break;
        case 8:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice!\n");
      }
   }
}
```

TEST CASE:

Test 1-

Input: Insert 10 at the beginning, Insert 20 at the end, Insert 15 at position 2, Display, Delete from the beginning, Display.
Output: After insertion: 10 15 20, After deletion: 15 20.

```
PS D:\Python codes> cd "d:\Python codes\DSAcodes\DSA OA\" ; if ($?) { gcc singlelinkedlist.c -o singlelinkedlist } ; if ($?) { .\singlelinkedlist }

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

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

Singly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete at Position
6. Delete from End
7. Display
8. Exit
Enter your choice: 3
Enter value to insert and position: 15 2
15 inserted at position 2
```

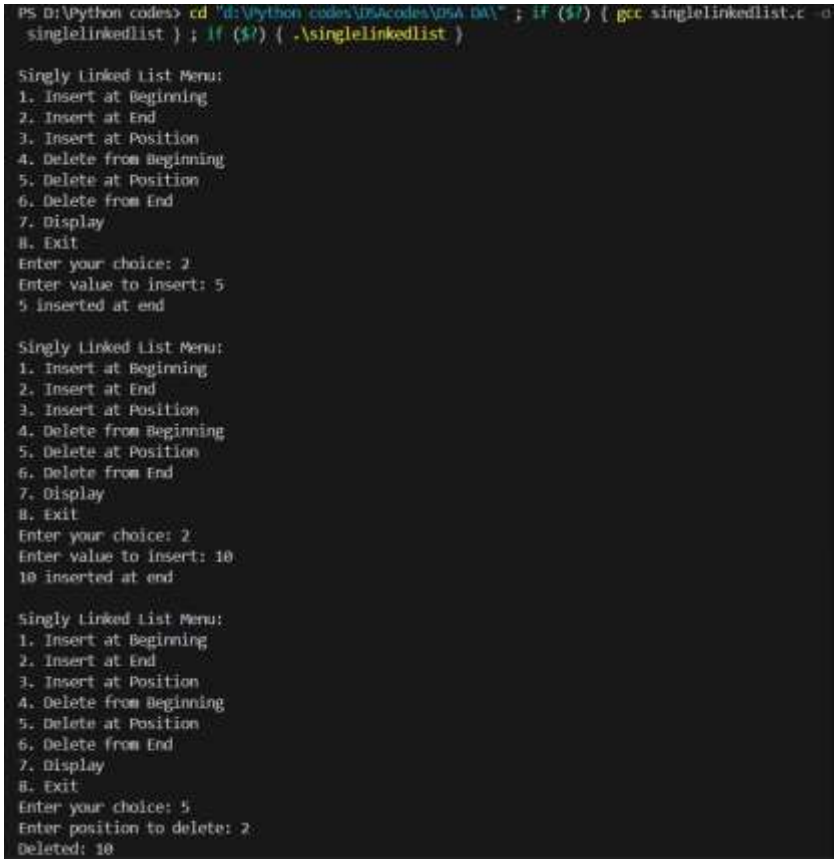
```
Singly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete at Position
6. Delete from End
7. Display
8. Exit
Enter your choice: 7
List elements: 10 15 20

Singly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete at Position
6. Delete from End
7. Display
8. Exit
Enter your choice: 4
Deleted: 10

Singly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete at Position
6. Delete from End
7. Display
8. Exit
Enter your choice: 7
List elements: 15 20
```

TEST 2:

Input: Insert 5, 10 at the end, Delete element at position 2, Display.
Output: After deleting position 3: 5



TEST 3:

Insert: 25 at position 10 (invalid position for an empty list), Delete element at position 3 (invalid position for a 2-element list), Display.
Output: Insert at position 10: Invalid position, Delete at position 3: Invalid position, Display: List is empty

```
PS D:\Python codes> cd "d:\Python codes\DSAcodes\DSA DA\" ; if ($?) { gcc singlelinkedlist.c -o
 singlelinkedlist } ; if ($?) { .\singlelinkedlist }

Singly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete at Position
6. Delete from End
7. Display
8. Exit
Enter your choice: 3
Enter value to insert and position: 25 10
Invalid position
25 inserted at position 10

Singly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete at Position
6. Delete from End
7. Display
8. Exit
Enter your choice: 5
Enter position to delete: 3
List is empty

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

QUESTION 5- Double Linked List

Pseudocode-

1. Insert at Beginning (insert_begin_dll):

1. Create a new node.
2. Set newNode.data = value.
3. Set newNode.prev = NULL and newNode.next = head.
4. If head != NULL, set head.prev = newNode.
5. Update head = newNode.

2. Insert at End (insert_end_dll):

1. Create a new node.
2. Set newNode.data = value.
3. Set newNode.next = NULL.
4. If head == NULL:
    - Set newNode.prev = NULL.
    - Update head = newNode.
5. Else:
    - Traverse to the last node using temp.
    - Set temp.next = newNode.
    - Set newNode.prev = temp.

3. Insert at Position (insert_at_position_dll):

1. If position == 1, call insert_begin_dll(value).
2. Else:
    - Create a new node.
    - Traverse to position - 1 using temp.
    - If temp == NULL, print "Invalid position".
    - Else:
        - Set newNode.next = temp.next.
        - Set newNode.prev = temp.
        - If temp.next != NULL, set temp.next.prev = newNode.
        - Set temp.next = newNode.

4. Delete from Beginning (delete_begin_dll):

1. If head == NULL, print "List is empty".
2. Else:
    - Set temp = head.
    - Update head = head.next.
    - If head != NULL, set head.prev = NULL.
    - Free temp.

5. Delete from End (delete_end_dll):

1. If head == NULL, print "List is empty".
2. Else if head.next == NULL:
    - Free head.
    - Set head = NULL.
3. Else:
    - Traverse to the last node using temp.
    - Update temp.prev.next = NULL.
    - Free temp.

6. Delete at Position (delete_at_position_dll):

1. If position == 1, call delete_begin_dll().
2. Else:
    - Traverse to position using temp.
    - If temp == NULL, print "Invalid position".
    - Else:
        - If temp.prev != NULL, set temp.prev.next = temp.next.
        - If temp.next != NULL, set temp.next.prev = temp.prev.
        - Free temp.

7. Display (display_list_dll):

1. If head == NULL, print "List is empty".
2. Else:
    - Traverse the list using temp.
    - Print temp.data for each node.

CODE-

```
#include <stdio.h>
#include <stdlib.h>
struct DNode {
   int data;
   struct DNode* prev;
   struct DNode* next;
};
struct DNode* head = NULL;
void insert_begin_dll(int value) {
   struct DNode* newNode = (struct DNode*)malloc(sizeof(struct DNode));
   newNode->data = value;
   newNode->prev = NULL;
   newNode->next = head;
   if (head != NULL) {
      head->prev = newNode;
   }
   head = newNode;
   printf("%d inserted at beginning\n", value);
}
void insert_end_dll(int value) {
   struct DNode* newNode = (struct DNode*)malloc(sizeof(struct DNode));
   newNode->data = value;
   newNode->next = NULL;
   if (head == NULL) {
      newNode->prev = NULL;
```

```
        head = newNode;
    } else {
        struct DNode* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
    printf("%d inserted at end\n", value);
}
void insert_at_position_dll(int value, int position) {
    struct DNode* newNode = (struct DNode*)malloc(sizeof(struct DNode));
    newNode->data = value;
    if (position == 1) {
        newNode->prev = NULL;
        newNode->next = head;
        if (head != NULL) {
            head->prev = newNode;
        }
        head = newNode;
    } else {
        struct DNode* temp = head;
        for (int i = 1; i < position - 1 && temp != NULL; i++) {
            temp = temp->next;
        }
```

```
        if (temp == NULL) {
            printf("Invalid position\n");
        } else {
            newNode->next = temp->next;
            newNode->prev = temp;
            if (temp->next != NULL) {
                temp->next->prev = newNode;
            }
            temp->next = newNode;
        }
    }
    printf("%d inserted at position %d\n", value, position);
}
void delete_begin_dll() {
    if (head == NULL) {
        printf("List is empty\n");
    } else {
        struct DNode* temp = head;
        head = head->next;
        if (head != NULL) {
            head->prev = NULL;
        }
        printf("Deleted: %d\n", temp->data);
        free(temp);
    }
}
```

```
void delete_end_dll() {
    if (head == NULL) {
        printf("List is empty\n");
    } else if (head->next == NULL) {
        printf("Deleted: %d\n", head->data);
        free(head);
        head = NULL;
    } else {
        struct DNode* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->prev->next = NULL;
        printf("Deleted: %d\n", temp->data);
        free(temp);
    }
}
void delete_at_position_dll(int position) {
    if (head == NULL) {
        printf("List is empty\n");
    } else if (position == 1) {
        struct DNode* temp = head;
        head = head->next;
        if (head != NULL) {
            head->prev = NULL;
        }
```

```
        printf("Deleted: %d\n", temp->data);
        free(temp);
    } else {
        struct DNode* temp = head;
        for (int i = 1; i < position && temp != NULL; i++) {
            temp = temp->next;
        }
        if (temp == NULL) {
            printf("Invalid position\n");
        } else {
            if (temp->prev != NULL) {
                temp->prev->next = temp->next;
            }
            if (temp->next != NULL) {
                temp->next->prev = temp->prev;
            }
            printf("Deleted: %d\n", temp->data);
            free(temp);
        }
    }
}
void display_list_dll() {
    if (head == NULL) {
        printf("List is empty\n");
    } else {
        struct DNode* temp = head;
```

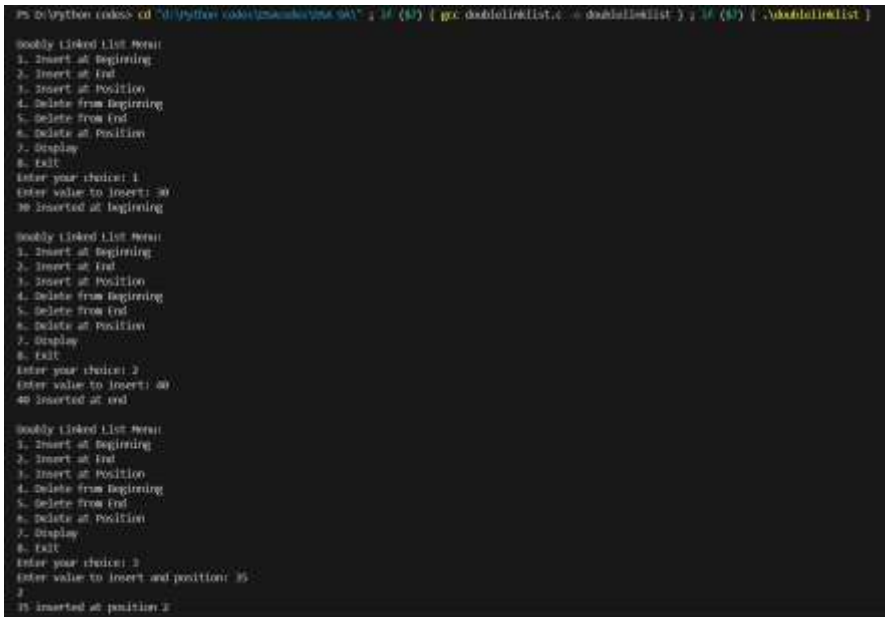
```
        printf("List elements: ");
        while (temp != NULL) {
            printf("%d ", temp->data);
            temp = temp->next;
        }
        printf("\n");
    }
}
int main() {
    int choice, value, position;
    while (1) {
        printf("\nDoubly Linked List Menu:\n1. Insert at Beginning\n2. Insert at End\n3. Insert at Position\n4. Delete from Beginning\n5. Delete from End\n6. Delete at Position\n7. Display\n8. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert_begin_dll(value);
                break;
            case 2:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert_end_dll(value);
                break;
            case 3:
```

```
            printf("Enter value to insert and position: ");
            scanf("%d %d", &value, &position);
            insert_at_position_dll(value, position);
            break;
        case 4:
            delete_begin_dll();
            break;
        case 5:
            delete_end_dll();
            break;
        case 6:
            printf("Enter position to delete: ");
            scanf("%d", &position);
            delete_at_position_dll(position);
            break;
        case 7:
            display_list_dll();
            break;
        case 8:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid choice!\n");
    }
  }
}
```

TEST CASES:

Test 1-

Input: Insert 30 at the beginning, Insert 40 at the end, Insert 35 at position 2, Display, Delete from the beginning, Display.
Output: After insertion: 30 35 40, After deletion: 35 40.



TEST 2-

Input: Insert 5, 10 at the end, Delete element at position 2, Display.
Output: After deleting position 2: 5

```
PS D:\Python codes> cd "d:\Python codes\DSAcodes

Doubly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete at Position
7. Display
8. Exit
Enter your choice: 2
Enter value to insert: 5
5 inserted at end

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

Doubly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete at Position
7. Display
8. Exit
Enter your choice: 6
Enter position to delete: 2
Deleted: 10
```

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

TEST 3-

Input: Insert 50 at position 5 (invalid position for an empty list), Delete element at position 3 (invalid position for a 1-element list), Display.
Output: Insert at position 5: Invalid position, Delete at position 3: Invalid position, Display: List is empty.

```
 if ($?) { gcc doublelinklist.c -o doublelinklist } ; if ($
?) { .\doublelinklist }

Doubly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete at Position
7. Display
8. Exit
Enter your choice: 3
Enter value to insert and position: 50 5
Invalid position
50 inserted at position 5

Doubly Linked List Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete at Position
7. Display
8. Exit
Enter your choice: 6
Enter position to delete: 3
List is empty

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