Name – ATULYA RAJ

Reg no – 24BBS0059

Subject - DSA

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

Algorithm

1. **Initialize:**
   - Define an empty stack with a fixed size.
   - Set the `top` pointer to `-1` (indicating an empty stack).
2. **Display Menu:**
   - Show the menu with the options:
     - 1. `PUSH`
     - 2. `POP`
     - 3. `Display`
     - 4. Exit
3. **Choice Input:**
   - Take the user's input to choose an operation.
4. **Perform Operation:**
   - **PUSH**:
     - Check if the stack is full (`top == maxSize - 1`).
     - If full, print "Stack Overflow".
     - Otherwise, increment `top` and add the new element at `stack[top]`.

- **POP**:
  - Check if the stack is empty (`top == -1`).
  - If empty, print "Stack Underflow".
  - Otherwise, print and remove the element at `stack[top]`, and decrement `top`.
- **Display**:
  - Check if the stack is empty (`top == -1`).
  - If empty, print "Stack is empty".
  - Otherwise, print all elements from `stack[0]` to `stack[top]`.

5. **Repeat:**
   - Loop back to step 2 until the user chooses to exit.

Pseudocode

```
BEGIN
  Initialize stack[MAX], top ← -1, maxSize ← MAX

  WHILE true DO
    PRINT "Menu:"
    PRINT "1. PUSH"
    PRINT "2. POP"
    PRINT "3. Display"
    PRINT "4. Exit"
    PRINT "Enter your choice: "
    READ choice

    SWITCH choice DO
      CASE 1:
        IF top == maxSize - 1 THEN
          PRINT "Stack Overflow"
        ELSE
```

```
        PRINT "Enter element to PUSH: "
        READ element
        top ← top + 1
        stack[top] ← element
        PRINT "Element pushed"
    ENDIF
    BREAK

CASE 2:
    IF top == -1 THEN
        PRINT "Stack Underflow"
    ELSE
        PRINT "Popped element: ", stack[top]
        top ← top - 1
    ENDIF
    BREAK

CASE 3:
    IF top == -1 THEN
        PRINT "Stack is empty"
    ELSE
        PRINT "Stack elements are: "
        FOR i ← 0 TO top DO
            PRINT stack[i]
        ENDFOR
    ENDIF
```

BREAK

CASE 4:

PRINT "Exiting..."

EXIT

DEFAULT:

PRINT "Invalid choice, please try again."

ENDSWITCH

ENDWHILE

END

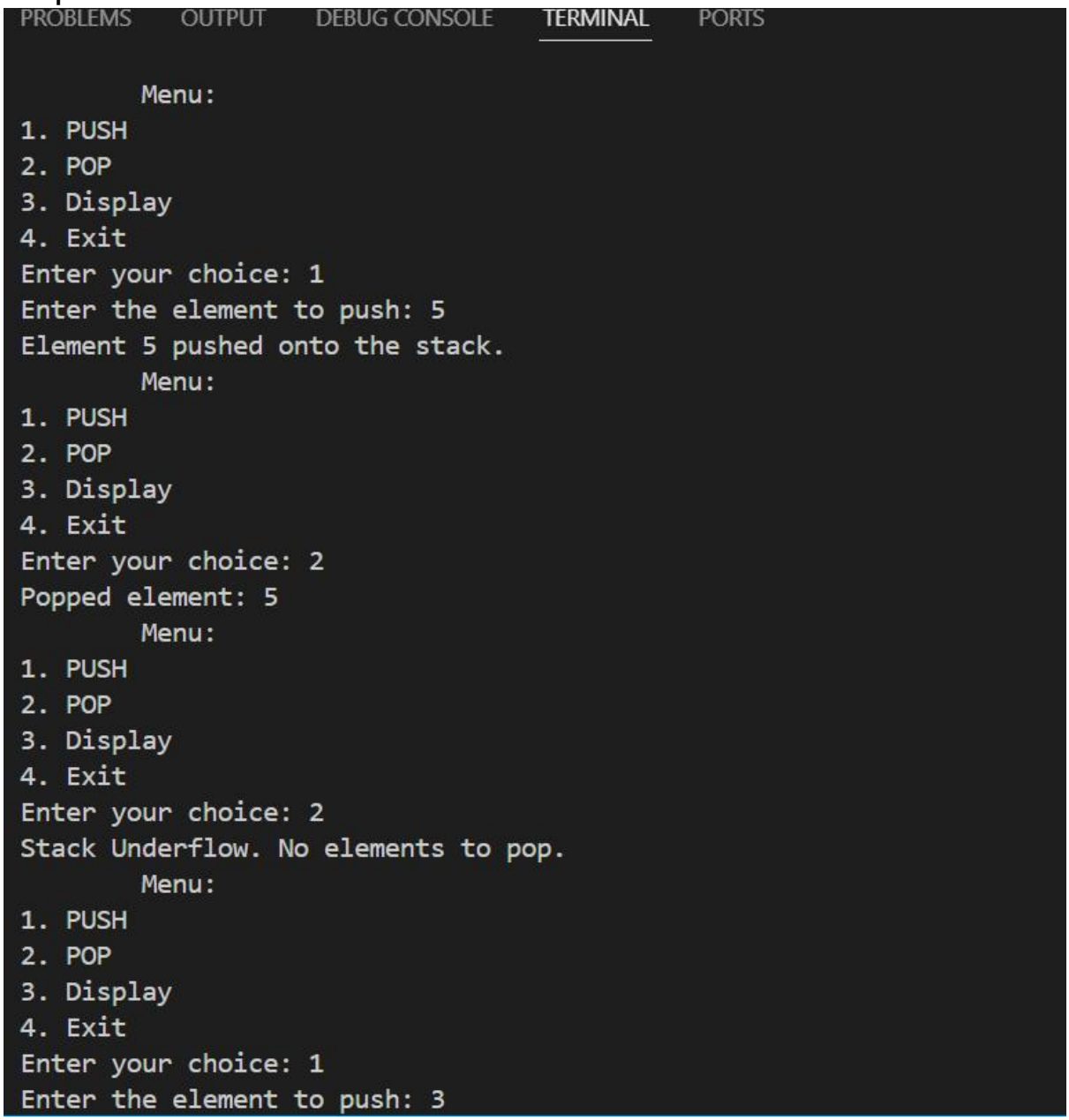
C program

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

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

void push() {
    int element;
    if (top == MAX - 1) {
        printf("Stack Overflow. Cannot push element.\n");
    } else {
        printf("Enter the element to push: ");
        scanf("%d", &element);
        top++;
        stack[top] = element;
        printf("Element %d pushed onto the stack.\n", element);
    }
}

void pop() {
```

```c
    if (top == -1) {
        printf("Stack Underflow. No elements to pop.\n");
    } else {
        printf("Popped element: %d\n", stack[top]);
        top--;
    }
}


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

int main() {
    int choice;

    do {

        printf("\tMenu:\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.\n");
        }
    } while (choice != 4);

    return 0;
}
```

Output


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Menu:
1. PUSH
2. POP
3. Display
4. Exit
Enter your choice: 1
Enter the element to push: 5
Element 5 pushed onto the stack.
Menu:
1. PUSH
2. POP
3. Display
4. Exit
Enter your choice: 2
Popped element: 5
Menu:
1. PUSH
2. POP
3. Display
4. Exit
Enter your choice: 2
Stack Underflow. No elements to pop.
Menu:
1. PUSH
2. POP
3. Display
4. Exit
Enter your choice: 1
Enter the element to push: 3

```
Enter the element to push: 3
Element 3 pushed onto the stack.
        Menu:
1. PUSH
2. POP
3. Display
4. Exit
Enter your choice: 1
Enter the element to push: 9
Element 9 pushed onto the stack.
        Menu:
1. PUSH
2. POP
3. Display
4. Exit
Enter your choice: 3
Stack elements are: 3 9
        Menu:
1. PUSH
2. POP
3. Display
4. Exit
Enter your choice: 2
Popped element: 9
        Menu:
1. PUSH
2. POP
3. Display
4. Exit
```

```
Enter your choice: 3
Stack elements are: 3
        Menu:
1. PUSH
2. POP
3. Display
4. Exit
Enter your choice: 4
Exiting program...
PS E:\DSA>
```

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

Algorithm

Initialization:

1. Define `MAX` as the maximum size of the queue.
2. Initialize:
    - `queue[MAX]` as the queue array.
    - `front = -1` and `rear = -1` as pointers to track the front and rear of the queue.

Enqueue Operation:

1. Check if the queue is full (`rear == MAX - 1`):
    - If true, print "Queue Overflow".
    - If false:
        - If the queue is empty (`front == -1`), set `front = 0`.
        - Increment `rear` by 1.
        - Insert the element at `queue[rear]`.

Dequeue Operation:

1. Check if the queue is empty (`front == -1` or `front > rear`):
    - If true, print "Queue Underflow".
    - If false:
        - Retrieve the element at `queue[front]`.
        - Increment `front` by 1.
        - If `front > rear`, reset `front = rear = -1`.

Display Operation:

1. Check if the queue is empty (`front == -1`):
    - If true, print "Queue is empty".
    - If false:

- Traverse the queue from `front` to `rear` and print each element.

Menu:

1. Present options to the user:
    - 1 for Enqueue
    - 2 for Dequeue
    - 3 for Display
    - 4 to Exit
2. Perform the selected operation until the user chooses to exit.

Pseudocode

```
BEGIN

  Initialize queue[MAX], front ← -1, rear ← -1


  WHILE true DO

    PRINT "Menu:"

    PRINT "1. Enqueue"

    PRINT "2. Dequeue"

    PRINT "3. Display"

    PRINT "4. Exit"

    PRINT "Enter your choice:"

    READ choice


    SWITCH choice DO

      CASE 1:
```

```
    IF rear == MAX - 1 THEN
        PRINT "Queue Overflow"
    ELSE
        PRINT "Enter element to enqueue:"
        READ element
        IF front == -1 THEN
            front ← 0
        ENDIF
        rear ← rear + 1
        queue[rear] ← element
        PRINT "Element enqueued"
    ENDIF
    BREAK

CASE 2:
    IF front == -1 OR front > rear THEN
        PRINT "Queue Underflow"
    ELSE
        PRINT "Dequeued element:", queue[front]
        front ← front + 1
        IF front > rear THEN
            front ← rear ← -1
```

```
        ENDIF
    ENDIF
    BREAK

  CASE 3:
    IF front == -1 THEN
        PRINT "Queue is empty"
    ELSE
        PRINT "Queue elements:"
        FOR i FROM front TO rear DO
            PRINT queue[i]
        ENDFOR
    ENDIF
    BREAK

  CASE 4:
    PRINT "Exiting program..."
    EXIT

  DEFAULT:
    PRINT "Invalid choice. Try again."
ENDSWITCH
```

ENDWHILE

END

C program

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

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


void enqueue() {
    if (rear == MAX - 1) {
        printf("Queue is Overflow\n");
        return;
    }
    int element;
    printf("Enter the element you want in queue: ");
    scanf("%d", &element);
    if (front == -1) {
        front = 0;
    }
    queue[++rear] = element;
    printf("Element enqueued successfully\n");
}


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


void display() {
    if (front == -1) {
```

```
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements: ");
    for (int i = front; i <= rear; i++) {
        printf("%d ", queue[i]);
    }
    printf("\n");
}

int main() {
    int choice;
    do {
        printf("Menu:\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.\n");
        }
    } while (choice != 4);

    return 0;
}
```

Output

```
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element you want in queue: 6
Element enqueued successfully
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element you want in queue: 10
Element enqueued successfully
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 6 10
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued element: 6
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
```

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 10
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 5
Invalid choice.
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 10
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting program...
PS E:\DSA>
```

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

Algorithm

1. Initialization:

- Define an array `queue[MAX]` to store queue elements.
- Initialize `front` and `rear` pointers to `-1` to represent an empty queue

2. Enqueue Operation:

- Check if the queue is full: `(rear + 1) % MAX == front`.
  - If full, print "Queue Overflow."
- Otherwise:
  - If the queue is initially empty (`front == -1`), set `front = rear = 0`.
  - Else, update `rear` to `(rear + 1) % MAX`.
  - Insert the new element at `queue[rear]`.

3. Dequeue Operation:

- Check if the queue is empty: `front == -1`.
  - If empty, print "Queue Underflow."
- Otherwise:
  - Retrieve the element at `queue[front]`.
  - If the queue has only one element (`front == rear`), set `front = rear = -1` (queue becomes empty).
  - Otherwise, update `front` to `(front + 1) % MAX`.

4. Display Operation:

- Check if the queue is empty: `front == -1`.
  - If empty, print "Queue is empty."
- Otherwise:
  - Start from `front` and iterate through the queue using `(index+1)% MAX` until `index == rear`.

5. Menu and User Interaction:

- Display menu options:
    - 1. Enqueue
    - 2. Dequeue
    - 3. Display
    - 4. Exit
- Repeat operations until the user chooses to exit

Pseudocode

```
BEGIN
  Initialize queue[MAX], front ← -1, rear ← -1

  WHILE true DO
    PRINT "Menu:"
    PRINT "1. Enqueue"
    PRINT "2. Dequeue"
    PRINT "3. Display"
    PRINT "4. Exit"
    PRINT "Enter your choice: "
```

```
READ choice

SWITCH choice DO
   CASE 1:
      IF (rear + 1) % MAX == front THEN
         PRINT "Queue Overflow"
      ELSE
         PRINT "Enter the element to enqueue: "
         READ element
         IF front == -1 THEN
            front ← 0
         ENDIF
         rear ← (rear + 1) % MAX
         queue[rear] ← element
         PRINT "Element enqueued."
      ENDIF
      BREAK

   CASE 2:
      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) % MAX
        ENDIF
    ENDIF
    BREAK

CASE 3:
    IF front == -1 THEN
        PRINT "Queue is empty"
    ELSE
        PRINT "Queue elements are: "
        index ← front
        WHILE true DO
            PRINT queue[index]
            IF index == rear THEN
                BREAK
            ENDIF
            index ← (index + 1) % MAX
```

ENDWHILE

ENDIF

BREAK

CASE 4:

PRINT "Exiting program..."

EXIT

DEFAULT:

PRINT "Invalid choice. Please try again."

ENDSWITCH

ENDWHILE

END

C program

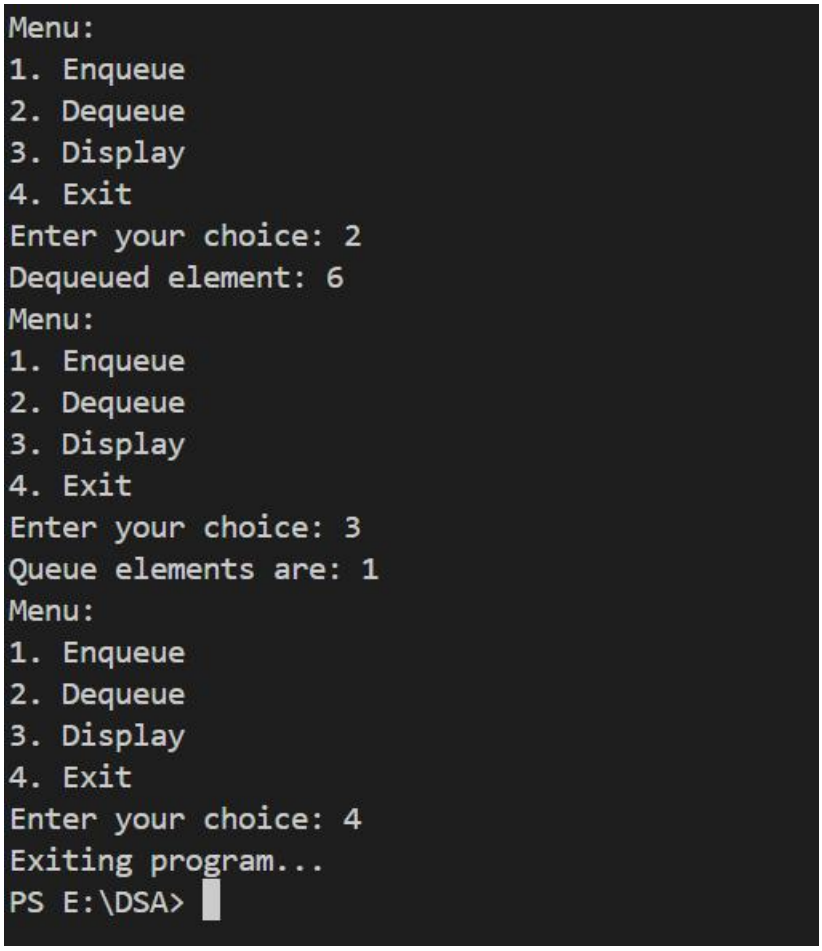
```c
#include <stdio.h>
#define MAX 5
int queue[MAX];
int front = -1, rear = -1;


void enqueue() {
    int element;
    if ((rear + 1) % MAX == front) {
        printf("Queue Overflow. Cannot enqueue element.\n");
    } else {
        printf("Enter the element to enqueue: ");
        scanf("%d", &element);
        if (front == -1) {
            front = 0;
        }
```

```
        rear = (rear + 1) % MAX;
        queue[rear] = element;
        printf("Element %d enqueued.\n", element);
    }
}


void dequeue() {
    if (front == -1) {
        printf("Queue Underflow. No elements to dequeue.\n");
    } else {
        printf("Dequeued element: %d\n", queue[front]);
        if (front == rear) {
            front = -1;
            rear = -1;
        } else {
            front = (front + 1) % MAX;
        }
    }
}

void display() {
    if (front == -1) {
        printf("Queue is empty.\n");
    } else {
        printf("Queue elements are: ");
        int i = front;
        while (1) {
            printf("%d ", queue[i]);
            if (i == rear) {
                break;
            }
            i = (i + 1) % MAX;
        }
        printf("\n");
    }
}

int main() {
    int choice;

    do {
```

```
        printf("Menu:\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;
}
```

Output

```
PS E:\DSA> .\a.exe
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 6
Element 6 enqueued.
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the element to enqueue: 1
Element 1 enqueued.
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 25
Invalid choice. Please try again.
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements are: 6 1
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
```


Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued element: 6
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements are: 1
Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting program...
PS E:\DSA>

4. 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

Algorithm

Initialization:

1. Define a structure `Node` with fields:
   - `data`: to store the value of the node.
   - `next`: a pointer to the next node.
2. Initialize `head = NULL`.

a. Insertion Operations:

i. Insert at Beginning:

1. Read the value to insert.
2. Create a new node and assign its `data` with the value.
3. Set the new node's `next` to point to `head`.
4. Update `head` to point to the new node.

ii. Insert at End:

1. Read the value to insert.
2. Create a new node and assign its `data` with the value.
3. If `head` is `NULL`, set `head` to the new node.
4. Otherwise, traverse the list until the last node.
5. Set the last node's `next` to the new node.

iii. Insert at a Specific Position:

1. Read the value and position.
2. Create a new node and assign its `data` with the value.
3. If the position is `1`, update the new node's `next` to `head` and set `head` to the new node.
4. Otherwise, traverse to the `(position - 1)` th node.

5. Update the new node's `next` to point to the current node at the position.
6. Update the `(position - 1)` th node's `next` to the new node.

b. Deletion Operations:

i. Delete from Beginning:

1. If `head` is `NULL`, print "List is empty".
2. Otherwise, store `head` in a temporary pointer.
3. Update `head` to the next node.
4. Free the temporary pointer.

ii. Delete from End:

1. If `head` is `NULL`, print "List is empty".
2. If `head->next` is `NULL`, free `head` and set `head = NULL`.
3. Otherwise, traverse to the second-last node.
4. Free the last node and update the second-last node's `next` to `NULL`.

iii. Delete from a Specific Position:

1. Read the position.
2. If the position is `1`, update `head` to the next node and free the old head.
3. Otherwise, traverse to the `(position - 1)` th node.
4. Store the node at the position in a temporary pointer.
5. Update the `(position - 1)` th node's `next` to skip the deleted node.

c. Search Operation:

1. Read the element to search.
2. Traverse the list while comparing the element with each node's `data`.
3. If a match is found, print the position and exit.
4. If the end of the list is reached, print "Element not found".

Pseudocode

```
BEGIN

  Initialize head ← NULL
```

```
WHILE true DO
    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 from a Position"
    PRINT "7. Search for an Element"
    PRINT "8. Exit"
    PRINT "Enter your choice:"
    READ choice

    SWITCH choice DO
        CASE 1:
            PRINT "Enter the element to insert:"
            READ data
            Create newNode with data
            newNode.next ← head
            head ← newNode
            PRINT "Element inserted at the beginning"
```

```
    BREAK


CASE 2:
    PRINT "Enter the element to insert:"
    READ data
    Create newNode with data
    IF head == NULL THEN
        head ← newNode
    ELSE
        temp ← head
        WHILE temp.next != NULL DO
            temp ← temp.next
        ENDWHILE
        temp.next ← newNode
    ENDIF
    PRINT "Element inserted at the end"
    BREAK


CASE 3:
    PRINT "Enter the element to insert:"
    READ data
    PRINT "Enter the position:"
```

```
    READ pos
    Create newNode with data
    IF pos == 1 THEN
        newNode.next ← head
        head ← newNode
    ELSE
        temp ← head
        FOR i ← 1 TO pos - 1 DO
            temp ← temp.next
        ENDFOR
        newNode.next ← temp.next
        temp.next ← newNode
    ENDIF
    PRINT "Element inserted at position", pos
    BREAK

CASE 4:
    IF head == NULL THEN
        PRINT "List is empty"
    ELSE
        temp ← head
        head ← head.next
```

```
        FREE(temp)
        PRINT "Element deleted from the beginning"
    ENDIF
    BREAK

CASE 5:
    IF head == NULL THEN
        PRINT "List is empty"
    ELSE IF head.next == NULL THEN
        FREE(head)
        head ← NULL
    ELSE
        temp ← head
        WHILE temp.next.next != NULL DO
            temp ← temp.next
        ENDWHILE
        FREE(temp.next)
        temp.next ← NULL
    ENDIF
    PRINT "Element deleted from the end"
    BREAK
```

```
CASE 6:
   PRINT "Enter the position to delete:"
   READ pos
   IF head == NULL THEN
      PRINT "List is empty"
   ELSE IF pos == 1 THEN
      temp ← head
      head ← head.next
      FREE(temp)
   ELSE
      temp ← head
      FOR i ← 1 TO pos - 1 DO
         temp ← temp.next
      ENDFOR
      toDelete ← temp.next
      temp.next ← toDelete.next
      FREE(toDelete)
   ENDIF
   PRINT "Element deleted from position", pos
   BREAK

CASE 7:
```

```
        PRINT "Enter the element to search:"
        READ element
        temp ← head
        pos ← 1
        WHILE temp != NULL DO
          IF temp.data == element THEN
            PRINT "Element found at position", pos
            BREAK
          ENDIF
          temp ← temp.next
          pos ← pos + 1
        ENDWHILE
        IF temp == NULL THEN
          PRINT "Element not found"
        ENDIF
        BREAK

      CASE 8:
        PRINT "Exiting program..."
        EXIT
DEFAULT:
        PRINT "Invalid choice. Please try again"
```

ENDSWITCH

ENDWHILE

END

C program

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

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

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

void insertBeginning() {
    int data;
    printf("Enter the element to insert: ");
    scanf("%d", &data);
    struct Node* newNode = createNode(data);
    newNode->next = head;
    head = newNode;
    printf("Element inserted at beginning.\n");
}

void insertEnd() {
    int data;
    printf("Enter the element to insert: ");
    scanf("%d", &data);
    struct Node* newNode = createNode(data);
    if (head == NULL) {
        head = newNode;
    } else {
```

```
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    printf("Element inserted at end.\n");
}

void insertAtPosition() {
    int data, pos;
    printf("Enter the element to insert: ");
    scanf("%d", &data);
    printf("Enter the position: ");
    scanf("%d", &pos);

    struct Node* newNode = createNode(data);
    if (pos == 1) {
        newNode->next = head;
        head = newNode;
        printf("Element inserted at position %d.\n", pos);
        return;
    }

    struct Node* temp = head;
    for (int i = 1; temp != NULL && i < pos - 1; i++) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Position out of range.\n");
        free(newNode);
        return;
    }

    newNode->next = temp->next;
    temp->next = newNode;
    printf("Element inserted at position %d.\n", pos);
}

void deleteBeginning() {
    if (head == NULL) {
```

```
        printf("List is empty.\n");
        return;
    }

    struct Node* temp = head;
    head = head->next;
    free(temp);
    printf("Element deleted from beginning.\n");
}

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

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

void deleteAtPosition() {
    int pos;
    printf("Enter the position to delete: ");
    scanf("%d", &pos);

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

    if (pos == 1) {
        struct Node* temp = head;
```

```
        head = head->next;
        free(temp);
        printf("Element deleted from position %d.\n", pos);
        return;
    }

    struct Node* temp = head;
    for (int i = 1; temp != NULL && i < pos - 1; i++) {
        temp = temp->next;
    }

    if (temp == NULL || temp->next == NULL) {
        printf("Position out of range.\n");
        return;
    }

    struct Node* toDelete = temp->next;
    temp->next = toDelete->next;
    free(toDelete);
    printf("Element deleted from position %d.\n", pos);
}

void search() {
    int element, pos = 1;
    printf("Enter the element to search: ");
    scanf("%d", &element);

    struct Node* temp = head;
    while (temp != NULL) {
        if (temp->data == element) {
            printf("Element %d found at position %d.\n", element,
pos);
            return;
        }
        temp = temp->next;
        pos++;
    }

    printf("Element %d not found in the list.\n", element);
}

int main() {
```

```c
    int choice;

    do {
        printf("Menu:\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 from a Position\n");
        printf("7. Search for an Element\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                insertBeginning();
                break;
            case 2:
                insertEnd();
                break;
            case 3:
                insertAtPosition();
                break;
            case 4:
                deleteBeginning();
                break;
            case 5:
                deleteEnd();
                break;
            case 6:
                deleteAtPosition();
                break;
            case 7:
                search();
                break;
            case 8:
                printf("Exiting program...\n");
                break;
            default:
                printf("Invalid choice.\n");
```

```
        }
    } while (choice != 8);

    return 0;
}
```

Output

```
PS E:\DSA>
gcc singlelist.c
PS E:\DSA> .\a.exe
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete from a Position
7. Search for an Element
8. Exit
Enter your choice: 1
Enter the element to insert: 25
Element inserted at 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 from a Position
7. Search for an Element
8. Exit
Enter your choice: 1
Enter the element to insert: 34
Element inserted at 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 from a Position
7. Search for an Element
8. Exit
```

```
Enter your choice: 1
Enter the element to insert: 59
Element inserted at 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 from a Position
7. Search for an Element
8. Exit
Enter your choice: 2
Enter the element to insert: 85
Element inserted at 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 from a Position
7. Search for an Element
8. Exit
Enter your choice: 4
Element deleted from 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 from a Position
7. Search for an Element
8. Exit
Enter your choice: 7
```

```
Enter the element to search: 85
Element 85 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 from a Position
7. Search for an Element
8. Exit
Enter your choice: 6
Enter the position to delete: 5
Position out of range.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete from a Position
7. Search for an Element
8. Exit
Enter your choice: 3
Enter the element to insert: 6
Enter the position: 15
Position out of range.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete from a Position
7. Search for an Element
8. Exit
```

```
Enter your choice: 8
Exiting program...
PS E:\DSA>
```

5. 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

Algorithm

1. Initialization:

- Define a structure `Node` with fields:
  - `data` to store the value.
  - `prev` to store the address of the previous node.
  - `next` to store the address of the next node.
- Initialize `head = NULL`.

2. Insertion Operations:

- **Beginning**:
  - Create a new node.
  - Set its `next` to the current head and `prev` to NULL.
  - Update the `prev` of the old head (if it exists) to point to the new node.
  - Update `head` to the new node.
- **End**:
  - Create a new node.
  - Traverse the list to the last node.
  - Update the `next` of the last node and set the new node's `prev` to the last node.
- **At a Given Position**:
  - Traverse to the specified position.
  - Insert the new node by updating the `next` and `prev` pointers of the neighboring nodes.

3. Deletion Operations:

- **Beginning**:
  - If `head` is NULL, print "List is empty."
  - Update `head` to `head->next`.
  - Set the `prev` of the new head to NULL.

- **End**:
  - Traverse to the last node.
  - Update the `next` of the second last node to NULL.
- **At a Given Position**:
  - Traverse to the specified position.
  - Update the `next` and `prev` pointers of the neighboring nodes to bypass the node to be deleted.

4. Search Operation:

- Traverse the list, comparing each node's `data` with the search key.
- If a match is found, print its position; otherwise, print "Element not found."

5. Menu and User Interaction:

- Display the menu with options for insertion, deletion, search, and exit.
- Repeat operations until the user chooses to exit.

Pseudocode

```
BEGIN

  Initialize head ← NULL



  WHILE true DO

    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 from a Position"
```

```
PRINT "7. Search for an Element"
PRINT "8. Exit"
PRINT "Enter your choice: "
READ choice

SWITCH choice DO
   CASE 1:
      CALL insertBeginning()
      BREAK

   CASE 2:
      CALL insertEnd()
      BREAK

   CASE 3:
      PRINT "Enter position: "
      READ pos
      CALL insertAtPosition(pos)
      BREAK

   CASE 4:
      CALL deleteBeginning()
```

```
    BREAK

CASE 5:
    CALL deleteEnd()
    BREAK

CASE 6:
    PRINT "Enter position: "
    READ pos
    CALL deleteAtPosition(pos)
    BREAK

CASE 7:
    PRINT "Enter element to search: "
    READ element
    CALL search(element)
    BREAK

CASE 8:
    PRINT "Exiting program..."
    EXIT
```

DEFAULT:

PRINT "Invalid choice. Please try again."

ENDSWITCH

ENDWHILE

END

C program

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

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

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

void insertBeginning() {
    int data;
    printf("Enter the element to insert: ");
    scanf("%d", &data);
    struct Node* newNode = createNode(data);
    if (head != NULL) {
        newNode->next = head;
        head->prev = newNode;
    }
```

```
    head = newNode;
    printf("Element inserted at the beginning.\n");
}

void insertEnd() {
    int data;
    printf("Enter the element to insert: ");
    scanf("%d", &data);
    struct Node* newNode = createNode(data);
    if (head == 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 data, pos;
    printf("Enter the element to insert: ");
    scanf("%d", &data);
    printf("Enter the position: ");
    scanf("%d", &pos);

    struct Node* newNode = createNode(data);
    if (pos == 1) {
        newNode->next = head;
        if (head != NULL) head->prev = newNode;
        head = newNode;
        printf("Element inserted at position %d.\n", pos);
        return;
    }

    struct Node* temp = head;
    for (int i = 1; temp != NULL && i < pos - 1; i++) {
        temp = temp->next;
    }
```

```
    if (temp == NULL) {
        printf("Position out of range.\n");
        free(newNode);
        return;
    }

    newNode->next = temp->next;
    if (temp->next != NULL) temp->next->prev = newNode;
    temp->next = newNode;
    newNode->prev = temp;
    printf("Element inserted at position %d.\n", pos);
}

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

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

    if (temp->prev != NULL) temp->prev->next = NULL;
    else head = NULL;

    free(temp);
```

```
    printf("Element deleted from the end.\n");
}

void deleteAtPosition() {
    int pos;
    printf("Enter the position to delete: ");
    scanf("%d", &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);
        printf("Element deleted from position %d.\n", pos);
        return;
    }

    for (int i = 1; temp != NULL && i < pos; i++) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Position out of range.\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 from position %d.\n", pos);
}

void search() {
    int element, pos = 1;
    printf("Enter the element to search: ");
    scanf("%d", &element);
```

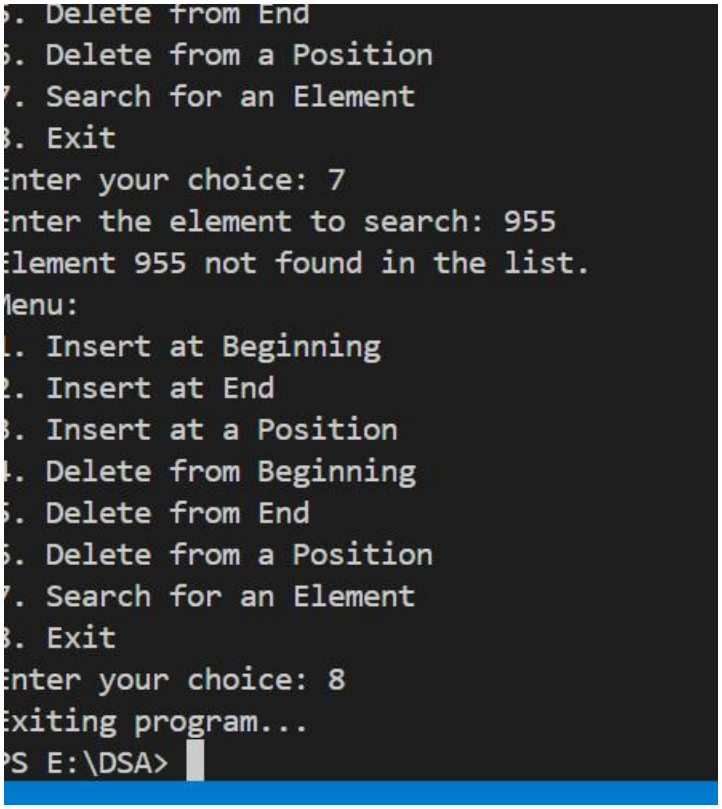
```
    struct Node* temp = head;
    while (temp != NULL) {
        if (temp->data == element) {
            printf("Element %d found at position %d.\n", element,
pos);
            return;
        }
        temp = temp->next;
        pos++;
    }

    printf("Element %d not found in the list.\n", element);
}


int main() {
    int choice;

    do {
        printf("Menu:\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 from a Position\n");
        printf("7. Search for an Element\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                insertBeginning();
                break;
            case 2:
                insertEnd();
                break;
            case 3:
                insertAtPosition();
                break;
```

```
            case 4:
                deleteBeginning();
                break;
            case 5:
                deleteEnd();
                break;
            case 6:
                deleteAtPosition();
                break;
            case 7:
                search();
                break;
            case 8:
                printf("Exiting program...\n");
                break;
            default:
                printf("Invalid choice.\n");
        }
    } while (choice != 8);

    return 0;
}
```

Output

```
1. Insert at Beginning
PS E:\DSA> .\a.exe
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete from a Position
7. Search for an Element
8. Exit
Enter your choice: 1
Enter the element to insert: 26
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 from a Position
7. Search for an Element
8. Exit
Enter your choice: 3
Enter the element to insert: 64
Enter the position: 3
Position out of range.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
6. Delete from a Position
7. Search for an Element
8. Exit
```

```
Enter your choice: 2
Enter the element to insert: 9846
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 from a Position
7. Search for an Element
8. Exit
Enter your choice: 3
Enter the element to insert: 269
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 from a Position
7. Search for an Element
8. Exit
Enter your choice: 6
Enter the position to delete: 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 from a Position
7. Search for an Element
```


. Delete from End
5. Delete from a Position
7. Search for an Element
8. Exit
Enter your choice: 7
Enter the element to search: 955
Element 955 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 from a Position
7. Search for an Element
8. Exit
Enter your choice: 8
Exiting program...
PS E:\DSA>