NAME: ANVI PHARANDE

REG NO.:24BBS0092

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\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() {
    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 {
```

```c
else {
printf("Stack elements are: ");
for (int i = 0; i <= top; i++) {
printf("%d", stack[i]);
}
printf("\n");
}}

int main() {
int choice;
do {

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

```
57  case 3:
58  display();
59  break;
60  case 4:
61  printf("Exiting program...\n");
62  break;
63  default:
64  printf("Invalid choice.\n");
65  }
66  } while (choice != 4);
67   return 0;
68  }
```

OUTPUT:

Output

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


=== Code Execution Successful ===
```

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:

main.c

Share Run

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

int queue[MAX];
int front = -1,rear=-1;
void enqueue() {
int element;
if (rear == MAX-1) {
printf("queue overflow\n");
return;
}
else {
printf("Enter the element: ");
scanf("%d", &element);
if(front==-1){
    front=0;
}
queue[++rear]=element;
printf("Element %d enqueued.\n", element);
}}
void dequeue() {
    if (front==-1||front>rear){
    printf("Queue Underflow\n");
        return;
    }
    printf("dequeued element:%d\n",queue[front++]
        );
    if(front>rear){
        front=rear=-1;
    }}

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

main.c

Share Run

```c
void display() {
if (front == -1) {
printf("queue is empty.\n");
return;
}
printf("Stack elements are: ");
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;
```

```
62  display();
63  break;
64  case 4:
65  printf("Exiting program...\n");
66  break;
67  default:
68  printf("Invalid choice.\n");
69  }
70  } while (choice != 4);
71   return 0;
72  }
```

OUTPUT:

Output

Clear

```
Menu:
1.enqueue
2.dequeue
3.Display
4.Exit
Enter your choice: 1
Enter the element: 5
Element 5 enqueued.
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: 3
queue is empty.
Menu:
1.enqueue
2.dequeue
3.Display
4.Exit
Enter your choice: 4
Exiting program...


=== Code Execution Successful ===
```

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

   b. Dequeue()

   c. display()

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:

main.c

Share Run

```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\n");
return;
}
else {
printf("Enter the element: ");
scanf("%d", &element);
if(front==-1){
    front=0;
}
queue[++rear]=element;
printf("Element %d enqueued.\n", element);
}}
void dequeue() {
    if (front==-1||front>rear){
    printf("Queue Underflow\n");
    }
    else{
    printf("dequeued element:%d\n",queue[front]);
    if(front==rear){
        front=-1;
        rear=-1;}
        else{
            front=(front+1)%MAX;
        }
    }}
```

main.c

Share Run

```
33      }}
34
35  void display() {
36      int i=front;
37  if (front == -1) {
38  printf("queue is empty.\n");
39  } else{
40  printf("queue elements are: ");
41  while(1){
42      printf("%d\t",queue[i]);
43      if(i==rear){
44          break;
45      }
46      i=(i=1)%MAX;
47  }
48  printf("\n");
49  }}
50
51  int main() {
52  int choice;
53  do {
54  printf("Menu:\n");
55  printf("1.enqueue\n");
56  printf("2.dequeue\n");
57  printf("3.Display\n");
58  printf("4.Exit\n");
59  printf("Enter your choice: ");
60  scanf("%d", &choice);
61
62  switch (choice) {
63  case 1:
64  enqueue();
65  break;
```

```
61
62  switch (choice) {
63  case 1:
64  enqueue();
65  break;
66  case 2:
67  dequeue();
68  break;
69  case 3:
70  display();
71  break;
72  case 4:
73  printf("Exiting program...\n");
74  break;
75  default:
76  printf("Invalid choice.\n");
77  }
78  } while (choice != 4);
79   return 0;
80  }
```

OUTPUT:

Output Clear

```
Menu:
1.enqueue
2.dequeue
3.Display
4.Exit
Enter your choice: 1
Enter the element: 3
Element 3 enqueued.
Menu:
1.enqueue
2.dequeue
3.Display
4.Exit
Enter your choice: 1
Enter the element: 6
Element 6 enqueued.
Menu:
1.enqueue
2.dequeue
3.Display
4.Exit
Enter your choice: 3
queue elements are: 3   6
Menu:
1.enqueue
2.dequeue
3.Display
4.Exit
Enter your choice: 2
dequeued element:3
Menu:
1.enqueue
2.dequeue
```

1.

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:

main.c

Share Run

```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){
```

main.c

Share Run

```
33      }}
34
35  void display() {
36      int i=front;
37  if (front == -1) {
38  printf("queue is empty.\n");
39  } else{
40  printf("queue elements are: ");
41  while(1){
42      printf("%d\t",queue[i]);
43      if(i==rear){
44          break;
45      }
46      i=(i=1)%MAX;
47  }
48  printf("\n");
49  }}
50
51  int main() {
52  int choice;
53  do {
54  printf("Menu:\n");
55  printf("1.enqueue\n");
56  printf("2.dequeue\n");
57  printf("3.Display\n");
58  printf("4.Exit\n");
59  printf("Enter your choice: ");
60  scanf("%d", &choice);
61
62  switch (choice) {
63  case 1:
64  enqueue();
65  break;
```

main.c

Share Run

```c
    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");
```

main.c

Share Run

```
92          return;
93      }
94
95      if (head->next == NULL) {
96          free(head);
97          head = NULL;
98      } else {
99          struct node* temp = head;
100         while (temp->next->next != NULL) {
101             temp = temp->next;
102         }
103         free(temp->next);
104         temp->next = NULL;
105     }
106     printf("Element deleted from end.\n");
107 }
108
109 void deleteAtPosition() {
110     int pos;
111     printf("Enter the position to delete: ");
112     scanf("%d", &pos);
113
114     if (head == NULL) {
115         printf("List is empty.\n");
116         return;
117     }
118
119     if (pos == 1) {
120 struct node* temp = head;
121         head = head->next;
122         free(temp);
123         printf("Element deleted from position %d
                .\n", pos);
```

main.c

Share Run

```
                .\n", pos);
124         return;
125     }
126
127     struct node* temp = head;
128     for (int i = 1; temp != NULL && i < pos - 1;
            i++) {
129         temp = temp->next;
130     }
131
132     if (temp == NULL || temp->next == NULL) {
133         printf("Position out of range.\n");
134         return;
135     }
136
137     struct node* toDelete = temp->next;
138     temp->next = toDelete->next;
139     free(toDelete);
140     printf("Element deleted from position %d.\n",
            pos);
141 }
142
143 void search() {
144     int element, pos = 1;
145     printf("Enter the element to search: ");
146     scanf("%d", &element);
147
148     struct node* temp = head;
149     while (temp != NULL) {
150         if (temp->data == element) {
151             printf("Element %d found at position
                    %d.\n", element, pos);
152             return;
```

main.c

Share Run

```
153         }
154         temp = temp->next;
155         pos++;
156     }
157
158     printf("Element %d not found in the list.\n",
            element);
159 }
160
161 int main() {
162 int choice;
163
164     do {
165         printf("Menu:\n");
166         printf("1. Insert at Beginning\n");
167         printf("2. Insert at End\n");
168         printf("3. Insert at a Position\n");
169         printf("4. Delete from Beginning\n");
170         printf("5. Delete from End\n");
171         printf("6. Delete from a Position\n");
172         printf("7. Search for an Element\n");
173         printf("8. Exit\n");
174         printf("Enter your choice: ");
175         scanf("%d", &choice);
176
177         switch (choice) {
178             case 1:
179                 insertbeginning();
180                 break;
181             case 2:
182                 insertend();
183                 break;
```

main.c

Share Run

```
178             case 1:
179                 insertbeginning();
180                 break;
181             case 2:
182                 insertend();
183                 break;
184             case 3:
185                 insertAtPosition();
186                 break;
187             case 4:
188                 deleteBeginning();
189                 break;
190             case 5:
191                 deleteEnd();
192                 break;
193             case 6:
194                 deleteAtPosition();
195                 break;
196             case 7:
197                 search();
198                 break;
199             case 8:
200                 printf("Exiting program...\n");
201                 break;
202             default:
203                 printf("Invalid choice.\n");
204         }
205     } while (choice != 8);
206
207     return 0;
208 }
209
```

OUTPUT:

```
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:3
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:5
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: 3
Enter the element to insert: 6
Enter the position: 1
Element inserted at position 1.
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: 5
Element deleted from 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
```

```
6. Delete from a Position
7. Search for an Element
8. Exit
Enter your choice: 6
Enter the position to delete: 2
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: 7
Enter the element to search: 0
Element 0 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...


=== Code Execution Successful ===
```

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:

main.c

Share Run

```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() {
```

main.c

Share Run

```c
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;
```

main.c

Share Run

```
        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;
```

main.c

Share Run

```c
    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;
```

main.c

Share Run

```c
    }

    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() {
```

main.c

Share Run

```c
    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: ");
```

main.c Share Run Output Clear

```
182
183     switch (choice) {
184         case 1:
185             insertBeginning();
186             break;
187         case 2:
188             insertEnd();
189             break;
190         case 3:
191             insertAtPosition();
192             break;
193         case 4:
194             deleteBeginning();
195             break;
196         case 5:
197             deleteEnd();
198             break;
199         case 6:
200             deleteAtPosition();
201             break;
202         case 7:
203             search();
204             break;
205         case 8:
206             printf("Exiting program...\n");
207             break;
208         default:
209             printf("Invalid choice.\n");
210     }
211 } while (choice != 8);
212
213 return 0;
```

```
6. Delete from a Position
7. Search for an Element
8. Exit
Enter your choice: 6
Enter the position to delete: 1
Element deleted from position 1.
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: 5
Element 5 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...


=== Code Execution Successful ===
```

OUTPUT:

```
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: 4
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: 1
Enter the element to insert: 5
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: 2
Enter the element to insert: 3
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: 6
Enter the position: 1
Element inserted at position 1.
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 the beginning.
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at a Position
4. Delete from Beginning
5. Delete from End
```

Output

Clear

```
5. Delete from End
6. Delete from a Position
7. Search for an Element
8. 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 from a Position
7. Search for an Element
8. Exit
Enter your choice: 6
Enter the position to delete: 1
Element deleted from position 1.
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: 5
Element 5 not found in the list.
Menu:
1. Insert at Beginning
2. Insert at End
```