Course Code: CBS1003

Coure Name: Data Structures and Algorithms

Assessment-1

Name : SASWAT T R

Reg No : 24BBS0139

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

a. PUSH()

Pseudocode :

```
Initialize stack[N] & top=-1
Function PUSH(X)
    if(top==N-1) then
        print("Stack Overflow")
    else
        ++top
        Stack[top] <- x
End Function
```

b. POP()

Pseudocode :

```
Initialize stack[N] & top=-1
Function POP()
    If(top==-1) then
        print("Stack Underflow")
    else
        initialize temp
        temp <- stack[top]
        top—
End Function
```

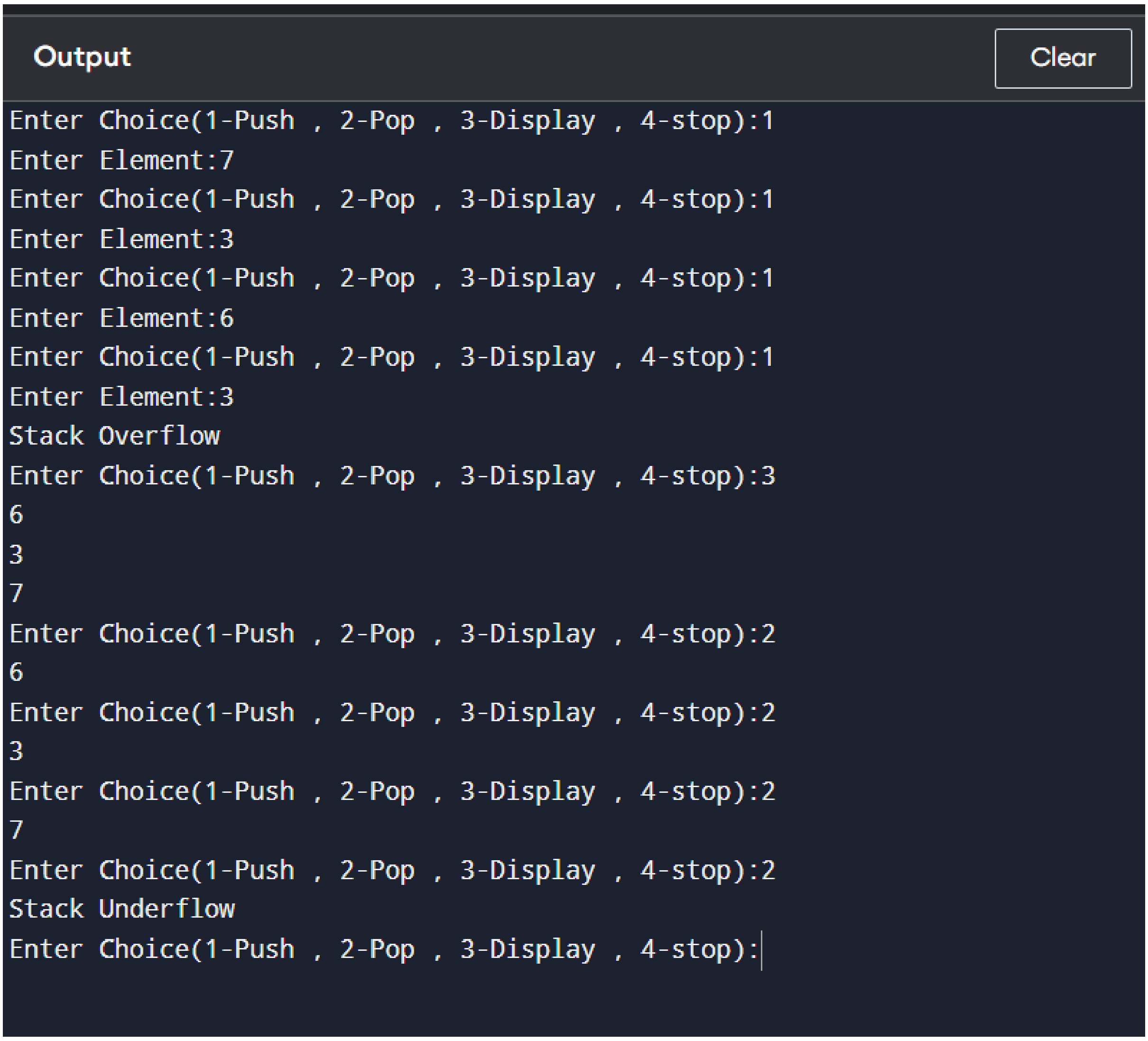
c.Display()

```
Function Display()
        if (top==-1) then
                print(“Stack Empty”)
       else
                for (i <- top to 0 ; i-- )
                          print(stack[i])
End Function
```

Code :

```
#include <stdio.h>
#define n 3
int stack[n];
int top = -1;

int Isfull(){
    if (top==n-1){
        return 1;
    }
    else {
        return 0;
    }
}

void PUSH(int x)
{
    if (Isfull()){
        printf("Stack Overflow \n");
    }
    else {
        ++top;
        stack[top]=x;
    }
}
int Isempty(){
    if(top==-1){
        return 1;
    }
    else {
        return 0;
    }
}
void POP(){
    if (Isempty()){
        printf("Stack Underflow \n");
    }
    else {
        int temp;
        temp=stack[top];
        top--;
        printf("%d \n",temp);
    }
}
void DISPLAY(){
    if (Isempty()){
        printf("Stack Empty \n");
    }
    else {
        for(int i=top;i>=0;i--){
            printf("%d \n",stack[i]);
        }
    }
}


int main()
{
    int i=1;
    while (i==1)
    {
        int y;
        printf("Enter Choice(1-Push , 2-Pop , 3-Display , 4-stop):" );
        scanf("%d",&y);
        switch (y)
        {
            case 1:
                int a;
                printf("Enter Element:");
                scanf("%d",&a);
                PUSH(a);
                break;
            case 2:
                POP();
                break;
            case 3:
                DISPLAY();
                break;
            case 4:
                i=0;
                break;
            default :
                printf("Enter Valid Choice");
                break;
        }
    }
    return 0;
}
```

Output:



```
Enter Choice(1-Push , 2-Pop , 3-Display , 4-stop):1
Enter Element:7
Enter Choice(1-Push , 2-Pop , 3-Display , 4-stop):1
Enter Element:3
Enter Choice(1-Push , 2-Pop , 3-Display , 4-stop):1
Enter Element:6
Enter Choice(1-Push , 2-Pop , 3-Display , 4-stop):1
Enter Element:3
Stack Overflow
Enter Choice(1-Push , 2-Pop , 3-Display , 4-stop):3
6
3
7
Enter Choice(1-Push , 2-Pop , 3-Display , 4-stop):2
6
Enter Choice(1-Push , 2-Pop , 3-Display , 4-stop):2
3
Enter Choice(1-Push , 2-Pop , 3-Display , 4-stop):2
7
Enter Choice(1-Push , 2-Pop , 3-Display , 4-stop):2
Stack Underflow
Enter Choice(1-Push , 2-Pop , 3-Display , 4-stop):
```

2. Write a menu driven program to implement the following operations on Queue:

a. Enqueue()

Pseudocode:

```
Initialize Queue[n] & front=-1 & rear=-1
Function Enqueue(x)
    if(rear==n-1) then
        print("Enqueue is not possible")
    else
        if (front==-1) then
            front +=1
        rear+=1
        Queue[rear] <- x
End Function
```

b. Dequeue()

```
Initialize Queue[n] & front=-1 & rear=-1
Function Dequeue()
    If (front==-1 or front>rear) then
        print("Queue empty")
    else
        temp <- queue[front]
        front+=1
End Function
```

c.Display()

```
Initialize Queue[n] & front=-1 & rear=-1
Function Display()
```

```
        If (front==-1 or front>rear) then
        print("Queue empty")
  else
       for( i <- front to rear)
            print(Queue[i])
End Function
```

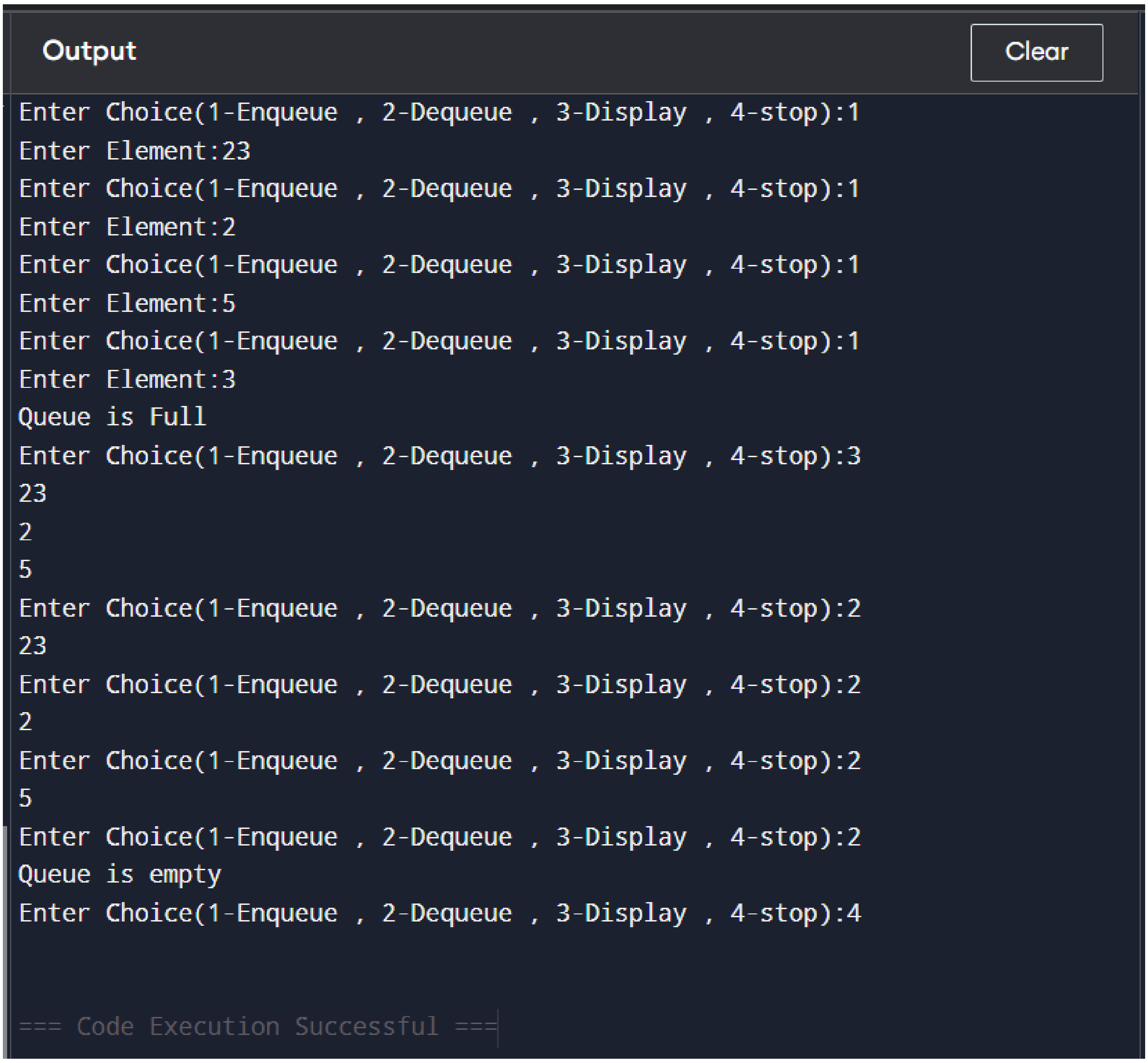
Code :

```
#include <stdio.h>
#define n 3
int queue[n];
int front=-1;
int rear=-1;
int IsQueuefull(){
  if (rear==n-1){
    return 1;
  }
  else {
    return 0;
  }
}
void Enqueue(int x){
  if (IsQueuefull()){
    printf("Queue is Full \n");
  }
  else{
    if (front==-1){
      front+=1;
```

```
        }
        rear+=1;
        queue[rear]=x;
    }
}
int IsQueueempty(){
    if (front==-1 || front>rear){
        return 1;
    }
    else {
        return 0;
    }
}
void Dequeue(){
    int temp;
    if (IsQueueempty()){
        printf("Queue is empty \n");
    }
    else {
        temp=queue[front];
        front+=1;
        printf("%d \n",temp);
    }
}
void Display(){
    if (IsQueueempty()){
        printf("Queue is empty \n");
    }
    else {
        for (int i=front ; i<=rear; i++){
            printf("%d \n",queue[i]);
        }
    }
}
int main() {
    int i=1;
    while (i==1)
    {
        int y;
        printf("Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):" );
        scanf("%d",&y);
        switch (y)
        {
            case 1:
                int a;
                printf("Enter Element:");
                scanf("%d",&a);
                Enqueue(a);
                break;
            case 2:
                Dequeue();
                break;
            case 3:
                Display();
                break;
            case 4:
```

```
            i=0;
            break;
          default :
            printf("Enter Valid Choice \n");
            break;
        }
    }

    return 0;
}
```

Output :



```
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):1
Enter Element:23
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):1
Enter Element:2
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):1
Enter Element:5
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):1
Enter Element:3
Queue is Full
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):3
23
2
5
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):2
23
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):2
2
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):2
5
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):2
Queue is empty
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):4


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

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

Pseudocode:

A. Enqueue()

```
Initialize queue[n] & front=-1 & rear=-1
Function Enqueue(x)
    if ((rear+1)%n == front) then
        print("Queue is full")
    else
        if (front==-1) then
            front <- 0
        rear <- (rear+1)%n
        queue[rear] <- x
End Function
```

B. Dequeue()

```
Initialize queue[n] & front=-1 & rear=-1
Function Dequeue()
    If (front==-1 and rear==-1) then
        print("Queue is empty")
    else if (front==rear) then
        temp <- queue[front]
        print(temp)
        front <-  0
        rear  <- 0
    else
        temp <- queue[front]
        print(temp)
```

```
            front <- (front+1)%n
    End Function
```

C. Display()

```
    Initialize queue[n] & front=-1 & rear=-1
    Function Display()
        if(front==-1 and rear==-1) then
            print("Queue is Empty")
        else
            i <- front
            while( i != rear) then
                print(queue[i])
                i <- (i+1)%n
            print(queue[rear])
    End Function
```

Code:

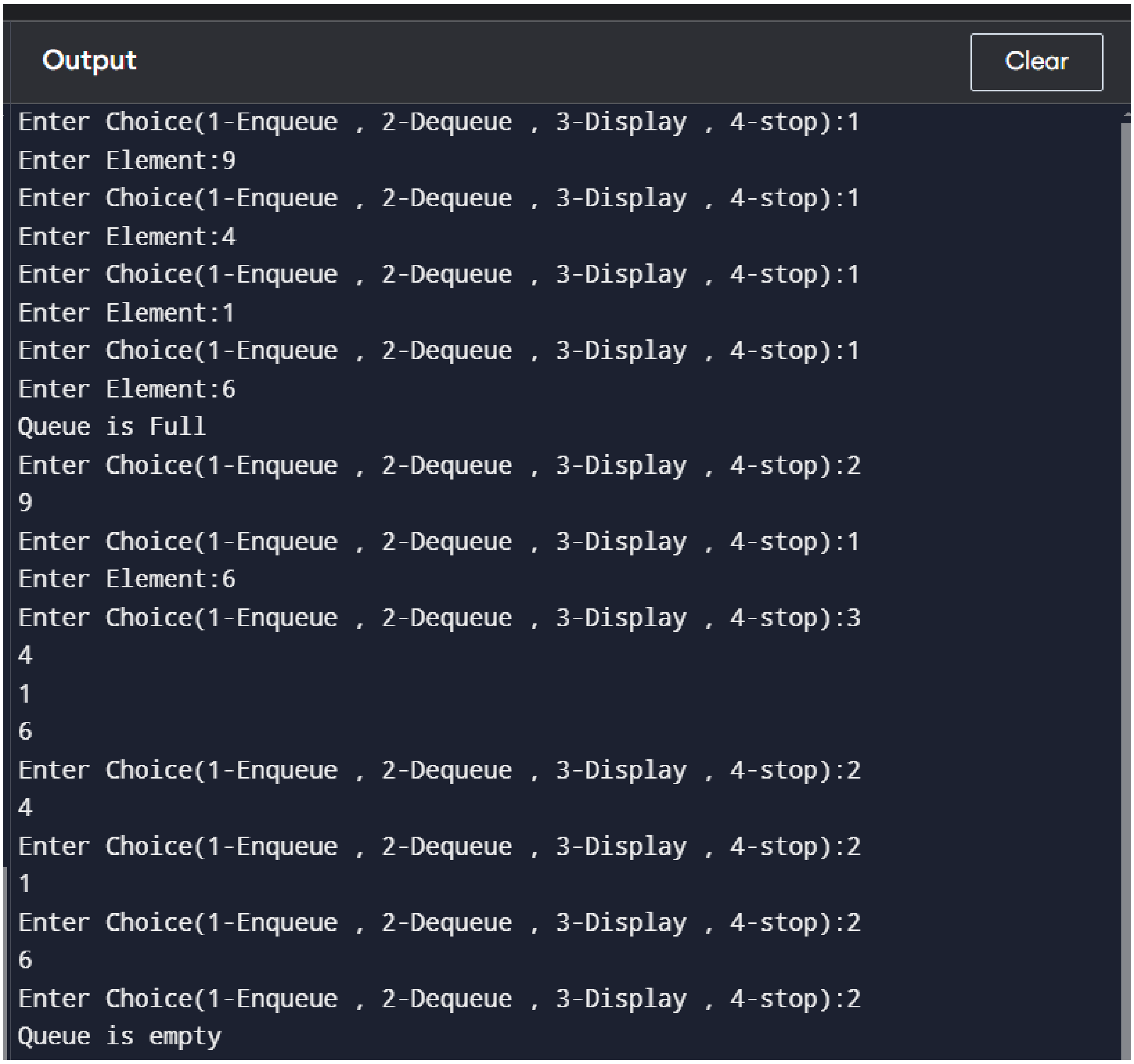
```
#include <stdio.h>
#define n 3
int queue[n];
int front=-1;
int rear=-1;
int Isqueuefull(){
   if((rear+1)%n == front){
      return 1;
   }
```

```
    else {
        return 0;
    }
}
void Enqueue(int x){
    if(Isqueuefull()){
        printf("Queue is Full \n");
    }
    else{
        if(front==-1){
            front=0;
        }
        rear=(rear+1)%n;
        queue[rear]=x;

    }
}
int Isqueueempty(){
    if(front==-1 && rear==-1){
        return 1;
    }
    else {
        return 0;
    }
}
void Dequeue(){
    int temp;
    if(Isqueueempty()){
        printf("Queue is empty \n");
    }
    else if (front==rear) {
        temp=queue[front];
        front=rear=-1;
        printf("%d \n",temp);
    }
    else {
        temp=queue[front];
        front=(front+1)%n;
        printf("%d \n",temp);
    }
}
void Display(){
    int i=front;
    if(Isqueueempty()){
        printf("Queue is empty \n");
    }
    else{
        while(i!=rear){
            printf("%d \n",queue[i]);
            i=(i+1)%n;
        }
        printf("%d \n",queue[rear]);
    }
}
int main() {
    int i=1;
```

```
while (i==1)
{
    int y;
    printf("Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):" );
    scanf("%d",&y);
    switch (y)
    {
        case 1:
            int a;
            printf("Enter Element:");
            scanf("%d",&a);
            Enqueue(a);
            break;
        case 2:
            Dequeue();
            break;
        case 3:
            Display();
            break;
        case 4:
            i=0;
            break;
        default :
            printf("Enter Valid Choice \n");
            break;
    }
}
```

```
    return 0;
}
```

Output :



```
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):1
Enter Element:9
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):1
Enter Element:4
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):1
Enter Element:1
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):1
Enter Element:6
Queue is Full
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):2
9
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):1
Enter Element:6
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):3
4
1
6
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):2
4
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):2
1
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):2
6
Enter Choice(1-Enqueue , 2-Dequeue , 3-Display , 4-stop):2
Queue is empty
```

4. Write a menu driven program to implement the following operations on singly linked list:

```
  Create a new node, t
    t.data <- data
    t.next <- start
    start <- t
    Return start
```

a. Insertion()

i. Beginning :

```
Function insert_b():
    Input num
    Create a new node t
    t.data <- num
    t.next <- NULL

    If start is NULL:
        start <- t
    Else:
        q <- start
        While q.next is not NULL:
            q <- q.next
        End While
        q.next <- t
End Function
```

ii. End:

```
Function insert_e():
   Input num
   Create a new node t
   t.data <- num
   t.next <- NULL

   If start is NULL :
      start <- t
   Else:
      q <- start
      While q.next is not NULL:
         q <- q.next
      End While
      q.next <- t
End Function
```

iii. At a given position

```
Function insert_p(p, n):
   Input: position p, data n
   If start is NULL:
      Print ("List is empty")
      Return 0
   Create a new node t
   t.data <- n
   q <- start
   For i <- 0 to p - 1:
      If q.next is NULL:
         Print ("There are fewer elements")
         Return 0
      End If
      q <- q.next
   End For
   t.next <- q.next
   q.next <- t
   Return 0
End Function
```

b)Deletion()

i. Beginning

```
Function delete_b():
   If start is NULL:
      Print "The list is empty"
   Else:
      q <- start
      start <- start.next
      Print ("Deleted element is", q.data)
      Free q
End Function
```

ii. End:

```
Function delete_e():
   If start is NULL:
      Print ("The list is empty")
      Return 0
   q = start
   While q.next.next is not NULL:
      q <- q.next
   End While
   t <- q.next
   q.next <- NULL
   Print ("Deleted element is", t.data)
   Free t
End Function
```

iii. At a given position:

```
Function delete_p(pos):
   If start is NULL:
      Print ("List is empty")
      Return 0
   q <- start
   For i <- 1 to pos - 1:
      If q.next is NULL:
         Print "There are fewer elements"
         Return 0
      End If
      q <- q.next
   End For
   t <- q.next
```

```
    q.next <- t.next
    Print ("Deleted element is", t.data)
    Free t
    Return 0
End Function
```

iii. At a given position

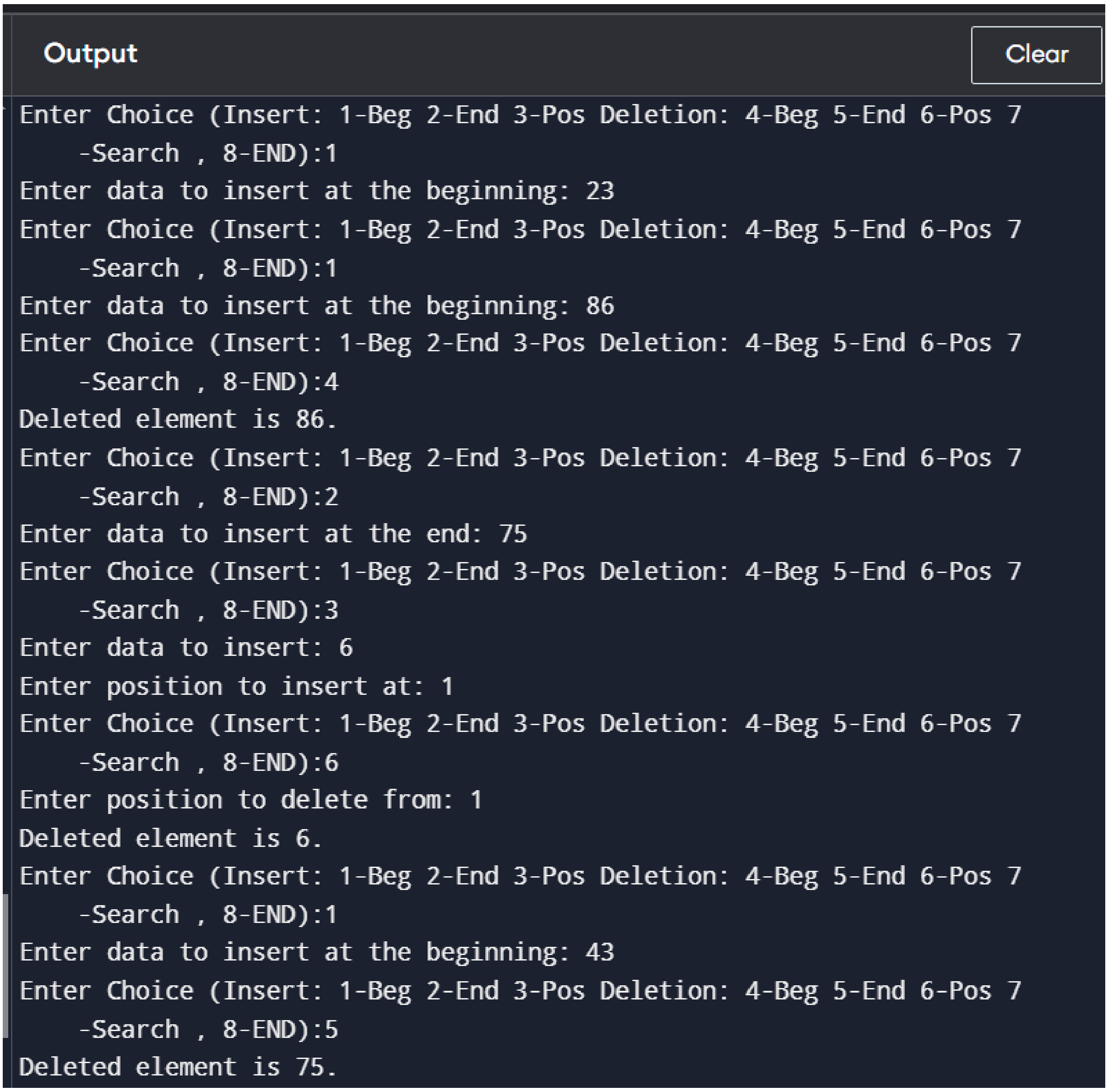
```
Function search(k):
    Input: Key k
    flag <- 0
    temp <- start
    While temp is not NULL:
        If k == temp.data:
            flag = 1
            Break
        End If
        temp <- temp.next
    End While
    If flag == 1:
        Print( "Key Found")
    Else:
        Print ("Key Not Found")
    End If
End Function
```

Code:

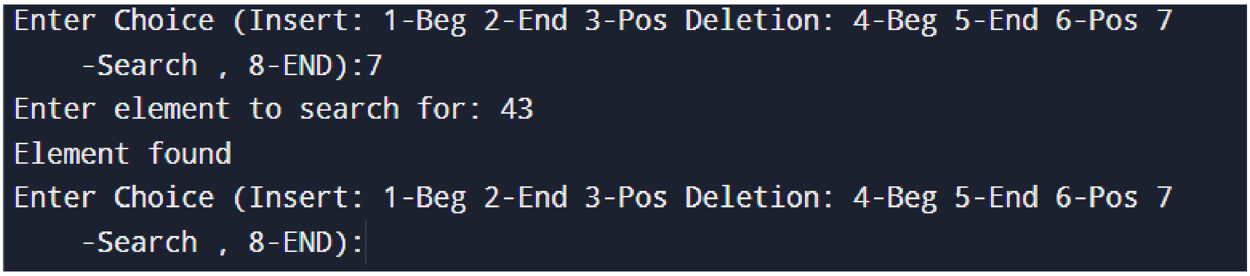
```
#include <stdio.h>
#include <stdlib.h>

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

void insert_b(int data) {
   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = head;
   head = newNode;
}

void insert_e(int data) {
   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
   struct Node *temp = head;
   newNode->data = data;
   newNode->next = NULL;

   if (head == NULL) {
      head = newNode;
      return;
   }

   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->next = newNode;
}

void insert_p(int data, int position) {
   if (position <= 0) {
      printf("Invalid position.\n");
      return;
   }

   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
   struct Node *temp = head;
   int i;

   newNode->data = data;

   if (position == 1) {
      newNode->next = head;
      head = newNode;
      return;
   }

   for (i = 1; i < position - 1; i++) {
      if (temp == NULL) {
          printf("Position exceeds the size of the list \n");
          free(newNode);
          return;
      }
      temp = temp->next;
   }

   newNode->next = temp->next;
   temp->next = newNode;
}

void delete_b() {
   if (head == NULL) {
      printf("The list is empty.\n");
      return;
   }

   struct Node *temp = head;
   head = head->next;
   printf("Deleted element is %d.\n", temp->data);
   free(temp);
}

void delete_e() {
   if (head == NULL) {
      printf("The list is empty.\n");
      return;
   }

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

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

   printf("Deleted element is %d.\n", temp->data);
   free(temp);
}

void delete_p(int position) {
   if (head == NULL) {
      printf("The list is empty.\n");
      return;
   }

   if (position <= 0) {
      printf("Invalid position.\n");
      return;
   }

   struct Node *temp = head;
   struct Node *prev = NULL;
   int i;

   if (position == 1) {
      head = head->next;
      printf("Deleted element is %d.\n", temp->data);
      free(temp);
      return;
   }

   for (i = 1; i < position; i++) {
      if (temp == NULL) {
          printf("Position exceeds the size of the list.\n");
          return;
      }
      prev = temp;
      temp = temp->next;
   }

   prev->next = temp->next;
   printf("Deleted element is %d.\n", temp->data);
   free(temp);
}

void search(int key) {
   struct Node *temp = head;
   int found = 0;
   while (temp != NULL) {
      if (temp->data == key) {
          found = 1;
          break;
      }
      temp = temp->next;
   }

   if (found) {
      printf("Element found \n");
   } else {
      printf("Elementnot found \n");
   }
}

int main() {
   int choice, d, p;
   int f=1;
   while (f==1) {
      printf("Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7-Search , 8-END):");
      scanf("%d", &choice);

      switch (choice) {
          case 1:
              printf("Enter data to insert at the beginning: ");
              scanf("%d", &d);
              insert_b(d);
              break;
          case 2:
              printf("Enter data to insert at the end: ");
              scanf("%d", &d);
              insert_e(d);
              break;
          case 3:
              printf("Enter data to insert: ");
              scanf("%d", &d);
              printf("Enter position to insert at: ");
              scanf("%d", &p);
              insert_p(d,p);
              break;
          case 4:
              delete_b();
              break;
          case 5:
              delete_e();
              break;
          case 6:
              printf("Enter position to delete from: ");
              scanf("%d", &p);
              delete_p(p);
              break;
          case 7:
              printf("Enter element to search for: ");
              scanf("%d", &d);
              search(d);
```

```
                break;
            case 8:
                f=0;
                break;
            default:
                printf("Invalid choice. Please try again.\n");
                break;
        }
    }

    return 0;
}
```

Output:



```
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):1
Enter data to insert at the beginning: 23
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):1
Enter data to insert at the beginning: 86
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):4
Deleted element is 86.
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):2
Enter data to insert at the end: 75
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):3
Enter data to insert: 6
Enter position to insert at: 1
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):6
Enter position to delete from: 1
Deleted element is 6.
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):1
Enter data to insert at the beginning: 43
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):5
Deleted element is 75.
```



```
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):7
Enter element to search for: 43
Element found
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):
```

5. Write a menu driven program to implement the following operations on Doubly linked list:

```
Define a structure Node:
    Integer data
    Pointer to Node prev
    Pointer to Node next
```

a. Insertion()

i. Beginning:

```
Function insertBeginning(data):
    Create a new node
    Set newNode->data <- data
    Set newNode->prev <- NULL
    Set newNode->next <- head
    If head is not NULL:
        Set head->prev <- newNode
    Set head <- newNode
End Function
```

ii. End:

```
Function insertEnd(data):
    Create a new node
    Set newNode->data <- data
    Set newNode->next <- NULL
    If head is NULL:
        Set newNode->prev <- NULL
        Set head <- newNode
        Return
    Set temp <- head
    While temp->next is not NULL:
        Set temp <- temp->next
    Set temp->next <- newNode
    Set newNode->prev <- temp
End Function
```

iii. At a given position:

```
Function insertAtPosition(data, position):
    If position <= 0:
        Print ("Invalid position")
        Return
    Create a new node
    Set newNode->data <- data
    If position is 1:
        Set newNode->prev <- NULL
        Set newNode->next <- head
        If head is not NULL:
            Set head->prev <- newNode
        Set head <- newNode
        Return
    Set temp <- head
    For i <- 1 to position - 1:
        If temp is NULL:
            Print ("Position exceeds the size of the list")
            Free newNode
            Return
        Set temp <- temp->next
    If temp is NULL:
        Print ("Position exceeds the size of the list")
        Free newNode
        Return
    Set newNode->next <- temp->next
    Set newNode->prev <- temp
    If temp->next is not NULL:
        Set temp->next->prev <- newNode
    Set temp->next <- newNode
End Function
```

b. Deletion()

i. Beginning

```
Function deleteBeginning():
    If head is NULL:
        Print("The list is empty")
        Return
    Set temp <- head
    Set head <- head->next
    If head is not NULL:
        Set head->prev <- NULL
    Print ("Deleted element is temp->data")
    Free temp
End Function
```

ii. End

```
Function deleteEnd():
    If head is NULL:
        Print ("The list is empty")
        Return
    Set temp <- head
    If head->next is NULL:
        Set head <- NULL
        Print ("Deleted element is temp->data")
        Free temp
        Return
    While temp->next is not NULL:
        Set temp <- temp->next
    Set temp->prev->next <- NULL
    Print ("Deleted element is temp->data")
    Free temp
End Function
```

iii. At a given position

```
Function deleteAtPosition(position):
    If head is NULL:
        Print ("The list is empty")
        Return
    If position <= 0:
        Print ("Invalid position")
        Return
    Set temp <- head
    If position is 1:
        Set head <- head->next
        If head is not NULL:
            Set head->prev <- NULL
        Print ("Deleted element is temp->data")
        Free temp
        Return
    For i <- 1 to position:
        If temp is NULL:
            Print ("Position exceeds the size of the list")
            Return
        Set temp <- temp->next
    If temp is NULL:
        Print( "Position exceeds the size of the list")
        Return
    If temp->next is not NULL:
        Set temp->next->prev <- temp->prev
    If temp->prev is not NULL:
        Set temp->prev->next <- temp->next
    Print ("Deleted element is temp->data")
    Free temp
End Function
```

c. Search(): search for the given element on the list:

```
Function search(key):
   Set temp <- head
   Set flag <- 0
   While temp is not NULL:
      If temp->data is equal to key:
         Set flag <- 1
      Set temp <- temp->next
   If flag is 1:
      Print ("Element found")
   Else:
      Print ("Element not found")
End Function
```

Code :

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

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

struct Node *head = NULL;
```

```
void insert_b(int data) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = head;
    if (head != NULL) {
        head->prev = newNode;
    }
    head = newNode;
}

void insert_e(int data) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    struct Node *temp = head;
    newNode->data = data;
    newNode->next = NULL;
    if (head == NULL) {
        newNode->prev = NULL;
        head = newNode;
        printf("Node inserted at the end.\n");
        return;
    }
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
```

```
void insert_p(int data, int position) {
    if (position <= 0) {
        printf("Invalid position.\n");
        return;
    }
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    struct Node *temp = head;
    int i;
    newNode->data = data;
    if (position == 1) {
        newNode->prev = NULL;
        newNode->next = head;
        if (head != NULL) {
            head->prev = newNode;
        }
        head = newNode;
        return;
    }
    for (i = 1; i < position - 1; i++) {
        if (temp == NULL) {
            printf("Position exceeds the size of the list.\n");
            free(newNode);
            return;
        }
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Position exceeds the size of the list.\n");
        free(newNode);
        return;
    }
    newNode->next = temp->next;
    newNode->prev = temp;
    if (temp->next != NULL) {
        temp->next->prev = newNode;
    }
    temp->next = newNode;
}

void delete_b() {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }
    struct Node *temp = head;
    head = head->next;
    if (head != NULL) {
        head->prev = NULL;
    }
    printf("Deleted element is %d.\n", temp->data);
    free(temp);
}

void delete_e() {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }
    struct Node *temp = head;
    if (head->next == NULL) {
        head = NULL;
        printf("Deleted element is %d.\n", temp->data);
        free(temp);
        return;
    }
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->prev->next = NULL;
    printf("Deleted element is %d.\n", temp->data);
    free(temp);
}

void delete_p(int position) {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }
    if (position <= 0) {
        printf("Invalid position.\n");
        return;
    }
    struct Node *temp = head;
```

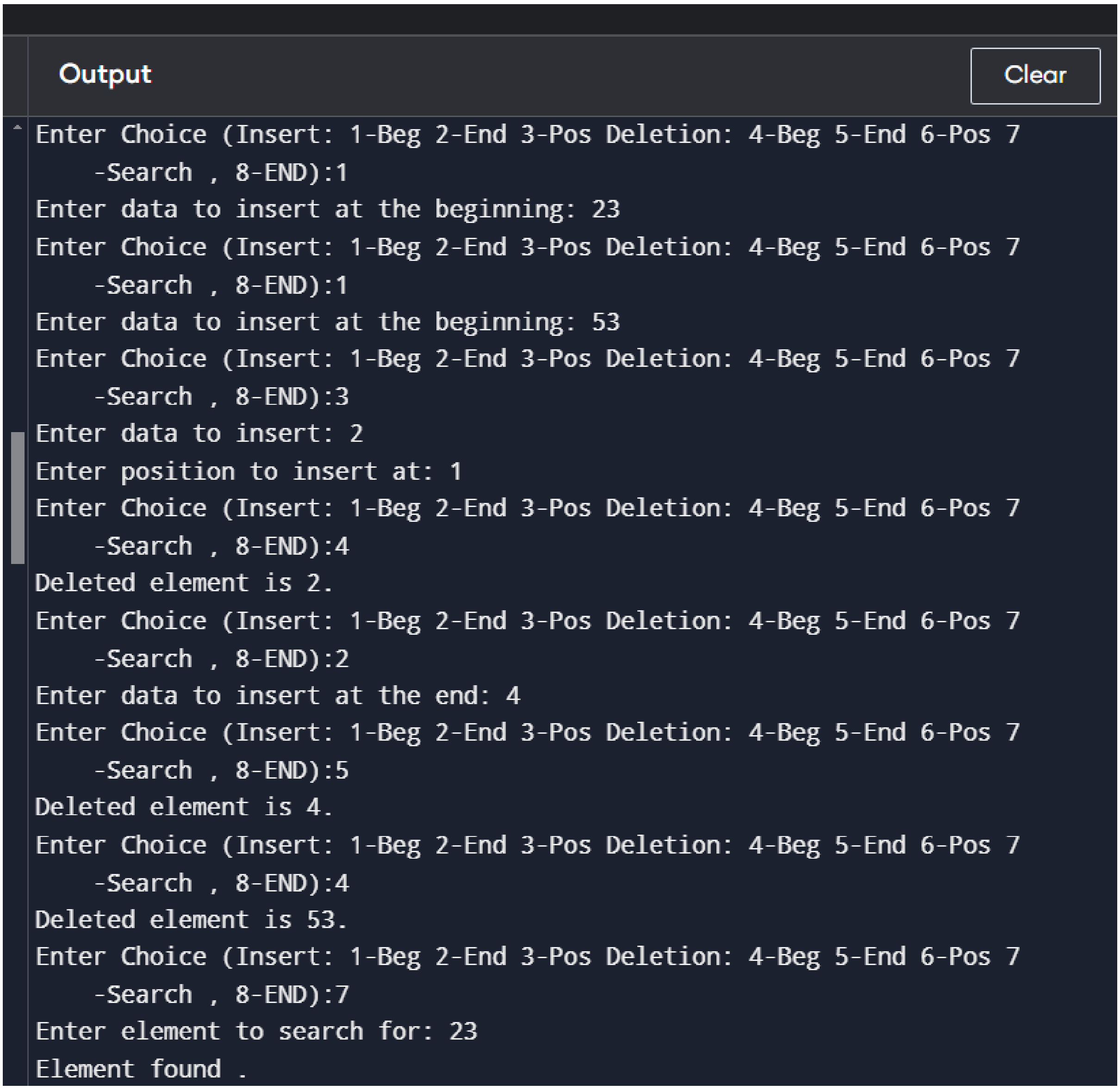
```
int i;
if (position == 1) {
    head = head->next;
    if (head != NULL) {
        head->prev = NULL;
    }
    printf("Deleted element is %d.\n", temp->data);
    free(temp);
    return;
}
for (i = 1; i < position; i++) {
    if (temp == NULL) {
        printf("Position exceeds the size of the list.\n");
        return;
    }
    temp = temp->next;
}
if (temp == NULL) {
    printf("Position exceeds the size of the list.\n");
    return;
}
if (temp->next != NULL) {
    temp->next->prev = temp->prev;
}
if (temp->prev != NULL) {
    temp->prev->next = temp->next;
}
printf("Deleted element is %d.\n", temp->data);
```

```
    free(temp);
}

void search(int key) {
    struct Node *temp = head;
    int flag=0;
    while (temp != NULL) {
        if (temp->data == key) {
            flag=1;
        }
        temp = temp->next;
    }
    if (flag==1){
        printf("Element found .\n");
    }
    else{
        printf("Element not found .");
    }
}

int main() {
    int choice, data, position;
    while (1) {
        printf("Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7-Search , 8-END):");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
```

```
    printf("Enter data to insert at the beginning: ");
    scanf("%d", &data);
    insert_b(data);
    break;
case 2:
    printf("Enter data to insert at the end: ");
    scanf("%d", &data);
    insert_e(data);
    break;
case 3:
    printf("Enter data to insert: ");
    scanf("%d", &data);
    printf("Enter position to insert at: ");
    scanf("%d", &position);
    insert_p(data, position);
    break;
case 4:
    delete_b();
    break;
case 5:
    delete_e();
    break;
case 6:
    printf("Enter position to delete from: ");
    scanf("%d", &position);
    delete_p(position);
    break;
case 7:
```

```
                printf("Enter element to search for: ");
                scanf("%d", &data);
                search(data);
                break;
            case 9:
                printf("Exiting program.\n");
                exit(0);

            default:
                printf("Invalid choice. Please try again.\n");
        }
    }

    return 0;
}
```

Output :



```
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):1
Enter data to insert at the beginning: 23
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):1
Enter data to insert at the beginning: 53
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):3
Enter data to insert: 2
Enter position to insert at: 1
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):4
Deleted element is 2.
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):2
Enter data to insert at the end: 4
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):5
Deleted element is 4.
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):4
Deleted element is 53.
Enter Choice (Insert: 1-Beg 2-End 3-Pos Deletion: 4-Beg 5-End 6-Pos 7
    -Search , 8-END):7
Enter element to search for: 23
Element found .
```