DATA STRUCTURES AND ALGORITHMS

<u>DIGITAL ASSIGNMENT-1</u>

NAME: VEDAANT AGARWAL

REGISTRATION NO: 24BBS0191

COURSE CODE: CBS1003

Q1. Write a menu driven program to implement the following operations on stack.

a. PUSH ()

b. POP ()

c. Display ()

Algorithm:

Define a Stack:

- A structure Stack with:
  - data: An array to store stack elements.
  - top: An integer to track the index of the top element.
  - maxSize: The maximum capacity of the stack.

Push Operation:

- **Input:** Stack s, element data.
- **Steps:**
  1. Check if the stack is full (top == maxSize - 1).
     - If yes, print "Stack Overflow" and return.
  2. Increment top.
  3. Add data to s.data[top].

Pop Operation:

- **Input:** Stack s.
- **Steps:**
    1. Check if the stack is empty (top == -1).
        - If yes, print "Stack Underflow" and return.
    2. Retrieve the element at s.data[top].
    3. Decrement top.
    4. Return the retrieved element.

Display Stack:

- **Input:** Stack s.
- **Steps:**
    1. If the stack is empty (top == -1), print "Stack is empty".
    2. Otherwise, print elements from s.data[top] to s.data[0].

Main Function:

1. Initialize a stack s with maxSize.
2. Provide menu options:
    - 1: Push an element.
    - 2: Pop an element.
    - 3: Display the stack.
3. Perform the chosen operation and display the updated stack.

Program:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
   int *items;
   int top;
   int maxSize;
}Stack;
void initializeStack(Stack *s, int n) {
   s->maxSize =n;
   s->items =(int *)malloc(n * sizeof(int));
   s->top =-1;
}
int isFull(Stack *s) {
```

```
    return s->top==s->maxSize - 1;
}
int isEmpty(Stack *s) {
    return s->top == -1;
}
void push(Stack *s, int element) {
    if (isFull(s)) {
        printf("Stack Overflow! Cannot push %d.\n", element);
        return;
    }
    s->items[++(s->top)] = element;
    printf("%d pushed onto the stack.\n", element);
}
int pop(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack Underflow! No elements to pop.\n");
        return -1;
    }
    return s->items[(s->top)--];
}
void display(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack is empty.\n");
        return;
    }
    printf("Stack elements: ");
    for (int i = s->top; i >= 0; i--) {
        printf("%d ", s->items[i]);
    }
    printf("\n");
}
void freeStack(Stack *s) {
    free(s->items);
}
int main() {
    Stack s;
    int n, choice, value;
    printf("Size of stack= ");
    scanf("%d", &n);
    initializeStack(&s, n);
    printf("\nEnter choice of operation\n");
    printf("1. Push\n");
    printf("2. Pop\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    while (1) {
        printf("Enter your choice: ");
```

```
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the value to push: ");
                scanf("%d", &value);
                push(&s, value);
                break;
            case 2:
                value = pop(&s);
                if (value != -1) {
                    printf("Popped element: %d\n", value);
                }
                break;
            case 3:
                display(&s);
                break;
            case 4:
                printf("Exiting program.\n");
                freeStack(&s);
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```

OUTPUT:

```
Size of stack= 5

Enter choice of operation
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 56
56 pushed onto the stack.
Enter your choice: 3
Stack elements: 56
Enter your choice: 1
Enter the value to push: 5
5 pushed onto the stack.
Enter your choice: 1
Enter the value to push: 65
65 pushed onto the stack.
Enter your choice: 3
Stack elements: 65 5 56
Enter your choice: 1
Enter the value to push: 88
88 pushed onto the stack.
Enter your choice: 1
Enter the value to push: 66
66 pushed onto the stack.
Enter your choice: 1
Enter the value to push: 55
Stack Overflow! Cannot push 55.
Enter your choice: 4
Exiting program.
```

```
Size of stack= 3

Enter choice of operation
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 56
56 pushed onto the stack.
Enter your choice: 2
Popped element: 56
Enter your choice: 2
Stack Underflow! No elements to pop.
Enter your choice: 3
Stack is empty.
Enter your choice: 4
Exiting program.
```

```
Size of stack= 3

Enter choice of operation
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 65
Invalid choice. Please try again.
Enter your choice: 2
Stack Underflow! No elements to pop.
Enter your choice: 1
Enter the value to push: 56
56 pushed onto the stack.
Enter your choice: 2
Popped element: 56
Enter your choice: 1
Enter the value to push: 66
66 pushed onto the stack.
Enter your choice: 4
Exiting program.
```

Q2. Write a menu driven program to implement the following operations on Queue:

a. Enqueue ()

b. Dequeue ()

c. Display ()

ALGORITHM:

Define a Queue:

- A structure Queue with:
  - data: An array to store queue elements.
  - front: An integer to track the index of the first element.
  - rear: An integer to track the index of the last element.
  - maxSize: The maximum capacity of the queue.

Enqueue Operation:

- **Input:** Queue q, element data.
- **Steps:**
  1. Check if the queue is full (rear == maxSize - 1).
     - If yes, print "Queue Overflow" and return.
  2. If front == -1, set front = 0.
  3. Increment rear.
  4. Add data to q.data[rear].

Dequeue Operation:

- **Input:** Queue q.
- **Steps:**
  1. Check if the queue is empty (front == -1 or front > rear).
     - If yes, print "Queue Underflow" and return.
  2. Retrieve the element at q.data[front].
  3. Increment front.
  4. If front > rear, reset front and rear to -1.
  5. Return the retrieved element.

Display Queue:

- **Input:** Queue q.
- **Steps:**
    1. If the queue is empty (front == -1), print "Queue is empty".
    2. Otherwise, print elements from q.data[front] to q.data[rear].

Main Function:

1. Initialize a queue q with maxSize.
2. Provide menu options:
    - 1: Enqueue an element.
    - 2: Dequeue an element.
    - 3: Display the queue.
3. Perform the chosen operation and display the updated queue.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
   int *items;
   int front, rear;
   int maxSize;
}Queue;
void initializeQueue(Queue *q, int size) {
   q->maxSize = size;
   q->items = (int *)malloc(size * sizeof(int));
   q->front = -1;
   q->rear = -1;
}
int isFull(Queue *q) {
   return (q->rear + 1) % q->maxSize == q->front;
}
int isEmpty(Queue *q) {
   return q->front == -1;
}
void enqueue(Queue *q, int element) {
   if (isFull(q)) {
      printf("Queue Overflow! Cannot enqueue %d.\n", element);
      return;
   }
   if (isEmpty(q)) q->front = 0;
   q->rear = (q->rear + 1) % q->maxSize;
```

```c
    q->items[q->rear] = element;
    printf("%d enqueued to the queue.\n", element);
}
int dequeue(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue Underflow! No elements to dequeue.\n");
        return -1;
    }
    int element = q->items[q->front];
    if (q->front == q->rear) {
        q->front = -1;
        q->rear = -1;
    } else {
        q->front = (q->front + 1) % q->maxSize;
    }
    return element;
}
void display(Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue elements: ");
    int i = q->front;
    while (1) {
        printf("%d ", q->items[i]);
        if (i == q->rear) break;
        i = (i + 1) % q->maxSize;
    }
    printf("\n");
}
void freeQueue(Queue *q) {
    free(q->items);
}
int main() {
    Queue q;
    int size, choice, value;
    printf("Enter the size of the queue: ");
    scanf("%d", &size);
    initializeQueue(&q, size);
    printf("\nEnter choice of operation\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");
    while (1) {
        printf("Enter your choice: ");
```

```
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the value to enqueue: ");
                scanf("%d", &value);
                enqueue(&q, value);
                break;
            case 2:
                value = dequeue(&q);
                if (value != -1) {
                    printf("Dequeued element: %d\n", value);
                }
                break;
            case 3:
                display(&q);
                break;
            case 4:
                printf("Exiting program.\n");
                freeQueue(&q);
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```

OUTPUT:

o

```
Enter the size of the queue: 4

Enter choice of operation
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 65
65 enqueued to the queue.
Enter your choice: 1
Enter the value to enqueue: 55
55 enqueued to the queue.
Enter your choice: 1
Enter the value to enqueue: 44
44 enqueued to the queue.
Enter your choice: 1
Enter the value to enqueue: 33
33 enqueued to the queue.
Enter your choice: 2
Dequeued element: 65
Enter your choice: 2
Dequeued element: 55
Enter your choice: 1
Enter the value to enqueue: 64
64 enqueued to the queue.
Enter your choice: 3
Queue elements: 44 33 64
Enter your choice: 4
Exiting program.
```

```
Enter the size of the queue: 3

Enter choice of operation
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 25
25 enqueued to the queue.
Enter your choice: 1
Enter the value to enqueue: 22
22 enqueued to the queue.
Enter your choice: 1
Enter the value to enqueue: 544
544 enqueued to the queue.
Enter your choice: 1
Enter the value to enqueue: 35
Queue Overflow! Cannot enqueue 35.
Enter your choice: 3
Queue elements: 25 22 544
Enter your choice: 4
Exiting program.
```

```
Enter the size of the queue: 3

Enter choice of operation
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Queue Underflow! No elements to dequeue.
Enter your choice: 3
Queue is empty.
Enter your choice: 1
Enter the value to enqueue: 4
4 enqueued to the queue.
Enter your choice: 3
Queue elements: 4
Enter your choice: 4
Exiting program.
```

Q3. Write a menu driven program to implement the following operations on circular Queue:

a. Enqueue ()

b. Dequeue ()

c. Display ()

ALGORITHM:

Define a Circular Queue:

- A structure CircularQueue with:
  - data: An array to store queue elements.
  - front: An integer to track the index of the first element.
  - rear: An integer to track the index of the last element.
  - maxSize: The maximum capacity of the queue.

Enqueue Operation:

- **Input:** CircularQueue cq, element data.
- **Steps:**
  1. Check if the queue is full ((rear + 1) % maxSize == front).
     - If yes, print "Queue Overflow" and return.
  2. If front == -1, set front = 0.
  3. Increment rear using rear = (rear + 1) % maxSize.
  4. Add data to cq.data[rear].

Dequeue Operation:

- **Input:** CircularQueue cq.
- **Steps:**
  1. Check if the queue is empty (front == -1).
     - If yes, print "Queue Underflow" and return.
  2. Retrieve the element at cq.data[front].
  3. If front == rear, reset front and rear to -1.
  4. Otherwise, increment front using front = (front + 1) % maxSize.
  5. Return the retrieved element.

Display Circular Queue:

- **Input:** CircularQueue cq.
- **Steps:**
    1. If the queue is empty (front == -1), print "Queue is empty".
    2. Otherwise:
        - Start from cq.data[front] and traverse circularly until rear.

Main Function:

1. Initialize a circular queue cq with maxSize.
2. Provide menu options:
    - 1: Enqueue an element.
    - 2: Dequeue an element.
    - 3: Display the queue.
3. Perform the chosen operation and display the updated queue.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
  int *items;
  int front, rear, maxSize;
} CircularQueue;
void initializeQueue(CircularQueue *q, int size) {
  q->maxSize = size;
  q->items = (int *)malloc(size * sizeof(int));
  q->front = q->rear = -1;
}
int isFull(CircularQueue *q) {
  return (q->rear + 1) % q->maxSize == q->front;
}
int isEmpty(CircularQueue *q) {
  return q->front == -1;
}
void enqueue(CircularQueue *q, int element) {
  if (isFull(q)) {
    printf("Queue Overflow! Cannot enqueue %d.\n", element);
    return;
  }
  if (isEmpty(q))
    q->front = 0;
  q->rear = (q->rear + 1) % q->maxSize;
```

```c
    q->items[q->rear] = element;
    printf("%d enqueued to the queue.\n", element);
}
int dequeue(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Queue Underflow! No elements to dequeue.\n");
        return -1;
    }

    int element = q->items[q->front];

    if (q->front == q->rear)
        q->front = q->rear = -1;
    else
        q->front = (q->front + 1) % q->maxSize;

    return element;
}
void display(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue elements: ");
    for (int i = q->front;; i = (i + 1) % q->maxSize) {
        printf("%d ", q->items[i]);
        if (i == q->rear)
            break;
    }
    printf("\n");
}
void freeQueue(CircularQueue *q) {
    free(q->items);
}
int main() {
    CircularQueue q;
    int size, choice, value;
    printf("Enter the size of the circular queue: ");
    scanf("%d", &size);
    initializeQueue(&q, size);
    printf("\nChoice of Operations\n");
    printf("1. Enqueue\n");
    printf("2. Dequeue\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    while (1) {
        printf("Enter your choice: ");
```

```
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the value to enqueue: ");
                scanf("%d", &value);
                enqueue(&q, value);
                break;
            case 2:
                value = dequeue(&q);
                if (value != -1)
                    printf("Dequeued element: %d\n", value);
                break;
            case 3:
                display(&q);
                break;
            case 4:
                freeQueue(&q);
                printf("Exiting program.\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```

OUTPUT:

```
Enter the size of the circular queue: 4

Choice of Operations
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Queue Underflow! No elements to dequeue.
Enter your choice: 1
Enter the value to enqueue: 65
65 enqueued to the queue.
Enter your choice: 3
Queue elements: 65
Enter your choice: 2
Dequeued element: 65
Enter your choice: 1
Enter the value to enqueue: 56
56 enqueued to the queue.
Enter your choice: 1
Enter the value to enqueue: 65
65 enqueued to the queue.
Enter your choice: 1
Enter the value to enqueue: 655
655 enqueued to the queue.
Enter your choice: 2
Dequeued element: 56
Enter your choice: 2
Dequeued element: 65
Enter your choice: 2
Dequeued element: 655
Enter your choice: 3
Queue is empty.
```

```
Enter the size of the circular queue: 2

Choice of Operations
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 56
56 enqueued to the queue.
Enter your choice: 1
Enter the value to enqueue: 666
666 enqueued to the queue.
Enter your choice: 1
Enter the value to enqueue: 22
Queue Overflow! Cannot enqueue 22.
Enter your choice: 2
Dequeued element: 56
Enter your choice: 3
Queue elements: 666
Enter your choice: 4
Exiting program.
```

```
Enter the size of the circular queue: 5

Choice of Operations
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 56
Invalid choice. Please try again.
Enter your choice: 1
Enter the value to enqueue: 44
44 enqueued to the queue.
Enter your choice: 1
Enter the value to enqueue: 66
66 enqueued to the queue.
Enter your choice: 3
Queue elements: 44 66
Enter your choice: 2
Dequeued element: 44
Enter your choice: 4
Exiting program.
```

Q4. Write a menu driven program to implement the following operations on singly linked list:

a. Insertion ()

i. Beginning

ii. End

iii. At a given position

b. Deletion ()

i. Beginning

ii. End

iii. At a given position

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

ALGORITHM:

Define a Node:

- Create a structure node with two fields:
    - data: Stores the value of the node.
    - next: Points to the next node in the list.

Insert at Beginning (insertAtBeg):

- Input: start (head of the list), data (data to be inserted).
- Allocate memory for a new node.
- Set the new node's data to the input data.
- Set the new node's next to point to the current start.
- Update start to point to the new node.
- Return the updated start.

Insert at End (insertAtEnd):

- Input: start (head of the list), data (data to be inserted).
- Create a new node with the input data.
- Traverse the list to find the last node (node where next is NULL).
- Set the last node's next to the new node.
- Set the new node's next to NULL.
- Return the updated start.

Insert at Position (insertAtPos):

- Input: start (head of the list), data (data to be inserted), pos (position).
- If the position is 0, call insertAtBeg to insert at the beginning and return.
- Traverse the list to find the node at position pos-1.
- If the node is NULL, print an error message and return.
- Insert the new node after the node at pos-1 by updating the pointers.
- Return the updated start.

Delete at Beginning (deleteAtBeg):

- Input: start (head of the list).

- If the list is empty (start == NULL), print an error message and return.
- Store the current start in a temporary pointer.
- Set start to the next node (start->next).
- Free the memory of the temporary node.
- Return the updated start.

Delete at End (deleteAtEnd):

- Input: start (head of the list).
- If the list is empty (start == NULL), print an error message and return.
- If the list has only one node (start->next == NULL), free the node and set start to NULL.
- Otherwise, traverse the list to find the second last node.
- Set the second last node's next to NULL and free the last node.
- Return the updated start.

Delete at Position (deleteAtPos):

- Input: start (head of the list), pos (position of the node to delete).
- If the list is empty (start == NULL), print an error message and return.
- If the position is 0, call deleteAtBeg to delete the first node and return.
- Traverse the list to find the node at position pos.
- If the node is NULL, print an error message and return.
- Update the next pointer of the previous node to skip the node to be deleted.
- Free the memory of the deleted node.
- Return the updated start.

Search Operation:

- Input: start (head of the list), data (data to search).
- Traverse the list, checking each node's data.
- If the data is found, print the position and return.
- If the end of the list is reached without finding the data, print an error message.

Display the List:

- Input: start (head of the list).
- Traverse the list, printing each node's data.
- End the display with NULL to indicate the end of the list.

Main Function:

- Initialize start as NULL.
- Provide a menu of operations:
    - 1: Insert at beginning.
    - 2: Insert at end.
    - 3: Insert at a specific position.
    - 4: Delete the first node.
    - 5: Delete the last node.
    - 6: Delete at a specific position.
    - 7: Search for an element.
- Execute the corresponding function based on user input.
- Display the updated list after each operation.
- Allow the user to continue or exit based on input.

PROGRAM:

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

```
#include<stdlib.h>
struct node{
   int data;
   struct node* next;
};
struct node* insertAtBeg(struct node* start, int data)
{
   struct node* temp=(struct node*)malloc(sizeof(struct node));
   temp->data=data;
   temp->next=start;
   start=temp;
   return start;
}
struct node* insertAtEnd(struct node* start, int data)
{
   struct node* temp=(struct node*)malloc(sizeof(struct node));
   struct node* p=start;
   while(p->next!=NULL)
   {
      P=P->next;
   }
   temp->data=data;
   p->next=temp;
   temp->next=NULL;
   return start;
}
struct node* insertAtPos(struct node* start, int data, int pos)
{
   struct node* temp=(struct node*)malloc(sizeof(struct node));
   if(pos==0)
   {
      start=insertAtBeg(start,data);
      return start;
   }
   struct node* p=start;
   for(int i=0;i<pos-1&&p!=NULL;i++)
   {
      p=p->next;
   }
   if(p==NULL){
      printf("The list is not big enough to insert the element at the given position\n");
      return start;
   }
   temp->data=data;
   temp->next=p->next;
   p->next=temp;
   return start;
```

```
}
struct node* deleteAtBeg(struct node* start)
{
   if(start==NULL)
   {
      printf("The given linked list is empty. No element deleted\n");
      return start;
   }
   struct node*temp=start;
   start=start->next;
   free(temp);
   return start;
}
struct node* deleteAtEnd(struct node* start)
{
   if(start==NULL)
   {
      printf("The given linked list is empty. No element deleted\n");
      return start;
   }
   if(start->next==NULL)
   {
      free(start);
      start=NULL;
      return start;
   }
   struct node* p=start;
   while(p->next->next!=NULL)
   {
      p=p->next;
   }
   free(p->next);
   p->next=NULL;
   return start;
}
struct node* deleteAtPos(struct node* start, int pos)
{
   if(start==NULL)
   {
      printf("The given linked list is empty. No element deleted\n");
      return start;
   }
   struct node* p=start;
   struct node* prev=NULL;
   if(pos==0)
   {
      start=deleteAtBeg(start);
```

```
        return start;
    }
    for(int i=0;i<pos&&p!=NULL;i++)
    {
        prev=p;
        p=p->next;
    }
    if(p=NULL)
    {
        printf("The given linked list is not long enough to delete the element at the given
position\n");
        return start;
    }
    prev->next=p->next;
    free(p);
    return start;
}
void search(struct node* start, int data)
{
    struct node* p=start;
    int counter=0;
    while(p!=NULL)
    {
        counter++;
        if(p->data==data)
        {
            printf("The given data exists in the linked list at %d position\n",counter);
            return;
        }
        p=p->next;
    }
    printf("The given data does not exist in the linked list\n");
    return;
}
void display(struct node* start)
{
    struct node* p=start;
    printf("THE LINKED LIST \n");
    while(p!=NULL)
    {
        printf("%d =>",p->data);
        p=p->next;
    }
    printf("NULL \n");
    return;
}
int main()
```

```
{
   struct node* start=NULL;
   int choice=1;
   printf("CHOICE OF OPERATIONS \n");
   printf("1 for INSERTION AT BEGINNING \n");
   printf("2 for INSERTION AT END \n");
   printf("3 for INSERTION AT A PARTICULAR POSITION \n");
   printf("4 for DELETION AT BEGINNING \n");
   printf("5 for DELETION AT END \n");
   printf("6 for DELETION AT A PARTICULAR POSITION \n");
   printf("7 for SEARCHING AN ELEMENT IN THE LINKED LIST \n");
   do{
      int x;
      printf("ENTER CHOICE \n");
      scanf("%d",&x);
      switch(x)
      {
         case 1:{
         int data;
         printf("Enter the data to be added to the linked list: ");
         scanf("%d",&data);
         start=insertAtBeg(start,data);
         printf("Data added\n");
         break;
         }
         case 2:{
            int data;
            printf("Enter the data to be added at the end of the linked list\n");
            scanf("%d",&data);
            start=insertAtEnd(start,data);
            printf("Data added\n");
            break;
         }
         case 3:{
            int data,pos;
            printf("Enter the data to be added \n");
            scanf("%d",&data);
            printf("\nEnter the position at which data has to be added\n");
            scanf("%d",&pos);
            start=insertAtPos(start,data,pos-1);
            printf("Data added\n");
            break;
         }
         case 4:{
            start=deleteAtBeg(start);
            printf("Data deleted\n");
            break;
```

```
            }
            case 5:{
               start=deleteAtEnd(start);
               printf("Data deleted\n");
               break;
            }
            case 6:{
               int pos;
               printf("Enter the position for the data to be deleted: ");
               scanf("%d",&pos);
               start=deleteAtPos(start,pos);
               printf("\nData Deleted\n");
               break;
            }
            case 7:{
               int data;
               printf("Enter data to be searched in the linked list\n");
               scanf("%d",&data);
               search(start,data);
               break;
            }
            default:
            printf("Invalid choice.\n");
         }
         display(start);
         printf("Enter 1 to continue use of program or else any other integer\n");
         scanf("%d",&choice);
      }while(choice==1);
      return 0;
   }
```

OUTPUT:

```
CHOICE OF OPERATIONS
1 for INSERTION AT BEGINNING
2 for INSERTION AT END
3 for INSERTION AT A PARTICULAR POSITION
4 for DELETION AT BEGINNING
5 for DELETION AT END
6 for DELETION AT A PARTICULAR POSITION
7 for SEARCHING AN ELEMENT IN THE LINKED LIST
ENTER CHOICE
1
Enter the data to be added to the linked list: 5
Data added
THE LINKED LIST
5 =>NULL
Enter 1 to continue use of program or else any other integer
1
ENTER CHOICE
2
Enter the data to be added at the end of the linked list
1
Data added
THE LINKED LIST
5 =>1 =>NULL
Enter 1 to continue use of program or else any other integer
1
ENTER CHOICE
6
Enter the position for the data to be deleted: 1

Data Deleted
THE LINKED LIST
5 =>NULL
Enter 1 to continue use of program or else any other integer
5
```

```
Enter the data to be added to the linked list: 44
Data added
THE LINKED LIST
44 =>NULL
Enter 1 to continue use of program or else any other integer
1
ENTER CHOICE
4
Data deleted
THE LINKED LIST
NULL
Enter 1 to continue use of program or else any other integer
1
ENTER CHOICE
5
The given linked list is empty. No element deleted
Data deleted
THE LINKED LIST
NULL
Enter 1 to continue use of program or else any other integer
1
ENTER CHOICE
3
Enter the data to be added
0

Enter the position at which data has to be added
0
The list is not big enough to insert the element at the given position
Data added
THE LINKED LIST
NULL
Enter 1 to continue use of program or else any other integer
1
ENTER CHOICE
1
Enter the data to be added to the linked list: 22
Data added
THE LINKED LIST
22 =>NULL
```

```
ENTER CHOICE
1
Enter the data to be added to the linked list: 11
Data added
THE LINKED LIST
11 =>NULL
Enter 1 to continue use of program or else any other integer
1
ENTER CHOICE
2
Enter the data to be added at the end of the linked list
6565
Data added
THE LINKED LIST
11 =>6565 =>NULL
Enter 1 to continue use of program or else any other integer
1
ENTER CHOICE
4
Data deleted
THE LINKED LIST
6565 =>NULL
Enter 1 to continue use of program or else any other integer
1
ENTER CHOICE
7
Enter data to be searched in the linked list
656556
The given data does not exist in the linked list
THE LINKED LIST
6565 =>NULL
Enter 1 to continue use of program or else any other integer
1
ENTER CHOICE
7
Enter data to be searched in the linked list
6565
The given data exists in the linked list at 1 position
```

Q5. Write a menu driven program to implement the following operations on Doubly linked list:

a. Insertion ()

i. Beginning

ii. End

iii. At a given position

b. Deletion ()

i. Beginning

ii. End

iii. At a given position

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

ALGORITHM:

Define a Node:
- A node structure with three fields:
  - data: Stores the value of the node.
  - prev: Points to the previous node in the list.
  - next: Points to the next node in the list.

Create a New Node (createNewNode):
- Allocate memory for a new node.
- Set the node's data, prev, and next to the provided values (prev = NULL, next = NULL).
- Return the new node.

Insert at Beginning (insertAtBeg):
- Input: start (head of the list), tail (tail of the list), data (data to insert).
- Allocate memory for a new node.
- If the list is empty (start == NULL), set both start and tail to the new node.
- Otherwise, set the new node's next to start, and update the prev pointer of the current start to point to the new node.
- Set start to the new node.

Insert at End (insertAtEnd):
- Input: start (head of the list), tail (tail of the list), data (data to insert).
- Allocate memory for a new node.
- If the list is empty (start == NULL), set both start and tail to the new node.
- Otherwise, set the current tail's next to the new node and the new node's prev to the current tail.
- Set tail to the new node.

Insert at Position (insertAtPos):
- Input: start (head of the list), tail (tail of the list), data (data to insert), pos (position).
- If the position is 0, call insertAtBeg to insert at the beginning and return.

- Traverse the list until reaching the node at position pos-1.
- If the node is NULL, print an error and free the new node.
- Insert the new node after the node at position pos-1, updating the prev and next pointers of adjacent nodes.

Delete at Beginning (deleteAtBeg):

- Input: start (head of the list), tail (tail of the list).
- If the list is empty (start == NULL), print an error and return.
- Otherwise, set start to the next node, and update the prev pointer of the new start to NULL.
- If the list becomes empty (start == NULL), set tail to NULL.
- Free the old start node.

Delete at End (deleteAtEnd):

- Input: start (head of the list), tail (tail of the list).
- If the list is empty (tail == NULL), print an error and return.
- If the list has only one node (start == tail), set both start and tail to NULL.
- Otherwise, set tail to the previous node and set its next pointer to NULL.
- Free the old tail node.

Delete at Position (deleteAtPos):

- Input: start (head of the list), tail (tail of the list), pos (position).
- If the list is empty (start == NULL), print an error and return.
- If the position is 0, call deleteAtBeg to delete the first node and return.
- Traverse the list until reaching the node at position pos.
- If the node is NULL, print an error.
- If the node is the first (start), delete it using deleteAtBeg.
- If the node is the last (tail), delete it using deleteAtEnd.
- Otherwise, update the prev and next pointers of adjacent nodes and free the current node.

Search Operation (search):

- Input: start (head of the list), data (data to search).
- Traverse the list, checking each node's data.
- If the data is found, print the position and return.
- If the end of the list is reached without finding the data, print an error.

Display the List (display):

- Input: start (head of the list).
- Traverse the list and print each node's data from start to tail.
- Also display the list in reverse order, starting from tail to start.

Main Function:

- Initialize start and tail as NULL.
- Provide a menu with options:
  - 1: Insert at beginning.
  - 2: Insert at end.
  - 3: Insert at a specific position.
  - 4: Delete the first node.
  - 5: Delete the last node.
  - 6: Delete at a specific position.
  - 7: Search for an element.
- Execute the corresponding function based on user input.

- Display the updated list after each operation.
- Allow the user to continue or exit based on input.

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
struct node{
   struct node* prev;
   int data;
   struct node* next;
};
struct node* createNewNode(int data)
{
   struct node* newNode=(struct node*)malloc(sizeof(struct node));
   newNode->data=data;
   newNode->prev=NULL;
   newNode->next=NULL;
   return newNode;
}
void insertAtBeg(struct node** start,struct node** tail, int data)
{
   struct node* newNode=createNewNode(data);
   if(*start==NULL)
   {
      *start=newNode;
      *tail=newNode;
      return;
   }
   newNode->next=start;
   (*start)->prev=newNode;
   *start=newNode;
   return;
}
void insertAtEnd(struct node** start,struct node** tail, int data)
{
   struct node* newNode=createNewNode(data);
   if(*start==NULL)
   {
      *start=newNode;
      *tail=newNode;
      return;
   }
   (tail)->next=newNode;
   newNode->prev=*tail;
   *tail=newNode;
   return;
}
```

```c
void insertAtPos(struct node** start, struct node** tail, int data, int pos) {
   struct node *newNode = createNewNode(data);
   if(pos == 0) {
      insertAtBeg(start, tail, data);
      return;
   }
   struct node* p = *start;
   for(int i=0; i<pos-1 && p != NULL; i++) {
      p = p->next;
   }
   if(p == NULL) {
      printf("There is not enough spaces in the linked list.\n\n");
      free(newNode);
      return;
   }
   newNode->next = p->next;
   newNode->prev = p;
   p->next->prev = newNode;
   p->next = newNode;
   return;
}
void deleteAtBeg(struct node** start, struct node** tail) {
   if (*start == NULL) {
      printf("List is empty. Nothing to delete.\n\n");
      return;
   }
   struct node* temp = *start;
   *start = (*start)->next;
   if (*start != NULL) {
      (*start)->prev = NULL;
   }
   else {
      *tail = NULL;
   }
   free(temp);
   printf("First node deleted.\n\n");
}
void deleteAtEnd(struct node** start, struct node** tail) {
   if (*tail == NULL) {
      printf("List is empty. Nothing to delete.\n\n");
      return;
   }
   struct node* temp = *tail;
   if (*tail == *start) {
      *start = NULL;
      *tail = NULL;
   } else {
```

```
        *tail = (*tail)->prev;
        (*tail)->next = NULL;
    }
    free(temp);
    printf("Last node deleted.\n\n");
}
void deleteAtPos(struct node** start, struct node** tail, int pos) {
    if (*start == NULL) {
        printf("List is empty. Nothing to delete.\n");
        return;
    }
    struct node* temp = *start;
    int curr = 1;
    while (temp != NULL && curr < pos) {
        temp = temp->next;
        curr++;
    }
    if (temp == NULL) {
        printf("Invalid position %d. No node found.\n", pos);
        return;
    }
    if (temp == *start) {
        deleteAtBeg(start, tail);
    }
    else if (temp == *tail) {
        deleteAtEnd(start, tail);
    }
    else {
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
    }

    free(temp);
    printf("Node at position %d deleted.\n", pos);
}
void search(struct node* start, int data) {
    struct node* p = start;
    int counter = 0;
    while(p != NULL) {
        counter++;
        if(p->data == data) {
            printf("The provided data exists in the linked list at %d position.\n\n", counter);
            return;
        }
        p = p->next;
    }
    printf("The given data does not exist in the linked list.\n\n");
```

```
    return;
}

void display(struct node* start) {
    struct node* p = start;
    printf("\n\nLIST : NULL -> ");
    while(p != NULL) {
        printf("%d -> ", p->data);
        p = p->next;
    }
    printf("NULL\n");
    p = start;
    printf("       NULL <- ");
    while(p != NULL) {
        printf("%d <- ", p->data);
        p = p->next;
    }
    printf("NULL\n\n");
    return;
}
int main() {
    struct node *start = NULL;
    struct node *tail = NULL;
    int choice = 1;
     printf("Enter 1 for insertion at the beginning.\n");
     printf("Enter 2 for insertion at the end.\n");
     printf("Enter 3 for insertion at a particular position.\n");
     printf("4 for deleting the first node.\n"
     printf("Enter 5 for deleting the last node.\n");
     printf("Enter 6 for deleting the node at a particular position.\n");
     printf("Enter 7 for searching an element in the linked list.\n");
    do {
        int x;
        printf("Enter Choice \n");
        scanf("%d", &x);
        printf("\n\n");

        switch(x) {
            case 1 : {
                int data;
                printf("Enter the data to be added : ");
                scanf("%d", &data);
                insertAtBeg(&start, &tail, data);
                printf("Data added.\n\n");
                break;
            }
            case 2 : {
```

```
        int data;
        printf("Enter the data to be added : ");
        scanf("%d", &data);
        insertAtEnd(&start, &tail, data);
        printf("Data added.\n\n");
        break;
    }
    case 3 : {
        int data, pos;
        printf("Enter the data to be added : ");
        scanf("%d", &data);
        printf("Enter the position for the data to be added : ");
        scanf("%d", &pos);
        insertAtPos(&start, &tail, data, pos-1);
        printf("Data added.\n\n");
        break;
    }
    case 4 : {
        deleteAtBeg(&start, &tail);
        printf("Data deleted.\n\n");
        break;
    }
    case 5 : {
        deleteAtEnd(&start, &tail);
        printf("Data deleted.\n\n");
        break;
    }
    case 6 : {
        int pos;
        printf("Enter the position for the data to be deleted : ");
        scanf("%d", &pos);
        deleteAtPos(&start, &tail, pos);
        printf("Data deleted.\n\n");
        break;
    }
    case 7 : {
        int data;
        printf("Enter the data to be searched for : ");
        scanf("%d", &data);
        search(start, data);
        break;
    }
    default : {
        printf("Invalid choice.\n\n");
    }
}
display(start);
```

```
    printf("Enter 1 to continue use of the program.\nEnter any other integer to exit.\nCHOICE : ");
    scanf("%d", &choice);
  } while(choice == 1);
  return 0;
}
```

OUTPUT:

```
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 1


Enter the data to be added : 1
Data added.




LIST : NULL -> 1 -> NULL
       NULL <- 1 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 2


Enter the data to be added : 2
Data added.




LIST : NULL -> 1 -> 2 -> NULL
       NULL <- 1 <- 2 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE :
```

```
LIST : NULL -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL
       NULL <- 1 <- 2 <- 3 <- 4 <- 5 <- 6 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 3


Enter the data to be added : 69
Enter the position for the data to be added : 4
Data added.



LIST : NULL -> 1 -> 2 -> 3 -> 69 -> 4 -> 5 -> 6 -> NULL
       NULL <- 1 <- 2 <- 3 <- 69 <- 4 <- 5 <- 6 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE :
```

```
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 4


First node deleted.

Data deleted.



LIST : NULL -> 2 -> 3 -> 69 -> 4 -> 5 -> 6 -> NULL
       NULL <- 2 <- 3 <- 69 <- 4 <- 5 <- 6 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 5


Last node deleted.

Data deleted.



LIST : NULL -> 2 -> 3 -> 69 -> 4 -> 5 -> NULL
       NULL <- 2 <- 3 <- 69 <- 4 <- 5 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE :
```

```
LIST : NULL -> 2 -> 3 -> 69 -> 4 -> 5 -> NULL
       NULL <- 2 <- 3 <- 69 <- 4 <- 5 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 6


Enter the position for the data to be deleted : 3
Node at position 3 deleted.
Data deleted.



LIST : NULL -> 2 -> 3 -> 4 -> 5 -> NULL
       NULL <- 2 <- 3 <- 4 <- 5 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE :
```

```
Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 1
Enter 1 for insertion at the beginning.
Enter 2 for insertion at the end.
Enter 3 for insertion at a particular position.
Enter 4 for deleting the first node.
Enter 5 for deleting the last node.
Enter 6 for deleting the node at a particular position.
Enter 7 for searching an element in the linked list.
CHOICE : 7



Enter the data to be searched for : 69
The given data does not exist in the linked list.




LIST : NULL -> 2 -> 3 -> 4 -> 5 -> NULL
       NULL <- 2 <- 3 <- 4 <- 5 <- NULL

Enter 1 to continue use of the program.
Enter any other integer to exit.
CHOICE : 7
```