SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Course Code: CBS1003

Coure Name: Data Structures and Algorithms

Assessment-1

Name: Samar Reja

Reg.no:24BBS0117

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

a. PUSH()

b. POP()

c. Display()

Algorithm:

StackOperations()

Input: Stack S

Output: Perform PUSH, POP, or Display based on the user's choice

Initialize top ← -1

Repeat the following steps until the user exits:

Print menu: "1. PUSH, 2. POP, 3. Display, 4. Exit"

Read user choice

```
If choice = 1 (PUSH):
    If top = maxSize - 1, print "Stack Overflow"
    Else:
      Read value to push
      Increment top ← top + 1
      Set S[top] ← value
      Print "Pushed value into stack"
Else if choice = 2 (POP):
    If top = -1, print "Stack Underflow"
    Else:
      Print "Popped value: S[top]"
      Decrement top ← top - 1
    Else if choice = 3 (Display):
    If top = -1, print "Stack is empty"
    Else:
      Print stack elements from top to 0
  Else if choice = 4, exit
  Else, print "Invalid choice"
End
```

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
int stack[SIZE], top = -1;
void push() {
   int value;
   if (top == SIZE - 1) {
      printf("Stack Overflow! Cannot add more elements.\n");
   } else {
      printf("Enter the value to push: ");
      scanf("%d", &value);
      stack[++top] = value;
      printf("%d pushed into the stack.\n", value);
   }
}
void pop() {
   if (top == -1) {
      printf("Stack Underflow! No elements to pop.\n");
   } else {
      printf("%d popped from the stack.\n", stack[top--]);
   }
}
```

```
void display() {
    if (top == -1) {
        printf("Stack is empty.\n");
    } else {
        printf("Stack elements are:\n");
        for (int i = top; i>= 0; i--) {
            printf("%d\n", stack[i]);
        }
    }
}
int main() {
    int choice;
    while (1) {
        printf("\nStack Operations 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 the program.\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}
```

Output:

```
Stack Operations Menu:
1. PUSH
2. POP
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 16
16 pushed into the stack.

Stack Operations Menu:
Stack Operations Menu:
1. PUSH
2. POP
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 12
12 pushed into the stack.

Stack Operations Menu:
1. PUSH
2. POP
3. Display
4. Exit
Enter your choice: 3
Stack elements are:
12
16

Stack Operations Menu:
1. PUSH
2. POP
3. Display
4. Exit
Enter your choice: 4
Exiting the program.
```

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

a. Enqueue()

b. Dequeue()

c. Disaply()

Algorithm:

QueueOperations()

Input: Queue Q

Output: Perform Enqueue, Dequeue, or Display based on the user’s choice

Initialize front ← -1, rear ← -1

Repeat the following steps until the user exits:

Print menu: "1. Enqueue, 2. Dequeue, 3. Display, 4. Exit"

Read user choice

If choice = 1 (Enqueue):

If rear = maxSize - 1, print "Queue Overflow"

Else:

Read value to enqueue

If front = -1, set front ← 0

Increment rear ← rear + 1

Set Q[rear] ← value

Print "Enqueued value into queue"

Else if choice = 2 (Dequeue):

If front = -1 or front > rear, print "Queue Underflow"

Else:

Print "Dequeued value: Q[front]"

Increment front ← front + 1

If front > rear, reset front ← -1 and rear ← -1

Else if choice = 3 (Display):

If front = -1, print "Queue is empty"

Else:

Print queue elements from front to rear

Else if choice = 4, exit

Else, print "Invalid choice"

End

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
struct Queue {
    int arr[MAX];
    int front, rear;
};
void initialize(struct Queue* q) {
    q->front = -1;
```

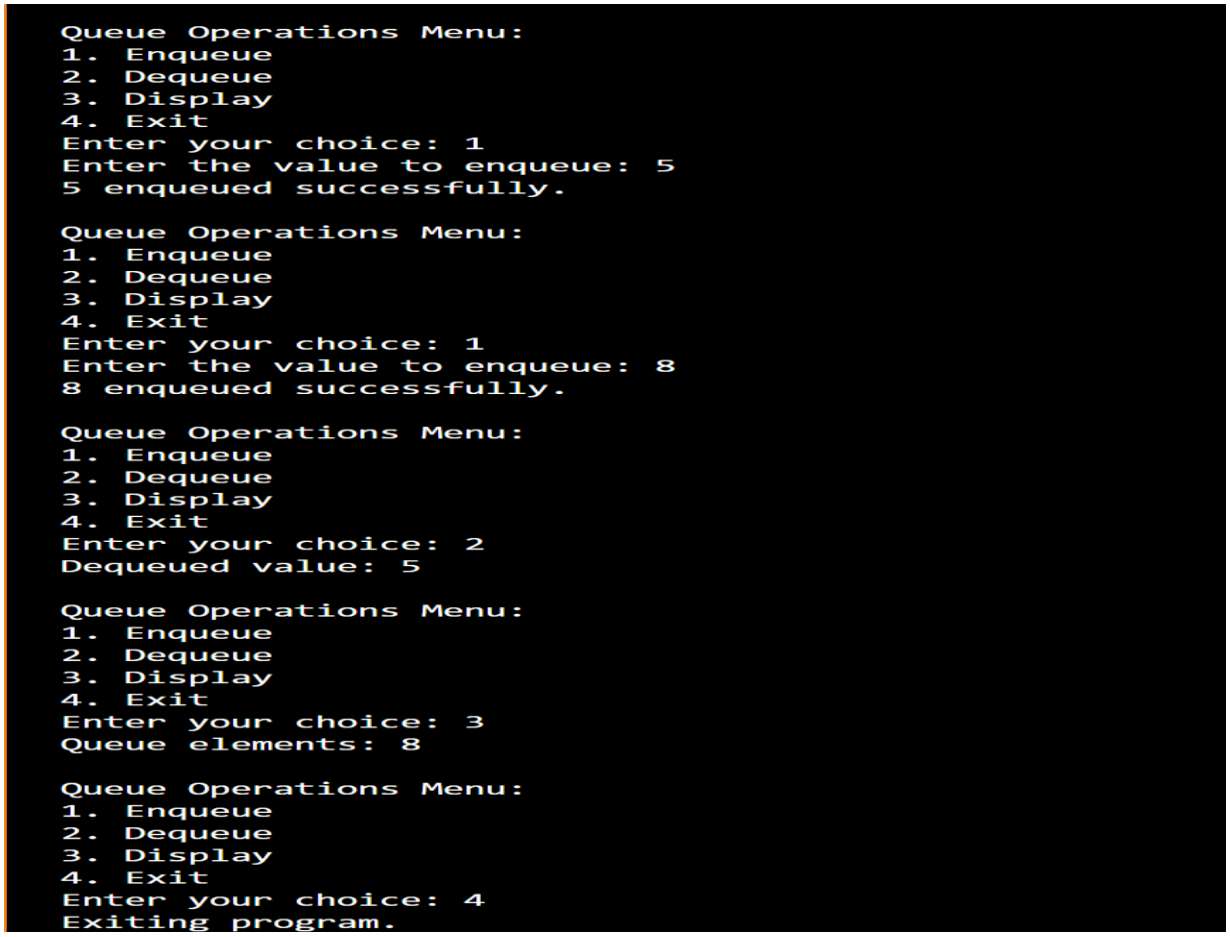
```
    q->rear = -1;
}
int isFull(struct Queue* q) {
    return (q->rear == MAX - 1);
}
int isEmpty(struct Queue* q) {
    return (q->front == -1 || q->front > q->rear);
}
void enqueue(struct Queue* q, int value) {
    if (isFull(q)) {
        printf("Queue is full! Cannot enqueue %d.\n", value);
    } else {
        if (q->front == -1) {
            q->front = 0;
        }
        q->rear++;
        q->arr[q->rear] = value;
        printf("%d enqueued successfully.\n", value);
    }
}
int dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty! Cannot dequeue.\n");
```

```
        return -1;
    } else {
        int dequeuedValue = q->arr[q->front];
        q->front++;
        if (q->front > q->rear) {
            q->front = q->rear = -1;
        }
        return dequeuedValue;
    }
}
void display(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty! Nothing to display.\n");
    } else {
        printf("Queue elements: ");
        for (int i = q->front; i<= q->rear; i++) {
            printf("%d ", q->arr[i]);
        }
        printf("\n");
    }
}
int main() {
    struct Queue q;
```

```
initialize(&q);
int choice, value;
    while (1) {
    printf("\nQueue Operations 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:
            printf("Enter the value to enqueue: ");
            scanf("%d", &value);
            enqueue(&q, value);
            break;
        case 2:
            value = dequeue(&q);
            if (value != -1) {
                printf("Dequeued value: %d\n", value);
            }
            break;
        case 3:
```

```
            display(&q);
            break;
        case 4:
            printf("Exiting program.\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
      }
   }
   return 0;
}
```

Output:



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

a. Enqueue()

b. Dequeue()

c. Disaply()

Algorithm:

Algorithm: CircularQueueOperations()

Input: Circular Queue Q of fixed size

Output: Perform Enqueue, Dequeue, or Display based on the user's choice

Initialize front ← -1 and rear ← -1

Repeat the following steps until the user exits:

Print menu: "1. Enqueue, 2. Dequeue, 3. Display, 4. Exit"

Read user choice

If choice = 1 (Enqueue):

If (front == 0 and rear == SIZE - 1) or (rear + 1 == front), print "Queue Overflow"

Else:

If front == -1, set front ← 0

Set rear ← (rear + 1) mod SIZE

Q[rear] ← value

Print "Element enqueued successfully"

Else if choice = 2 (Dequeue):

If front == -1, print "Queue Underflow"

Else:

Print "Dequeued element: Q[front]"

If front == rear, set front ← rear ← -1

Else, set front ← (front + 1) mod SIZE

Else if choice = 3 (Display):

If front == -1, print "Queue is empty"

Else:

Set i ← front

While i != rear, print Q[i] and update i ← (i + 1) mod SIZE

Print Q[i] (last element)

Else if choice = 4, exit

Else, print "Invalid choice"

End

Code:

```
#include <stdio.h>

#define SIZE 5


int cQueue[SIZE];

int front = -1, rear = -1;


void enqueue(int value) {

   if ((front == 0 && rear == SIZE - 1) || (rear + 1 == front)) {

      printf("Queue Overflow\n");

   } else {

      if (front == -1) {

         front = 0;

      }

      rear = (rear + 1) % SIZE;

      cQueue[rear] = value;

      printf("Enqueued %d into the circular queue\n", value);
```

```
    }
}

void dequeue() {
    if (front == -1) {
        printf("Queue Underflow\n");
    } else {
        printf("Dequeued element: %d\n", cQueue[front]);
        if (front == rear) {
            front = rear = -1;
        } else {
            front = (front + 1) % SIZE;
        }
    }
}

void display() {
    if (front == -1) {
        printf("Queue is empty\n");
    } else {
        printf("Queue elements are: ");
        int i = front;
        while (i != rear) {
```

```
            printf("%d ", cQueue[i]);
            i = (i + 1) % SIZE;
        }
        printf("%d\n", cQueue[i]);
    }
}

int main() {
    int choice, value;
    while (1) {
        printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d", &value);
                enqueue(value);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
```

```
            break;
        case 4:
            return 0;
        default:
            printf("Invalid choice\n");
      }
   }
   return 0;
}
```

Output:

```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 10
Enqueued 10 into the circular queue

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 20
Enqueued 20 into the circular queue

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued element: 10

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements are: 20

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
```

QUESTION 4: 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:

: SinglyLinkedListOperations()

Input: Singly linked list L

Output: Perform insertion, deletion, or search based on the user's choice

Initialize head ← NULL

Repeat the following steps until the user exits:

Print menu: "1. Insert, 2. Delete, 3. Search, 4. Display, 5. Exit"

Read user choice

If choice = 1 (Insertion):

Print "1. Beginning, 2. End, 3. At a given position"

Read sub-choice

If sub-choice = 1 (Beginning):

Create newNode with given value

Set newNode → next ← head

Set head ← newNode

If sub-choice = 2 (End):

Create newNode with given value

If head = NULL, set head ← newNode

Else, traverse to the last node and set lastNode → next ← newNode

If sub-choice = 3 (At a given position):

Read position

If position = 1, perform insertion at the beginning

Else:

Traverse to (position - 1)-th node

Create newNode with given value

Set newNode → next ← currentNode → next

Set currentNode → next ← newNode

Else if choice = 2 (Deletion):

Print "1. Beginning, 2. End, 3. At a given position"

Read sub-choice

If sub-choice = 1 (Beginning):

If head = NULL, print "List is empty"

Else, set head ← head → next

If sub-choice = 2 (End):

If head = NULL, print "List is empty"

Else:

Traverse to the second last node

Set secondLastNode → next ← NULL

If sub-choice = 3 (At a given position):

Read position

If position = 1, perform deletion at the beginning

Else:

Traverse to (position - 1)-th node

Set currentNode → next ← currentNode → next → next

Else if choice = 3 (Search):

Read value to search

Traverse the list and check if value exists

If found, print "Element found"

Else, print "Element not found"

Else if choice = 4 (Display):

If head = NULL, print "List is empty"

Else, traverse the list and print each node value

Else if choice = 5, exit

Else, print "Invalid choice"

End

Code:

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

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

struct Node* head = NULL;

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

void insertAtEnd(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
```

```
    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    printf("Inserted %d at the end\n", value);
}

void insertAtPosition(int value, int position) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    if (position == 1) {
        newNode->next = head;
        head = newNode;
    } else {
        struct Node* temp = head;
        for (int i = 1; i< position - 1 &&temp != NULL; i++) {
            temp = temp->next;
        }
```

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

void deleteAtBeginning() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    head = head->next;
    printf("Deleted %d from the beginning\n", temp->data);
    free(temp);
}
```

```
void deleteAtEnd() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    if (head->next == NULL) {
        printf("Deleted %d from the end\n", head->data);
        free(head);
        head = NULL;
        return;
    }
    struct Node* temp = head;
    while (temp->next->next != NULL) {
        temp = temp->next;
    }
    printf("Deleted %d from the end\n", temp->next->data);
    free(temp->next);
    temp->next = NULL;
}

void deleteAtPosition(int position) {
    if (head == NULL) {
        printf("List is empty\n");
```

```
        return;
    }
    if (position == 1) {
        struct Node* temp = head;
        head = head->next;
        printf("Deleted %d from position %d\n", temp->data, position);
        free(temp);
        return;
    }
    struct Node* temp = head;
    for (int i = 1; i< position - 1 && temp->next != NULL; i++) {
        temp = temp->next;
    }
    if (temp->next != NULL) {
        struct Node* toDelete = temp->next;
        temp->next = toDelete->next;
        printf("Deleted %d from position %d\n", toDelete->data, position);
        free(toDelete);
    } else {
        printf("Invalid position\n");
    }
}
```

```
void search(int value) {
    struct Node* temp = head;
    int position = 1;
    while (temp != NULL) {
        if (temp->data == value) {
            printf("Element %d found at position %d\n", value, position);
            return;
        }
        temp = temp->next;
        position++;
    }
    printf("Element %d not found\n", value);
}

void display() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    printf("List elements: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
```

```
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int choice, value, position;
    while (1) {
        printf("\n1. Insert\n2. Delete\n3. Search\n4. Display\n5. Exit\nEnter your choice:
");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("1. Beginning\n2. End\n3. At a given position\nEnter your choice: ");
                scanf("%d", &position);
                printf("Enter value: ");
                scanf("%d", &value);
                if (position == 1)
                    insertAtBeginning(value);
                else if (position == 2)
                    insertAtEnd(value);
                else {
                    printf("Enter position: ");
                    scanf("%d", &position);
```

```
            insertAtPosition(value, position);
        }
        break;
    case 2:
        printf("1. Beginning\n2. End\n3. At a given position\nEnter your choice: ");
        scanf("%d", &position);
        if (position == 1)
            deleteAtBeginning();
        else if (position == 2)
            deleteAtEnd();
        else {
            printf("Enter position: ");
            scanf("%d", &position);
            deleteAtPosition(position);
        }
        break;
    case 3:
        printf("Enter value to search: ");
        scanf("%d", &value);
        search(value);
        break;
    case 4:
        display();
```

```
            break;
        case 5:
            return 0;
        default:
            printf("Invalid choice\n");
      }
   }
   return 0;
}
```

Output:

```
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 1
1. Beginning
2. End
3. At a given position
Enter your choice: 1
Enter value: 10
Inserted 10 at the beginning

1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 1
1. Beginning
2. End
3. At a given position
Enter your choice: 2
Enter value: 100
Inserted 100 at the end

1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 1
1. Beginning
2. End
3. At a given position
Enter your choice: 3
Enter value: 30
Enter position: 2
Inserted 30 at position 2

1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 4
List elements: 10 30 100
```

```
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 2
1. Beginning
2. End
3. At a given position
Enter your choice: 1
Deleted 10 from the beginning

1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 2
1. Beginning
2. End
3. At a given position
Enter your choice: 3
Enter position: 2
Deleted 100 from position 2

1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 2
1. Beginning
2. End
3. At a given position
Enter your choice: 2
Deleted 30 from the end

1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 4
List is empty
```

```
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 4
List elements: 5 1 10

1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 3
Enter value to search: 1
Element 1 found at position 2

1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 3
Enter value to search: 100
Element 100 not found
```

QUESTION 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:

DoublyLinkedListOperations()

Input: Doubly Linked List L

Output: Perform Insertion, Deletion, or Search based on the user's choice

Repeat the following steps until the user exits:

Print menu: "1. Insertion, 2. Deletion, 3. Search, 4. Exit"

Read user choice

If choice = 1 (Insertion):

Print "1. Insert at Beginning, 2. Insert at End, 3. Insert at Position"

Read insertion choice

If choice = 1:

Create a new node

Set newNode.next ← head

If head ≠ NULL, set head.prev ← newNode

Set head ← newNode

Print "Inserted at beginning"

Else if choice = 2:

Create a new node

Traverse to the last node

Set lastNode.next ← newNode

Set newNode.prev ← lastNode

Print "Inserted at end"

Else if choice = 3:

Read position

If position = 1, perform insertion at the beginning

Else:

Traverse to the (position - 1)th node

Create a new node

Set newNode.next ← current.next

Set newNode.prev ← current

If current.next ≠ NULL, set current.next.prev ← newNode

Set current.next ← newNode

Print "Inserted at position"

Else if choice = 2 (Deletion):

Print "1. Delete at Beginning, 2. Delete at End, 3. Delete at Position"

Read deletion choice

If choice = 1:

If head = NULL, print "List is empty"

Else:

Set head ← head.next

If head ≠ NULL, set head.prev ← NULL

Print "Deleted from beginning"

Else if choice = 2:

If head = NULL, print "List is empty"

Else:

Traverse to the last node

Set lastNode.prev.next ← NULL

Print "Deleted from end"

Else if choice = 3:

Read position

If position = 1, perform deletion at the beginning

Else:

Traverse to the (position - 1)th node

Set current.next ← current.next.next

If current.next ≠ NULL, set current.next.prev ← current

Print "Deleted from position"

Else if choice = 3 (Search):

Read value to search

Traverse the list

If any node.data = value, print "Element found"

Else, print "Element not found"

Else if choice = 4, exit

Else, print "Invalid choice"

End

Code:

```
#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 insertAtBeginning(int data) {
    struct Node* newNode = createNode(data);
```

```
    if (head == NULL) {
        head = newNode;
    } else {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
    printf("Inserted %d at the beginning.\n", data);
}
void insertAtEnd(int 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("Inserted %d at the end.\n", data);
}
```

```
void insertAtPosition(int data, int position) {
    struct Node* newNode = createNode(data);
    if (position == 1) {
        insertAtBeginning(data);
        return;
    }
    struct Node* temp = head;
    for (int i = 1; i< position - 1 &&temp != NULL; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Invalid position!\n");
    } else {
        newNode->next = temp->next;
        if (temp->next != NULL) {
            temp->next->prev = newNode;
        }
        temp->next = newNode;
        newNode->prev = temp;
        printf("Inserted %d at position %d.\n", data, position);
    }
}
void deleteFromBeginning() {
```

```
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    struct Node* temp = head;
    head = head->next;
    if (head != NULL) {
        head->prev = NULL;
    }
    printf("Deleted %d from the beginning.\n", temp->data);
    free(temp);
}
void deleteFromEnd() {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    struct Node* temp = head;
    if (temp->next == NULL) {
        head = NULL;
    } else {
        while (temp->next != NULL) {
            temp = temp->next;
```

```
        }
        temp->prev->next = NULL;
    }
    printf("Deleted %d from the end.\n", temp->data);
    free(temp);
}
void deleteFromPosition(int position) {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    if (position == 1) {
        deleteFromBeginning();
        return;
    }
    struct Node* temp = head;
    for (int i = 1; i< position &&temp != NULL; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Invalid position!\n");
    } else {
        if (temp->next != NULL) {
```

```
            temp->next->prev = temp->prev;
        }
        if (temp->prev != NULL) {
            temp->prev->next = temp->next;
        }
        printf("Deleted %d from position %d.\n", temp->data, position);
        free(temp);
    }
}
void search(int key) {
    struct Node* temp = head;
    int position = 1;
    while (temp != NULL) {
        if (temp->data == key) {
            printf("Element %d found at position %d.\n", key, position);
            return;
        }
        temp = temp->next;
        position++;
    }
    printf("Element %d not found in the list.\n", key);
}
void display() {
```

```
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    struct Node* temp = head;
    printf("Doubly Linked List: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main() {
    int choice, subChoice, data, position;
    while (1) {
        printf("\nMain Menu:\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Search\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
switch (choice) {
    case 1:
        printf("\nInsert Options:\n");
        printf("1. At Beginning\n");
        printf("2. At End\n");
        printf("3. At Position\n");
        printf("Enter your sub-choice: ");
        scanf("%d", &subChoice);
        printf("Enter data to insert: ");
        scanf("%d", &data);
        if (subChoice == 1) {
            insertAtBeginning(data);
        } else if (subChoice == 2) {
            insertAtEnd(data);
        } else if (subChoice == 3) {
            printf("Enter position: ");
            scanf("%d", &position);
            insertAtPosition(data, position);
        } else {
            printf("Invalid sub-choice!\n");
        }
        break;
```

```
case 2:
    printf("\nDelete Options:\n");
    printf("1. From Beginning\n");
    printf("2. From End\n");
    printf("3. From Position\n");
    printf("Enter your sub-choice: ");
    scanf("%d", &subChoice);
    if (subChoice == 1) {
        deleteFromBeginning();
    } else if (subChoice == 2) {
        deleteFromEnd();
    } else if (subChoice == 3) {
        printf("Enter position: ");
        scanf("%d", &position);
        deleteFromPosition(position);
    } else {
        printf("Invalid sub-choice!\n");
    }
    break;

case 3:
    printf("Enter element to search: ");
```

```
            scanf("%d", &data);
            search(data);
            break;

        case 4:
            display();
            break;

        case 5:
            exit(0);

        default:
            printf("Invalid choice! Please try again.\n");
      }
   }
   return 0;
}
```

OUTPUT:

```
Main Menu:
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 1

Insert Options:
1. At Beginning
2. At End
3. At Position
Enter your sub-choice: 1
Enter data to insert: 23
Inserted 23 at the beginning.

Main Menu:
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 1

Insert Options:
1. At Beginning
2. At End
3. At Position
Enter your sub-choice: 2
Enter data to insert: 45
Inserted 45 at the end.

Main Menu:
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 1

Insert Options:
1. At Beginning
2. At End
3. At Position
Enter your sub-choice: 3
Enter data to insert: 67
Enter position: 2
Inserted 67 at position 2.
```

```
Main Menu:
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 2

Delete Options:
1. From Beginning
2. From End
3. From Position
Enter your sub-choice: 1
Deleted 23 from the beginning.

Main Menu:
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 2

Delete Options:
1. From Beginning
2. From End
3. From Position
Enter your sub-choice: 3
Enter position: 2
Deleted 45 from position 2.

Main Menu:
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 2

Delete Options:
1. From Beginning
2. From End
3. From Position
Enter your sub-choice: 2
Deleted 67 from the end.
```

```
Insert Options:
1. At Beginning
2. At End
3. At Position
Enter your sub-choice: 1
Enter data to insert: 23
Inserted 23 at the beginning.

Main Menu:
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 1

Insert Options:
1. At Beginning
2. At End
3. At Position
Enter your sub-choice: 1
Enter data to insert: 456
Inserted 456 at the beginning.

Main Menu:
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 1

Insert Options:
1. At Beginning
2. At End
3. At Position
Enter your sub-choice: 1
Enter data to insert: 789
Inserted 789 at the beginning.

Main Menu:
1. Insert
2. Delete
3. Search
4. Display
5. Exit
Enter your choice: 4
Doubly Linked List: 789 456 23
```